Tuomas Jokinen

Automating Virtual Machine Management in VMware vSphere

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

1 March 2021

Abstract

Author:	Tuomas Jokinen
Title:	Automating Virtual Machine Management in VMware vSphere
Number of Pages:	23 pages
Date:	1 March 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	IoT & Cloud Computing
Supervisors:	Janne Salonen, Head of School (ICT)

The goal of this thesis was to develop an application for Neste Engineering Solutions Oy that would automate power state management of virtual machines. The application had to be created in a manner that would allow it to be integrated in a different software product of the company.

The application was written using the C# programming language, using VMware vSphere Automation API to interact with the virtualization environment. The targeted version of the VMware software was 6.5U3.

As the application's code was planned to be integrated into another piece of software from the very start, the project as shown in the thesis is more of a proof of concept. A small-scale development environment was created using disused computer hardware, with the entire VMware vSphere infrastructure converged onto a single computer.

The development environment consisted of a single computer. Due to the nature of using consumer hardware as the platform, some unusual steps had to be taken as the ESXi installation image was modified for support of the integrated networking card of the motherboard, and the vSphere installer's requirement for DNS server was bypassed.

The resulting C# application was able to be perform the required automation actions of powering on and powering off the specified virtual machines. This code was then taken and integrated into another piece of software internally. This created benefits in both reduced hardware utilization and saving system administrators time as the virtual machines in the virtualization platform would no longer remain idle, unnecessarily consuming resources.

Keywords: automation, API, virtualization, VMware, vSphere, C#

Tiivistelmä

Tekijä:	Tuomas Jokinen
Otsikko:	Virtuaalikoneiden hallinnan automointi VMware vSpheressä
Sivumäärä:	23 sivua
Aika:	1.3.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Verkot ja pilvipalvelut
Ohjaajat:	Osaamisaluepäällikkö Janne Salonen

Insinöörityön tavoitteena oli kehittää VMware vSphere ympäristön virtuaalikoneiden virranhallinnan automatisoiva ohjelma Neste Engineering Solutions Oy:lle. Ohjelma tuli luoda tavalla, jolla sen koodi ja toiminnallisuus voitaisiin hyödyntää osana toista yrityksessä kehitettyä ohjelmistoa.

Ohjelmisto kirjoitettiin käyttäen C#-ohjelmointikieltä. hyödyntäen VMwaren vSphere Automation rajapintaa virtualisointijärjestelmän ja sen sisältämien virtuaalikoneiden käskyttämiseen. Ohjelmistokehityksen kohteena oli VMware ympäristön versio 6.5U3.

Ohjelmiston koodin lopullisen käyttötarkoituksen ollessa hyödyntäminen osana toista laajempaa yrityksen kehittämää ohjelmistoa, tässä opinnäytetyössä esiteltyä tekstipohjaista käyttöliittymää käytetään vain ohjelman toimivuuden havainnollistamiseen. Projektin toteutusta varten luotiin pieni kehitysympäristö hyödyntäen käytöstä poistettua tietokonetta, joka toimi koko VMware vSphere ympäristön palvelimena.

Projektin aikana kohdattiin rajoituksia ja ongelmia, jotka johtuivat siitä, että ympäristö koostui vain yhdestä tietokoneesta. Tavanomaisen kuluttajille tarkoitettujen tietokonekomponenttien käyttö aiheutti ongelmia ESXin asennusprosessissa, jonka takia ESXin asennusmediaa jouduttiin muuttamaan tietokoneen emolevyn verkkokortille sopivaksi. Toinen poikkeus oli vSpheren asennusohjelman DNS-vaatimuksen ohittaminen nimeämällä virtuaalikone sen IP-osoitteella.

Kehitetty C# ohjelmisto pystyi suorittamaan vaaditut automaatiotehtävät vSphere ympäristössä ja myös onnistuneesti hyödynnettiin osana toista ohjelmistoa. Ohjelmisto paransi ympäristön käyttötehokkuutta vähentämällä tyhjäkäynnillä olevien virtuaalikoneiden määrää, säästäen palvelinten resursseja.

Avainsanat: automaatio, API, virtualisointi, VMware, vSphere, C#

Contents

List of Abbreviations

SDK: Software Development Kit. Set of software tools for making a program targeting a particular vendor’s hardware or software.

IDE: Integrated Development Environment. Software package with graphical user interface for editing source code, automation of building software from said source code, and debugger for testing the built piece of software.

1 Introduction

This thesis describes the procedure of automating VMware vSphere virtual machine control with VMware vSphere Automation SDK for .NET. The thesis work was commissioned by Neste Engineering Solutions Oy. The need of automated virtual machine power state control appeared during planning of the future scale of a virtualized environment. It was considered that the current state of having virtual machines always powered on in case they were needed was very wasteful and could cause unreasonable increase in use of hardware resources by idle virtual machines. To counter this, it would be better to incorporate the ability to change virtual machine power states as the services of the machine were required.

The goal of the thesis work was to create a solution that would be integrated into another software, that would communicate with the VMware vSphere environment, powering on virtual machines when they were needed and power them off after they are no longer in use, drastically reducing the idle load in the virtual environment. The application of this thesis was developed using the C# programming language, as it would be integrated into another software that was written in the same language. The thesis work uses VMware's vSphere Automation API for automating the virtual machine power state control.

2 Virtualization

In the context of computing, the term virtualization refers to use of software to create an abstraction layer above the physical resources of a server, commonly referred to as a "host", in order to facilitate use of the resources by virtual machines, commonly referred to as "guests". The software that handles the abstraction of the server resources is called a hypervisor. Hypervisors are commonly divided into two groups, type 1 and type 2. Since VMware ESXi is a type 1 hypervisor, type 2 hypervisor is not described here as it falls outside the scope of this project. (1, 23)

A type 1 hypervisor runs on the physical server hardware without an operating system in between. Another term used to describe this type of hypervisor is “bare metal”, as the hypervisor directly interfaces with the hardware. The benefits of a type 1 hypervisor include performance, as the hypervisor is a specially developed software that has less overhead than that of a server operating system. Type 1 hypervisors isolate the virtual machines, so that they can only interact with their own resources. This improves the resiliency of the system, meaning that in cases where a virtual machine would fail, it would not be able to cause issues to other virtual machines hosted on the same server. (1, 23-24)

2.1 VMware vSphere

VMware vSphere is a suite of virtualization software from VMware corporation. The relevant programs in the suite for this project are the VMware ESXi hypervisor and VMware vCenter Server management interface. (2, 9) VMware ESXi is a hypervisor product from VMware that that abstracts the hardware on which the virtual machines run. Through its abstraction layer it provides access to the CPU, RAM and storage of the system as a unified pool that can be allocated to the virtual machines hosted on the system. ESXi provides a web-based management interface that is less featured than that of VMware vCenter Server. For this project, use of VMware vCenter Server is required, as it provides additional services and hosts the APIs that are used in this project. In this project, VMware vCenter is deployed as a dedicated virtual machine hosted on the ESXi server, called vCenter Server Appliance, or vCSA for short. (2, 125)

3 Preparing the development environment

The development environment of this project consists of a computer hosting the vSphere environment and a second computer hosting the IDE and software used to automate VMware vSphere.

3.1 Installing VMware vSphere

The first step of setting up the development environment is installing the VMware ESXi hypervisor and VMware vSphere management interface. Installers for both software can be downloaded from My VMware portal. The version of ESXi and vSphere used in this work is 6.5.0U3. As VMware ESXi is a hypervisor targeting server hardware, some additional steps need to be taken in order to install the software on consumer hardware.

Table 1. Hardware of the host computer.

Component	Specifications
CPU	Intel Core i7-3770K
RAM	16GB 2400MHz DDR3
Motherboard	Asus P8Z77-V LX
Storage	Crucial BX500

The motherboard used in the computer features an onboard network interface card featuring Realtek 8111 chipset which is not supported by the drivers included in the ESXi installation image. In order to proceed with the installation, it is necessary to create a customized installation image.

3.1.1 Creating a customized installation image

In order to create the customized image, the offline bundle for ESXi 6.5.0U3 was downloaded from My VMware. The network driver is unofficial, created by Dmitry Nechaev and downloaded from V-Front Online Depot. The new image is created using VMware's PowerCLI PowerShell module.

To start, the bundles are added to a new software depot.

```
Add-EsxSoftwareDepot "C:\Users\tjokinen\Downloads\net55-r8168-8.045a-napi-
offline_bundle.zip", "C:\users\tjokinen\Downloads\update-from-esxi6.5-
6.5_update03.zip"
```

Listing 1. PowerCLI command to add the bundles to a new software depot.

Then, a new image profile must be created. The easiest way is by copying an existing one.

```
New-EsxImageProfile -CloneProfile ESXi-6.5.0-20190702001-standard -Name ESXi-
6.5.0-20190702001-RTL8111 -Vendor Tjokinen
```

Listing 2. New image profile created by cloning an existing one.

Following this, the acceptance level must be changed to community supported. This is to allow Realtek driver to function, as it is provided by a third party. Then the driver can be added to the image profile.

```
Set-EsxImageProfile -ImageProfile ESXi-6.5.0-20190702001-RTL8111 -
AcceptanceLevel CommunitySupported
Add-EsxSoftwarePackage -ImageProfile ESXi-6.5.0-20190702001-RTL8111 -
SoftwarePackage net55-r8168
```

Listing 3. Changing acceptance level and adding the driver to the image profile.

Finally, the new image can be exported.

```
Export-EsxImageProfile -ImageProfile ESXi-6.5.0-20190702001-RTL8111 -
ExportToIso -FilePath "C:\Users\tjokinen\Desktop\ESXi-6.5.0-20190702001-
RTL8111.iso"
```

Listing 4. Exporting the image as an ISO file.

3.1.2 Installing VMware ESXi

In order to install VMware ESXi, a bootable USB drive of the image was created using Rufus.

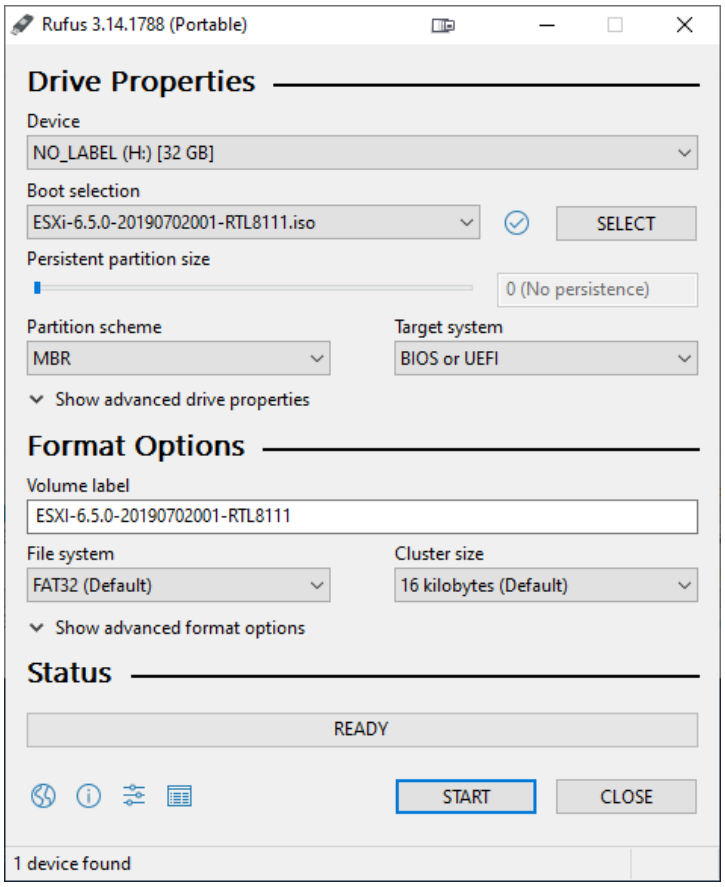

Figure 1. Creating the bootable USB drive.

The bootable USB drive was then connected to a USB port on the computer and booted into via the computer's BIOS. While ESXi supports automated installation, in this case the installation was performed via the interactive installer which requires user input.

The ESXi installer is started by booting the computer into the BIOS. From there the USB drive was selected as the boot device. Following this the computer automatically boots into the installer. After accepting the EULA, the installer proceeds to scan the system for compatible storage devices that the software can be installed on. In this case the hypervisor is installed on an SSD installed in the computer. The installer prompts for a choice of keyboard layout, in this case the US default layout is used. Following this the installer prompts for a root password which is written twice, after which the user is asked to confirm their choice of drive as it will be repartitioned. After confirming the drive selected by

pressing the F11 key the installation begins. After the installation is complete the user is prompted to remove the installation media and to reboot the server.

The web management interface of the ESXi hypervisor is not yet available directly after the installation. The server needs to have an IP address set to be able to access the hypervisor from another computer and to install the vSphere server. By default, the hypervisor is set to get an IP address over DHCP, which was not set up in the development environment. Instead, the network settings were changed to a static IP address.

3.1.3 Installing VMware vSphere Server

The vSphere Server installer comes as a separate image file that can be mounted on a computer in order to access the installer. As the development environment consists of a single ESXi host machine, vSphere Server will be installed with embedded platform services controller. The vSphere installer effectively creates a new Linux virtual machine on the host and configures it according to details entered in the installer.

The vSphere installer image is mounted and run on a remote computer which connects to the ESXi server to perform the vSphere server installation. After launching the installer and accepting the terms of the license agreement, the deployment type is selected.

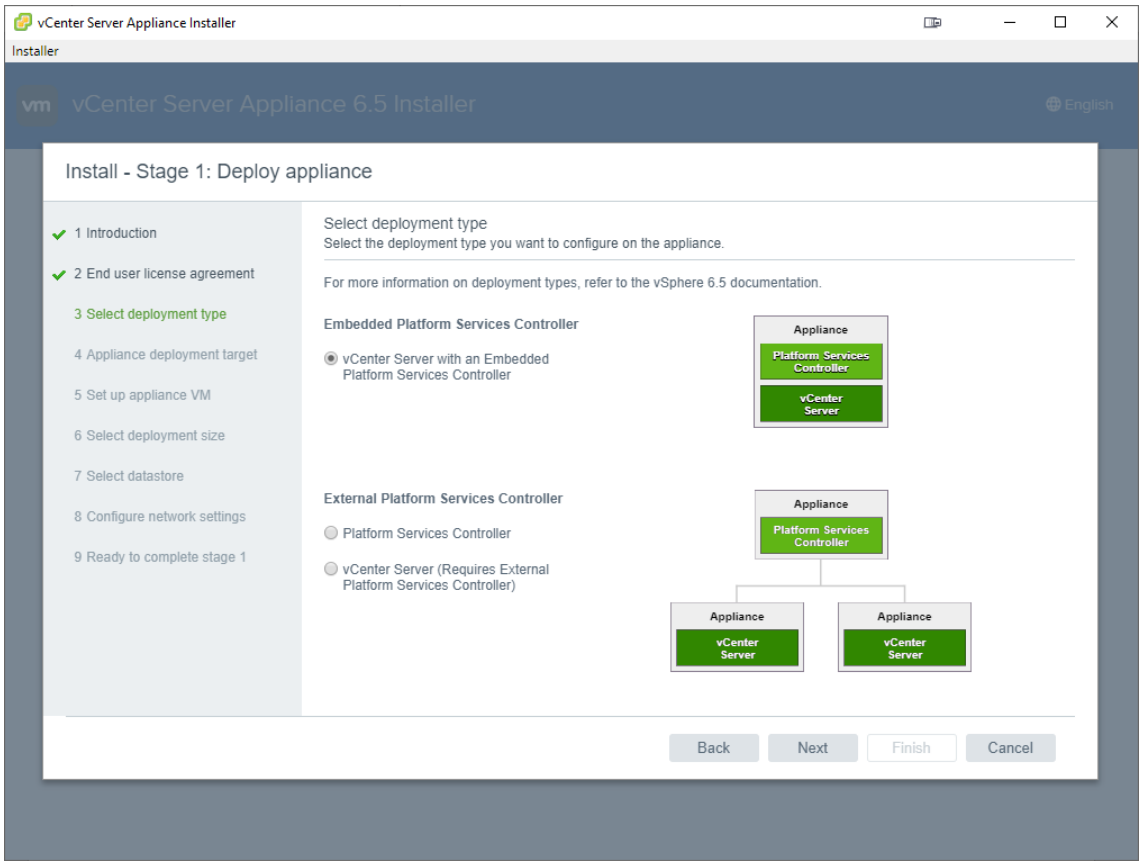

Figure 2. Selecting deployment type.

As the deployment is very simple, with all of the infrastructure converged onto one server, the option of Embedded Platform Services Controller is selected.

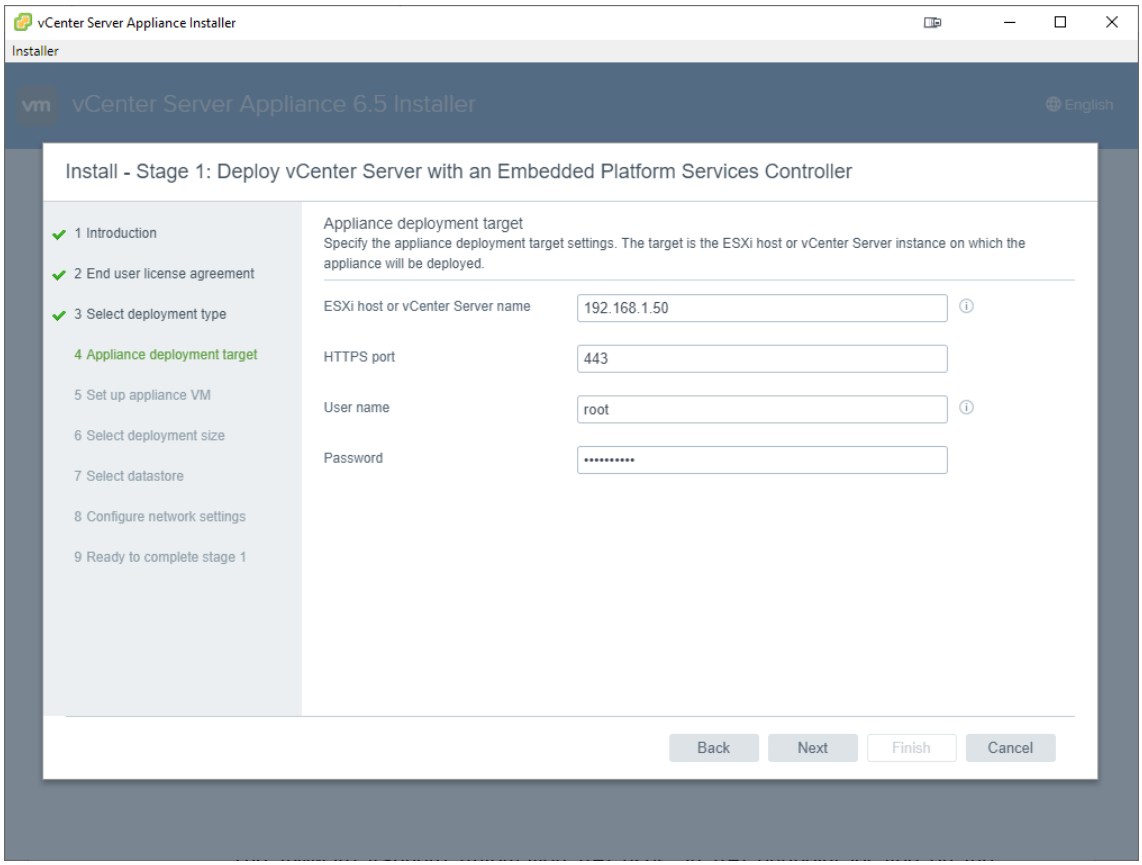

Figure 3. Deployment target settings

The settings for connecting to the target ESXi server are given next. The server is logged into as root, as there is no domain configured yet. The installer warns about the SSL certificate at this point, as it has not been saved in the trusted certificate storage of the computer on which the installation is performed. The certificate is accepted to progress the installation.

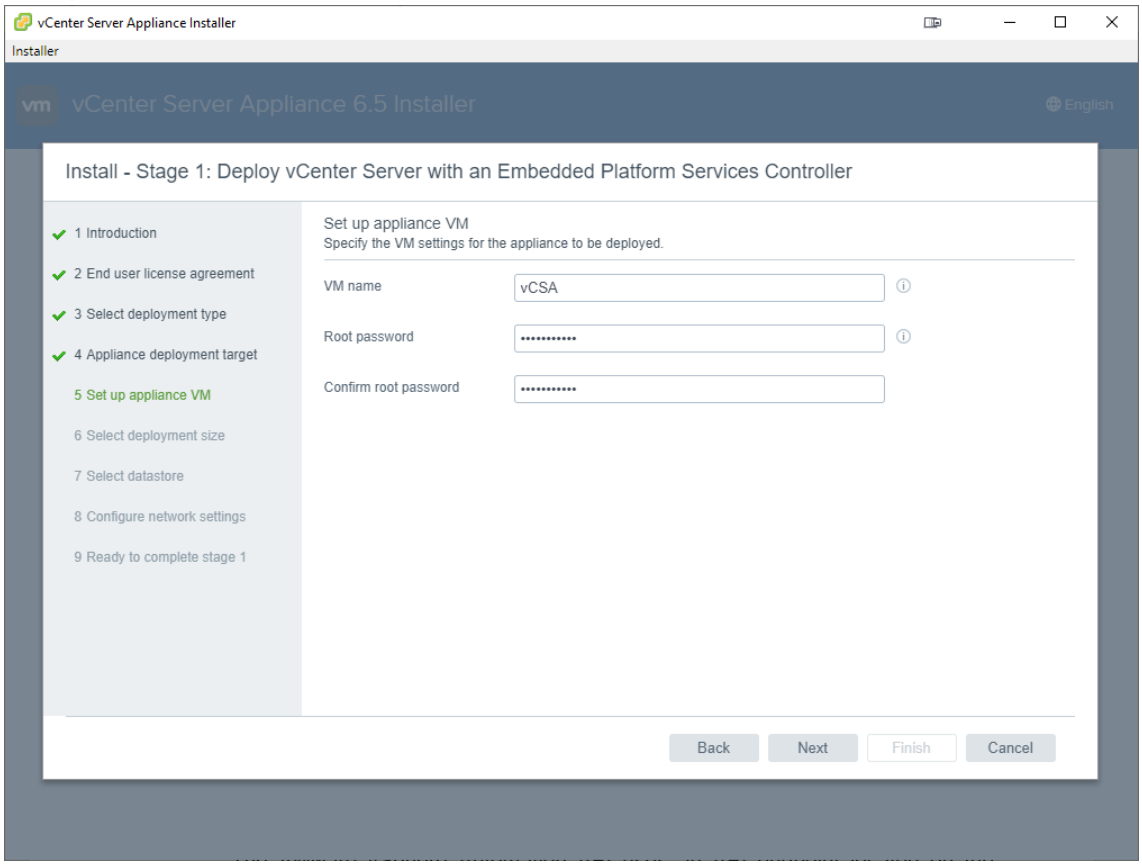

Figure 4. Set up appliance VM

The installer is given the name which the virtual machine acting as the vCenter Server appliance will have. The root password for the appliance is set also. This is separate from the credentials used to login to the vCenter web client.

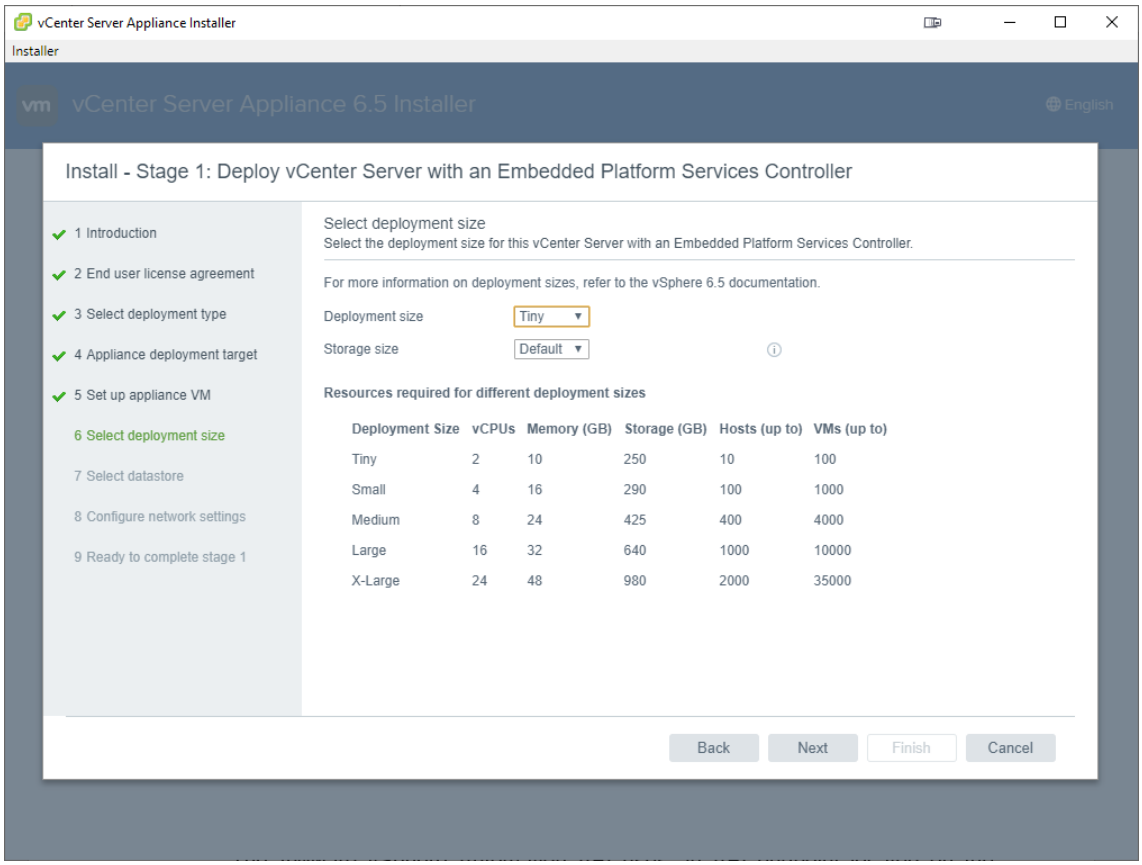

Figure 5. Deployment size

The deployment size is set to tiny, as the development environment is temporary and contains only a single host and a small number of virtual machines to demonstrate functionality of the application created in this project.

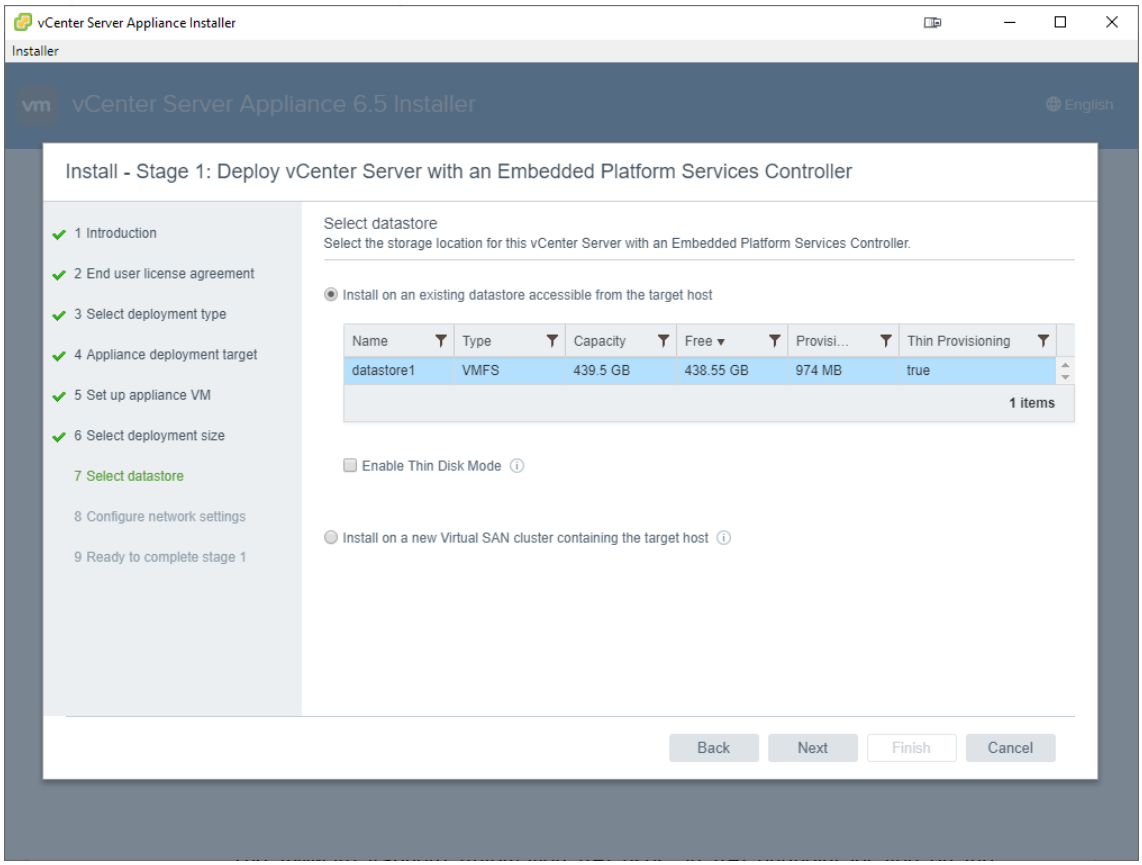

Figure 6. Datastore selection.

The datastore which will contain the vCenter server is selected.

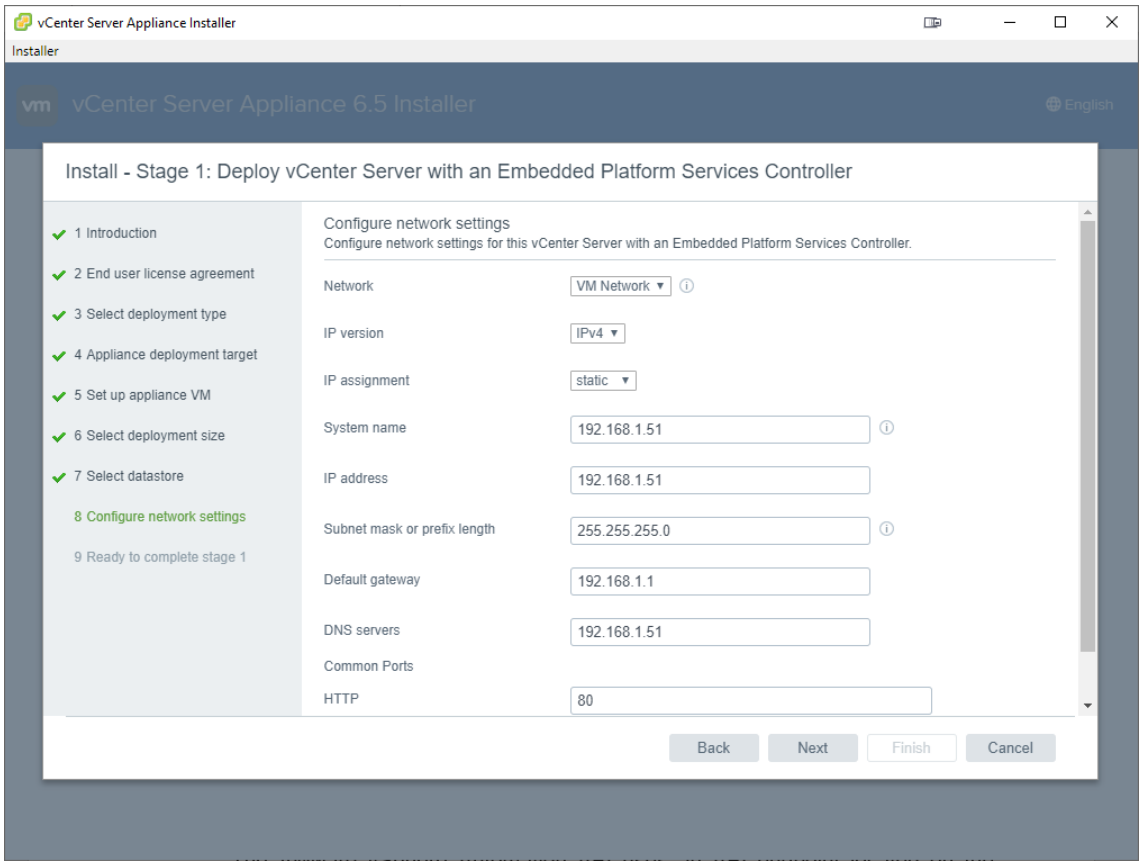

Figure 7. Configure network settings

Network settings for the vCenter server appliance are configured. To bypass the installer's requirement for a DNS server, the server appliance had the system name set to its IP address. The HTTP and HTTPS ports are left as default. Following this the installer gives a summary of settings configured thus far. Upon clicking the finish button, the installer proceeds to deploy the appliance to the ESXi server. After the installer has successfully deployed the appliance, the installer moves to the next phase where the appliance itself is configured.

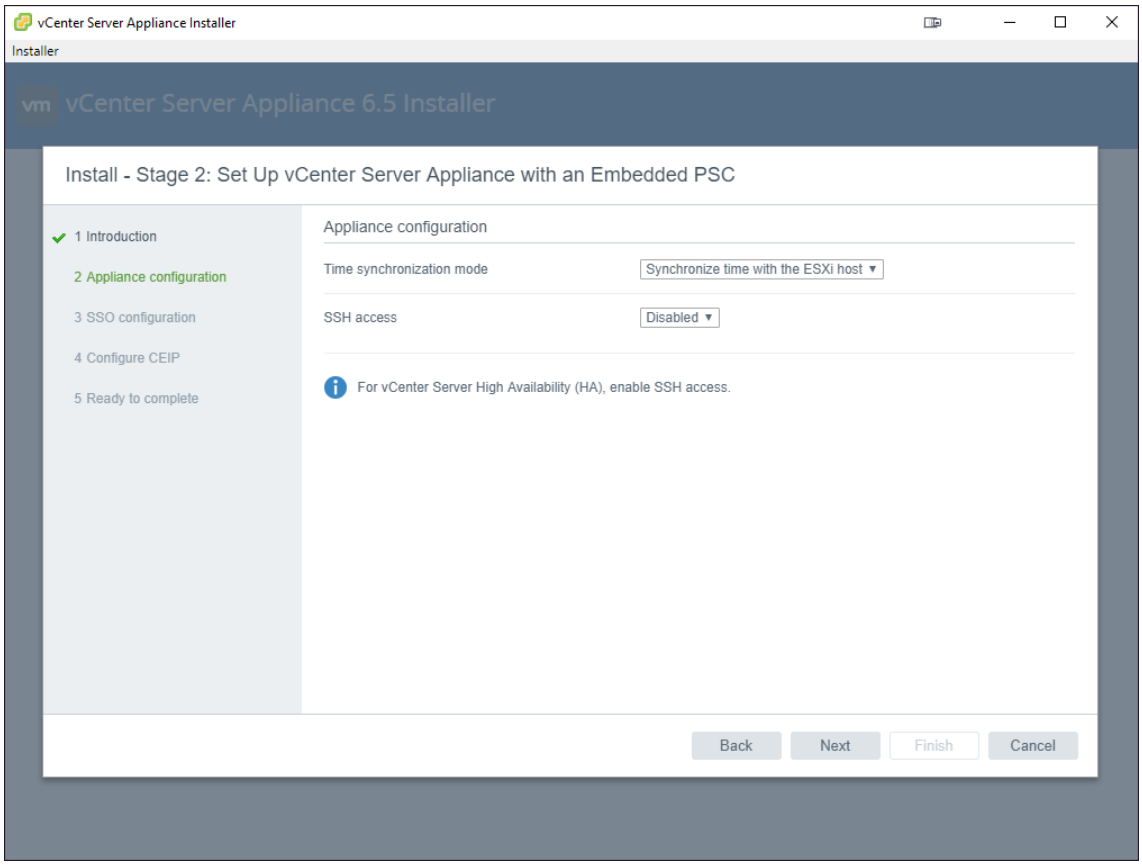

Figure 8. Appliance configuration

The vCenter server appliance is set to synchronize it’s time with the ESXi server and the SSH access is disabled, as the server management is performed over the web user interface.

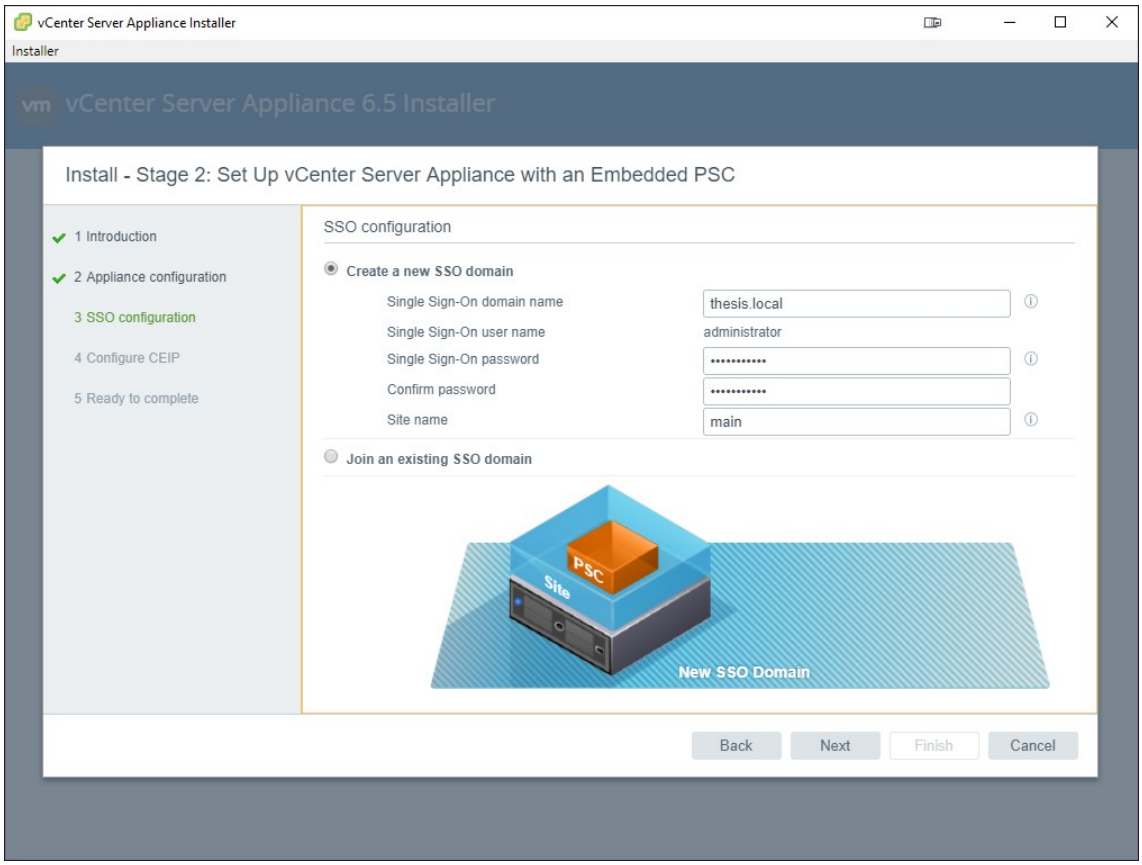

Figure 9. SSO configuration

The server has a new Single Sign On (SSO) domain created, with administrator user credentials created in this step. These credentials are used later to login to the web user interface of the vSphere. Following this, the server appliance is opted out of the VMware Customer Experience Improvement Program. The final step shows a list of the configured settings, which are applied when the finish button is pressed. After this step is complete, the web user interface is available at the IP address specified earlier in the installation progress.

After logging onto the vSphere client, a new datacenter was created and the ESXi host was added to it. A new virtual machine was created, and a Lubuntu ISO was uploaded to the datastore. The virtual machine's virtual CD drive was set to point to the ISO file and the operating system was installed on the virtual machine. This virtual machine was then cloned a couple times to create the virtual machine base used for demonstrating the use of the application.

4 VMware vSphere Automation SDK

The VMware vSphere Automation SDK uses an API endpoint located on the vCenter server, which grants access to services and resources managed by the vCenter server, such as virtual machines.

4.1 Authenticating with vSphere

The application must authenticate with vSphere Server in order to perform actions on the items and services managed by vSphere. For the purposes of the development version, some non-ideal approaches to security are taken. These will be mitigated in the production use version of the software.

The connection to the vSphere API endpoint is done over a connection secured with SSL. As by default the vSphere installation comes with a self-signed certificate that is not considered valid, the certificate verification is skipped.

```
ServicePointManager.ServerCertificateValidationCallback =
        (sender, certificate, chain, sslPolicyErrors) => true;
```

Listing 5. The certificate validation is overridden as true to bypass invalid certificate error.

The authentication with the development environment is done with a username and password. The automation SDK also offers the option of using security tokens for authentication, which will be used in production. The application connects to the endpoint with Single Sign On (SSO) user credentials to obtain a session identifier. The session identifier is only valid for the service endpoint being requested. (3, 33-34)

To access the resources managed by vSphere, the application must connect to a vSphere automation endpoint. Access to the endpoints is provided by vCenter lookup service. For this project the primary points of interest are the vCenter Single Sign-On and vSphere Automation API endpoints, as these are used to create the authenticated session with the vSphere server and to manage vSphere

services and resources. To form the connection to a lookup service, a HTTPS connection stub is formed.

The capacity to perform virtual machine management using the vSphere Automation API is limited in this project by the fact that the target environment is running vSphere 6.5U3 which has less management functionality included in it's API compared to more recent 7.0 version.

The primary use of the virtual machine management functionality in this project is in relation to virtual machine power states. The intention is to create a program that will be able to retrieve the power state of a virtual machine or several virtual machines and perform actions as specified. The API has support for starting, stopping, and suspending virtual machines. Suspending of a virtual machine functions similarly to the hibernation function on physical computers. As that functionality is not used in the production environment in an effort to conserve resources, the focal point is powering on and powering off virtual machines as requested. The target environment hosts a number of virtual machines which all perform the same task. The intention is that when a software asks to connect to a virtual machine serving that role, this program will take that request and check which virtual machines in the group are already running. It is naively assumed that running virtual machines are already in use, and therefore an unused virtual machine will be powered on, and the requesting software will be informed as to which virtual machine it will be connecting to. Once the software is done using the services of the virtual machine, it relays its request to the program which then proceeds to shut down the virtual machine in question.

5 The Application

The application is written in C# so that parts of it can be integrated into another software product also written in the same language. The software product would use the code from this project to list virtual machines and change their power states to provide services from said virtual machines as needed. For the purposes

of demonstrating the functionality of the program in this report, a separate console-based text user interface (Figure 2) was created.

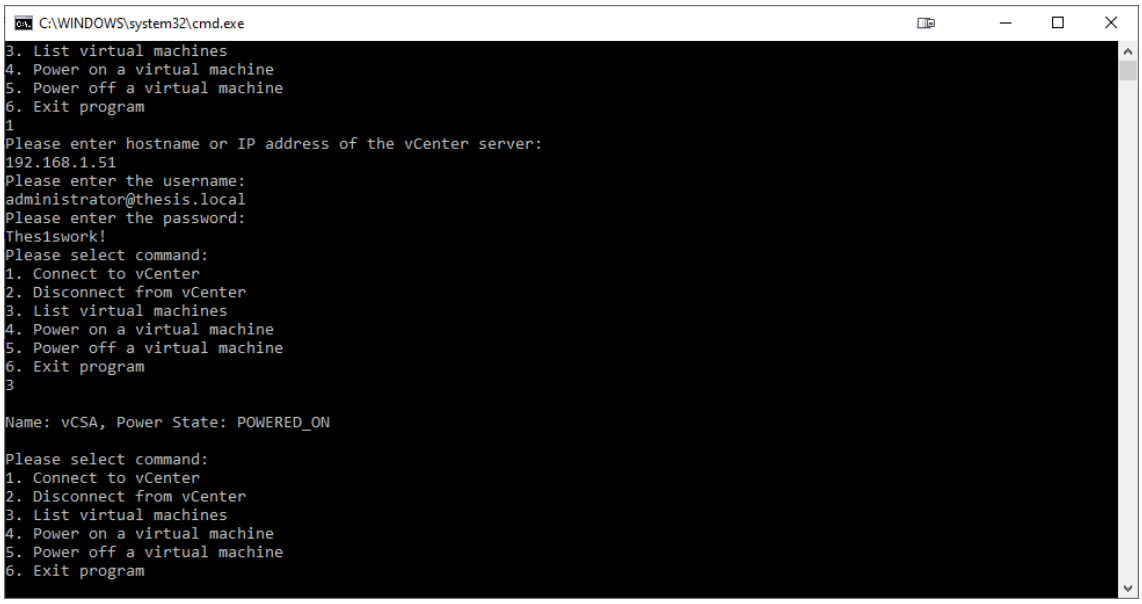

Figure 10. Console-based text user interface.

The application begins in a while-loop, printing out the basic text user interface. The user can make their selection by typing a number and pressing enter. In the case where the user types something that cannot be parsed as a number, the application prompts the user to enter a number as their selection. Before the other functionality of the application can be used, the user has to authenticate with the vCenter server to login. In the example screenshot seen in Figure 10, the user has authenticated as the administrator user. In the production use scenario, the authentication would not be handled with a username and password combination, as storing such information in a configuration file or program code itself would be unsafe. Instead, the SAML token would be used as the authentication method.

```
Console.WriteLine("Please enter hostname or IP address of the vCenter server:");
hostname = Console.ReadLine();
Console.WriteLine("Please enter the username:");
username = Console.ReadLine();
Console.WriteLine("Please enter the password:");
password = Console.ReadLine();
//ignores self-signed certificate error
ServicePointManager.ServerCertificateValidationCallback = (sender, certificate,
chain, sslPolicyErrors) => true;
```

```
try
{
      SessionStub = vapiAuthenticationHelper.LoginByUsernameAndPassword(hostname,
username, password);
}
catch (System.AggregateException)
{
      Console.WriteLine(“Incorrect credentials or hostname/IP address, please
attempt again.”);
}
```

Listing 6. Authentication code.

The authentication requires username, password, and hostname or IP address of the vCenter server that will be connected to. These all are read from user input on the console and then fed to the session stub constructor. If there is an error of some kind in this process, the exception is caught, and the user will see a message telling them to attempt making the connection again.

```
vapiAuthenticationHelper.Logout();
```

Listing 7. Logout code

The logout option in the application simply calls the logout function of the vapiAuthenticationHelper class, which deletes the session service that was created upon authenticating in the previous step.

```
List<VMTypes.Summary> vmList = new List<VMTypes.Summary>();
VM vmService = vapiAuthenticationHelper.StubFactory.CreateStub<VM>(SessionStub);
VirtualMachineList lister = new VirtualMachineList(vapiAuthenticationHelper,
SessionStub, vmService);
vmList = lister.GetVirtualMachineList();
foreach (VMTypes.Summary vm in vmList)
{
Console.WriteLine();
Console.WriteLine("Name: " + vm.GetName() + ", Power State: " +
vm.GetPowerState().ToString());
Console.WriteLine();
}
```

Listing 8. Listing all virtual machines and their power states

The option of listing virtual machines will print a list of all virtual machines that are part of the datacenter, as well as their power states. In order to process the virtual machines and their states, an array is created which will store data with the type VMTypes.Summary. The VMtypes.Summary class contains several

useful pieces of information about the virtual machine, including its name, and especially importantly for this work, the power state of the virtual machine. (4)

The VirtualMachineList class uses the VM interface to get a list of up to 1000 virtual machines from vSphere, the results of which are stored in the vmList variable. Following this the contents of the array are iterated over a foreach loop, with the application printing a new line to console for each virtual machine containing its name and power state. (5)

```
int powerOn = 0;
Power powerOnService =
vapiAuthenticationHelper.StubFactory.CreateStub<Power>(SessionStub);
List<VMTypes.Summary> vmPoweronList = new List<VMTypes.Summary>();
VM vmPoweronService =
vapiAuthenticationHelper.StubFactory.CreateStub<VM>(SessionStub);
VirtualMachineList poweronLister = new
VirtualMachineList(vapiAuthenticationHelper, SessionStub, vmPoweronService);
vmPoweronList = poweronLister.GetVirtualMachineList();
foreach (VMTypes.Summary vm in vmPoweronList)
{
    if (powerOn != 0)
    {
        break;
    }
    if (vm.GetName().Contains(vmName) &&
vm.GetPowerState().Equals(vmware.vcenter.vm.PowerTypes.State.POWERED_OFF))
    {
        try
        {
            powerOn = 1;
            powerOnService.Start(vm.GetVm());
            Console.WriteLine();
            Console.WriteLine("Virtual machine " + vm.GetName() + " has been
powered on.");
            Console.WriteLine();
        }
        catch (System.AggregateException)
        {
            Console.WriteLine("The operation failed, the virtual machine might be
already powered on.");
        }
}
```

Listing 9. Power service variable created, virtual machines listed, matching virtual machine powered on.

The power on and power off parts of the application are very functionally similar to that of the listing of all virtual machines in the vSphere environment. In the scenario where the user wants to power on a virtual machine, the application

asks for the name of the virtual machine from the user using the console. This is then matched against each virtual machine in the list, using the Contains function. The Contains function compares the string given by the user to the name of the virtual machine currently being processed by the foreach loop. If the virtual machine name matches even partially, the virtual machine is powered on. The reason for handling the virtual machine power on events in this manner is that the system has several virtual machines with almost identical names, with only numbers appended to the end of the name to indicate the instance of the virtual machine. As the virtual machines perform an identical purpose, it does not matter which virtual machine our application decides to power on. The int variable powerOn is used to check if a virtual machine has already been powered on as the loop iterates through the array. Once a virtual machine has been powered on, the value of the variable is changed from 0 to 1. This indicates that the application has started a virtual machine and it can exit the loop. The virtual machines are needed only one at a time, with more than one being powered on with a single command would be wasteful. Therefore, the application exits the loop after it has powered on one virtual machine. (6)

In the case where the user attempts to power on a virtual machine that already has the power state POWERED_ON, the application will raise System.AggregateException. This is handled with a try catch statement, which will print a line in the console, informing the user that the virtual machine they are attempting to power on might already have been powered on. In the production use this information would not appear in a console window, but instead it would be written into a log file that is used by other components of the program that will use the functionality of this application.

```
foreach (VMTypes.Summary vm in vmPoweroffList)
{
    if (vm.GetName().Equals(vmName))
    {
        try
        {
            powerOffService.Stop(vm.GetVm());
            Console.WriteLine();
            Console.WriteLine("Virtual machine " + vm.GetName() + " has been
powered off.");
            Console.WriteLine();
```

```
            }
            catch (System.AggregateException)
            {
            Console.WriteLine("The operation failed, the virtual machine might be
already powered off.");
            }
        }
}
```

Listing 10. Foreach loop of the virtual machine power off portion of the application.

The functionality in the application that powers off the virtual machine differs only very slightly to that of the powering on functionality. As there are several virtual machines capable of performing the same task, it does not matter which matching virtual machine is powered on. On the other hand, it would be very bad if the application would indiscriminately power off virtual machines that could be powered on and in use. Therefore, instead of searching for partial matches in the virtual machine names, the if statement in the power off section only accepts exact matches of virtual machine names before powering off the virtual machine. The power off section contains similar error catching code as the powering on of virtual machines, as the application could raise the same error if the user attempts to power off a virtual machine that already has a power state of POWERED_OFF. This is again handled with a try catch statement that will inform the user via a line printed in the console if the powering off operation raised an error.

```
System.Environment.Exit(0);
```

Listing 11. Cleanly exiting the application.

If the user chooses to exit the application, this is done by using the System.Environment.Exit method with the parameter of 0. This closes the application, with the parameter 0 indicating that the application closed successfully, not because of an error. (7)

6 Conclusion

The goal of this thesis was to create a solution that would automate the power state management of virtual machines in a VMware vSphere environment. The application created as a part of this thesis work is mostly for demonstration purposes, as the actual implementation into another program created internally at the corporation differs somewhat and could not be shown here for confidentiality reasons. The application in its current form is a useful tool for managing the power states of virtual machines, allowing for lower resource utilization across the virtualized environment, but it is very simple. The application could be further improved by supporting other functionality afforded by the VMware vSphere Automation API. The application helped the company by saving money by reducing the amount of idle virtual machines, reducing the necessity of upgrades of existing hardware or purchases additional hardware.

References

1 Portnoy, Matthew. 2016. Virtualization Essentials Second Edition. Indianapolis: Sybex.

2 Gavanda, Martin; Mauro, Andrea; Valsecchi, Paolo & Novak, Karel. 2019. Mastering VMware vSphere 6.7 Second Edition. Birmingham: Packt Publishing Ltd.

3 VMware, Inc. 2020. VMware vSphere Automation SDKs Programming Guide. Palo Alto: VMware, Inc.

4 VMware, Inc. Online. VMware vSphere Automation SDK for .NET 6.5.0: vmware::vcenter::VMTypes::Summary Class Reference https://vmware.github.io/vsphere-automation-sdk-.net/vsphere/6.5.0/vapi-client-bindings/classvmware_1_1vcenter_1_1VMTypes_1_1Summary.html Accessed May 2021.

5 VMware, Inc. Online. VMware vSphere Automation SDK for .NET 6.5.0: vmware::vcenter::VM Interface Reference https://vmware.github.io/vsphere-automation-sdk-.net/vsphere/6.5.0/vapi-client-bindings/interfacevmware_1_1vcenter_1_1VM.html Accessed May 2021.

6 VMware, Inc. Online. VMware vSphere Automation SDK for .NET 6.5.0: vmware::vcenter::vm:Power Interface Reference https://vmware.github.io/vsphere-automation-sdk-.net/vsphere/6.5.0/vapi-client-bindings/interfacevmware_1_1vcenter_1_1vm_1_1Power.html Accessed May 2021.

7 Microsoft Corporation. Online. Environment.Exit(Int32) Method (System) | Microsoft Docs. https://docs.microsoft.com/en-us/dotnet/api/system.environment.exit?view=net-5.0 Accessed June 2021.