Azure Dev/Test Lab Automation – Technical Guide

Dev Test on Azure Resource Manager - Jumpstart

Prepared for

**Update [Customer] in Doc Properties**

31-Aug-2015

Version 0.1 Draft

Prepared by

**Author**

Update [Author Position] in Doc Properties

Update [Author Email] in Doc Properties

Contributors

**Update [Contributors] in Doc Properties**

Table of Contents

[1 Introduction 4](#_Toc432062899)

[2 Objectives 5](#_Toc432062900)

[3 Dev Test on Azure (Infrastructure as Code) 6](#_Toc432062901)

[4 Azure Resource Groups 8](#_Toc432062902)

[5 Azure Resource Manager (ARM) Templates 9](#_Toc432062903)

[6 Scenarios 12](#_Toc432062904)

[6.1 Deploy resources into a resource group 12](#_Toc432062905)

[6.2 Stop all VMs in a resource group 12](#_Toc432062906)

[6.3 Start all VMs in a resource group 13](#_Toc432062907)

[6.4 Back up an Azure VM 13](#_Toc432062908)

[6.5 Restore or clone an Azure VM to a previously backed up point. 13](#_Toc432062909)

[6.6 Generate RDCMan file to manage all VMs in a resource group. 14](#_Toc432062910)

[6.7 Configure an Azure VM as a DSC pull server 14](#_Toc432062911)

[6.8 Install windows update on all VMs in a resource group 14](#_Toc432062912)

[6.9 Copy VSO files into an Azure VM 14](#_Toc432062913)

[6.10 Remove a resource group 14](#_Toc432062914)

[6.11 Upload VSO files to an Azure storage account 15](#_Toc432062915)

[6.12 Sync runbooks from VSO git repository 15](#_Toc432062916)

[6.13 Deploying a Linux VM on Azure with a LAMP app 15](#_Toc432062917)

[6.13.1 Create a new Ubuntu VM with Apache, My SQL, PHP and sample PHP app installed. 16](#_Toc432062918)

[6.13.2 Create a new Ubuntu VM with LAMP and JRE installed 18](#_Toc432062919)

[6.13.3 Installation of Apache, My SQL, and PHP on an existing Ubuntu VM 19](#_Toc432062920)

[6.13.4 Testing Apache Web Server via Http Request 19](#_Toc432062921)

[6.13.5 Installation of JRE on an existing Ubuntu VM. 20](#_Toc432062922)

[6.13.6 How to create a custom Linux VHD for Azure 20](#_Toc432062923)

[7 How to use this offering 51](#_Toc432062924)

[7.1 Resources 51](#_Toc432062925)

[7.1.1 Runbooks 51](#_Toc432062926)

[7.1.2 Files 51](#_Toc432062927)

[7.1.3 Templates 52](#_Toc432062928)

[7.2 Setting up your Azure subscription for automation 52](#_Toc432062929)

[7.2.1 Upload ARM powershell module 52](#_Toc432062930)

[7.2.2 Create OrgId in your Active Directory 52](#_Toc432062931)

[7.2.3 Create OrgId Credential Asset 53](#_Toc432062932)

[7.3 Runbooks explained 53](#_Toc432062933)

[7.3.1 Deploy-ResourceGroup 53](#_Toc432062934)

[7.3.2 Stop-AzureVMs 54](#_Toc432062935)

[7.3.3 Start-AzureVMs 55](#_Toc432062936)

[7.3.4 Backup-AzureVM 55](#_Toc432062937)

[7.3.5 Restore-AzureVM 57](#_Toc432062938)

[7.3.6 Generate-RDCManFile 59](#_Toc432062939)

[7.3.7 Configure-DSCPullServer 60](#_Toc432062940)

[7.3.8 Install-WindowsUpdate 62](#_Toc432062941)

[7.3.9 Copy-VSOFilesToAzureVM 63](#_Toc432062942)

[7.3.10 Remove-ResourceGroup 65](#_Toc432062943)

[7.3.11 Upload-VSOFilesToAzureStorage 65](#_Toc432062944)

[7.3.12 Sync-AzureAutomationRunbooks 67](#_Toc432062945)

[7.3.13 Install-PackageOnLinuxVM 68](#_Toc432062946)

[7.3.14 Test-URL 70](#_Toc432062947)

[8 Appendix 71](#_Toc432062948)

[8.1 View runbooks logs 71](#_Toc432062949)

[8.2 Design Decisions 72](#_Toc432062950)

[8.2.1 Using custom script extensions 72](#_Toc432062951)

1. Introduction

Organizations face increasing pressure to develop, test and deliver more software faster, organizations are seeking new and innovative ways to shorten time to market. Development and Test environments allow to develop and to test applications faster at lower costs. A Dev/Test solution not only provide infrastructure on demand, but also allow enterprises to reduce development time, increase quality at minimal cost. And just like many other workloads, Dev and Test infrastructure can be moved to the cloud, which makes it so compelling: the ability to scale up and down at any time without delay to adjust to actual need; and economic factors unique to the cloud that will generate enormous cost savings compared to on-premises infrastructure resources.

This solution helps customers take advantage of Microsoft Azure’s scale, elasticity, and per-minute billing to provide more agile and cost effective dev and test labs. In it, Microsoft helps you design a lab solution that lets you balance developer self-service and productivity with standardization and operational control. Microsoft not only helps design the right Windows Azure topology to meet your needs, but also provides a library of lab automation code that makes it fast and efficient to build and operate complex lab environments in the cloud.

1. Objectives

The objective of this offering is to automate the practical scenarios around Azure Dev/Test infrastructure. We provide automation runbooks and dependent files that can be imported into any Azure subscription and automate common tasks that will minimize the Azure utilization costs and improve the efficiency.

This document will explain the resources provided in the offering, how to prepare and setup an Azure environment and how to use them in an engagement to deliver a Dev/Test environment in Azure.

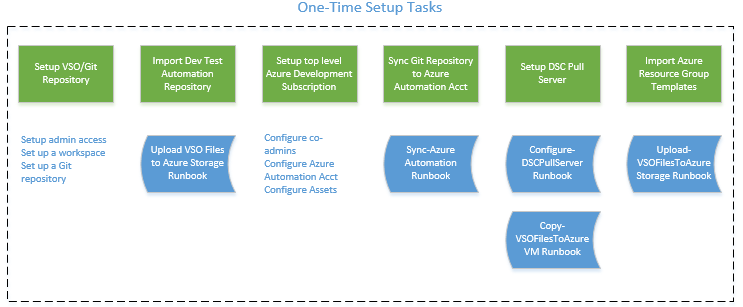
This offering targets the new Azure portal (<http://portal.azure.com> ) and not the management portal (<http://manage.windowsazure.com> ). Unless otherwise specified explicitly, mention of Azure portal in this document will mean the new portal (<http://portal.azure.com> )

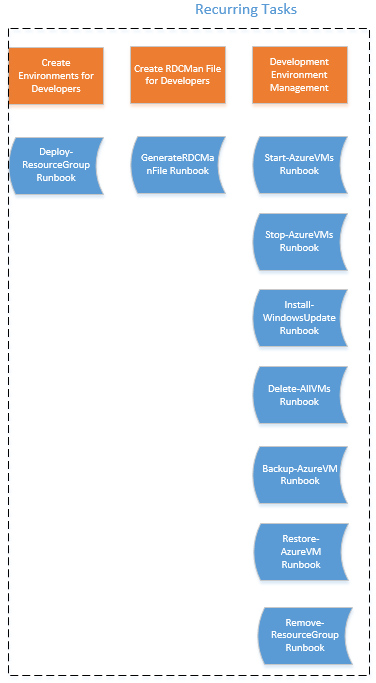
1. Dev Test on Azure (Infrastructure as Code)

This offering and solution for Dev/Test on Azure is implementing using Infrastructure as Code (IaC) using Azure core components. IaC is one fundamental practice of DevOps and is the practice in which the techniques, processes, and tool sets used in software development are leveraged to manage the deployment and configuration of systems, applications, and middleware. A significant number of testing and deployment defects occur when developers’ environments defining the application and underlying infrastructure differ from testing and production environments. Standardizing these environment definitions, putting them under version control, and deploying and configuring the infrastructure and application automatically from the code in version control, yields immediate benefits in consistency, time savings, error rates, and auditability.

This offering coves two main areas for implementing Dev/Test on Azure. One-Time tasks that are required to setup the entire environment, and finally recurring tasks that are executed whenever needed, but most likely in a recurring fashion.

Below all required tasks are listed as well as their assigned automation runbooks that will help automating those tasks within Azure.





A single or multiple runbooks that can be used to implement a comprehensive Dev/Test on Azure scenario support these tasks.

1. Azure Resource Groups

In the Azure service management portal (<http://manage.windowsazure.com> ), managing a resource (a VM, database, website, etc.) requires you to perform operations against one resource at a time. But typically applications are made up of many resources – maybe a web app, database, database server, storage, and 3rd party services. You do not see these resources as separate entities, instead you see them as related and interdependent parts of a single entity. You want to deploy, manage, and monitor them as a group. Managing all these resources as a single application is a complex task in the service management portal.

In the Azure preview portal (<http://portal.azure.com> ), you can create resource groups to manage all your resources in an application together. Resource group serves as the lifecycle boundary for every resource contained within it. You can deploy, update or delete all of the resources for your application in a single, coordinated operation. You can clarify billing for your application by viewing the costs for the entire group.

Typically, a resource group will contain resources related to a specific application. For example, a group may contain a Website resource that hosts your public website, a SQL Database that stores relational data used by the site, and a Storage Account that stores non-relational assets.

The resources for different environments such as development, testing, staging and production are typically placed under separate resource groups.

You can learn more about resource groups in this article <https://azure.microsoft.com/en-in/documentation/articles/resource-group-portal/>

1. Azure Resource Manager (ARM) Templates

Rather than deploying and managing each resource separately, you can create an Azure Resource Manager template that deploys and provisions all of the resources for your application in a single, coordinated operation. In the template, you define the resources that are needed for the application and specify deployment parameters to input values for different environments. The template consists of JSON and expressions which you can use to construct values for your deployment.

The following example shows the sections that make up the basic structure of a template.

{

"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "",

"parameters": { },

"variables": { },

"resources": [ ],

"outputs": { }

}

The template structure contains 6 top level elements.

**$schema:** Required element. Location of the JSON schema file that describes the version of the template language.

**contentVersion:** Required element. Version of the template (such as 1.0.0.0). When deploying resources using the template, this value can be used to make sure that the right template is being used.

**Parameters:** Optional element. Values that are provided when deployment is executed to customize resource deployment. You can use these parameter values throughout the template to set values for the deployed resources.

**Variables:** Optional element. Values that are used as JSON fragments in the template to simplify template language expressions.

**Resources:** Required element. Types of services that are deployed or updated in a resource group. This is where you specify all resources that you want in your resource group along with their properties.

**Outputs:** Optional element. Values that are returned after deployment.

We have given below a sample template that creates a public IP with DNS Name in a resource group. This is taken as an example from <https://github.com/Azure/azure-quickstart-templates>, where you can find a list of quickstart ARM templates.

{

"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"parameters" : {

"dnsNameForPublicIP" : {

"type" : "string",

"metadata": {

"description": "DNS Name for the Public IP. Must be lowercase."

}

},

"location": {

"type": "string",

"defaultValue" : "West US",

"allowedValues": ["West US", "East US", "North Central US", "North Europe", "West Europe", "East Asia", "Southeast Asia"],

"metadata": {

"description": "Location of the Public IP."

}

}

},

"variables": {

"publicIPAddressName" : "publicIp1",

"publicIPAddressType" : "Dynamic"

},

"resources": [

{

"apiVersion": "2015-05-01-preview",

"type": "Microsoft.Network/publicIPAddresses",

"name": "[variables('publicIPAddressName')]",

"location": "[parameters('location')]",

"properties": {

"publicIPAllocationMethod": "[variables('publicIPAddressType')]",

"dnsSettings": {

"domainNameLabel": "[parameters('dnsNameForPublicIP')]"

}

}

}

]

}

You can notice that this template accepts two parameters dnsNameForPublicIP and location. These parameters can be given in a different JSON file during deployment, as given below.

{

"dnsNameForPublicIP": {

"value": ""

},

"location": {

"value": "West US"

}

}

This file is called ARM template parameter file.

This offering will use both ARM template and template parameter files for deploying resources. You can get more information about how you can author ARM templates for your application in this article: [Authoring Azure Resource Manager Templates](https://azure.microsoft.com/en-in/documentation/articles/resource-group-authoring-templates/).

1. Scenarios

Given below are the scenarios that are covered in this offering and that are successfully tested. This section will explain each of the scenarios. All required resources to deploy those scenarios can be found as part of this offering.

1. Deploy resources into a resource group
2. Stop all VMs in a resource group
3. Start all VMs in a resource group
4. Back up an Azure VM
5. Restore or clone an Azure VM to a previously backed up point.
6. Generate RDCMan file to manage all VMs in a resource group.
7. Configure an Azure VM as a DSC pull server
8. Install windows update on all VMs in a resource group
9. Copy VSO files into an Azure VM
10. Remove a resource group
11. Upload VSO files to an Azure storage account
12. Sync runbooks from VSO git repository
    1. Deploy resources into a resource group

This runbook deploys Azure resources to an Azure resource group through ARM template. The resources to be deployed should be formatted in an ARM template and uploaded to an Azure storage account. This runbook accepts the Uri to the uploaded ARM template and template parameter file and deploys the resources as designed in the template.

You can refer to section 7.3.1 to know how to use runbook ‘Deploy-ResourceGroup’ for this scenario.

* 1. Stop all VMs in a resource group

This runbook stops all the VMs in a resource group.

The recommendation is to schedule this runbook to be run at 8 PM every week day (or when the team usually finishes up the days’ work). Scheduling this runbook along with Start-AzureVMs runbook can minimize the cost associated with Azure virtual machine usage cost.

You can refer to section 7.3.2 to know how to use runbook ‘Stop-AzureVMs’ for this scenario.

* 1. Start all VMs in a resource group

This runbook starts all the VMs in a resource group.

The recommendation is to schedule this runbook to be run at 8 AM every week day (or when the team usually starts the days’ work). Scheduling this runbook along with Stop-AzureVMs runbook can minimize the cost associated with Azure virtual machine usage cost.

You can refer to section 7.3.3 to know how to use runbook ‘Start-AzureVMs’ for this scenario.

* 1. Back up an Azure VM

This runbook backs up the vhds (both OS and data disks) and VM configuration of the input VM to a Backup storage container. This accepts a ‘BackupSuffix parameter which is used to name the backed up files and to identify the backed up files while restoring.

You can refer to section 7.3.4 to know how to use runbook ‘Backup-AzureVM’ for this scenario.

* 1. Restore or clone an Azure VM to a previously backed up point.

This runbook restores or clones an Azure VM to a backed up state that was already backed up by the runbook Backup-AzureVM. The backed up state is identified by the parameter ‘BackupSuffix’. The backup can be done in 2 modes; Restore mode and Clone mode.

**Restore Mode:**

In the restore mode, the existing VM is deleted and a new one is created in its place with the target state of the VM.

**Clone Mode:**

In the clone mode, the existing VM is not deleted. A new VM is created with the target state in the same or a different resource group.

You can refer to section 7.3.5 to know how to use runbook ‘Restore-AzureVM’ for this scenario.

* 1. Generate RDCMan file to manage all VMs in a resource group.

This runbook generates a remote desktop connection manager file to manage remote connections to all the VMs in a resource group. The generated file will be uploaded to the given storage account. With the file, you can manage all VMs’ remoted connections in a single place.

You can refer to section 7.3.6 to know how to use runbook ‘Generate-RDCManFile’ for this scenario.

* 1. Configure an Azure VM as a DSC pull server

This runbook configures an Azure VM as a DSC pull server. Having a DSC pull server will be very useful when you want to push configurations to multiple VMs in your resource group. You can configure one of the VMs in your resource group as a pull server using this runbook and use this have other VMs pull configurations from this server.

You can refer to section 7.3.7 to know how to use runbook ‘Configure-DSCPullServer’ for this scenario.

* 1. Install windows update on all VMs in a resource group

This runbook Installs windows update on all VMs in a resource group. You can choose to automatically or manually reboot the VMs after the updates are installed.

You can refer to section 7.3.8 to know how to use runbook ‘Install-WindowsUpdate’ for this scenario.

* 1. Copy VSO files into an Azure VM

This runbook copies files from VSO git repository to an Azure VM. All files under the given folder and its subfolders are copied to the given path in the Azure VM.

You can refer to section 7.3.9 to know how to use runbook ‘Copy-VSOFilesToAzureVM’ for this scenario.

* 1. Remove a resource group

This runbook removes an Azure resource group and all its resources.

You can refer to section 7.3.10 to know how to use runbook ‘Remove-ResourceGroup for this scenario.

* 1. Upload VSO files to an Azure storage account

This runbook uploads files from a VSO git repository location into an Azure storage account. This runbook traverses through all folders and sub folders in the repository location and upload the files to the storage account. The folder structure hierarchy doesn’t apply to storage account containers and so all files are stored at the same level under the container as blobs.

You can refer to section 7.3.11 to know how to use runbook ‘Upload-VSOFilesToAzureStorage’ for this scenario.

* 1. Sync runbooks from VSO git repository

This runbook syncs all runbooks in a VSO git repository location to an Azure automation account starting with dependent (child) runbooks, followed by parent runbooks. The .ps1 files under the subfolders are treated as child runbooks. This runbook will traverse through all folders in the location, for each folder start from its sub folders, if any, all the way up to the parent folder and publish the .ps1 files as runbooks in the given Azure automation account.

You can refer to section 7.3.12 to know how to use runbook ‘Sync-AzureAutomationRunbooks’ for this scenario.

* 1. Deploying a Linux VM on Azure with a LAMP app

The deployment of a Linux VM on Azure can be done in 2 ways:

1. Using a Linux gallery image
2. Uploading a custom Linux image to Azure.

The custom Linux image approach is mostly required when:

* The required Linux distributions and versions is not available in the gallery.
* A customized image is required with some operating system settings and some applications already deployed.

**Important**: The Azure platform SLA applies to virtual machines running the Linux OS only when one of the endorsed distributions is used with the configuration details as specified under 'Supported Versions' in [Linux on Azure-Endorsed Distributions](http://azure.microsoft.com/en-us/documentation/articles/virtual-machines-linux-endorsed-distributions/). All Linux distributions in the Azure image gallery are endorsed distributions with the required configuration.

Also, depending on whether Apache server, My SQL, PHP, JRE etc. are required to be installed during the creation of the VM or on existing Linux VMs, we may choose to

1. Use an ARM template with custom script to deploy a new Linux VM.
2. Run the custom script extension on an existing Linux VM.

All the above scenarios are explained in this section along with sample runbooks, sh scripts and templates.

The Scenarios covered are as follows:

1. Create a new Ubuntu VM with Apache, My SQL, and PHP installed. Also a sample PHP application is deployed. This is done using an ARM template (slightly modified version of the GIT template <https://github.com/Azure/azure-quickstart-templates/tree/master/lamp-app> )
2. Create a new Ubuntu VM with LAMP and JRE installed. Also a sample PHP application is deployed. This is done using an ARM template.
3. Installation of Apache, My SQL, and PHP on an existing Ubuntu VM.
4. Testing the apache web server Http Response via Azure runbook
5. Installation of JRE on an existing Ubuntu VM.
6. Documentation on creation of custom Linux VHD for Azure
   * 1. Create a new Ubuntu VM with Apache, My SQL, PHP and sample PHP app installed.

This scenario explains creation of a new Ubuntu VM with Apache, My SQL, and PHP installed.

Artefacts used for this scenario:

* **Template**: “DeployLampApp.Json”
* **Parameter Json:** “DeployLampApp-Parameters.Json”
* **Script:** “install\_lamp.sh”
* **Runbook:** 7.3.1 Deploy-ResourceGroup.

This would create a LAMP VM and host a sample PHP application. The deployed php page can be opened at url http://<serverIP>/phpinfo.php.

The DeployLampApp-Parameters.Json file should be updated with the following parameter values:

***vmName***: Name of the LAMP VM to be created.

***Location*:** Location of the resources. Please take care of the Alphabet case and spacing. The correct location casing and spelling examples:

* "West US",
* "East US",
* "West Europe",
* "East Asia",
* "Southeast Asia"

***ubuntuOSVersion***: The version of Ubuntu OS. The allowed values are:

* "12.04.5-LTS",
* "14.04.2-LTS",
* "15.04"

***newStorageAccountName*:** Unique DNS Name for the Storage Account to be created where the Virtual Machine's disks will be placed.

***adminUsername*:** User name for the Virtual Machine.

***adminPassword*:** Password for the Virtual Machine.

***dnsNameForPublicIP*:** Unique DNS Name for the Public IP used to access the Virtual Machine.

***mySqlPassword*:** Password for the MySQL admin user.

***scriptName***: install\_ lamp.sh.

***scriptUri***: Uri of the script blob for the script “**install\_lamp.sh**“ Ex: "http://devteststorageaccount1.blob.core.windows.net/scripts/install\_lamp.sh"

***scriptStorageAccount*:** Azure storage account where script is stored.

Ex: devteststorageaccount1

***scriptStorageAccountKey*:** Key of the azure storage account where script is stored.

**Note**: These templates and scenario is taken and modified from <https://github.com/Azure/azure-quickstart-templates/tree/master/lamp-app>. Please refer to the above link for more information on the scenario.

Please check section 7.3.1 for details on running this runbook.

* + 1. Create a new Ubuntu VM with LAMP and JRE installed

This scenario creates an Ubuntu VM with LAMP installed exactly as the above mentioned scenario, but in addition to the LAMP, it also installs JRE on the VM.

Artefacts used for this scenario:

* **Template**: “DeployLampApp.Json”
* **Parameter Json:** “DeployLampApp-Parameters.Json”
* **Script:** “install\_LAMPandJRE.sh”
* **Runbook:** 7.3.1 Deploy-ResourceGroup.

This would create a LAMP VM and host a sample PHP application. The deployed php page can be opened at url http://<serverIP>/phpinfo.php.

The DeployLampApp-Parameters.Json file should be updated with the following parameter values:

***vmName***: Name of the LAMP VM to be created.

***Location*:** Location of the resources. Please take care of the Alphabet case and spacing. The correct location casing and spelling examples:

* "West US",
* "East US",
* "West Europe",
* "East Asia",
* "Southeast Asia"

***ubuntuOSVersion***: The version of Ubuntu OS. The allowed values are:

* "12.04.5-LTS",
* "14.04.2-LTS",
* "15.04"

***newStorageAccountName*:** Unique DNS Name for the Storage Account to be created where the Virtual Machine's disks will be placed.

***adminUsername*:** User name for the Virtual Machine.

***adminPassword*:** Password for the Virtual Machine.

***dnsNameForPublicIP*:** Unique DNS Name for the Public IP used to access the Virtual Machine.

***mySqlPassword*:** Password for the MySQL admin user.

***scriptName***: install\_LAMPandJRE.sh.

***scriptUri***: Uri of the script blob for the script “**install\_LAMPandJRE.sh**“.

Ex: "http://devteststorageaccount1.blob.core.windows.net/scripts/ install\_LAMPandJRE.sh"

***scriptStorageAccount*:** Azure storage account where script is stored.

Ex: devteststorageaccount1

***scriptStorageAccountKey*:** Key of the azure storage account where script is stored.

* + 1. Installation of Apache, My SQL, and PHP on an existing Ubuntu VM

In this scenario, Apache, My SQL, and PHP are installed on an **existing** Ubuntu VM. This is done using PowerShell ARM cmdlets and custom script extension for Linux.

Artefacts used for this scenario:

* **Script:** “install\_lamp.sh”
* **Runbook:** 7.3.13 Install-PackageOnLinuxVM.

Please refer to the Section 7.3.13 Install-PackageOnLinuxVM for the details to run this runbook.

Ensure that you provide filename and fileUri for the script “install\_lamp.sh”, and scriptParameters param as the password for the MySQL admin.

* + 1. Testing Apache Web Server via Http Request

In this scenario, a runbook is used to test the working of the Apache Web Server post deployment. The runbook tests a website by sending an HTTP request and validating the Http Response code from the server. In order to test if Apache Server is up and running, even if no website is deployed, just mention the server name or IP Address in the url, which will provide a response from the default page.

The runbook returns true if the Http Response Code is 200, and returns false in all other cases.

Please refer to the section 7.3.14 Test-URLTest-URL for details regarding the runbook execution.

* + 1. Installation of JRE on an existing Ubuntu VM.

In this scenario, JRE is installed on an **existing** Ubuntu VM. This is done using PowerShell ARM cmdlets and custom script extension for Linux.

Artefacts used for this scenario:

* **Script:** “install\_JRE.sh”
* **Runbook:** 7.3.13 Install-PackageOnLinuxVM.

Please refer to the Section 7.3.13 Install-PackageOnLinuxVM for the details to run this runbook.

Ensure that you provide filename and fileUri for the script “install\_ JRE.sh”, and scriptParameters param as empty string.

* + 1. How to create a custom Linux VHD for Azure

Please refer to the document published at the following Campus link for detailed description and steps:

<https://microsoft.sharepoint.com/teams/CampusIPLibraries/Campus/Community%20Shared%20IP/How%20to%20create%20RHEL%20hard%20disk%20for%20azure.docx#search=linux%20azure>

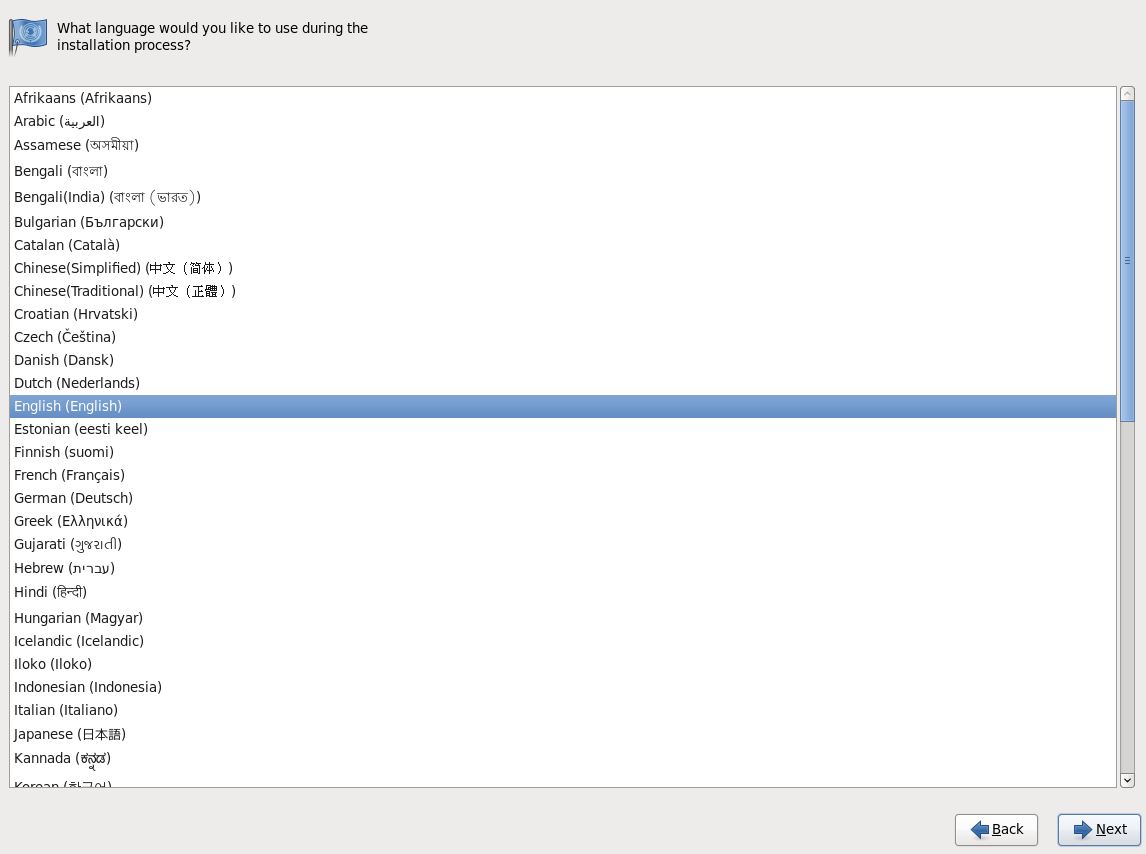
Below are the important steps mentioned in the above IP, but its recommended to go through the complete document for more details:

* + - 1. Create the Linux VHD

1. Boot Computer with RED HAT 6.3 OS Installation CD/DVD.
2. Select Install or Upgrade existing system options.

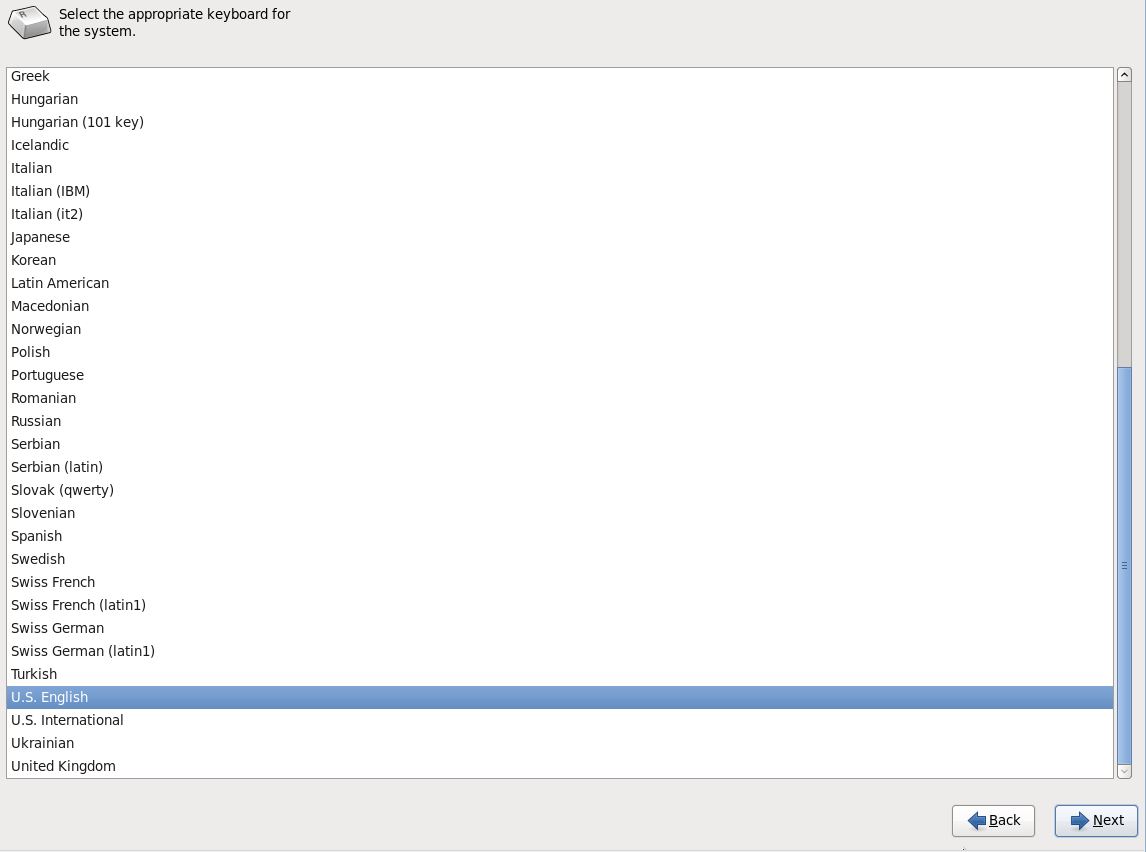


1. Select Language.



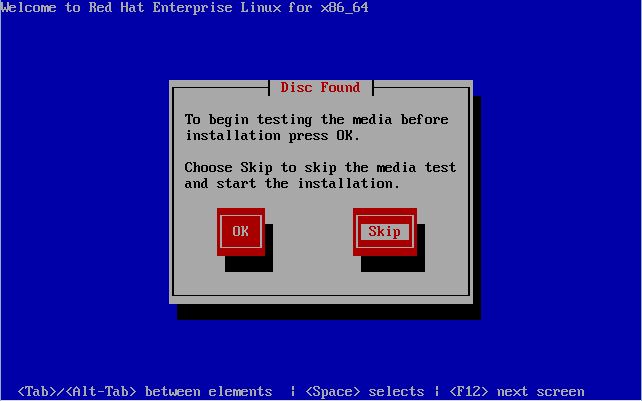
Click Next

1. Select keyboard type.

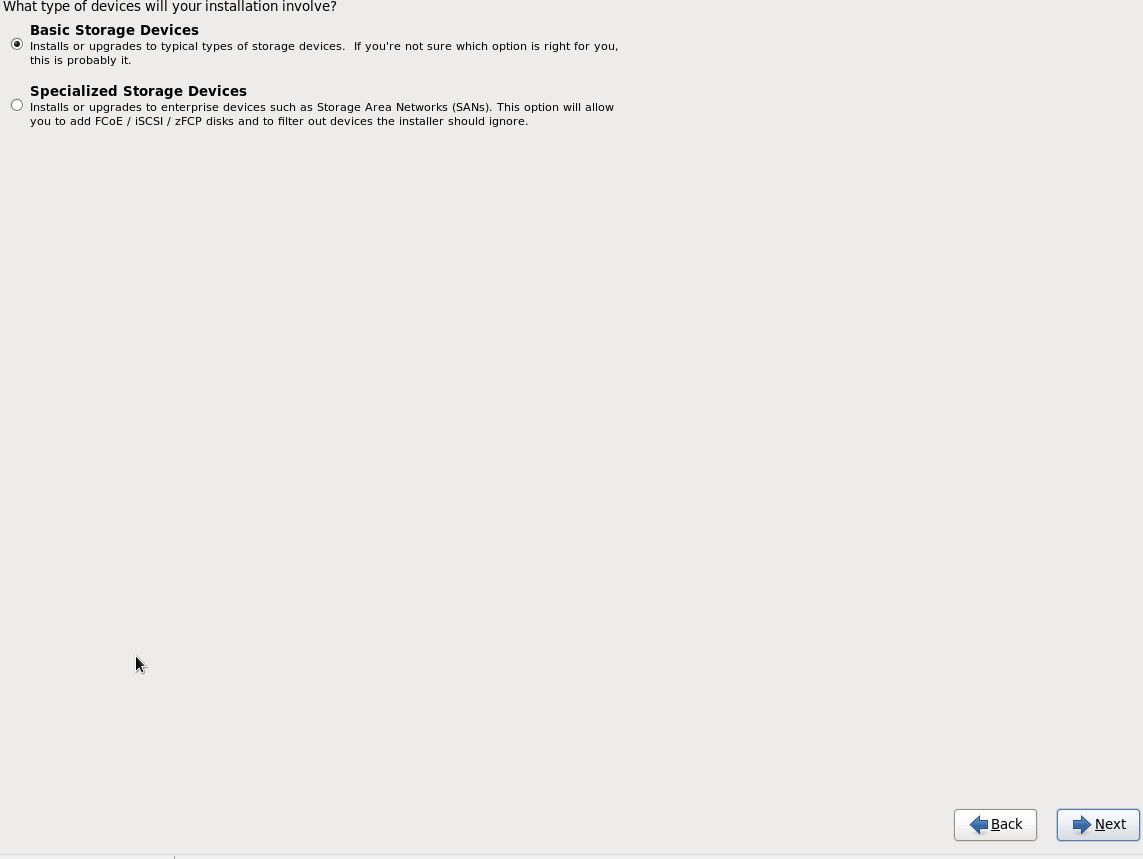


Click Next

1. Choose **skip** media test as it may take long time to check media.

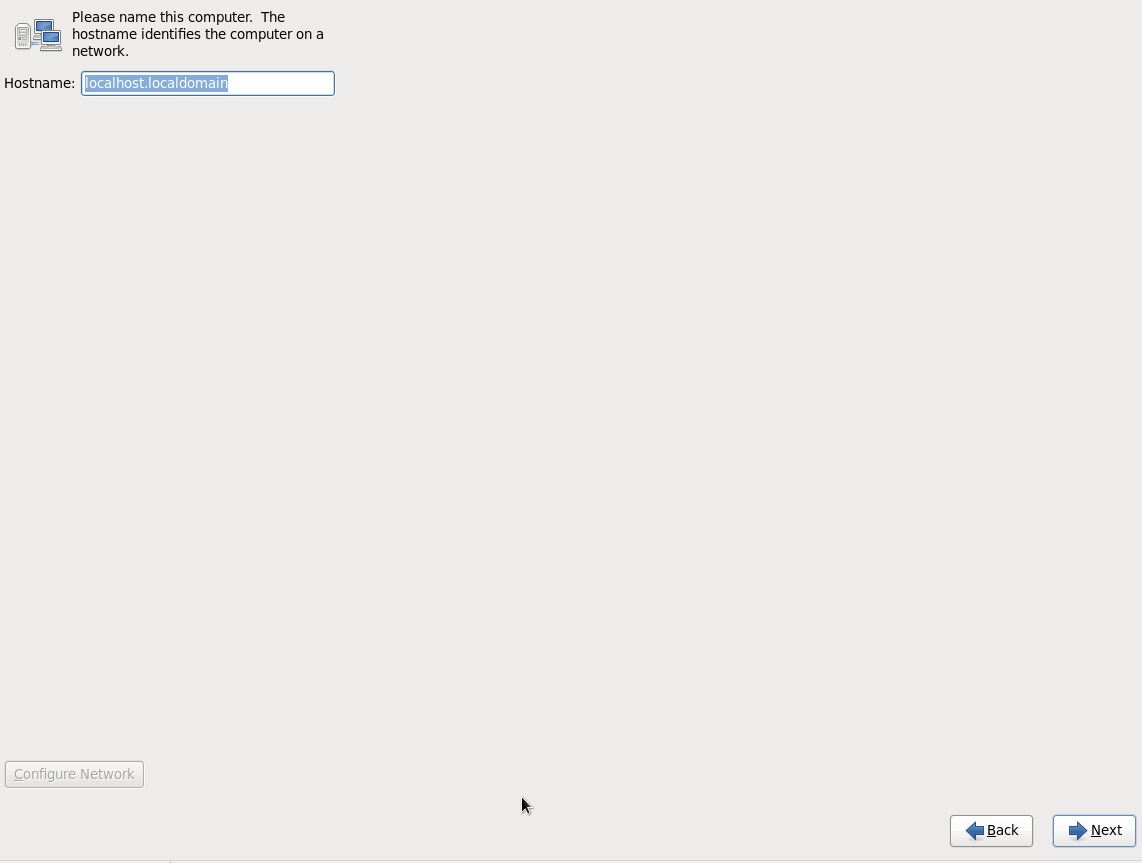


1. Select storage device.



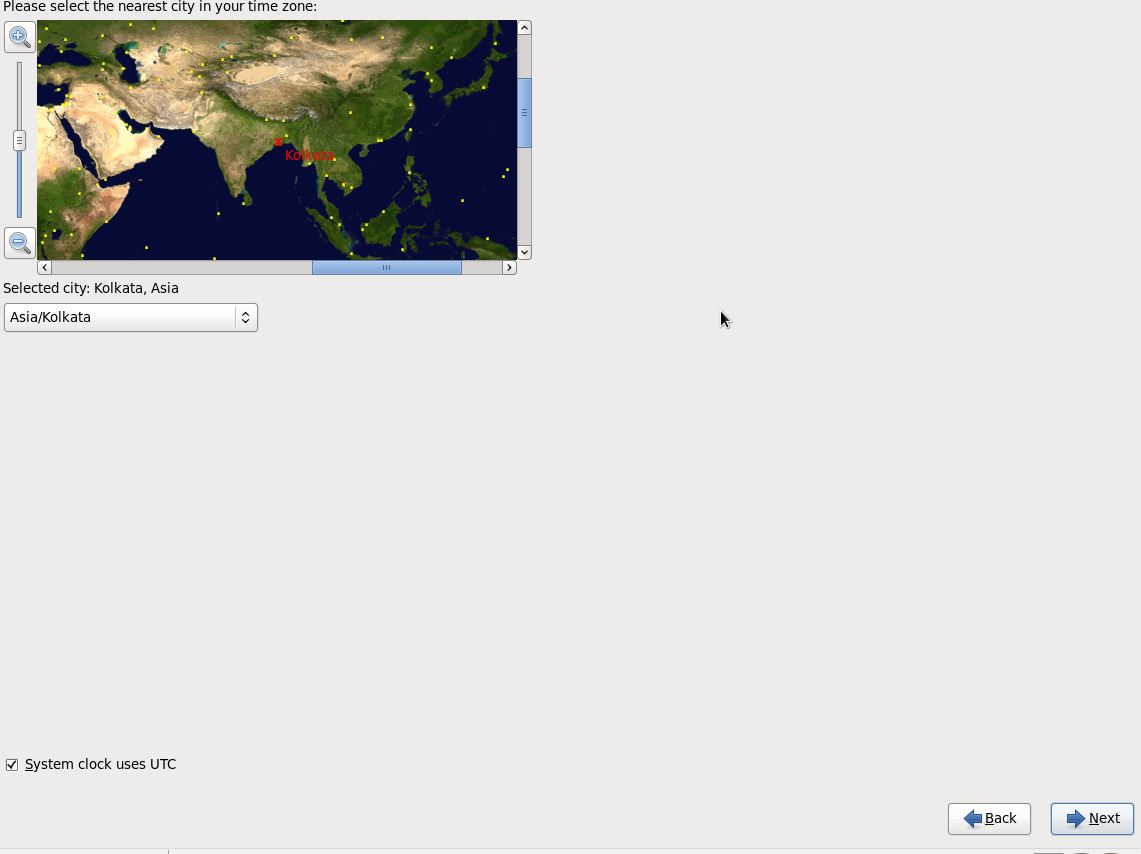
Click next

1. Type computer name or hostname



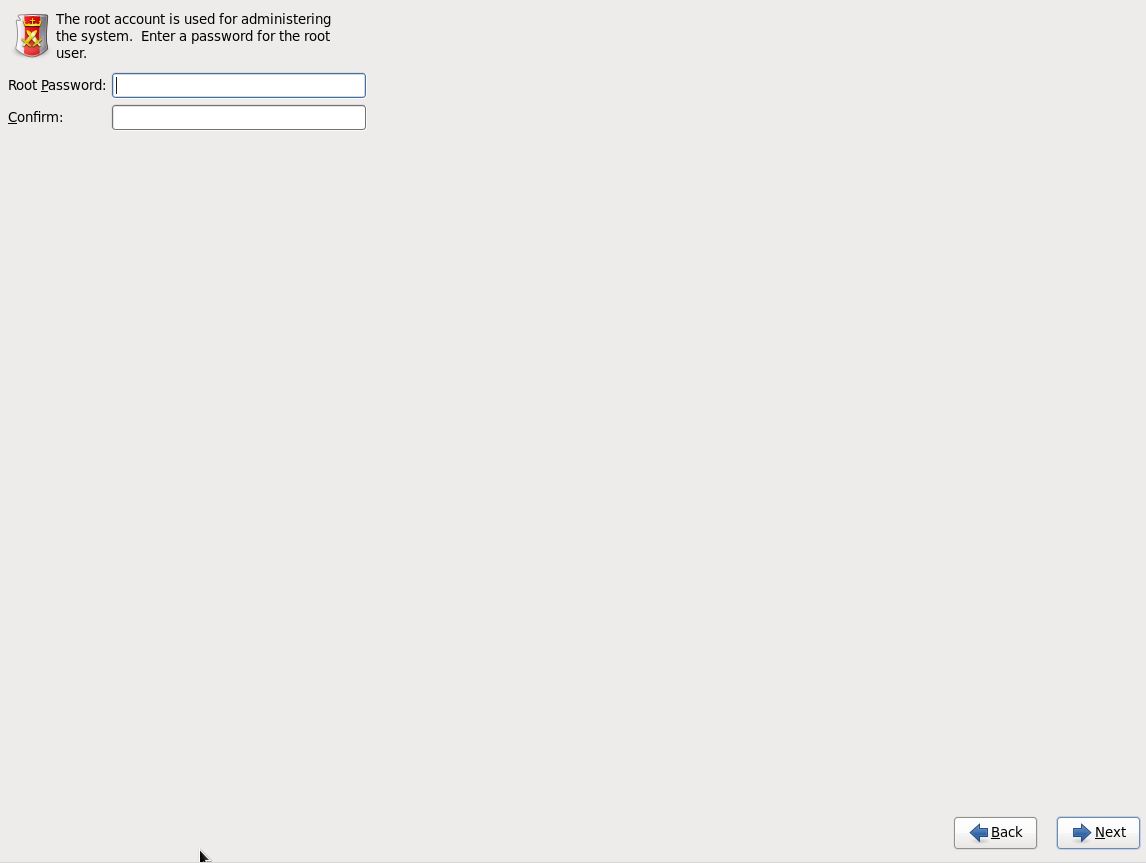
Click Next

1. Select time zone location.



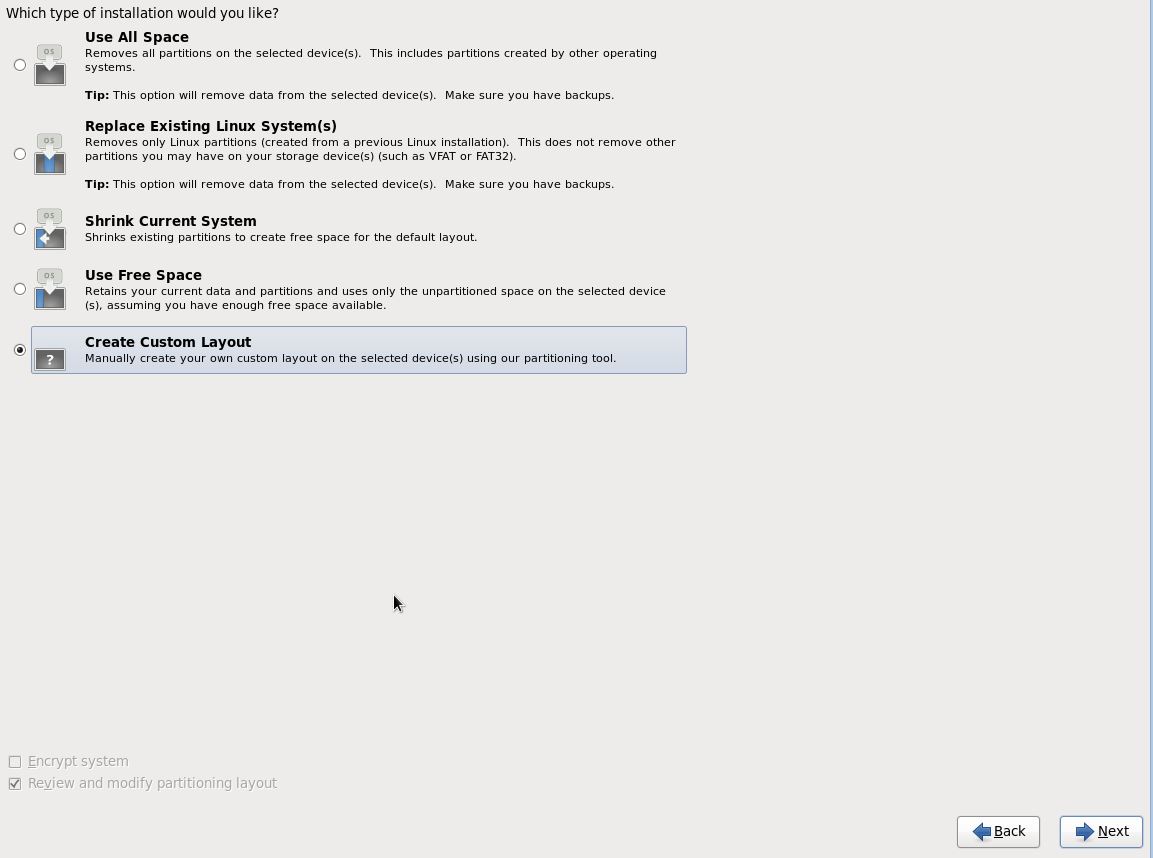
Click Next

1. Enter password for root user.

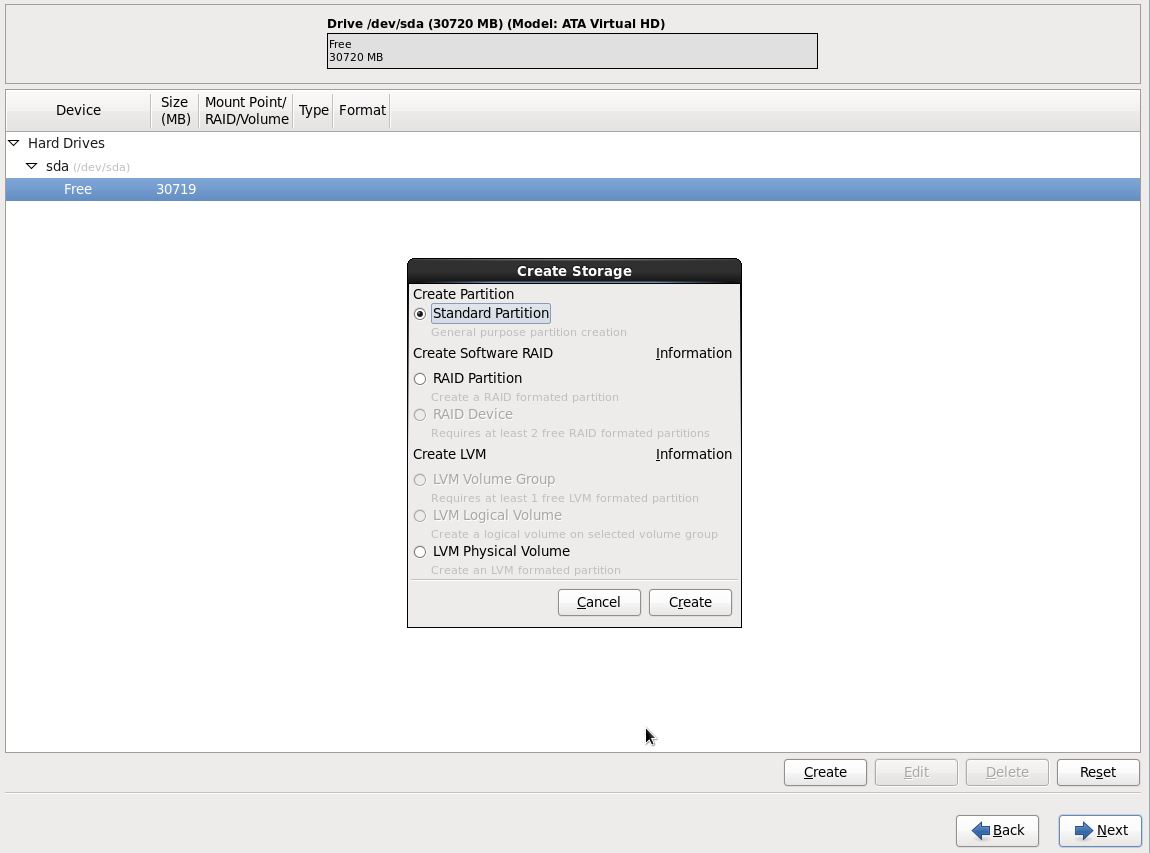


Click Next

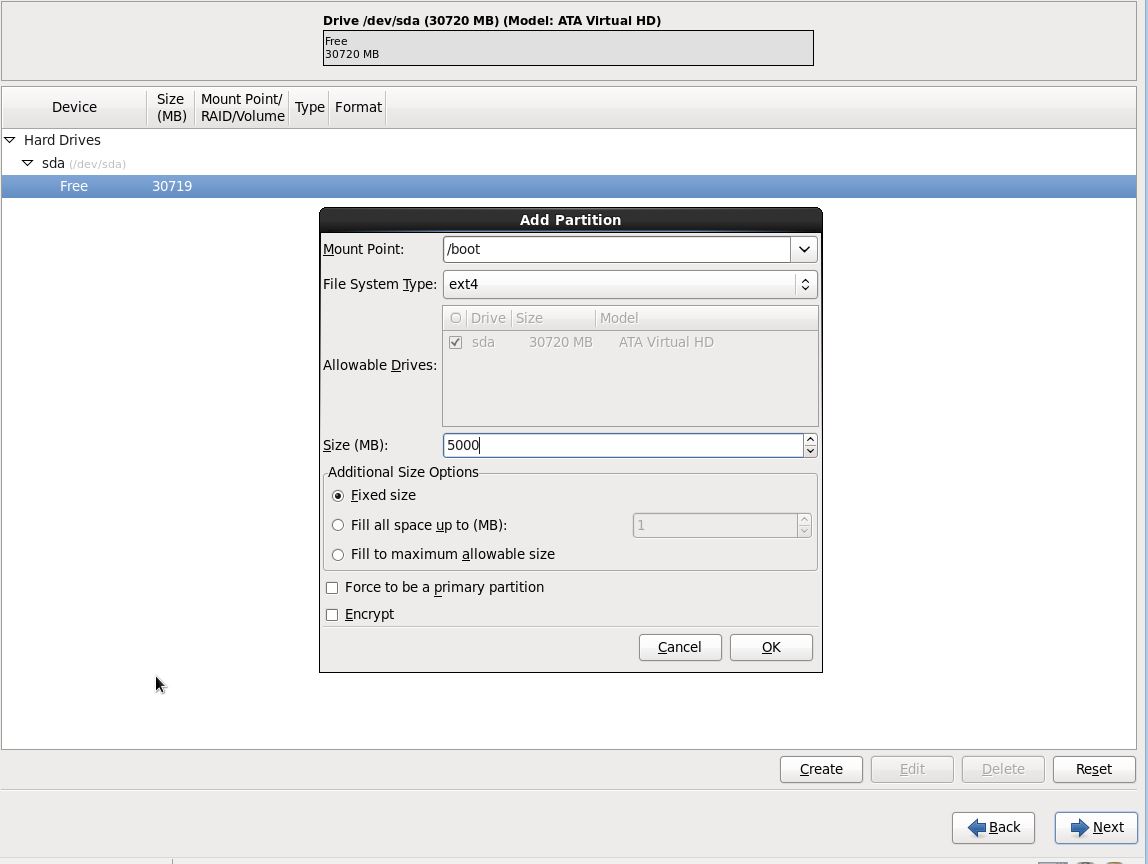
1. Select the option Create custom layout.



1. Review partitioning Layout and Create Manually configuration of /boot & /

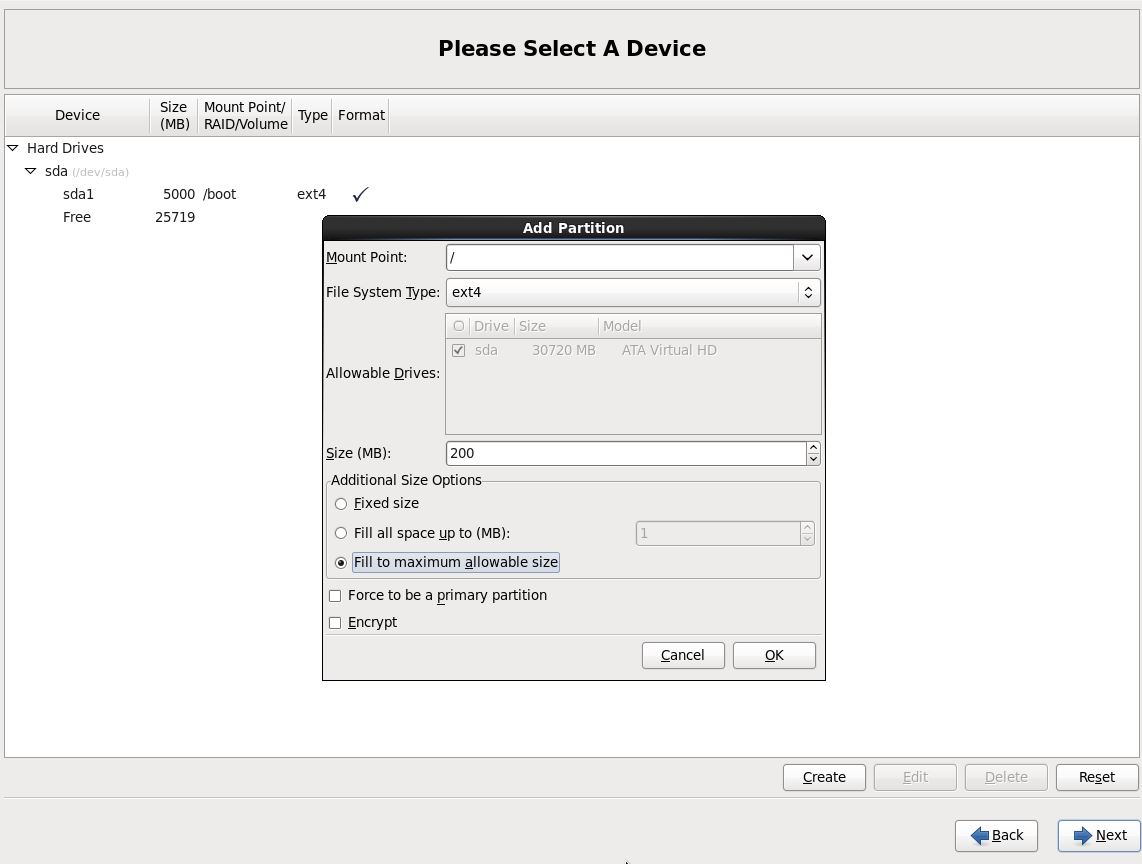
Click Next

1. Create /boot Partition

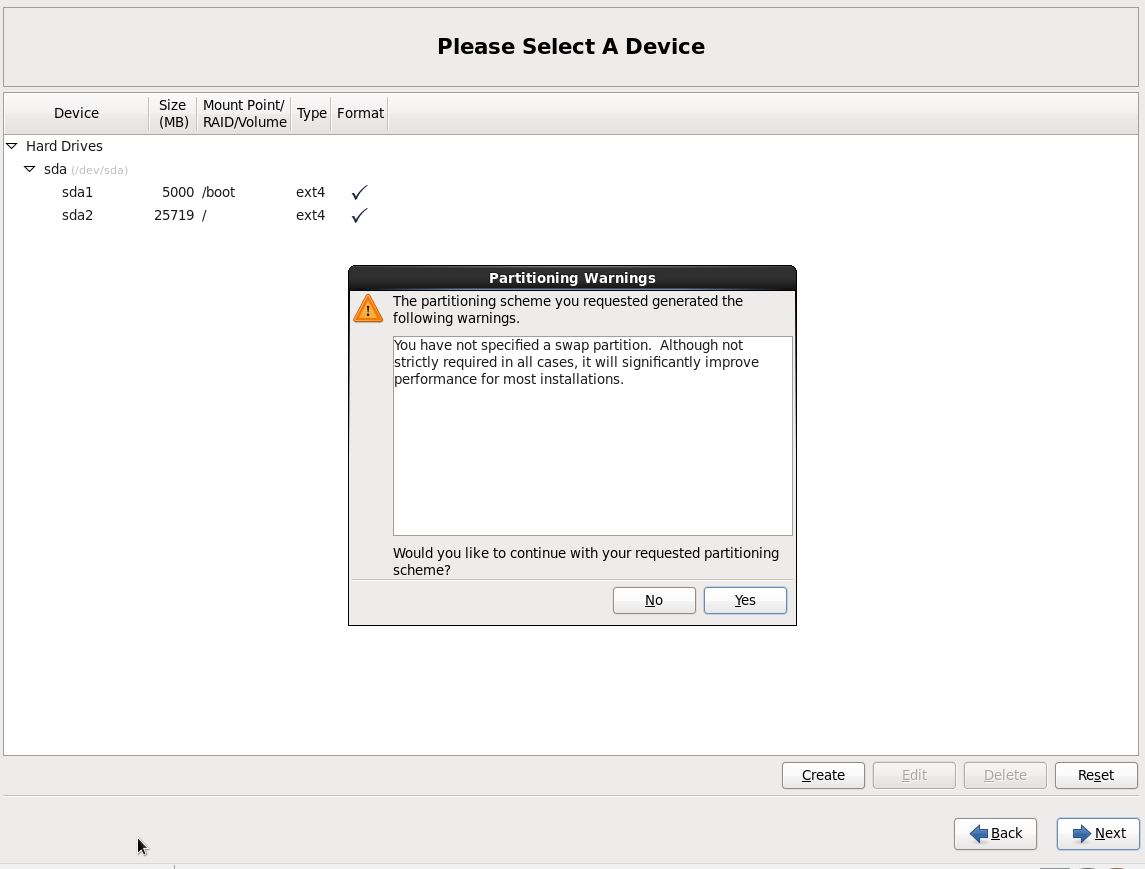


Click Next

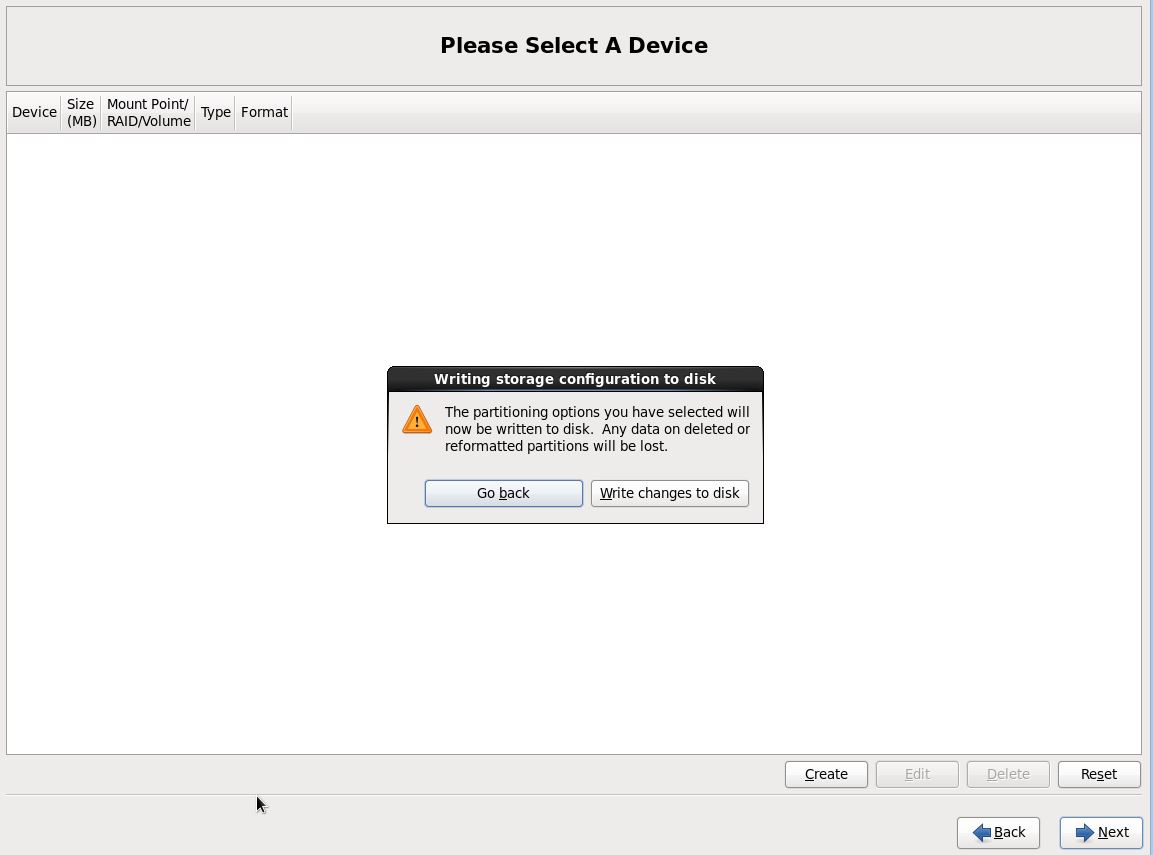
1. Create / Partition with clicking option Fill to maximum allowable size



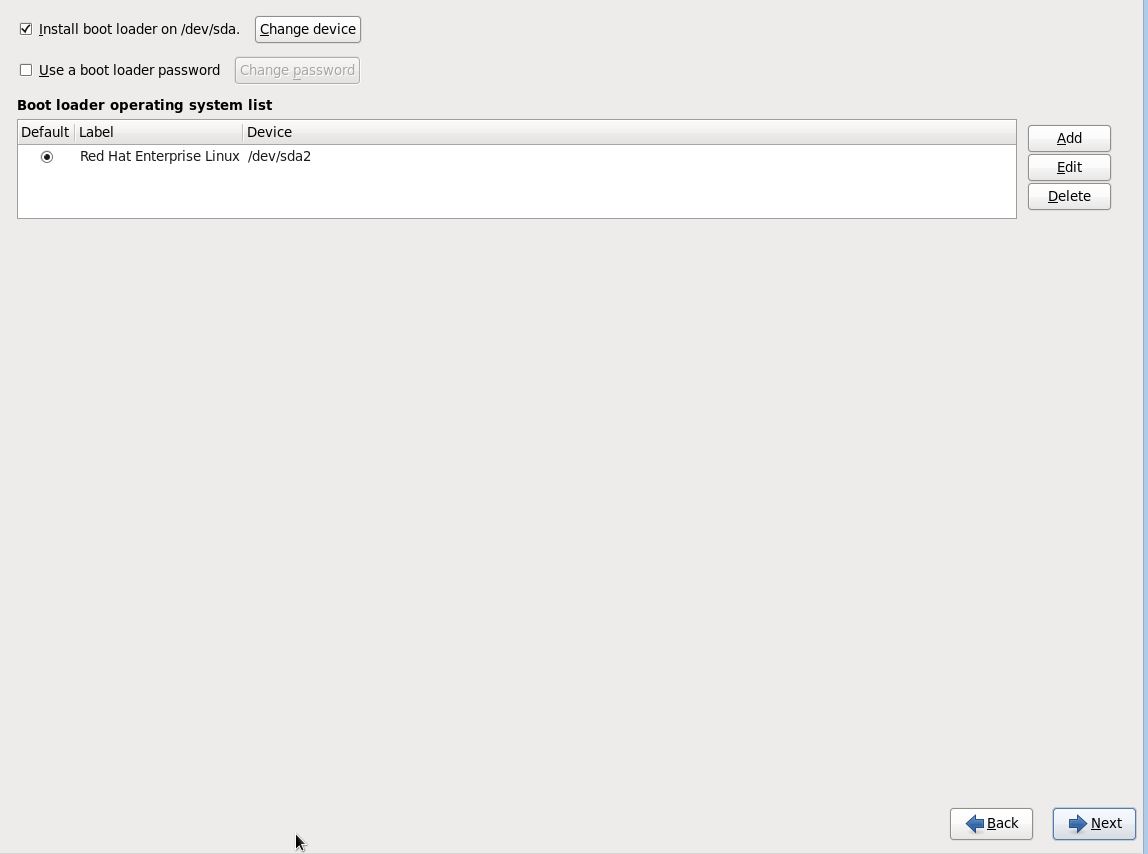
Click Next



Click Yes and next

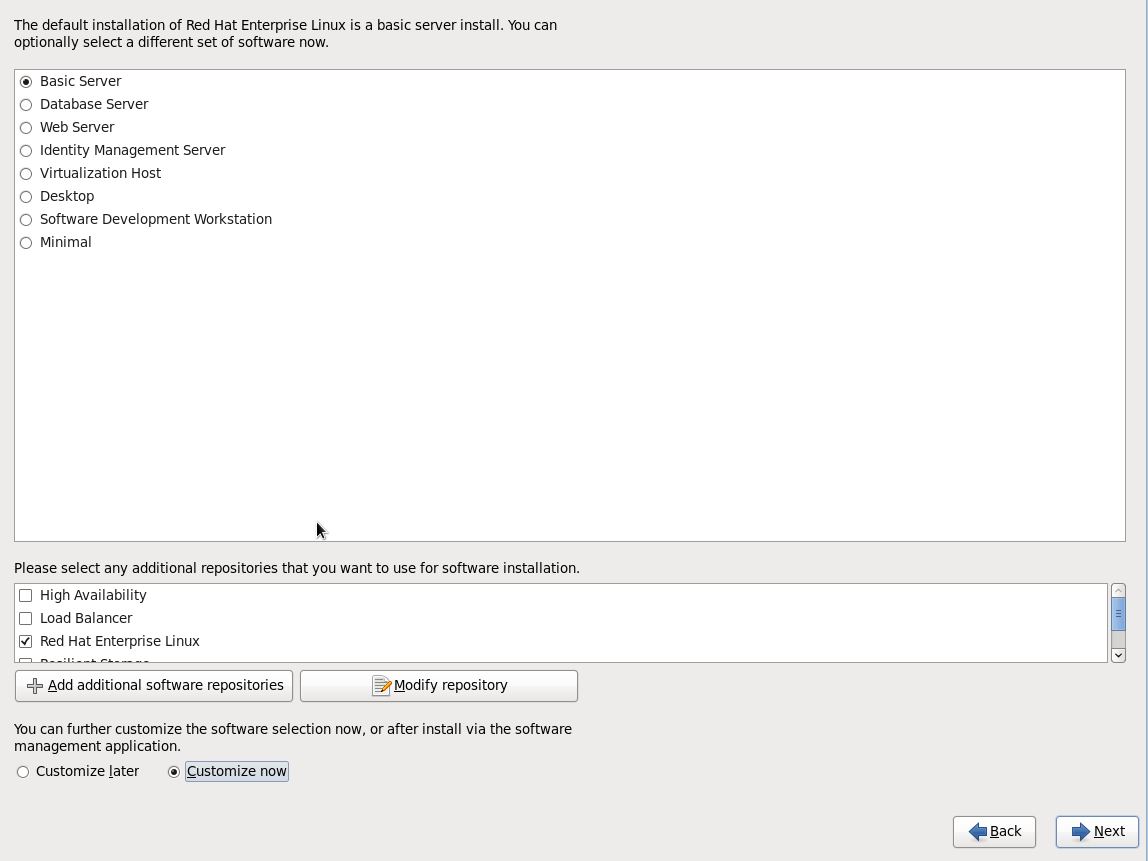


1. Click Write changes to disk and next
2. Configuring boot loader options, also can give boot loader password for security reason.

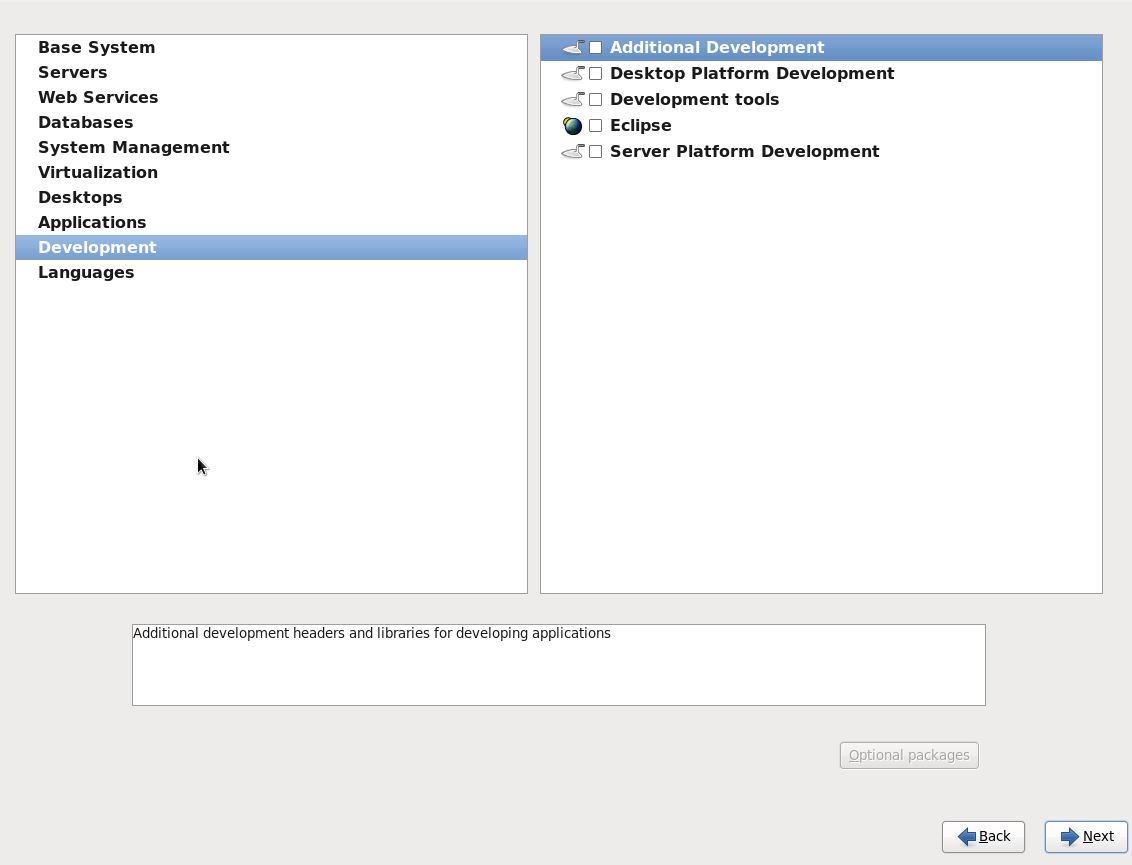


Click Next

1. Select applications to install and select customize now.

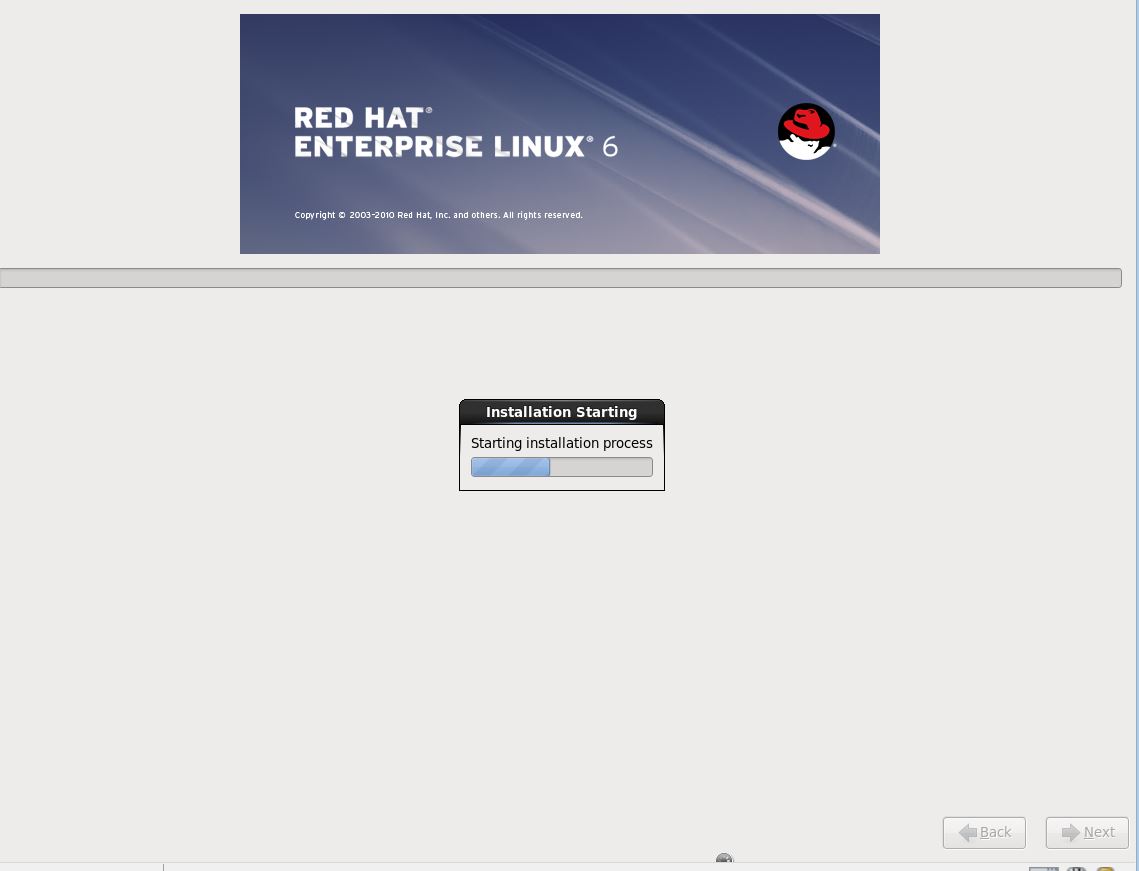


1. Click Customize now and next
2. Customize package selections.

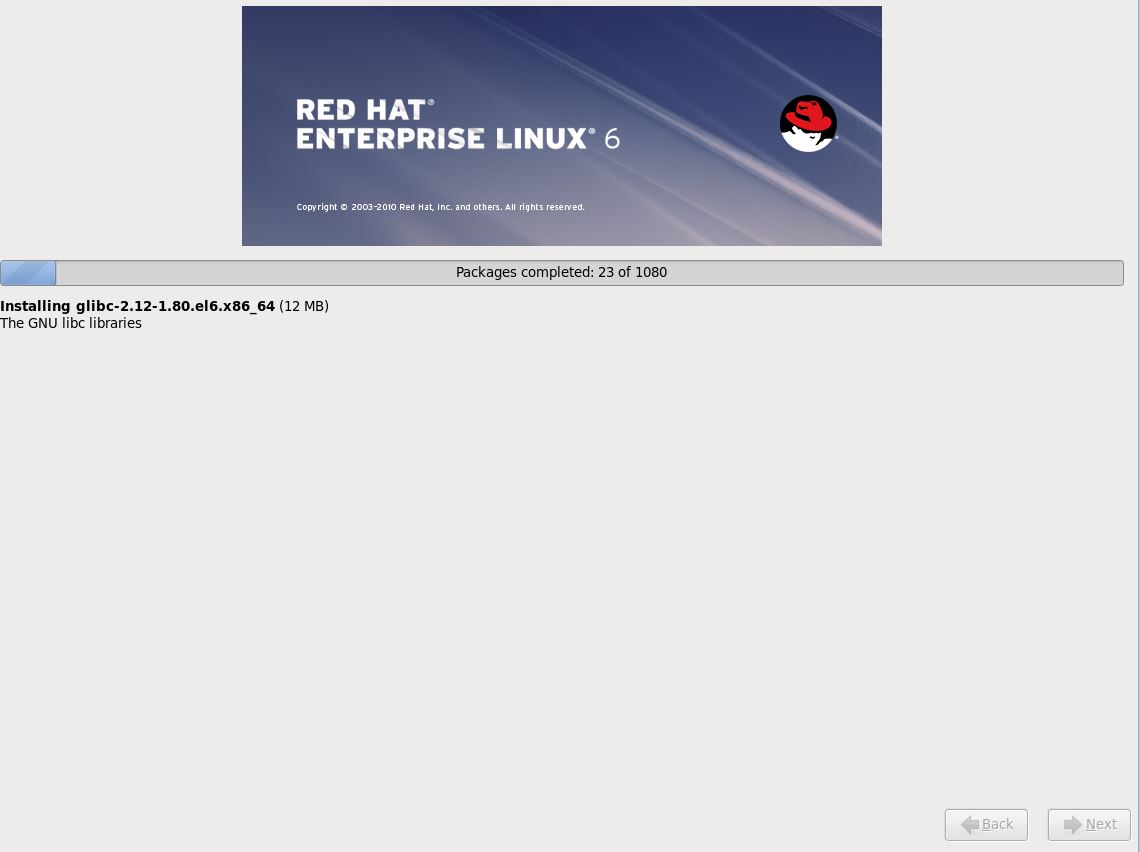


1. Select all required packages and click next

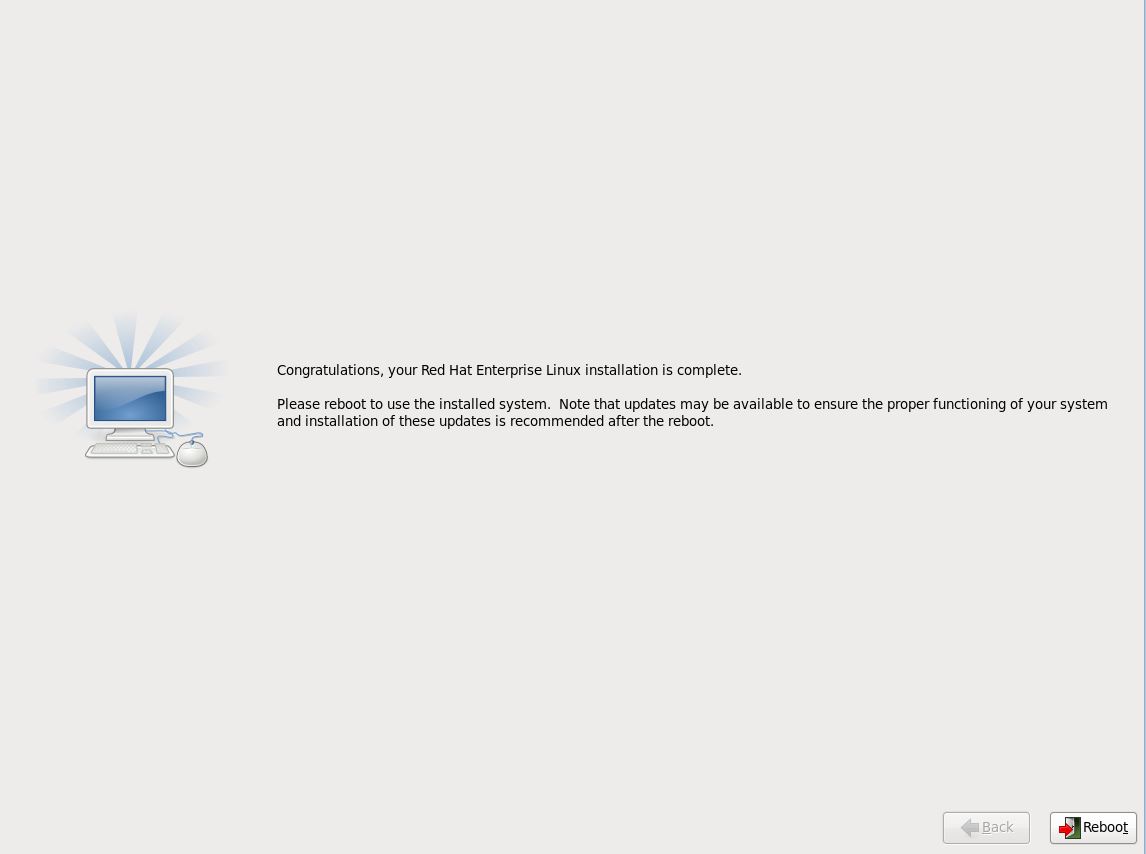
Installation progress.



1. Installation is completed successfully.



1. Please reboot your computer and login with root credentials.



1. Login Screen.



* + - 1. Image readiness for azure

Below are the steps for creating a virtual disk for uploading the image on azure

Step1: Uninstall NetworkManager

Step2: Create a file named network in the /etc/sysconfig/ directory

Step3: Create a file named ifcfg-eth0 in the /etc/sysconfig/network-scripts/ directory

Step4: Move (or remove) udev rules to avoid generating static rules for the Ethernet

interface. These rules cause problems when cloning a virtual machine in Microsoft Azure or Hyper-V

Step5: Ensure the network service will start at boot time

Step6: Install the drivers for the Linux Integration Services

Step7: Add the following line to /etc/yum.conf:

Step8: Modify the kernel boot line in your grub configuration to include additional kernel

parameters for azure

Step9: How to Add an internet connected network adapter through the VM’s Hyper-V settings

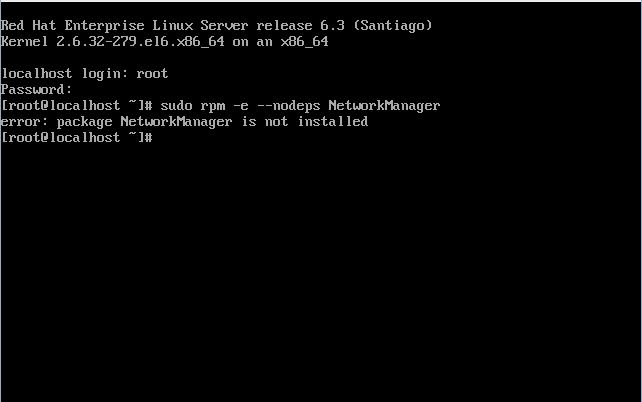
Step10: How to install WALinuxAgent

Step11:Changes to be made on Waagent

Step12: Deprovision the VM (equivalent so sysprep of Windows):

1.Uninstall NetworkManager by running the following command.

# sudo rpm -e --nodeps NetworkManager



**Note:** If the package is not already installed, this command will fail with an error message. This is expected.

2. Create a file named **network** in the /etc/sysconfig/ directory that contains the following text:



# NETWORKING=yes

#HOSTNAME=localhost.localdomain

Press the button I and type the above text



Once the above text is enter save and exit :wq!

3. Create a file named **ifcfg-eth0** in the /etc/sysconfig/network-scripts/ directory that contains the following text.

DEVICE=eth0

ONBOOT=yes

BOOTPROTO=dhcp

TYPE=ethernet

USERCTL=no

PEERDNS=yes

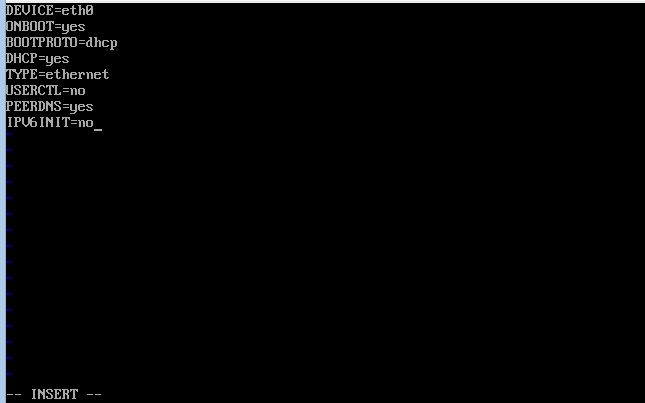
IPV6INIT=no

# cd /etc/sysconfig/network-scripts

Use the following command **vi ifcfg-eth0** to enter the above text



Press the button I and type the above text



Once the above text is enter save and exit :wq!

4. Move (or remove) udev rules to avoid generating static rules for the Ethernet interface. These rules cause problems when cloning a virtual machine in Microsoft Azure or Hyper-V:

Type the command

sudo mkdir –m 0700 /var/lib/waagent



Move the file lib/udev/rules.d/75-persistent-net-generator.rules to /var/lib/waagent



5. Ensure the network service will start at boot time by running the following command:

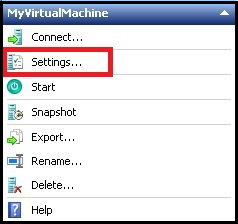


6.**Redhat Linux OS 6.3 & below**: Install the drivers for the Linux Integration Services

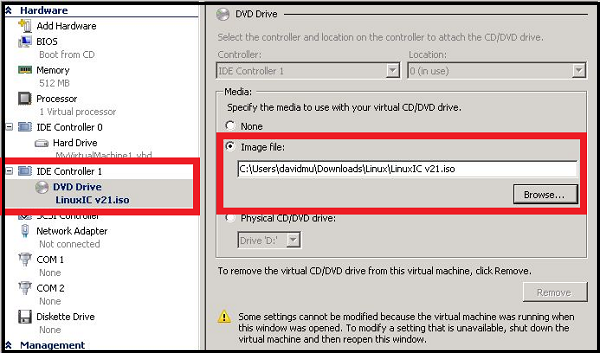
**Important: The step is only valid for Redhat Linux OS 6.3 and earlier.** In Redhat LinuxOS 6.4+ the Linux Integration Services are already available in the kernel.

a) Obtain the .iso file that contains the drivers for the Linux Integration Services from the [Microsoft Download Center](http://www.microsoft.com/download/details.aspx?id=41554).

b) In Hyper-V Manager, in the Actions pane, click Settings.



c) In the Hardware pane, click IDE Controller 1.



d) In the IDE Controller box, click DVD Drive, and then click Add.

e) Select Image file, browse to Linux IC v3.2.iso, and then click Open.

f) In the Settings page, click OK.

g) Click Connect to open the window for the virtual machine.

h) In the Command Prompt window, type the following commands:

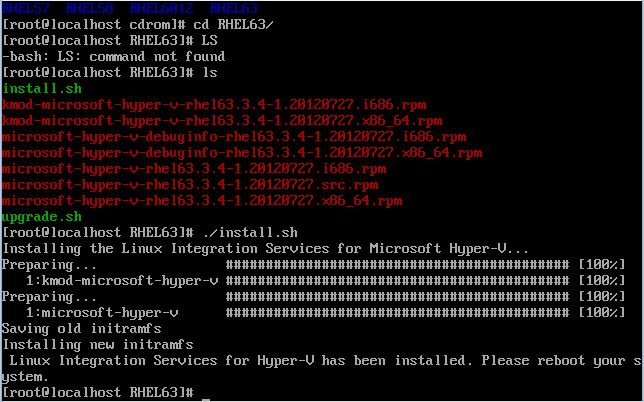
# mount -t auto /dev/dvd /mnt/cdrom

# cd /mnt/cdrom

# cd RHEL63

# ./install

# reboot

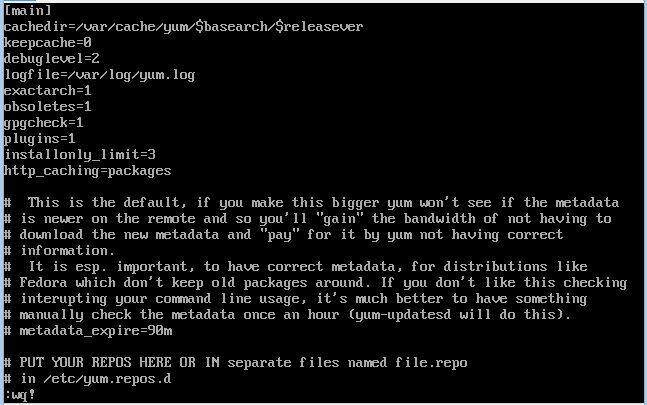


7.Add the following line to /etc/yum.conf

#http\_caching=packages



Press the button I and type the above text



Once the above text is entered save and exit :wq!

8. Modify the kernel boot line in your grub configuration to include additional kernel parameters for Azure. To do this open "/boot/grub/menu.lst" in a text editor and ensure that the default kernel includes the following parameters.

This will also ensure all console messages are sent to the first serial port, which can assist Azure support with debugging issues. This will disable NUMA due to a bug in the kernel version used by RedhatOS 6.

**Remove the below parameters**

# rhgb quiet crashkernel=auto

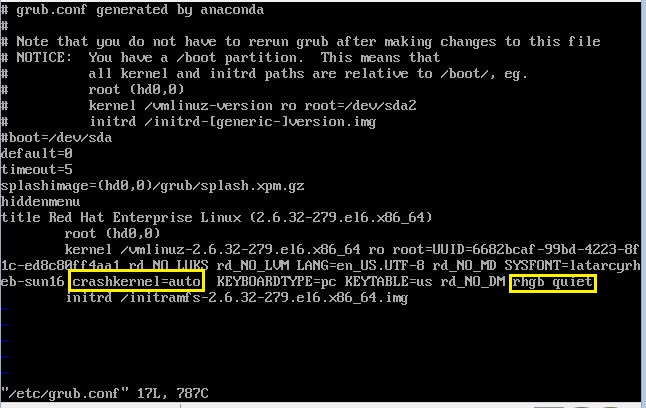
**Add the below parameters**

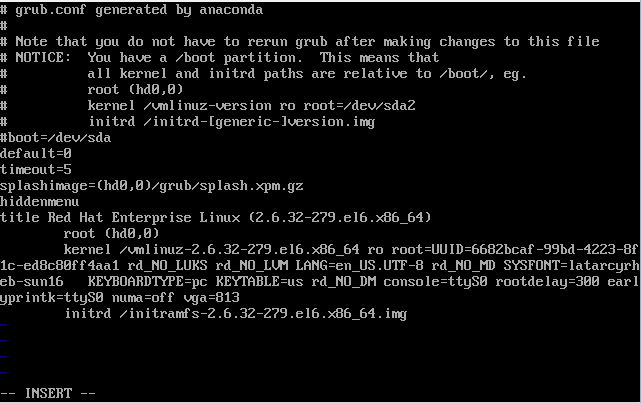
# console=ttyS0 earlyprintk=ttyS0 rootdelay=300 numa=off vga=813

Below is the command to do modifications on Kernel parameters

# vi /etc/boot/grub/menu.lst

Press the button I for removing and adding the above text



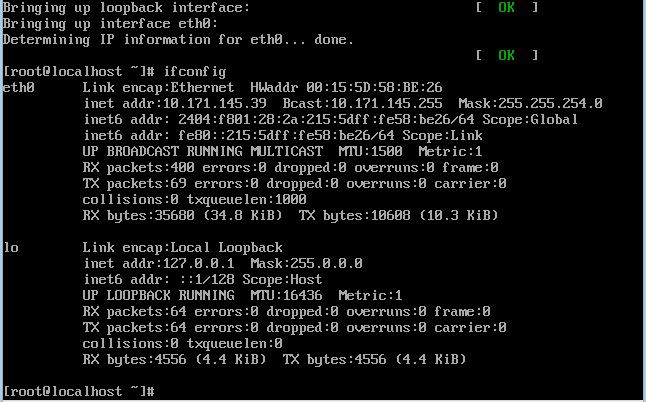


Once the above text is entered save and exit :wq!

9. Add an internet connected network adapter through the VM’s Hyper-V settings and restart the VM from Hyper-V Manager

Use the below command to find the **IPADDRESS** and connect to the server by using the IPADDRESS through SSH tools like WINSCP etc…..

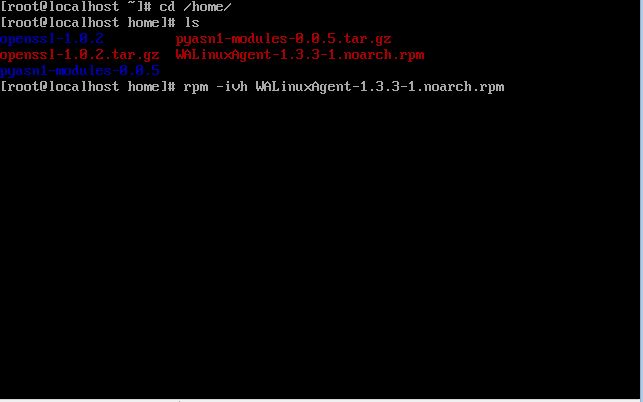
#ifconfig

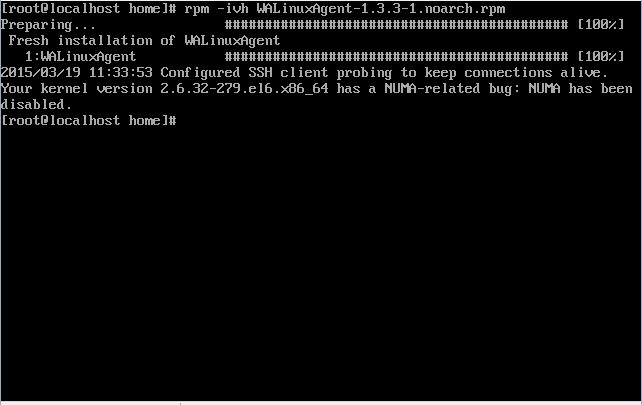


10. Download WALinuxAgent-1.3.3-1.noarch.rpm from belwo url http://olcentgbl.trafficmanager.net/openlogic/6/openlogic/x86\_64/RPMS/WALinuxAgent-1.3.3-1.noarch.rpm

# Install python-pyasn1, openssl and openssh “waagent” depends on it

# Install WALinuxagent-1.3.3-1.noarch.rpm using the command rpm –ivh WALinuxagent-1.3.3.1.noarch.rpm





Note that installing the WALinuxAgent package will remove the NetworkManager and NetworkManager-gnome packages if they were not already removed as described in step 1.

11. Do not create swap space on the OS disk

The Azure Linux Agent can automatically configure swap space using the local resource disk that is attached to the VM after provisioning on Azure. Note that the local resource disk is a temporary disk, and might be emptied when the VM is deprovisioned. After installing the Azure Linux Agent (see previous step), modify the following parameters in /etc/waagent.conf appropriately:

#vi /etc/waagent.conf

#ResourceDisk.Format=y

#ResourceDisk.Filesystem=ext4

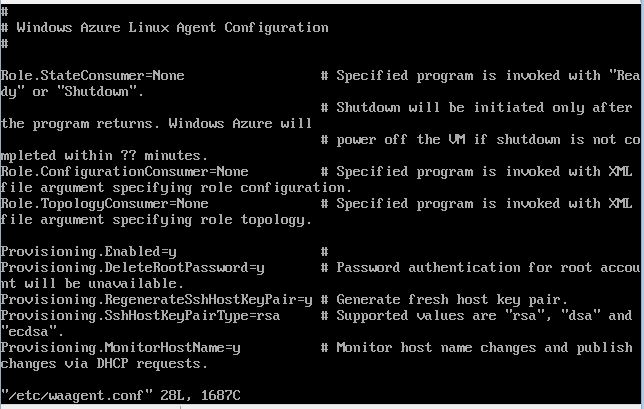
#ResourceDisk.Mountpoint=/mnt/resource

#ResourceDisk.EnableSwap=y

#ResourceDisk.SwapSizeMB=2048 ## NOTE: set this to whatever you need it to be.



Press the button I and type the above text



Once the above text is entered save and exit :wq!

12. Run the following command sequence to deprovision the VM (equivalent so sysprep of Windows):

# waagent –force –deprovision

# export HISTSIZE=0

# logout

13. Click Action -> Shut Down in Hyper-V Manager. Your Linux VHD is now ready to be uploaded to Azure.

1. How to use this offering

This section explains how you can leverage this offering for the scenarios explained in section 6. It explains the resources provided by this offering, how to setup your Azure subscription for automation and how to run the runbooks.

* 1. Resources

The resources provided by this offering are organized in 3 folders at the top level. Those are,

1. Runbooks
2. Files
3. Templates.
   * 1. Runbooks

This folder contains the Azure Runbooks that will automate the real time dev test scenarios. These are powershell workflow scripts with .ps1 extension. You can directly import these into Azure as runbooks.

There are child and parent runbooks. The child runbooks are re-usable components and are used by the parent runbooks. These child runbooks are put under ‘ChildRunbooks’ sub folder. The parent runbooks are the ones that actually perform the scenarios. These runbooks call the child runbooks as and when required. The parent runbooks are put directly under the Runbooks folder.

When you import runbooks into Azure manually, you need to upload the child runbooks first and then upload the parent runbooks. This ensures that when the parent runbooks are run, the child runbooks are already loaded and recognized by Azure.

* + 1. Files

This folder contains files that will be used by the runbooks. All these files should be uploaded to Azure storage account for the runbooks to use them. There is a runbook named “Upload-VSOFilesToAzureStorage” that will do this automatically for you. We will explain that runbook in section 7.3.9.

* + 1. Templates

This folder contains the tested ARM templates for some common workloads as an example that can be used after modifying the parameter JSON file. If your application is related to them, you can take these as reference and build ARM template for your application deployment in Azure. These templates are just for reference and are not used by any of the runbooks.

* 1. Setting up your Azure subscription for automation

For the runbooks to work correctly, you need to complete few tasks in your Azure subscription.

1. Upload ARM powershell module to your Azure automation account
2. Create an OrgId in your Azure active directory
3. Create a credential asset for the OrgId
   * 1. Upload ARM powershell module

The runbooks provided by this offering use Azure resource manager powershell module to manage Azure resources. As of this document creation date, the Azure resource manager powershell module is not shipped out of the box with the new Azure portal (<http://portal.azure.com> ). You need to import the module manually to your Azure subscription for the runbooks to work.

The ARM powershell module is available as part of this offering as “AzureResourceManager.zip”. To upload this to your automation account, go to Assets in your automation account -> click on Modules -> Add a Module -> Upload “AzureResourceManager.zip” and click Ok.

* + 1. Create OrgId in your Active Directory

To authenticate to Azure subscription using Azure automation, we are using Azure active directory organizational identity credentials based authentication. For this to work, you need to create organization identity (OrgId) in your Azure active directory. Note: Microsoft account – formerly known as Live IDs – will not work in Azure automation.

1. Log in to the Azure service management portal (<http://manage.windowsAzure.com> ) as the service administrator for the Azure subscription you want to manage using Azure Automation.
2. In the Azure portal click on Active Directory service.
3. Click the directory name that is associated with this Azure subscription.
4. Click on the Users tab and then click the Add User button.
5. For type of user, select “New user in your organization.” Enter a username for the user to create.
6. Fill out the user’s profile. For role, pick “User.” Don’t enable multi-factor authentication. Multi-factor accounts cannot be used with Azure Automation. Click Create.
7. The temporary password will be shown. Login to Azure using this credentials and change the password for the user.

The OrgId should be allowed to manage your subscription.

1. From the service management portal, click on Settings.
2. Click on Administrator tab
3. Click the Add button. Type the full user name (including part after @ symbol) of the Azure Active Directory user you want to set up to manage Azure. For subscriptions, choose the Azure subscriptions you want this user to be able to manage. Click the check mark.
   * 1. Create OrgId Credential Asset

Create an Azure Automation credential asset containing the username and password of the Azure Active Directory user that you have just created. You can create a credential asset in Azure Automation by clicking into an Automation Account and then clicking the Assets tab, then the Add Setting button.

All the runbooks will use this credential asset to manage your subscription. You need to pass the name of the credential you just created above to ORGIDCredential parameter of the runbooks.

* 1. Runbooks explained

This section explains all the parent runbooks provided by this offering, the parameters they accept, the files and child runbooks they depend on, example usage and implications, if any.

* + 1. Deploy-ResourceGroup

This runbook deploys Azure resources to an Azure resource group through ARM template. The resources to be deployed should be formatted in an ARM template and uploaded to an Azure storage account. This runbook accepts the uri to the uploaded ARM template and deploys the resources as designed in the template.

If the resource group doesn't exist, it will be created automatically. Else it will be updated.

This runbook has a dependency on Connect-Azure runbook. The Connect-Azure runbook must be published for this runbook to run correctly.

**PARAMETERS**

This runbook accepts the following parameters.

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

. PARAMETER ResourceGroupName

Name of the resource group where the resources should be deployed to.

If the resource group doesn't exist, it will be created automatically. Else it will be updated.

. PARAMETER ResourceGroupLocation

The location of the resource group

. PARAMETER DeploymentName

The name for the current deployment

. PARAMETER TemplateFile

The Uri of the ARM template file stored in Azure storage account.

. PARAMETER TemplateParameterFile

The Uri of the ARM template parameters file stored in Azure storage account.

**EXAMPLE**

Deploy-ResourceGroup -OrgIDCredential "AutomationUser" -ResourceGroupName "WebAppDevTeam" -ResourceGroupLocation "West US" -DeploymentName "InitialDeployment" -TemplateFile "https:// storageaccount.blob.core.windows.net/dsc/Azuredeploy.json" -TemplateParameterFile https:// storageaccount.blob.core.windows.net/dsc/Azuredeploy.parameters.json"

* + 1. Stop-AzureVMs

This runbook stops all the VMs in a resource group.

This runbook has a dependency on Connect-Azure runbook. The Connect-Azure runbook must be published for this runbook to run correctly.

**PARMATERS**

This runbook accepts the following parameters.

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

. PARAMETER ResourceGroupName

Name of the resource group that contains the VM to be stopped

**EXAMPLE**

Stop-AzureVMs -OrgIDCredential "AutomationUser" -ResourceGroupName "WebAppDevTeam"

* + 1. Start-AzureVMs

This runbook starts all the VMs in a resource group.

This runbook has a dependency on Connect-Azure runbook. The Connect-Azure runbook must be published for this runbook to run correctly.

**PARAMETERS**

This runbook accepts the following parameters

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & password for the user who is having access to the Azure subscription.

. PARAMETER ResourceGroupName

Name of the resource group that contains the VM

**EXAMPLE**

Start-AzureVMs -OrgIDCredential "AutomationUser" -ResourceGroupName "WebAppDevTeam"

* + 1. Backup-AzureVM

This runbook backs up the vhds (both OS and data disks) and VM configuration of the input VM to a Backup storage container. This accepts a ‘BackupSuffix parameter which is used to name the backed up files and to identify the backed up files while restoring.

The vhds of the VM are snapshoted and copied to the Backup container following the naming convention “ResourceGroupName-VMName-vhdname-BackupSuffix.vhd". The VM configuration is saved as a JSON file with naming convention "ResourceGroupName-VMName-BackupSuffix.json"

Apart from these, a VM configuration parameter JSON file is also generated with the VM configuration values and copied with naming convention "VMResourceGroupName-VMName-ParamsTemplate-BackupSuffix.json". This file will be required when cloning the VM.

The VMs are shutdown forcefully before being backed up. It is recommended that you shut down the VMs before backing up.

The backup storage can belong to a separate resource group. We learnt from our tests that if the backup storage is in a different resource group (even in the same Azure location), it takes a couple of hours when the runbook copies the backed up vhds to the storage account, depending on the vhd size, the number of vhds and Azure location etc., but it works. We recommend that you take this into consideration when you plan your VM backup strategy.

This runbook has dependency on the runbooks Connect-Azure, Write-Log, Copy-FileFromAzStorage, Copy-FileToAzStorage, Copy-VHD and Get-TargetName. All these runbooks must be published first for this runbook to work.

**PARAMETERS**

This runbook accepts the following parameters

. PARAMETER SourceVMResourceGroupName

Name of the resource group of the source VM

. PARAMETER SourceVMName

Name of the source VM to be backed up

. PARAMETER backupStorageAccountRG

Name of the resource group that contains the backup storage account.

. PARAMETER backupStorageAccount

Name of the backup storage account.

. PARAMETER backupContainer

Name of the backup storage account container. The backed up vhds and json files will be stored in this container.

. PARAMETER backupSuffix

Unique name/label for the backup. This unique label will be appended to the backup, and will be used to identify the backup during restoring process.

. PARAMETER ParametersTemplateName

Name of the template to be used for parameters to an Azure ARM template for new VM creation. This is a fixed generic template delivered as part of this offering “VMTemplate.json” and should be present in the backup container.

. PARAMETER SubscriptionName

Name of the target Azure subscription.

. PARAMETER OrgIDCredential

Name of the orgid credential asset. This asset contains the credential for the user that should have access to the target Azure subscription.

**EXAMPLE**

Backup-AzureVM -SourceVMResourceGroupName "WebAppDevTeam" -SourceVMName "SqlVM1" -backupStorageAccountRG "RepositoryRG" -backupStorageAccount "RepositoryStorage" -backupContainer "VMBackupFiles" -backupSuffix "Drop2" -ParametersTemplateName "VMTemplate.json" -SubscriptionName "AzureSubscription" -OrgIDCredential "OrgId"

* + 1. Restore-AzureVM

This runbook restores or clones an Azure VM already backed by the runbook Backup-AzureVM to a backed up state. The backed up state is identified by the parameter ‘BackupSuffix’.

**Restore Mode:**

For restore mode, set the ‘*CloneToASeperateStorage’* parameter to ‘No’. In this mode, the existing VM is deleted and a new one is created in its place with the target state of the VM.

In this mode, the runbook identifies the backup files by the unique backupSuffix, copies the associated VHDs back to the VMs original container, deletes the VM and recreates it with the values of the Parameter template and VM backup config JSON file.

**Clone Mode:**

For Clone mode, set the ‘*CloneToASeperateStorage’* parameter to ‘No’. In this mode, the existing VM is not deleted. A new VM is created with the target state in the same or a new resource group.

The VM configuration parameter file ("*VMResourceGroupName-$VMName-ParamsTemplate-BackupSuffix.json*") that was created while backing up needs to be updated with the properties of the new VM. These properties include the name of the new VM, storage account that will hold the vhds of the new VM and NIC details. NIC will not be created new; a pre-created NIC name should be given. The updated configuration file can be saved in any name in the backup container and the name must be passed to the parameter ‘*ParametersTemplateName*’.

In this mode, the runbook copies the vhds to the mentioned storage account and creates a new VM with the configuration and properties as given by the configuration and parameter JSON files respectively.

This runbook has dependency on the runbooks Connect-Azure, Write-Log, Copy-FileFromAzStorage, Copy-FileToAzStorage, Copy-VHD and Get-TargetName. All these runbooks must be published first for this runbook to work.

**PARAMETERS**

The runbook accepts the following parameters.

. PARAMETER SourceVMResourceGroupName

Name of the resource group that contains VM to be restored.

. PARAMETER SourceVMName

Name of the VM that needs to be restored.

. PARAMETER backupStorageAccountRG

Name of the resource group of the backup storage account.

. PARAMETER backupStorageAccount

Name of the backup storage account.

. PARAMETER backupContainer

Name of the backup storage account container.

. PARAMETER backupSuffix

The target backup state to be restored to. The VM will be restored to this state of the VM.

. PARAMETER NewVMTemplateName

Name of the VM configuration files saved during back up. <VMResourceGroup>-<VMName>-<BackupSuffix>.json

. PARAMETER CloneToASeperateStorage

"Yes" for clone mode. "No" for restore mode.

. PARAMETER ParametersTemplateName

Name of the parameters JSON file updated with the properties of the new VM.

This is required only in the case of cloning the VM to a different location

. PARAMETER DestVMResourceGroupName

Name of the resource group where the new VM will be created under

. PARAMETER SubscriptionName

Name of the target Azure subscription.

. PARAMETER OrgIDCredential

Name of the orgid credential asset. This asset contains the credential for the user that should have access to the target Azure subscription.

**EXAMPLE**

Example usage for cloning a VM

Restore-AzureVM -SubscriptionName "AzureSubscription" -SourceVMResourceGroupName "WebAppDevTeam" -SourceVMName "SqlVM1" -backupStorageAccountRG "RepositoryRG" -backupStorageAccount "RepositoryStorage" -backupContainer "VMBackupFiles" -backupSuffix "Drop2" -NewVMTemplateName "<savedfile>.json" -CloneToASeperateStorage "Yes" -ParametersTemplateName "ClonedSqlVMParams.json" -DestVMResourceGroupName "WebAppDevTeam2" –ORGIDCredential “AutomationUser”

Example usage for restoring a VM

Restore-AzureVM -SubscriptionName "AzureSubscription" -SourceVMResourceGroupName "WebAppDevTeam" -SourceVMName "SqlVM1" -backupStorageAccountRG "RepositoryRG" -backupStorageAccount "RepositoryStorage" -backupContainer "CommonFiles" -backupSuffix "Drop2" -NewVMTemplateName "<savedfile>.json" -CloneToASeperateStorage "No" -ParametersTemplateName "ClonedSqlVMParams" –ORGIDCredential “AutomationUser”

* + 1. Generate-RDCManFile

This runbook generates a remote desktop connection manager file to manage remote connections to all the VMs in a resource group. The generated file will be uploaded to the given storage account. With the file, you can manage all VMs’ remoted connections in a single place.

The file generated has the same credentials for all the VMs in the generated file. If each VM has different credentials, which is likely, the credentials can be changed in the remote desktop connection manager when opening the file.

This runbook has a dependency on Connect-Azure runbook. The Connect-Azure runbook must be published for this runbook to run correctly.

**PARAMETERS**

This runbook accepts the following parameters

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & password for the user who is having access to the Azure subscription.

. PARAMETER ResourceGroupName

Name of the resource group that contains the VMs to be managed

. PARAMETER StorageAccountName

Name of the storage account where the generated file should be uploaded

. PARAMETER StorageContainerName

Name of the storage container where the generated file should be uploaded

. PARAMETER AdminUser

Name of the admin user for the VMs. If the VMs have difference credentials, you can give admin name for any one of the VM. The file will be generated with the same credentials for all VMs. You can then later change the credentials after opening the file.

. PARAMETER AdminPwd

Password for the user account given in AdminUser parameters. If the VMs have difference credentials, you can give credentials for any one of the VM. The file will be generated with the same credentials for all VMs. You can then later change the credentials after opening the file.

. PARAMETER Domain

Domain name for the user account given in AdminUser parameter. If the VMs have difference credentials, you can give credentials for any one of the VM. The file will be generated with the same credentials for all VMs. You can then later change the credentials after opening the file.

**EXAMPLE**

Generate-RDCManFile -OrgIDCredential "AutomationUser" -ResourceGroupName "WebAppDevTeam" -StorageAccountName "WebAppDevStorage" -StorageContainerName "managementfiles" -AdminUser "webadminstrator" -AdminPwd "P@ssw0rd" -Domain "appdev"

* + 1. Configure-DSCPullServer

This runbook configures an Azure VM as a DSC pull server. Having a DSC pull server will be very useful when you want to push configurations to multiple VMs in your resource group. You can configure one of the VMs in your resource group as a pull server using this runbook and use this have other VMs pull configurations from this server.

This runbook uses a powershell script SetupDSCPullServer.ps1 to install the updates. This script file should be uploaded to an Azure storage container, the details of which should be passed as parameters. This powershell script is copied into the VM and executed via Set-AzureVMCustomExtension Azure powershell cmdlet. We use custom script extension for executing commands inside the Azure VM rather than running commands via remote powershell session. The reason behind this is explained in section 8.2.1.

This runbook has a dependency on Connect-Azure and Run-ScriptInAzureVM runbooks. Both these runbooks must be published for this runbook to run correctly.

**PARAMETERS**

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

. PARAMETER VMResourceGroup

Name of the resource group that contains the VM to be configured

. PARAMETER VMName

Name of the Azure VM to be configured

. PARAMETER DSCInstallerResourceGroup

Name of the resource group that contains the storage account where the script file SetupDSCPullServer.ps1 is uploaded

. PARAMETER DSCInstallerStorageAccount

Name of the storage account where the script file SetupDSCPullServer.ps1 is uploaded

. PARAMETER DSCInstallerStorageContainer

Name of the storage container where the script file SetupDSCPullServer.ps1 is uploaded

. PARAMETER DSCInstallerFile

The script file that configures the DSC pull server. "SetupDSCPullServer.ps1" is the script file name.

**EXAMPLE**

Configure-DSCPullServer -ORGIDCredential "AutomationUser" -VMResourceGroup "WebAppDevTeam" -VMName "DSC-Pullserver" -DSCInstallerResourceGroup "RepositoryRG" -DSCInstallerStorageAccount "RepositoryStorage" -DSCInstallerStorageContainer "DevTestFiles" -DSCInstallerFile "SetupDSCPullServer.ps1"

* + 1. Install-WindowsUpdate

This runbook Installs windows update on all VMs in a resource group. By default, the VMs will be restarted automatically after the windows update if required. If required otherwise, you can set the optional parameter “*AutoReboot*” to ‘*No*’

This runbook uses a powershell script InstallWindowsUpdateLocally.ps1 to install the updates. This script file should be uploaded to an Azure storage container, the details of which should be passed as parameters. This powershell script uses the windows update powershell module provided by 'Michael Gajda' at <https://gallery.technet.microsoft.com/scriptcenter/2d191bcd-3308-4edd-9de2-88dff796b0bc> This powershell script is copied into the VM and executed via Set-AzureVMCustomExtension Azure powershell cmdlet. We use custom script extension for executing commands inside the Azure VM rather than running commands via remote powershell session. The reason behind this is explained in section 8.2.1.

This runbook has a dependency on Connect-Azure and Run-ScriptInAzureVM runbooks. Both these runbooks must be published for this runbook to run correctly.

**PARAMETERS**

This runbook accepts the following parameters

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

. PARAMETER ResourceGroup

Name of the resource group that contains the VMs to be updated

. PARAMETER StorageAccountResourceGroup

Name of the resource group that contains the storage account where the script file SetupDSCPullServer.ps1 is uploaded

. PARAMETER StorageAccount

Name of the storage account where the script file InstallWindowsUpdateLocally.ps1 is uploaded

. PARAMETER StorageContainer

Name of the storage container where the script file InstallWindowsUpdateLocally.ps1 is uploaded

. PARAMETER InstallerFile

The script file that installs the windows update. "InstallWindowsUpdateLocally.ps1" is the script file name.

. PARAMETER WindowsUpdateModuleFileUri

"https://gallery.technet.microsoft.com/scriptcenter/2d191bcd-3308-4edd-9de2-88dff796b0bc/file/41459/43/PSWindowsUpdate.zip"

**EXAMPLE**

Install-WindowsUpdate -ORGIDCredential "AutomationUser" -ResourceGroup "WebAppDevTeam" -StorageAccountResourceGroup "RepositoryRG" -StorageAccount "RepositoryStorage" -StorageContainer "CommonFiles" –InstallerFile "InstallWindowsUpdateLocally.ps1" -WindowsUpdateModuleFileUri "https://gallery.technet.microsoft.com/scriptcenter/2d191bcd-3308-4edd-9de2-88dff796b0bc/file/41459/43/PSWindowsUpdate.zip" -AutoReboot "Yes"

* + 1. Copy-VSOFilesToAzureVM

This runbook copies files from VSO git repository to an Azure VM. All files under the given folder and its subfolders in the VSO git path are copied to the given path in the Azure VM.

This runbook uses a powershell script Get-FilesFromVSO.ps1 to copy the files from VSO git. This script file should be uploaded to an Azure storage container, the details of which should be passed as parameters.

This powershell script is copied into the VM and executed via Set-AzureVMCustomExtension Azure powershell cmdlet. We use custom script extension for executing commands inside the Azure VM rather than running commands via remote powershell session. The reason behind this is explained in section 8.2.1.

This runbook requires a VSO Alternate Authentication Credential for connecting with VSO-Git repository. That credentials should be stored in an automation credential asset and passed as parameter to this runbook.

This runbook has a dependency on Connect-Azure runbook. The Connect-Azure runbook must be published for this runbook to run correctly.

**PARAMETERS**

This runbook accepts the following parameters

. PARAMETER VSOCredentialName

Name of the credential asset containing the VSO Alternate Authentication Credential name and password configured from VSO Profile dialog.

. PARAMETER VSORepository

Name of the repository that contains the runbook project

. PARAMETER VSOProject

Name of the VSO project that contains the repository

. PARAMETER VSOBranch

Optional name of the Git branch to retrieve the runbooks from. Defaults to "master"

. PARAMETER VSOFolderPath

Path to the folder from where the files should be copied from. Ex. /Project1/ProjectRoot

. PARAMETER DownloadFolderPath

The path in the VM to which the files are to be copied to

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service.This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

. PARAMETER ResourceGroup

Name of the resource group that contains the VM

. PARAMETER VMName

Name of the Azure VM

. PARAMETER StorageAccountResourceGroup

Name of the resource group that contains the storage account where the script file Get-FilesFromVSO.ps1 is uploaded

. PARAMETER StorageAccount

Name of the storage account where the script file Get-FilesFromVSO.ps1 is uploaded

. PARAMETER StorageContainer

Name of the storage container where the script file Get-FilesFromVSO.ps1 is uploaded

. PARAMETER ScriptFile

The script file that configures the DSC pull server."Get-FilesFromVSO.ps1" is the script file name.

. PARAMETER SubscriptionName

Name of the Azure subscription.

**EXAMPLE**

Remove-ResourceGroup -OrgIDCredential "OrgId" -ResourceGroup "SqlDevTeam"

* + 1. Remove-ResourceGroup

This runbook removes an Azure resource group and all its resources.

This runbook has a dependency on Connect-Azure runbook. The Connect-Azure runbook must be imported and published first for this runbook to work.

**PARAMETERS**

This runbook accepts the following parameters.

. PARAMETER OrgIDCredential

Name of the credential asset containing the OrgId credentials name and password that is the co-administrator of the subscription

. PARAMETER ResourceGroup

Name of the resource group to be removed

**EXAMPLE**

Remove-ResourceGroup -OrgIDCredential "OrgId" -ResourceGroup "SqlDevTeam"

* + 1. Upload-VSOFilesToAzureStorage

This runbook uploads files from a VSO git repository location into an Azure storage account. This runbook traverses through all folders and sub folders in the repository location and upload the files to the storage account. The folder structure hierarchy doesn’t apply to storage account containers and so all files are stored at the same level under the container as blobs.

This runbook has a dependency on Connect-Azure runbook. The Connect-Azure runbook must be published for this runbook to run correctly.

**PARAMETERS**

This runbook accepts the following parameters.

. PARAMETER VSOCredentialName

Name of the credential asset containing the VSO Alternate Authentication Credential name and password configured from VSO Profile dialog.

. PARAMETER VSOAccount

Name of the account name for VSO Online. Ex. https://accountname.visualstudio.com

. PARAMETER VSOProject

Name of the VSO project that contains the repository

. PARAMETER VSORepository

Name of the repository that contains the runbook project

. PARAMETER VSOjsonFolderPath

Folder path in the VSO repository

. PARAMETER AutomationAccount

Name of the Automation Account where the runbooks should be synced to

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service.

This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

. PARAMETER AzureSubscriptionName

Name of the Azure subscription

. PARAMETER VSOBranch

Optional name of the Git branch to retrieve the runbooks from. Defaults to "master"

**EXAMPLE**

Copy-VSOFilesToAzureVM -VSOAccount "managedservicesdev" -VSORepository "automationlib" -VSOProject "ManagedServices" -VSOBranch "master" -VSOFolderPath "/ProjectFiles" -DownloadFolderPath "C:\VSOFiles" -VSOCredentialName "vsoCredentialAmit" -ORGIDCredential "AutomationUser" -ResourceGroup "WebAppDevTeam" -VMName "WebFE2" -StorageAccountResourceGroup "CommonRG" -StorageAccount "ProjectFilesStorage" -StorageContainer "ProjectFiles" -ScriptFile "Get-FilesFromVSO.ps1" -SubscriptionName "AzureSubscription"

* + 1. Sync-AzureAutomationRunbooks

This runbook syncs all runbooks in a VSO git repository location to an Azure automation account starting with dependent (child) runbooks, followed by parent runbooks. The .ps1 files under the subfolders are treated as child runbooks. This runbook will traverse through all folders in the location, for each folder start from its sub folders, if any, all the way up to the parent folder and publish the .ps1 files as runbooks in the given Azure automation account.

This runbook is aimed at the phase where runbooks are under development and uploaded to VSO after update. Running this runbook will keep the Azure automation runbooks in sync with the latest changes on VSO.

When syncing the runbooks, the existing automation runbook will be deleted and a new one will be created in place. This job history of the runbook is also deleted in this process. However, if you want to keep your job history, you can set ‘DeleteExistingRunbook’ parameter to ‘no’, which will not delete the runbook at the expense of not syncing it.

This runbook has a dependency on another child runbook Connect-Azure, The Connect-Azure runbook must be published for this runbook to run correctly

**PARAMETERS**

The runbook accepts the following parameters.

. PARAMETER VSOCredentialName

Name of the credential asset containing the VSO Alternate Authentication Credential name and password configured from VSO Profile dialog.

. PARAMETER VSOAccount

Name of the account name for VSO Online. Ex. https://accountname.visualstudio.com

. PARAMETER VSOBranch

Optional name of the Git branch to retrieve the runbooks from. Defaults to "master"

. PARAMETER VSOProject

Name of the VSO project that contains the repository

. PARAMETER VSORepository

Name of the repository that contains the runbook project

. PARAMETER VSORunbookFolderPath

Project path to the root where the runbooks are located. Ex. /Project1/ProjectRoot where ProjectRoot contains the parent runbooks

. PARAMETER ResourceGroupName

Name of the resource group of the automation account.

. PARAMETER AutomationAccount

Name of the Automation Account where the runbooks should be synced to

. PARAMETER DeleteExistingRunbook

If "Yes", it deletes the existing runbook in the automation account runbook along will all the associated jobs and logs, and then imports the runbook from VSO.

If "No", it imports the runbook from VSO if it is not already existing in Automation account. If the runbook already exists, it does not overwrite it.

. PARAMETER AzureSubscriptionName

Name of the Azure subscription.

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

**EXAMPLE**

Sync-AzureAutomationRunbooks -VSOCredentialName "vsoCredentialAmit" -VSOAccount "managedservicesdev" -VSORepository "ManagedServices" -VSOBranch "master" -VSOProject "ManagedServices" -VSORunbookFolderPath "/SyncRunbooks" -ResourceGroupName "TestARMResourceGroup" -AutomationAccount "TestARMAccount" -DeleteExistingRunbook "Yes" -AzureSubscriptionName "Gopi -AzureSubscription" -OrgIDCredential "AutomationUser"

* + 1. Install-PackageOnLinuxVM

This runbook installs a package like JRE, Apache, MySQL, PHP etc on an existing Ubuntu VM. It uses Custom Script Extension for Linux to achieve this. The custom script extension for Linux can execute a Linux shell script on the VM. The code to install the package is written as a shell script and this script is stored in Azure Storage container. The uri of the script is passed to the custom script extension.

**Prerequisites:**

* An Ubuntu VM on Azure
* Linux shell script copied to an Azure Storage container

**PARAMETERS**

The runbook accepts the following parameters.

. PARAMETER VMRGName

Name of the Resource Group of the VM.

. PARAMETER VmName

Name of the Linux VM

. PARAMETER Location

Location of the VM. Please ensure the correctness of alphabet case and spacing. The correct location casing and spelling examples:

* "West US",
* "East US",
* "West Europe",
* "East Asia",
* "Southeast Asia"

. PARAMETER ExtensionName

Name of the Extension. Any name can be provided, but if the custom script extension is already run once, then this should be the same as previously provided in the first run.

. PARAMETER fileName

Name of the script.

Ex: "install\_LAMPandJRE.sh"

. PARAMETER scriptParameters

Input parameters for the script.

Ex: "Test@123"

. PARAMETER fileUri

Uri of the script blob for the script.

Ex: [http://devteststorageaccount1.blob.core.windows.net/scripts/ install\_LAMPandJRE.sh](http://devteststorageaccount1.blob.core.windows.net/scripts/%20install_LAMPandJRE.sh)

. PARAMETER StorageRGName

Name of the resource group of the storage account where the script is stored

. PARAMETER storageAccountName

Name of the storage account where the script is stored.

. PARAMETER SubscriptionName

Name of the Azure subscription.

. PARAMETER OrgIDCredential

Name of the Azure credential asset that was created in the Automation service. This credential asset contains the user id & passowrd for the user who is having access to the Azure subscription.

* + 1. Test-URL

This runbook tests a website by sending an HTTP request and validating the Http Response code from the server. In order to test if Apache Server is up and running, even if no website is deployed, just mention the server name or IP Address in the url, which will provide a response from the default page.

The runbook returns true if the Http Response Code is 200, and returns false in all other cases.

**PARAMETERS**

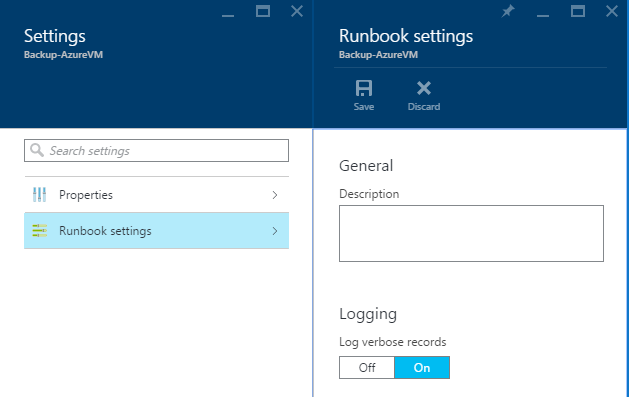
The runbook accepts the following parameters.

. PARAMETER URL

URL to be tested. In case apache server needs to be tested, url would be “http://<serverName or IP>”

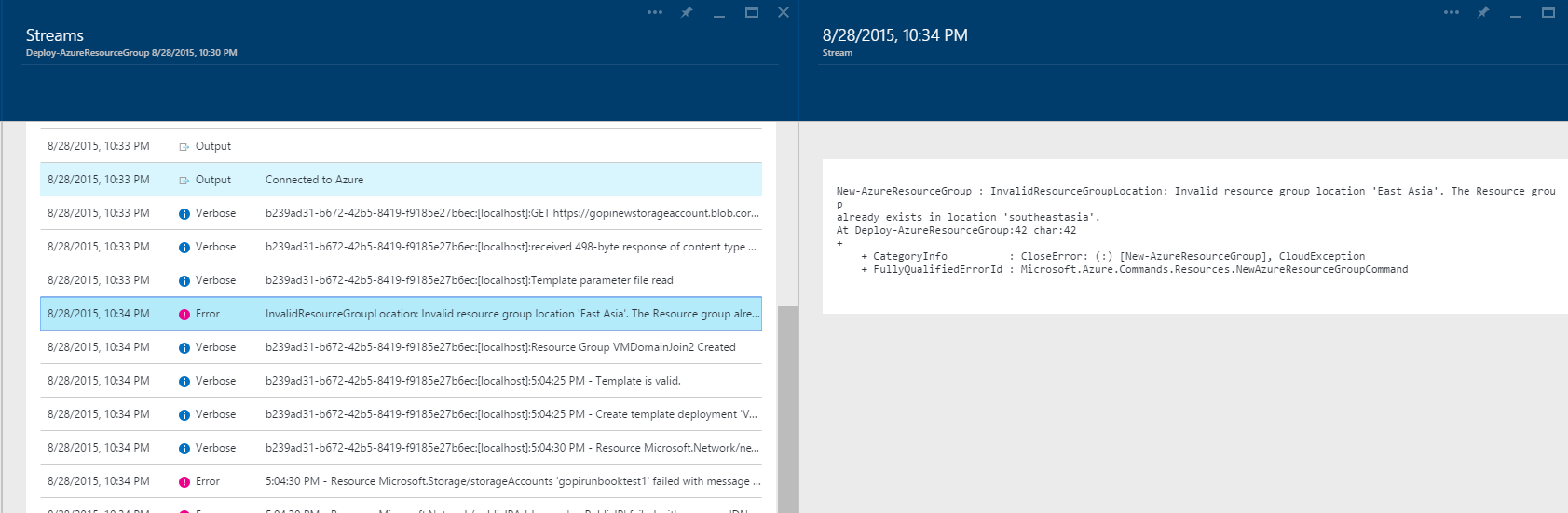
1. Appendix
   1. View runbooks logs

You need to enable verbose logging for your automation runbooks to view the logs. To enable this, go to All Settings of the runbook -> Runbook Settings -> Turn “On” log verbose records. -> Save.



Note that for runbooks added by ‘Sync-AzureAutomationRunbooks.ps1’ runbook, this will be automatically set.

Once you enable this setting, you can view the real time logs by clicking on the stream tile of the runbook jobs. This should give verbose level logging and error details and exception if any. This should give you an idea of what went wrong if the runbook fails.



* 1. Design Decisions
     1. Using custom script extensions

For running powershell scripts inside an Azure VM, we choose custom script extension cmdlets provided by Azure powershell module rather than creating remote powershell session into the VM and executing the commands.

Remote powershell sessions can be done via http and https protocols. To use https protocol, the VMs should have winrm configure to use https with certificate installed. We do not expect https to be configured in all VMs. The http protocol cannot work when the ip address is used for the machine name and the machine is not domain joined. This restricts us from using the remote powershell sessions to run scripts inside the virtual machines.