





Document Research Project

# Research Project

**Installation Manual** – Vandale Ward, De Herdt Mathias & Demuynck Simon

# Table of contents

# 1 General information

## 1.1 Introduction

This document has been prepared by Vandale Ward, De Herdt Mathias and Demuynck Simon, third year students at Howest in the field of MCT – IoT Infrastructure Engineer.

In our second semester we have a module called research project in which we have chosen a research question in our field of education. During this module, which lasts three weeks, we will fully research, elaborate and document our research question.

For our research question we have chosen for: “What alternatives exist to VMware vSphere for server virtualization in an enterprise environment and how can an existing VMware environment be migrated to it?” Each of us has chosen a different platform and we are going to work this out in detail. Ward chose Proxmox, Mathias chose Azure Stack HCI and Simon chose Citrix.

What do we want to achieve with this research question? We are looking what platforms are there as alternative for vSphere and on the other hand how do we migrate to them. This is what we are trying to answer with our research question.

## 1.2 Objective of this document

This document describes in detail how we build our environment, how everything works and a technical explanation for how we migrate to our platform of our choosing.

## 1.3 Target audience

System administrators that are looking for different platforms for their environment and how they can migrate to it from a technical point of view.

## 1.4 Research question

What alternatives exist to VMware vSphere for server virtualization in an enterprise environment and how can an existing VMware environment be migrated to it?

## 1.5 Chosen platforms

- Proxmox
- Azure Stack HCI
- Citrix

# 2 Proxmox

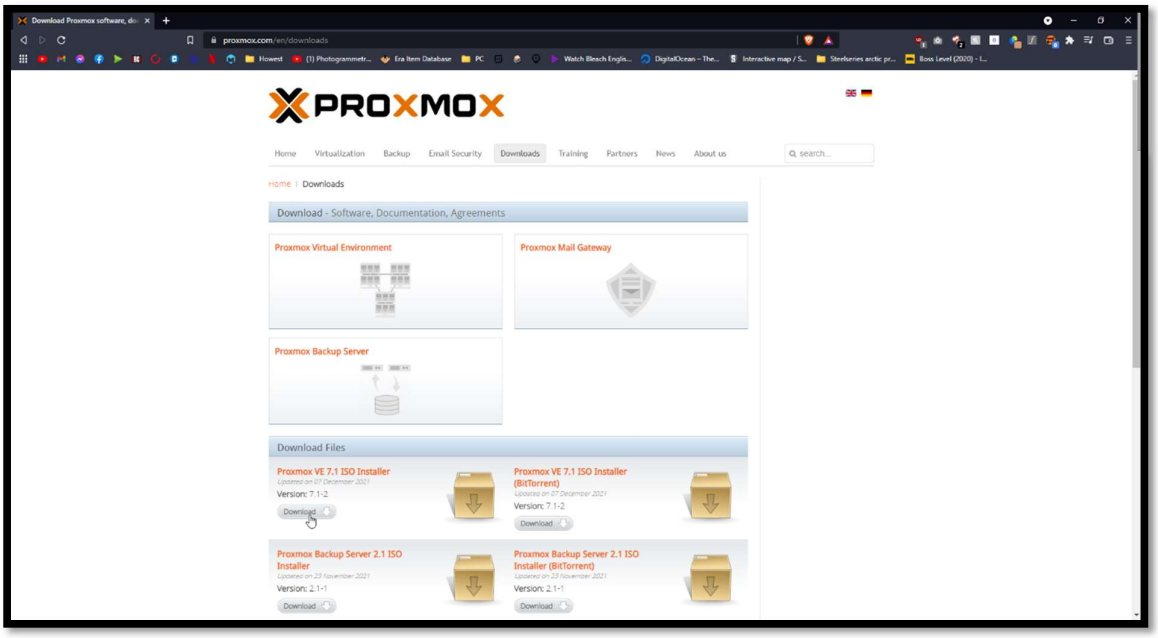
## 2.1 Downloading Proxmox VE

Before we can install our Proxmox VE server, we'll have to download the ISO file and link it to our server.

Go to: https://www.proxmox.com/en/downloads

See the system requirements here: https://www.proxmox.com/en/proxmox-ve/requirements

click on the download button at "Proxmox VE 7.1 ISO Installer".




## 2.2 Linking the ISO to the server

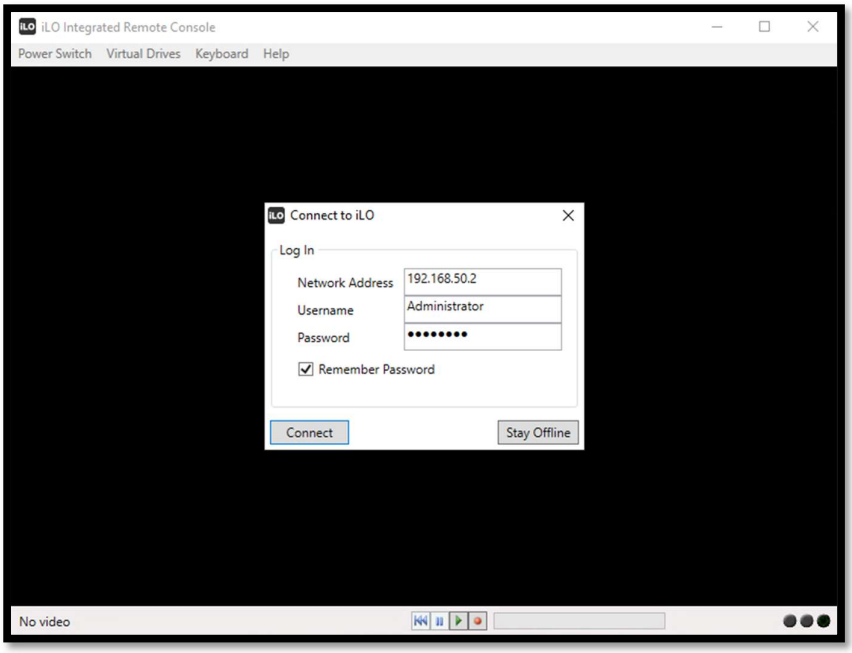
There are 2 main ways to link the ISO to the server:

- Make a bootable USB
- Connect to the Server with iLO / other remote management tool

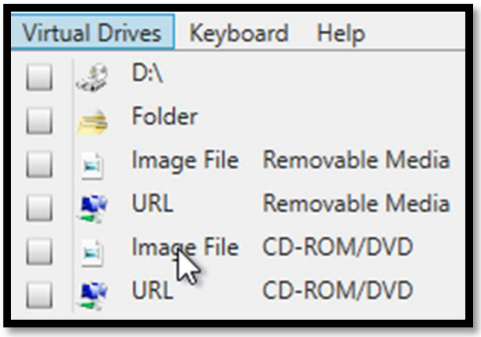
I'll be using the latter of these choices. I'm currently using a "HP ProLiant DL380 G7" with iLO 3.

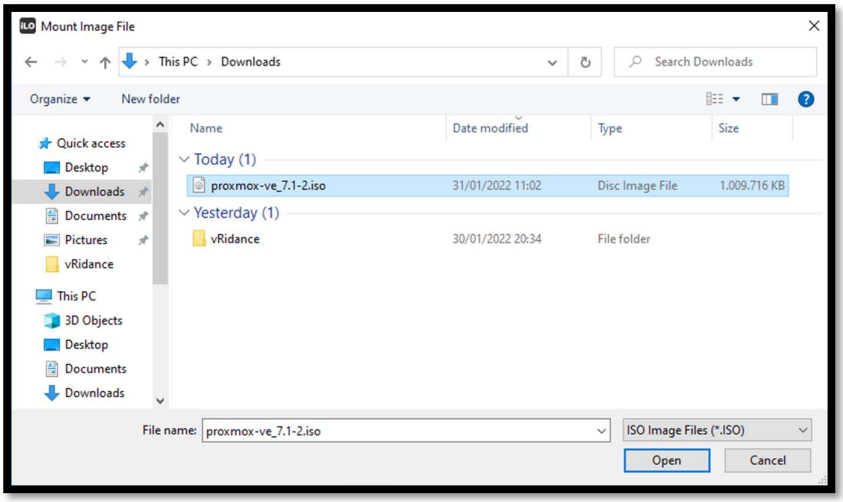
Go to:
https://support.hpe.com/hpesc/public/swd/detail?swItemId=MTX_bc8e3ffa59904ec3b505d9964d
to install the iLO Standalone Remote Console application.

My iLO has the IP 192.168.50.2. After installing the iLO application, open it and log in with the IP, username and password.

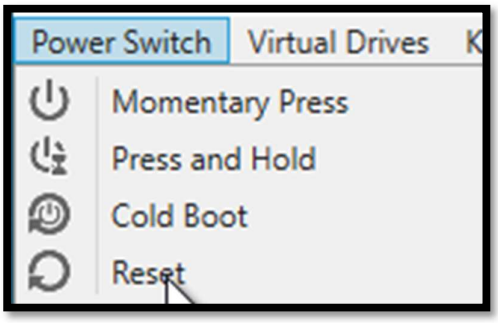



When you're connected, click on Virtual Drives and link the ISO file we've just downloaded.



After selecting our ISO file, click on “Power Switch” and click on “Reset”.

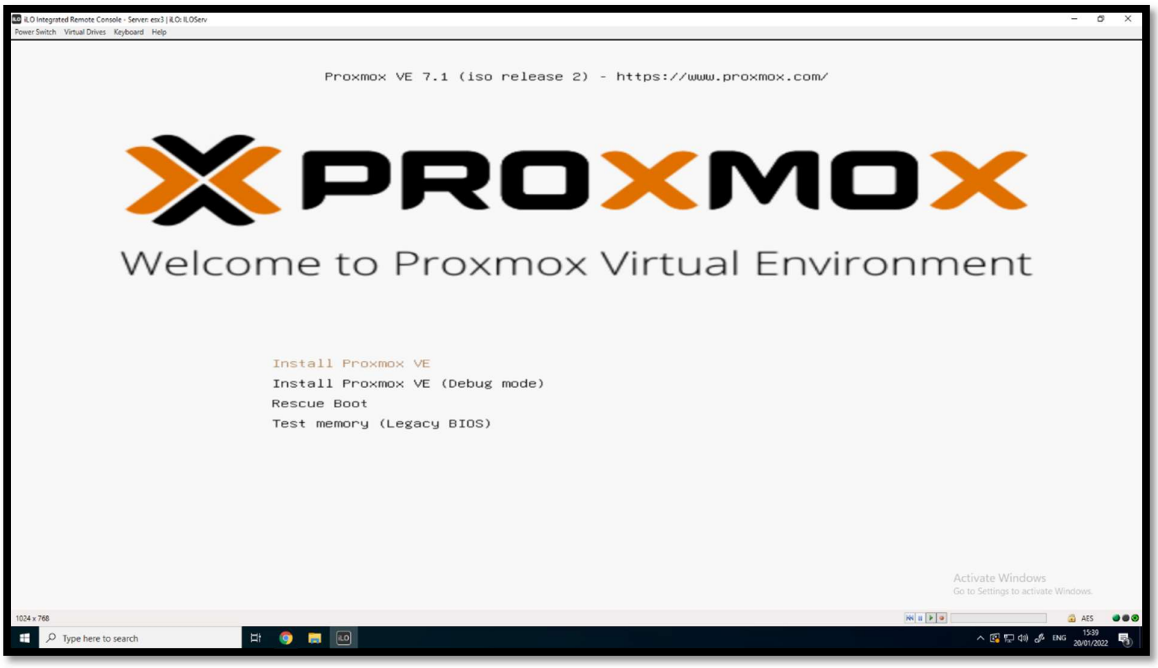



Now the Proxmox installation will begin.

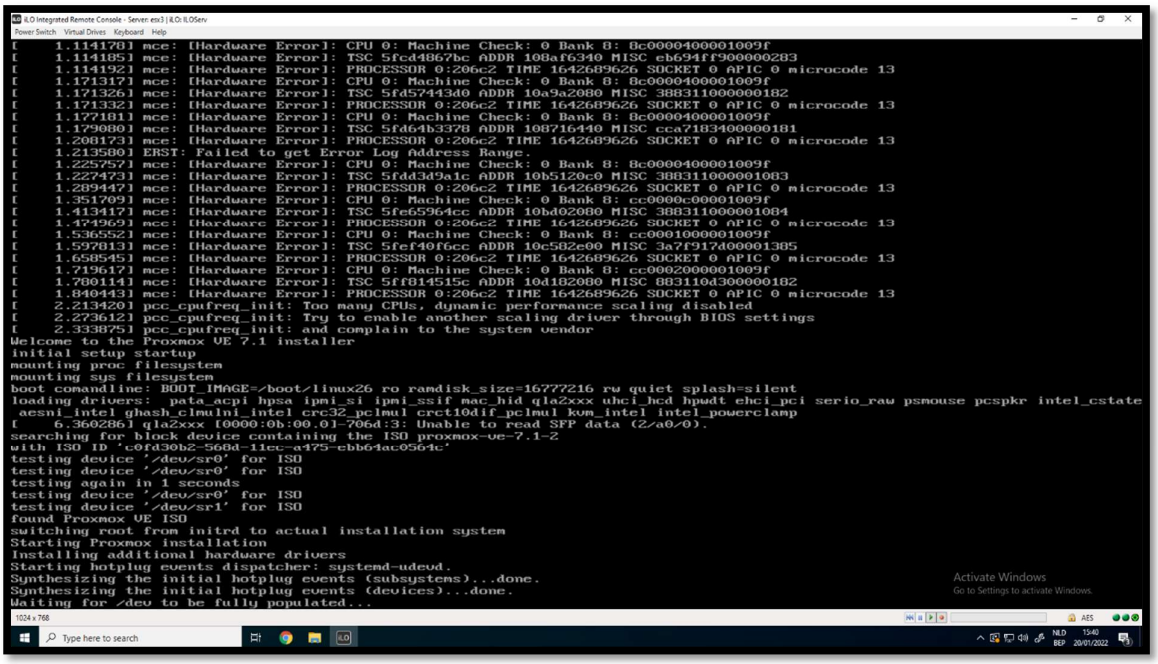
## 2.3 Installation of Proxmox VE

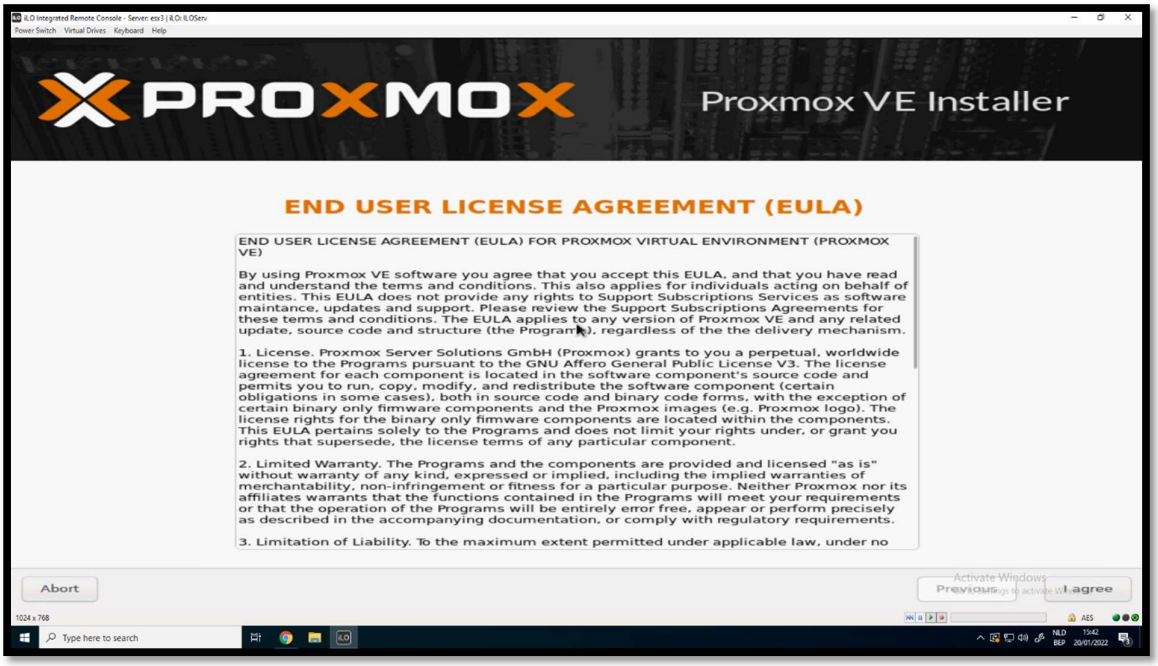
As said before, I am connected to a HP ProLiant DL380 G7 server using iLO. I’ve linked the Proxmox VE iso and reset the server to load the iso.

After the server has been booted and the iso has been loaded in, following screen should appear.



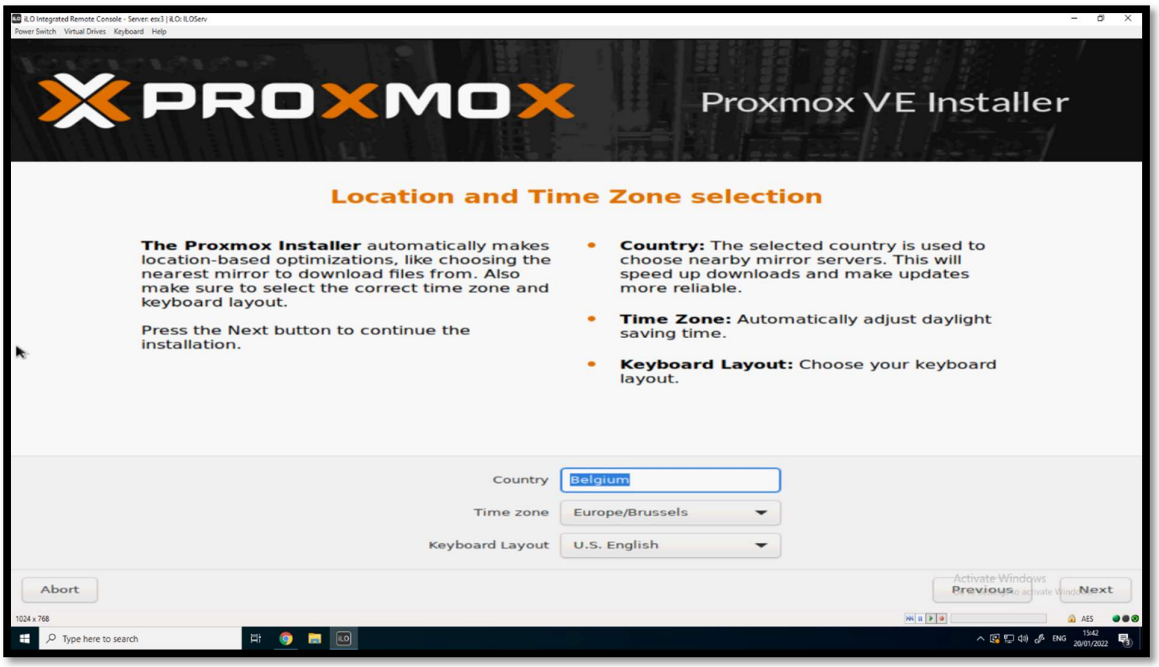

Of course we want to install Proxmox VE on our server, so we’ll chose for the first option called “Install Proxmox VE”. When you’ve clicked on install, you’ll see some command outputs that will start the installation process.



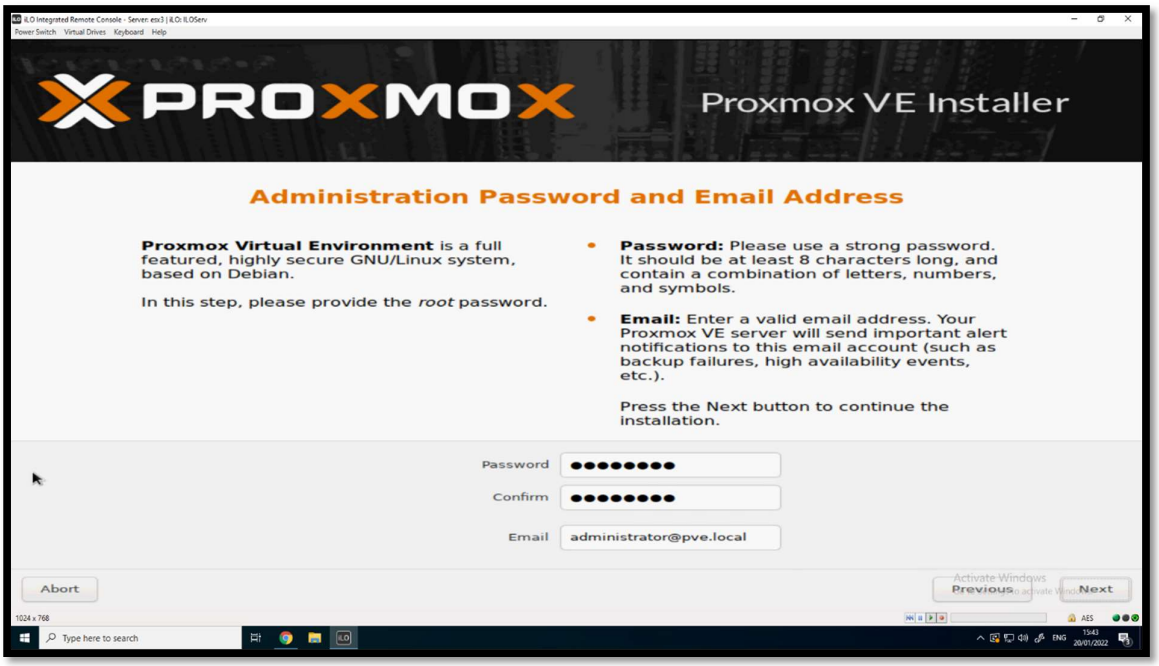




Each software installation has their own EULA, even the Hypervisor of Proxmox. Read through the UELA and click on “I agree” to officially start the installation steps.

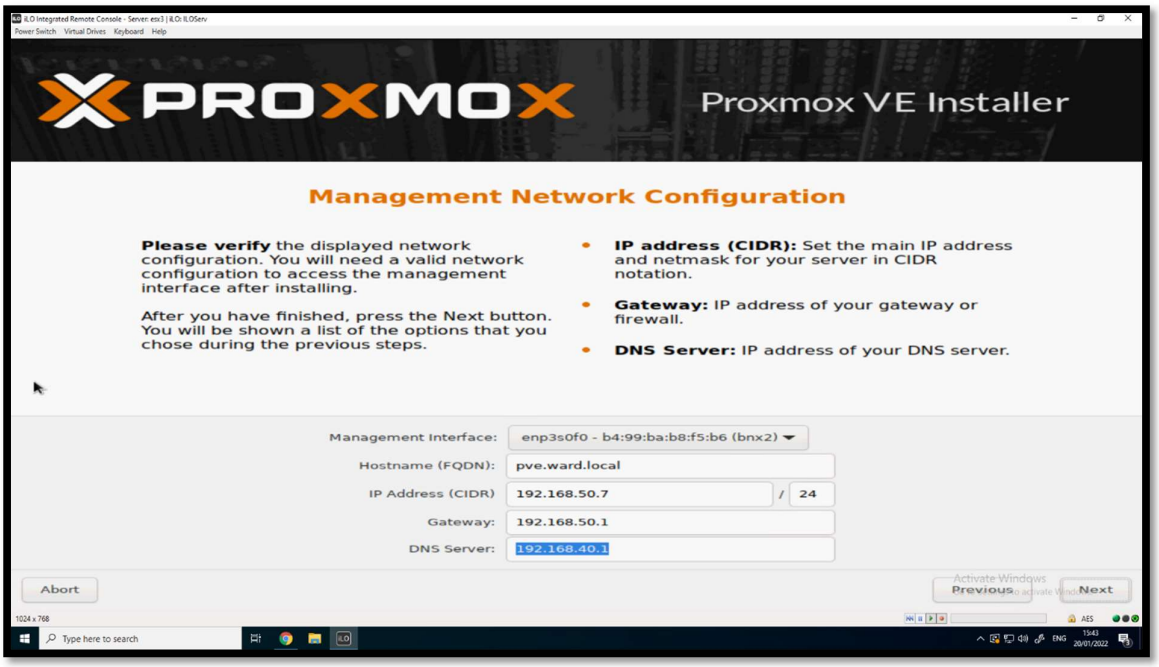

The next step will be selecting the drive to install the OS of Proxmox VE to. If you want to add extra storage to the server later, you can add the drives through the Debian CLI of the server. The OS volume of Proxmox will be only 100GB, so we'll have to make that larger.



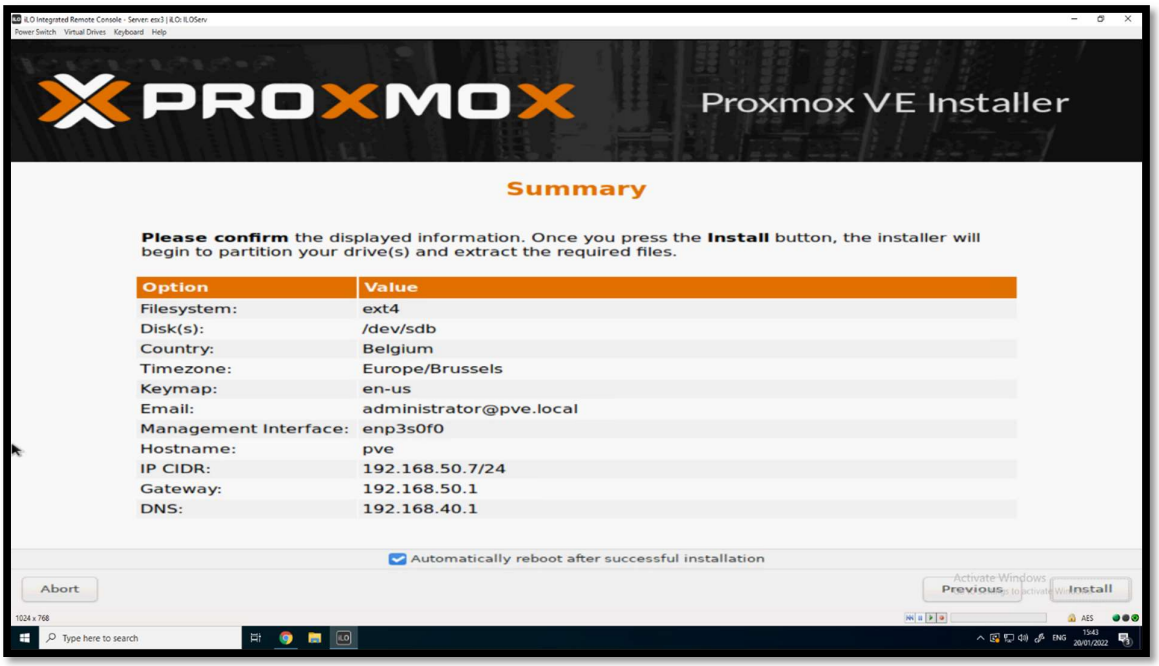

Your server will, if it's successfully connected with an ethernet cable, automatically prepare some of the settings. In this step It automatically detected that I was in Belgium, so it filled it in and also chose the Time zone for me. That's in Europe/Brussels of course.




The next step is choosing a password for the root user. This root user will be used to log in on our Proxmox CLI, but also the Web Interface. If you want to get warnings when something goes wrong or something like that, you can enter your Email address. I've chosen "administrator@pve.local".




After that we'll set up our network configuration. Just as the previous step, our server got a DHCP IP address from our DHCP Server. We want to use an static IP "192.168.50.7". This is also the IP we need to connect to the Web Interface and also the CLI. Make sure the Default Gateway and DNS Server is also right.




Finally we'll get a Summary of all the settings for installing our Proxmox VE Hypervisor, if everything is right, we can click "Install" to start the Proxmox VE installation. This will take about 5 to 10 minutes.

## 2.4 Problem after installing Proxmox VE

After installing my Proxmox VE server, I've noticed that the root volume only got 100GB of size to play with. When you have a clean install, Proxmox will set up multiple storage volumes. When you go to the Web Interface, you'll see that there are two storage volumes:

- pve
- local-lvm

"pve" is a storage volume that is linked to your root volume of your Proxmox OS, which has a size of only 100GB.

The "local-lvm" volume is linked to another volume that got all the remaining space of your disks. In my case, this was about 1.7TB.

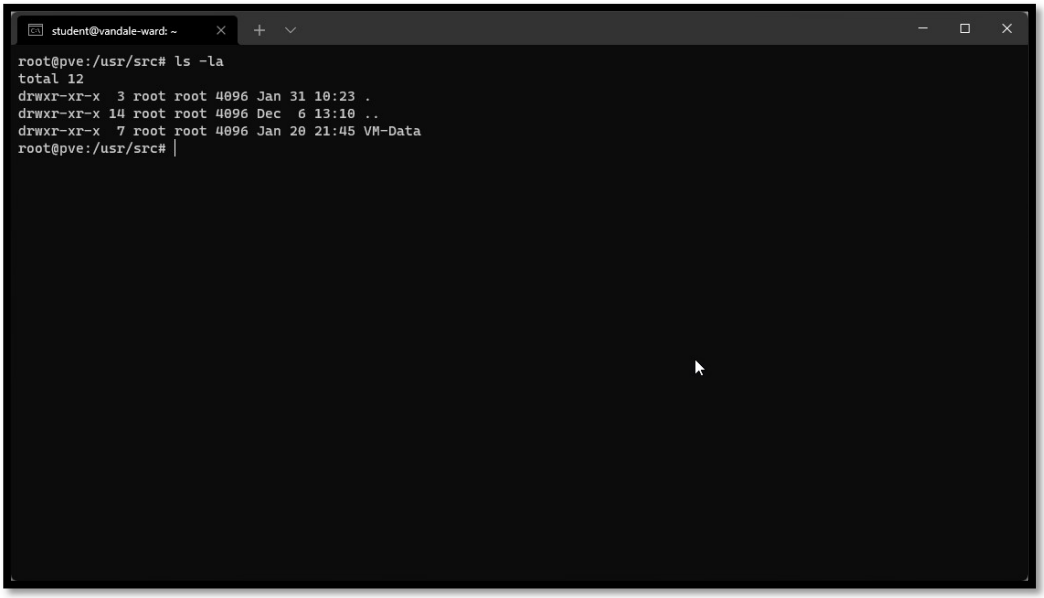
The problem I stumbled upon is that the root filesystem is used to store the VMs, but the disks are stored on the local-lvm storage. After installing about 4 VMs, my server crashed because it has ran out of space (the 100 GB). So to fix this I had to reinstall my Proxmox server and extend the size of my root volume. The root volume is called "pve-root" and is located at "/dev/pve/root". I followed this GitHub link to resize my volume by deleting the "local-lvm" volume located at "/dev/pve/data" and linking that data to the "pve-root" volume.

https://gist.github.com/laineantti/4fc29acbbd25593619a76b413e42b78f

## 2.5 Creating a new volume to store VM Disks and ISO files

Now that our "local-lvm" volume has been removed and we've extended the size of our "pve-root" volume, we don't have a storage volume to store our VM disks. For this we will create a storage volume ourselves.

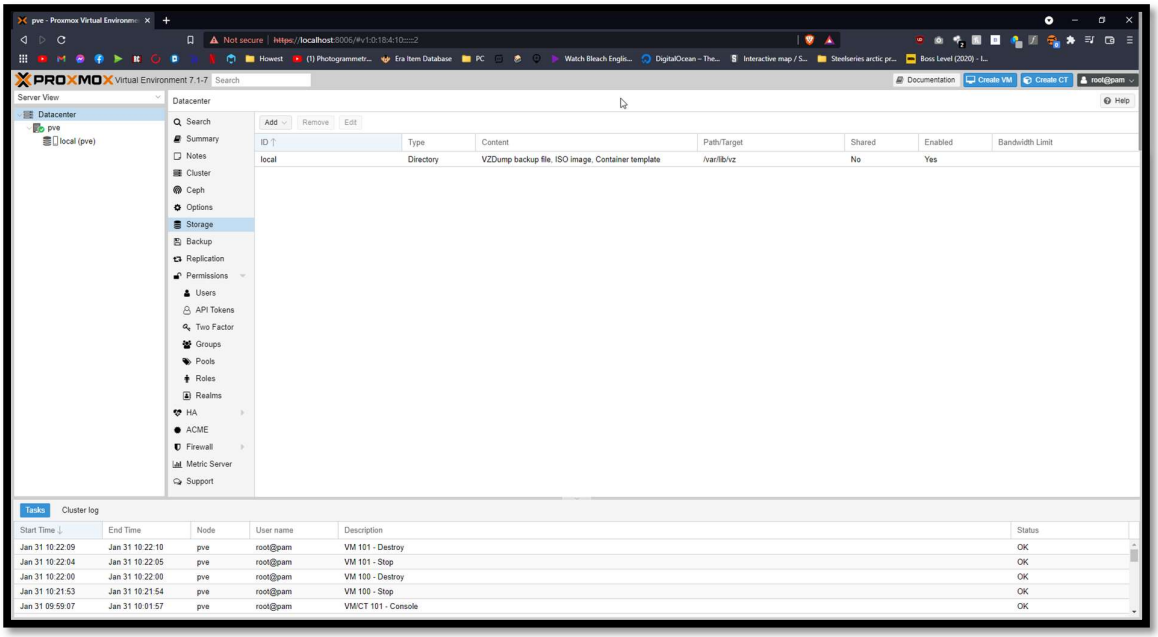
First we'll create an directory to where all this data will be stored. I will create a directory "VM-Data" located at "/usr/src/". It should look like this:



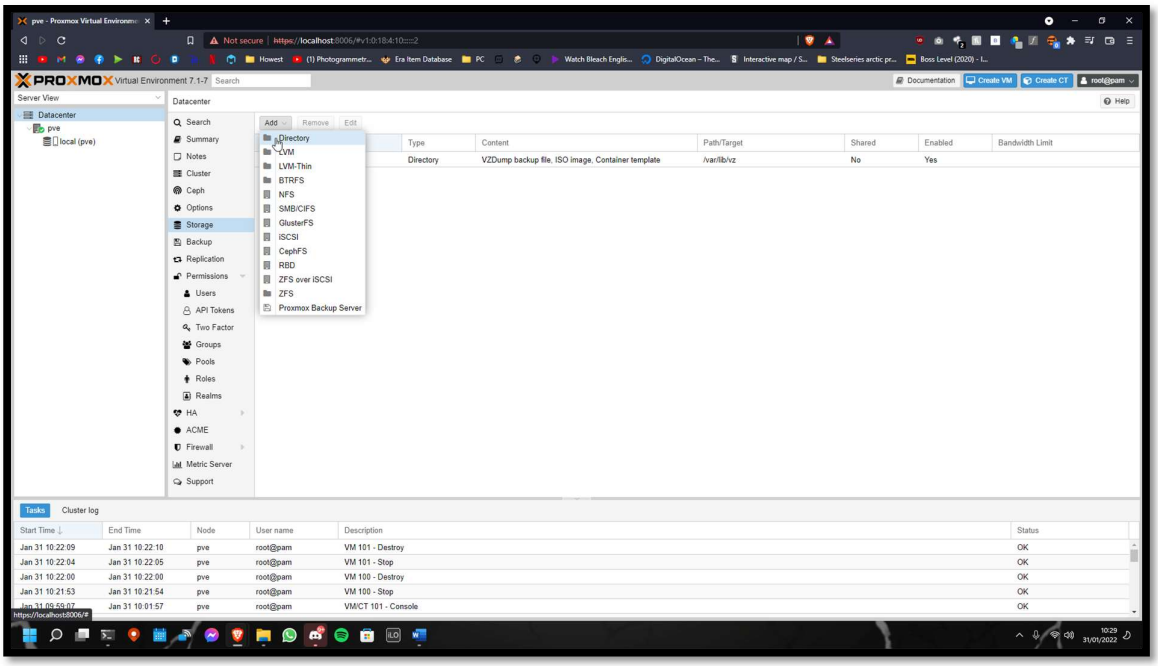
Now that we have created the directory at “/usr/src/VM-Data”.

Now we can create the storage volume on the Web Interface of our Proxmox VE server.

Go to your “Datacenter > Storage”



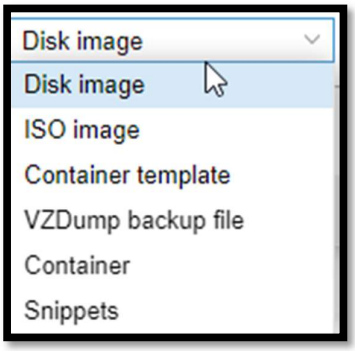

Now click on “Add” and choose for “Directory”



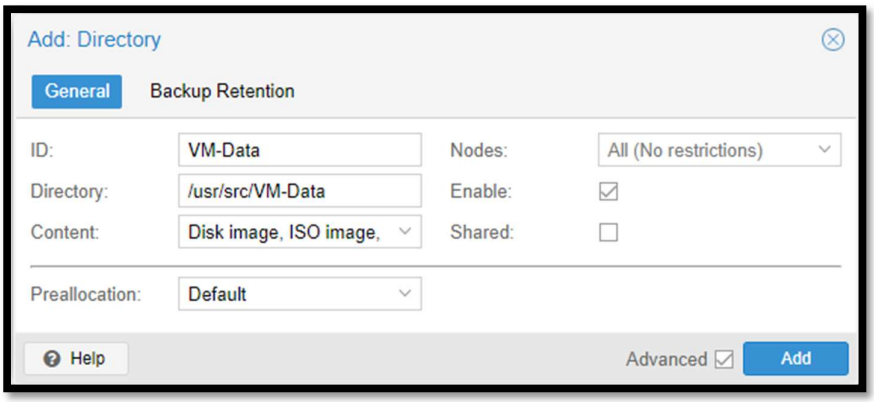
This will open a pop-up, asking some information for your Directory:

This will open a pop-up, asking some information for your Directory:

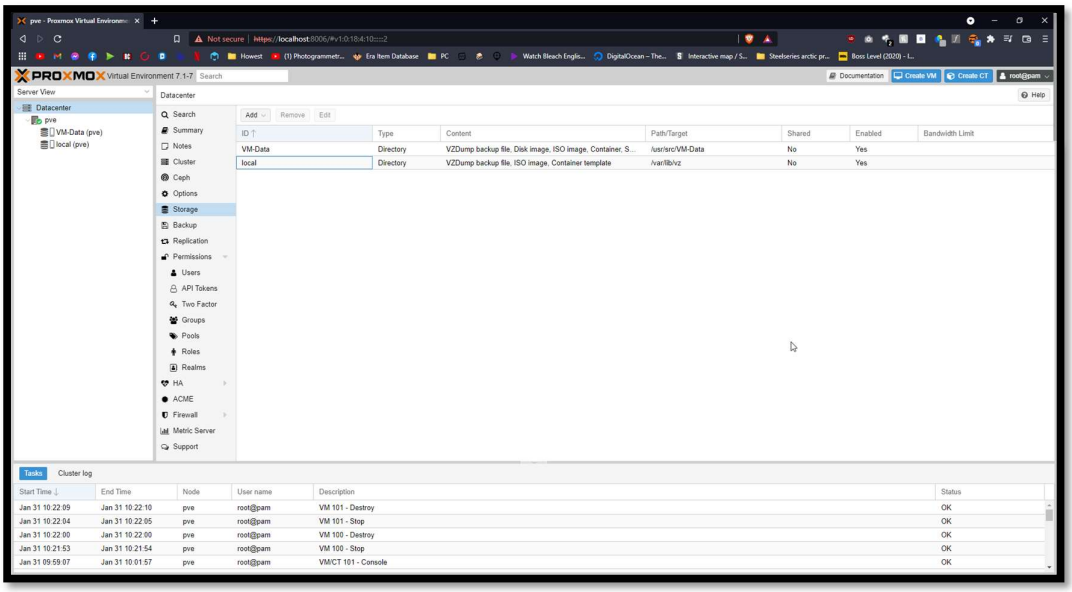
- ID — This will be the name of the storage volume used in Proxmox VE
- Directory — This will be the path to the Directory. In our case this is "/usr/src/VM-Data"
- Content — Choose what content will be saved to this Directory there are multiple choices. If you don't want to make a new storage, select all the things below.




These are the main things you'll need to set up to create a new storage volume. I will select all of the above options. It should look something like this:




After you've clicked on add, you will get the following result:

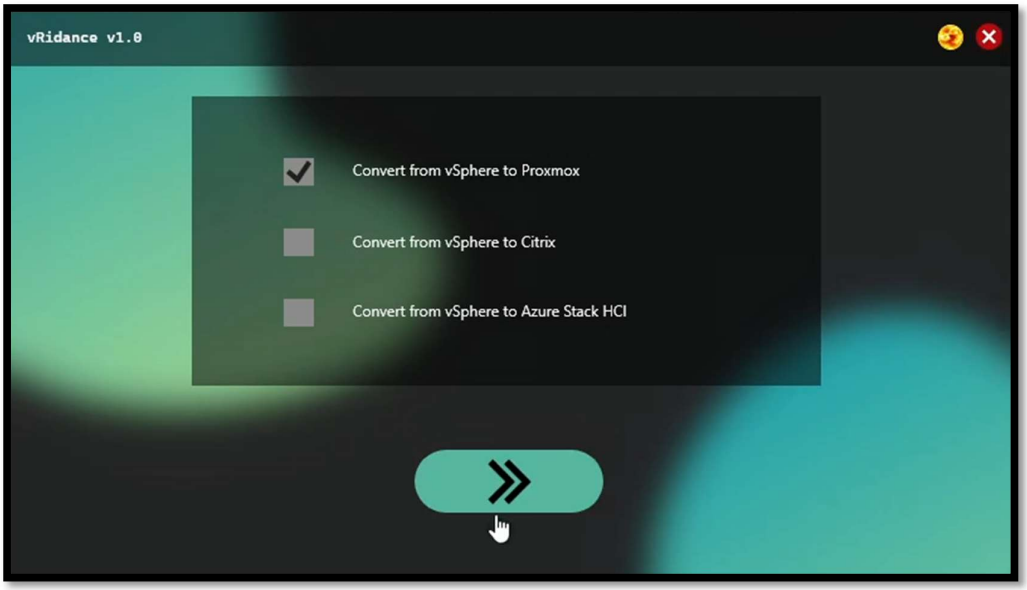



Now we've prepared our Proxmox VE Server to start virtualizing!

## 2.6 Explaining the code behind the Proxmox Migration

The vRidance tool has been created to migrate from vSphere to 3 different Hypervisors. I will be focusing on explaining the code behind the Proxmox Migration

### 2.6.1 MigrationSelection.xaml.cs




On this screen you'll be able to choose the different migration hypervisors, click on the checkbox next to the "Convert from vSphere to Proxmox", then next button will be enabled. This will button will detect which checkbox has been ticked and will run the following code.

#### 2.6.1.1 Function rectNext2_MouseLeftButtonUp

```
private void rectNext2_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            CheckBox[] checkBoxes = new CheckBox[3];
            checkBoxes[0] = chckProxmox;
            checkBoxes[1] = chckCitrix;
            checkBoxes[2] = chckAzureStack;

            for (int i = 0; i <= checkBoxes.Count() - 1; i++)
            {

                if (checkBoxes[i].IsChecked == true && checkBoxes[i].IsEnabled)
                {
                    ConvertToPlatform(i);
                }

            }
        }
```

The code will create a new list of CheckBoxes called “checkboxes” with a total length of 3 checkboxes. Once this list has been created, we’re adding the checkboxes to our list.

Now that our list has been filled, we will go through the list to see which checkbox has been selected. If a checkbox is checked, we’re going to the ConvertToPlatform function and we’re giving an int that is 0 – 3 as an argument.

#### 2.6.1.2 Function ConvertToPlatform

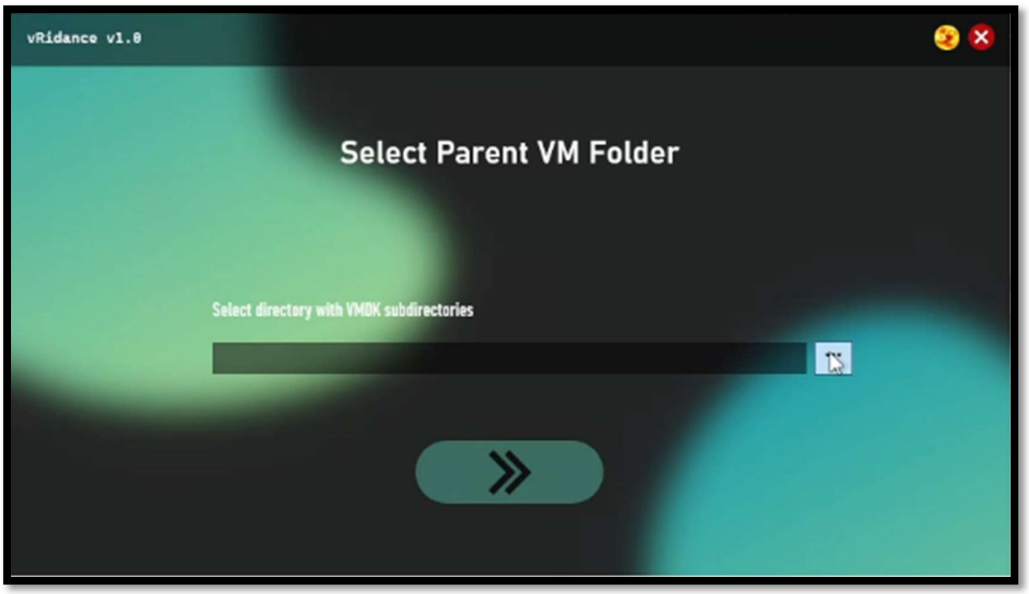
```
public void ConvertToPlatform(int i)
        {
            string curTheme = "";
            if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
            else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";
            switch (i)
            {
                case 0:
                    ChooseFolder chooseFolderWindow = new ChooseFolder(curTheme, "proxmox");
                    ((MainWindow)this.Owner).Content = chooseFolderWindow.Content;

                    chooseFolderWindow.Owner = ((MainWindow)this.Owner);
                    break;
                case 1:
                    ChooseFolder chooseFolderWindowCitrix = new ChooseFolder(curTheme,
"citrix");
                    ((MainWindow)this.Owner).Content = chooseFolderWindowCitrix.Content;

                    chooseFolderWindowCitrix.Owner = ((MainWindow)this.Owner);
                    break;
                case 2:
                    AzureStackHCI MigrateToAzureStackHCI = new AzureStackHCI(curTheme);
                    ((MainWindow)this.Owner).Content = MigrateToAzureStackHCI.Content;

                    MigrateToAzureStackHCI.Owner = ((MainWindow)this.Owner);
                    break;
            }
        }
```

This code will redirect the user to the next screen. First we’re making a sting called “curTheme”. This string is used to set the theme of the next screen. It’s scanning if the button to enable dark mode is shown or hidden. If it’s hidden, it means that dark mode has already been activated, when it’s shown, it means that the user is using light mode.

This aside, we're going to the more important code. We've used a switch case to see what number has been given as argument. When the number is "0", we're creating a new window ChooseFolder called "chooseFolderWindow". In this window we'll be giving the theme and the platform we want to migrate to.

In order for our window to be shown on top of the current window, we'll have to make content of the current window change to the content of the new window. The owner of the new window is also the same owner as the current one. This also means that we could possibly use functions from this window in the next.

### 2.6.2 ChooseFolder.xaml.cs




On this screen you will have to select the directory that has one or multiple subdirectories. In the subdirectories will be the "*.vmdk" and "*-flat.vmdk" file that we will have to upload.

#### 2.6.2.1 Function btnSearch_Click

```
private void btnSearch_Click(object sender, RoutedEventArgs e)
        {
            using (var fbd = new FolderBrowserDialog())
            {
                DialogResult result = fbd.ShowDialog();
                if (!string.IsNullOrWhiteSpace(fbd.SelectedPath))
                {
                    txtPath.Text = fbd.SelectedPath;
                }
                else if (txtPath.Text == "")
                {
                    System.Windows.MessageBox.Show("ERROR: Filepath empty!");
                }
            }
        }
```

When you've clicked on the browse button, a Windows Explorer dialog will appear, asking the user to choose a folder that contains the data I've said before. The user has two options. Either select a folder or click cancel.

If the user clicks on cancel, and the file path is empty, the user will get an error saying that the file path is empty.

When the user has chosen a folder, the Textbox “txtPath”. This layout change will automatically trigger the following functions.

#### 2.6.2.2 Function txtPath_TextChanged

```
private void txtPath_TextChanged(object sender, TextChangedEventArgs e)
        {
            string systemPath = txtPath.Text;
            if (Directory.Exists(systemPath) && (systemPath != null || systemPath != ""))
            {
                rectNext.Opacity = 1;
                rectNext.IsEnabled = true;
            }
            else
            {
                rectNext.Opacity = 0.5;
                rectNext.IsEnabled = false;
            }
        }
```

When the text of the TextBox “txtPath” has been changed, this piece of code will trigger to see if the given directory is an existing directory on your windows machine. It also looks with the text is not nothing. If all this is true, the next button will get enabled. When the given parameters in the if statement is false, it will disable the button again. Now that we have a path, we can click on the button to go to the next function.

#### 2.6.2.3 Function rectNext_MouseLeftButtonUp

```
public void rectNext_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            string curTheme = "";
            string path2vmdk = txtPath.Text;
            if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
            else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";
            string[] subdirectoryEntries = Directory.GetDirectories(path2vmdk);
            switch (platform)
            {
                case "proxmox":

                    if (subdirectoryEntries.Length == 0)
                    {
                        System.Windows.MessageBox.Show($"{path2vmdk} does not contain any
subdirectories with VMDKs");

                    }
                    else
                    {
                        foreach (var var_subdirectory in subdirectoryEntries)
                        {
                            string[] VMDirectory = Directory.GetFiles(@"" + var_subdirectory,
"*.vmdk");
                            if(VMDirectory.Length == 0)
                            {
                                System.Windows.MessageBox.Show($"The folder {var_subdirectory}
does not have any .vmdk files");
                                break;
                            }
                            else
                            {
                                ProxMoxLogin proxmoxLogin = new ProxMoxLogin(curTheme,
path2vmdk);
                                ((MainWindow)this.Owner).Content = proxmoxLogin.Content;
```

```
                            proxmoxLogin.Owner = ((MainWindow)this.Owner);
                        }
                    }
                }
                break;

            case "citrix":
                ……
                break;
        }
    }
```

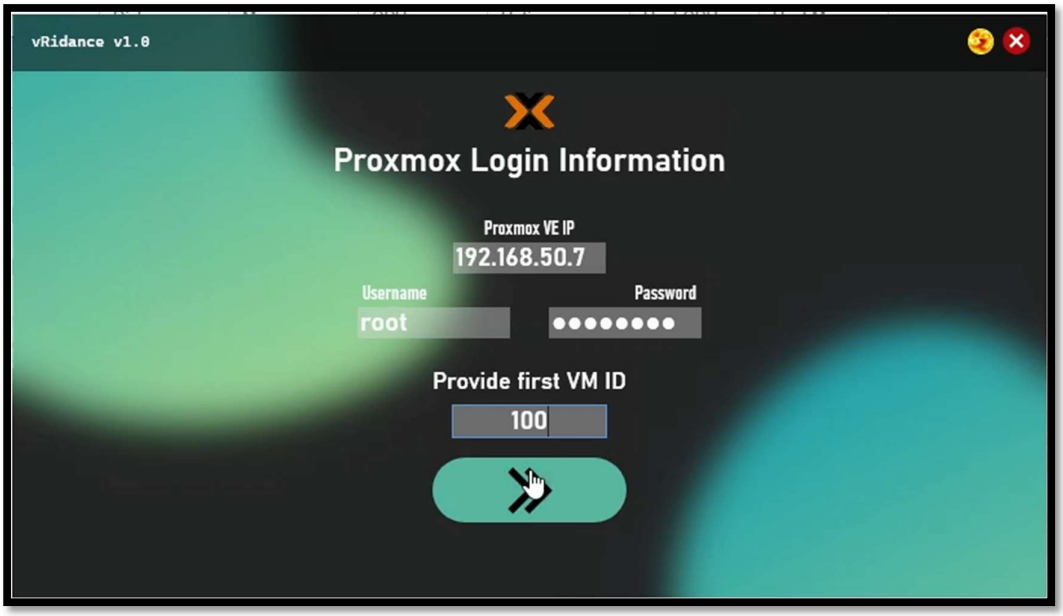
When the next button is clicked the same will happen as in the previous window change function, we will create a string that holds the current theme and send it to the new window. What we'll do as well is making a string of our path to the directory we've selected.

There is a list of strings called "subdirectoryEntries". This str list contains all the subdirectories there are in the selected directory.

In the constructor of ChooseFolder.xaml.cs we've given the string platform as an needed argument. This platform is either "proxmox" or "citrix". When the code goes to the Proxmox route, the code will go through the list of subdirectories. If the list is empty, aka if the selected directory doesn't have any subdirectories, you will get an error message. If your selected folder does contain subdirectories, the code will search for "*.vmdk" files in these subdirectories. If there aren't any "*.vmdk" files detected, the user also gets an error message.

When all goes right, and the directory structure is how it should be, the code will redirect you to the login windows for Proxmox .

### 2.6.3 ProxMoxLogin.xaml.cs




In this screen, the user will be able to give the login information of their Proxmox VE server. Here they'll have to give the IP, username and password. On this screen they'll also be asked to give the ID of the first VM they want to migrate. Proxmox VE uses VM IDs for their VMs.

#### 2.6.3.1 Function grdMain4_LayoutUpdated

```
private void grdMain4_LayoutUpdated(object sender, EventArgs e)
        {
            try
            {
                System.Net.IPAddress ipAddress;
                if (txtUsername.Text != "" && txtUsername.Text != null && pwbPassword.Password
!= "" && txtVMID.Text != "" && txtVMID.Text != null && System.Net.IPAddress.TryParse(txtIP.Text,
out ipAddress) && int.Parse(txtVMID.Text) % 1 == 0 && int.Parse(txtVMID.Text) > 0)
                {
                    rectNext.Opacity = 1;
                    rectNext.IsEnabled = true;
                }
                else
                {
                    rectNext.Opacity = 0.5;
                    rectNext.IsEnabled = false;
                }
            }
            catch (Exception) { rectNext.Opacity = 0.5; rectNext.IsEnabled = false; }
        }
```

Instead of checking if every textbox is the right input, we're doing a big check on the grid. When the layout of the grid is updated, meaning when something changes on the grid, e.g. you put a letter in a textbox, this code will trigger and check if the given if statement is try or false.

First I chose to add a try and catch, but doing nothing with the Exception if there is one. The reason I'm doing this is that you don't want to get errors all the time in your code. Let me explain.

First I'm adding the "System.Net.IPAddress" feature to check if a given text is a valid IP address. Then I have a big if-statement that checks through the 4 inputs:

- txtUsername
  - Not allowed to be null and empty
- pwbPassword
  - not allowed to be empty
- txtVMID
  - not allowed to be null and empty
  - has to be an int dividable by 1 with rest of 0
  - the int is not allowed to go under 0
- txtIP
  - has to be a valid IP. Here we don't have to check if the txtIP is empty or null, because and empty string is not a valid IP-address

If the statement above is true, the next button will be enabled. If the statement is false, the button will be disabled again.

The reason I've put this code in a try/catch is because a user can type letters in the "txtVMID" field. If the user does this, this will result in an error, the catch will handle this, and the button will stay disabled. Now that our button is enabled, it's possible for us to click it.

#### 2.6.3.2 Function rectNext_MouseLeftButtonUp

```
private void rectNext_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            string ip = txtIP.Text;
            string username = txtUsername.Text;
            string password = pwbPassword.Password.ToString();
            int first_vmid = int.Parse(txtVMID.Text);

            AuthenticationMethod method_checkCon = new PasswordAuthenticationMethod(username,
password);
            ConnectionInfo connection_checkCon = new ConnectionInfo(ip, username,
method_checkCon);
            var client_checkCon= new SshClient(connection_checkCon);
            string curTheme = "";
            if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
            else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";

            try
            {
                client_checkCon.Connect();
                string thepath = this.path;
                Migrate2Prox migration2proxmox = new Migrate2Prox(curTheme, ip, username,
password, thepath, first_vmid);
                ((MainWindow)this.Owner).Content = migration2proxmox.Content;
                client_checkCon.Disconnect();
                migration2proxmox.Owner = ((MainWindow)this.Owner);
            }
            catch (Exception)
            {
                MessageBox.Show($"Unable to make a connection to {ip}");
            }
        }
```

When the user has clicked next, it will put all the variables into strings and an int.

The next thing that happens is the AuthenticationMethod “method_checkCon”. The code will create an authentication method to test our connection. In the feature ConnectionInfo.

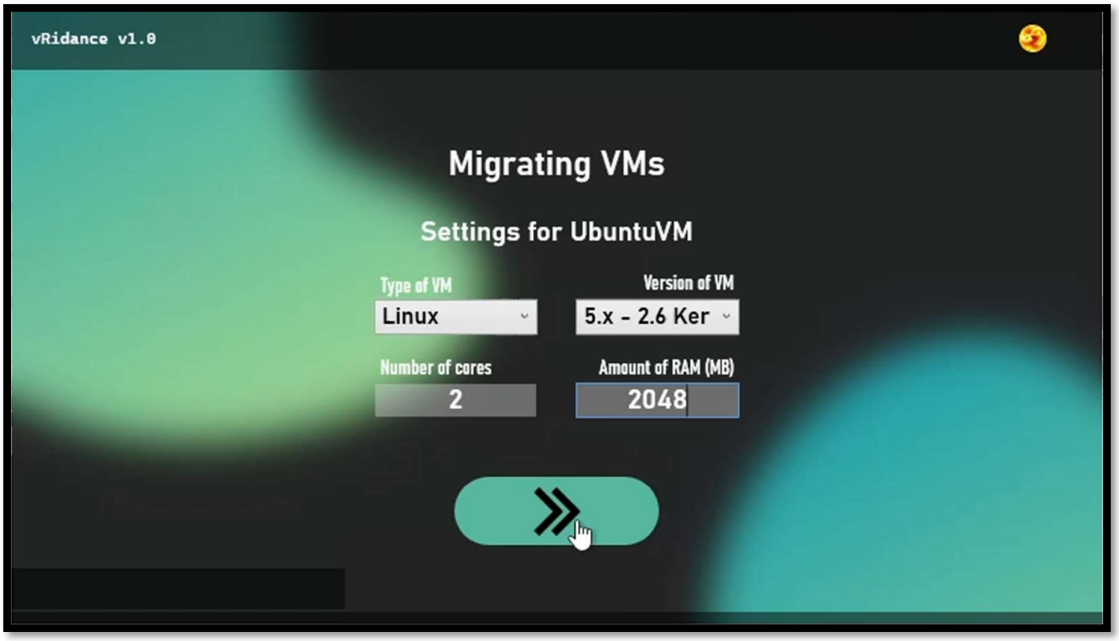
“connection_checkCon” we have the full log in information for our SSH session. In the next line, there is a var created that creates an SshClient session.

After all these variables have been created, the code will try to make a connection to the SSH Server with the IP, username and password provided. If the connection fails, the user will get the error that creating the connection has failed. When, however, a connection has been made, the code will redirect you to the next window. This windows will have the following arguments:

- curTheme — automatically generated “dark” or “light”
- ip — the IP of the Proxmox VE Hypervisor
- username — the given username of Proxmox, default is root
- password — the given password of Proxmox
- thepath — the path to the directory given in the previous window
- first_vmid — the first VM id that has to be assigned to the first VM

The made SSH connection will also disconnect after the new window has been shown. This makes sure that there aren’t too many SSH connections running

### 2.6.4 Migrate2Prox.xaml.cs




On this screen, the user will be asked to give some basic settings for a VM. In this case, the user is creating an Ubuntu VM named “UbuntuVM”. When everything is given right, the user can click on next to start the migration of the first VM. This windows keeps resetting every time a VM has been migrated. This window resets for as long as there are subdirectories.

#### 2.6.4.1 Beginning of code

```
public partial class Migrate2Prox : Window
    {
        string prox_host, prox_username, prox_password, folderPath;
        int start_vmid, cpu_cores, memory;

        string os_type;

        bool nextIsClicked = false, vmCreated = false;

        double progress;

        Thread changeParamsThread;
        public Migrate2Prox(string theme, string host, string var_username, string var_password,
string var_path, int var_start_id)
        {
            this.prox_host = host;
            this.prox_username = var_username;
            this.prox_password = var_password;
            this.folderPath = var_path;
            this.start_vmid = var_start_id;

            this.nextIsClicked = false;

            InitializeComponent();

            rectClose.Visibility = Visibility.Hidden;
            lblInfo.Content = "";
            changeTheme(theme);

            changeParamsThread = new Thread(changeParams);
```

```
            changeParamsThread.Start();
        }
        …
    }
```

At the start of my code, I've made some global variables, because they're used throughout the whole file. When the constructor is called, the variables will get filled with the parameters from the constructor. An other thing that will happen is the creation of a thread. I'm using a thread that runs next to the main code. The thread will be responsible to start the creation of the VM after the parameters (settings) have been changed.

#### 2.6.4.2 Function cbType_SelectionChanged

```
private void cbType_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (cbType.SelectedIndex == 0)
            {
                List<string> versions = new List<string>();
                versions.Add("5.x - 2.6 Kernel");
                versions.Add("2.4 Kernel");
                cbVersion.ItemsSource = versions;
                cbVersion.IsEnabled = true;
            }
            else if (cbType.SelectedIndex == 1)
            {
                List<string> versions = new List<string>();
                versions.Add("11/2022");
                versions.Add("10/2016/2019");
                versions.Add("8.x/2012/2012r2");
                versions.Add("7/2008r2");
                versions.Add("Vista/2008");
                versions.Add("XP/2003");
                versions.Add("2000");
                cbVersion.ItemsSource = versions;
                cbVersion.IsEnabled = true;
            }
        }
```

When the window is being called, there are already two selections in the "cbType" ComboBox. The two possible selections are "Linux" and "Microsoft Windows". When the user selects for "Linux" 0, there will be a list created with all the possible versions of VMs that can be created. For Linux we have:

- 5.x - 2.6 Kernel
- 2.4 Kernel

When these two possible choices have been added in the list, the list will be added to the "cbVersions" ComboBox. After this the ComboBox will be enabled so that the user can choose their version.

If the user selects "Windows" 1, the list will look like this:

- 11/2022 (Windows 11 / Windows Server 2022)
- 10/2016/2019 (Windows 10 / Windows Server 2016 & 2019)
- 8.x/2012/2012r2 (Windows 8 / Windows Server 2012 & 2012r2)
- 7/2008r2 (Windows 7 / Windows Server 2008r2)
- Vista/2008 (Windows Vista / Windows Server 2008)

- XP/2003 (Windows XP / Windows Server 2003)
- 2000 (Windows 2000)

#### 2.6.4.3 Function txtCores_TextChanged

```
private void txtCores_TextChanged(object sender, TextChangedEventArgs e)
        {
            try
            {
                if (txtCores.Text == "" && txtCores.Text == null)
                {
                    rectNext.Opacity = 0.5;
                    rectNext.IsEnabled = false;
                }
                else if (txtCores.Text != "" && txtCores.Text != null && txtMemory.Text != "" &&
txtMemory.Text != null && cbVersion.SelectedIndex >= 0)
                {
                    if (((int.Parse(txtCores.Text) % 2 == 0 || int.Parse(txtCores.Text) == 1) &&
int.Parse(txtCores.Text) > 0) && (int.Parse(txtMemory.Text) % 256 == 0 &&
int.Parse(txtMemory.Text) > 0))
                    {
                        rectNext.Opacity = 1;
                        rectNext.IsEnabled = true;
                    }
                    else
                    {
                        rectNext.Opacity = 0.5;
                        rectNext.IsEnabled = false;
                    }
                }
            }
            catch (Exception) { rectNext.Opacity = 0.5; rectNext.IsEnabled = false; }
        }
```

When the TextBox “txtcores” changes text, this function will trigger, it’s looking if the text field is not empty. If it’s empty, it will disable the button. If it’s not empty, it will do another if statement that check if the fields of “txtCores” AND “txtMemory is not empty”. If they’re not, then a big if statement will handle the rest. This statement checks the following things:

- txtCores
  - should be int, dividable by 2, with rest 0 OR should be int, being 1
  - the int should be a number above 0
- txtMemory
  - should be int, dividable by 256, with rest 0
  - the int should be a number above 0

if the following statements are all true, the button will be enabled. If one of the statements is false, the button will be disabled.

#### 2.6.4.4 Function txtMemory_TextChanged

```
private void txtMemory_TextChanged(object sender, TextChangedEventArgs e)
        {
            try
            {
                if (txtMemory.Text == "" && txtMemory.Text == null)
                {
                    rectNext.Opacity = 0.5;
                    rectNext.IsEnabled = false;
                }
                else if (txtMemory.Text != "" && txtMemory.Text != null && txtCores.Text != ""
&& txtCores.Text != null && cbVersion.SelectedIndex >= 0)
                {
                    if ((int.Parse(txtMemory.Text) % 256 == 0 && int.Parse(txtMemory.Text) > 0)
&& ((int.Parse(txtCores.Text) % 2 == 0 || int.Parse(txtCores.Text) == 1) &&
int.Parse(txtCores.Text) > 0))
                    {
                        rectNext.Opacity = 1;
                        rectNext.IsEnabled = true;
                    }
                    else
                    {
                        rectNext.Opacity = 0.5;
                        rectNext.IsEnabled = false;
                    }
                }
            }
            catch (Exception) { rectNext.Opacity = 0.5; rectNext.IsEnabled = false; }
        }
```

This piece of code does the same as the code for “txtCores_TextChanged”, but more focused to the “txtMemory” TextBox.

#### 2.6.4.5 Function rectNext_MouseLeftButtonUp

```
private void rectNext_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            if (cbType.SelectedIndex == 0)
            {
                if (cbVersion.SelectedIndex == 0) os_type = "l26";
                else if (cbVersion.SelectedIndex == 1) os_type = "l24";
            }else if (cbType.SelectedIndex == 1)
            {
                if (cbVersion.SelectedIndex == 0) os_type = "win11";
                else if (cbVersion.SelectedIndex == 1) os_type = "win10";
                else if (cbVersion.SelectedIndex == 2) os_type = "win8";
                else if (cbVersion.SelectedIndex == 3) os_type = "win7";
                else if (cbVersion.SelectedIndex == 4) os_type = "w2k8";
                else if (cbVersion.SelectedIndex == 5) os_type = "w2k3";
                else if (cbVersion.SelectedIndex == 6) os_type = "w2k";
            }
            cpu_cores = int.Parse(txtCores.Text);
            memory = int.Parse(txtMemory.Text);
            nextIsClicked = true;
        }
```

When the next button is clicked, this code will change the parameters that we need to migrate our VMs there parameters are “os_type”, “cpu_cores” and “memory”. When this is done, a bool “nextIsClicked” will be set to true, triggering an action in our running Thread.

### 2.6.4.6 Function ChangeParams

#### 2.6.4.6.1 Creating VM per subdirectory

```
public void changeParams()
        {
            if (folderPath != "")
            {
                string[] subdirectoryEntries = Directory.GetDirectories(folderPath);

                foreach (var var_subdirectory in subdirectoryEntries)
                {
                    vmCreated = false;
                    DirectoryInfo di = new DirectoryInfo(@"" + var_subdirectory);
                    string DirectoryName = di.Name;
                    this.Dispatcher.Invoke(() => {

                        lblCurrVM.Content = $"Settings for {DirectoryName}";
                        rectNext.Opacity = 0.5;
                        rectNext.IsEnabled = false;
                        cbVersion.IsEnabled = false;
                        List<string> types = new List<string>();
                        types.Add("Linux");
                        types.Add("Microsoft Windows");
                        cbType.ItemsSource = types;
                    });

                    while(true)
                    {
                        Thread.Sleep(100);
                        if (nextIsClicked == true)
                        {
                            this.Dispatcher.Invoke(() =>
                            {
                                cbType.IsEnabled = false;
                                cbVersion.IsEnabled = false;
                                txtCores.IsEnabled = false;
                                txtMemory.IsEnabled = false;
                                rectNext.IsEnabled = false;
                                rectNext.Opacity = 0.5;
                                lblCurrVM.Content = $"Creating VM {DirectoryName}";
                            });
                            nextIsClicked = false;
                            createTheVMS(var_subdirectory.ToString());
                            while (true) {
                                Thread.Sleep(100);
                                if (vmCreated == true) break;
                            }
                            this.Dispatcher.Invoke(() =>
                            {
                                txtCores.Text = "";
                                txtMemory.Text = "";
                                cbType.ItemsSource = "";
                                cbVersion.ItemsSource = "";
                                cbType.IsEnabled = true;
                                cbVersion.IsEnabled = true;
                                txtCores.IsEnabled = true;
                                txtMemory.IsEnabled = true;
                                rectNext.IsEnabled = true;
                                rectNext.Opacity = 1;
                            });
                            start_vmid++;
                            break;
                        }
                    }
                }
            }
            … code to be explained later (finished migration)
```

This is our thread continuously running. If the selected folder path is not empty the following code will be executed.

First the code will create the same list of strings from the same directory. Then the list will be read out in an foreach loop that will go through the list.

In the beginning of the loop, the bool “vmCreated” is set to false, because the VM hasn’t been created yet. After that we’ll create a DirectoryInfo “di” of our subdirectory. This directory info will be used to get the name of our subdirectory. The name of the subdirectory will be the name of the VM we will create on Proxmox.

Next we want to control some items on our program, in order to change items outside of our thread, we need to use an “this.Dispatcher.Invoke”. This will run the commands on the main thread of the program. There will be multiple Invokes throughout the code, so keep an eye out for them 😉.

After changing these items, we have created a while loop that has a sleep of 100ms. This is because the “.exe” version of the program is too fast and skips commands. The if statement in this program checks if the bool “nextIsClicked” has been set to true. If this is the case, the whole window will get blocked from making changes.

When everything is blocked, the migration of the first VM will start. While the migration is running, the user is not able give the next settings, because it will give errors in the code. So I’ve made a new loop that checks if the bool “vmCreated” has been set to true. When this happens, the rest of the code starts running. There is another Invoke command that resets and unlocks the items in the window. This is done so that the user can input the settings for their next VM.

Now the int “start_vmid” will get added up by 1, so that you don’t have 2 VMs trying to be on the same VM id.

You can find the full explanation of the function createTheVMS

#### 2.6.4.6.2 finished migration

```
this.Dispatcher.Invoke(() =>
            {
                string curTheme = "";
                if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
                else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";
                EndScreen endScreen = new EndScreen(curTheme);
                ((MainWindow)this.Owner).Content = endScreen.Content;

                endScreen.Owner = ((MainWindow)this.Owner);
            });
```

Just as all the previous code, this will redirect you to the finished window when the VMs have been migrated.

### 2.6.4.7 Function createTheVMS

#### 2.6.4.7.1 Uploading the VMDK files

```
public async Task createTheVMS(string subdirectory)
        {
            string[] VMDirectory = Directory.GetFiles(@"" + subdirectory, "*.vmdk");

            DirectoryInfo di = new DirectoryInfo(@"" + subdirectory);
            string DirectoryName = di.Name;

            AuthenticationMethod method_mkdir = new PasswordAuthenticationMethod(prox_username,
prox_password);
            ConnectionInfo connection_mkdir = new ConnectionInfo(prox_host, prox_username,
method_mkdir);
            var client_mkdir = new SshClient(connection_mkdir);
            client_mkdir.Connect();
            client_mkdir.RunCommand($"rm -rf /usr/src/{DirectoryName}");
            client_mkdir.RunCommand($"mkdir /usr/src/{DirectoryName}");
            client_mkdir.Disconnect();
            Thread.Sleep(100);

            using (SftpClient upload = new SftpClient(new PasswordConnectionInfo(prox_host,
prox_username, prox_password)))
            {
                upload.Connect();
                foreach (var file in VMDirectory)
                {
                    FileInfo fi = new FileInfo(@"" + file);
                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Uploading
{System.IO.Path.GetFileName(file)}"; });
                    long size = fi.Length;
                    using (Stream stream = File.OpenRead(file))
                    {
                        upload.UploadFile(stream, @"/usr/src/" + DirectoryName + "/" +
System.IO.Path.GetFileName(file), x =>
                        {
                            long current = long.Parse(x.ToString());
                            progress = ((double)current / size) * 70;
                            this.Dispatcher.Invoke(() => { pbProgress.Value = progress; });
                        });
                    }
                }
                upload.Disconnect();
                Thread.Sleep(100);
            }
            … code to be explained later (migrating to windows AND migrating to Linux)
        }
```

The beginning of this code should seem familiar, so I'm not going through this code again. My explanation starts at the "var client_mkdir" line. There I've created a new SshClient. Here I don't have to check if the connection is possible, because I've tested this in the previous step. But here the SshClient connects to the server and removes the previous VMDK files and folders, if they should exist.

The next thing we're going to do, is creating an SftpClient session to the Proxmox VE server. In here we'll have an foreach loop that goes through every VMDK file in our subdirectory. Per file that is read in, we're looking what size it is, in order to do this, we'll need to use the Length attribute of the FileInfo "fi" variable.

After this we're opening a Stream session that will upload the file for us. During the upload, the current size of the file is being store in var "x". we're using the var x to convert it into a long. Now with a double, we can calculate the current upload progress. This will fill 70% of our progress bar at the bottom of the application.

During the creation of the VM, you're going to see a lot of for loops. These loops are used to invoke some commands to the progress bar int order to fill it. When the upload of both files has been completed, the client will disconnect and continue with the code after some sleep to make sure everything is disconnected.

Now we're going to talk about the creation of our Windows Migration.

#### 2.6.4.7.2 Migrating to Windows

```
if (os_type != null)
            {
                AuthenticationMethod method_prox = new
PasswordAuthenticationMethod(prox_username, prox_password);
                ConnectionInfo connection_prox = new ConnectionInfo(prox_host, prox_username,
method_prox);
                var client = new SshClient(connection_prox);
                string start = $"qm start {start_vmid}";

                if (os_type == "win11" || os_type == "win10" || os_type == "win8" || os_type ==
"win7" || os_type == "w2k8" || os_type == "w2k3" || os_type == "w2k")
                {

                    string createVM = $"qm create {start_vmid} --balloon 1024 --memory {memory}
--sockets 1 --cores {cpu_cores} --onboot yes --name {DirectoryName} --ostype {os_type} --
bootdisk ide0 --net0 e1000,bridge=vmbr0,firewall=1 --scsihw virtio-scsi-pci --bios ovmf";
                    string importDisk = $"qm importdisk {start_vmid}
/usr/src/{DirectoryName}/{DirectoryName}.vmdk VM-Data --format raw";
                    string useDisk = $"qm set {start_vmid} --ide0 VM-Data:{start_vmid}/vm-
{start_vmid}-disk-0.raw";
                    string changeBootOrder = $"qm set {start_vmid} --boot order='ide0;net0'";
                    string setUefi = $"qm set {start_vmid} --efidisk0 VM-
Data:{start_vmid},format=qcow2,efitype=4m,pre-enrolled-keys=1";
                    string setTpm = $"qm set {start_vmid} --tpmstate0 VM-
Data:{start_vmid},version=v2.0";

                    client.Connect();


                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Creating VM
{DirectoryName}";});
                    var sendCommand = client.RunCommand(createVM);
                    for (int i = 70; i <= 75; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Importing VMDK
{DirectoryName}.vmdk to VM"; });
                    sendCommand = client.RunCommand(importDisk);
                    for (int i = 75; i <= 80; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Using Disk vm-
{start_vmid}-disk-0.raw"; });
                    sendCommand = client.RunCommand(useDisk);
                    for (int i = 80; i <= 85; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Changing boot order"; });
                    sendCommand = client.RunCommand(changeBootOrder);
                    for (int i = 85; i <= 90; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Creating EFI Disk"; });
                    sendCommand = client.RunCommand(setUefi);
                    for (int i = 90; i <= 95; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Creating TPM State"; });
```

```
                    sendCommand = client.RunCommand(setTpm);
                    for (int i = 95; i <= 100; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    sendCommand = client.RunCommand(start);
                    vmCreated = true;
                    client.Disconnect();
                }
```

After we've created an SshClient, we're making a lot of strings that contain the commands that have to be executed:

- start — Command used to start the Windows VM and the Linux VM
- createVM — Command used to create the Windows VM.
- importDisk — Command used to import the VMDK as a RAW virtual disk
- useDisk — Command to use the imported disk
- changeBootOrder — Command to change to boot order to set imported disk first
- setUefi — Command to create and set the EFI disk
- setTpm — Command to add an virtual TPM module to the VM

I chose to use an UEFI based Windows 10 VM to show that it's completely possible to migrate them, In my code, the UEFI is enabled and there's also a TPM module, so in theory, it's possible to just upgrade the windows 10 machine to windows 11, and the VM would still work

After this the code does the following per step:

- Change the content of Label "lblInfo" with the current progress of the creation
- Executing the command given above
- Updating the progress bar to the new state

The progress bar update has been put in an for loop in order to give it a smooth feeling.

If the VM has been created, it will start the VM and also change the bool "vmCreated" to true, this will trigger the steps to migrate the next VM.

#### 2.6.4.7.3 Migrating to Linux

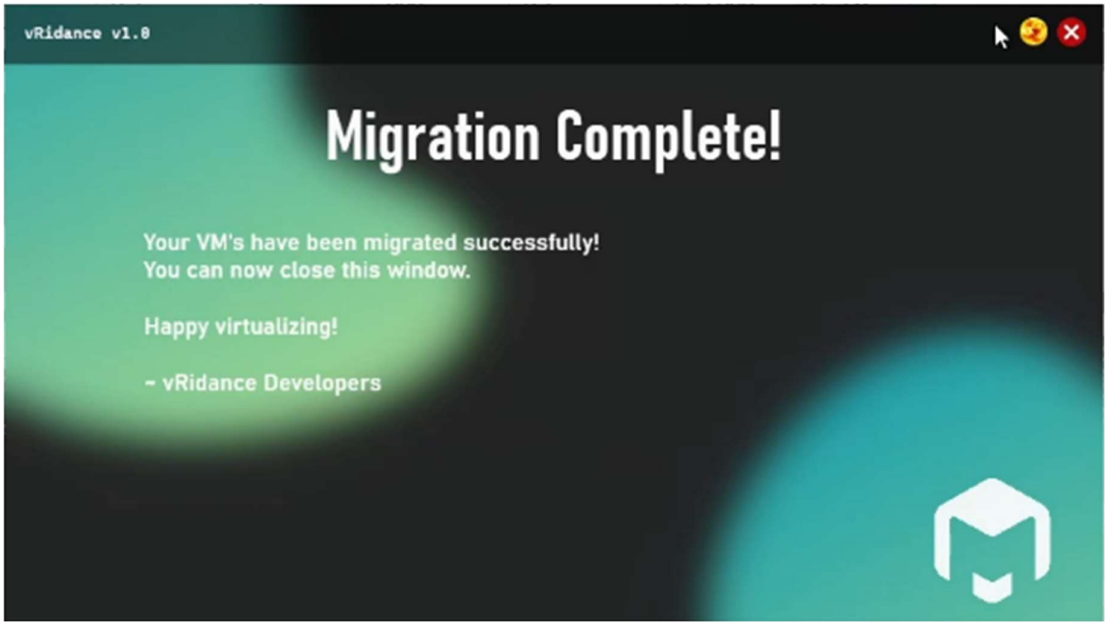
```
else if (os_type == "126" || os_type == "124")
                {
                    string createVM = $"qm create {start_vmid} --balloon 1024 --memory {memory}
--sockets 1 --cores {cpu_cores} --onboot yes --name {DirectoryName} -ostype {os_type} --bootdisk
scsi0 --net0 virtio,bridge=vmbr0,firewall=1 --scsihw virtio-scsi-pci";
                    string importDisk = $"qm importdisk {start_vmid}
/usr/src/{DirectoryName}/{DirectoryName}.vmdk VM-Data --format raw";
                    string useDisk = $"qm set {start_vmid} --scsi0 VM-Data:{start_vmid}/vm-
{start_vmid}-disk-0.raw";
                    string changeBootOrder = $"qm set {start_vmid} --boot order='scsi0;net0'";

                    client.Connect();

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Creating VM
{DirectoryName}"; });
                    var sendCommand = client.RunCommand(createVM);
                    for (int i = 70; i <= 77; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Importing VMDK
{DirectoryName}.vmdk to VM"; });
                    sendCommand = client.RunCommand(importDisk);
                    for (int i = 77; i <= 86; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Using Disk vm-
{start_vmid}-disk-0.raw"; });
                    sendCommand = client.RunCommand(useDisk);
                    for (int i = 86; i <= 93; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    this.Dispatcher.Invoke(() => { lblInfo.Content = $"Changing boot order"; });
                    sendCommand = client.RunCommand(changeBootOrder);
                    for (int i = 93; i <= 101; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }

                    sendCommand = client.RunCommand(start);
                    vmCreated = true;
                    client.Disconnect();
                }
```

This code does the same as previous code, but there are some steps that this code doesn't do, these steps being:

- setUefi
- setTpm

Linux runs on BIOS; not on UEFI, so you don't need to set the UEFI, neither the TPM module.

### 2.6.5 EndScreen.xaml.cs




After the migration has been completed, the thread will end. Before the thread is ending, it will send you to the EndScreen. There is no Proxmox-specific code here.

Happy Virtualizing!

# 3 Azure Stack HCI

## 3.1 What is Azure Stack HCI

"Azure Stack HCI is a new hyperconverged infrastructure (HCI) operating system delivered as an Azure service." [1]

It can make use of Azure services like:

- Cloud backup
- Site recovery
- Cloud based (security) monitoring

### 3.1.1 Pricing

The first 2 months are free but after that you pay 9 dollars each month for every core in your cluster. So when you have a cluster with 16 cores you pay 16x9 dollar per month.

Azure Stack HCI does checks on how many physical cores you have and sends that information to Azure for billing.

You can also make use of Azure services, this will be linked to your Azure Subscription account.

Windows Admin Center is a free software that you use with Azure Stack HCI.

You also get the latest updates and features without paying extra.

## 3.2 Building the environment

### 3.2.1 Requirements

For this demonstration I'm building the cluster with the bear minimum. This will also be not as optimale because we are virtualising inside virtualisation, keep this in mind.

Before we can start we need a few things.

- 2 Windows servers 2022
    - 1 for the domain controller (DC)
    - 1 for running admin center, this can't be a domain controller because this is not supported.
- 2 Azure Stack HCI machines (nodes)

### 3.2.2 How can you download the ISO?

This can be downloaded from azure.microsoft.com, after registration you get an email with the download link for the ISO, keep in mind this is a free trail version for only 60 days.

### 3.2.3 Creating the Virtual Machines

The Azure Stack HCI machines should ideally be on dedicated hardware, it is not at all recommended to use VMs. When using VMs you will have a huge performance problem.

The minimum requirements:

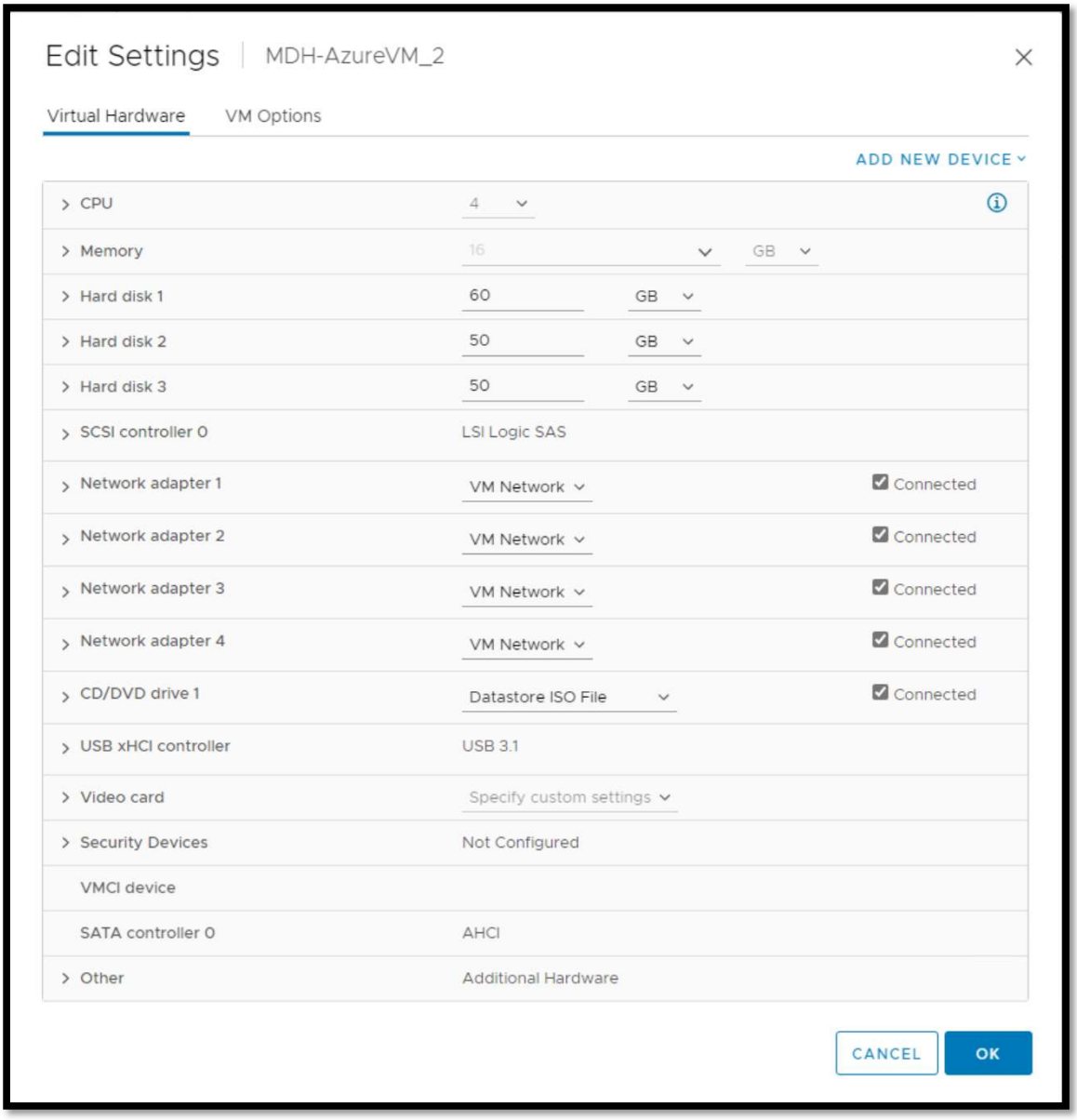
- 2 nodes
- 4 SSD or 2 SSD + 4HDD per server
- No minimum processor and memory
- 1 x 10 Gbps network adapter

The maximum requirements:

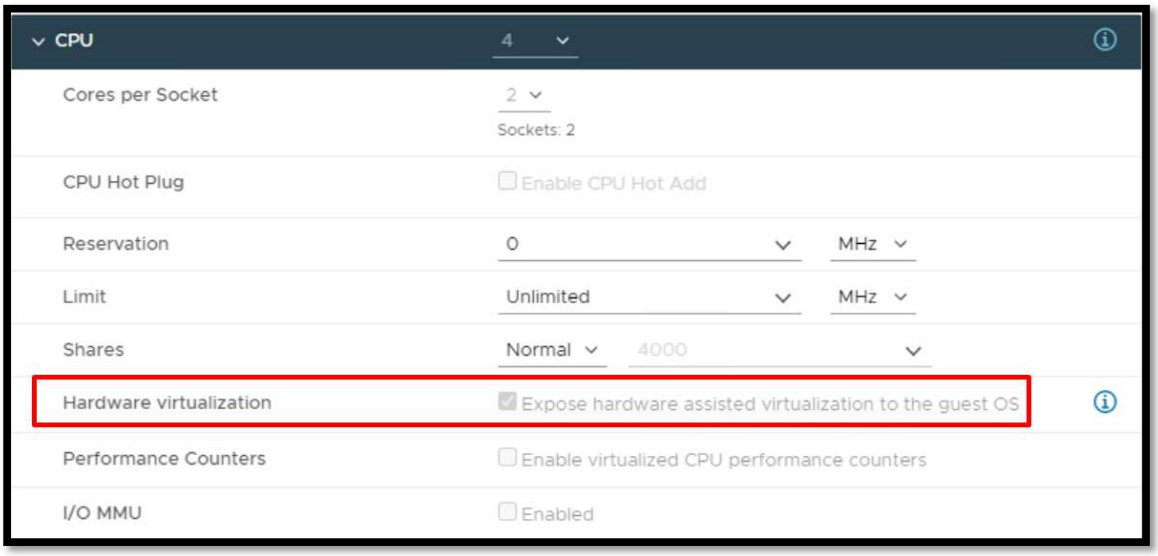
- 16 nodes
- 16 000 TB storage per clusters

My virtual machines, once again this is not best practice and was used in a testing environment.

In my case I used 2 nodes with identical specs.

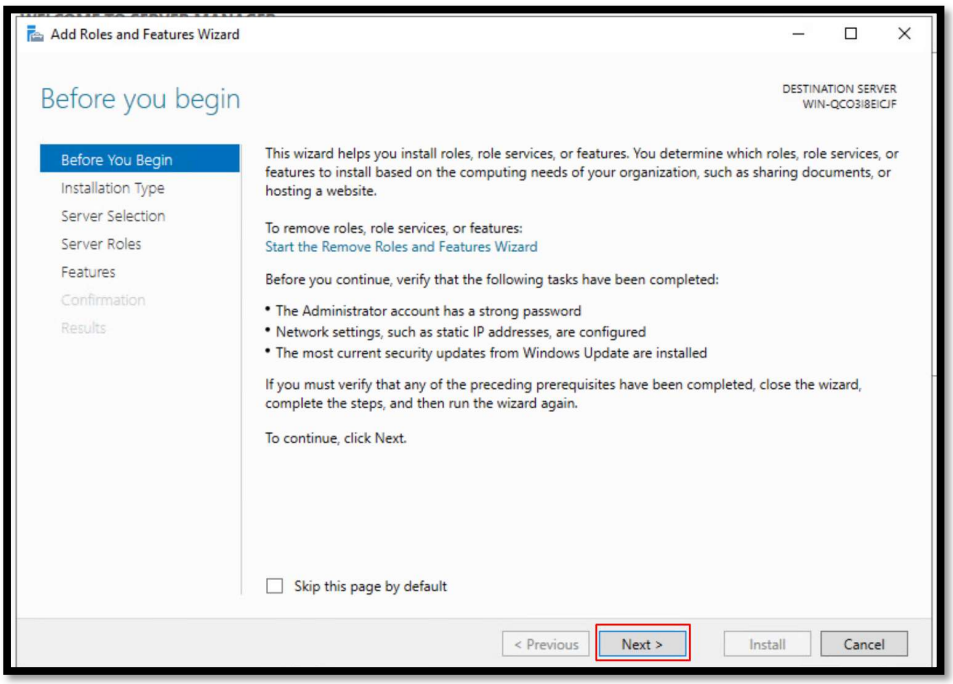


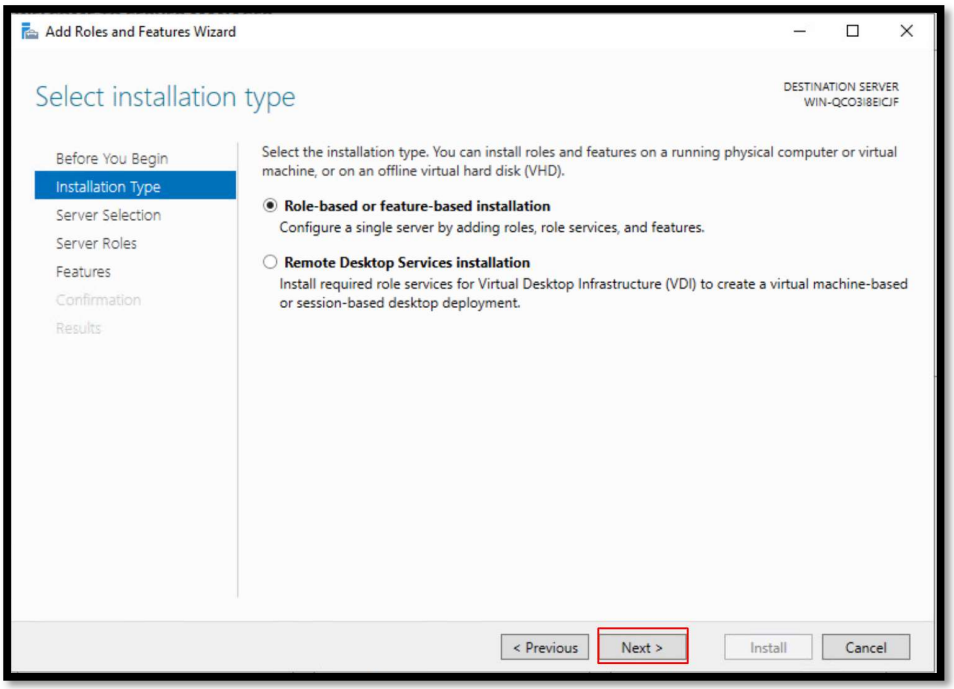
Make sure under CPU that hardware virtualization is enabled.



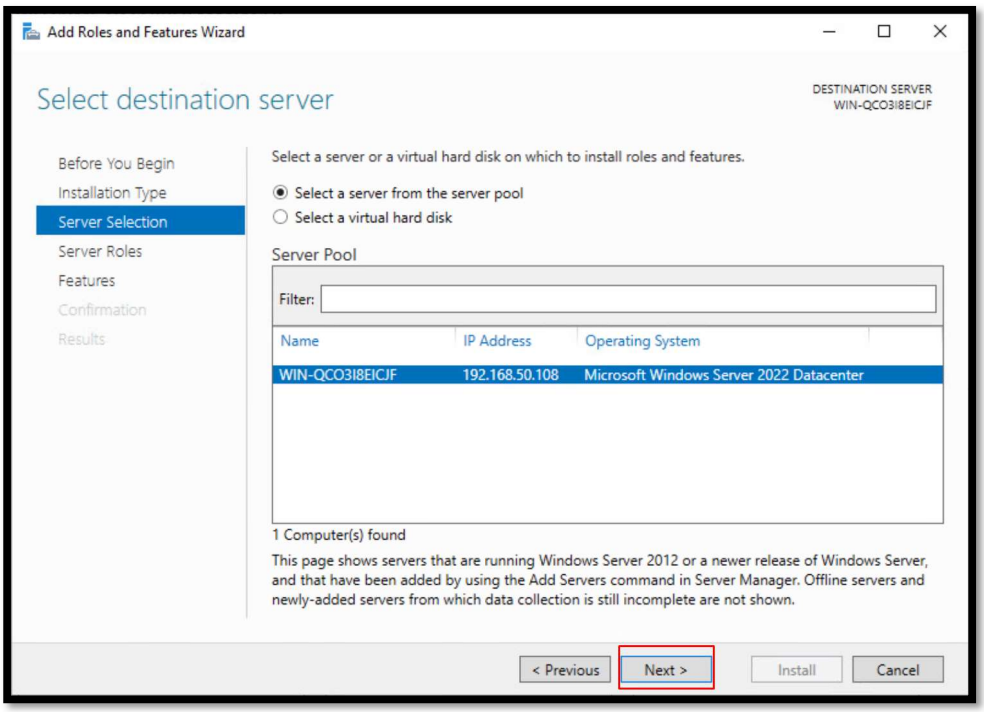

### 3.2.4 Create the domain

On the DC in server manager we need to add some roles and features.

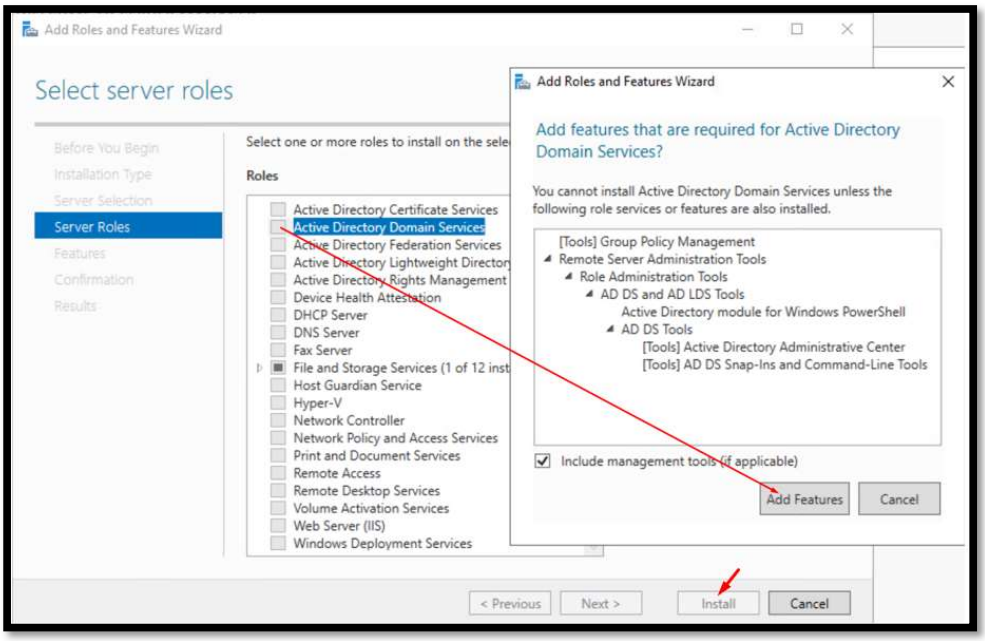


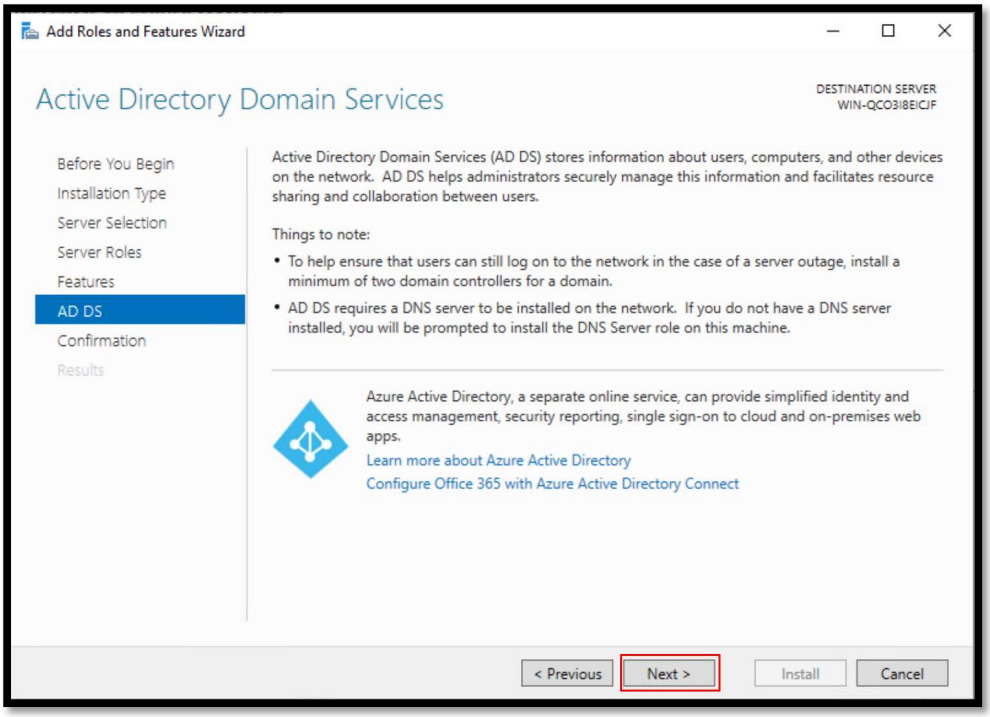


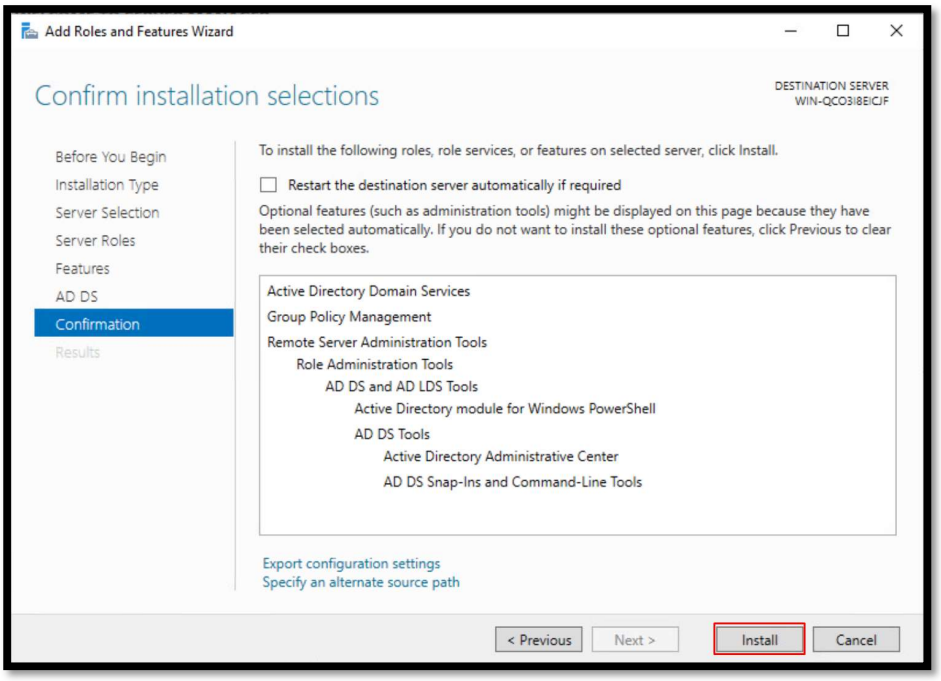

Select your server, in this case it will be the DC.



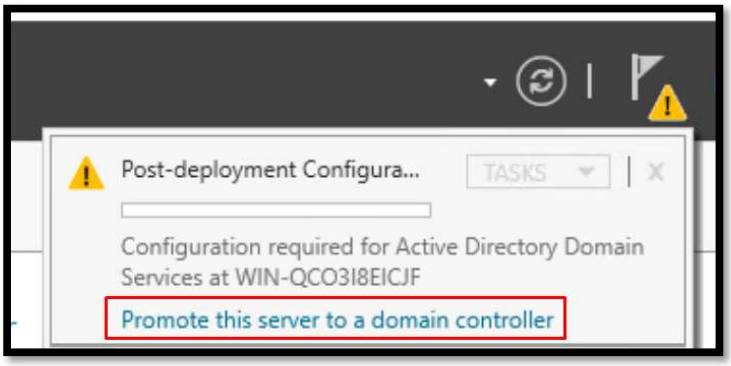

Here we must install “Active Directory Domain Services”. This is needed for creating the domain.


Add Roles and Features Wizard
Select server roles
Before You Begin
Installation Type
Server Selection
Server Roles
Features
Confirmation
Results
Select one or more roles to install on the sele
Roles
Active Directory Certificate Services
Active Directory Domain Services
Active Directory Federation Services
Active Directory Lightweight Director
Active Directory Rights Management
Device Health Attestation
DHCP Server
DNS Server
Fax Server
File and Storage Services (1 of 12 inst
Host Guardian Service
Hyper-V
Network Controller
Network Policy and Access Services
Print and Document Services
Remote Access
Remote Desktop Services
Volume Activation Services
Web Server (IIS)
Windows Deployment Services
Add Roles and Features Wizard
Add features that are required for Active Directory Domain Services?
You cannot install Active Directory Domain Services unless the following role services or features are also installed.
[Tools] Group Policy Management
Remote Server Administration Tools
Role Administration Tools
AD DS and AD LDS Tools
Active Directory module for Windows PowerShell
AD DS Tools
[Tools] Active Directory Administrative Center
[Tools] AD DS Snap-Ins and Command-Line Tools
Include management tools (if applicable)
Add Features
Cancel
< Previous
Next >
Install
Cancel

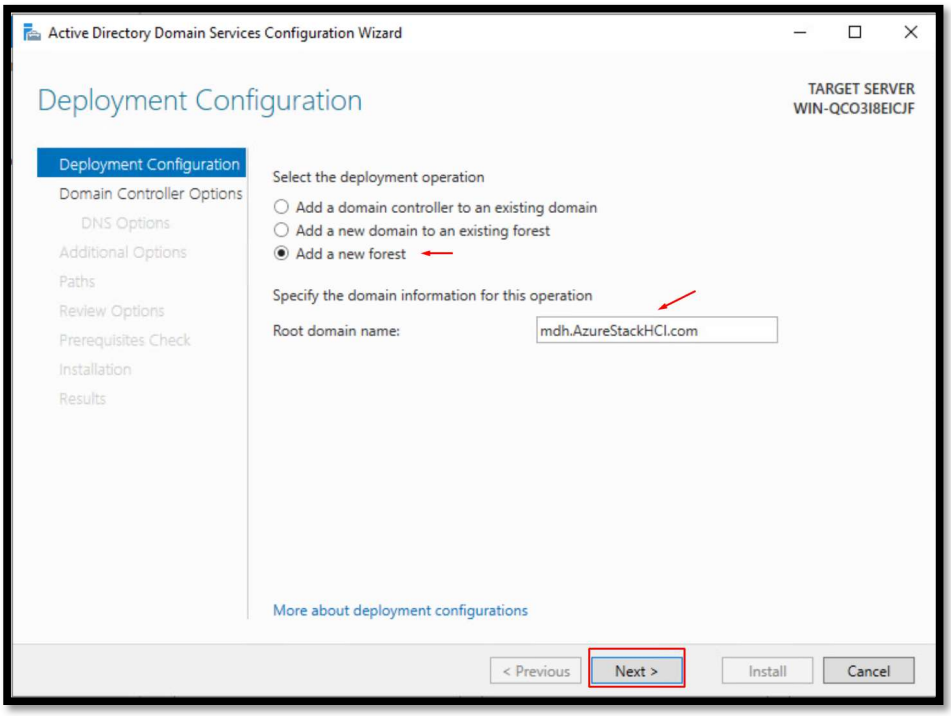

Add Roles and Features Wizard
Active Directory Domain Services
DESTINATION SERVER
WIN-QCO3I8EICJF
Before You Begin
Installation Type
Server Selection
Server Roles
Features
AD DS
Confirmation
Results
Active Directory Domain Services (AD DS) stores information about users, computers, and other devices on the network. AD DS helps administrators securely manage this information and facilitates resource sharing and collaboration between users.
Things to note:
• To help ensure that users can still log on to the network in the case of a server outage, install a minimum of two domain controllers for a domain.
• AD DS requires a DNS server to be installed on the network. If you do not have a DNS server installed, you will be prompted to install the DNS Server role on this machine.
Azure Active Directory, a separate online service, can provide simplified identity and access management, security reporting, single sign-on to cloud and on-premises web apps.
Learn more about Azure Active Directory
Configure Office 365 with Azure Active Directory Connect
< Previous
Next >
Install
Cancel



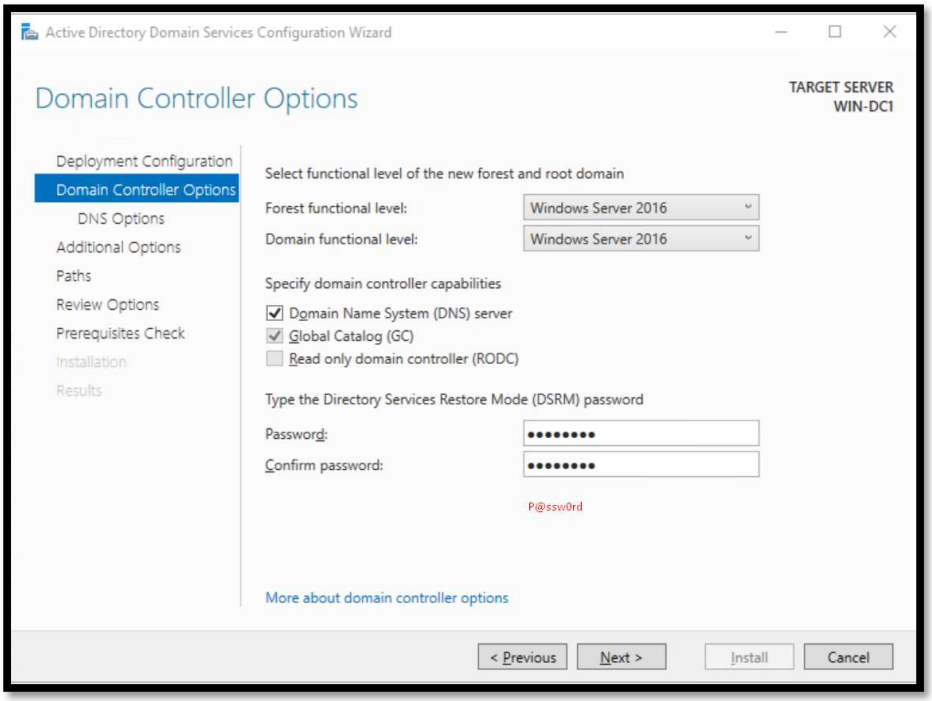

Back on the server manager in the right hand corner you should see this after the installation.

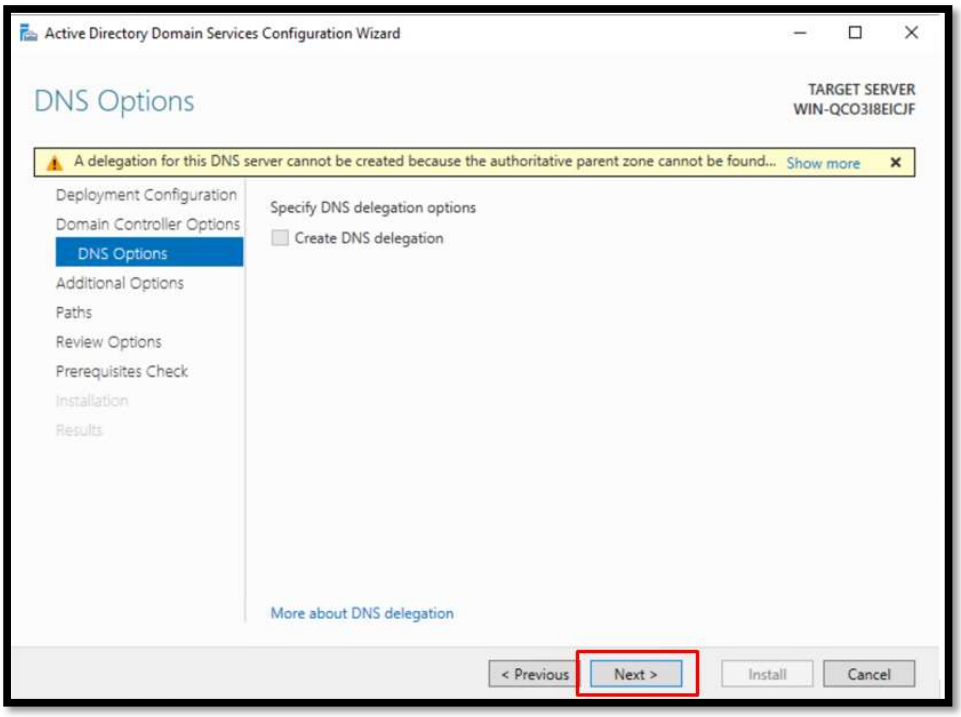


We create a new forest and enter the root domain name

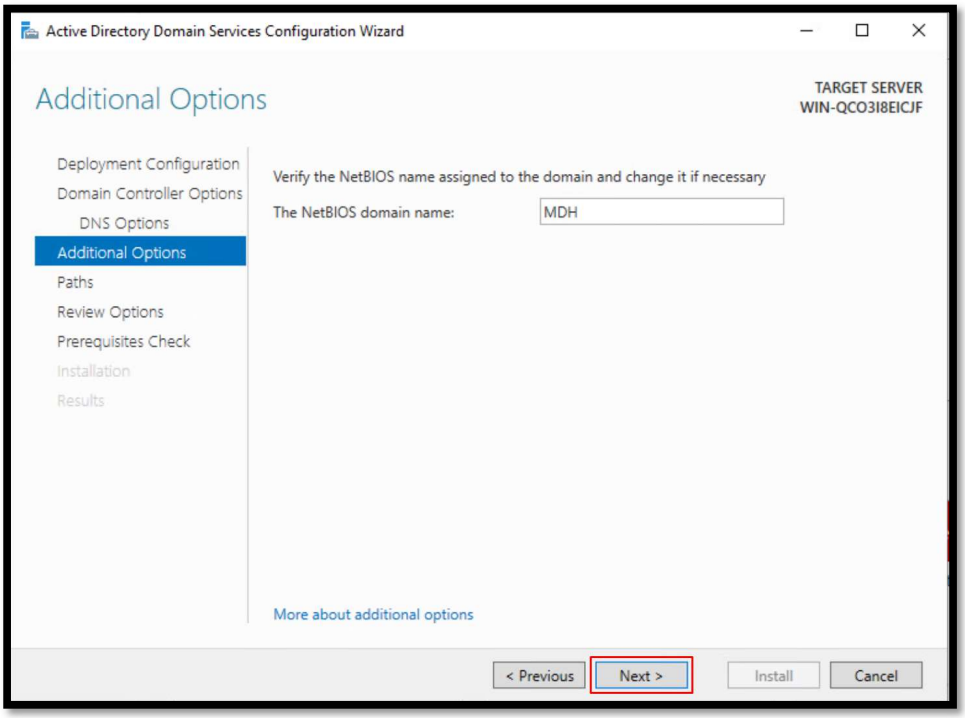


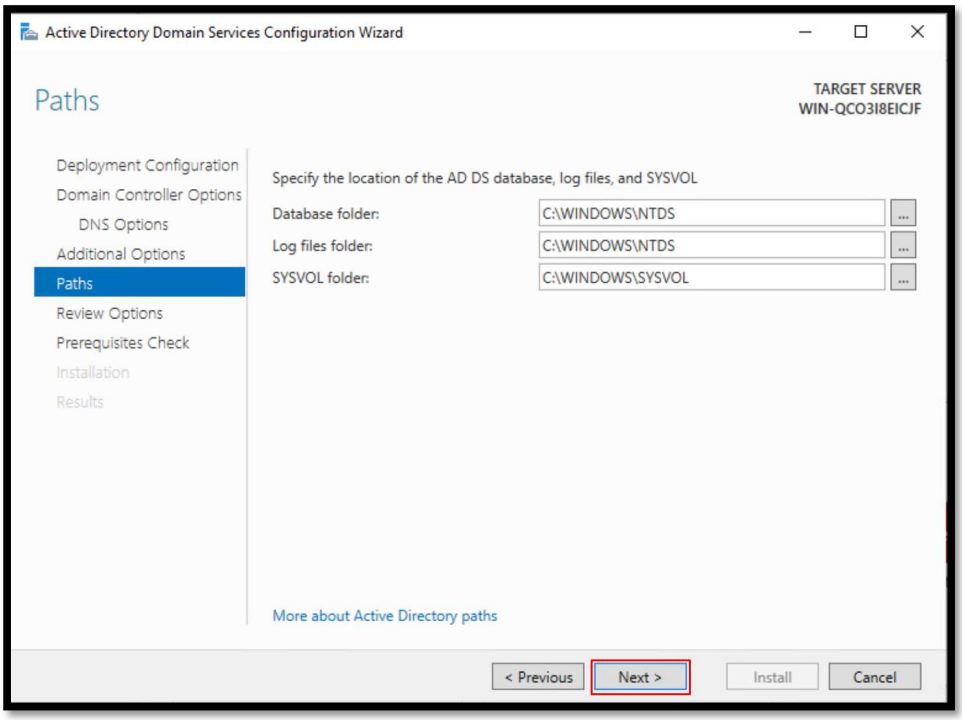

Give your deomain a strong password, in my test environment I used a weak password that never should be used in a production environment!

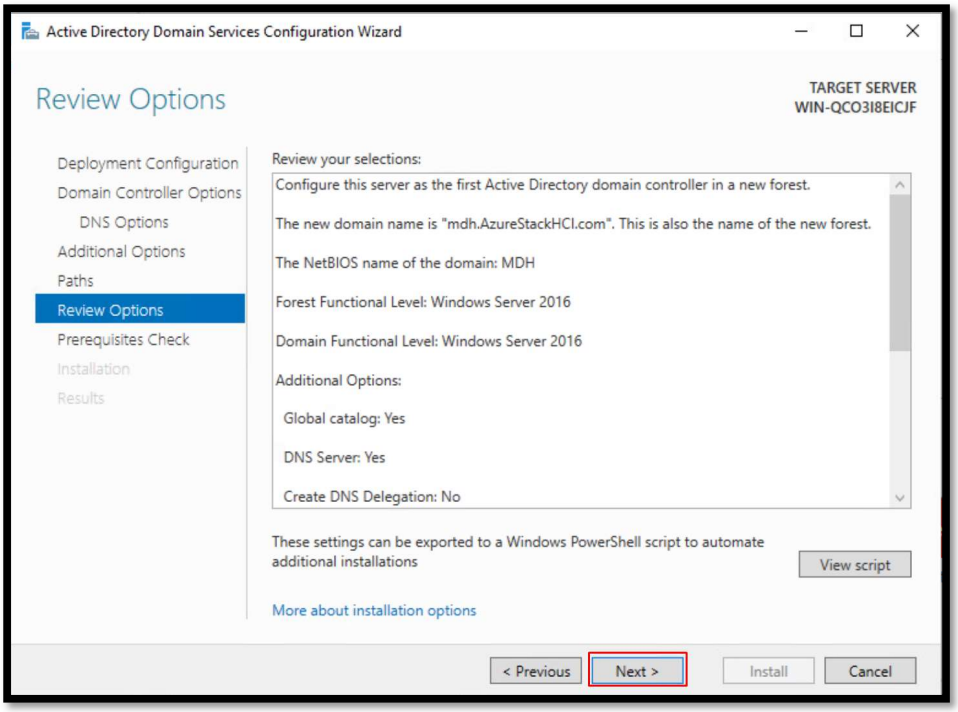


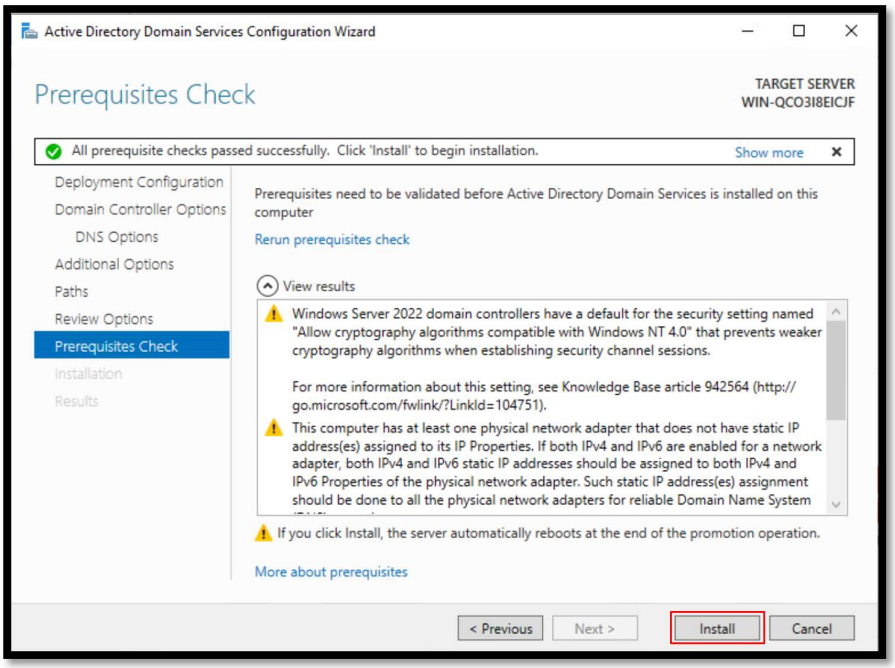



This is automatically generated, wait a few seconds for it to load.




Active Directory Domain Services Configuration Wizard
Paths
TARGET SERVER
WIN-QCO3I8EICJF
Deployment Configuration
Domain Controller Options
DNS Options
Additional Options
Paths
Review Options
Prerequisites Check
Installation
Results
Specify the location of the AD DS database, log files, and SYSVOL
Database folder:
C:\WINDOWS\NTDS
Log files folder:
C:\WINDOWS\NTDS
SYSVOL folder:
C:\WINDOWS\SYSVOL
More about Active Directory paths
< Previous
Next >
Install
Cancel


Active Directory Domain Services Configuration Wizard
Review Options
TARGET SERVER
WIN-QCO3I8EICJF
Deployment Configuration
Domain Controller Options
DNS Options
Additional Options
Paths
Review Options
Prerequisites Check
Installation
Results
Review your selections:
Configure this server as the first Active Directory domain controller in a new forest.
The new domain name is "mdh.AzureStackHCI.com". This is also the name of the new forest.
The NetBIOS name of the domain: MDH
Forest Functional Level: Windows Server 2016
Domain Functional Level: Windows Server 2016
Additional Options:
Global catalog: Yes
DNS Server: Yes
Create DNS Delegation: No
These settings can be exported to a Windows PowerShell script to automate additional installations
View script
More about installation options
< Previous
Next >
Install
Cancel




After the domain is created, all the other VM's must join the domain. You can do this manually now or later in Windows Admin Center. I choose to do it now. Keep in mind Windows Admin Center only adds our Azure Stack HCI VM's to our domain. From now on the Azure Stack HCI VM's will be called nodes, because they are nodes from the cluster that we are building. The other windows server (that will be running Windows Admin Center) we must join it ourselves to the domain.

## 3.2.5 Windows Admin Center

### 3.2.5.1 What is Windows Admin Center?

A tool for remote server management.

- You can manage servers
- Create clusters
- Create virtual machines
- Create servers

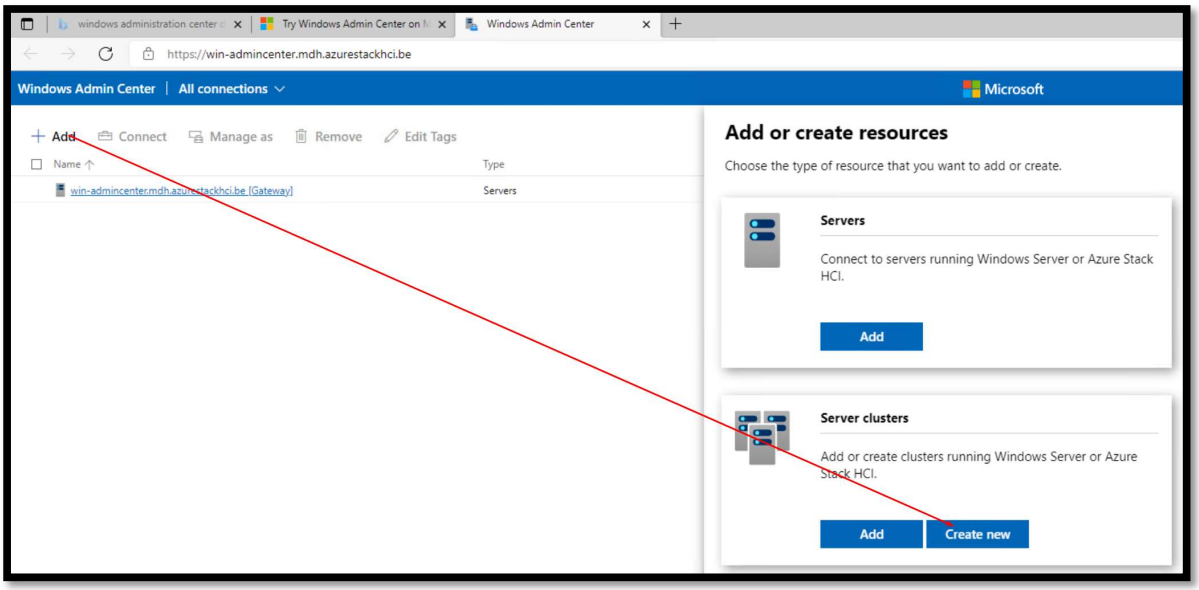
### 3.2.5.2 Install Windows Admin Center

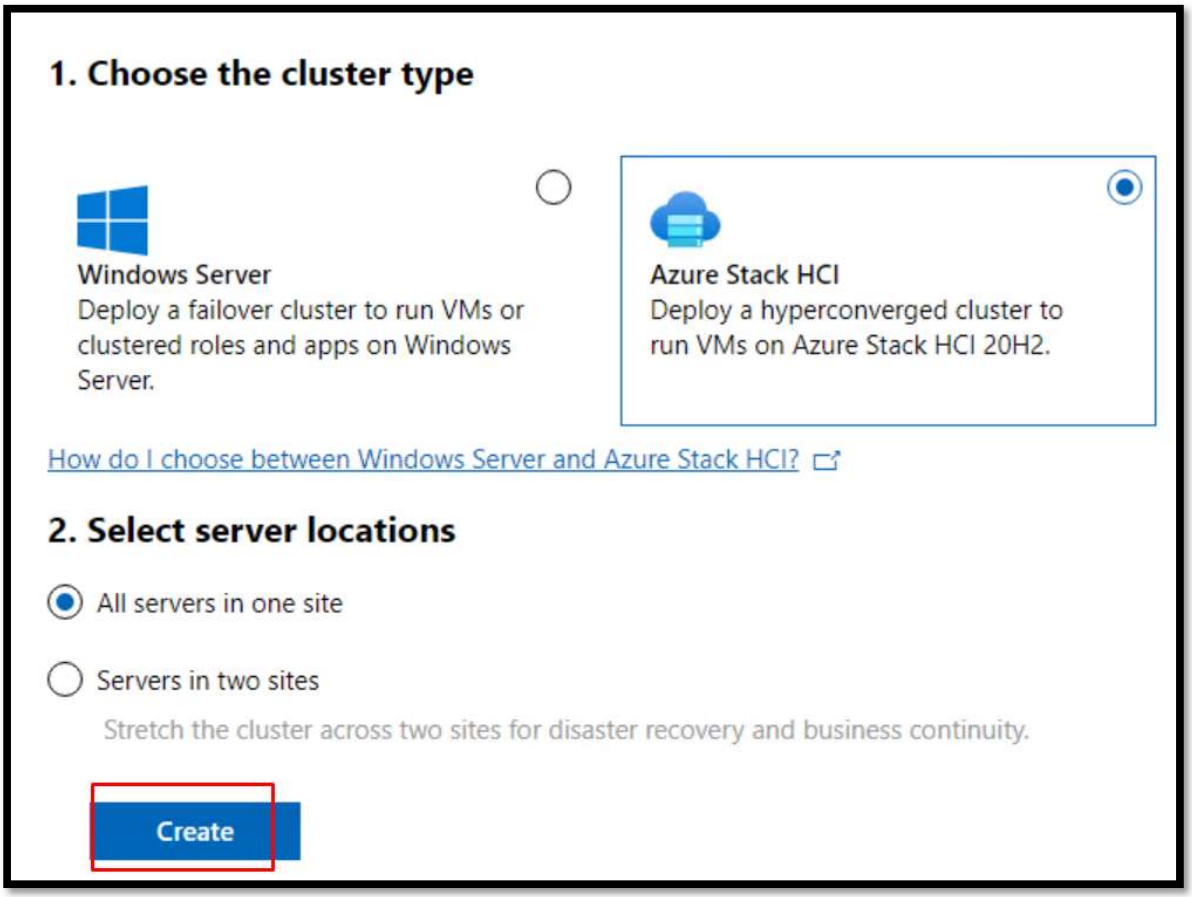
Is a free to download software at this link.

https://www.microsoft.com/en-us/evalcenter/evaluate-windows-admin-center

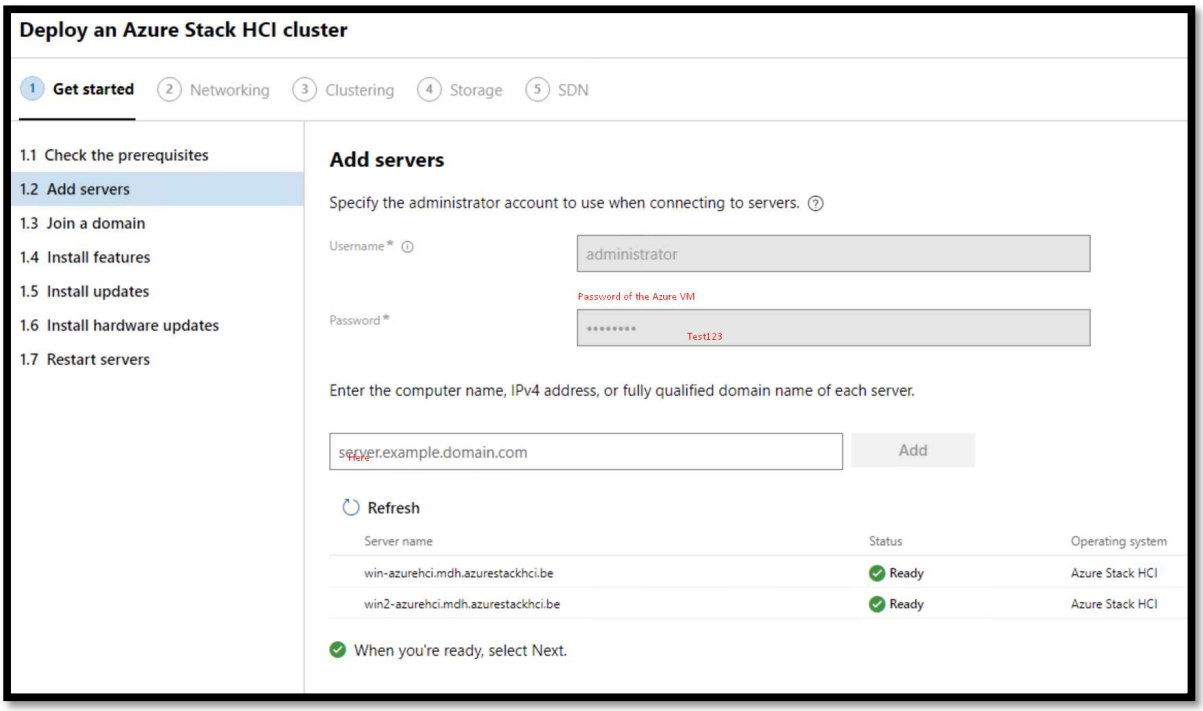
### 3.2.5.3 Configure Windows Admin Center

https://win-admincenter.mdh.azurestackhci.be

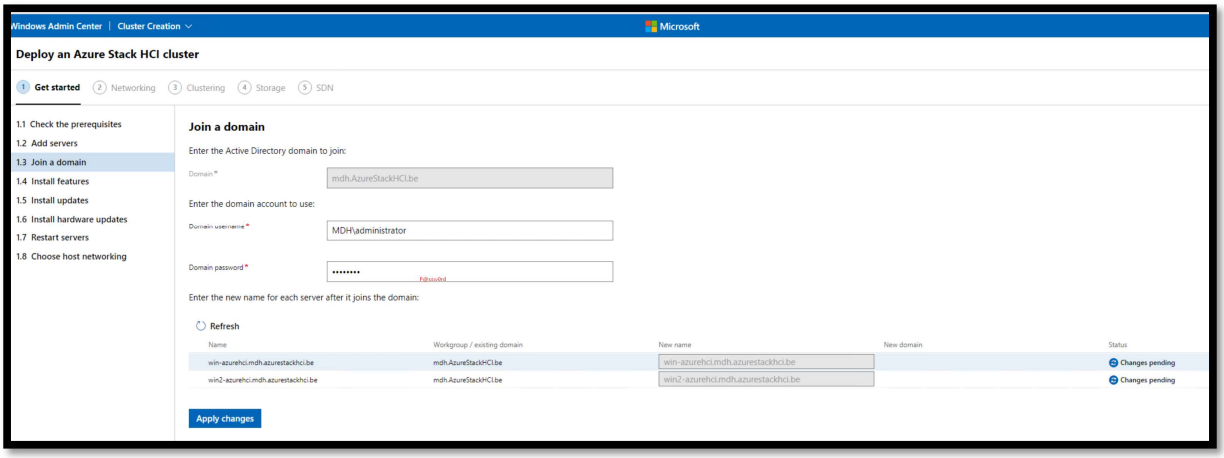

Windows Admin Center
https://win-admincenter.mdh.azurestackhci.be
Windows Admin Center | All connections
Microsoft
Add
Connect
Manage as
Remove
Edit Tags
Name
Type
win-admincenter.mdh.azurestackhci.be [Gateway]
Servers
Add or create resources
Choose the type of resource that you want to add or create.
Servers
Connect to servers running Windows Server or Azure Stack HCI.
Add
Server clusters
Add or create clusters running Windows Server or Azure Stack HCI.
Add
Create new


1. Choose the cluster type
Windows Server
Deploy a failover cluster to run VMs or clustered roles and apps on Windows Server.
Azure Stack HCI
Deploy a hyperconverged cluster to run VMs on Azure Stack HCI 20H2.
How do I choose between Windows Server and Azure Stack HCI?
2. Select server locations
All servers in one site
Servers in two sites
Stretch the cluster across two sites for disaster recovery and business continuity.
Create

Here we must add our 2 nodes. You can do this by IP address or hostname. Make sure that the username and password match those of the nodes.

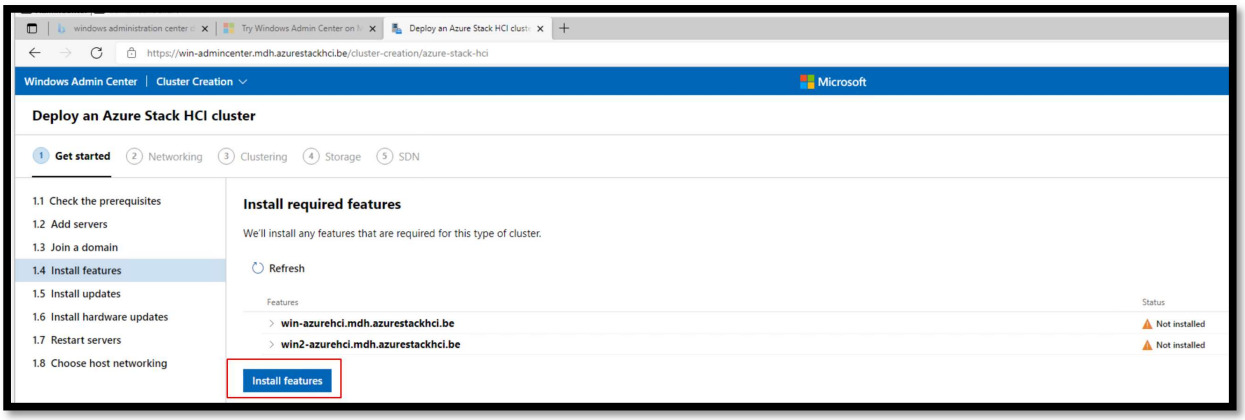



Here we can join our 2 nodes to the domain if needed.

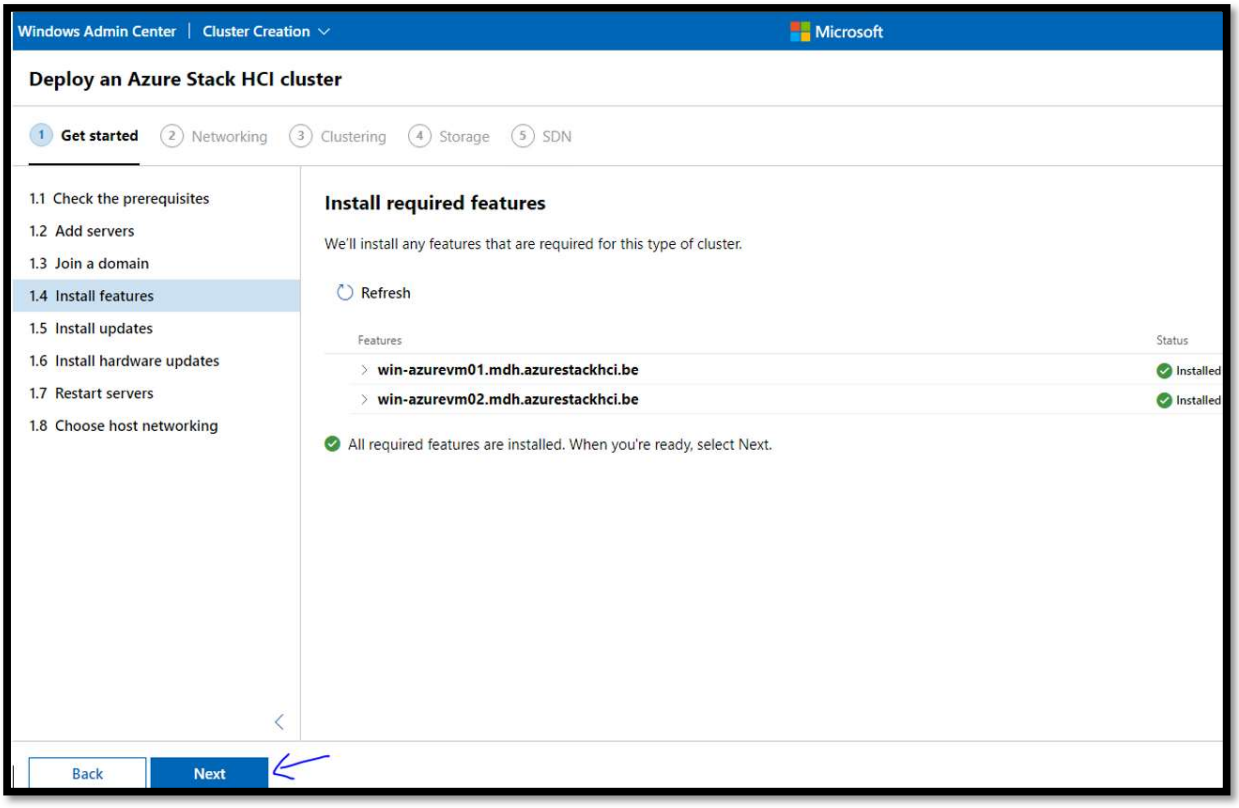


Install required features to the nodes, here I encounterd a problem with on of the features. Hyper-V has trouble installing, I found a solution to this problem. Best to manually install the feature on the nodes with the following powershell command:

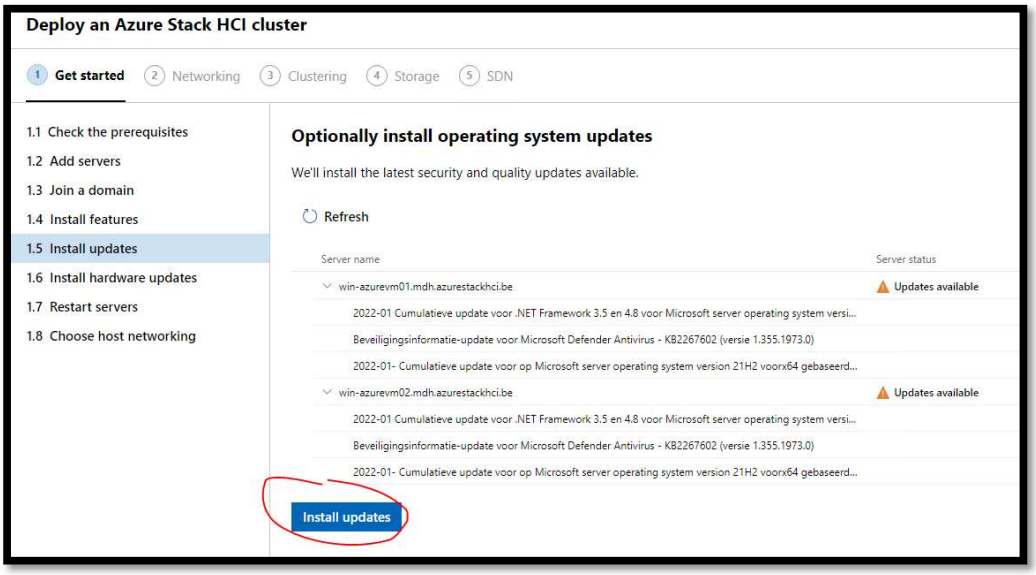
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All



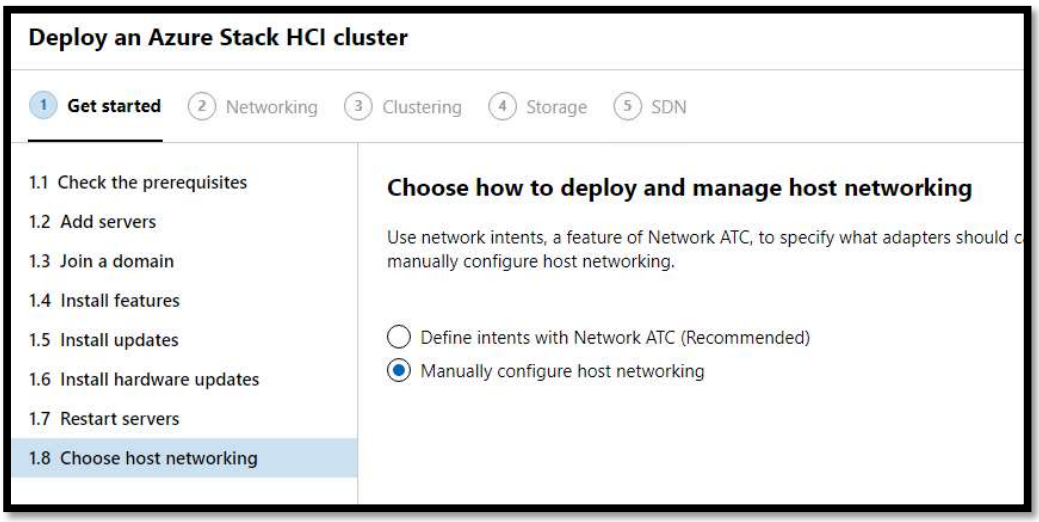

After all the required features are installed we can go to the next step.



Install the latest Windows updates




After the installation we can go the the next step “Install hardware updates”, we can just press next here. Now at the next step we must reboot our nodes. After the nodes are back online go to the next step “Choose host networking”. Here we choose to manually configure the host networking.

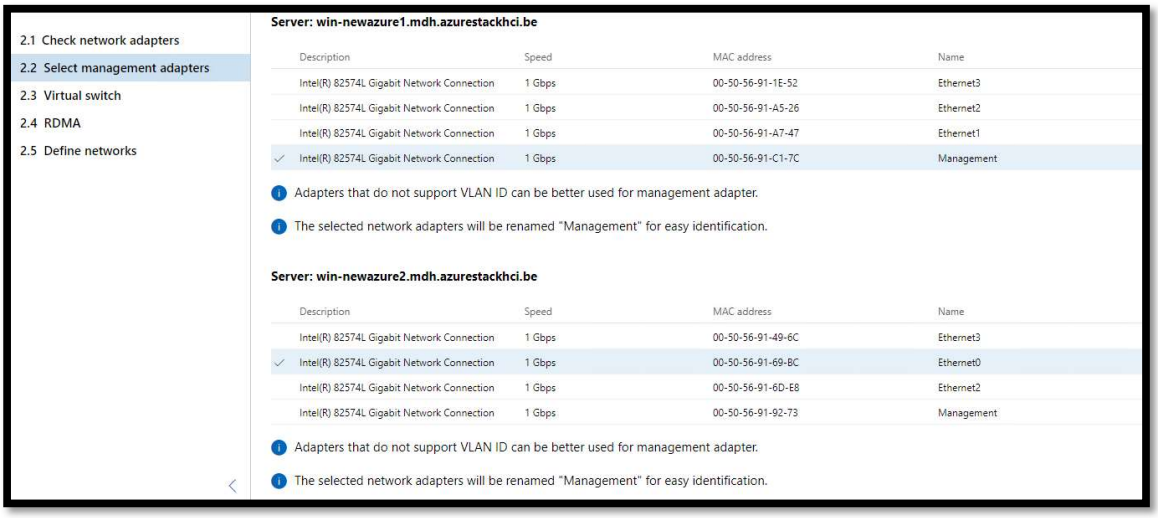


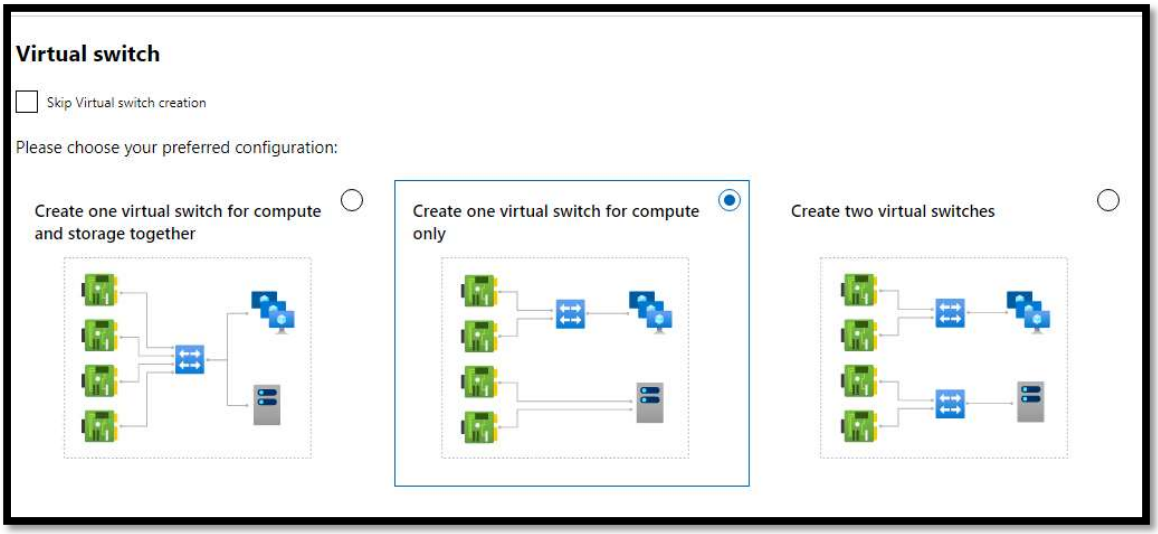

In the first step of the Networking tab, Windows Admin Center checks our network adapters and sees how many we have. After this step is completed we can go to the next one.

Here I made a mistake, I chose for the second node the wrong adapter first, that’s why I’m not selecting “Management” here. For both VM’s I want to select Ethernet0 but after you select an adapter and press next it automatically changes the name to “Management” hence why I’m selecting Ethernet0 on the second VM and not the adapter with the name “Management”.

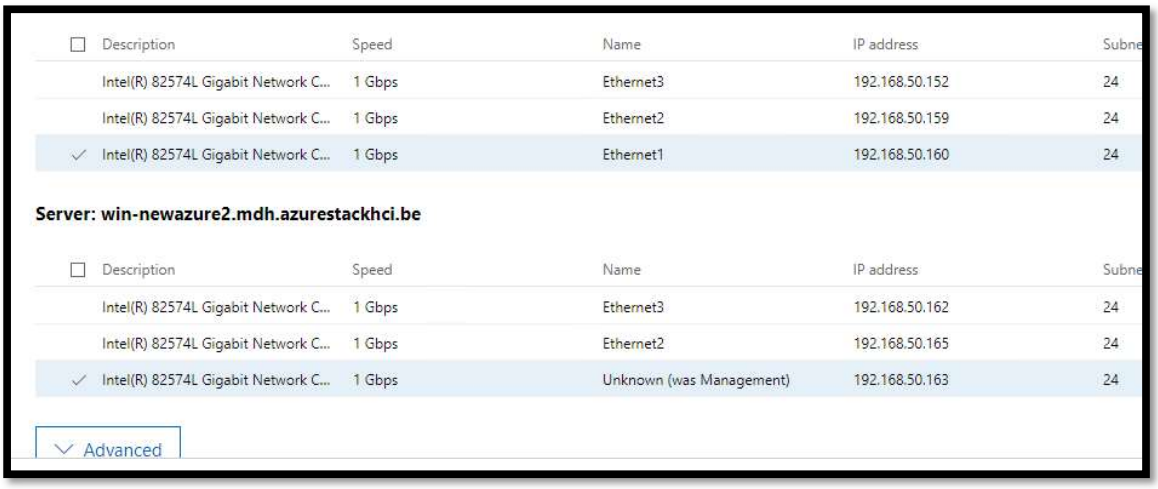
Is this crucial that both of them are Ethernet 0? Well not really but for consistency and Clarity it is important.

After selecting the right adapters we can go to the next step.

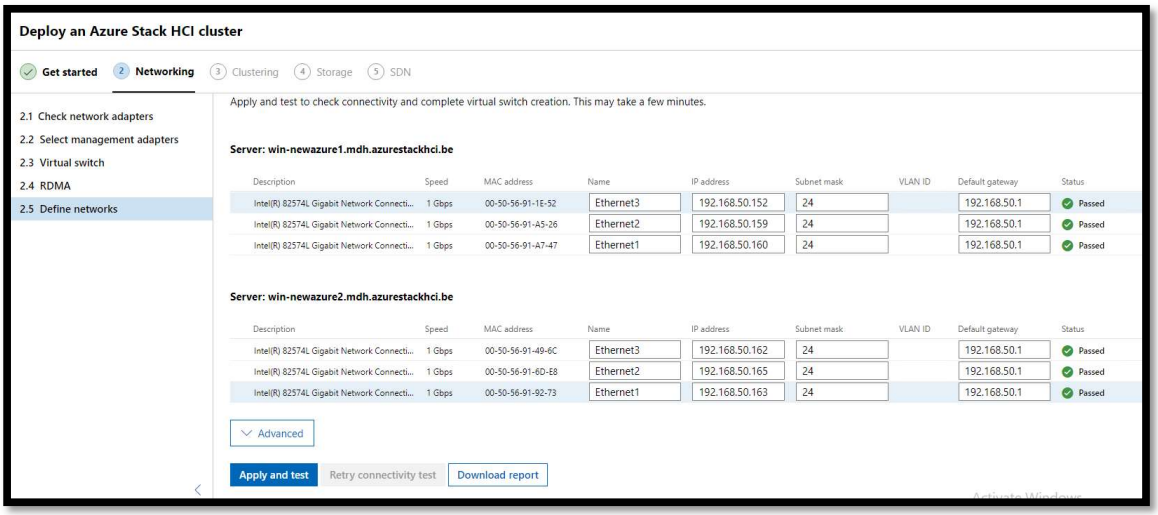


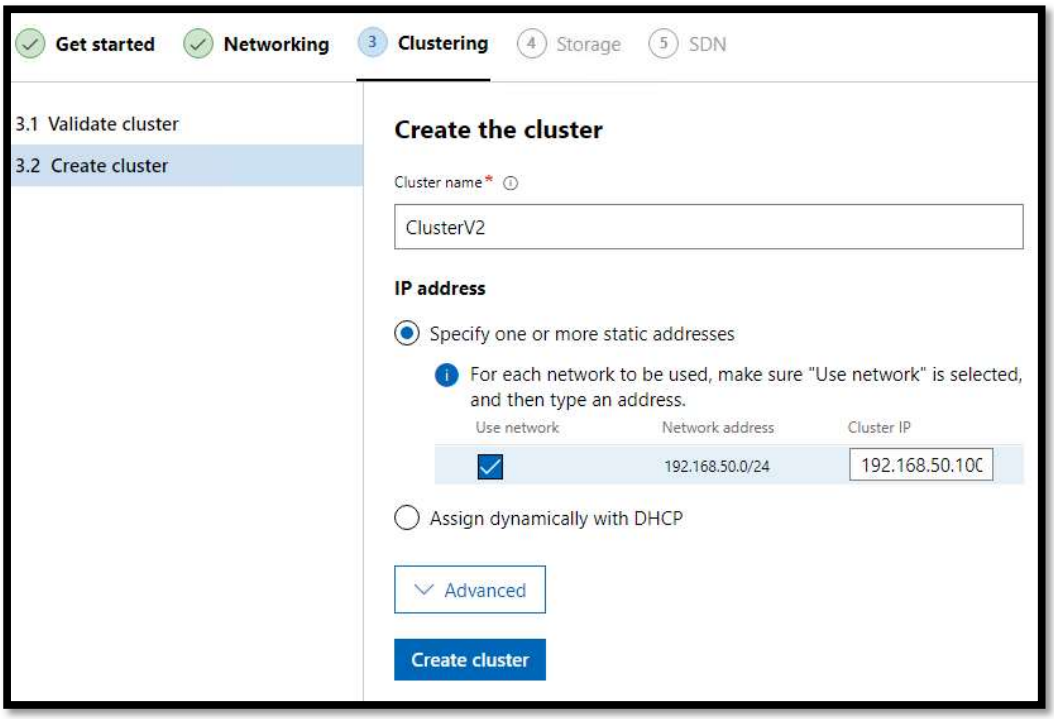
Here we both select Ethernet1, keep in mind that the name on the second node is not Ethernet1 because that was the old management adapter that I by accident chose in the previouse step.




Here in the final step of Networking we can change the network configuration of the adapters if needed.

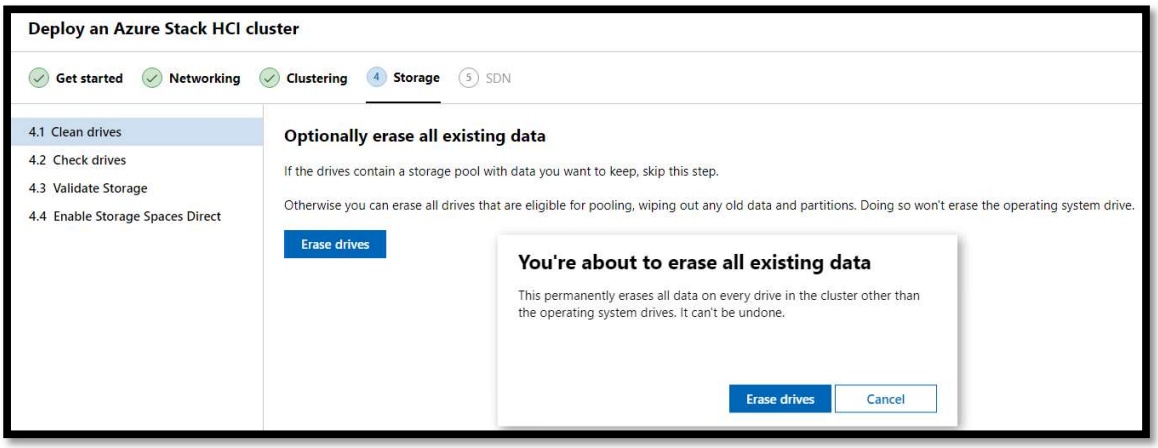


Here we are creating the cluster, but first we must validate it. After the validation we can give the cluster a name and an IP address.

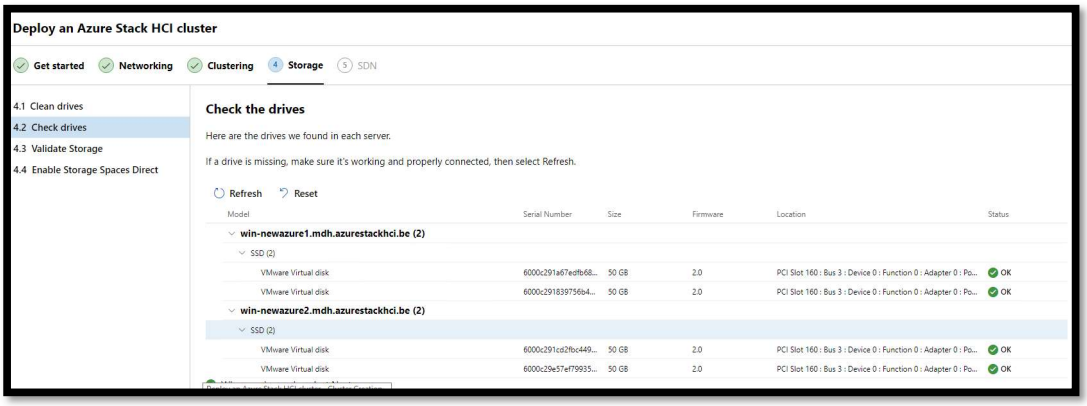



On the final tab “Storage” we are going to check and validate our storage on the nodes.

Here we can erase the drives, keep in mind that it removes everything on the disks except the OS.

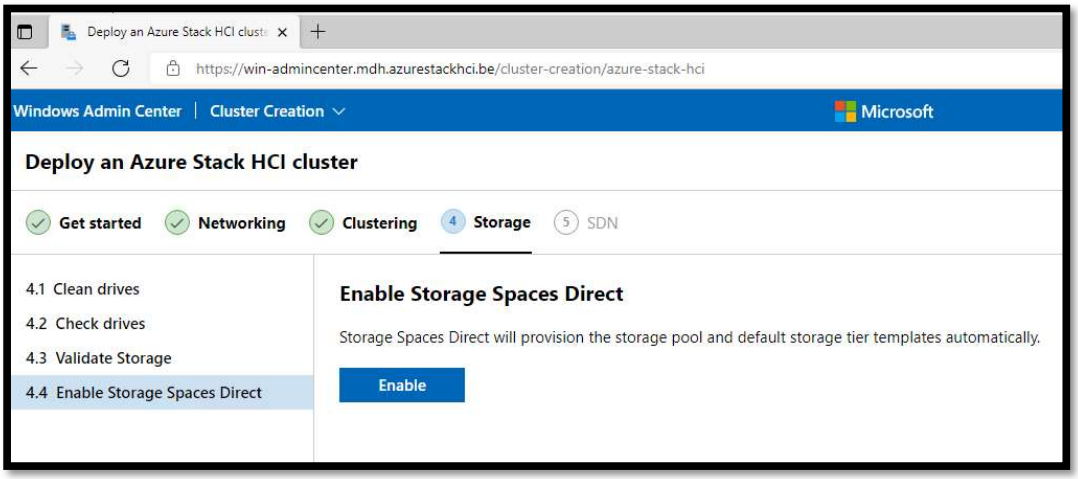


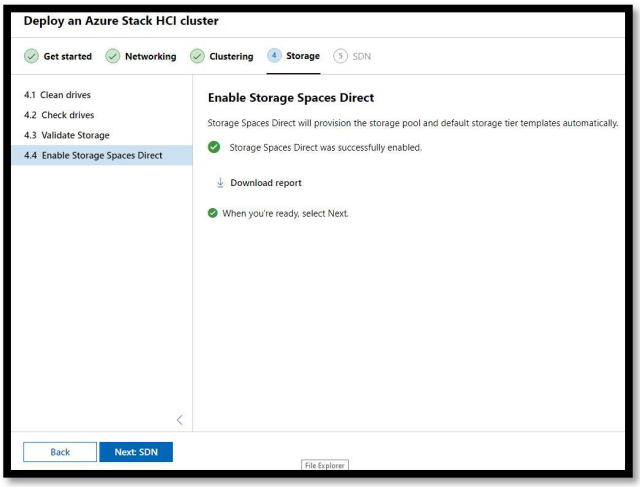
Here we can see our storage that is attached to our nodes.




Next Windows admin Center is going to validate the storage, after the validation we can go to the last step “Enable Storage Space Direct”.

Select enable.




Now go to SDN but here you can skip this part, this is not needed in my case.

### 3.2.5.4 Powershell

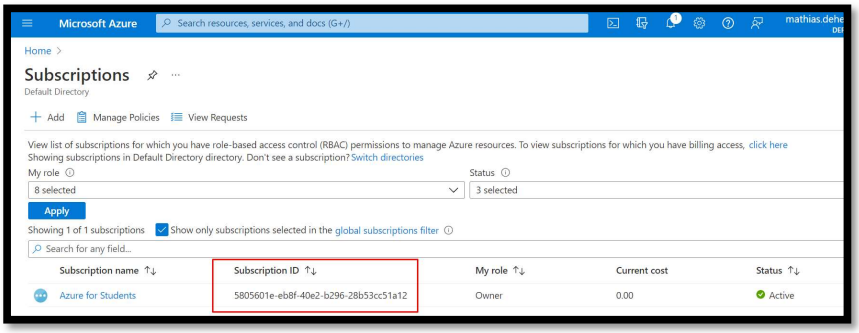
On our Windows Server that runs Windows Admin Center open powershell. This are the commando's that are needed.

```
# Install the modules
# ------------------------
# install the necessary components on the azure stack HCI nodes
$nodes = "WIN-NEWAZURE1", "WIN-NEWAZURE2"

Invoke-Command -ComputerName $nodes -ScriptBlock {
    install-windowsfeature RSAT-Azure-Stack-HCI
}

# install the powershell module locally
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Force
Install-Module Az.StackHCI

# check the current registration status
Invoke-Command -ComputerName "WIN-NEWAZURE1" -ScriptBlock {
    get-azurestackHCI
}


# Registration
# ------------------------
Register-AzStackHCI `
    -SubscriptionId "5805601e-eb8f-40e2-b296-28b53cc51a12" `
    -ResourceName "ClusterV2" `
    -ResourceGroupName "WE-MDH-AzureStack" `
    -EnvironmentName "AzureCloud" `
    -ComputerName "WIN-NEWAZURE1" `
    -Credential $azhciNodeCreds
```

### 3.2.5.5 Registration explained

```
# Registration
# ------------------------
Register-AzStackHCI `
    -SubscriptionId "5805601e-eb8f-40e2-b296-28b53cc51a12" `
    -ResourceName "ClusterV2" `
    -ResourceGroupName "WE-MDH-AzureStack" `
    -EnvironmentName "AzureCloud" `
    -ComputerName "WIN-NEWAZURE1" `
    -Credential $azhciNodeCreds
```

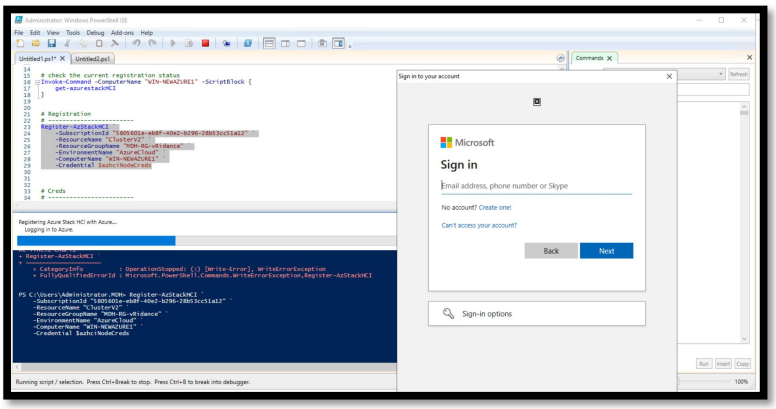
- SubscriptionID = the ID of your subscription




- ResourceName = the name of the cluster you created
- ResourceGroupName = the name of your resource group, keep in mind that the region North Europe is not supported for your resource group.
- EnviromentName = the name of your environment, this can be whatever you want

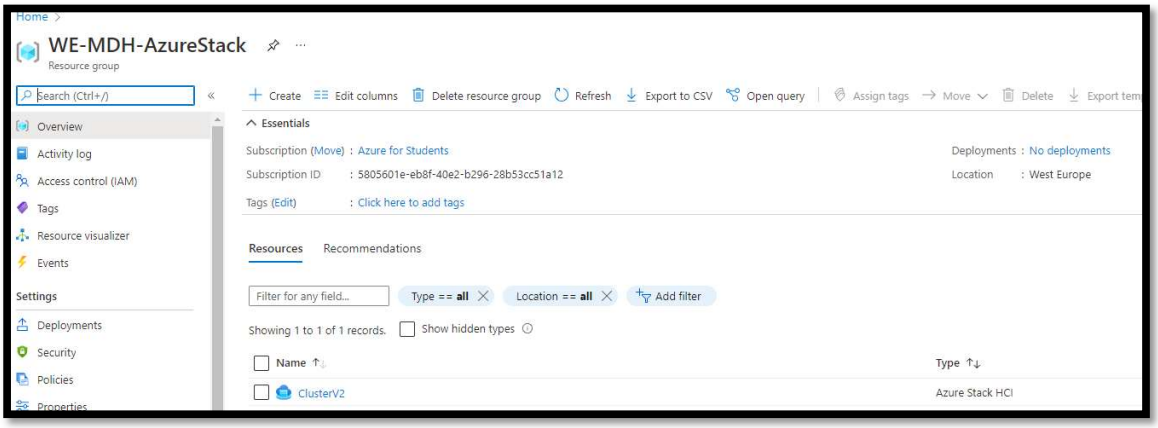
- ComputerName = the name of one the nodes, I choose for the first one.
- Credential = $azhciNodeCreds → this wel use the credentials of your Azure Stack HCI VM

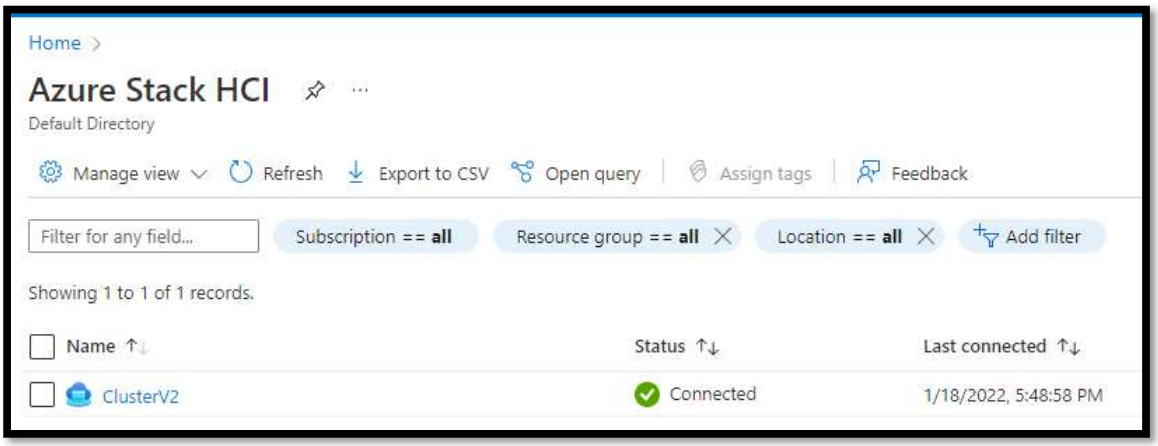
After running the registration script block you will get a prompt to sign in, sign in with a global administrator account for Azure. This must be a global administrator account because you need certain privileges to install features.

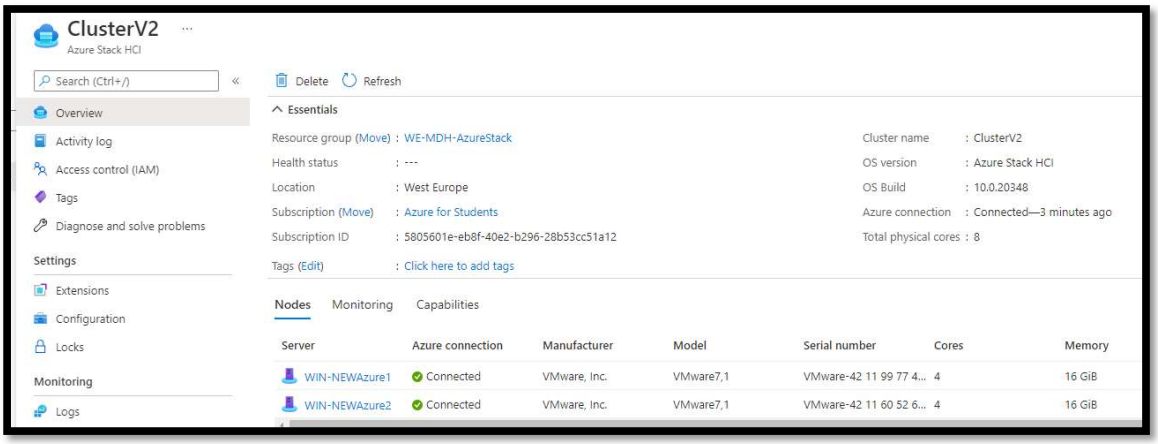



## 3.2.6 In the Azure Portal

When we go to our resource group you can see the cluster we just created.





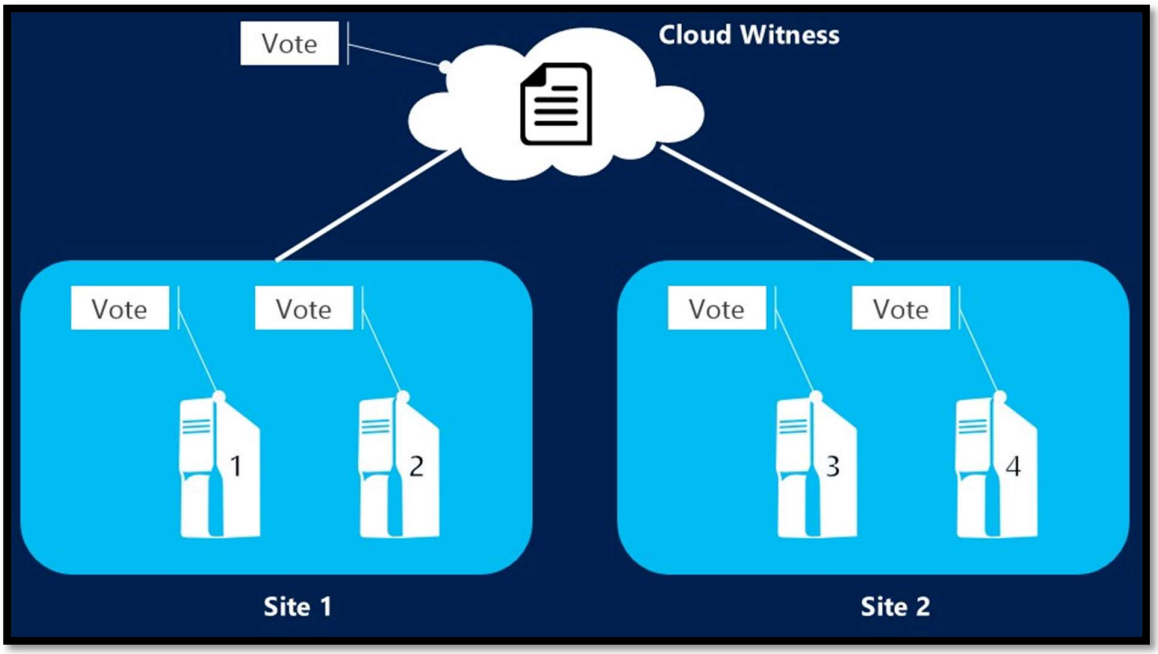

### 3.2.7 Cloud Witness

Azure Stack HCI is the only 2 node solution with server failure protection and drive failure in a 2 node configuration. This is possible to the Cloud Witness. When one of the nodes goes offline, the Cloud Witness makes sure that the other node does not become unavailable.

3 or 4 node clusters can also make use of Cloud Witness when 2 nodes become unavailable.

The quorum for a cluster is determined by the number of votes in elements that must be part of active cluster memebership for that cluster to start properly or continue running.

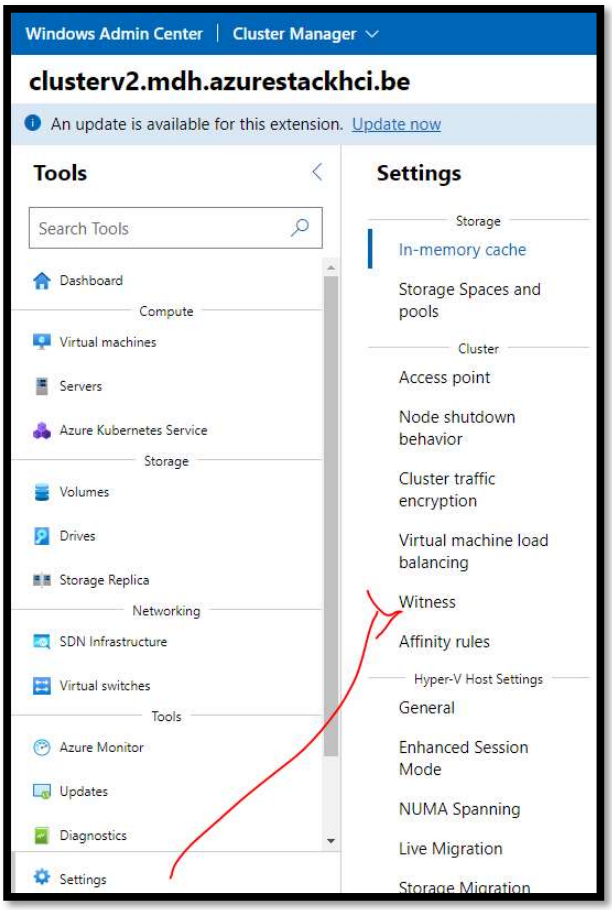
Each node typically has a vote, but in our case we have an even number of nodes we use this Cloud Witness to provide the third decisive vote.



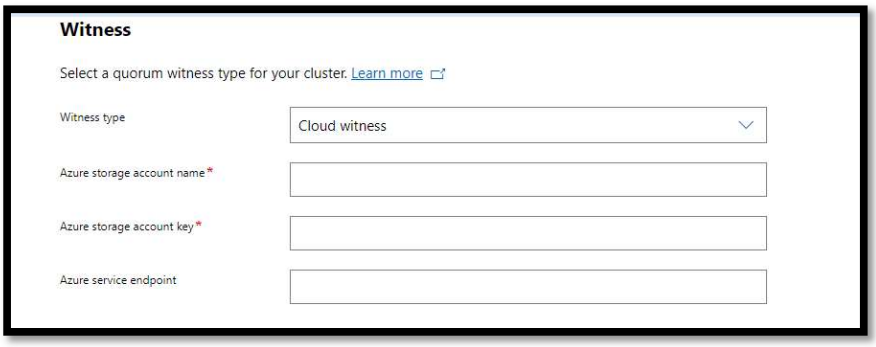

[2]

### 3.2.7.1 Setup Cloud Witness

Go to windows Admin Center.

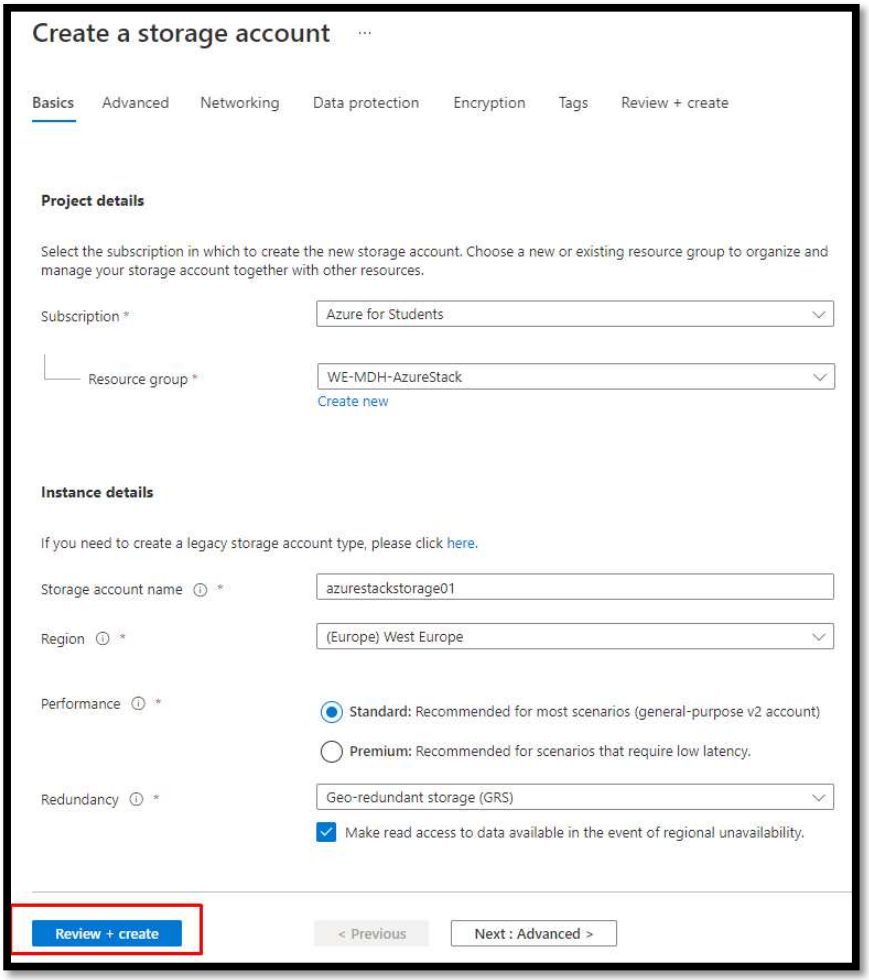
For this Witness we first have to create a storage account. After creating the storage account we will come back to this page.



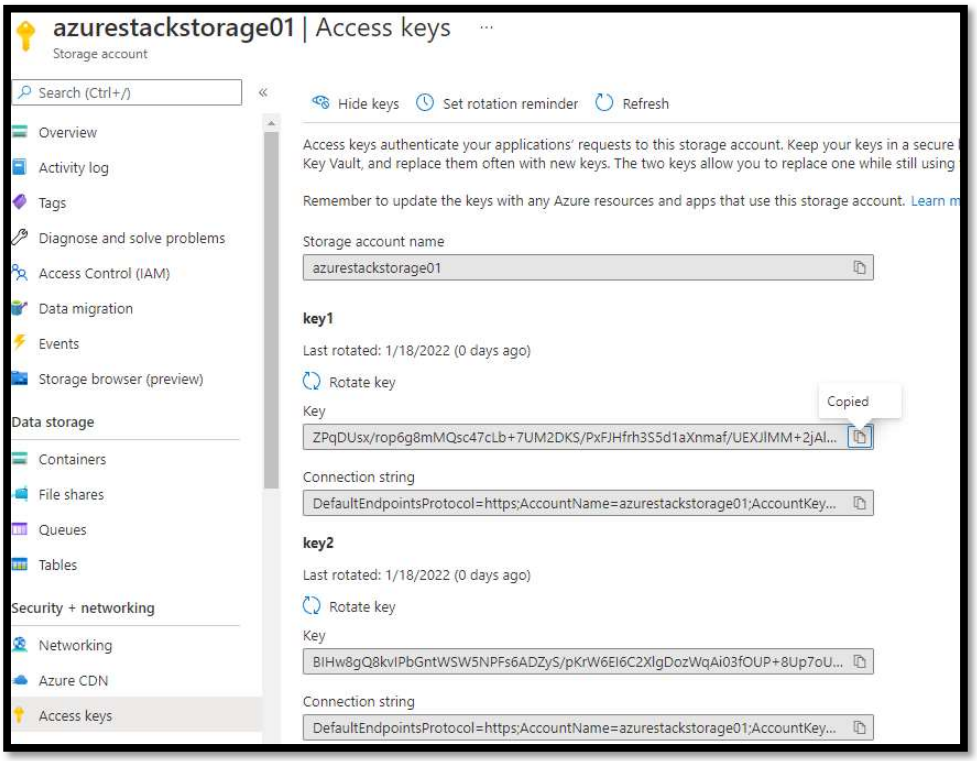

### 3.2.7.2 Storage Account

In the azure portal create a new storage account.

Make sure you put it in the resrouce group where our cluster is in and that the regions are consistend.



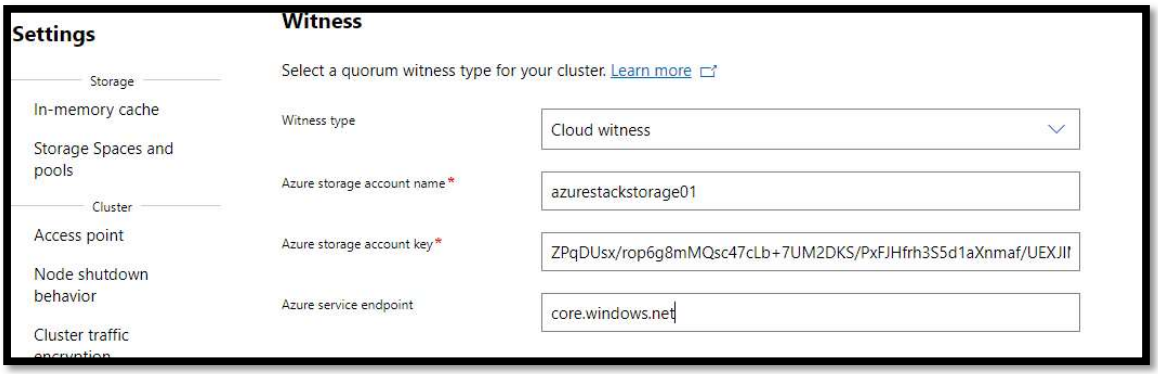
After we created the storage account go to Access keys.



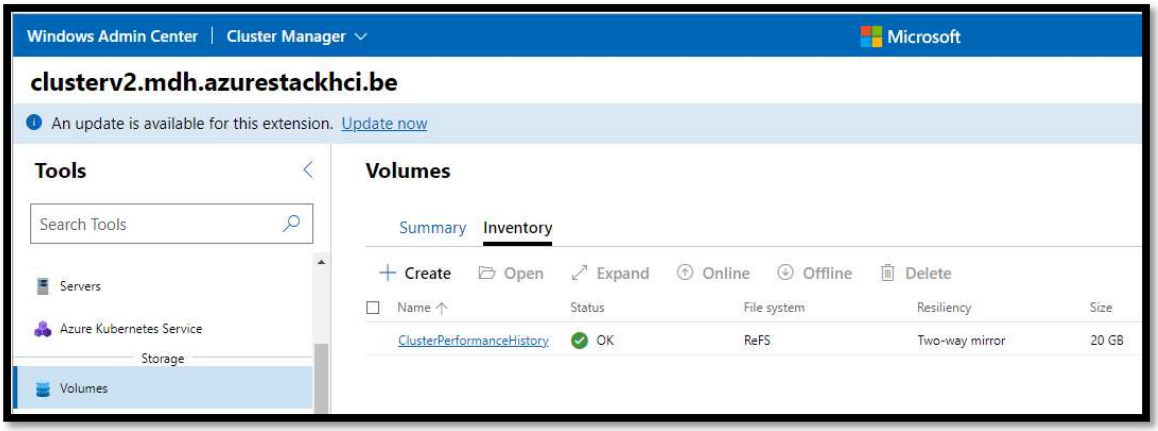

We need these credentials for our Cloud Witness.

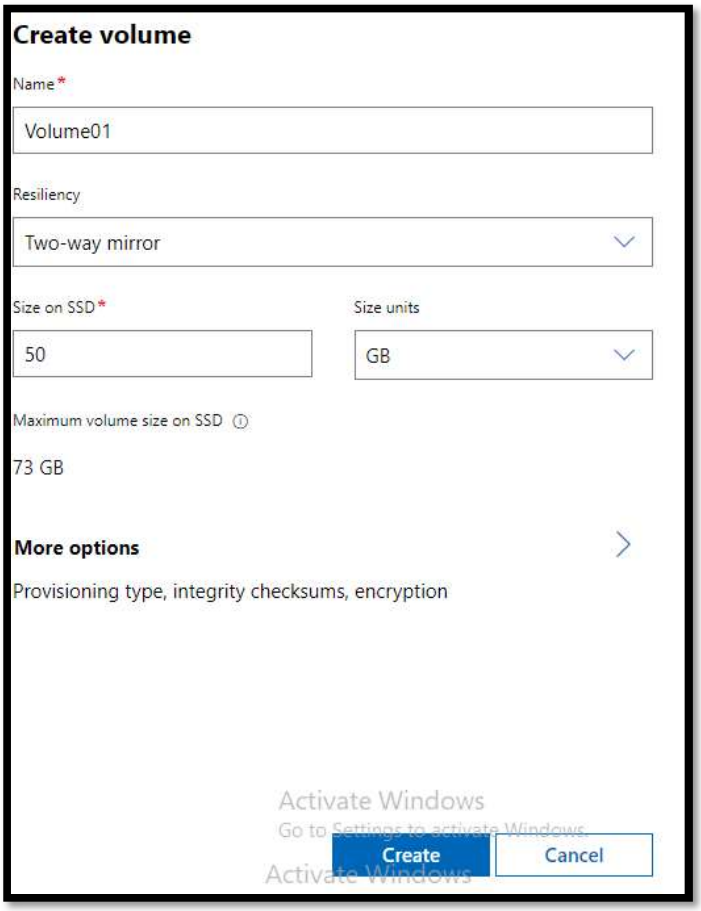
### 3.2.7.3 Back to Windows Admin Center – Cloud Witness

- Account name = the name of the storage account we just created
- Account key = the access key we copied
- Azure service endpoint = core.windows.net -> this is a build in endpoint that we can use



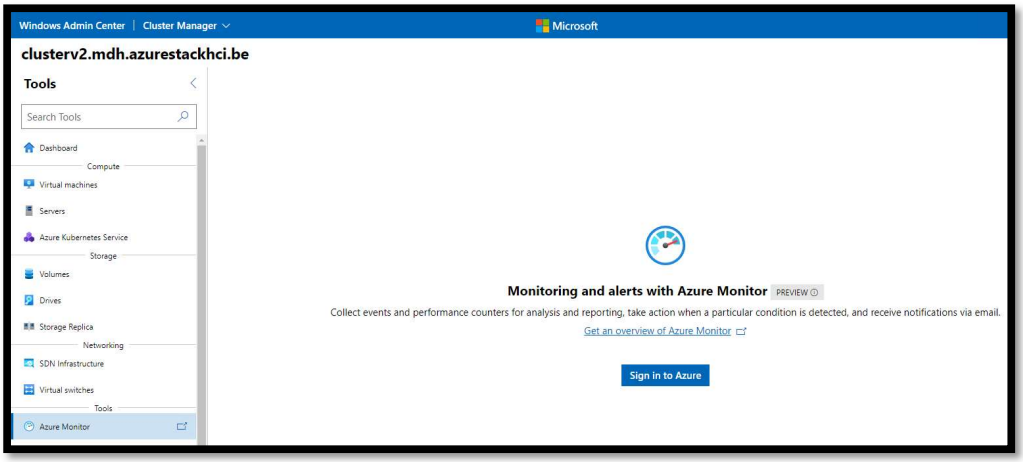
### 3.2.8 Define storage in Windows Admin Center

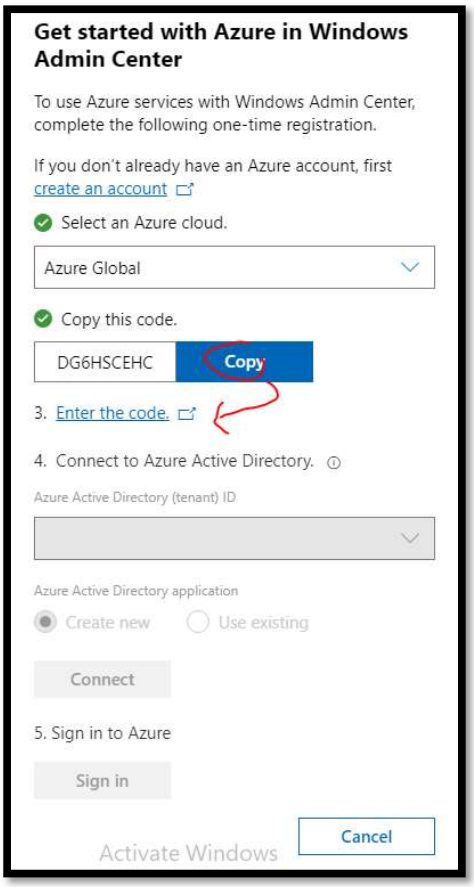


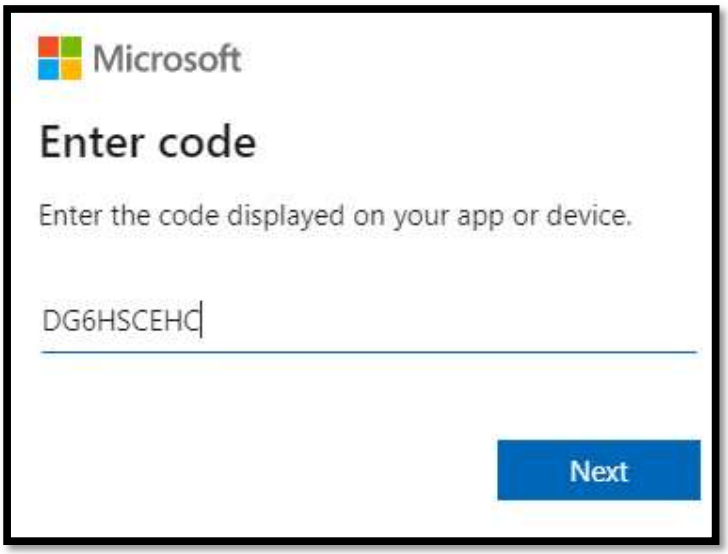



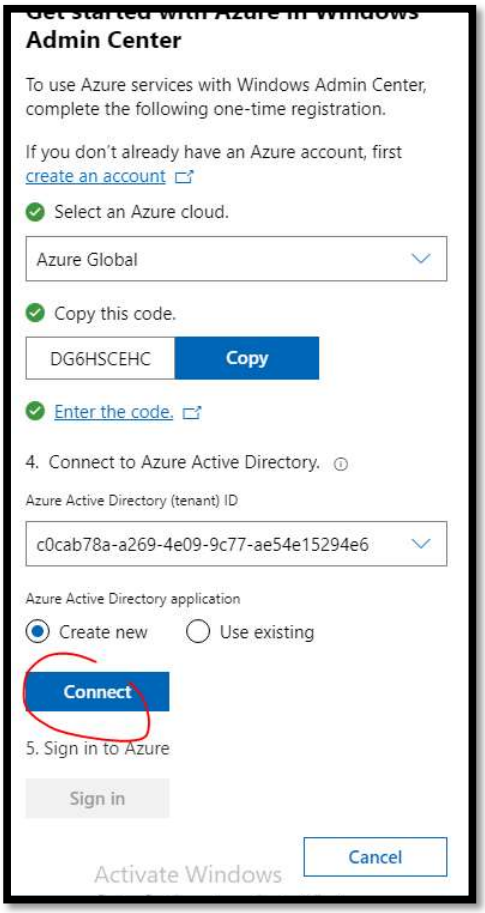
### 3.2.9 Enable logs and monitoring

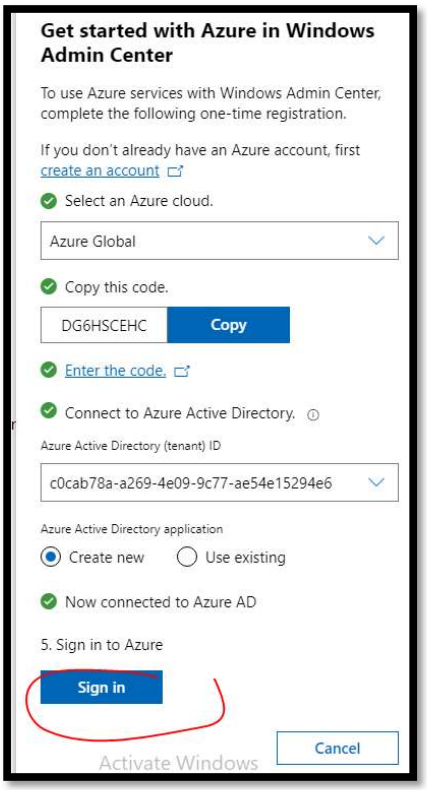
This way we can monitor our cluster in the Azure Portal. Sign in with your Azure global administrator account.



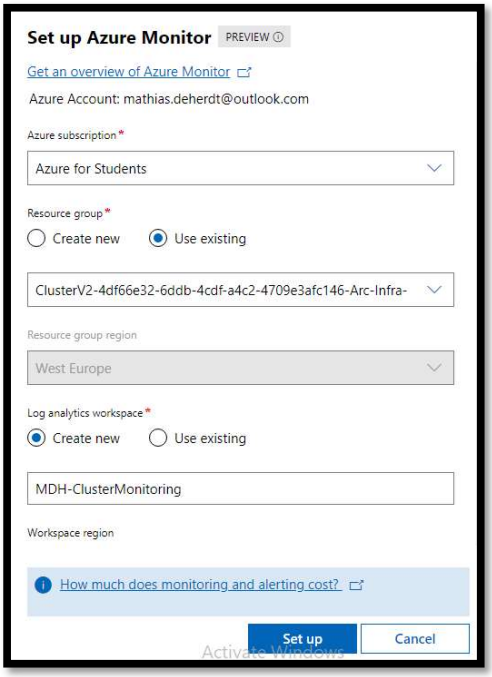



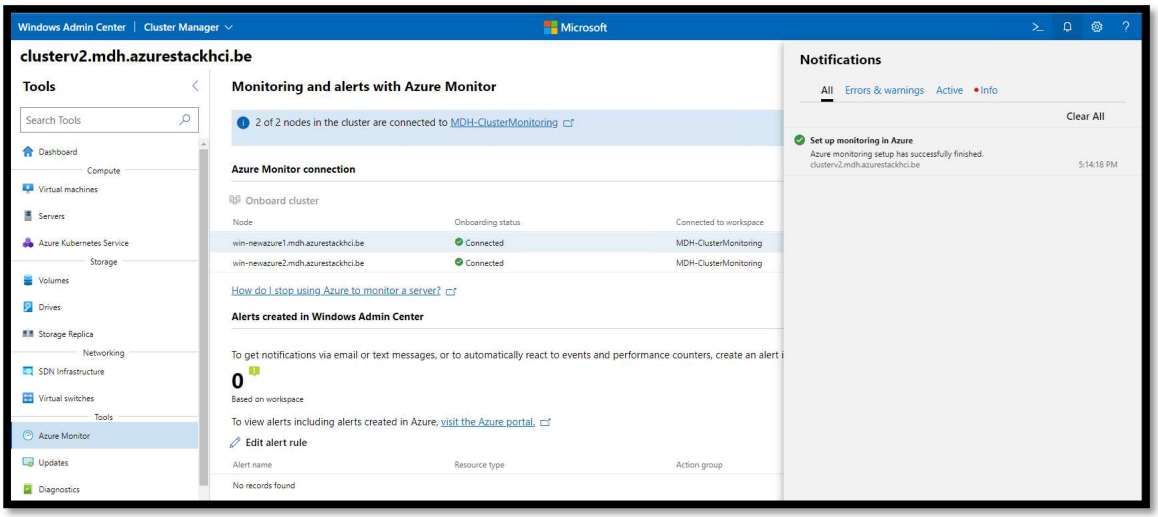

Microsoft
Enter code
Enter the code displayed on your app or device.
DG6HSCEHC
Next


Admin Center
To use Azure services with Windows Admin Center, complete the following one-time registration.
If you don't already have an Azure account, first create an account
Select an Azure cloud.
Azure Global
Copy this code.
DG6HSCEHC
Copy
Enter the code.
4. Connect to Azure Active Directory.
Azure Active Directory (tenant) ID
c0cab78a-a269-4e09-9c77-ae54e15294e6
Azure Active Directory application
Create new
Use existing
Connect
5. Sign in to Azure
Sign in
Cancel
Activate Windows



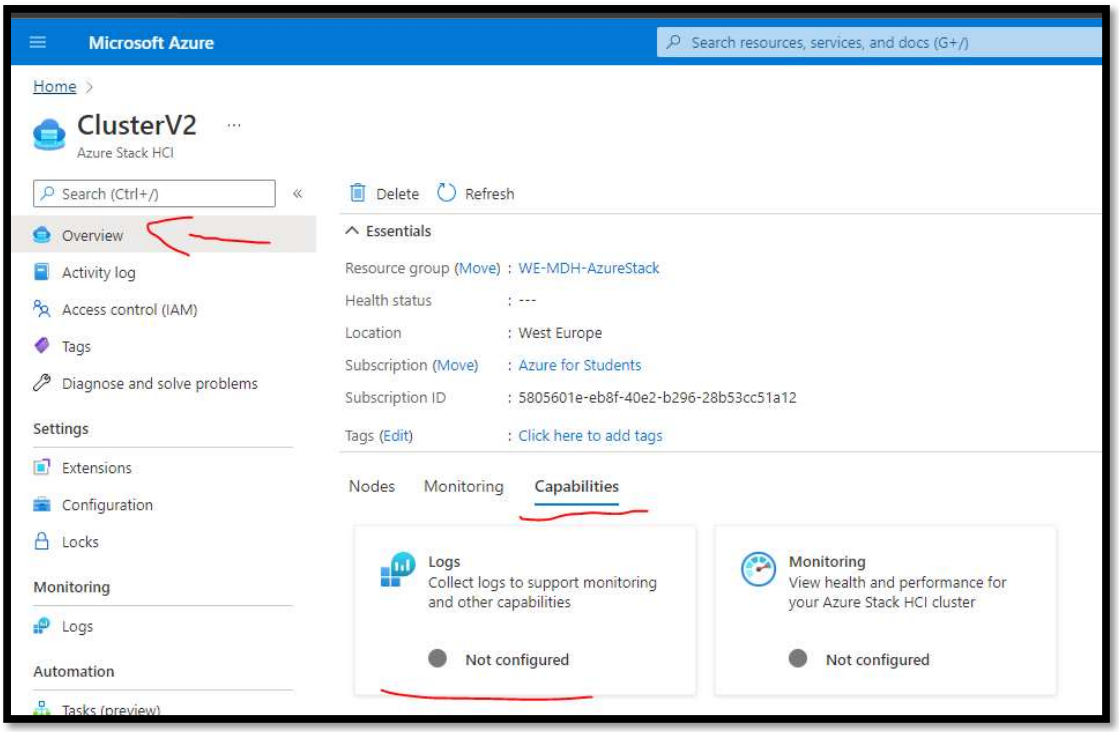

Create a new log analytics workspace



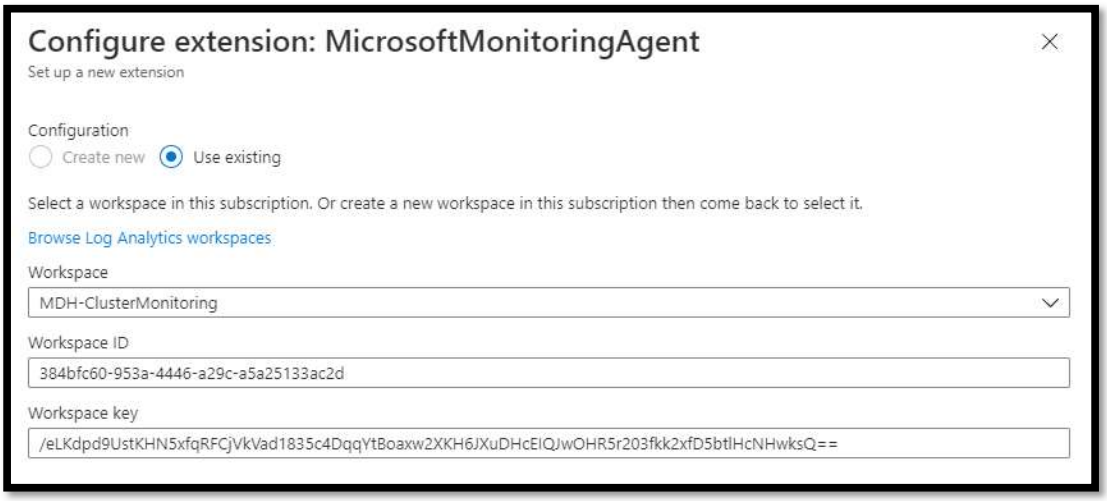



### 3.2.9.1 Enable logs

In the Azure Portal go to your clusters overview and select “Capabilities”.

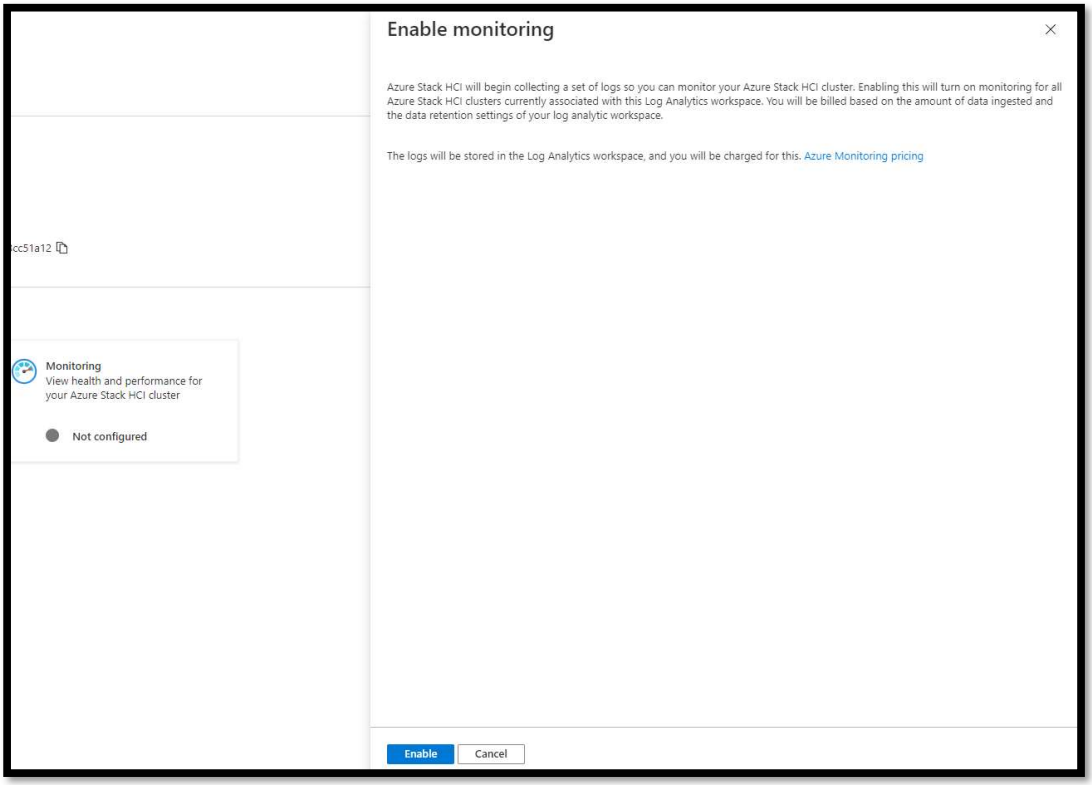


Here we can select our existing workspace we just created in Windows Admin Center

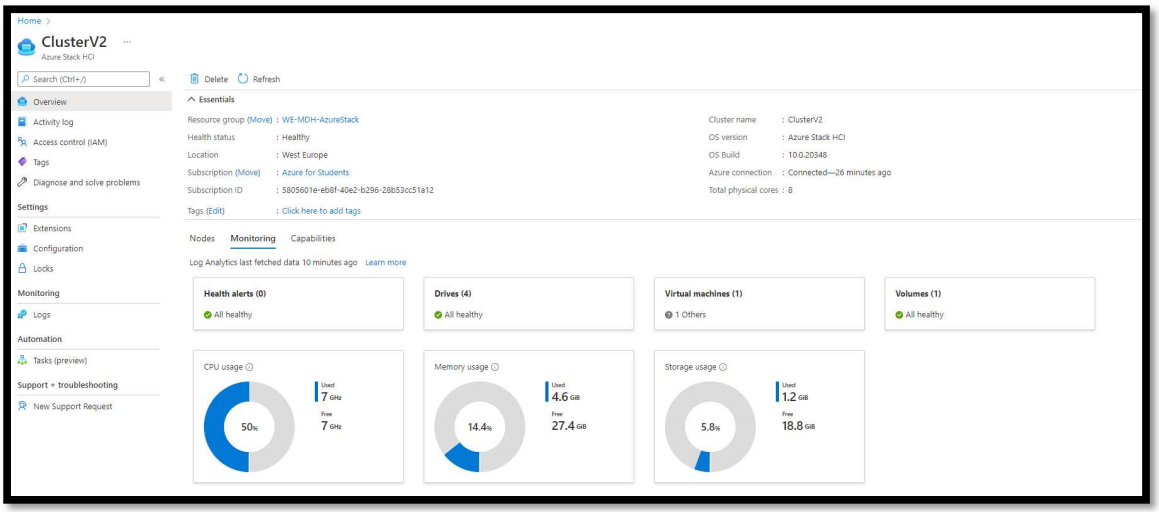



### 3.2.9.2 Enable Monitoring

In the overview of our cluster under “Capabilities” select “Monitoring” and click on enable.



Now in the overview under “Monitoring” after this has loaded (that can take up to 1 hour) you can monitor the cluster.

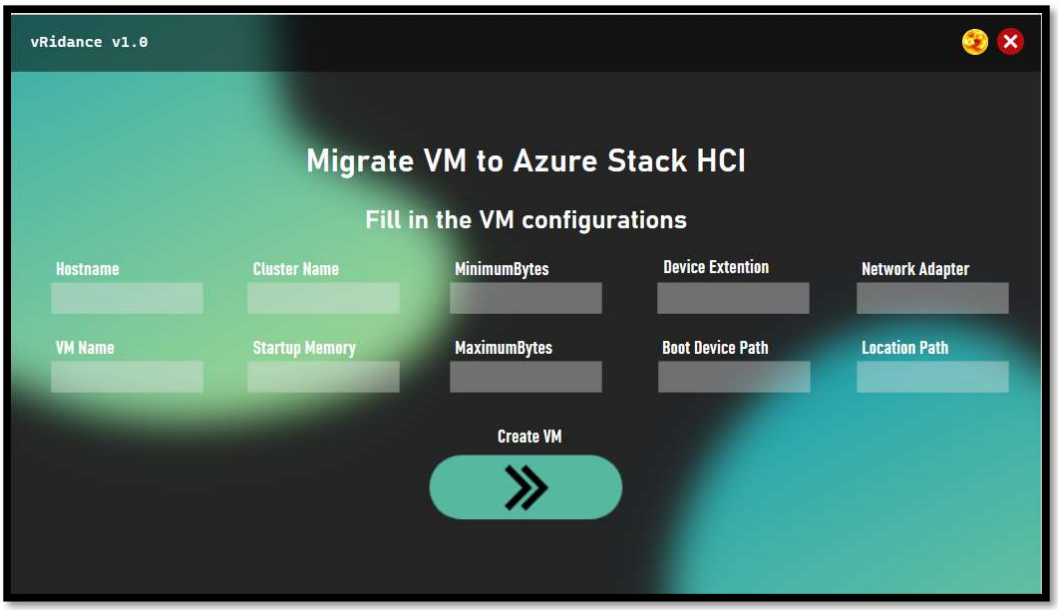



Now the environment is complete enough, at this point we can start to migrate VM’s.

## 3.3 The code behind everything

### 3.3.1 C#: AzureStackHCI.XML

This is the windows users can fill in, here we enter all the configurations for the VM we are going to Create.



- Hostname = The name of the Host where the VM is being placed
- VMName = Name of the VM
- ClusterName = Name of the cluster where we place the VM
- StartupMemory = How much memory we give at startup
- MinimumMemory = Minimum memory we give to the VM
- MaximumMemory = Maximum memory we give to the VM
- Device Extention = Extention of the harddisk file
- Boot Device Path = Location of the harddisk file
- Network adapter = Name of the network adapter
- Location Path = Location where we store the VM

Example:

- Hostname = Win-newazure1
- VMName = WindowsServer2022-001
- ClusterName = ClusterV2
- StartupMemory = 4GB
- MinimumMemory = 1GB
- MaximumMemory = 4GB
- Device Extention = VHD
- Boot Device Path = c:\diskpath\disk.vhd
- Network adapter = ComputeSwitch
- Location Path = c:\storagepath\

## 3.3.2 C#: AzureStackHCI.XML.CS

### 3.3.2.1 C#: Using

All the packages that the script will be using.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Xml;
using System.Collections.ObjectModel;
using System.Reflection;
```

### 3.3.2.2 C#: Namespace

This is everything in our namespace that will be executed.

```
19  namespace vRidance
20  {
21      /// <summary>
22      /// Interaction logic for AzureStackHCI.xaml
23      /// </summary>
        4 references
24      public partial class AzureStackHCI : Window
25      {
            1 reference
26          public AzureStackHCI(string theme)...
34
            1 reference
35          private void recCreateVM_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)...
68
            1 reference
69          private void rectClose_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)...
73
            1 reference
74          private void Grid_LayoutUpdated(object sender, EventArgs e)...
87      }
88  }
89
```

### 3.3.2.3 C#: public AzureStackHCI

Here we begin the script of the Azure Stack HCI migration page where we can fill in all the configurations. First we disable the button and change the opacity to make it appear lighter.

```
26      public AzureStackHCI(string theme)
27      {
28          InitializeComponent();
29          Debug.WriteLine("Begin");
30
31          recCreateVM.IsEnabled = false;
32          recCreateVM.Opacity = 0.5;
33      }
```

### 3.3.2.4 C#: Create VM button

Here we read everything insde the textboxes and send the values the to XML configuration file.

First we need to use XmlTextWrite function where we use a variable “xmlWriter2”. Now we can specify the location of the config.xml file and what type of encoding we are using.

With WriteStartElement we create the first element “Configuration” and again we use the WriteStartElement to create a new element under configuration called “CreateVM”.

Now we can use WriteElementString to assign the value to the element. We do this for each value we read out of the textboxes in the XAML file.

After we have send all the values we write an EndElement and WriteEndDocument to close everything.

When writing to the XML file is complete we stop this application.

```
1 reference
private void recCreateVM_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{

    Debug.WriteLine("Creating VM");

    Debug.WriteLine("Writing to xml");

    XmlTextWriter xmlWriter2 = new XmlTextWriter("config.xml", System.Text.Encoding.UTF8);

    xmlWriter2.Formatting = Formatting.Indented;
    xmlWriter2.WriteStartDocument();
    xmlWriter2.WriteComment("creating an XML file");

    xmlWriter2.WriteStartElement("Configuration");

    xmlWriter2.WriteStartElement("CreateVM");
    xmlWriter2.WriteElementString("HostName", txtHostName.Text.ToString());
    xmlWriter2.WriteElementString("VMName", txtVMName.Text.ToString());
    xmlWriter2.WriteElementString("ClusterName", txtClusterName.Text.ToString());
    xmlWriter2.WriteElementString("MemoryStartup", txtStartMemory.Text.ToString());
    xmlWriter2.WriteElementString("MinimumBytes", txtMinimumBytes.Text.ToString());
    xmlWriter2.WriteElementString("MaximumBytes", txtMaximumBytes.Text.ToString());
    xmlWriter2.WriteElementString("BootDeviceExstention", txtBootDeviceExtention.Text.ToString());
    xmlWriter2.WriteElementString("BootDevicePath", txtBootDevicePath.Text.ToString());
    xmlWriter2.WriteElementString("NetworkAdapter", txtNetworkAdapter.Text.ToString());
    xmlWriter2.WriteElementString("LocationPath", txtLocationPath.Text.ToString());

    xmlWriter2.WriteEndElement();
    xmlWriter2.WriteEndDocument();
    xmlWriter2.Flush();
    xmlWriter2.Close();

    System.Windows.Application.Current.Shutdown();
}
```

### 3.3.2.5 C#: Close GUI button

If needed with this function we can close the GUI.

```
1 reference
private void rectClose_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    System.Windows.Application.Current.Shutdown();
}
```

### 3.3.2.6 C#: Check if textbox is empty

Here we check for each textbox if it is empty or null. If one or more textbox is empty we keep the opacity at 0.5 and disable the button. When everything if filled in we change the opacity to 1 and make the button active.

```
1 reference
private void Grid_LayoutUpdated(object sender, EventArgs e)
{
    if(txtBootDevicePath.Text != "" && txtBootDevicePath != null && txtLocationPath.Text != "" && txtLocationPath != null && txtMinimumBytes.Text != "" &&
    {
        recCreateVM.IsEnabled = true;
        recCreateVM.Opacity = 1;
    }
    else
    {
        recCreateVM.IsEnabled = false;
        recCreateVM.Opacity = 0.5;
    }
}
```

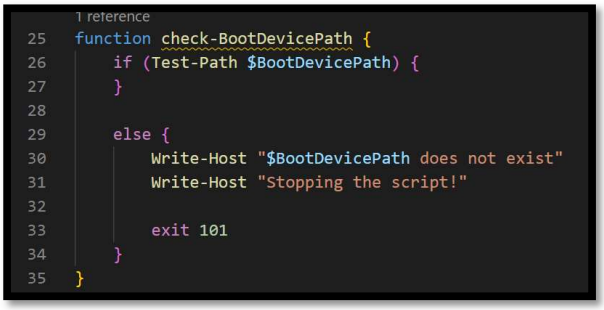
### 3.3.3 XML: config.xml

This is being created by our C# script. This is going to be used by the powershell script.

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
    <CreateVM>
        <HostName>Fill in a Name of the Host where we put the VM</HostName>
        <VMName>Name of your VM</VMName>
        <ClusterName>Name of the cluster</ClusterName>
        <MemoryStartup>INT + GB  → example 4GB</MemoryStartup>
        <MinimumBytes>INT + GB   → example 1GB</MinimumBytes>
        <MaximumBytes>INT + GB   → example 4GB</MaximumBytes>
        <BootDeviceExstention>VHD|VHDX</BootDeviceExstention>
        <BootDevicePath>Path to VHD file → example c:\temp\disk.vhd</BootDevicePath>
        <NetworkAdapter>Name of the network adapter</NetworkAdapter>
        <LocationPath>Path where we store the VM</LocationPath>
    </CreateVM>
</Configuration>
```

### 3.3.4 Powershell: check-PathXML

This function will check if the xml file is reachable. If not the script is stopped and exited with code 100. In the main-Script function we give a value to XML_Path




### 3.3.5 Powershell: check-BootDevicePath

Here we will check if our path to the VHD file exists and if it is reachable. If not we return errorcode 101

### 3.3.6 Powershell: check-BootDeviceExtention

In this fuction we are checking if the value from the xml file are equal to VHD or VHDX. These are the only file extentions that are supported by this script and by Azure Stack HCI for what we a trying to accomplish.

```
1 reference
function check-BootDeviceExtention {
    if ($BootDeviceExstention -eq "VHD") {
    }

    elseif ($BootDeviceExstention -eq "VHDX") {
    }

    else {
        Write-Host "$BootDeviceExstention is incorrect"
        Write-Host "    Should be VHD or VHDX"
        Write-Host "Stopping the script!"

        exit 103
    }
}
```

### 3.3.7 Powershell: read-DataFromXML

In this function we are reading in our xml file and putting all the values in variables for later use and after everything is put into variables and the necessary check functions done and approved we go to the create-VirtualMachine function. But first let's take a closer look to the read-DataFromXML function.

```
function read-DataFromXML {
    param([string]$FileLocation)

    [Xml.XmlDocument]$script:XML_ConfigFile     = Get-Content -LiteralPath $FileLocation

    $script:XML_Config                          = $XML_ConfigFile.Configuration

    # ============================================================================
    # Credential not used anymore
    # ============================================================================
    [Xml.XmlElement]$CredentialElement          = $XML_Config.Credentials

    $Script:RemoteMachineName                   = $CredentialElement.MachineName
    $Script:DomainAdmin                         = $CredentialElement.DomainAdmin

    # ============================================================================
    # Create Virtual Machine
    # ============================================================================
    [Xml.XmlElement]$CreateVMElement            = $XML_Config.CreateVM

    $Script:Hostname                            = $CreateVMElement.HostName
    $Script:VMName                              = $CreateVMElement.VMName
    $Script:ClusterName                         = $CreateVMElement.ClusterName
    $Script:MemoryStartup                       = $CreateVMElement.MemoryStartup
    $Script:MinimumBytes                        = $CreateVMElement.MinimumBytes
    $Script:MaximumBytes                        = $CreateVMElement.MaximumBytes
    $Script:BootDeviceExstention                = $CreateVMElement.BootDeviceExstention
    $Script:BootDevicePath                      = $CreateVMElement.BootDevicePath
    $Script:SwitchName                          = $CreateVMElement.NetworkAdapter
    $Script:LocationPath                        = $CreateVMElement.LocationPath

    check-BootDeviceExtention $BootDeviceExstention
    check-BootDevicePath $BootDevicePath

    create-VirtualMachine
}
```

Here we will be reading the whole file from the location and telling our script that it is an xml document so we can read it. Here we read everything in from our configuration element and assign it to the variable XML_Config.

```
60  function read-DataFromXML {
61      param([string]$FileLocation)
62
63      [Xml.XmlDocument]$script:XML_ConfigFile     = Get-Content -LiteralPath $FileLocation
64
65      $script:XML_Config                          = $XML_ConfigFile.Configuration
66
```

In this part of the code we read in our credential element. We use our previouse created XML_Config variable to tell the script we are using the credential element inside the config element.

```
67      # ================================================================================
68      # Credential
69      # ================================================================================
70      [Xml.XmlElement]$CredentialElement          = $XML_Config.Credentials
71
72      $Script:RemoteMachineName                   = $CredentialElement.MachineName
73      $Script:DomainAdmin                         = $CredentialElement.DomainAdmin
```

Now we take the XML_Config element and read the CreateVMElement from inside it and assign it as a variable so we can read al the sub elements from CreateVMElement and put it all in variables.

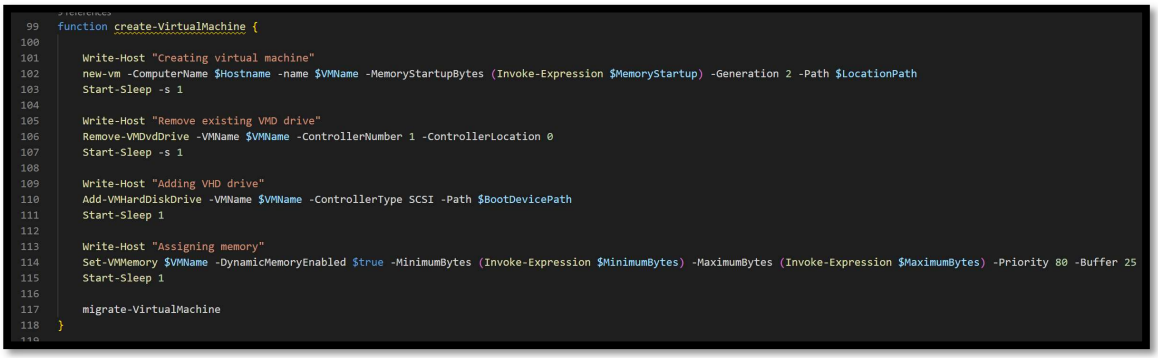
```
78      [Xml.XmlElement]$CreateVMElement            = $XML_Config.CreateVM
79
80      $Script:Hostname                            = $CreateVMElement.HostName
81      $Script:VMName                              = $CreateVMElement.VMName
82      $Script:ClusterName                         = $CreateVMElement.ClusterName
83      $Script:MemoryStartup                       = $CreateVMElement.MemoryStartup
84      $Script:MinimumBytes                        = $CreateVMElement.MinimumBytes
85      $Script:MaximumBytes                        = $CreateVMElement.MaximumBytes
86      $Script:BootDeviceExstention                = $CreateVMElement.BootDeviceExstention
87      $Script:BootDevicePath                      = $CreateVMElement.BootDevicePath
88      $Script:SwitchName                          = $CreateVMElement.NetworkAdapter
89      $Script:LocationPath                        = $CreateVMElement.LocationPath
```

In the final part of our function we do a couple of checks and after they are successfully completed we go to the create-VirtualMachine function where we create a VM using all the variables from the XML file.

```
90      check-BootDeviceExtention $BootDeviceExstention
91      check-BootDevicePath $BootDevicePath
92
93      create-VirtualMachine
```

### 3.3.8 Powershell: create-VirtualMachine

In this fuction the VM is being created, let's take a closer look at each part of the function.



```
function create-VirtualMachine {

    Write-Host "Creating virtual machine"
    new-vm -ComputerName $Hostname -name $VMName -MemoryStartupBytes (Invoke-Expression $MemoryStartup) -Generation 2 -Path $LocationPath
    Start-Sleep -s 1

    Write-Host "Remove existing VMD drive"
    Remove-VMDvdDrive -VMName $VMName -ControllerNumber 1 -ControllerLocation 0
    Start-Sleep -s 1

    Write-Host "Adding VHD drive"
    Add-VMHardDiskDrive -VMName $VMName -ControllerType SCSI -Path $BootDevicePath
    Start-Sleep 1

    Write-Host "Assigning memory"
    Set-VMMemory $VMName -DynamicMemoryEnabled $true -MinimumBytes (Invoke-Expression $MinimumBytes) -MaximumBytes (Invoke-Expression $MaximumBytes) -Priority 80 -Buffer 25
    Start-Sleep 1

    migrate-VirtualMachine
}
```

#### 3.3.8.1 New-VM

The new-vm command is part of the Hyper-V module we installed in Windows Admin center. It is used to create a new virtual machine.

- ComputerName = Node where you put the VM
- VMName = Name of the VM
- MemortyStartupBytes = Memory that is given to the VM at startyp
- Generation = Genaration of the VM, 1 = VHD | 2 = VHDX
    - (this did not work for me, I have to use 2 when using a VHD)
- Path = Location for where to store the VM
- SwitchName = Name of the switch we made in the cluster

Example:

- ComputerName = win-newazure1
- VMName = WindowsServer2022-01
- MemortyStartupBytes = 4GB
- Generation = 2
- Path = \\WIN-NEWAZURE1\c$\ClusterStorage\Volume01\VMs
- SwitchName = ComputeSwitch

```
Write-Host "Creating virtual machine"
new-vm -ComputerName $Hostname -name $VMName -MemoryStartupBytes (Invoke-Expression $MemoryStartup) -Generation 2 -Path $LocationPath -SwitchName $SwitchName
Start-Sleep -s 1
```

#### 3.3.8.2 Remove-VMDvdDrive

After creating our new VM it automatically has a harddrive attached to it, we need to remove this one and later we attached our VHD file from the VM we want to migrate.

- VMName = Name of the VM we want to remove the harddrive from
- ControllerNumber = the controller number from which the DVD drive is to be deleted
- ControllerLocation = the number of the location from which the DVD drive is to be deleted

Example:

- VMName = WindowsServer2022-01
- ControllerNumber = 1
- ControllerLocation = 0

```
105     Write-Host "Remove existing VMD drive"
106     Remove-VMDvdDrive -VMName $VMName -ControllerNumber 1 -ControllerLocation 0
107     Start-Sleep -s 1
```

#### 3.3.8.3 Add-VMHardDiskDrive

We connect the VHD drive we converted from the VM in vSphere.

```
109     Write-Host "Adding VHD drive"
110     Add-VMHardDiskDrive -VMName $VMName -ControllerType SCSI -Path $BootDevicePath
111     Start-Sleep 1
```

#### 3.3.8.4 Set-VMMemory

Here we assign the memory for the VM.

```
113     Write-Host "Assigning memory"
114     Set-VMMemory $VMName -DynamicMemoryEnabled $true -MinimumBytes (Invoke-Expression $MinimumBytes) -MaximumBytes (Invoke-Expression $MaximumBytes) -Priority 80 -Buffer 25
115     Start-Sleep 1
```

### 3.3.9 Powershell: migrate-VirtualMachineToCluster

Here we specify the VM that we want to place onto the cluster.

```
123 function migrate-VirtualMachineToCluster {
124     Write-Host "Adding VM to cluster"
125     Add-ClusterVirtualMachineRole -Name $VMName -VirtualMachine $VMName -Cluster $ClusterName
126 }
127
```

### 3.3.10 Powershell: main-Script

This function will start our script and get the location from where this script (and the xml file) are located. Before we go to the read-DataFromXML function, we first need to check if the path to our xml file is reachable, if not the script will stop here.

```
131  function main-Script {
132      $Path_Location = Get-Location
133      $XML_Path = "$Path_Location\Config.xml"
134
135      check-PathXML
136
137      read-DataFromXML $XML_Path
138  }
139
140  main-Script
```

### 3.3.11 Powershell exit codes

- 100 = path to the xml file is incorrect
- 101 = path to the VHD (or VHDX) file is incorrect
- 102 = Wrong value in BootDeviceExtention

## 3.4 Migrate VM

Open you mgmt machine where you can access Windows Admin Center and your Azre nodes.

In the sources folder we provided you can see our source code for vRidance.exe (with all the other files), !Converter.cmd, a powershell script and a xml file.

You will also need to download the StarWind V2V Converter tool. This tool let's us convert a vmdk to a vhd. You will need to fill in your email address and information and you will get a mail with the download link.

StarWind V2V Converter - Converting VM Formats (starwindsoftware.com)

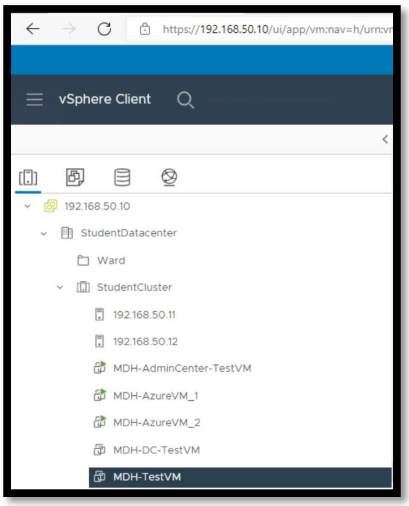
Make sure you have all these scripts at the ready on the mgmt machine, we will use them later on.

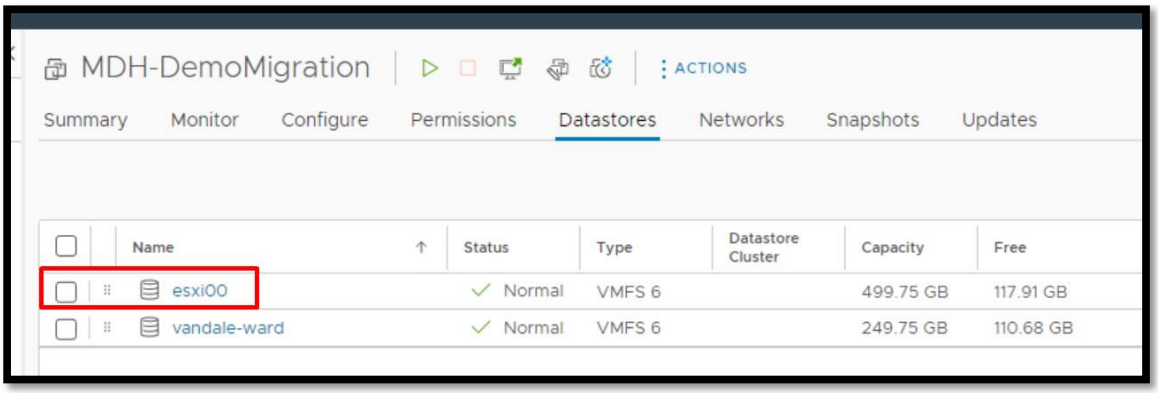
### 3.4.1 Step 1: Select a VM

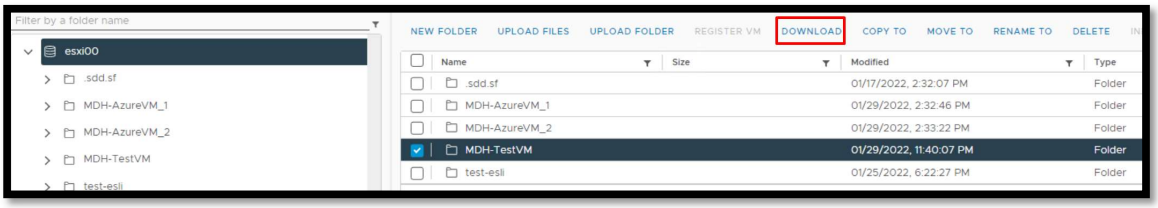
On your mgmt machine log into vSphere and look for a VM you want to migrate. Go to its datastore files and download the whole VM folder.

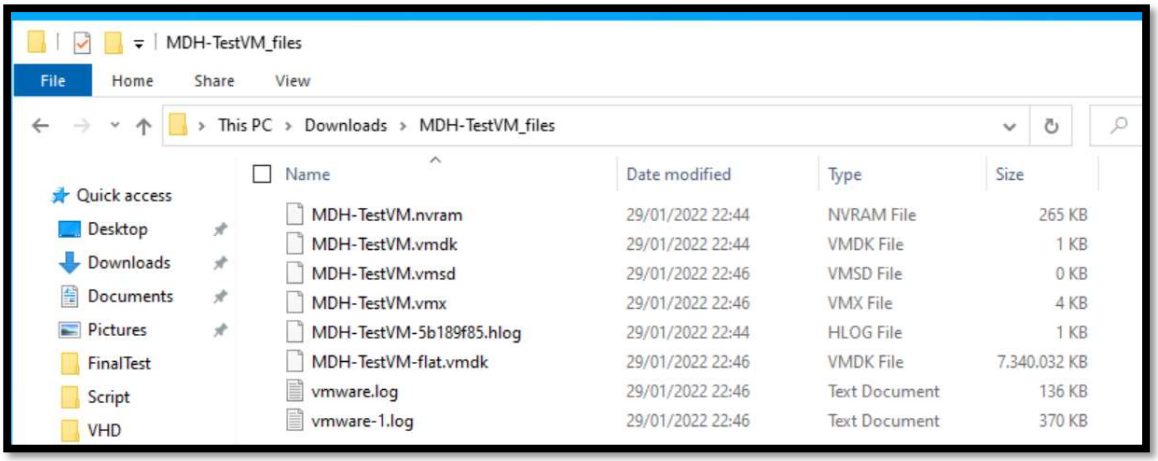
Once downloaded this will be in the downloads folder of the mgmt machine.

Why do we need to do it this way? Well the general idea is that we need the vmdk file of the VM we want to migrate and convert it to a vhd. After converting the harddisk we create a new VM in Azure Stack HCI and attach the vhd file from the vSphere VM to the Azure Stack HCI VM.x


https://192.168.50.10/ui/app/vm;nav=h/urn:v
vSphere Client
192.168.50.10
StudentDatacenter
Ward
StudentCluster
192.168.50.11
192.168.50.12
MDH-AdminCenter-TestVM
MDH-AzureVM_1
MDH-AzureVM_2
MDH-DC-TestVM
MDH-TestVM


MDH-DemoMigration
ACTIONS
Summary
Monitor
Configure
Permissions
Datastores
Networks
Snapshots
Updates
Name
Status
Type
Datastore Cluster
Capacity
Free
esxi00
Normal
VMFS 6
499.75 GB
117.91 GB
vandale-ward
Normal
VMFS 6
249.75 GB
110.68 GB


Filter by a folder name
esxi00
.sdd.sf
MDH-AzureVM_1
MDH-AzureVM_2
MDH-TestVM
test-esli
NEW FOLDER
UPLOAD FILES
UPLOAD FOLDER
REGISTER VM
DOWNLOAD
COPY TO
MOVE TO
RENAME TO
DELETE
Name
Size
Modified
Type
.sdd.sf
01/17/2022, 2:32:07 PM
Folder
MDH-AzureVM_1
01/29/2022, 2:32:46 PM
Folder
MDH-AzureVM_2
01/29/2022, 2:33:22 PM
Folder
MDH-TestVM
01/29/2022, 11:40:07 PM
Folder
test-esli
01/25/2022, 6:22:27 PM
Folder


MDH-TestVM_files
File
Home
Share
View
This PC
Downloads
MDH-TestVM_files
Quick access
Desktop
Downloads
Documents
Pictures
FinalTest
Script
VHD
Name
Date modified
Type
Size
MDH-TestVM.nvram
29/01/2022 22:44
NVRAM File
265 KB
MDH-TestVM.vmdk
29/01/2022 22:44
VMDK File
1 KB
MDH-TestVM.vmsd
29/01/2022 22:46
VMSD File
0 KB
MDH-TestVM.vmx
29/01/2022 22:46
VMX File
4 KB
MDH-TestVM-5b189f85.hlog
29/01/2022 22:44
HLOG File
1 KB
MDH-TestVM-flat.vmdk
29/01/2022 22:46
VMDK File
7.340.032 KB
vmware.log
29/01/2022 22:46
Text Document
136 KB
vmware-1.log
29/01/2022 22:46
Text Document
370 KB

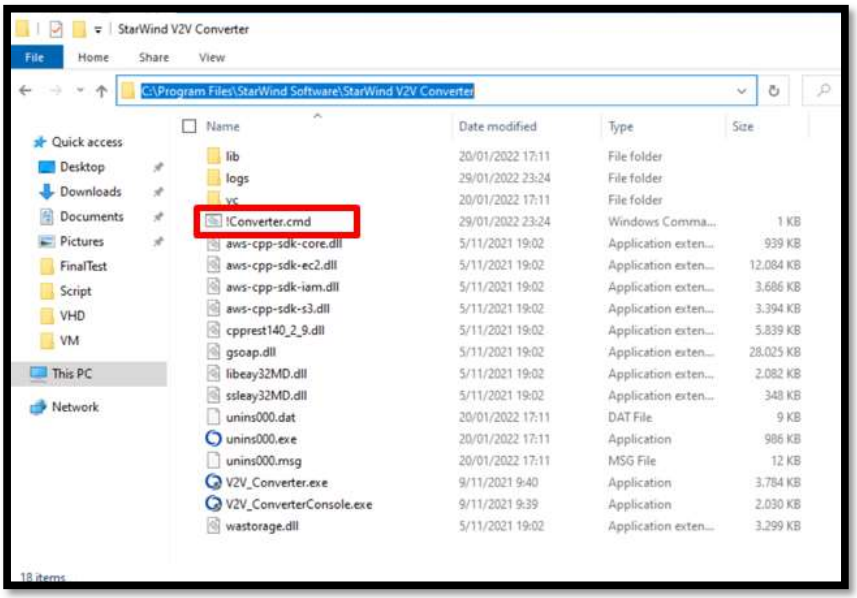
### 3.4.2 Step 2: Convert vmdk to vhd

First install StarWind V2V converter from the link we provided.

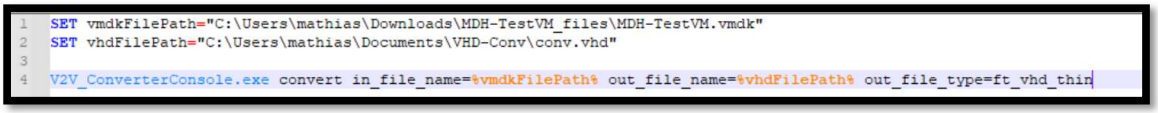
After installing open file explorer to this path:

C:\Program Files\StarWind Software\StarWind V2V Converter\

Here past “!Converter.cmd” into this folder.




And edit “!Converter.cmd” with notepad. You will see this code:



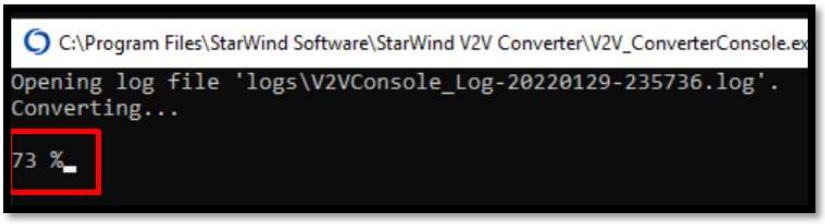

```
SET vmdkFilePath=""
SET vhdFilePath=""

V2V_ConverterConsole.exe      convert      in_file_name=%vmdkFilePath%      out_file_name=%vhdFilePath%
out_file_type=ft_vhd_thin
```

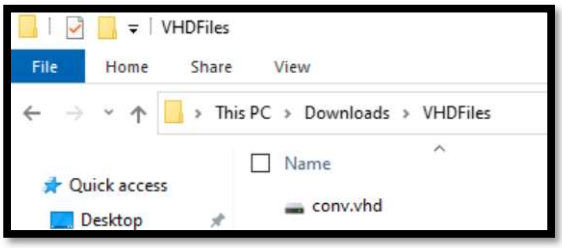
- vmdkFilePath = path to the vmdk of the VM from vSphere
- vmdFilePath = Path where we want to store the converted file

example:

- vmdkFilePath = “C:\Users\mathias\Downloads\test-VM.vmdk”
- vmdFilePath = “C:\Users\mathias\Downloads\VHDFiles\conv.vhd”

after editing the file save it and double click on it to start the converting. If everything is correct you should start seeing this window.




Now cut and paste the vhd file in to the cluster storage. We must copy past this because if we convert and immediately place it onto the cluster storage we must wait upto 3 hours to complete. And there is a lot that can go wrong, if we even lose connection the vhd file can go corrupt so the safest solution is to just copy and past them into the cluster storage.

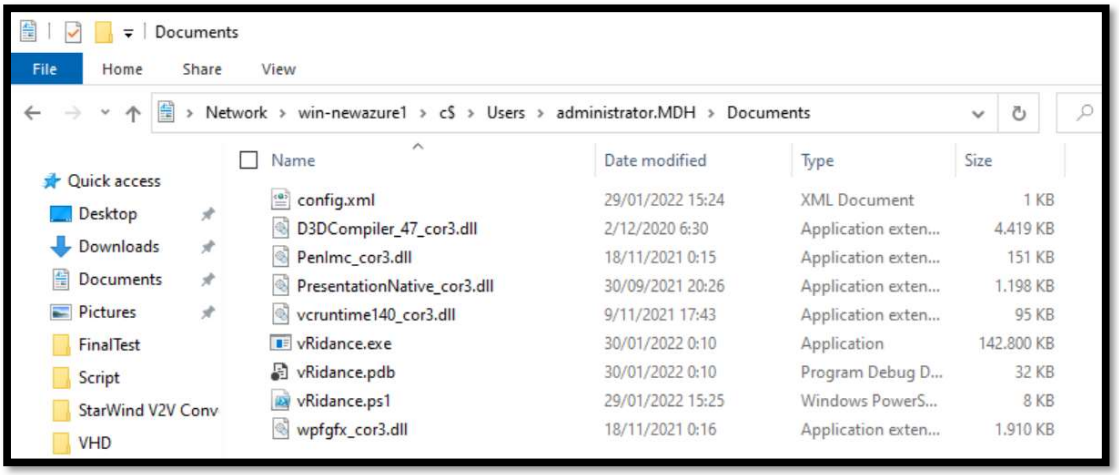
\\win-newazure1\c$\ClusterStorage\Volume01\VHD\

### 3.4.3 Step 3: Start the migration GUI

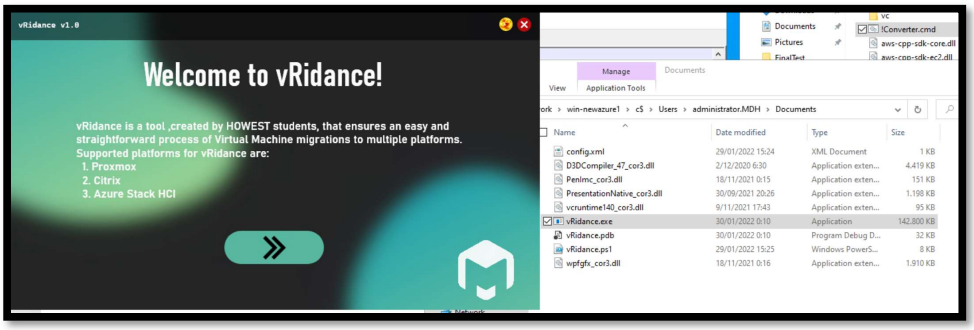
Open your mgmt machine where we can both access Windows Admin Center and the Azure nodes.

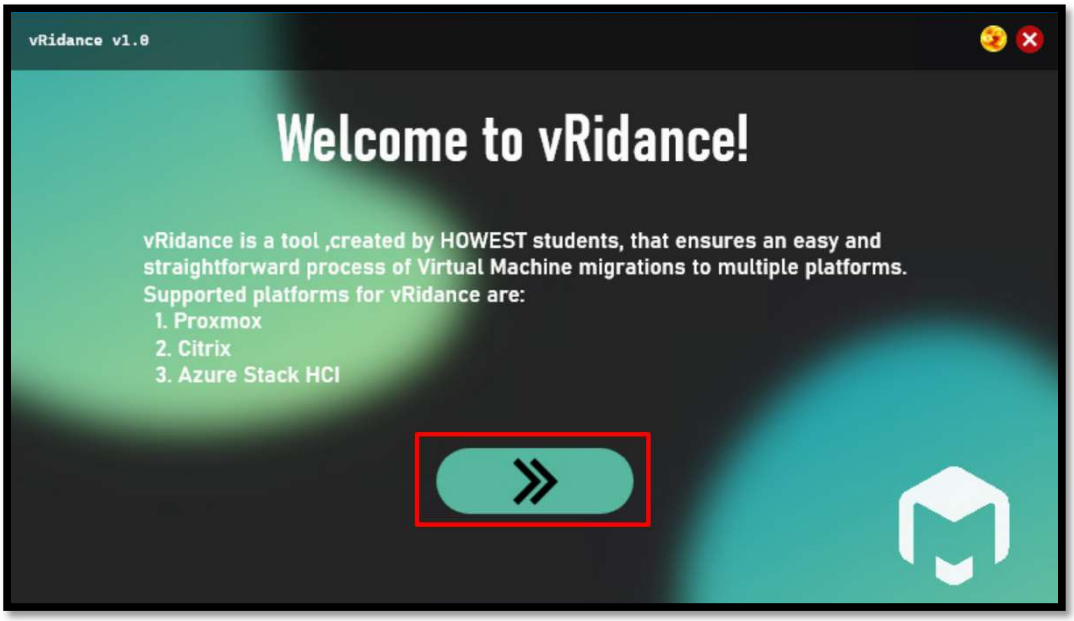
In the sources folder we provided all the necessary files. Simply place them on one of your nodes. I put them on win-newazure1 in this path:

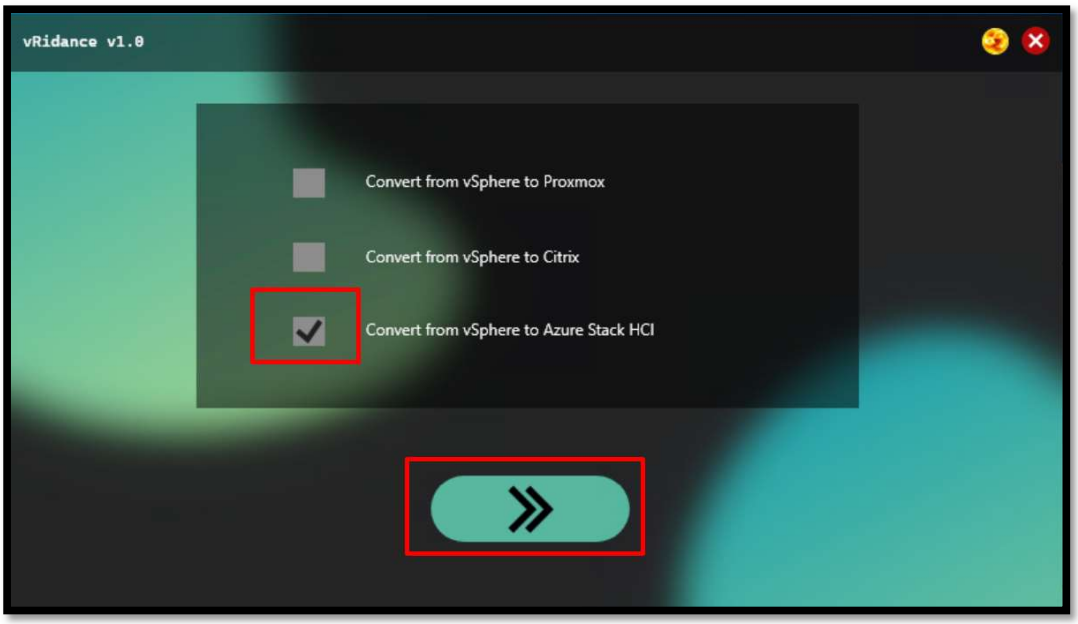
\\win-newazure1\c$\Users\administrator.MDH\Documents



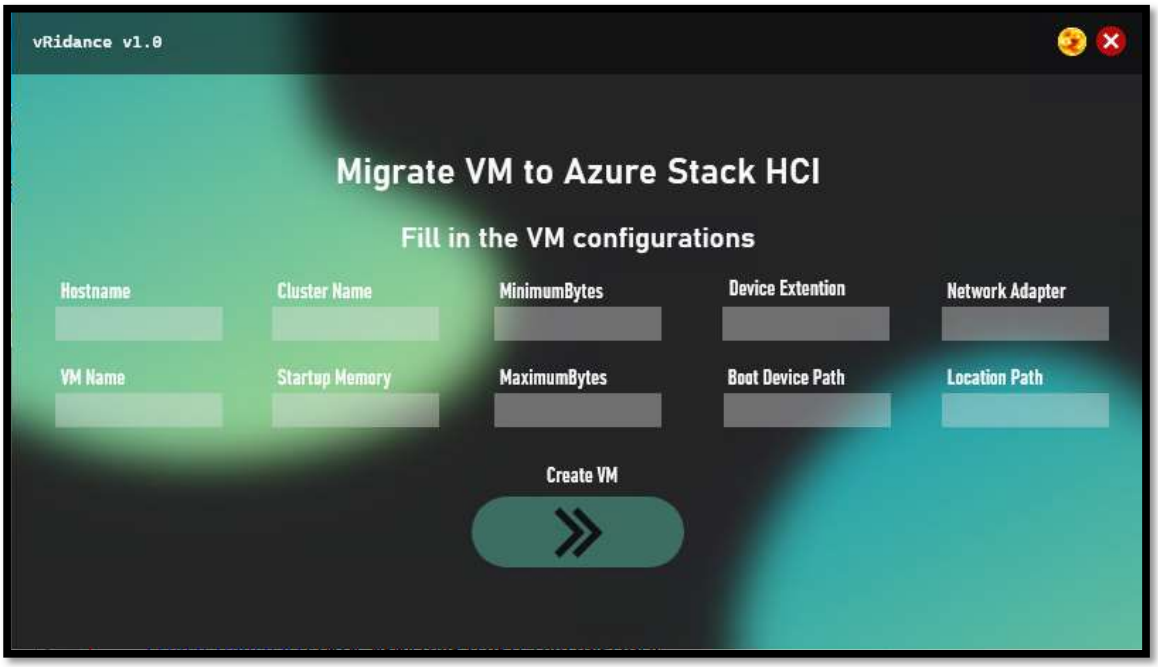
Double click on vRidance.exe, this will open the GUI.



In this window we fill in all the configurations for the VM.

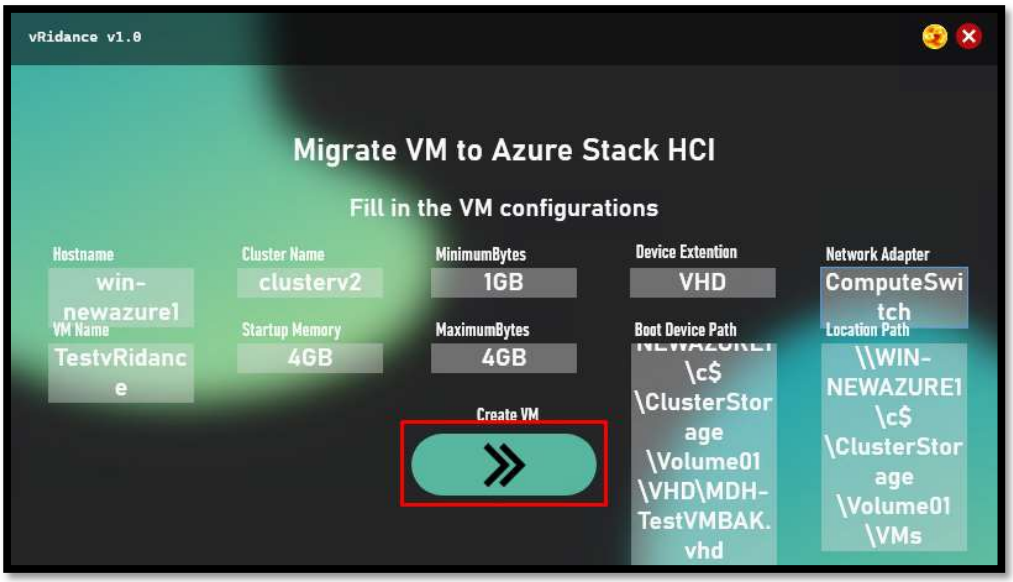



- Hostname = The name of the Host where the VM is being placed
- VMName = Name of the VM
- ClusterName = Name of the cluster where we place the VM
- StartupMemory = How much memory we give at startup
- MinimumMemory = Minimum memory we give to the VM
- MaximumMemory = Maximum memory we give to the VM
- Device Extention = Extention of the harddisk file
- Boot Device Path = Location of the harddisk file
- Network adapter = Name of the network adapter
- Location Path = Location where we store the VM

Example:

- Hostname = Win-newazure1
- VMName = WindowsServer2022-001
- ClusterName = ClusterV2
- StartupMemory = 4GB
- MinimumMemory = 1GB
- MaximumMemory = 4GB
- Device Extention = VHD
- Boot Device Path = c:\diskpath\disk.vhd
- Network adapter = ComputeSwitch
- Location Path = c:\storagepath\

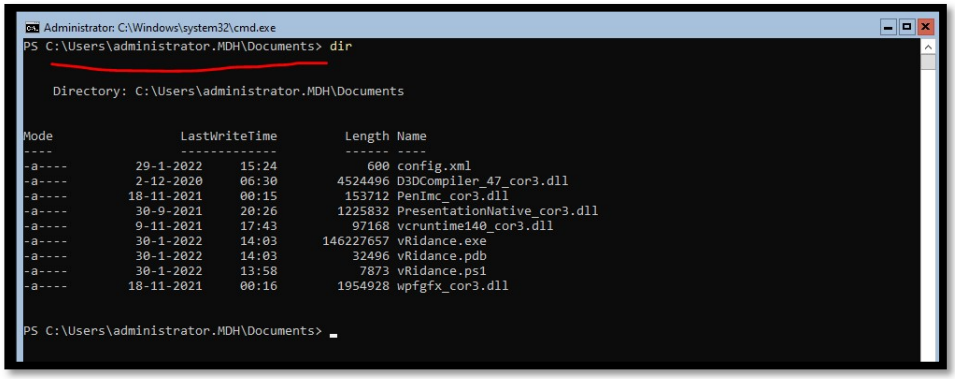
When filling in these textboxes, the code behind this GUI will send all the values into an xml file that the powershell script will need for the migration and installation of the VM.




### 3.4.4 Step 4: Start the powershell script on the azure node

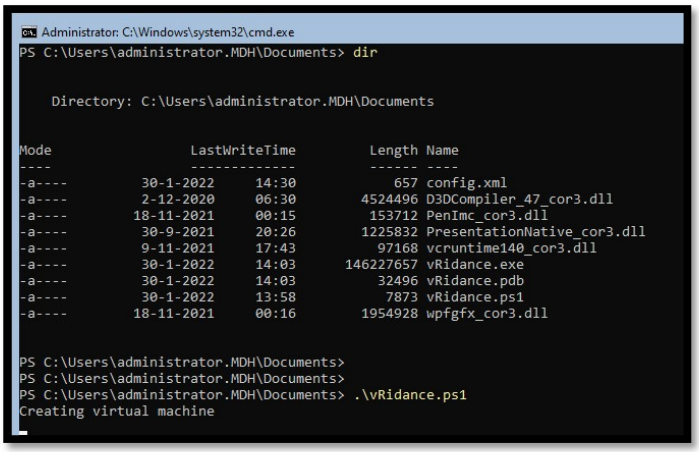
Now go to the machine with the hostname you typed in. In my case I used “win-newazure1”.

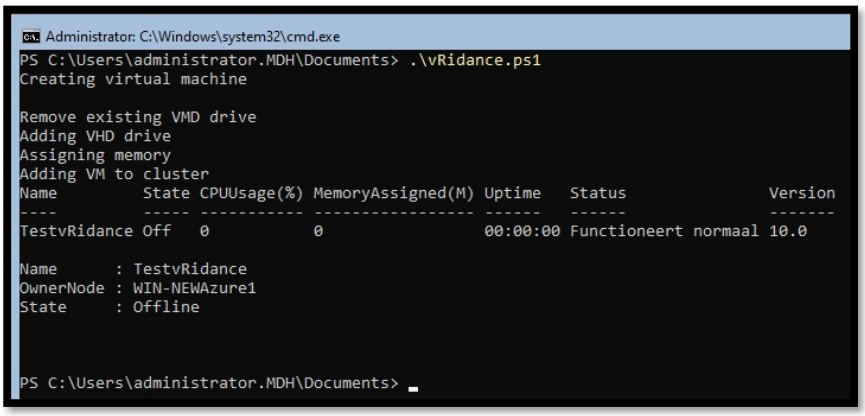
In the source folder copy and past all these files inside the document folder of the user.




Why must we do this? Well the GUI works with WPF and this eis not supported one these Azure nodes and we only found this out at the last moment.

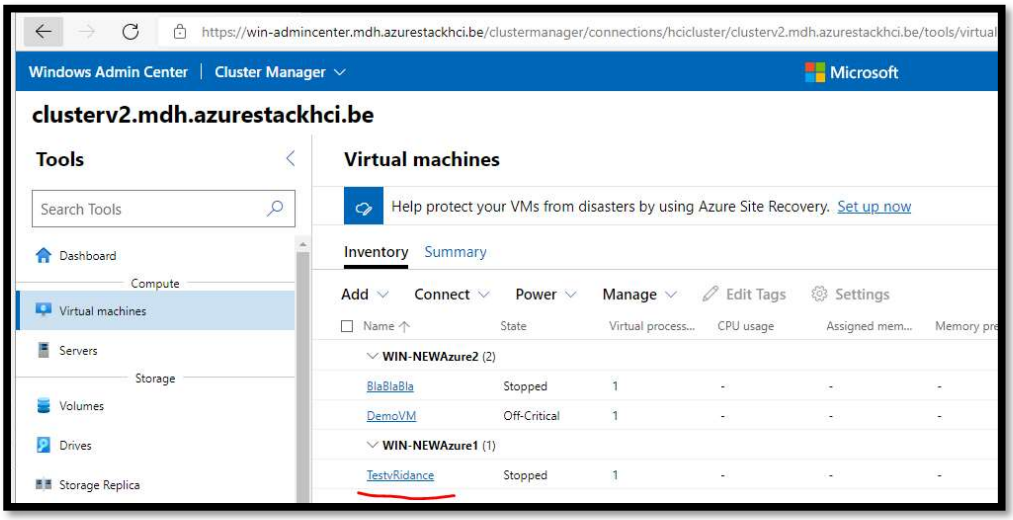
Now we use a mgmt machine to start and complete the GUI. The GUI makes the config.xml file and after that we go to the Azure Node VM (the hostname we specified) and run vRidance.ps1




### 3.4.5 Step 5: Windows Admin Center

Open windows admin center, and log into the cluster. In the virtual machine tab you should see the VM.



# 4 Citrix

## 4.1 What is Citrix?

Citrix is an American software company specialized in virtualization, networking and Software as a Service (SaaS) based on Xen. Xen is a free open source Virtual Machine Monitor (VMM). Xen makes it possible to run multiple machines on the same hardware. This is being accomplished by the technique called paravirtualization.

## 4.2 Target Audience

This document is targeted to System Administrators that have a basic knowledge of networking and virtualization. This being Hypervisors, Virtualization Software, Basic subnetting, system installation and basic Windows/Linux knowledge.

### 4.2.1 Requirements

#### 4.2.1.1 System Requirements

H5 Citrix Hypervisor

The Citrix Hypervisor server must be a 64-bit x86 server-class machine devoted to hosting VMs. Citrix Hypervisor creates an optimized and hardened Linux partition with a Xen-enabled kernel. This kernel controls the interaction between the virtualized devices seen by VMs and the physical hardware.

(https://docs.citrix.com/en-us/citrix-hypervisor/system-requirements.html)

- CPU: One or more 64-bit x86 CPUs, 1.5 GHz minimum, 2 GHz or faster multicore CPU recommended.
- Storage: Locally attached storage with 46 GB of free disk space minimum, 70 GB recommended. SAN via HBA (not through software) when installing with multipath boot from SAN.
- RAM: 2 GB minimum, 4 GB or more recommended
- Network: 100 Mbit/s or faster NIC. One or more Gb, or 10 Gb NICs is recommended for faster export/import data transfers and VM live migration. (Requires an IPv4 network for management and storage traffic)

H5 Windows Server (Tested on 2019 and 2022)

The windows server will serve its purpose as the Citrix Xencenter. This software is the same piece of software as vSphere. Here you will be able to create, edit or remove Virtual Machines.

- CPU: 750 MHz minimum, 1 GHz or faster recommended
- RAM: 1 GB minimum, 2 GB or more recommended
- Storage: 100 MB minimum
- Network: 100 Mbit/s or faster NIC
- Screen Res: 1024x768 pixels, minimum

#### 4.2.1.2 Software Requirements

H5 Citrix Hypervisor

- /

H5 Windows Server

- .NET Framework: Version 4.8
- Citrix XenCenter

#### 4.2.1.3 Other Requirements

- Machines must have a consistent IP address (a static IP address on the server or a static DHCP lease). This requirement also applies to the servers providing shared NFS or iSCSI storage.
- Its system clock must be synchronized to the pool master (for example, through NTP).
- It cannot have any running or suspended VMs or any active operations in progress on its VMs, such as shutting down or exporting. Shut down all VMs on the server before adding it to a pool.
- It cannot have any shared storage already configured.
- It cannot have a bonded management interface. Reconfigure the management interface and move it on to a physical NIC before adding the server to the pool. After the server has joined the pool, you can reconfigure the management interface again.
- It must be running the same version of Citrix Hypervisor, at the same patch level, as servers already in the pool.
- It must be configured with the same supplemental packs as the servers already in the pool. Supplemental packs are used to install add-on software into the Citrix Hypervisor control domain, dom0. To prevent an inconsistent user experience across a pool, all servers in the pool must have the same supplemental packs at the same revision installed.
- It must have the same Citrix Hypervisor license as the servers already in the pool. You can change the license of any pool members after joining the pool. The server with the lowest license determines the features available to all members in the pool.
- Servers providing shared NFS or iSCSI storage for the pool must have a static IP address or be DNS addressable.

## 4.3 Environment Setup

In this document, we will guide you through the installation process of the Citrix Virtualization Environment.

### 4.3.1 Preparing the Environment

To prepare our installation, we will need a few things before we start. To start off, let's go to the Citrix website and download the few necessities for this project. In order to download items from the Citrix website you will need an account. If you do not have one, create one first.

#### 4.3.1.1 Citrix Hypervisor

- Download the Citrix Hypervisor 8.2.0 Base Installation ISO

- (https://secureportal.citrix.com/udl.asp?DLID=17363&URL=https://downloads.citrix.com/17363/CitrixHypervisor-8.2.0-install-cd.iso)

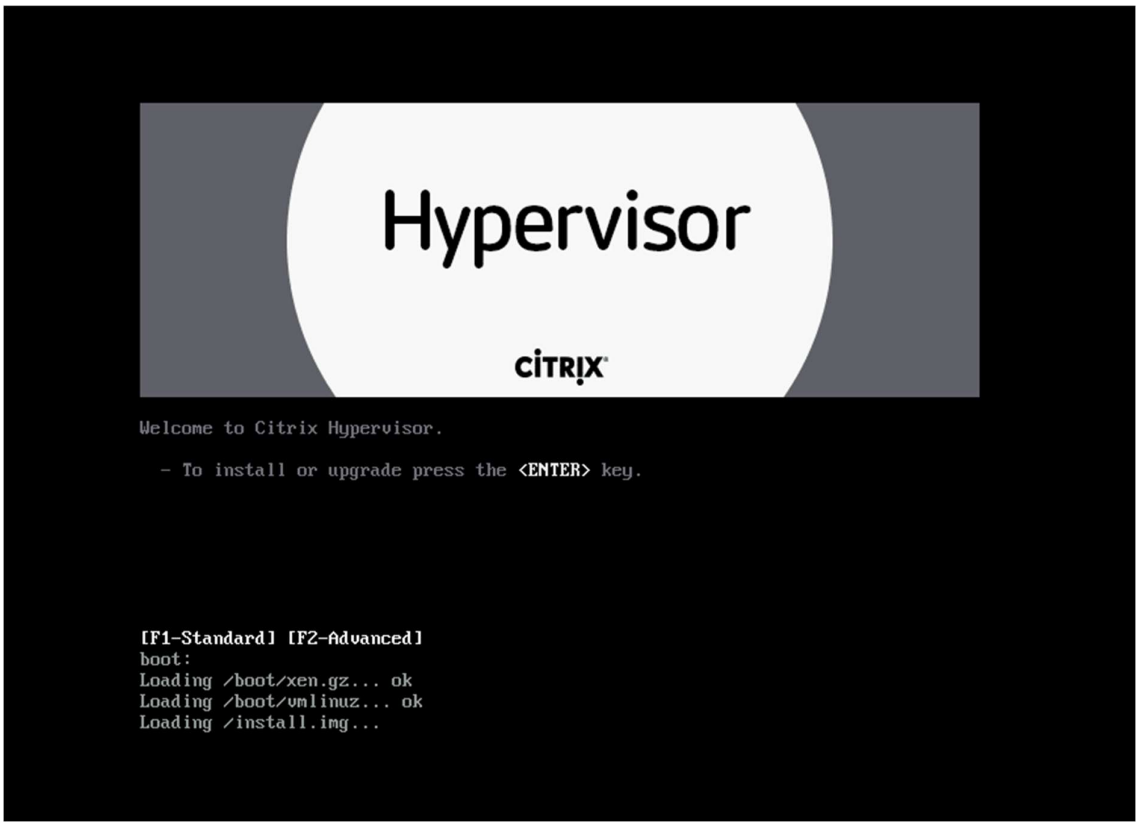
#### 4.3.1.2 Windows Server 2019

- Download the XenCenter 8.2.3 Windows Management Console software.
  - (https://secureportal.citrix.com/Licensing/Downloads/UnrestrictedDL.aspx?DLID=17407&URL=https://downloads.citrix.com/17407/CitrixHypervisor-XenCenter-8.2.3.msi)

### 4.3.2 Installing the Citrix Hypervisor

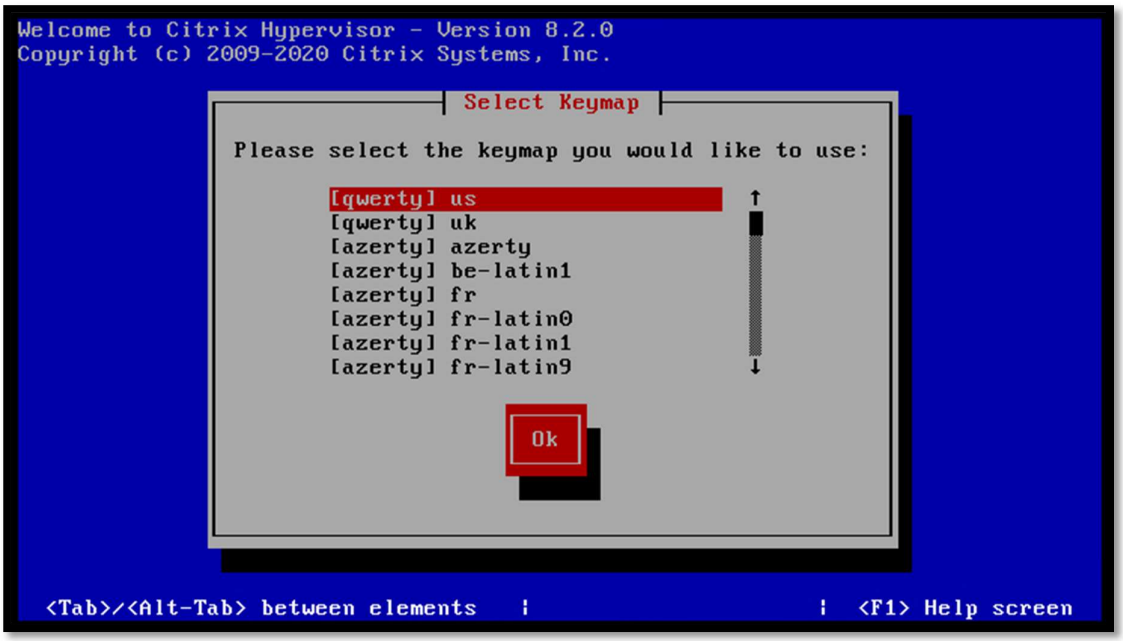
To install the Citrix Hypervisor you will need the basic knowledge on how to mount an ISO file to either a physical or virtual machine.

Boot up your machine with the ISO file attached, you can follow the following steps in this document to complete the Citrix Hypervisor installation.



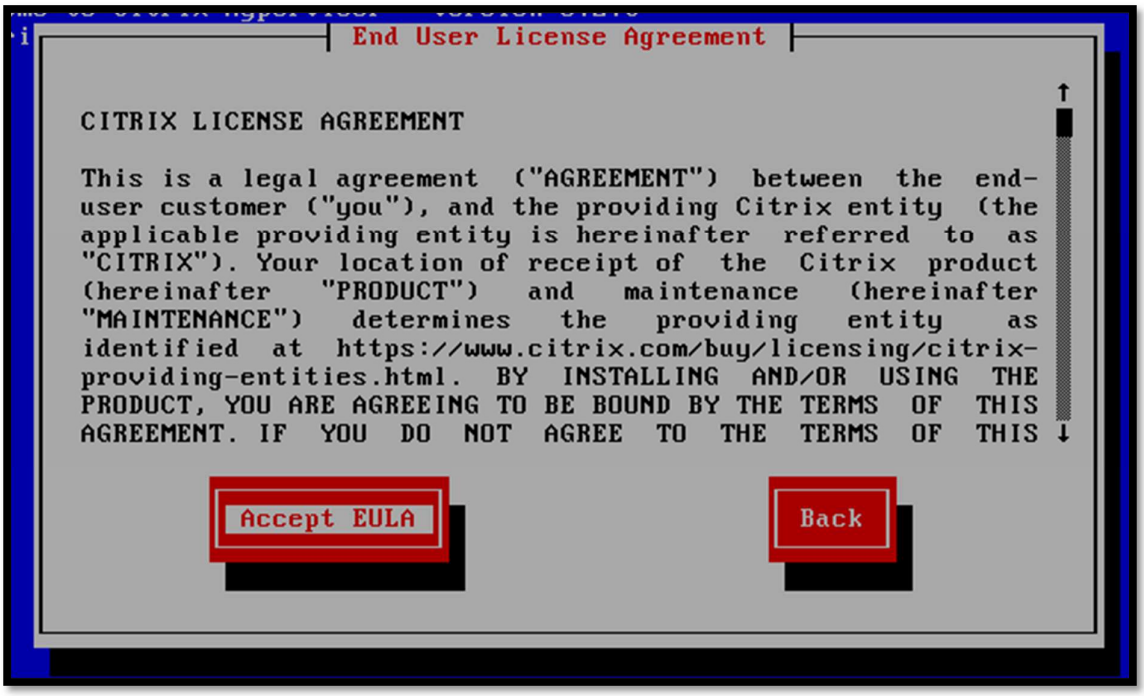

Press the ENTER key to start installation of Citrix HyperVisor.

It'll load up some stuff first before we get prompted with the next screen;

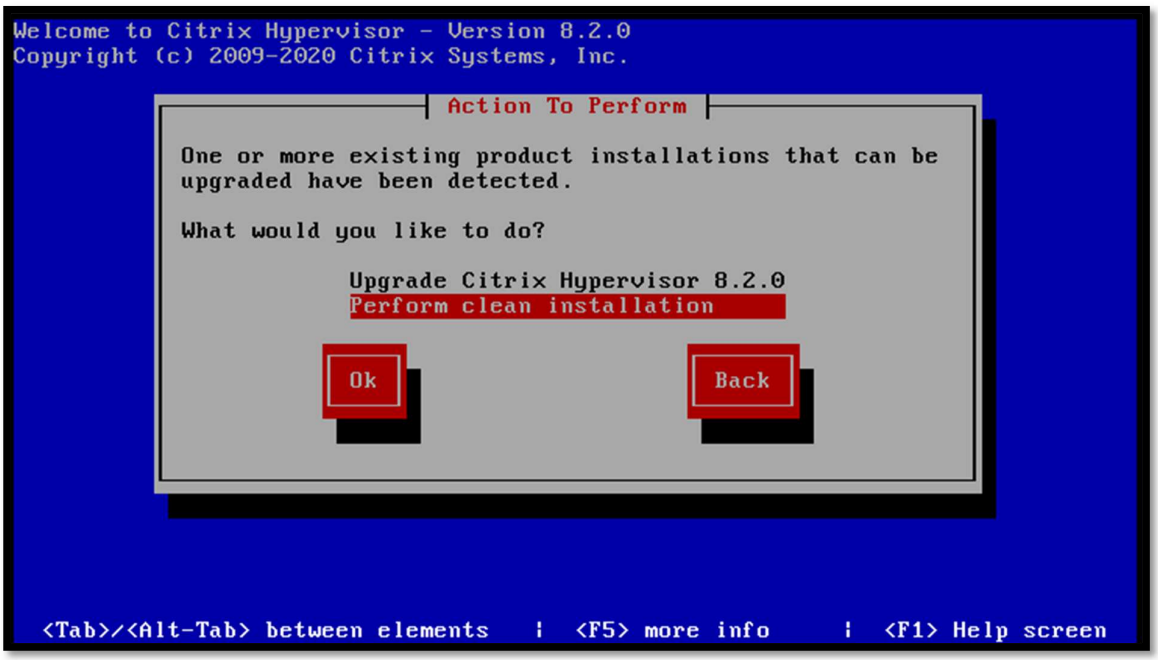



Select your keyboard layout.


Click on “Ok” to continue.

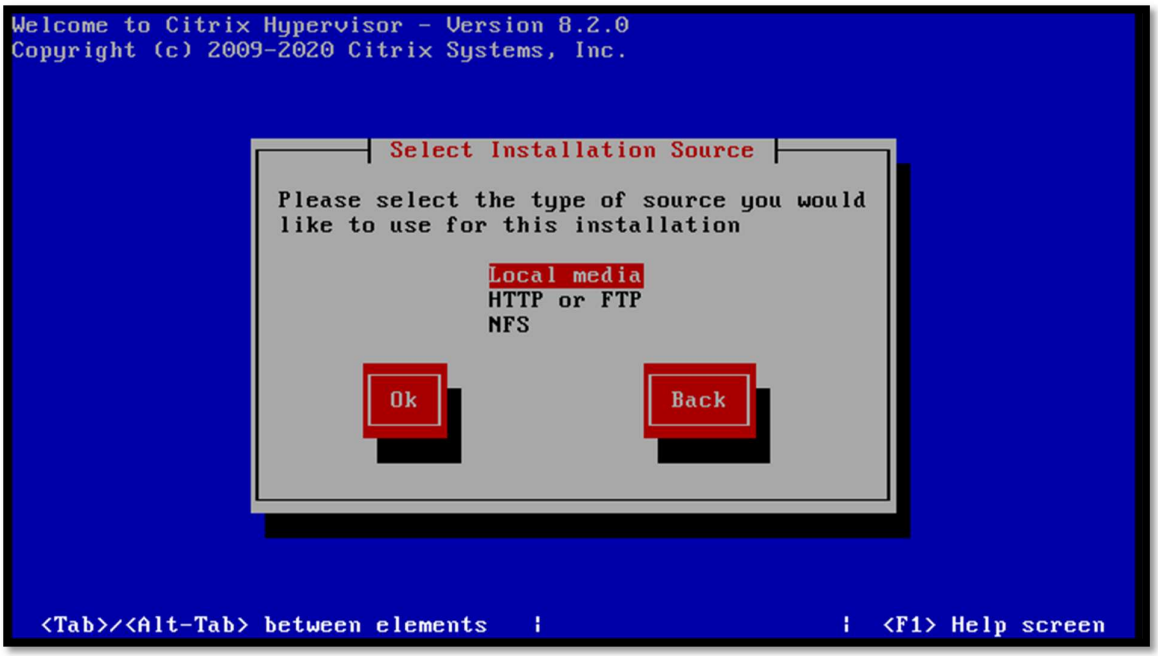



Accept the EULA.

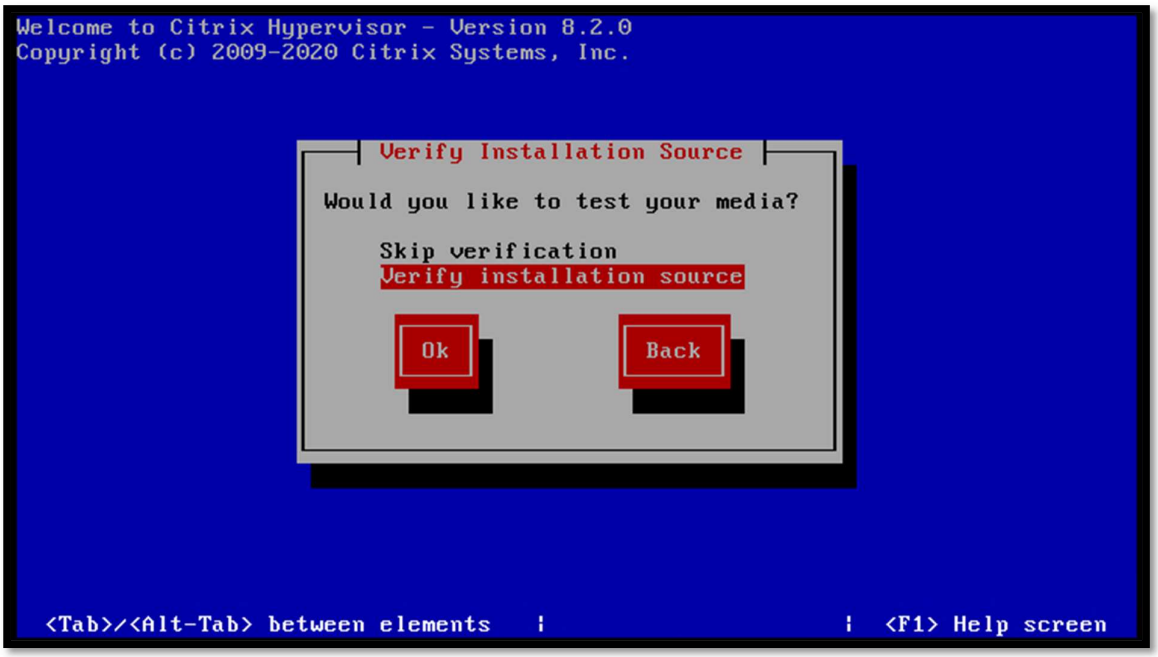



I already had a hypervisor installation present before, so we'll be performing a clean installation.

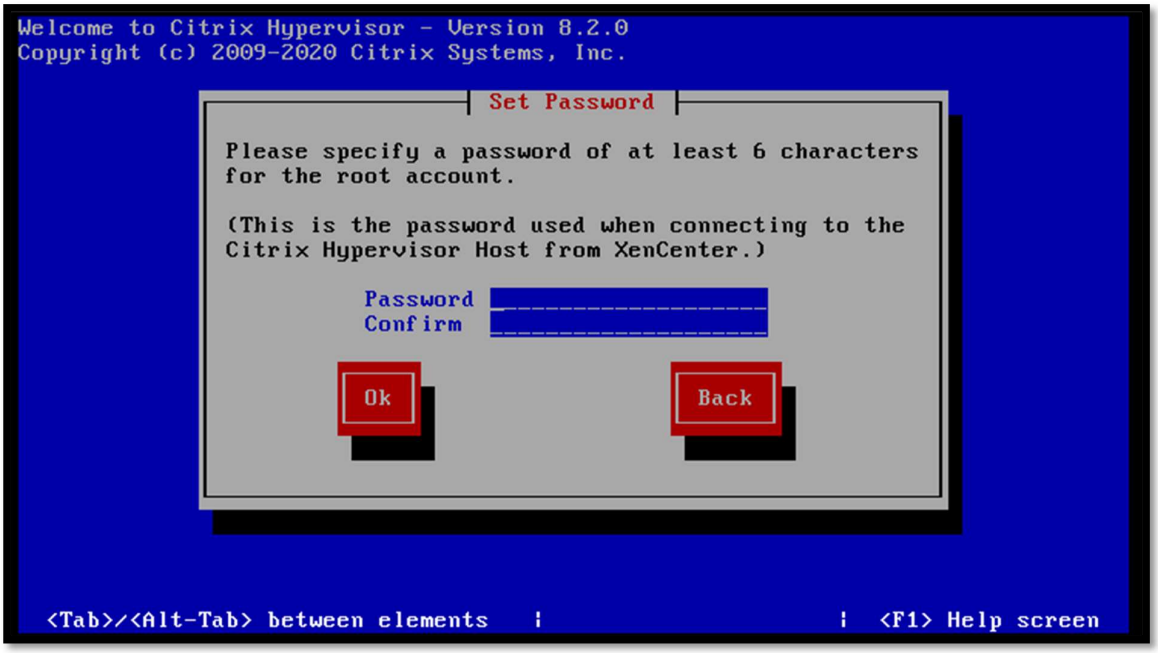

Select the disk you would like to use and enable thin provisioning. Then click “Ok”.



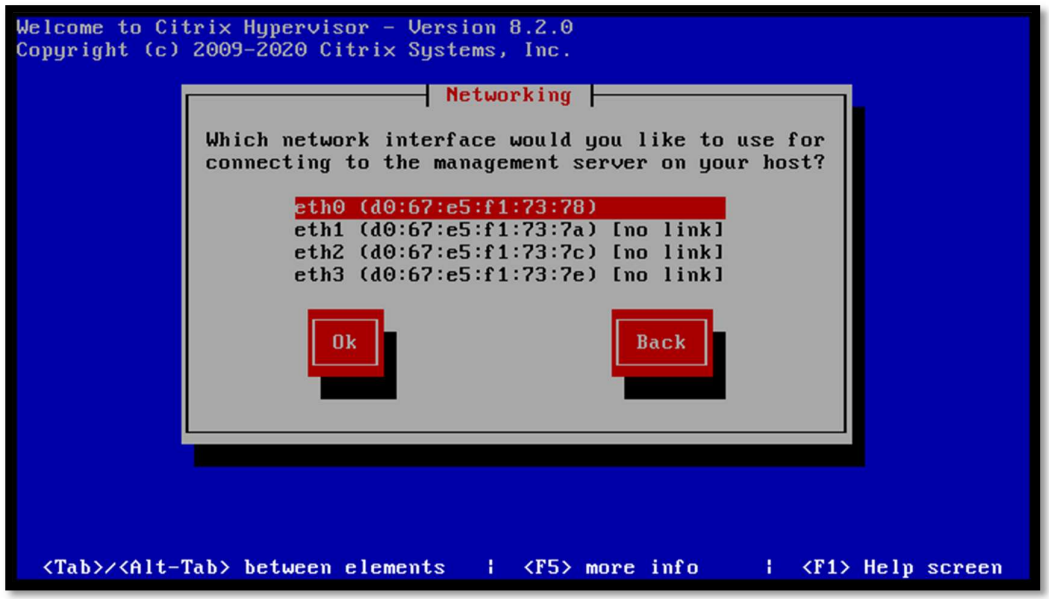

For the source, I will be going with local media.



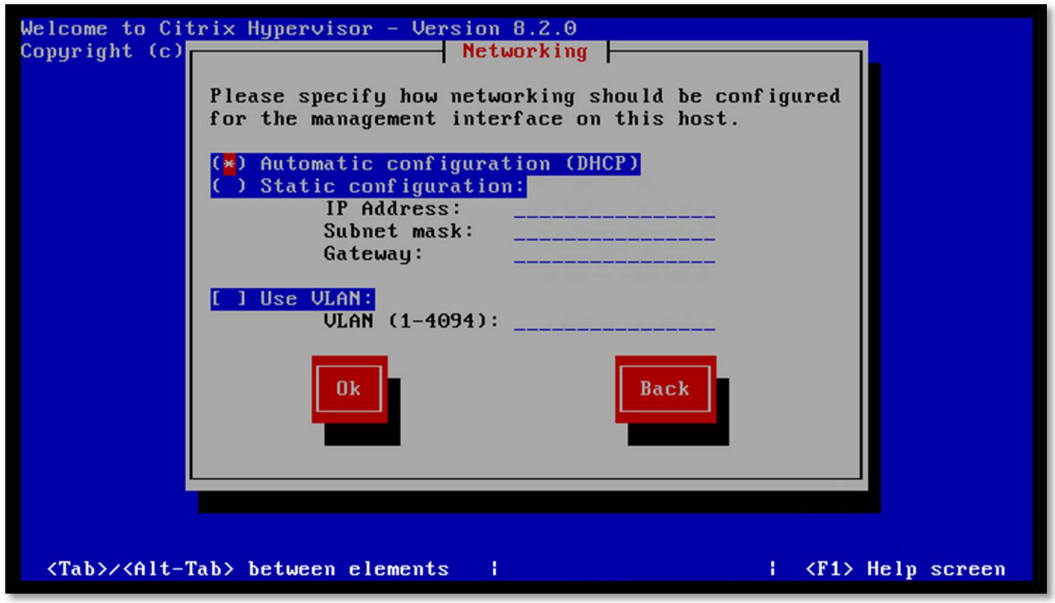

You can opt in to skip your installation source verification, however, we have downloaded from the Citrix downloads pages, so we know the source is valid. I will skip verification.

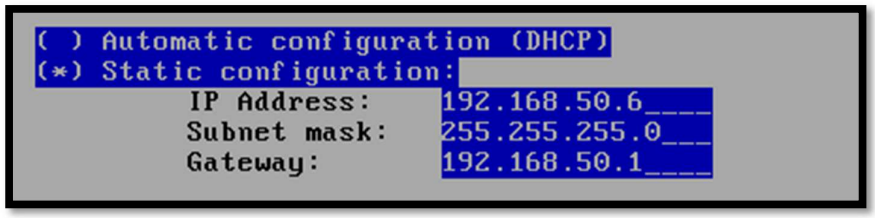



Now we'll configure a Hypervisor password. Remember this password, as you will need it a lot in the future.

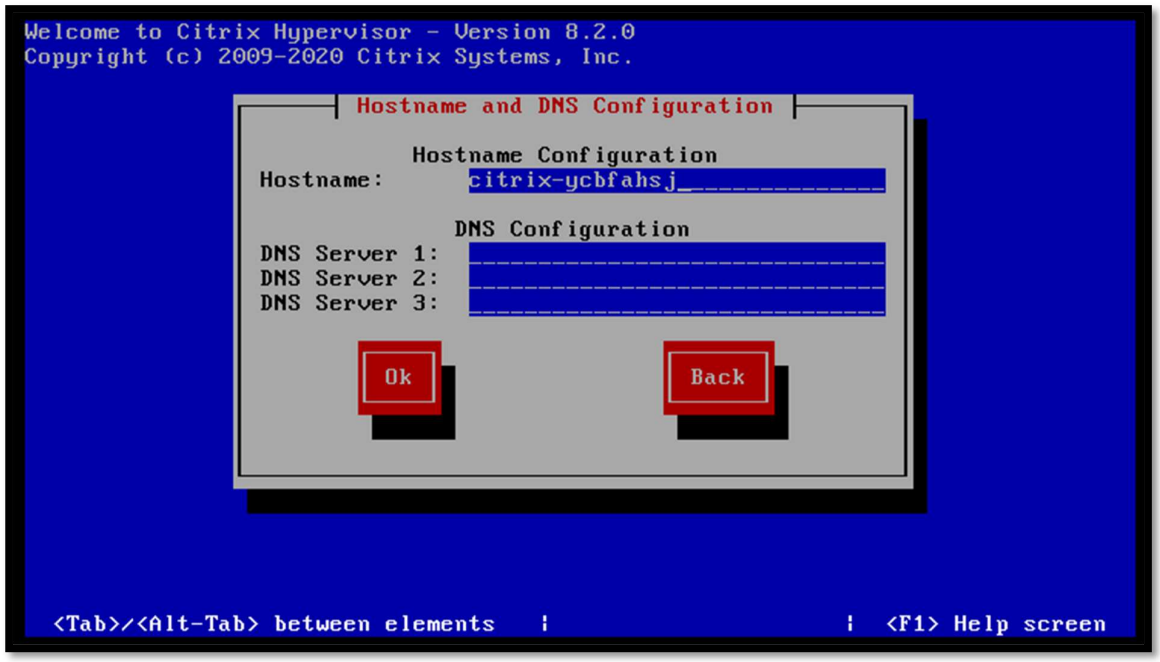



You can select the Ethernet adapter of your choice. I'm going with eth0.

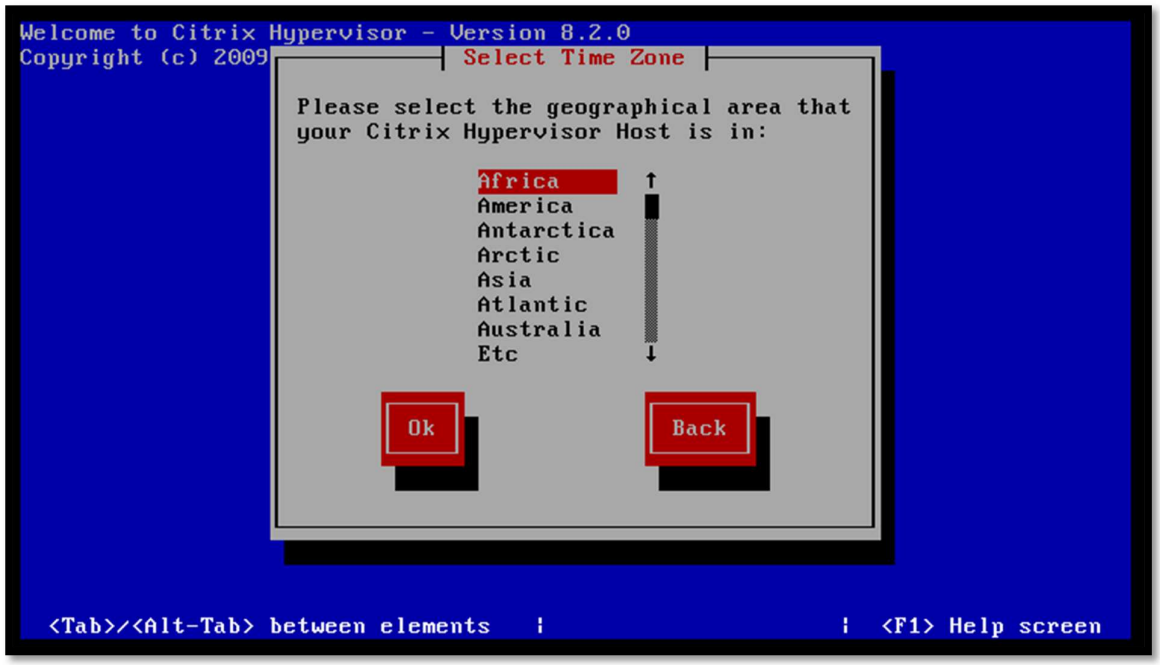



Now we'll be giving our server a static IP address. Which is recommended for servers in a whole and also described on the Citrix forums. So we'll give it a static IP.

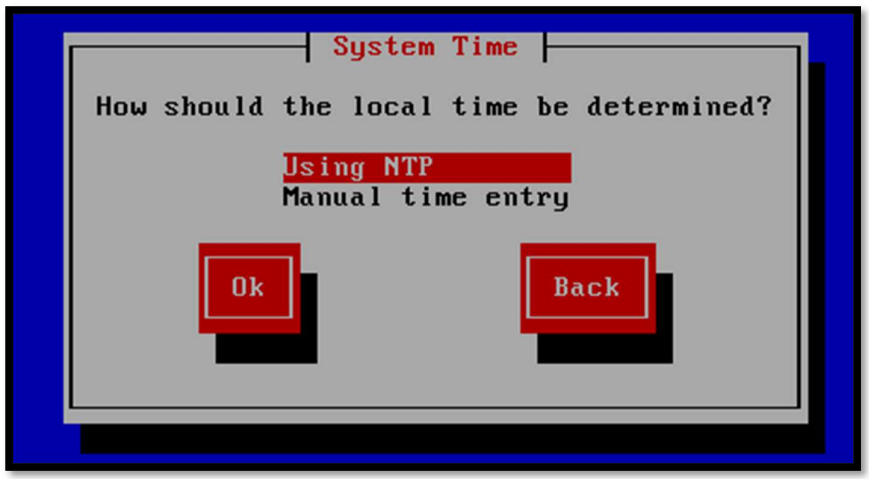



On this screen we'll configure hostname and DNS. This will be specific to your site.

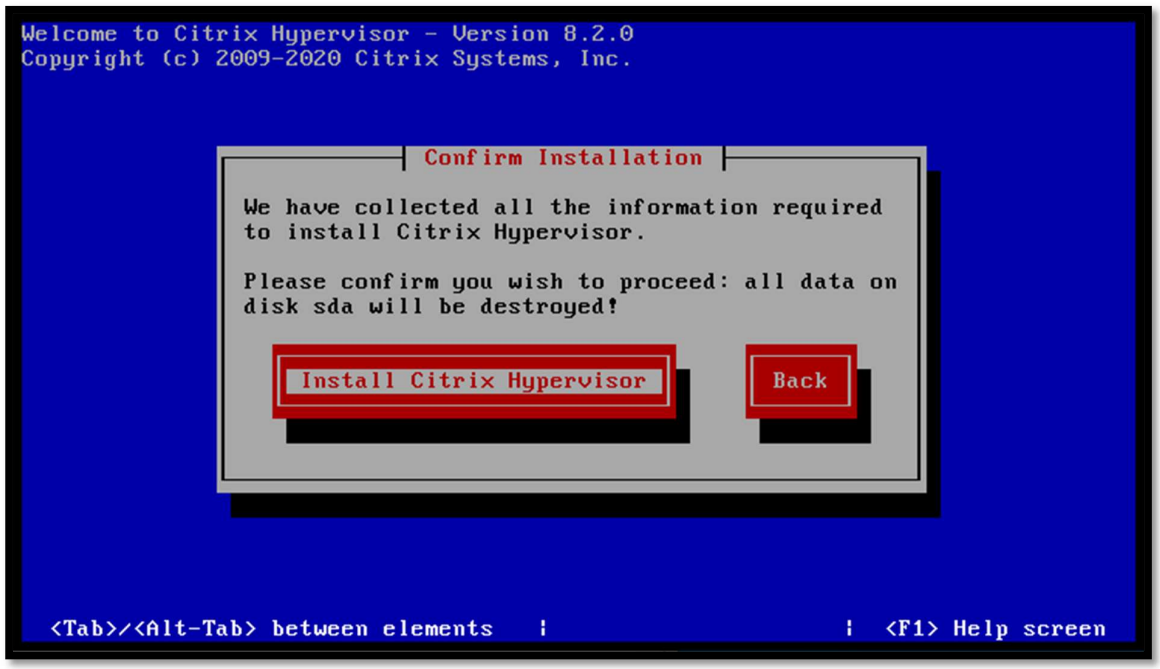



Select your Area.

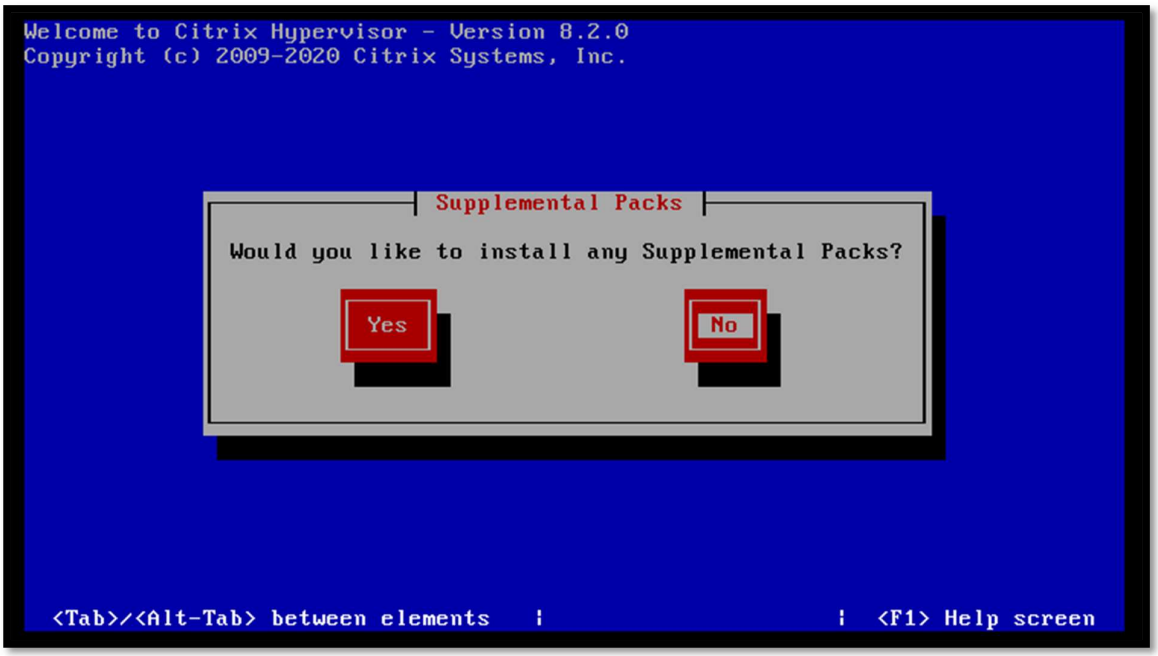
It'll prompt you to determine system time. I chose NTP synchronization.



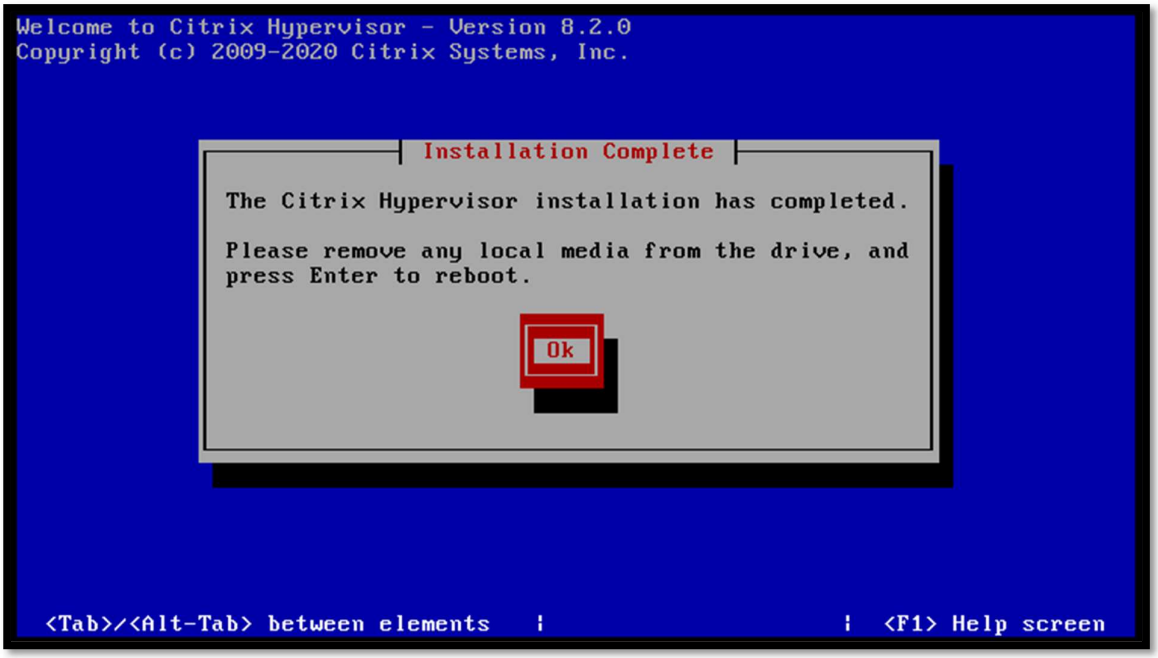

Since I'm in Europe, Belgium, I chose the be.pool.ntp.org




The next screen will install the Citrix Hypervisor! You can go ahead and click install.

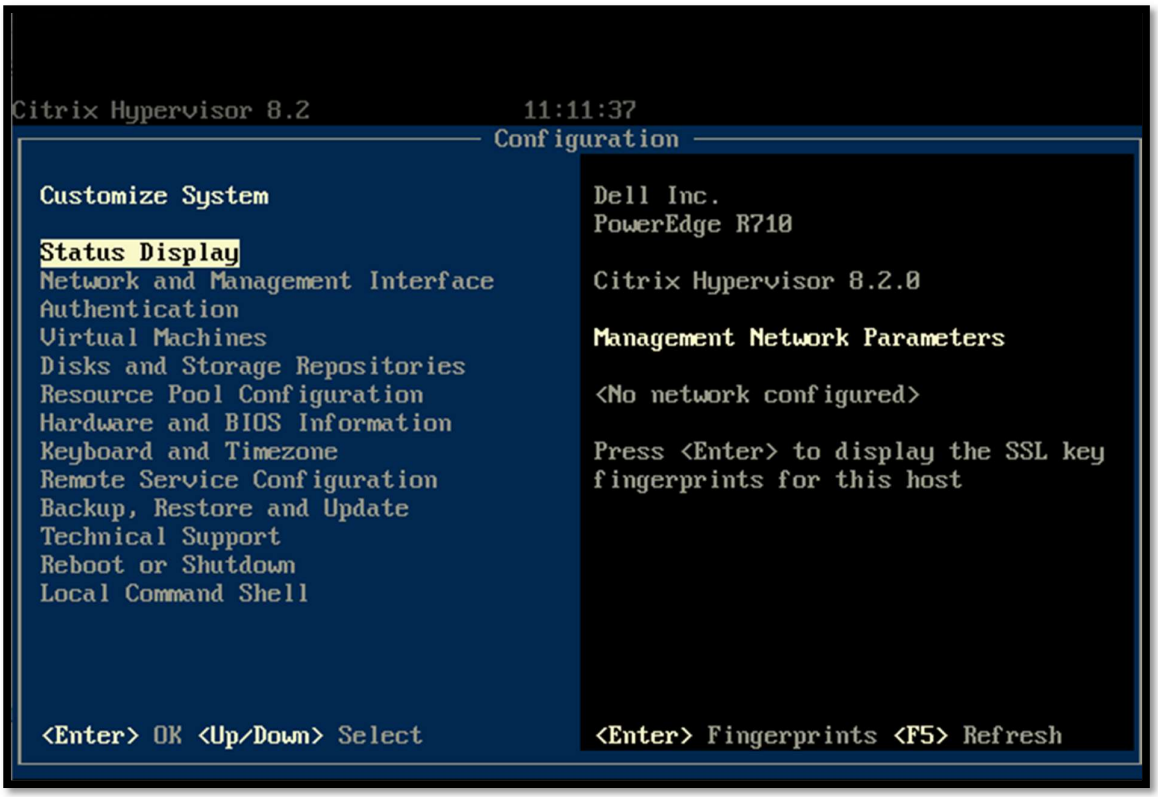



You will have the option to install supplemental packs. I chose not to, since we won't be needing them.




After finishing up the installation, your Citrix Hypervisor is now installed! You can now boot from the installation harddisk.


When booting. Select the Citrix Hypervisor. This will, obviously, boot into the Citrix Hypervisor.



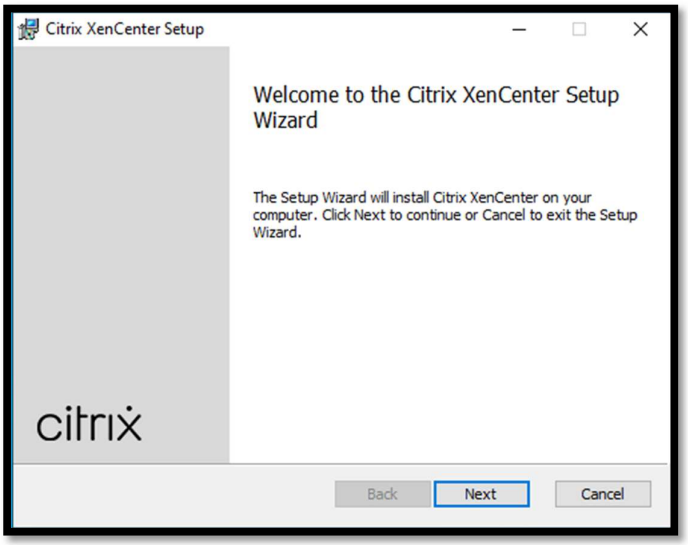

Now you're all finished! The hypervisor is now installed. We can now move on to the XenCenter install.

### 4.3.3 Installing Citrix XenCenter

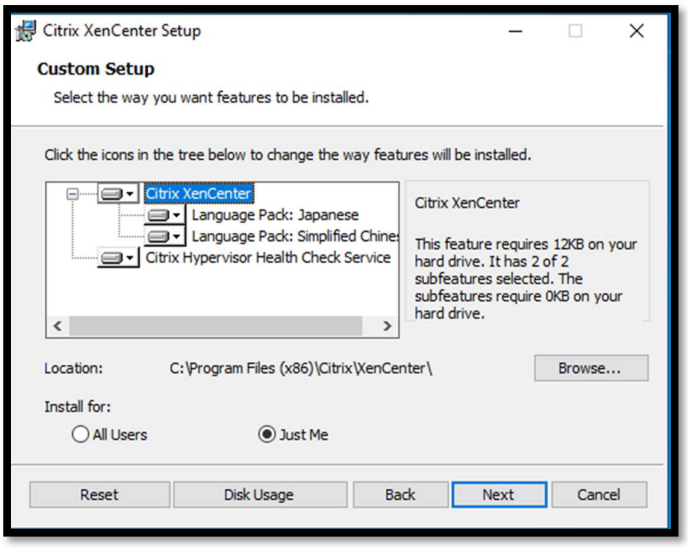
To install the Citrix XenCenter, you will need a Windows Server machine. I am using the Windows Server 2019 for this document.

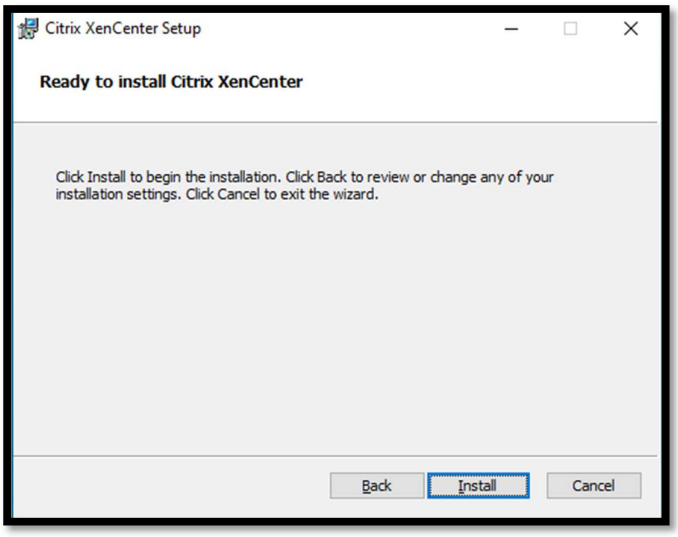
The software requirements required you to download the XenCenter MSI package. We will be installing this on our Windows Server first.

Locate the MSI installer and run it.

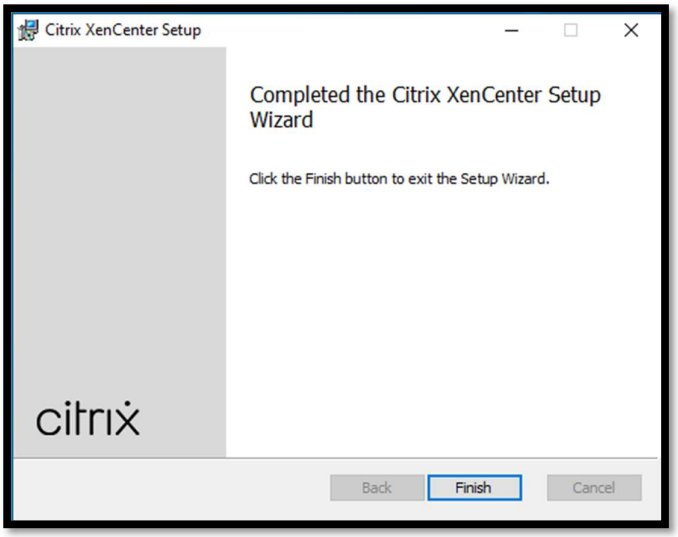



Choose where you want XenCenter installed, and for who.

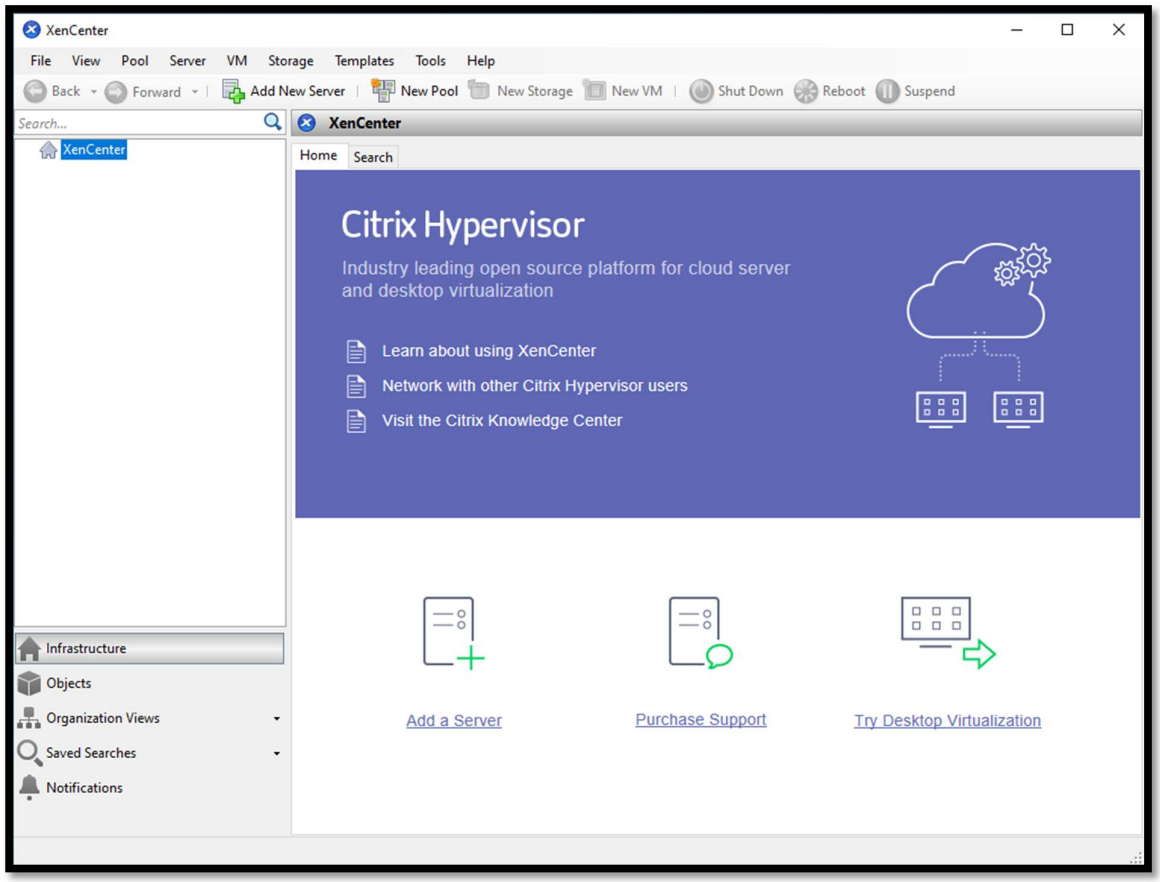




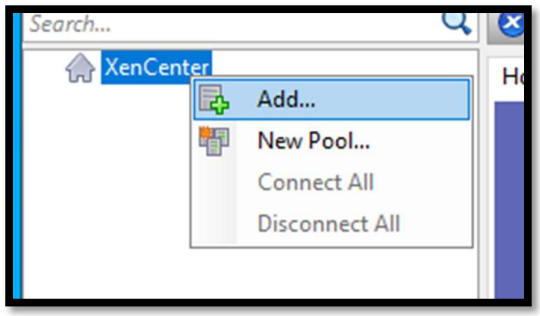

Click on the “Install” button.



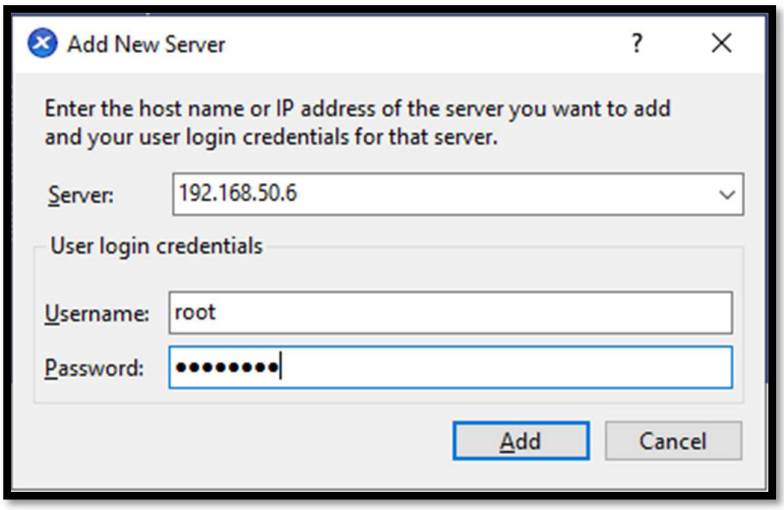

XenCenter is now installed on your Windows Server machine. You can run XenCenter from the start menu.



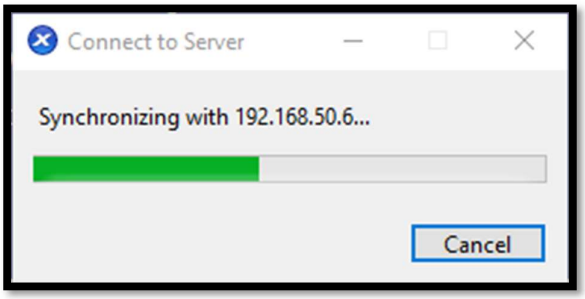

You'll be presented with this screen after initial installation of XenCenter. Here we will link our Hypervisor where our virtual machines will be hosted on. You'll be able to manage your VM's through XenCenter.




Right click XenCenter and click Add




Here, you'll need the login credentials to your Citrix Hypervisor. Go ahead and fill them out with your Hypervisor IP, Username and password and click "Add".




The Hypervisor will now sync its current state with XenCenter.

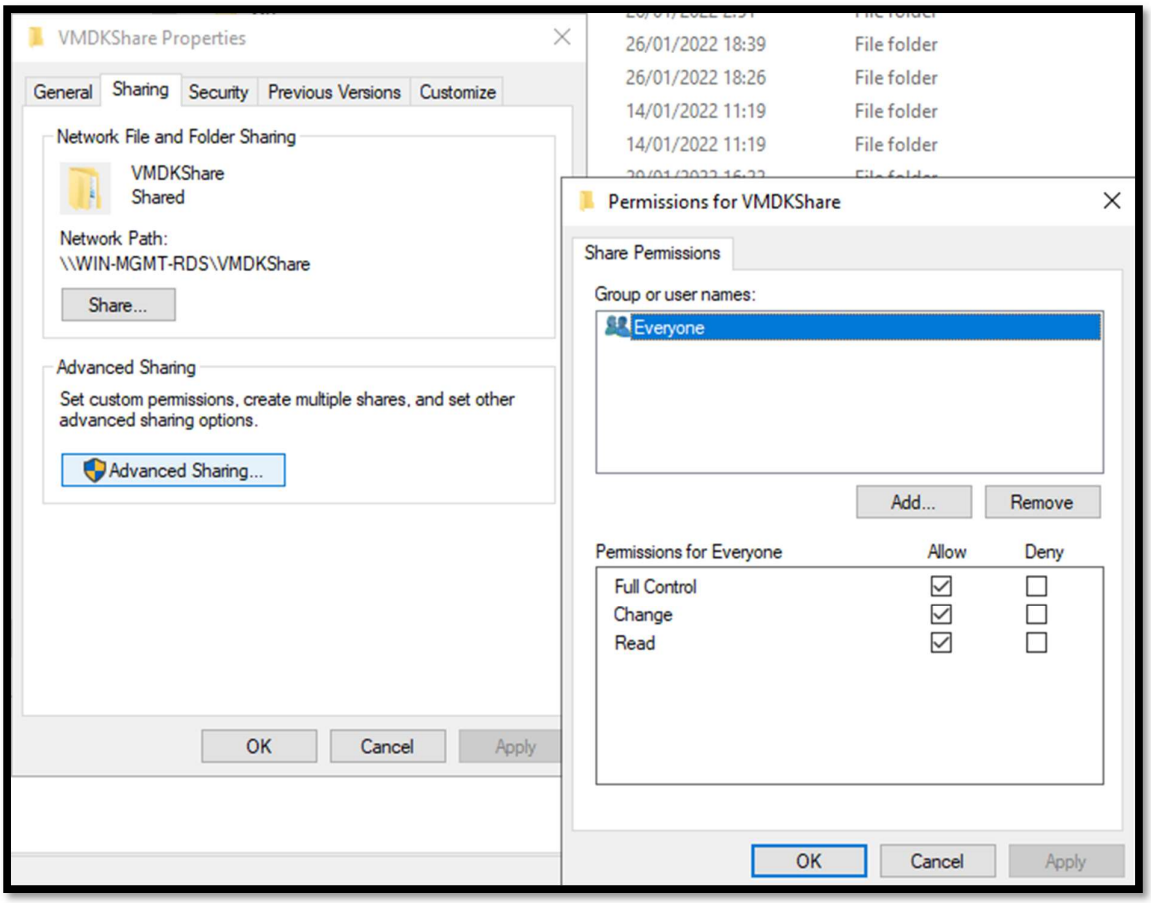

Your Hypervisor is now linked with your XenCenter software. You'll be able to do everything from here now.

### 4.3.4 Configuring NFS Share

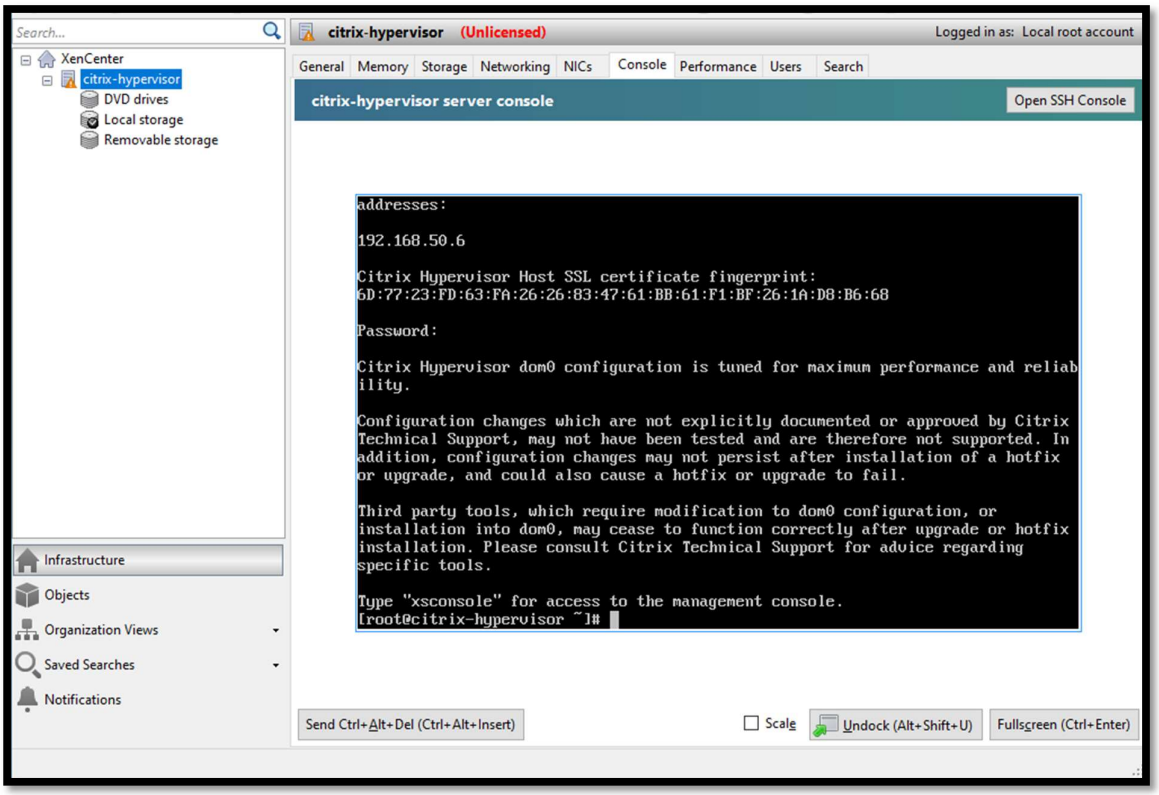
To be able to make our converted vSphere VMDK's available to the Citrix Hypervisor, I opted for an NFS share. We'll now go through the steps on how to configure that.

On the machine of your choice (preferably one with a lot of free storage, depending on how much machines you will be migrating) create a folder that's easily accessible. I chose for the C:/ drive.


To make this a shared folder. We'll go into the sharing options and share this folder.




You can set the permissions to your sites needs. I chose Everyone full control for ease of access and smooth demo. Now that this is shared, we'll link this share to our Citrix Hypervisor.

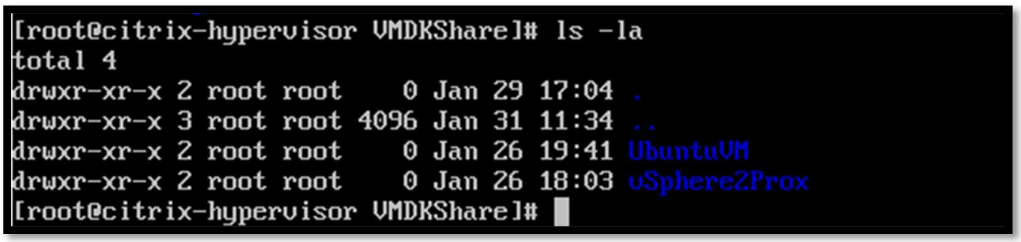



In XenCenter you can use the built-in CLI tool for the hypervisor. If you prefer, you can use an SSH connection through CMD/Putty aswell.

Now we'll mount the NFS share with the following command.

```
mount -t cifs //192.168.50.5/VMDKShare -o username=Administrator,password=YourPassword /mnt/VMDKShare
```

This command essentially links your Windows share to a mountpoint on the hypervisor.

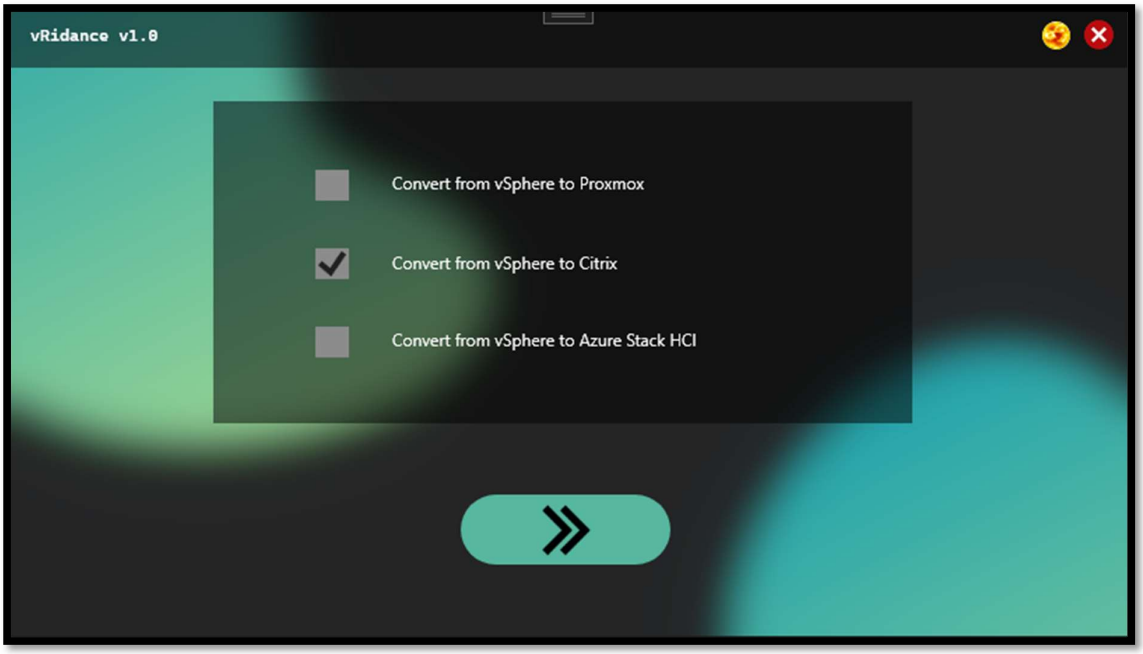



As you can see, the VMDK files from the Windows machine are now present on the hypervisor.

This sums up the basic installation for Citrix Hypervisor and Citrix XenCenter to be able to migrate Virtual Appliances from vSphere.

The next part will contain a detailed description of how the vRidance migration tool works, and how you can understand the code behind to maybe alter the source code to your needs.

## 4.4 Explaining the code behind (Citrix Pages)

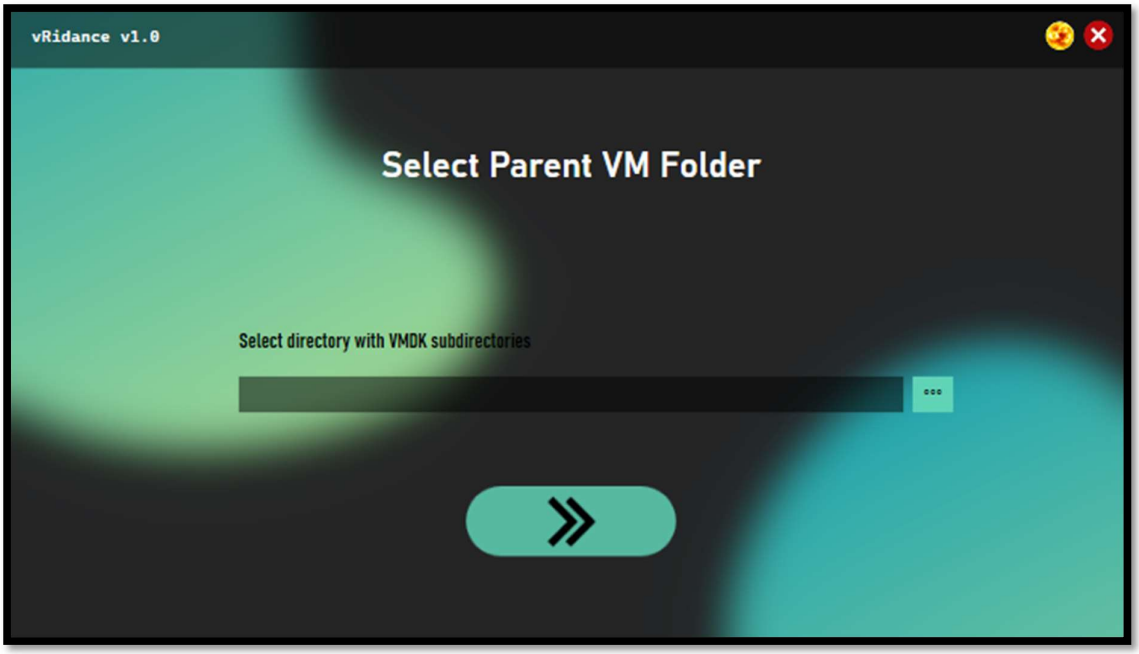



```
        private void rectNext2_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            CheckBox[] checkBoxes = new CheckBox[3];
            checkBoxes[0] = chckProxmox;
            checkBoxes[1] = chckCitrix;
            checkBoxes[2] = chckAzureStack;

            for (int i = 0; i <= checkBoxes.Count() - 1; i++)
            {

                if (checkBoxes[i].IsChecked == true && checkBoxes[i].IsEnabled)
                {
                    ConvertToPlatform(i);
                }

            }
        }
```

This will check which checkbox has been checked, it returns the index value, as i, to ConvertToPlatform as parameter.

```
public void ConvertToPlatform(int i)
{
    string curTheme = "";
    if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
    else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";
    switch (i)
    {
        case 0:
        case 1:
            ChooseFolder chooseFolderWindowCitrix = new ChooseFolder(curTheme,"citrix");
            ((MainWindow)this.Owner).Content = chooseFolderWindowCitrix.Content;

            chooseFolderWindowCitrix.Owner = ((MainWindow)this.Owner);
            break;
        case 2:
            break;
    }
}
```

As you can see ConvertToPlatform has the parameter int i. A switchcase will determine which screen will be opened depending which checkbox was selected. In this case, it's the citrix checkbox. This will open a new screen "ChooseFolder" and set itself as the new owner. The owner property is needed for dragging the screen around, since we have our own custom window. We have a parameter for the light/dark mode that we forward to the next screen, aswell as the platform selected.




Next screen will give you the option to select the VMDK parent folder. This is well documented in the User Manual for vRidance. The code behind will be explained below.

```
        private void txtPath_TextChanged(object sender, TextChangedEventArgs e)
        {
            string systemPath = txtPath.Text;
            if (Directory.Exists(systemPath) && (systemPath != null || systemPath != ""))
            {
                rectNext.Opacity = 1;
                rectNext.IsEnabled = true;
            }
            else
            {
                rectNext.Opacity = 0.5;
                rectNext.IsEnabled = false;
            }
        }
```

When altering the text of txtPath (path input) it will check with each keystroke if your entered directory exists. This will, when the path is valid, enable the next button at the bottom of the screen.

A folderselection is also implemented but essentially does the same thing besides getting the path from a selected folder in the UI of windows. The code below is what handles this;

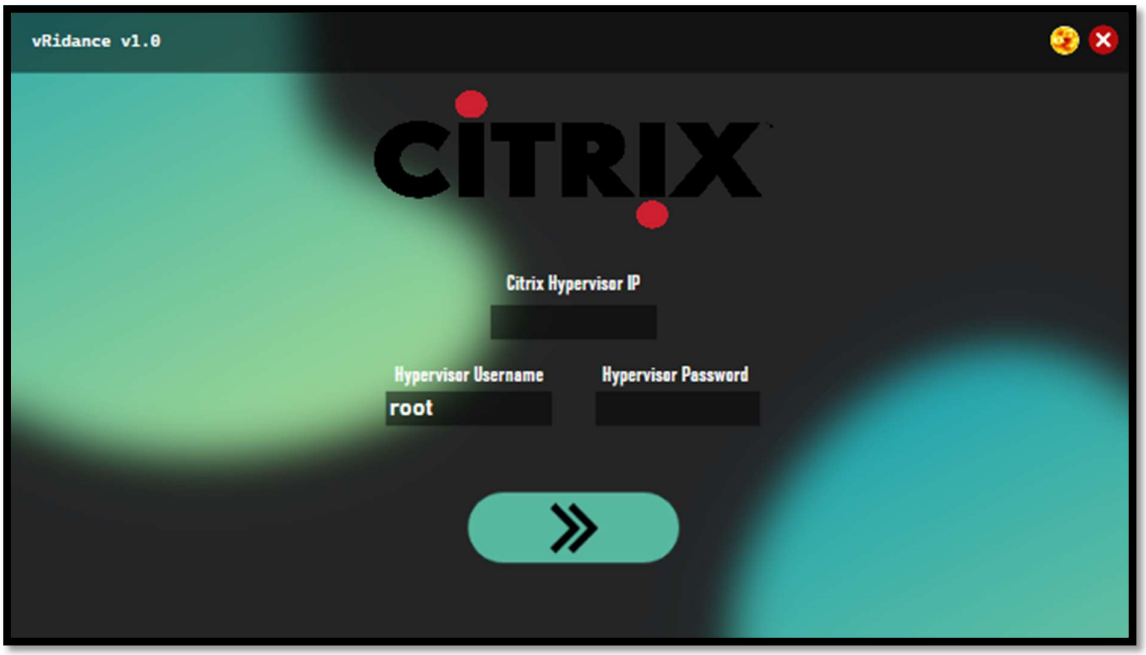
```
        private void btnSearch_Click(object sender, RoutedEventArgs e)
        {
            using (var fbd = new FolderBrowserDialog())
            {
                DialogResult result = fbd.ShowDialog();
                if (!string.IsNullOrWhiteSpace(fbd.SelectedPath))
                {
                    txtPath.Text = fbd.SelectedPath;
                }
                else if (txtPath.Text == "")
                {
                    System.Windows.MessageBox.Show("ERROR: Filepath empty!");
                }
            }
        }
```

This opens a browser dialog, when selecting a folder it will get the system path of the selected folder and in terms updates the txtPath, which triggers the textchanged function to check if the path exists/is valid or not.

Clicking the continue button triggers the following code;

```
public void rectNext_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            string curTheme = "";
            string path2vmdk = txtPath.Text;
            if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
            else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";
            string[] subdirectoryEntries = Directory.GetDirectories(path2vmdk);
            switch (platform)
            {
                case "proxmox":
                    break;

                case "citrix":
                    if (subdirectoryEntries.Length == 0)
                    {
                        System.Windows.MessageBox.Show($"{path2vmdk}  does  not  contain  any
subdirectories with VMDKs");

                    }
                    else
                    {
                        foreach (var var_subdirectory in subdirectoryEntries)
                        {
                            string[]  VMDirectory  =  Directory.GetFiles(@""  +  var_subdirectory,
"*.vmdk");
                            if (VMDirectory.Length == 0)
                            {
                                System.Windows.MessageBox.Show($"The  folder  {var_subdirectory}
does not have any .vmdk files");
                                break;
                            }
                            else
                            {
                                CitrixLogin citrixLogin = new CitrixLogin(curTheme, path2vmdk);
                                ((MainWindow)this.Owner).Content = citrixLogin.Content;
                                citrixLogin.Owner = ((MainWindow)this.Owner);
                            }
                        }
                    }
                    break;
            }
        }
```

From the previous screen, we forwarded a parameter called “citrix”. This is where this parameter comes in. When clicking the Continue button, it will check which parameter was forwarded in a switchcase. This is in our case Citrix. This code will check if the Parent folder has subfolders i.e. UbuntuVM, vSphere2Prox… These will each contain their own VMDK files. If these don’t exist within the parent directory provided, it will throw you an error saying no VMDK files were found. If it does however have VMDK files present, it will continue to the next screen “CitrixLogin” where we’ll forward the path to the VMDK files and initialize the next screen.




```
This screen will ask you for the hypervisor credentials. The below code is how this is handled.
        private void grdMain4_LayoutUpdated(object sender, EventArgs e)
        {
            System.Net.IPAddress ipAddress;
            if (txtUsername.Text != "" && txtUsername != null && pwbPassword.Password != "" &&
System.Net.IPAddress.TryParse(txtIP.Text, out ipAddress))
            {
                rectNext.Opacity = 1;
                rectNext.IsEnabled = true;
            }
            else
            {
                rectNext.Opacity = 0.5;
                rectNext.IsEnabled = false;
            }
        }
```

When entering your credentials, various checks are executed on the users input.

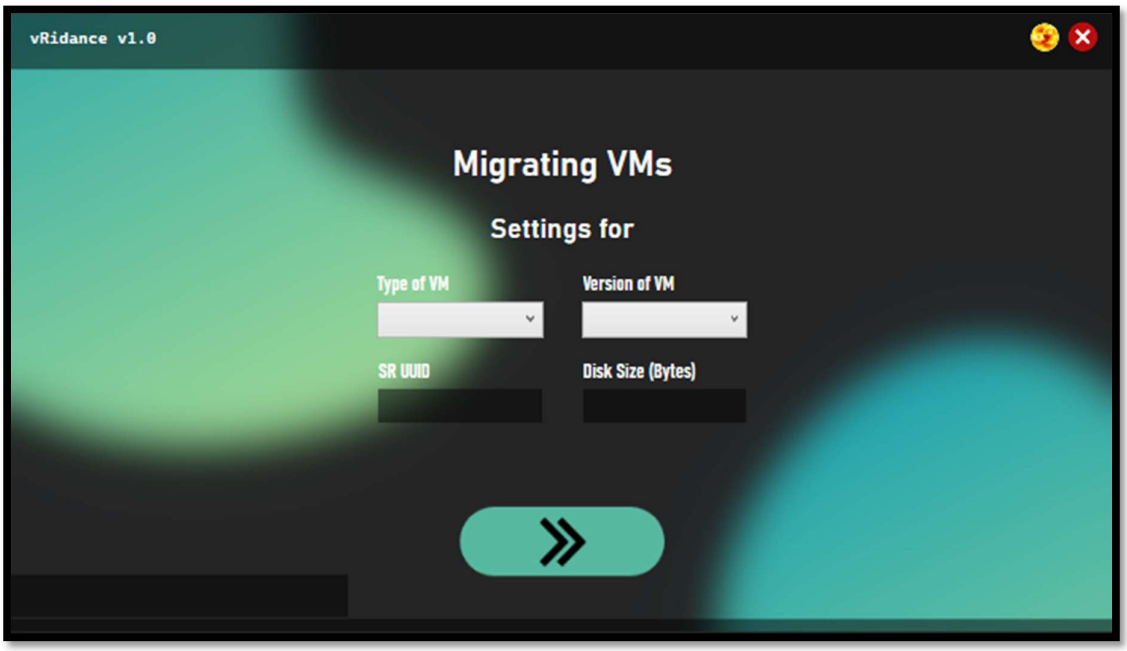
1. Checking if the Hypervisor IP is actually a valid IP Address
2. Checking if Username is not empty or null
3. Checking if password is not empty or null

If all of the above are met, the Continue button will be enabled, where more checks will be done.

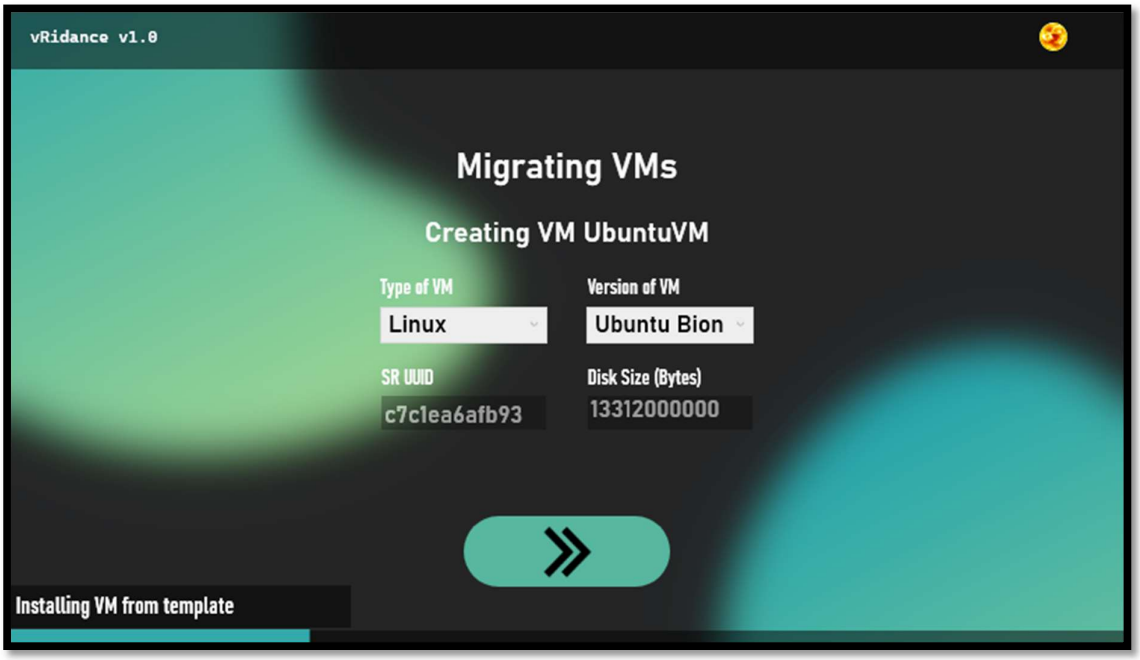
You can find this in the code below.

```
private void rectNext_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {
            string ip = txtIP.Text;
            string username = txtUsername.Text;
            string password = pwbPassword.Password.ToString();
            string curTheme = "";
            string thepath = this.path;
            if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
            else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";

            AuthenticationMethod method_checkCon = new PasswordAuthenticationMethod(username,
password);
            ConnectionInfo connection_checkCon = new ConnectionInfo(ip, username,
method_checkCon);
            var client_checkCon = new SshClient(connection_checkCon);

            try
            {
                client_checkCon.Connect();
                Migrate2Citrix migration2citrix = new Migrate2Citrix(curTheme, ip, username,
password, thepath);
                ((MainWindow)this.Owner).Content = migration2citrix.Content;
                migration2citrix.Owner = ((MainWindow)this.Owner);
                client_checkCon.Disconnect();
            }
            catch (Exception)
            {

                MessageBox.Show($"Unable to reach {ip}, or username and password is
incorrect.");
            }

        }
```

When pressing the Continue button, it will check the connection of the credentials you provided. This will in turn either succeed and go to the next screen, or it won't be able to connect and throw you the error saying the IP is either unreachable, or your username or password is incorrect. When going to the next screen, we'll forward the parameters ip, username, password, VMDK path and the current theme being light or dark.



On this screen, the migrations will be done. When running the code, it will ask you for the above input. This will be needed to trigger successful CLI commands over SSH. At the bottom of the screen we have 2 black boxes, one long and a shorter one. The shorter one tells us the status at which stage it is in the migration. The bottom long one is a simple progressbar. Below picture is how a running migration can look.




Below, I'll explain the code for this screen.

```
        private void grdMain4_LayoutUpdated(object sender, EventArgs e)
        {
            if (cbVersion.SelectedIndex >= 0 && txtDiskSize.Text != "" && txtDiskSize.Text != 
null && txtSrUuid.Text != "" && txtSrUuid.Text != null)
            {
                rectNext.Opacity = 1;
                rectNext.IsEnabled = true;
            }
        }
```

This code will check if the layout was updated. It'll check if you have a version selected, if disksize isn't empty or null and if your storage UUID isn't empty or null. If these all return true, it will enable the Continue button.

We only check on version because when we can only select a version when the OS type is selected, which is the type of VM input.

```
        private void cbType_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (cbType.SelectedIndex == 0)
            {
                List<string> versions = new List<string>();
                versions.Add("CentOS 7");
                versions.Add("CentOS 8");
                versions.Add("CoreOS");
                versions.Add("Debian Buster 10");
                versions.Add("Debian Jessie 8.0");
                versions.Add("Debian Stretch 9.0");
                versions.Add("NeoKylin Linux Server 7");
                versions.Add("Oracle Linux 7");
                versions.Add("Oracle Linux 8");
                versions.Add("Red Hat Enterprise Linux 7");
                versions.Add("Red Hat Enterprise Linux 8");
                versions.Add("Scientific Linux 7");
                versions.Add("SUSE Linux Enterprise 15 (64-bit)");
                versions.Add("SUSE Linux Enterprise Desktop 12 SP3 (64-bit)");
                versions.Add("SUSE Linux Enterprise Desktop 12 SP4 (64-bit)");
                versions.Add("SUSE Linux Enterprise Server 12 SP3 (64-bit)");
                versions.Add("SUSE Linux Enterprise Server 12 SP4 (64-bit)");
                versions.Add("SUSE Linux Enterprise Server 12 SP5 (64-bit)");
                versions.Add("Ubuntu Bionic Beaver 18.04");
                versions.Add("Ubuntu Focal Fossa 20.04");
                versions.Add("Ubuntu Xenial Xerus 160.4");
                cbVersion.ItemsSource = versions;
                cbVersion.IsEnabled = true;
            }
            else if (cbType.SelectedIndex == 1)
            {
                List<string> versions = new List<string>();
                versions.Add("Windows 8.1 (32-bit)");
                versions.Add("Windows 8.1 (64-bit)");
                versions.Add("Windows 10 (32-bit)");
                versions.Add("Windows 10 (64-bit)");
                versions.Add("Windows Server 2012 (64-bit)");
                versions.Add("Windows Server 2012 R2 (64-bit)");
                versions.Add("Windows Server 2016 (64-bit)");
                versions.Add("Windows Server 2019 (64-bit)");
                cbVersion.ItemsSource = versions;
                cbVersion.IsEnabled = true;
            }
        }
```

This will enable the version dropdown menu when either of these values were selected in the type dropdown. The versions displayed here are templates provided within Citrix XenCenter, so there is no need to enter CPU Cores or RAM upon migrating a new VM. This code knows which version was selected because of SelectedIndex 1 or 0. 1 being Windows and 0 being Linux machines.

When clicking on the Continue button, a lot of things will be going on behind the screen. I will elaborate with the following pieces of code.

When we initialized the page, we start a new thread called `Thread changeParamsThread`. This will call the ChangeParams function, which looks like this;

```
changeParamsThread = new Thread(changeParams);
```

```
public void changeParams()
        {
            if (folderPath != "" || folderPath != null)
            {
                string[] subdirectoryEntries = Directory.GetDirectories(folderPath);

                foreach (var var_subdirectory in subdirectoryEntries)
                {
                    vmCreated = false;
                    DirectoryInfo di = new DirectoryInfo(@"" + var_subdirectory);
                    string DirectoryName = di.Name;
                    this.Dispatcher.Invoke(() =>
                    {

                        this.Dispatcher.Invoke(() => { pbProgress.Value = 0; });
                        lblCurrVM.Content = $"Settings for {DirectoryName}";
                        this.Dispatcher.Invoke(() => { lblInfo.Content = ""; });
                        if (txtSrUuid.IsEnabled == false && txtDiskSize.IsEnabled == false)
                        {
                            txtDiskSize.IsEnabled = true;
                            txtSrUuid.IsEnabled = true;
                        }
                        rectNext.Opacity = 0.5;
                        rectNext.IsEnabled = false;
                        cbVersion.IsEnabled = false;
                        List<string> types = new List<string>();
                        types.Add("Linux");
                        types.Add("Microsoft Windows");
                        cbType.ItemsSource = types;
                    });

                    while (true)
                    {
                        Thread.Sleep(100);
                        if (nextIsClicked == true)
                        {
                            this.Dispatcher.Invoke(() =>
                            {
                                cbType.IsEnabled = false;
                                cbVersion.IsEnabled = false;
                                rectNext.IsEnabled = false;
                                rectNext.Opacity = 0.5;
                                txtDiskSize.IsEnabled = false;
                                txtSrUuid.IsEnabled = false;
                                lblCurrVM.Content = $"Creating VM {DirectoryName}";
                                lblInfo.Visibility = Visibility.Visible;
                            });
                            nextIsClicked = false;
                            createTheVMS(var_subdirectory.ToString());
                            while (true)
                            {
                                Thread.Sleep(100);
                                if (vmCreated == true) break;
                            }
                            this.Dispatcher.Invoke(() =>
                            {
                                cbType.ItemsSource = "";
                                cbVersion.ItemsSource = "";
                                cbType.IsEnabled = true;
                                cbVersion.IsEnabled = true;
                                rectNext.IsEnabled = true;
                                rectNext.Opacity = 1;
                                txtDiskSize.IsEnabled = true;
                                txtSrUuid.IsEnabled = true;
                            });
                            break;
```

```
                    }
                }
            }
        }

        this.Dispatcher.Invoke(() =>
        {
            rectClose.Visibility = Visibility.Visible;
            cbType.IsEnabled = false;
            cbVersion.IsEnabled = false;
            this.Dispatcher.Invoke(() => { pbProgress.Value = 0; });
            this.Dispatcher.Invoke(() => { lblInfo.Content = ""; });
            lblInfo.Visibility = Visibility.Hidden;

            string curTheme = "";
            if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
            else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";
            EndScreen endScreen = new EndScreen(curTheme);
            ((MainWindow)this.Owner).Content = endScreen.Content;

            endScreen.Owner = ((MainWindow)this.Owner);
        });
    }
```

You can see a lot of things are happening here. While this is a lot of code, the internals are pretty simple. When the thread is started it checks if the folderpath is not empty or null and saves the names of the child directories as strings in a string list. This will determine the name of the VM in XenCenter.

For each entry in the stringlist, it'll do the following:

- Sets the progressbar to 0
- Changes a label to display the current VM that is being worked on.
- Checks if Storage UUID input and Disk Size input are disabled
    - If they happen to be disabled, enable them so we can enter the desired values
- Disable the Continue button
- Add the versions that we had from the previous code

The code will then enter a while loop, this will continue running until all conditions are met. We will return to this code later down the document.

When the next button is clicked, the following code is executed;

```
        private void rectNext_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)
        {

            try
            {
                if (int.Parse(txtDiskSize.Text) > 0 && txtSrUuid.Text != "" || txtSrUuid.Text !=
null )
                {
                    MessageBox.Show("SR UUID or Disk Size was empty. Or the Disk Size value is
less or equal to 0");
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Something went wrong");
            }

            if (cbType.SelectedIndex == 0)
            {
                if (cbVersion.SelectedIndex == 0) os_type = "CentOS 7";
                else if (cbVersion.SelectedIndex == 1) os_type = "CentOS 8";
                else if (cbVersion.SelectedIndex == 2) os_type = "CoreOS";
                else if (cbVersion.SelectedIndex == 3) os_type = "Debian Buster 10";
                else if (cbVersion.SelectedIndex == 4) os_type = "Debian Jessie 8.0";
                else if (cbVersion.SelectedIndex == 5) os_type = "Debian Stretch 9.0";
                else if (cbVersion.SelectedIndex == 6) os_type = "NeoKylin Linux Server 7";
                else if (cbVersion.SelectedIndex == 7) os_type = "Oracle Linux 7";
                else if (cbVersion.SelectedIndex == 8) os_type = "Oracle Linux 8";
                else if (cbVersion.SelectedIndex == 9) os_type = "Red Hat Enterprise Linux 7";
                else if (cbVersion.SelectedIndex == 10) os_type = "Red Hat Enterprise Linux 8";
                else if (cbVersion.SelectedIndex == 11) os_type = "Scientific Linux 7";
                else if (cbVersion.SelectedIndex == 12) os_type = "SUSE Linux Enterprise 15 (64-
bit)";
                else if (cbVersion.SelectedIndex == 13) os_type = "SUSE Linux Enterprise Desktop
12 SP3 (64-bit)";
                else if (cbVersion.SelectedIndex == 14) os_type = "SUSE Linux Enterprise Desktop
12 SP4 (64-bit)";
                else if (cbVersion.SelectedIndex == 15) os_type = "SUSE Linux Enterprise Server
12 SP3 (64-bit)";
                else if (cbVersion.SelectedIndex == 16) os_type = "SUSE Linux Enterprise Server
12 SP4 (64-bit)";
                else if (cbVersion.SelectedIndex == 17) os_type = "SUSE Linux Enterprise Server
12 SP5 (64-bit)";
                else if (cbVersion.SelectedIndex == 18) os_type = "Ubuntu Bionic Beaver 18.04";
                else if (cbVersion.SelectedIndex == 19) os_type = "Ubuntu Focal Fossa 20.04";
                else if (cbVersion.SelectedIndex == 20) os_type = "Ubuntu Xenial Xerus 160.4";
            }
            else if (cbType.SelectedIndex == 1)
            {
                if (cbVersion.SelectedIndex == 0) os_type = "Windows 8.1 (32-bit)";
                else if (cbVersion.SelectedIndex == 1) os_type = "Windows 8.1 (64-bit)";
                else if (cbVersion.SelectedIndex == 2) os_type = "Windows 10 (32-bit)";
                else if (cbVersion.SelectedIndex == 3) os_type = "Windows 10 (64-bit)";
                else if (cbVersion.SelectedIndex == 4) os_type = "Windows Server 2012 (64-bit)";
                else if (cbVersion.SelectedIndex == 5) os_type = "Windows Server 2012 R2 (64-
bit)";
                else if (cbVersion.SelectedIndex == 6) os_type = "Windows Server 2016 (64-bit)";
                else if (cbVersion.SelectedIndex == 7) os_type = "Windows Server 2019 (64-bit)";
            }

            SrUuid = txtSrUuid.Text;
            DiskSize = txtDiskSize.Text;
            lblCurrVM.Visibility = Visibility.Visible;

            nextIsClicked = true;

        }
```

This will set a global variable os_type to the selected type from the dropdown menu.

These are linked to the XenCenter templates. The last line however is very important. Remember the code from above? This last line is what triggers the while loop to continue. Were setting nextIsClicked to true. This will in terms run the next part of the previous code;

```
while (true)
                    {
                        Thread.Sleep(100);
                        if (nextIsClicked == true)
                        {
                            this.Dispatcher.Invoke(() =>
                            {
                                cbType.IsEnabled = false;
                                cbVersion.IsEnabled = false;
                                rectNext.IsEnabled = false;
                                rectNext.Opacity = 0.5;
                                txtDiskSize.IsEnabled = false;
                                txtSrUuid.IsEnabled = false;
                                lblCurrVM.Content = $"Creating VM {DirectoryName}";
                                lblInfo.Visibility = Visibility.Visible;
                            });
                            nextIsClicked = false;
                            createTheVMS(var_subdirectory.ToString());
                            while (true)
                            {
                                Thread.Sleep(100);
                                if (vmCreated == true) break;
                            }
                            this.Dispatcher.Invoke(() =>
                            {
                                cbType.ItemsSource = "";
                                cbVersion.ItemsSource = "";
                                cbType.IsEnabled = true;
                                cbVersion.IsEnabled = true;
                                rectNext.IsEnabled = true;
                                rectNext.Opacity = 1;
                                txtDiskSize.IsEnabled = true;
                                txtSrUuid.IsEnabled = true;
                            });
                            break;
                        }
                    }
                }
            }

            this.Dispatcher.Invoke(() =>
            {
                rectClose.Visibility = Visibility.Visible;
                cbType.IsEnabled = false;
                cbVersion.IsEnabled = false;
                this.Dispatcher.Invoke(() => { pbProgress.Value = 0; });
                this.Dispatcher.Invoke(() => { lblInfo.Content = ""; });
                lblInfo.Visibility = Visibility.Hidden;

                string curTheme = "";
                if (rectDark.Visibility == Visibility.Hidden) curTheme = "dark";
                else if (rectDark.Visibility == Visibility.Visible) curTheme = "light";
                EndScreen endScreen = new EndScreen(curTheme);
                ((MainWindow)this.Owner).Content = endScreen.Content;

                endScreen.Owner = ((MainWindow)this.Owner);
            });
        }
```

It will disable most of the UI elements present on the screen and show you which VM is being migrated to XenCenter. It will enable the info label to display what is happening behind the screen and call the function createTheVMS which will create your selected VM in XenCenter.
You can find explained code in the page below.

```
        public async Task createTheVMS(string subdirectory)
        {
            string[] VMDirectory = Directory.GetFiles(@"" + subdirectory, "*-flat.vmdk");

            DirectoryInfo di = new DirectoryInfo(@"" + subdirectory);
            string DirectoryName = di.Name;

            if (os_type != null)
            {
                AuthenticationMethod              method_citrix              =              new
PasswordAuthenticationMethod(citrix_username, citrix_password);
                ConnectionInfo     connection_citrix     =     new     ConnectionInfo(citrix_host,
citrix_username, method_citrix);
                var client = new SshClient(connection_citrix);
                Thread.Sleep(100);

                if (os_type.Contains("Windows"))
                {
                    this.Dispatcher.Invoke(() => { pbProgress.Value = 0; });
                    Debug.WriteLine("Connecting to host...");
                    this.Dispatcher.Invoke(()  =>  {  lblInfo.Content  =  "Connecting  to  remote
host"; });
                    client.Connect();
                    Debug.WriteLine("Connected!");
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Connected!"; });
                    Debug.WriteLine($"Creating VM {DirectoryName}, Please Wait...");
                    this.Dispatcher.Invoke(()  =>  {  lblInfo.Content  =  "Installing  VM  from
template"; });
                    string   createVM   =   $"xe   vm-install   template=\"{os_type}\"   new-name-
label=\"{DirectoryName}\" {SrUuid}";
                    for  (int  i  =  0;  i  <=  9;  i++)  {  this.Dispatcher.Invoke(()  =>  {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(createVM); var createVMResult = client.RunCommand(createVM);
                    string vmUuid = createVMResult.Result.TrimEnd();
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Creating virtual disk";
});
                    string    createVdi    =    $"xe    vdi-create    sr-uuid=\"{SrUuid}\"    name-
label=\"{DirectoryName + "_Disk"}\" virtual-size={DiskSize}";
                    for  (int  i  =  9;  i  <=  27;  i++)  {  this.Dispatcher.Invoke(()  =>  {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(createVdi);          var          createVdiResult          =
client.RunCommand(createVdi);
                    string vdiUuid = createVdiResult.Result.TrimEnd();
                    Debug.WriteLine(vdiUuid);
                    string importResult = "";
                    foreach (var file in VMDirectory)
                    {
                        FileInfo fi = new FileInfo(@"" + file);
                        string fileName = fi.Name;
                        for  (int  i  =  27;  i  <=  36;  i++)  {  this.Dispatcher.Invoke(()  =>  {
pbProgress.Value = i; }); await Task.Delay(50); }
                        this.Dispatcher.Invoke(() => { lblInfo.Content = "Importing VMDK, this
may take a while"; });
                        string      vmdkImport      =      $"xe      vdi-import      uuid={vdiUuid}
filename=/mnt/VMDKShare/{DirectoryName}/{fileName} format=raw --progress";
                        Debug.WriteLine(vmdkImport);       var       createImportResult       =
client.RunCommand(vmdkImport);
                        importResult = createImportResult.Result;
                        Debug.WriteLine("Importing file, please wait.");

                    }
                    string  getVmVDI  =  $"xe  vm-disk-list  vm={vmUuid}  |  grep  'uuid  (  RO)
:' | sed 's/^.*: //' | sed -n 2p";
                    for  (int  i  =  36;  i  <=  45;  i++)  {  this.Dispatcher.Invoke(()  =>  {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(getVmVDI);       var       createExistingVDIResult       =
client.RunCommand(getVmVDI);
                    List<string>                          templateVDI                          =
createExistingVDIResult.Result.ToCharArray().Select(c => c.ToString()).ToList();
```

```
                    string newLine = "";
                    foreach (string character in templateVDI)
                    {
                        newLine = newLine.Insert(newLine.Length, character.ToString());
                    }
                    string removeTemplateDisk = $"xe vdi-destroy uuid={newLine}";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Removing template disk";
});
                    for (int i = 45; i <= 54; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(removeTemplateDisk);        var        removedDisk        =
client.RunCommand(removeTemplateDisk);
                    string createVdb = $"xe vbd-create vm-uuid={vmUuid} device=0 vdi-
uuid={vdiUuid} bootable=true mode=RW type=Disk";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Creating VDB"; });
                    for (int i = 45; i <= 59; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(createVdb);          var          createVdbResult          =
client.RunCommand(createVdb);
                    string vdbUuid = createVdbResult.Result.TrimEnd();
                    Debug.WriteLine(vdbUuid);
                    string startVm = $"xe vm-start uuid={vmUuid}";
                    for (int i = 59; i <= 63; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(startVm); var vmStartedResult = client.RunCommand(startVm);
                    string attachDisk = $"xe vbd-plug uuid={vdbUuid} device=0 vdi-
uuid={vdiUuid}";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Attaching Disk to VM"; });
                    for (int i = 63; i <= 72; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(attachDisk);          var          attachDiskResult          =
client.RunCommand(attachDisk);
                    string stopVm = $"xe vm-shutdown uuid={vmUuid}";
                    for (int i = 72; i <= 81; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(stopVm); var stopVmResult = client.RunCommand(stopVm);
                    string changeBootOption = $"xe vm-param-set uuid={vmUuid} HVM-boot-
params:firmware=uefi";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Changing boot parameter to
UEFI"; });
                    for (int i = 81; i <= 90; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(changeBootOption);      var      changeBootOptionResult      =
client.RunCommand(changeBootOption);
                    string     changeSecureBoot     =     $"xe     vm-param-set     uuid={vmUuid}
platform:secureboot=false ";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Disabling secure boot";
});
                    for (int i = 90; i <= 100; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(changeSecureBoot);      var      changeSecureBootResult      =
client.RunCommand(changeSecureBoot);
                    Debug.WriteLine(startVm); client.RunCommand(startVm);
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Done!"; });
                    client.Disconnect();
                    vmCreated = true;
                    Thread.Sleep(50);

                }
                else    if    (os_type.Contains("CentOS")    ||    os_type.Contains("CoreOS")    ||
os_type.Contains("Debian")  ||  os_type.Contains("Linux")  ||  os_type.Contains("Red  Hat")  ||
os_type.Contains("Ubuntu") || os_type.Contains("SUSE"))
                {
                    this.Dispatcher.Invoke(() => { pbProgress.Value = 0; });
                    Debug.WriteLine("Connecting to host...");
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Connecting to remote
host"; });
```

```
                    client.Connect();
                    Debug.WriteLine("Connected!");
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Connected!"; });

                    Debug.WriteLine($"Creating VM {DirectoryName}, Please Wait...");
                    string createVM = $"xe vm-install template=\"{os_type}\" new-name-
label=\"{DirectoryName}\" {SrUuid}";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Installing VM from
template"; });
                    for (int i = 0; i <= 27; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(createVM); var createVMResult = client.RunCommand(createVM);
                    string vmUuid = createVMResult.Result.TrimEnd();
                    string createVdi = $"xe vdi-create sr-uuid=\"{SrUuid}\" name-
label=\"{DirectoryName + "_Disk"}\" virtual-size={DiskSize}";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Creating Virtual Disk";
});
                    for (int i = 27; i <= 36; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(createVdi); var createVdiResult =
client.RunCommand(createVdi);
                    string vdiUuid = createVdiResult.Result.TrimEnd();
                    Debug.WriteLine(vdiUuid);
                    string importResult = "";
                    foreach (var file in VMDirectory)
                    {
                        FileInfo fi = new FileInfo(@"" + file);
                        string fileName = fi.Name;
                        for (int i = 36; i <= 45; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                        this.Dispatcher.Invoke(() => { lblInfo.Content = "Importing VMDK, this
may take a while"; });
                        string vmdkImport = $"xe vdi-import uuid={vdiUuid}
filename=/mnt/VMDKShare/{DirectoryName}/{fileName} format=raw --progress";
                        Debug.WriteLine(vmdkImport); var createImportResult =
client.RunCommand(vmdkImport);
                        importResult = createImportResult.Result;
                        Debug.WriteLine("Importing file, please wait.");

                    }
                    string getVmVDI = $"xe vm-disk-list vm={vmUuid} | grep 'uuid ( RO)
:' | sed 's/^.*: //' | sed -n 2p";
                    for (int i = 45; i <= 54; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(getVmVDI); var createExistingVDIResult =
client.RunCommand(getVmVDI);
                    List<string> templateVDI =
createExistingVDIResult.Result.ToCharArray().Select(c => c.ToString()).ToList();
                    string newLine = "";
                    foreach (string character in templateVDI)
                    {
                        newLine = newLine.Insert(newLine.Length, character.ToString());
                    }
                    string removeTemplateDisk = $"xe vdi-destroy uuid={newLine}";
                    for (int i = 54; i <= 63; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(removeTemplateDisk); var removedDisk =
client.RunCommand(removeTemplateDisk);
                    string createVdb = $"xe vbd-create vm-uuid={vmUuid} device=0 vdi-
uuid={vdiUuid} bootable=true mode=RW type=Disk";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Creating VDB"; });
                    for (int i = 63; i <= 72; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(createVdb); var createVdbResult =
client.RunCommand(createVdb);
                    string vdbUuid = createVdbResult.Result.TrimEnd();
                    Debug.WriteLine(vdbUuid);
                    string startVm = $"xe vm-start uuid={vmUuid}";
                    Debug.WriteLine(startVm); var vmStartedResult = client.RunCommand(startVm);
```

```
                    string attachDisk = $"xe vbd-plug uuid={vdbUuid} device=0 vdi-
uuid={vdiUuid}";
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Attaching Virtual Disk to
VM"; });
                    for (int i = 72; i <= 100; i++) { this.Dispatcher.Invoke(() => {
pbProgress.Value = i; }); await Task.Delay(50); }
                    Debug.WriteLine(attachDisk);          var          attachDiskResult          =
client.RunCommand(attachDisk);
                    this.Dispatcher.Invoke(() => { lblInfo.Content = "Done!"; });
                    client.Disconnect();
                    vmCreated = true;
                    Thread.Sleep(50);

                }
            }

        }
```

This looks like a whole spaghetti of baby language. Don't worry, most of this code is Xen CLI commands executed in a SSH connection and the progressbar being updated. The only part that's important to know is that for each SSH command that gets executed, the result is being saved in a variable which in terms is being used on other commands. The second last line of code is vmCreated = true, this is important for the code to know when a VM has been created, this will go back to the loop in changeParams and reload the loop for the second VM. Along with some small UI updates, this is all this code does.

After the full loop is created i.e. all the child directories have been looped through, the code will execute the next and final screen telling the user migrations have been successful.

```
                EndScreen endScreen = new EndScreen(curTheme);
                ((MainWindow)this.Owner).Content = endScreen.Content;

                endScreen.Owner = ((MainWindow)this.Owner);
```

```
                        this.Dispatcher.Invoke(() =>
                        {
                            cbType.IsEnabled = false;
                            cbVersion.IsEnabled = false;
                            rectNext.IsEnabled = false;
                            rectNext.Opacity = 0.5;
                            txtDiskSize.IsEnabled = false;
                            txtSrUuid.IsEnabled = false;
                            lblCurrVM.Content = $"Creating VM {DirectoryName}";
                            lblInfo.Visibility = Visibility.Visible;
                        });
```

Important note: When doing UI updates, it is important to use the below line of code, where you will nest your UI update in.
For example;

```
for (int i = 0; i <= 9; i++) { this.Dispatcher.Invoke(() => { pbProgress.Value = i; }); await
Task.Delay(50);
```

This is because were working with multiple threads, when the main thread (changeParams) is running, a UI update can't take place on the same thread, since it is still "stuck" in the changeParams while loop. This will not break anything but just won't update your UI.

# 5 Sources

## 5.1 Proxmox

"Virtualize Windows 10 with Proxmox VE - Techno Tim." https://www.youtube.com/watch?v=6c-6xBkD2J4 (accessed Jan. 14, 2022).

"Trying Out Proxmox VE, an open source virtualization platform – Micheal Niehaus." https://oofhours.com/2021/12/08/trying-out-proxmox-ve-an-open-source-virtualization-platform/ (accessed Jan. 17, 2022).

"Migrate VMWare ESXi Virtual Machines to Proxmox KVM with LVM-Thin Logical Volumes – Martin Seener." https://www.sysorchestra.com/migrate-vmware-esxi-virtual-machines-to-proxmox-kvm-with-lvm-thin-logical-volumes/ (accessed Jan. 17, 2022).

"Migration of servers to Proxmox VE - Proxmox VE." https://pve.proxmox.com/wiki/Migration_of_servers_to_Proxmox_VE#VMware_to_Proxmox_VE_.28KVM.29 (accessed Jan. 17, 2021).

"How to import vmdk storage to local-lvm storage - Proxmox VE." https://forum.proxmox.com/threads/how-to-import-vmdk-storage-to-local-lvm-storage.45007/ (accessed Jan. 17, 2021).
"How to Configure Nested

Virtualization In Proxmox (Step-by-Step) - Koroma Tech." https://www.youtube.com/watch?v=HE6ZtXcnlwo (accessed Jan. 17, 2021).
"VMWare to Proxmox – Alfio Munoz." https://www.youtube.com/watch?v=DZG2P36nlcY (accessed Jan. 18, 2021).

"Proxmox resize pve root - Laineantti ." https://gist.github.com/laineantti/4fc29acbbd25593619a76b413e42b78f (accessed Jan. 20, 2021).
"qm(1) – Proxmox pve ." https://pve.proxmox.com/pve-docs/qm.1.html (accessed Jan. 21, 2021).

## 5.2 Azure Stack HCI

[1]

"Azure Stack HCI – Hyperconverged Infrastructure | Microsoft Azure." https://azure.microsoft.com/en-us/products/azure-stack/hci/ (accessed Nov. 10, 2021).

[2]

JasonGerend, "Deploy a cloud witness for a Failover Cluster." https://docs.microsoft.com/en-us/windows-server/failover-clustering/deploy-cloud-witness (accessed Jan. 31, 2022).

JasonGerend, “Add-ClusterVirtualMachineRole (FailoverClusters).” https://docs.microsoft.com/en-us/powershell/module/failoverclusters/add-clustervirtualmachinerole (accessed Jan. 31, 2022).

JasonGerend, “Add-VMHardDiskDrive (Hyper-V).” https://docs.microsoft.com/en-us/powershell/module/hyper-v/add-vmharddiskdrive (accessed Jan. 31, 2022).

JasonGerend, “Hyper-V Module.” https://docs.microsoft.com/en-us/powershell/module/hyper-v/ (accessed Jan. 31, 2022).

JasonGerend, “Start-VM (Hyper-V).” https://docs.microsoft.com/en-us/powershell/module/hyper-v/start-vm (accessed Jan. 31, 2022).

## 5.3 Citrix

“‘[Xen-users] Re: [XCP] Error when trying to import vmware images’ - MARC.” https://marc.info/?l=xen-users&m=132075439700731&w=2 (accessed Feb. 01, 2022).

B. Khatri, “Add VirtualDisk on Xen VM using XE command line,” bidhankhatri.com.np, Jan. 25, 2018. https://www.bidhankhatri.com.np/system/Add-VirtualDisk-on-Xen-VM-using-XE-command-line/ (accessed Feb. 01, 2022).

“After importing of an OVF package from a VMware environment Virtual Machine Fails to Boot with STOP 7B or ‘Device Does Not Exist.’” https://support.citrix.com/article/CTX124925 (accessed Feb. 01, 2022).

“Command-line interface | Citrix Hypervisor 8.2.” https://docs.citrix.com/en-us/citrix-hypervisor/command-line-interface.html#installing-the-xe-cli (accessed Feb. 01, 2022).

“How to Convert VMware Virtual Machines to XenServer Virtual Machines.” https://support.citrix.com/article/CTX116603 (accessed Feb. 01, 2022).

L. Rendek, “How to create a new virtual machine on XenServer using command line,” Linux Tutorials - Learn Linux Configuration. https://linuxconfig.org/how-to-create-a-new-virtual-machine-on-xenserver-using-command-line (accessed Feb. 01, 2022).

“Problems importing OVF/VMDK from vSphere to Xencenter,” Discussions. https://discussions.citrix.com/topic/415299-problems-importing-ovfvmdk-from-vsphere-to-xencenter/ (accessed Feb. 01, 2022).

“Quick start | Citrix Hypervisor 8.2.” https://docs.citrix.com/en-us/citrix-hypervisor/quick-start.html (accessed Feb. 01, 2022).

“VHD import by command,” Discussions. https://discussions.citrix.com/topic/394782-vhd-import-by-command/ (accessed Feb. 01, 2022).

FKIT, Xenserver - Add a Local ISO repository, (2015). Accessed: Feb. 01, 2022. [Online]. Available: https://www.youtube.com/watch?v=qb9mI4RjQR4