Overview

What are virtual machine scale sets?

Get Started

Deploy a scale set in Azure portal
Deploy a scale set using PowerShell or Azure CLI
Deploy an autoscaling app using a template

How To

Plan and design

Design considerations

Create a template

Learn about scale set templates
Use an existing virtual network
Use a custom image
Use guest-based autoscaling with a Linux scale set template

Deploy

Create with Azure portal
Create with Visual Studio
Create with Azure PowerShell or Azure CLI
Scale set with managed disks
Autoscale a scale set
Applications on scale sets
Use data disks with scale sets
Work with large scale sets
Convert a scale set template to use managed disk

Manage

Common management tasks
Vertical scaling in a scale set
Automatic OS upgrades
Upgrades for scale sets
Virtual machine scale sets are an Azure compute resource that you can use to deploy and manage a set of identical VMs. With all VMs configured the same, scale sets are designed to support true autoscale, and no pre-provisioning of VMs is required. So it’s easier to build large-scale services that target big compute, big data, and containerized workloads.

For applications that need to scale compute resources out and in, scale operations are implicitly balanced across fault and update domains. For a further introduction to scale sets, refer to the Azure blog announcement.

For more information about scale sets, watch these videos:

- Mark Russinovich talks Azure scale sets
- Virtual Machine Scale Sets with Guy Bowerman

Creating and managing scale sets

You can create a scale set in the Azure portal by selecting new and typing scale on the search bar. Virtual machine scale set is listed in the results. From there, you can fill in the required fields to customize and deploy your scale set. You also have options to set up basic autoscale rules based on CPU usage in the portal. To manage your scale set, you can use the Azure portal, Azure PowerShell cmdlets, or the Azure CLI 2.0.

Scale sets can be deployed to an availability zone.

**NOTE**
Currently virtual machine scale sets only supports deploying to a single availability zone. Multi-zone deployment will be supported in the future.

You can define and deploy scale sets by using JSON templates and REST APIs, just like individual Azure Resource Manager VMs. Therefore, you can use any standard Azure Resource Manager deployment methods. For more information about templates, see Authoring Azure Resource Manager templates.

You can find a set of example templates for virtual machine scale sets in the Azure Quickstart templates GitHub repository. (Look for templates with vmss in the title.)

For the Quickstart template examples, a "Deploy to Azure" button in the readme for each template links to the portal deployment feature. To deploy the scale set, click the button and then fill in any parameters that are required in the portal.

**Autoscale**

To maintain consistent application performance, you can automatically increase or decrease the number of VM instances in your scale set. This autoscale ability reduces the management overhead to monitor and tune your scale set as customer demand changes over time. You define rules based on performance metrics, application response, or a fixed schedule, and your scale set autoscales as needed.

For basic autoscale rules, you can use host-based performance metrics such as CPU usage or disk I/O. These host-based metrics are available out of the box, with no additional agents or extensions to install and configure. Autoscale rules that use host-based metrics can be created with one of the following tools:
To monitor the application performance itself, you can install and configure a small instrumentation package into your application for App Insights. Detailed performance metrics for the application response time or number of sessions can then be streamed back from your app. You can then create autoscale rules with defined thresholds for the application-level performance itself. For more information about App Insights, see What is Application Insights.

Manually scaling a scale set out and in

You can manually change the capacity of a scale set in the Azure portal by clicking the Scaling section under Settings.

To change scale set capacity on the command line, use the scale command in Azure CLI. For example, use this command to set a scale set to a capacity of 10 VMs:

```
az vmss scale -g resourcegroupname -n scalesetname --new-capacity 10
```

To set the number of VMs in a scale set by using PowerShell, use the Update-AzureRmVmss command:

```
$vms = Get-AzureRmVmss -ResourceGroupName resourcegroupname -VMSetName scalesetname
$vms.Sku.Capacity = 10
Update-AzureRmVmss -ResourceGroupName resourcegroupname -Name scalesetname -VirtualMachineScaleSet $vms
```

To increase or decrease the number of virtual machines in a scale set by using an Azure Resource Manager template, change the capacity property and redeploy the template. This simplicity makes it easy to integrate scale sets with Azure Autoscale, or to write your own custom scaling layer if you need to define custom scale events that Azure Autoscale does not support.

If you are redeploying an Azure Resource Manager template to change the capacity, you can define a much smaller template that includes only the SKU property packet with the updated capacity. Here's an example.

Monitoring your scale set

The Azure portal lists scale sets and shows their properties. The portal also supports management operations. You can perform management operations on both scale sets and individual VMs within a scale set. The portal also provides a customizable resource usage graph.

If you need to see or edit the underlying JSON definition of an Azure resource, you can also use Azure Resource Explorer. Scale sets are a resource under the Microsoft.Compute Azure resource provider. From this site, you can see them by expanding the following links:

```
Subscriptions > your subscription > resourceGroups > providers > Microsoft.Compute > virtualMachineScaleSets > your scale set > etc.
```

Scale set scenarios
This section lists some typical scale set scenarios. Some higher-level Azure services (like Batch, Service Fabric, and Container Service) use these scenarios.

- **Use RDP or SSH to connect to scale set instances**: A scale set is created inside a virtual network, and individual VMs in the scale set are not allocated public IP addresses by default. This policy avoids the expense and management overhead of allocating separate public IP addresses to all the nodes in your compute grid. If you do need direct external connections to scale set VMs, you can configure a scale set to automatically assign public IP addresses to new VMs. Alternatively, you can connect to VMs from other resources in your virtual network that can be allocated public IP addresses, for example, load balancers and standalone virtual machines.

- **Connect to VMs by using NAT rules**: You can create a public IP address, assign it to a load balancer, and define an inbound NAT pool. These actions map ports on the IP address to a port on a VM in the scale set. For example:

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>SOURCE PORT</th>
<th>DESTINATION</th>
<th>DESTINATION PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public IP</td>
<td>Port 50000</td>
<td>vmss_0</td>
<td>Port 22</td>
</tr>
<tr>
<td>Public IP</td>
<td>Port 50001</td>
<td>vmss_1</td>
<td>Port 22</td>
</tr>
<tr>
<td>Public IP</td>
<td>Port 50002</td>
<td>vmss_2</td>
<td>Port 22</td>
</tr>
</tbody>
</table>

In this example, NAT rules are defined to enable an SSH connection to every VM in a scale set, by using a single public IP address.

This example does the same with RDP and Windows.

- **Connect to VMs by using a “jumpbox”**: If you create a scale set and a standalone VM in the same virtual network, the standalone VM and the scale set VM can connect to one another by using their internal IP addresses, as defined by the virtual network or subnet. If you create a public IP address and assign it to the standalone VM, you can use RDP or SSH to connect to the standalone VM. You can then connect from that machine to your scale set instances. You might notice at this point that a simple scale set is inherently more secure than a simple standalone VM with a public IP address in its default configuration.

For example, this template deploys a simple scale set with a standalone VM.

- **Load balancing to scale set instances**: If you want to deliver work to a compute cluster of VMs by using a round-robin approach, you can configure an Azure load balancer with layer-4 load-balancing rules accordingly. You can define probes to verify that your application is running by pinging ports with a specified protocol, interval, and request path. Azure Application Gateway also supports scale sets, along with layer-7 and more sophisticated load-balancing scenarios.

This example creates a scale set that runs Apache web servers, and it uses a load balancer to balance the load that each VM receives. (Look at the Microsoft.Network/loadBalancers resource type and networkProfile and extensionProfile in virtualMachineScaleSet.)

This Linux example and this Windows example use Application Gateway.

- **Deploying a scale set as a compute cluster in a PaaS cluster manager**: Scale sets are sometimes described as a next-generation worker role. Though a valid description, it does run the risk of confusing scale set features with Azure Cloud Services features. In a sense, scale sets provide a true worker role or worker resource. They are a generalized compute resource that is platform/runtime independent, is customizable, and integrates into Azure Resource Manager IaaS.

A Cloud Services worker role is limited in terms of platform/runtime support (Windows platform images only). But it also includes services such as VIP swap, configurable upgrade settings, and runtime/app deployment-specific settings. These services are not yet available in scale sets, or they’re delivered by other
higher-level PaaS services like Azure Service Fabric. You can look at scale sets as an infrastructure that supports PaaS. PaaS solutions like Service Fabric build on this infrastructure.

In this example of this approach, Azure Container Service deploys a cluster based on scale sets with a container orchestrator.

Scale set performance and scale guidance

- A scale set supports up to 1,000 VMs. If you create and upload your own custom VM images, the limit is 300. For considerations in using large scale sets, see Working with large virtual machine scale sets.
- You do not have to pre-create Azure storage accounts to use scale sets. Scale sets support Azure managed disks, which negate performance concerns about the number of disks per storage account. For more information, see Azure virtual machine scale sets and managed disks.
- Consider using Azure Premium Storage instead of Azure Storage for faster, more predictable VM provisioning times and improved I/O performance.
- The vCPU quota in the region in which you are deploying limits the number of VMs you can create. You might need to contact Customer Support to increase your compute quota limit, even if you have a high limit of vCPUs for use with Azure Cloud Services today. To query your quota, run this Azure CLI command: `azure vm list-usage`. Or, run this PowerShell command: `Get-AzureRmVMUsage`.

Frequently asked questions for scale sets

Q. How many VMs can I have in a scale set?

A. A scale set can have 0 to 1,000 VMs based on platform images, or 0 to 300 VMs based on custom images.

Q. Are data disks supported within scale sets?

A. Yes. A scale set can define an attached data disks configuration that applies to all VMs in the set. For more information, see Azure scale sets and attached data disks. Other options for storing data include:

- Azure files (SMB shared drives)
- OS drive
- Temp drive (local, not backed by Azure Storage)
- Azure data service (for example, Azure tables, Azure blobs)
- External data service (for example, remote database)

Q. Which Azure regions support scale sets?

A. All regions support scale sets.

Q. How do I create a scale set by using a custom image?

A. Create a managed disk based on your custom image VHD and reference it in your scale set template. Here’s an example.

Q. If I reduce my scale set capacity from 20 to 15, which VMs are removed?

A. Virtual machines are removed from the scale set evenly across update domains and fault domains to maximize availability. VMs with the highest IDs are removed first.

Q. What if I then increase the capacity from 15 to 18?

A. If you increase capacity to 18, then 3 new VMs are created. Each time, the VM instance ID is incremented from the previous highest value (for example, 20, 21, 22). VMs are balanced across fault domains and update domains.

Q. When I’m using multiple extensions in a scale set, can I enforce an execution sequence?
A. Not directly, but for the customScript extension, your script can wait for another extension to finish. You can get additional guidance on extension sequencing in the blog post Extension Sequencing in Azure VM Scale Sets.

Q. Do scale sets work with Azure availability sets?

A. Yes. A scale set is an implicit availability set with 5 fault domains and 5 update domains. Scale sets of more than 100 VMs span multiple placement groups, which are equivalent to multiple availability sets. For more information about placement groups, see Working with large virtual machine scale sets. An availability set of VMs can exist in the same virtual network as a scale set of VMs. A common configuration is to put control node VMs (which often require unique configuration) in an availability set and put data nodes in the scale set.

You can find more answers to questions about scale sets in the Azure virtual machine scale sets FAQ.
How to create a Virtual Machine Scale Set with the Azure portal

This tutorial shows you how easy it is to create a Virtual Machine Scale Set in just a few minutes, by using the Azure portal. If you don't have an Azure subscription, create a free account before you begin.

Choose the VM image from the marketplace

From the portal, you can easily deploy a scale set with CentOS, CoreOS, Debian, Ubuntu Server, other Linux images, and Windows Server images.

First, navigate to the Azure portal in a web browser. Click New, search for scale set, and then select the Virtual machine scale set entry:

Now you can use the default settings and quickly create the scale set.

- Enter a name for the scale set. This name becomes the base of the FQDN of the load balancer in front of the scale set, so make sure the name is unique across all Azure.

- Select your desired OS type.

- Enter your desired username, and select which authentication type you prefer. If you choose a password, it must be at least 12 characters long and meet three out of the four following complexity requirements: one lower case character, one upper case character, one number, and one special character. See more about username and password requirements. If you choose SSH public key, be sure to only paste in your public key, NOT your private key:

- Select Yes or No to Enable scaling beyond 100 instances. If Yes, the scale set can span across multiple placement groups. For more information, see this documentation.

- Make sure to select an appropriate instance size.
For more information about virtual machine sizes, see either Windows VM sizes or Linux VM sizes.

- Enter your desired resource group name and location.
  If your region and **instance size** supports availability zones, the **Availability zones** field is enabled. For more information on availability zones, see this overview article.

- Enter your desired domain name label (the base of the FQDN for the load balancer in front of the scale set). This label must be unique across all Azure.

- Choose your desired operating system disk image, instance count, and machine size.

- Choose your desired disk type: managed or unmanaged.
  For more information, see this documentation. If you chose to have the scale set span multiple placement groups, this option will not be available because managed disk is required for scale sets to span placement groups.

- Enable or disable autoscale and configure if enabled.

---

**Connect to a VM in the scale set**

If you chose to limit your scale set to a single placement group, then the scale set is deployed with NAT rules.
configured to let you connect to the scale set easily (if not, to connect to the virtual machines in the scale set, you likely need to create a jumpbox in the same virtual network as the scale set). To see them, navigate to the Inbound NAT Rules tab of the load balancer for the scale set:

![Inbound NAT Rules](image)

You can connect to each VM in the scale set using these NAT rules. For instance, for a Windows scale set, if there is a NAT rule on incoming port 50000, you could connect to that machine via RDP on `<load-balancer-ip-address>:50000`. For a Linux scale set, you would connect using the command `ssh -p 50000 <username>@<load-balancer-ip-address>`.

**Next steps**

For documentation on how to deploy scale sets from the CLI, see [this documentation](#).

For documentation on how to deploy scale sets from PowerShell, see [this documentation](#).

For documentation on how to deploy scale sets from Visual Studio, see [this documentation](#).

For general documentation, check out the [documentation overview page for scale sets](#).

For general information, check out the [main landing page for scale sets](#).
Virtual machine scale sets make it easy for you to deploy and manage identical virtual machines as a set. Scale sets provide a highly scalable and customizable compute layer for hyperscale applications, and they support Windows platform images, Linux platform images, custom images, and extensions. For more information about scale sets, see Virtual machine scale sets.

This tutorial shows you how to create a virtual machine scale set without using the Azure portal. For information about how to use the Azure portal, see How to create a virtual machine scale set with the Azure portal.

NOTE
For more information about Azure Resource Manager resources, see Azure Resource Manager vs. classic deployment.

Sign in to Azure

If you’re using Azure CLI 2.0 or Azure PowerShell to create a scale set, you first need to sign in to your subscription.

For more information about how to install, set up, and sign in to Azure with Azure CLI or PowerShell, see Getting Started with Azure CLI 2.0 or Get started with Azure PowerShell cmdlets.

```
az login
```

```
Login-AzureRmAccount
```

Create a resource group

You first need to create a resource group that the virtual machine scale set is associated with.

```
az group create --location westus2 --name MyResourceGroup1
```

```
New-AzureRmResourceGroup -Location westus2 -Name MyResourceGroup1
```

Create from Azure CLI

With Azure CLI, you can create a virtual machine scale set with minimal effort. If you omit default values, they are provided for you. For example, if you don’t specify any virtual network information, a virtual network is created for you. If you omit the following parts, they are created for you:

- A load balancer
- A virtual network
- A public IP address

When choosing the virtual machine image that you want to use on the virtual machine scale set, you have a few choices:
• **URN**
  The identifier of a resource:
  
  Win2012R2Datacenter

• **URN alias**
  The friendly name of a URN:
  

• **Custom resource id**
  The path to an Azure resource:
  
  /subscriptions/subscription-guid/resourceGroups/MyResourceGroup/providers/Microsoft.Compute/images/MyImage

• **Web resource**
  The path to an HTTP URI:
  
  http://contoso.blob.core.windows.net/vhds/osdiskimage.vhd

(TIP)
You can get a list of available images with **az vm image list**.

To create a virtual machine scale set, you must specify the following:

• Resource group
• Name
• Operating system image
• Authentication information

The following example creates a basic virtual machine scale set (this step might take a few minutes).

```
az vmss create --resource-group MyResourceGroup1 --name MyScaleSet --image UbuntuLTS --authentication-type password --admin-username azureuser --admin-password P@ssw0rd!
```

Once the command finishes you will now have your virtual machine scale set created. You may need to get the IP address of the virtual machine so that you can connect to it. You can get a lot of different information about the virtual machine (including the IP address) with the following command.

```
az vmss list-instance-connection-info --resource-group MyResourceGroup1 --name MyScaleSet
```

**Create from PowerShell**

PowerShell is more complicated to use than Azure CLI. While Azure CLI provides defaults for networking-related resources (such as load balancers, IP addresses, and virtual networks), PowerShell does not. Referencing an image with PowerShell is a slightly more complicated too. You can get images with the following cmdlets:

1. Get-AzureRMVMImagePublisher
2. Get-AzureRMVMImageOffer
3. Get-AzureRmVMImageSku

The cmdlets work can be piped in sequence. Here is an example of how to get all images for the **West US 2** region with a publisher that has the name **microsoft** in it.
The workflow for creating a virtual machine scale set is as follows:

1. Create a config object that holds information about the scale set.
2. Reference the base OS image.
3. Configure the operating system settings: authentication, VM name prefix, and user/pass.
4. Configure networking.
5. Create the scale set.

This example creates a basic two-instance scale set for a computer that has Windows Server 2016 installed.
# Resource group name from above
$rg = "MyResourceGroup1"
$location = "WestUS2"

# Create a config object
$vmssConfig = New-AzureRmVmssConfig -Location $location -SkuCapacity 2 -SkuName Standard_A0 -UpgradePolicyMode Automatic

# Reference a virtual machine image from the gallery

# Set up information for authenticating with the virtual machine
Set-AzureRmVmssOsProfile $vmssConfig -AdminUsername azureuser -AdminPassword P@ssw0rd! -ComputerNamePrefix myvmssvm

# Create the virtual network resources

## Basics
$subnet = New-AzureRmVirtualNetworkSubnetConfig -Name "my-subnet" -AddressPrefix 10.0.0.0/24
$vnet = New-AzureRmVirtualNetwork -Name "my-network" -ResourceGroupName $rg -Location $location -AddressPrefix 10.0.0.0/16 -Subnet $subnet

## Load balancer
$publicIP = New-AzureRmPublicIpAddress -Name "PublicIP" -ResourceGroupName $rg -Location $location -AllocationMethod Static -DomainNameLabel "myuniquedomain"
$frontendIP = New-AzureRmLoadBalancerFrontendIpConfig -Name "LB-Frontend" -PublicIpAddress $publicIP
$backendPool = New-AzureRmLoadBalancerBackendAddressPoolConfig -Name "LB-backend"
$probe = New-AzureRmLoadBalancerProbeConfig -Name "HealthProbe" -Protocol Tcp -Port 80 -IntervalInSeconds 15 -ProbeCount 2
$inboundNATRule1 = New-AzureRmLoadBalancerRuleConfig -Name "webserver" -FrontendIpConfiguration $frontendIP -Protocol Tcp -FrontendPort 80 -BackendPort 80 -IdleTimeoutInMinutes 15 -Probe $probe -BackendAddressPool $backendPool
$inboundNATPool1 = New-AzureRmLoadBalancerInboundNatPoolConfig -Name "RDP" -FrontendIpConfigurationId $frontendIP.Id -Protocol TCP -FrontendPortRangeStart 53380 -FrontendPortRangeEnd 53390 -BackendPort 3389
New-AzureRmLoadBalancer -ResourceGroupName $rg -Name "LB1" -Location $location -FrontendIpConfiguration $frontendIP -LoadBalancingRule $inboundNATRule1 -InboundNatPool $inboundNATPool1 -BackendAddressPool $backendPool -Probe $probe

## IP address config
$ipConfig = New-AzureRmVmssIpConfig -Name "my-ipaddress" -LoadBalancerBackendAddressPoolsId $backendPool.Id -SubnetId $vnet.Subnets[0].Id -LoadBalancerInboundNatPoolsId $inboundNATPool1.Id

# Attach the virtual network to the IP object
Add-AzureRmVmssNetworkInterfaceConfiguration -VirtualMachineScaleSet $vmssConfig -Name "network-config" -Primary $true -IPConfiguration $ipConfig

# Create the scale set with the config object (this step might take a few minutes)
New-AzureRmVmss -ResourceGroupName $rg -Name "MyScaleSet1" -VirtualMachineScaleSet $vmssConfig

Using a custom virtual machine image

If you are creating a scale set from your own custom image, instead of referencing a virtual machine image from the gallery, the `Set-AzureRmVmssStorageProfile` command would look like this:

```
Set-AzureRmVmssStorageProfile -OsDiskCreateOption FromImage -ManagedDisk PremiumLRS -OsDiskCaching "None" -OsDiskOsType Linux -ImageReferenceId (Get-AzureRmImage -ImageName $VMImage -ResourceGroupName $rg).id
```

Create from a template

You can deploy a virtual machine scale set by using an Azure Resource Manager template. You can create your own template or use one from the template repository. These templates can be deployed directly to your Azure
subscription.

**NOTE**
To create your own template, you create a JSON text file. For general information about how to create and customize a template, see Azure Resource Manager templates.

A sample template is available on GitHub. For more information about how to create and use that sample, see Minimum viable scale set.

**Create from Visual Studio**

With Visual Studio, you can create an Azure resource group project and add a virtual machine scale set template to it. You can choose whether you want to import it from GitHub or the Azure Web Application Gallery. A deployment PowerShell script is also generated for you. For more information, see How to create a virtual machine scale set with Visual Studio.

**Create from the Azure portal**

The Azure portal provides a convenient way to quickly create a scale set. For more information, see How to create a virtual machine scale set with the Azure portal.

**Next steps**

Learn more about data disks.

Learn how to manage your apps.
Azure Resource Manager templates are a great way to deploy groups of related resources. This tutorial builds on Deploy a simple scale set and describes how to deploy a simple autoscaling application on a scale set using an Azure Resource Manager template. You can also set up autoscaling using PowerShell, CLI, or the portal. For more information, see Autoscale overview.

Two quickstart templates

When you deploy a scale set you can install new software on a platform image using a VM Extension. A VM extension is a small application that provides post-deployment configuration and automation tasks on Azure virtual machines, such as deploying an app. Two different sample templates are provided in Azure/azure-quickstart-templates which show how to deploy an autoscaling application onto a scale set using VM extensions.

Python HTTP server on Linux

The Python HTTP server on Linux sample template deploys a simple autoscaling application running on a Linux scale set. Bottle, a Python web framework, and a simple HTTP server are deployed on each VM in the scale set using a custom script VM extension. The scale set scales up when average CPU utilization across all VMs is greater than 60% and scales down when the average CPU utilization is less than 30%.

In addition to the scale set resource, the azuredeploy.json sample template also declares virtual network, public IP address, load balancer, and autoscale settings resources. For more information on creating these resources in a template, see Linux scale set with autoscale.

In the azuredeploy.json template, the extensionProfile property of the Microsoft.Compute/virtualMachineScaleSets resource specifies a custom script extension. fileUris specifies the script(s) location. In this case, two files: workserver.py, which defines a simple HTTP server, and installserver.sh, which installs Bottle and starts the HTTP server. commandToExecute specifies the command to run after the scale set has been deployed.

```json
"extensionProfile": {
  "extensions": [
    {
      "name": "lapextension",
      "properties": {
        "publisher": "Microsoft.Azure.Extensions",
        "type": "CustomScript",
        "typeHandlerVersion": "2.0",
        "autoUpgradeMinorVersion": true,
        "settings": {
          "fileUris": [
            "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/201-vmss-bottle-autoscale/workserver.py"
          ],
          "commandToExecute": "bash installserver.sh"
        }
      }
    }
  ]
}
```

ASP.NET MVC application on Windows
The **ASP.NET MVC application on Windows** sample template deploys a simple ASP.NET MVC app running in IIS on Windows scale set. IIS and the MVC app are deployed using the **PowerShell desired state configuration (DSC)** VM extension. The scale set scales up (on VM instance at a time) when CPU utilization is greater than 50% for 5 minutes.

In addition to the scale set resource, the **azuredeploy.json** sample template also declares virtual network, public IP address, load balancer, and autoscale settings resources. This template also demonstrates application upgrade. For more information on creating these resources in a template, see **Windows scale set with autoscale**.

In the **azuredeploy.json** template, the `extensionProfile` property of the `Microsoft.Compute/virtualMachineScaleSets` resource specifies a desired state configuration (DSC) extension which installs IIS and a default web app from a WebDeploy package. The `IISInstall.ps1` script installs IIS on the virtual machine and is found in the DSC folder. The MVC web app is found in the `WebDeploy` folder. The paths to the install script and the web app are defined in the `powershelldscZip` and `webDeployPackage` parameters in the `azuredeploy.parameters.json` file.

```
"extensionProfile": {
  "extensions": [
    {
      "name": "Microsoft.Powershell.DSC",
      "properties": {
        "publisher": "Microsoft.Powershell",
        "type": "DSC",
        "typeHandlerVersion": "2.9",
        "autoUpgradeMinorVersion": true,
        "forceUpdateTag": "[parameters('powershelldscUpdateTagVersion')]",
        "settings": {
          "configuration": {
            "url": "[variables('powershelldscZipFullPath')]",
            "script": "IISInstall.ps1",
            "function": "InstallIIS"
          },
          "configurationArguments": {
            "nodeName": "localhost",
            "WebDeployPackagePath": "[variables('webDeployPackageFullPath')]"
          }
        }
      }
    }
  ]
}
```

**Deploy the template**

The simplest way to deploy the **Python HTTP server on Linux** or **ASP.NET MVC application on Windows** template is to use the **Deploy to Azure** button found in the in the readme files in GitHub. You can also use PowerShell or Azure CLI to deploy the sample templates.

**PowerShell**

Copy the **Python HTTP server on Linux** or **ASP.NET MVC application on Windows** files from the GitHub repo to a folder on your local computer. Open the `azuredeploy.parameters.json` file and update the default values of the `vmssName`, `adminUsername`, and `adminPassword` parameters. Save the following PowerShell script to `deploy.ps1` in the same folder as the `azuredeploy.json` template. To deploy the sample template run the `deploy.ps1` script from a PowerShell command window.

```
param(
  [Parameter(Mandatory=$True)]
  [string]
  $subscriptionId,
  ...
)
Function RegisterRP {
    Param(
        [string]$ResourceProviderNamespace
    )
    Write-Host "Registering resource provider '$ResourceProviderNamespace';";
    Register-AzureRmResourceProvider -ProviderNamespace $ResourceProviderNamespace;
}

#******************************************************************************
# Script body
# Execution begins here
#******************************************************************************
$ErrorActionPreference = "Stop"
# sign in
Write-Host "Logging in...";
Login-AzureRmAccount;
# select subscription
Write-Host "Selecting subscription '$subscriptionId';";
Select-AzureRmSubscription -SubscriptionID $subscriptionId;
# Register RPs
$resourceProviders = @("microsoft.compute", "microsoft.insights", "microsoft.network");
if($resourceProviders.length) {
    Write-Host "Registering resource providers";
    foreach($resourceProvider in $resourceProviders) {
        RegisterRP($resourceProvider);
    }
}
#Create or check for existing resource group
$resourceGroup = Get-AzureRmResourceGroup -Name $resourceGroupName -ErrorAction SilentlyContinue
if(!$resourceGroup) {
    Write-Host "Resource group '$resourceGroupName' does not exist. To create a new resource group, please enter a location.";
    if(!$resourceGroupLocation) {
        $resourceGroupLocation = Read-Host "resourceGroupLocation";
    }
    Write-Host "Creating resource group '$resourceGroupName' in location '$resourceGroupLocation';";
    New-AzureRmResourceGroup -Name $resourceGroupName -Location $resourceGroupLocation
} else {
    Write-Host "Using existing resource group '$resourceGroupName';";
}
# Start the deployment
Write-Host "Starting deployment..."
if(Test-Path $parametersFilePath) {
    New-AzureRmResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateFile $templateFilePath -
    TemplateParameterFile $parametersFilePath;
} else {
    New-AzureRmResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateFile $templateFilePath;
}

---

# Azure CLI

#!/bin/bash
set -euo pipefail
IFS=\n\t

# -e: immediately exit if any command has a non-zero exit status
# -o: prevents errors in a pipeline from being masked
# IFS new value is less likely to cause confusing bugs when looping arrays or arguments (e.g. $0)

usage() { echo "Usage: $0 -i <subscriptionId> -g <resourceGroupName> -n <deploymentName> -l
    <resourceGroupLocation>" 1>&2; exit 1; }

declare subscriptionId=""
declare resourceGroupName=""
declare deploymentName=""
declare resourceGroupLocation=""

# Initialize parameters specified from command line
while getopts ":i:g:n:l:" arg; do
case "$(arg)" in
    i)
        subscriptionId=${OPTARG}
        ;;
    g)
        resourceGroupName=${OPTARG}
        ;;
    n)
        deploymentName=${OPTARG}
        ;;
    l)
        resourceGroupLocation=${OPTARG}
        ;;
    esac
done
shift $((OPTIND-1))

#Prompt for parameters is some required parameters are missing
if [[ -z "$subscriptionId" ]]; then
    echo "Subscription Id:"
    read subscriptionId
    [[ "$subscriptionId" ]]
fi

if [[ -z "$resourceGroupName" ]]; then
    echo "ResourceGroupName:"
    read resourceGroupName
    [[ "$resourceGroupName" ]]
fi

if [[ -z "$deploymentName" ]]; then
    echo "DeploymentName:"
    read deploymentName
    fi

if [[ -z "$resourceGroupLocation" ]]; then
    echo "Enter a location below to create a new resource group else skip this"
    read resourceGroupLocation
    fi
echo "ResourceGroupLocation:"
read resourceGroupLocation

#templateFile Path - template file to be used
templateFilePath="template.json"

if [ ! -f "$templateFilePath" ]; then
    echo "$templateFilePath not found"
    exit 1
fi

#parameter file path
parametersFilePath="parameters.json"

if [ ! -f "$parametersFilePath" ]; then
    echo "$parametersFilePath not found"
    exit 1
fi

if [ -z "$subscriptionId" ] || [ -z "$resourceGroupName" ] || [ -z "$deploymentName" ]; then
    echo "Either one of subscriptionId, resourceGroupName, deploymentName is empty"
    usage
fi

#login to azure using your credentials
az account show 1> /dev/null

if [ $? != 0 ]; then
    az login
fi

#set the default subscription id
az account set --name $subscriptionId

set +e

#Check for existing RG
az group show $resourceGroupName 1> /dev/null

if [ $? != 0 ]; then
    echo "Resource group with name" $resourceGroupName "could not be found. Creating new resource group.."
    set -e
    (set -x
        az group create --name $resourceGroupName --location $resourceGroupLocation 1> /dev/null
    )
else
    echo "Using existing resource group..."
fi

#Start deployment
echo "Starting deployment..."
(
    set -x
    az group deployment create --name $deploymentName --resource-group $resourceGroupName --template-file $templateFilePath --parameters $parametersFilePath
)

if [ $? == 0 ]; then
    echo "Template has been successfully deployed"
fi

Next steps
You can deploy the preceding template by following the Azure Resource Manager documentation.

You can start this tutorial series from the minimum viable scale set template article.

You can see how to modify the minimum viable scale set template to deploy the scale set into an existing virtual network.

You can see how to modify the minimum viable scale set template to deploy the scale set with a custom image.

You can see how to modify the minimum viable scale set template to deploy a Linux scale set with guest-based autoscale.

For more general information about scale sets, refer to the scale set overview page.
This topic discusses design considerations for Virtual Machine Scale Sets. For information about what Virtual Machine Scale Sets are, refer to Virtual Machine Scale Sets Overview.

When to use scale sets instead of virtual machines?

Generally, scale sets are useful for deploying highly available infrastructure where a set of machines have similar configuration. However, some features are only available in scale sets while other features are only available in VMs. In order to make an informed decision about when to use each technology, we should first take a look at some of the commonly used features that are available in scale sets but not VMs:

**Scale set-specific features**

- Once you specify the scale set configuration, you can simply update the "capacity" property to deploy more VMs in parallel. This is much simpler than writing a script to orchestrate deploying many individual VMs in parallel.
- You can use Azure Autoscale to automatically scale a scale set but not individual VMs.
- You can reimage scale set VMs but not individual VMs.
- You can overprovision scale set VMs for increased reliability and quicker deployment times. You cannot do this with individual VMs unless you write custom code to do this.
- You can specify an upgrade policy to make it easy to roll out upgrades across VMs in your scale set. With individual VMs, you must orchestrate updates yourself.

**VM-specific features**

On the other hand, some features are only available in VMs (at least for the time being):

- You can attach data disks to specific individual VMs, but attached data disks are configured for all VMs in a scale set.
- You can attach non-empty data disks to individual VMs but not VMs in a scale set.
- You can snapshot an individual VM but not a VM in a scale set.
- You can capture an image from an individual VM but not from a VM in a scale set.
- You can migrate an individual VM from native disks to managed disks, but you cannot do this for VMs in a scale set.
- You can assign IPv6 public IP addresses to individual VM nics but cannot do so for VMs in a scale set. Note that you can assign IPv6 public IP addresses to load balancers in front of either individual VMs or scale set VMs.

Storage

**Scale sets with Azure Managed Disks**

Scale sets can be created with Azure Managed Disks instead of traditional Azure storage accounts. Managed Disks provide the following benefits:

- You do not have to pre-create a set of Azure storage accounts for the scale set VMs.
- You can define attached data disks for the VMs in your scale set.
- Scale sets can be configured to support up to 1,000 VMs in a set.

If you have an existing template, you can also update the template to use Managed Disks.

**User-managed Storage**
A scale set that is not defined with Azure Managed Disks relies on user-created storage accounts to store the OS disks of the VMs in the set. A ratio of 20 VMs per storage account or less is recommended to achieve maximum IO and also take advantage of overprovisioning (see below). It is also recommended that you spread the beginning characters of the storage account names across the alphabet. Doing so helps spread load across different internal systems.

**Overprovisioning**

Scale sets currently default to "overprovisioning" VMs. With overprovisioning turned on, the scale set actually spins up more VMs than you asked for, then deletes the extra VMs once the requested number of VMs are successfully provisioned. Overprovisioning improves provisioning success rates and reduces deployment time. You are not billed for the extra VMs, and they do not count toward your quota limits.

While overprovisioning does improve provisioning success rates, it can cause confusing behavior for an application that is not designed to handle extra VMs appearing and then disappearing. To turn overprovisioning off, ensure you have the following string in your template: "overprovision": "false". More details can be found in the Scale Set REST API documentation.

If your scale set uses user-managed storage, and you turn off overprovisioning, you can have more than 20 VMs per storage account, but it is not recommended to go above 40 for IO performance reasons.

**Limits**

A scale set built on a Marketplace image (also known as a platform image) and configured to use Azure Managed Disks supports a capacity of up to 1,000 VMs. If you configure your scale set to support more than 100 VMs, not all scenarios work the same (for example load balancing). For more information, see Working with large virtual machine scale sets.

A scale set configured with user-managed storage accounts is currently limited to 100 VMs (and 5 storage accounts are recommended for this scale).

A scale set built on a custom image (one built by you) can have a capacity of up to 300 VMs when configured with Azure Managed disks. If the scale set is configured with user-managed storage accounts, it must create all OS disk VHDS within one storage account. As a result, the maximum recommended number of VMs in a scale set built on a custom image and user-managed storage is 20. If you turn off overprovisioning, you can go up to 40.

For more VMs than these limits allow, you need to deploy multiple scale sets as shown in this template.
Azure Resource Manager templates are a great way to deploy groups of related resources. This tutorial series shows how to create a minimum viable scale set template and how to modify this template to suit various scenarios. All examples come from this GitHub repository.

This template is intended to be simple. For more complete examples of scale set templates, see the Azure Quickstart Templates GitHub repository and search for folders that contain the string `vmss`.

If you are already familiar with creating templates, you can skip to the "Next steps" section to see how to modify this template.

Review the template

Use GitHub to review our minimum viable scale set template, `azuredeploy.json`.

In this tutorial, we examine the diff (`git diff master minimum-viable-scale-set`) to create the minimum viable scale set template piece by piece.

Define $schema and contentVersion

First, we define `$schema` and `contentVersion` in the template. The `$schema` element defines the version of the template language and is used for Visual Studio syntax highlighting and similar validation features. The `contentVersion` element is not used by Azure. Instead, it helps you keep track of the template version.

```json
{
  "contentVersion": "1.0.0.0",
}
```

Define parameters

Next, we define two parameters, `adminUsername` and `adminPassword`. Parameters are values you specify at the time of deployment. The `adminUsername` parameter is simply a `string` type, but because `adminPassword` is a secret, we give it type `securestring`. Later, these parameters are passed into the scale set configuration.

```json
"parameters": {
  "adminUsername": {
    "type": "string"
  },
  "adminPassword": {
    "type": "securestring"
  }
},
```

Define variables

Resource Manager templates also let you define variables to be used later in the template. Our example doesn't use any variables, so we've left the JSON object empty.

```json
"variables": {},
```
Define resources

Next is the resources section of the template. Here, you define what you actually want to deploy. Unlike parameters and variables (which are JSON objects), resources is a JSON list of JSON objects.

```
"resources": [
    {
        "type": "Microsoft.Network/virtualNetworks",
        "name": "myVnet",
        "apiVersion": "2016-12-01",
    }
]
```

All resources require type, name, apiVersion, and location properties. This example’s first resource has type Microsoft.Network/virtualNetwork, name myVnet, and apiVersion 2016-03-30. (To find the latest API version for a resource type, see the Azure REST API documentation.)

Specify location

To specify the location for the virtual network, we use a Resource Manager template function. This function must be enclosed in quotes and square brackets like this: `[<template-function>]`. In this case, we use the `resourceGroup()` function. It takes no arguments and returns a JSON object with metadata about the resource group this deployment is being deployed to. The resource group is set by the user at the time of deployment. We then index into this JSON object with `.location` to get the location from the JSON object.

```
"location": [resourceGroup().location],
```

Specify virtual network properties

Each Resource Manager resource has its own properties section for configurations specific to the resource. In this case, we specify that the virtual network should have one subnet using the private IP address range 10.0.0.0/16. A scale set is always contained within one subnet. It cannot span subnets.

```
"properties": {
    "addressSpace": {
        "addressPrefixes": [
            "10.0.0.0/16"
        ],
    },
    "subnets": [
        {
            "name": "mySubnet",
            "properties": {
                "addressPrefix": "10.0.0.0/16"
            }
        }
    ]
},
```

Add dependsOn list

In addition to the required type, name, apiVersion, and location properties, each resource can have an optional dependsOn list of strings. This list specifies which other resources from this deployment must finish before deploying this resource.

In this case, there is only one element in the list, the virtual network from the previous example. We specify this dependency because the scale set needs the network to exist before creating any VMs. This way, the scale set can give
these VMs private IP addresses from the IP address range previously specified in the network properties. The format of each string in the dependsOn list is `<type>/<name>`. Use the same `type` and `name` used previously in the virtual network resource definition.

```json
{
  "type": "Microsoft.Compute/virtualMachineScaleSets",
  "name": "myScaleSet",
  "apiVersion": "2016-04-30-preview",
  "location": "[resourceGroup().location]",
  "dependsOn": [
    "Microsoft.Network/virtualNetworks/myVnet"
  ],
}
```

**Specify scale set properties**

Scale sets have many properties for customizing the VMs in the scale set. For a full list of these properties, see the scale set REST API documentation. For this tutorial, we will set only a few commonly used properties.

**Supply VM size and capacity**

The scale set needs to know what size of VM to create ("sku name") and how many such VMs to create ("sku capacity"). To see which VM sizes are available, see the VM Sizes documentation.

```json
"sku": {
  "name": "Standard_A1",
  "capacity": 2
},
```

**Choose type of updates**

The scale set also needs to know how to handle updates on the scale set. Currently, there are two options, **Manual** and **Automatic**. For more information on the differences between the two, see the documentation on how to upgrade a scale set.

```json
"properties": {
  "upgradePolicy": {
    "mode": "Manual"
  }
},
```

**Choose VM operating system**

The scale set needs to know what operating system to put on the VMs. Here, we create the VMs with a fully patched Ubuntu 16.04-LTS image.

```json
"virtualMachineProfile": {
  "storageProfile": {
    "imageReference": {
      "publisher": "Canonical",
      "offer": "UbuntuServer",
      "sku": "16.04-LTS",
      "version": "latest"
    }
  }
},
```

**Specify computerNamePrefix**

The scale set deploys multiple VMs. Instead of specifying each VM name, we specify `computerNamePrefix`. The scale set appends an index to the prefix for each VM, so VM names have the form `<computerNamePrefix>_auto-generated-index`.

In the following snippet, we use the parameters from before to set the administrator username and password for all VMs in the scale set. We do this with the `parameters` template function. This function takes in a string that specifies
which parameter to refer to and outputs the value for that parameter.

```json
"osProfile": {
    "computerNamePrefix": "vm",
    "adminUsername": "[parameters('adminUsername')]",
    "adminPassword": "[parameters('adminPassword')]"
},

"networkProfile": {
    "networkInterfaceConfigurations": [
        {
            "name": "myNic",
            "properties": {
                "primary": "true",
                "ipConfigurations": [
                    {
                        "name": "myIpConfig",
                        "properties": {
                            "subnet": {
                                "id": "[concat(resourceId('Microsoft.Network/virtualNetworks', 'myVnet'), '/subnets/mySubnet')]"
                            }
                        }
                    }
                ]
            }
        }
    ]
}
```

Specify VM network configuration

Finally, we need to specify the network configuration for the VMs in the scale set. In this case, we only need to specify the ID of the subnet created earlier. This tells the scale set to put the network interfaces in this subnet.

You can get the ID of the virtual network containing the subnet by using the `resourceId` template function. This function takes in the type and name of a resource and returns the fully qualified identifier of that resource. This ID has the form:

```
/subscriptions/<subscriptionId>/resourceGroups/<resourceGroupName>/<resourceProviderNamespace>/<resourceType>/<resourceName>
```

However, the identifier of the virtual network is not enough. You must specify the specific subnet that the scale set VMs should be in. To do this, concatenate `/subnets/mySubnet` to the ID of the virtual network. The result is the fully qualified ID of the subnet. Do this concatenation with the `concat` function, which takes in a series of strings and returns their concatenation.

```
"networkProfile": {
    "networkInterfaceConfigurations": [
        {
            "name": "myNic",
            "properties": {
                "primary": "true",
                "ipConfigurations": [
                    {
                        "name": "myIpConfig",
                        "properties": {
                            "subnet": {
                                "id": "[concat(resourceId('Microsoft.Network/virtualNetworks', 'myVnet'), '/subnets/mySubnet')]"
                            }
                        }
                    }
                ]
            }
        }
    ]
}
```

Next steps

You can deploy the preceding template by following the Azure Resource Manager documentation.

You can start this tutorial series from the minimum viable scale set template article.

You can see how to modify the minimum viable scale set template to deploy the scale set into an existing virtual network.

You can see how to modify the minimum viable scale set template to deploy the scale set with a custom image.

You can see how to modify the minimum viable scale set template to deploy a Linux scale set with guest-based autoscale.
For more general information about scale sets, refer to the scale set overview page.
This article shows how to modify the minimum viable scale set template to deploy into an existing virtual network instead of creating a new one.

Change the template definition

Our minimum viable scale set template can be seen here, and our template for deploying the scale set into an existing virtual network can be seen here. Let's examine the diff used to create this template (git diff minimum-viable-scale-set existing-vnet) piece by piece:

First, we add a subnetId parameter. This string will be passed into the scale set configuration, allowing the scale set to identify the pre-created subnet to deploy virtual machines into. This string must be of the form:

```
/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Network/virtualNetworks/<virtual-network-name>/subnets/<subnet-name>
```

For instance, to deploy the scale set into an existing virtual network with name myvnet, subnet mysubnet, resource group myrg, and subscription 00000000-0000-0000-0000-000000000000, the subnetId would be:

```
/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/myrg/providers/Microsoft.Network/virtualNetworks/myvnet/subnets/mysubnet
```

```
+    "subnetId": {
+      "type": "string"
+    }
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
+
```

Next, we can delete the virtual network resource from the resources array, since we are using an existing virtual network and don’t need to deploy a new one.
The virtual network already exists before the template is deployed, so there is no need to specify a dependsOn clause from the scale set to the virtual network. Thus, we delete these lines:

```
    
    
```

Finally, we pass in the \[subnetId\] parameter set by the user (instead of using \[resourceId\] to get the id of a vnet in the same deployment, which is what the minimum viable scale set template does).

```
    
    
```

Next steps

You can deploy the preceding template by following the Azure Resource Manager documentation.

You can start this tutorial series from the minimum viable scale set template article.

You can see how to modify the minimum viable scale set template to deploy the scale set into an existing virtual network.
You can see how to modify the minimum viable scale set template to deploy the scale set with a custom image.

You can see how to modify the minimum viable scale set template to deploy a Linux scale set with guest-based autoscale.

For more general information about scale sets, refer to the scale set overview page.
This article shows how to modify the minimum viable scale set template to deploy from custom image.

### Change the template definition

Our minimum viable scale set template can be seen [here](#), and our template for deploying the scale set from a custom image can be seen [here](#). Let's examine the diff used to create this template (`git diff minimum-viable-scale-set custom-image`) piece by piece:

#### Creating a managed disk image

If you already have a custom managed disk image (a resource of type `Microsoft.Compute/images`), then you can skip this section.

First, we add a `sourceImageVhdUri` parameter, which is the URI to the generalized blob in Azure Storage that contains the custom image to deploy from.

```json
},
  "adminPassword": {
    "type": "securestring"
  },
  + "sourceImageVhdUri": {
    + "type": "string",
    + "metadata": {
      + "description": "The source of the generalized blob containing the custom image"
    }
  }
}
```

Next, we add a resource of type `Microsoft.Compute/images`, which is the managed disk image based on the generalized blob located at URI `sourceImageVhdUri`. This image must be in the same region as the scale set that uses it. In the properties of the image, we specify the OS type, the location of the blob (from the `sourceImageVhdUri` parameter), and the storage account type:
"resources": [
  {
    "type": "Microsoft.Compute/images",
    "apiVersion": "2016-04-30-preview",
    "name": "myCustomImage",
    "location": "[resourceGroup().location]",
    "properties": {
      "storageProfile": {
        "osDisk": {
          "osType": "Linux",
          "osState": "Generalized",
          "blobUri": "[parameters('sourceImageVhdUri')]",
          "storageAccountType": "Standard_LRS"
        }
      }
    }
  },
  {
    "type": "Microsoft.Network/virtualNetworks",
    "name": "myVnet",
    "location": "[resourceGroup().location]",
    "apiVersion": "2016-04-30-preview",
    "dependsOn": [
      "Microsoft.Network/virtualNetworks/myVnet",
      "Microsoft.Compute/images/myCustomImage"
    ],
    "sku": {
      "name": "Standard_A1",
      "location": "[resourceGroup().location]"
    }
  },
  {
    "type": "Microsoft.Network/virtualNetworks",
    "name": "myVnet",
    "location": "[resourceGroup().location]",
    "apiVersion": "2016-04-30-preview",
    "dependsOn": [
      "Microsoft.Network/virtualNetworks/myVnet"
    ],
    "sku": {
      "name": "Standard_A1"
    }
  }]

In the scale set resource, we add a `dependsOn` clause referring to the custom image to make sure the image gets created before the scale set tries to deploy from that image:

```
"location": "[resourceGroup().location]",
"apiVersion": "2016-04-30-preview",
"dependsOn": [
  "Microsoft.Network/virtualNetworks/myVnet",
  "Microsoft.Compute/images/myCustomImage"
],
"sku": {
  "name": "Standard_A1",
  "location": "[resourceGroup().location]"
}
```

### Changing scale set properties to use the managed disk image

In the `imageReference` of the scale set `storageProfile`, instead of specifying the publisher, offer, sku, and version of a platform image, we specify the `id` of the `Microsoft.Compute/images` resource:

```
"virtualMachineProfile": {
  "storageProfile": {
    "imageReference": {
      "publisher": "Canonical",
      "offer": "UbuntuServer",
      "sku": "16.04-LTS",
      "version": "latest"
    },
    "id": "[resourceId('Microsoft.Compute/images', 'myCustomImage')]"
  },
  "osProfile": {
    "imageReference": {
      "publisher": "Canonical",
      "offer": "UbuntuServer",
      "sku": "16.04-LTS",
      "version": "latest"
    },
    "id": "[resourceId('Microsoft.Compute/images', 'myCustomImage')]"
  }
},
```

In this example, we use the `resourceId` function to get the resource ID of the image created in the same template. If you have created the managed disk image beforehand, you should provide the id of that image instead. This id must be of the form:

```
/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Compute/images/<image-name>
```

### Next Steps
You can deploy the preceding template by following the Azure Resource Manager documentation.

You can start this tutorial series from the minimum viable scale set template article.

You can see how to modify the minimum viable scale set template to deploy the scale set into an existing virtual network.

You can see how to modify the minimum viable scale set template to deploy the scale set with a custom image.

You can see how to modify the minimum viable scale set template to deploy a Linux scale set with guest-based autoscale.

For more general information about scale sets, refer to the scale set overview page.
There are two types of metrics in Azure that are gathered from VMs and scale sets: some come from the host VM and others come from the guest VM. At a high level, if you are using standard CPU, disk, and network metrics, then host metrics are probably a good fit. If, however, you need a larger selection of metrics, then guest metrics are probably a better fit. Let’s take a look at the differences between the two:

Host metrics are simpler and more reliable. They do not require additional setup because they are collected by the host VM, whereas guest metrics require us to install the Windows Azure Diagnostics extension or the Linux Azure Diagnostics extension in the guest VM. One common reason to use guest metrics instead of host metrics is that guest metrics provide a larger selection of metrics than host metrics. One such example is memory-consumption metrics, which are only available via guest metrics. The supported host metrics are listed here, and commonly used guest metrics are listed here. This article shows how to modify the minimum viable scale set template to use autoscale rules based on guest metrics for Linux scale sets.

Change the template definition

Our minimum viable scale set template can be seen here, and our template for deploying the Linux scale set with guest-based autoscale can be seen here. Let’s examine the diff used to create this template (git diff minimum-viable-scale-set existing-vnet) piece by piece:

First, we add parameters for storageAccountName and storageAccountSasToken. The diagnostics agent will store metric data in a table in this storage account. As of the Linux Diagnostics Agent version 3.0, using a storage access key is no longer supported. We must use a SAS Token.

Next, we modify the scale set extensionProfile to include the diagnostics extension. In this configuration, we specify the resource ID of the scale set to collect metrics from, as well as the storage account and SAS token to use to store the metrics. We also specify how frequently the metrics are aggregated (in this case every minute) and which metrics to track (in this case percent used memory). For more detailed information on this configuration and metrics other than percent used memory, see this documentation.
Finally, we add an `autoscaleSettings` resource to configure autoscale based on these metrics. This resource has a `dependsOn` clause that references the scale set to ensure that the scale set exists before attempting to autoscale it. If we choose a different metric to autoscale on, we would use the `counterSpecifier` from the diagnostics extension configuration as the `metricName` in the autoscale configuration. For more information on autoscale configuration, see the autoscale best practices and the Azure Monitor REST API reference documentation.
+    },
+    +    {
+      "type": "Microsoft.Insights/autoscaleSettings",
+      "apiVersion": "2015-04-01",
+      "name": "guestMetricsAutoscale",
+      "location": "[resourceGroup().location]",
+      "dependsOn": [
+        "Microsoft.Compute/virtualMachineScaleSets/myScaleSet"
+      ],
+      "properties": {
+        "name": "guestMetricsAutoscale",
+        "targetResourceUri": "[resourceId('Microsoft.Compute/virtualMachineScaleSets', 'myScaleSet')]",
+        "enabled": true,
+        "profiles": [  
+          {
+            "name": "Profile1",
+            "capacity": {
+              "minimum": "1",
+              "maximum": "10",
+              "default": "3"
+            },
+            "rules": [
+              {
+                "metricTrigger": {
+                  "metricName": "/builtin/memory/percentUsedMemory",
+                  "metricNamespace": "",
+                  "metricResourceUri": "[resourceId('Microsoft.Compute/virtualMachineScaleSets', 'myScaleSet')]",
+                  "timeGrain": "PT1M",
+                  "statistic": "Average",
+                  "timeWindow": "PT5M",
+                  "timeAggregation": "Average",
+                  "operator": "GreaterThan",
+                  "threshold": 60
+                },
+                "scaleAction": {
+                  "direction": "Increase",
+                  "type": "ChangeCount",
+                  "value": "1",
+                  "cooldown": "PT1M"
+                }
+              },
+              {
+                "metricTrigger": {
+                  "metricName": "/builtin/memory/percentUsedMemory",
+                  "metricNamespace": "",
+                  "metricResourceUri": "[resourceId('Microsoft.Compute/virtualMachineScaleSets', 'myScaleSet')]",
+                  "timeGrain": "PT1M",
+                  "statistic": "Average",
+                  "timeWindow": "PT5M",
+                  "timeAggregation": "Average",
+                  "operator": "LessThan",
+                  "threshold": 30
+                },
+                "scaleAction": {
+                  "direction": "Decrease",
+                  "type": "ChangeCount",
+                  "value": "1",
+                  "cooldown": "PT1M"
+                }
+              }
+            ]
+          }
+        ]
+      }
+    ]
}
Next steps

You can deploy the preceding template by following the Azure Resource Manager documentation.

You can start this tutorial series from the minimum viable scale set template article.

You can see how to modify the minimum viable scale set template to deploy the scale set into an existing virtual network.

You can see how to modify the minimum viable scale set template to deploy the scale set with a custom image.

You can see how to modify the minimum viable scale set template to deploy a Linux scale set with guest-based autoscale.

For more general information about scale sets, refer to the scale set overview page.
This tutorial shows you how easy it is to create a Virtual Machine Scale Set in just a few minutes, by using the Azure portal. If you don't have an Azure subscription, create a free account before you begin.

Choose the VM image from the marketplace

From the portal, you can easily deploy a scale set with CentOS, CoreOS, Debian, Ubuntu Server, other Linux images, and Windows Server images.

First, navigate to the Azure portal in a web browser. Click New, search for scale set, and then select the Virtual machine scale set entry:

Create the scale set

Now you can use the default settings and quickly create the scale set.

- Enter a name for the scale set.
  This name becomes the base of the FQDN of the load balancer in front of the scale set, so make sure the name is unique across all Azure.

- Select your desired OS type.

- Enter your desired username, and select which authentication type you prefer.
  If you choose a password, it must be at least 12 characters long and meet three out of the four following complexity requirements: one lower case character, one upper case character, one number, and one special character. See more about username and password requirements. If you choose SSH public key, be sure to only paste in your public key, NOT your private key:

- Select Yes or No to Enable scaling beyond 100 instances.
  If Yes, the scale set can span across multiple placement groups. For more information, see this documentation.

- Make sure to select an appropriate instance size.
For more information about virtual machine sizes, see either Windows VM sizes or Linux VM sizes.

- Enter your desired resource group name and location.
  If your region and instance size supports availability zones, the Availability zones field is enabled. For more information on availability zones, see this overview article.

- Enter your desired domain name label (the base of the FQDN for the load balancer in front of the scale set).
  This label must be unique across all Azure.

- Choose your desired operating system disk image, instance count, and machine size.

- Choose your desired disk type: managed or unmanaged.
  For more information, see this documentation. If you chose to have the scale set span multiple placement groups, this option will not be available because managed disk is required for scale sets to span placement groups.

- Enable or disable autoscale and configure if enabled.

Connect to a VM in the scale set
If you chose to limit your scale set to a single placement group, then the scale set is deployed with NAT rules configured to let you connect to the scale set easily (if not, to connect to the virtual machines in the scale set, you likely need to create a jumpbox in the same virtual network as the scale set). To see them, navigate to the Inbound NAT Rules tab of the load balancer for the scale set:

![Inbound NAT Rules tab](image)

You can connect to each VM in the scale set using these NAT rules. For instance, for a Windows scale set, if there is a NAT rule on incoming port 50000, you could connect to that machine via RDP on `<load-balancer-ip-address>:50000`. For a Linux scale set, you would connect using the command `ssh -p 50000 <username>@<load-balancer-ip-address>`.

**Next steps**

For documentation on how to deploy scale sets from the CLI, see [this documentation](#).

For documentation on how to deploy scale sets from PowerShell, see [this documentation](#).

For documentation on how to deploy scale sets from Visual Studio, see [this documentation](#).

For general documentation, check out the [documentation overview page for scale sets](#).

For general information, check out the [main landing page for scale sets](#).
How to create a Virtual Machine Scale Set with Visual Studio

7/13/2017 • 3 min to read • Edit Online

This article shows you how to deploy an Azure Virtual Machine Scale Set using a Visual Studio Resource Group Deployment.

**Azure Virtual Machine Scale Sets** is an Azure Compute resource to deploy and manage a collection of similar virtual machines with auto-scale and load balancing. You can provision and deploy Virtual Machine Scale Sets using **Azure Resource Manager Templates**. Azure Resource Manager Templates can be deployed using Azure CLI, PowerShell, REST and also directly from Visual Studio. Visual Studio provides a set of example templates, which can be deployed as part of an Azure Resource Group Deployment project.

Azure Resource Group deployments are a way to group and publish a set of related Azure resources in a single deployment operation. You can learn more about them here: Creating and deploying Azure resource groups through Visual Studio.

**Pre-requisites**

To get started deploying Virtual Machine Scale Sets in Visual Studio, you need the following:

- Visual Studio 2013 or later
- Azure SDK 2.7, 2.8 or 2.9

**NOTE**

These instructions assume you are using Visual Studio with Azure SDK 2.8.

**Creating a Project**

1. Create a new project in Visual Studio by choosing **File | New | Project**.

2. Under **Visual C# | Cloud**, choose **Azure Resource Manager** to create a project for deploying an Azure Resource Manager Template.
3. From the list of Templates, select either the Linux or Windows Virtual Machine Scale Set Template.

4. Once your project is created you see PowerShell deployment scripts, an Azure Resource Manager Template, and a parameter file for the Virtual Machine Scale Set.
Customize your project

Now you can edit the Template to customize it for your application's needs, such as adding VM extension properties or editing load balancing rules. By default the Virtual Machine Scale Set Templates are configured to deploy the AzureDiagnostics extension, which makes it easy to add autoscale rules. It also deploys a load balancer with a public IP address, configured with inbound NAT rules.

The load balancer lets you connect to the VM instances with SSH (Linux) or RDP (Windows). The front-end port range starts at 50000. For Linux this means that if you SSH to port 50000, you are routed to port 22 of the first VM in the Scale Set. Connecting to port 50001 is routed to port 22 of the second VM and so on.

A good way to edit your Templates with Visual Studio is to use the JSON Outline to organize the parameters, variables, and resources. With an understanding of the schema Visual Studio can point out errors in your Template before you deploy it.
Deploy the project

1. Deploy the Azure Resource Manager Template to create the Virtual Machine Scale Set resource. Right-click on the project node and choose **Deploy | New Deployment**.

2. Select your subscription in the “Deploy to Resource Group” dialog.
3. From here, you can create an Azure Resource Group to deploy your Template to.

4. Next, click **Edit Parameters** to enter parameters that are passed to your Template. Provide the username and password for the OS, which is required to create the deployment. If you don’t have PowerShell Tools for Visual Studio installed, it is recommended to check **Save passwords** to avoid a hidden PowerShell command-line prompt, or use **keyvault support**.
5. Now click **Deploy**. The **Output** window shows the deployment progress. Note that the action is executing the **Deploy-AzureResourceGroup.ps1** script.

**Exploring your Virtual Machine Scale Set**

Once the deployment completes, you can view the new Virtual Machine Scale Set in the Visual Studio **Cloud Explorer** (refresh the list). Cloud Explorer lets you manage Azure resources in Visual Studio while developing applications. You can also view your Virtual Machine Scale Set in the **Azure portal** and **Azure Resource Explorer**.
The portal provides the best way to visually manage your Azure infrastructure with a web browser, while Azure Resource Explorer provides an easy way to explore and debug Azure resources, giving a window into the "instance view" and also showing PowerShell commands for the resources you are looking at.

Next steps

Once you’ve successfully deployed Virtual Machine Scale Sets through Visual Studio, you can further customize your project to suit your application requirements. For example, configure auto-scale by adding an Insights resource, adding infrastructure to your Template (like standalone VMs), or deploying applications using the custom script extension. Good example templates can be found in the Azure Quickstart Templates GitHub repository (search for "vmss").
Virtual machine scale sets make it easy for you to deploy and manage identical virtual machines as a set. Scale sets provide a highly scalable and customizable compute layer for hyperscale applications, and they support Windows platform images, Linux platform images, custom images, and extensions. For more information about scale sets, see Virtual machine scale sets.

This tutorial shows you how to create a virtual machine scale set without using the Azure portal. For information about how to use the Azure portal, see How to create a virtual machine scale set with the Azure portal.

**NOTE**
For more information about Azure Resource Manager resources, see Azure Resource Manager vs. classic deployment.

## Sign in to Azure

If you're using Azure CLI 2.0 or Azure PowerShell to create a scale set, you first need to sign in to your subscription.

For more information about how to install, set up, and sign in to Azure with Azure CLI or PowerShell, see Getting Started with Azure CLI 2.0 or Get started with Azure PowerShell cmdlets.

```bash
az login
```

```powershell
Login-AzureRmAccount
```

## Create a resource group

You first need to create a resource group that the virtual machine scale set is associated with.

```bash
az group create --location westus2 --name MyResourceGroup1
```

```powershell
New-AzureRmResourceGroup -Location westus2 -Name MyResourceGroup1
```

## Create from Azure CLI

With Azure CLI, you can create a virtual machine scale set with minimal effort. If you omit default values, they are provided for you. For example, if you don’t specify any virtual network information, a virtual network is created for you. If you omit the following parts, they are created for you:

- A load balancer
- A virtual network
- A public IP address

When choosing the virtual machine image that you want to use on the virtual machine scale set, you have a few choices:
To create a virtual machine scale set, you must specify the following:

- Resource group
- Name
- Operating system image
- Authentication information

The following example creates a basic virtual machine scale set (this step might take a few minutes).

```
az vmss create --resource-group MyResourceGroup1 --name MyScaleSet --image UbuntuLTS --authentication-type password --admin-username azureuser --admin-password P@ssw0rd!
```

Once the command finishes you will now have your virtual machine scale set created. You may need to get the IP address of the virtual machine so that you can connect to it. You can get a lot of different information about the virtual machine (including the IP address) with the following command.

```
az vmss list-instance-connection-info --resource-group MyResourceGroup1 --name MyScaleSet
```

Create from PowerShell

PowerShell is more complicated to use than Azure CLI. While Azure CLI provides defaults for networking-related resources (such as load balancers, IP addresses, and virtual networks), PowerShell does not. Referencing an image with PowerShell is a slightly more complicated too. You can get images with the following cmdlets:

1. Get-AzureRmVMImagePublisher
2. Get-AzureRmVMImageOffer
3. Get-AzureRmVMImageSku

The cmdlets work can be piped in sequence. Here is an example of how to get all images for the West US 2 region with a publisher that has the name `microsoft` in it.
The workflow for creating a virtual machine scale set is as follows:

1. Create a config object that holds information about the scale set.
2. Reference the base OS image.
3. Configure the operating system settings: authentication, VM name prefix, and user/pass.
4. Configure networking.
5. Create the scale set.

This example creates a basic two-instance scale set for a computer that has Windows Server 2016 installed.
# Resource group name from above
$rg = "MyResourceGroup1"
$location = "WestUS2"

# Create a config object
$vmssConfig = New-AzureRmVmssConfig -Location $location -SkuCapacity 2 -SkuName Standard_A0 -UpgradePolicyMode Automatic

# Reference a virtual machine image from the gallery

# Set up information for authenticating with the virtual machine
Set-AzureRmVmssOsProfile $vmssConfig -AdminUsername azureuser -AdminPassword $@ssw0rd! -ComputerNamePrefix myvmssvm

# Create the virtual network resources

## Basics
$subnet = New-AzureRmVirtualNetworkSubnetConfig -Name "my-subnet" -AddressPrefix 10.0.0.0/24 $vnet = New-AzureRmVirtualNetwork -Name "my-network" -ResourceGroupName $rg -Location $location -AddressPrefix 10.0.0.0/16 -Subnet $subnet

## Load balancer
$publicIP = New-AzureRmPublicIpAddress -Name "PublicIP" -ResourceGroupName $rg -Location $location -AllocationMethod Static -DomainNameLabel "myuniquedomain"

New-AzureRmLoadBalancer -ResourceGroupName $rg -Name "LB1" -Location $location -FrontendIpConfiguration $frontendIP -LoadBalancingRule $inboundNATRule1 -InboundNatPool $inboundNATPool1 -BackendAddressPool $backendPool -Probe $probe

## IP address config
$ipConfig = New-AzureRmVmssIpConfig -Name "my-ipaddress" -LoadBalancerBackendAddressPoolsId $backendPool.Id -SubnetId $vnet.Subnets[0].Id -LoadBalancerInboundNatPoolsId $inboundNATPool1.Id

# Attach the virtual network to the IP object
Add-AzureRmVmssNetworkInterfaceConfiguration -VirtualMachineScaleSet $vmssConfig -Name "network-config" -Primary $true -IPConfiguration $ipConfig

# Create the scale set with the config object (this step might take a few minutes)
New-AzureRmVmss -ResourceGroupName $rg -Name "MyScaleSet1" -VirtualMachineScaleSet $vmssConfig

---

### Using a custom virtual machine image

If you are creating a scale set from your own custom image, instead of referencing a virtual machine image from the gallery, the `Set-AzureRmVmssStorageProfile` command would look like this:

```
Set-AzureRmVmssStorageProfile -OsDiskCreateOption FromImage -ManagedDisk PremiumLRS -OsDiskCaching "None" -OsDiskOsType Linux -ImageReferenceId (Get-AzureRmImage -ImageName $VMImage -ResourceGroupName $rg).id
```

### Create from a template

You can deploy a virtual machine scale set by using an Azure Resource Manager template. You can create your own template or use one from the [template repository](#). These templates can be deployed directly to your Azure...
A sample template is available on GitHub. For more information about how to create and use that sample, see Minimum viable scale set.

Create from Visual Studio

With Visual Studio, you can create an Azure resource group project and add a virtual machine scale set template to it. You can choose whether you want to import it from GitHub or the Azure Web Application Gallery. A deployment PowerShell script is also generated for you. For more information, see How to create a virtual machine scale set with Visual Studio.

Create from the Azure portal

The Azure portal provides a convenient way to quickly create a scale set. For more information, see How to create a virtual machine scale set with the Azure portal.

Next steps

Learn more about data disks.

Learn how to manage your apps.
Azure virtual machine scale sets supports virtual machines with managed disks. Using managed disks with scale sets has several benefits, including:

- You no longer need to pre-create and manage storage accounts to store the OS disks for the scale set VMs.
- You can attach managed data disks to the scale set.
- With managed disk, a scale set can have capacity as high as 1,000 VMs if based on a platform image or 300 VMs if based on a custom image.

**Get started**

A simple way to get started with managed disk scale sets is to deploy one from the Azure portal. For more information, see this article. Another simple way to get started is to use Azure CLI 2.0 to deploy a scale set. The following example shows how to create an Ubuntu based scale set with 10 VMs, each with a 50-GB and 100-GB data disk:

```
az group create -l southcentralus -n dsktest
az vmss create -g dsktest -n dskvmss --image ubuntults --instance-count 10 --data-disk-sizes-gb 50 100
```

Alternatively, you could look in the Azure Quickstart Templates GitHub repo for folders that contain `vmss` to see pre-built examples of templates that deploy scale sets. To tell which templates are already using managed disks, you can refer to this list.

**Next steps**

For more information on managed disks in general, see this article.

To see how to convert a Resource Manager template to provision scale sets with managed disks, see this article. The same modifications to the Resource Manager templates apply to the Azure REST API as well.

To learn more about using managed data disks with scale sets, see this article.

To begin working with large scale sets, refer to this article.
An Azure virtual machine scale set can automatically increase or decrease the number of VM instances that run your application. This automated and elastic behavior reduces the management overhead to monitor and optimize the performance of your application. You create rules that define the minimally acceptable performance for a positive customer experience. When those defined thresholds are met, autoscale rules take action to adjust the capacity of your scale set. You can also schedule events to automatically increase or decrease the capacity of your scale set at fixed times. This article provides an overview of which performance metrics are available and what actions autoscale can perform.

**Benefits of autoscale**

If your application demand increases, the load on the VM instances in your scale set increases. If this increased load is consistent, rather than just a brief demand, you can configure autoscale rules to increase the number of VM instances in the scale set.

When these VM instances are created and your applications are deployed, the scale set starts to distribute traffic to them through the load balancer. You control what metrics to monitor, such as CPU or memory, how long the application load must meet a given threshold, and how many VM instances to add to the scale set.

On an evening or weekend, your application demand may decrease. If this decreased load is consistent over a period of time, you can configure autoscale rules to decrease the number of VM instances in the scale set. This scale-in action reduces the cost to run your scale set as you only run the number of instances required to meet the current demand.

**Use host-based metrics**

You can create autoscale rules that built-in host metrics available from your VM instances. Host metrics give you visibility into the performance of the VM instances in a scale set without the need to install or configure additional agents and data collections. Autoscale rules that use these metrics can scale out or in the number of VM instances in response to CPU usage, memory demand, or disk access.

Autoscale rules that use host-based metrics can be created with one of the following tools:

- Azure portal
- Azure PowerShell
- Azure CLI 2.0

To create autoscale rules that use more detailed performance metrics, you can install and configure the Azure diagnostics extension on VM instances, or configure your application use App Insights.

Autoscale rules that use host-based metrics, in-guest VM metrics with the Azure diagnostic extension, and App Insights can use the following configuration settings.

**Metric sources**

Autoscale rules can use metrics from one of the following sources:
### METRIC SOURCE
<table>
<thead>
<tr>
<th>Current scale set</th>
<th>For host-based metrics that do not require additional agents to be installed or configured.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage account</td>
<td>The Azure diagnostic extension writes performance metrics to Azure storage that is then consumed to trigger autoscale rules.</td>
</tr>
<tr>
<td>Service Bus Queue</td>
<td>Your application or other components can transmit messages on an Azure Service Bus queue to trigger rules.</td>
</tr>
<tr>
<td>Application Insights</td>
<td>An instrumentation package installed in your application that streams metrics directly from the app.</td>
</tr>
</tbody>
</table>

### Autoscale rule criteria
The following host-based metrics are available for use when you create autoscale rules. If you use the Azure diagnostic extension or App Insights, you define which metrics to monitor and use with autoscale rules.

<table>
<thead>
<tr>
<th>METRIC NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage CPU</td>
</tr>
<tr>
<td>Network In</td>
</tr>
<tr>
<td>Network Out</td>
</tr>
<tr>
<td>Disk Read Bytes</td>
</tr>
<tr>
<td>Disk Write Bytes</td>
</tr>
<tr>
<td>Disk Read Operations/Sec</td>
</tr>
<tr>
<td>Disk Write Operations/Sec</td>
</tr>
<tr>
<td>CPU Credits Remaining</td>
</tr>
<tr>
<td>CPU Credits Consumed</td>
</tr>
</tbody>
</table>

When you create autoscale rules to monitor a given metric, the rules look at one of the following metrics aggregation actions:

<table>
<thead>
<tr>
<th>AGGREGATION TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
</tr>
<tr>
<td>Minimum</td>
</tr>
<tr>
<td>Maximum</td>
</tr>
<tr>
<td>Total</td>
</tr>
<tr>
<td>Last</td>
</tr>
</tbody>
</table>
### AGGREGATION TYPE

- Count

The autoscale rules are then triggered when the metrics are compared against your defined threshold with one of the following operators:

### OPERATOR

- Greater than
- Greater than or equal to
- Less than
- Less than or equal to
- Equal to
- Not equal to

### Actions when rules trigger

When an autoscale rule triggers, your scale set can automatically scale in one of the following ways:

<table>
<thead>
<tr>
<th>SCALE OPERATION</th>
<th>USE CASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase count by</td>
<td>A fixed number of VM instances to create. Useful in scale sets with a smaller number of VMs.</td>
</tr>
<tr>
<td>Increase percent by</td>
<td>A percentage-based increase of VM instances. Good for larger scale sets where a fixed increase may not noticeably improve performance.</td>
</tr>
<tr>
<td>Increase count to</td>
<td>Create as many VM instances are required to reach a desired maximum amount.</td>
</tr>
<tr>
<td>Decrease count to</td>
<td>A fixed number of VM instances to remove. Useful in scale sets with a smaller number of VMs.</td>
</tr>
<tr>
<td>Decrease percent by</td>
<td>A percentage-based decrease of VM instances. Good for larger scale sets where a fixed increase may not noticeably reduce resource consumption and costs.</td>
</tr>
<tr>
<td>Decrease count to</td>
<td>Remove as many VM instances are required to reach a desired minimum amount.</td>
</tr>
</tbody>
</table>

### In-guest VM metrics with the Azure diagnostics extension

The Azure diagnostics extension is an agent that runs inside a VM instance. The agent monitors and saves performance metrics to Azure storage. These performance metrics contain more detailed information about the status of the VM, such as *AverageReadTime* for disks or *PercentIdleTime* for CPU. You can create autoscale rules based on a more detailed awareness of the VM performance, not just the percentage of CPU usage or memory consumption.
To use the Azure diagnostics extension, you must create Azure storage accounts for your VM instances, install the Azure diagnostics agent, then configure the VMs to stream specific performance counters to the storage account.

For more information, see the articles for how to enable the Azure diagnostics extension on a Linux VM or Windows VM.

**Application-level metrics with App Insights**

To gain more visibility into the performance of your applications, you can use Application Insights. You install a small instrumentation package in your application that monitors the app and sends telemetry to Azure. You can monitor metrics such as the response times of your application, the page load performance, and the session counts. These application metrics can be used to create autoscale rules at a granular and embedded level as you trigger rules based on actionable insights that may impact the customer experience.

For more information about App Insights, see [What is Application Insights](#).

**Scheduled autoscale**

You can also create autoscale rules based on schedules. These schedule-based rules allow you to automatically scale the number of VM instances at fixed times. With performance-based rules, there may be a performance impact on the application before the autoscale rules trigger and the new VM instances are provisioned. If you can anticipate such demand, the additional VM instances are provisioned and ready for the additional customer use and application demand.

The following examples are scenarios that may benefit the use of schedule-based autoscale rules:

- **Automatically scale out the number of VM instances at the start of the work day when customer demand increases.** At the end of the work day, automatically scale in the number of VM instances to minimize resource costs overnight when application use is low.
- **If a department uses an application heavily at certain parts of the month or fiscal cycle,** automatically scale the number of VM instances to accommodate their additional demands.
- **When there is a marketing event, promotion, or holiday sale,** you can automatically scale the number of VM instances ahead of anticipated customer demand.

**Next steps**

You can create autoscale rules that use host-based metrics with one of the following tools:

- Azure portal
- Azure PowerShell
- Azure CLI 2.0

This overview detailed how to use autoscale rules to scale horizontally and increase or decrease the number of VM instances in your scale set. You can also scale vertically to increase or decrease the VM instance size. For more information, see [Vertical autoscale with Virtual Machine Scale sets](#).

For information on how to manage your VM instances, see [Manage virtual machine scale sets with Azure PowerShell](#).

To learn how to generate alerts when your autoscale rules trigger, see [Use autoscale actions to send email and webhook alert notifications in Azure Monitor](#). You can also [Use audit logs to send email and webhook alert notifications in Azure Monitor](#).
When you create a scale set, you define the number of VM instances that you wish to run. As your application demand changes, you can automatically increase or decrease the number of VM instances. The ability to autoscale lets you keep up with customer demand or respond to application performance changes throughout the lifecycle of your app.

This article shows you how to create autoscale rules in the Azure portal that monitor the performance of the VM instances in your scale set. These autoscale rules increase or decrease the number of VM instances in response to these performance metrics. You can also complete these steps with Azure PowerShell or the Azure CLI 2.0.

Prerequisites

To create autoscale rules, you need an existing virtual machine scale set. You can create a scale set with the Azure portal, Azure PowerShell, or Azure CLI 2.0.

Create a rule to automatically scale out

If your application demand increases, the load on the VM instances in your scale set increases. If this increased load is consistent, rather than just a brief demand, you can configure autoscale rules to increase the number of VM instances in the scale set. When these VM instances are created and your applications are deployed, the scale set starts to distribute traffic to them through the load balancer. You control what metrics to monitor, such as CPU or disk, how long the application load must meet a given threshold, and how many VM instances to add to the scale set.

1. Open the Azure portal and select Resource groups from the menu on the left-hand side of the dashboard.
2. Select the resource group that contains your scale set, then choose your scale set from the list of resources.
3. Choose Scaling from the menu on the left-hand side of the scale set window. Select the button to Enable autoscale.
4. Enter a name for your settings, such as *autoscale*, then select the option to **Add a rule**.

5. Let’s create a rule that increases the number of VM instances in a scale set when the average CPU load is greater than 70% over a 10-minute period. When the rule triggers, the number of VM instances is increased by 20%. In scale sets with a small number of VM instances, you could set the **Operation** to *Increase count by* and then specify 1 or 2 for the **Instance count**. In scale sets with a large number of VM instances, an increase of 10% or 20% VM instances may be more appropriate.

Specify the following settings for your rule:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>EXPLANATION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Aggregation</td>
<td>Defines how the collected metrics should be aggregated for analysis.</td>
<td>Average</td>
</tr>
<tr>
<td>Metric Name</td>
<td>The performance metric to monitor and apply scale set actions on.</td>
<td>Percentage CPU</td>
</tr>
<tr>
<td>Time grain statistic</td>
<td>Defines how the collected metrics in each time grain should be aggregated for analysis.</td>
<td>Average</td>
</tr>
<tr>
<td>Operator</td>
<td>Operator used to compare the metric data against the threshold.</td>
<td>Greater than</td>
</tr>
<tr>
<td>Threshold</td>
<td>The percentage that causes the autoscale rule to trigger an action.</td>
<td>70</td>
</tr>
<tr>
<td>PARAMETER</td>
<td>EXPLANATION</td>
<td>VALUE</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Duration</td>
<td>The amount of time monitored before the metric and threshold values are compared.</td>
<td>10 minutes</td>
</tr>
<tr>
<td>Operation</td>
<td>Defines if the scale set should scale up or down when the rule applies and by what increment</td>
<td>Increase percent by</td>
</tr>
<tr>
<td>Instance count</td>
<td>The percentage of VM instances should be changed when the rule triggers.</td>
<td>20</td>
</tr>
<tr>
<td>Cool down (minutes)</td>
<td>The amount of time to wait before the rule is applied again so that the autoscale actions have time to take effect.</td>
<td>5 minutes</td>
</tr>
</tbody>
</table>

The following examples show a rule created in the Azure portal that matches these settings:
Create a rule to automatically scale in

On an evening or weekend, your application demand may decrease. If this decreased load is consistent over a period of time, you can configure autoscale rules to decrease the number of VM instances in the scale set. This scale-in action reduces the cost to run your scale set as you only run the number of instances required to meet the current demand.

1. Choose to **Add a rule** again.
2. Create a rule that decreases the number of VM instances in a scale set when the average CPU load then drops below 30% over a 10-minute period. When the rule triggers, the number of VM instances is decreased by 20%.

   Use the same approach as with the previous rule. Adjust the following settings for your rule:

<table>
<thead>
<tr>
<th>Metric source</th>
<th>Current resource (myScaleSet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource type</td>
<td>Virtual machine scale sets</td>
</tr>
<tr>
<td>Resource</td>
<td>myScaleSet</td>
</tr>
<tr>
<td>Criteria</td>
<td></td>
</tr>
<tr>
<td>Time aggregation (1 minute time grain)</td>
<td>Average</td>
</tr>
<tr>
<td>Metric name</td>
<td>Percentage CPU</td>
</tr>
<tr>
<td>Time grain statistic (Average)</td>
<td></td>
</tr>
<tr>
<td>Operator (Greater than)</td>
<td></td>
</tr>
<tr>
<td>Threshold</td>
<td></td>
</tr>
<tr>
<td>Duration (in minutes)</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
</tr>
<tr>
<td>Operation (Increase percent by 20%)</td>
<td></td>
</tr>
<tr>
<td>Instance count</td>
<td>20</td>
</tr>
<tr>
<td>Cool down (minutes)</td>
<td>5</td>
</tr>
</tbody>
</table>

6. To create the rule, select **Add**
Define autoscale instance limits

Your autoscale profile must define a minimum, maximum, and default number of VM instances. When your autoscale rules are applied, these instance limits make sure that you do not scale out beyond the maximum number of instances, or scale in beyond the minimum of instances.

1. Set the following instance limits:

<table>
<thead>
<tr>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>DEFAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

2. To apply your autoscale rules and instance limits, select **Save**.

Monitor number of instances in a scale set

To see the number and status of VM instances, select **Instances** from the menu on the left-hand side of the scale set window. The status indicates if the VM instance is **Creating** as the scale set automatically scales out, or is **Deleting** as the scale automatically scales in.

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>LATEST MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>myScaleSet_1</td>
<td>Running</td>
<td>Yes</td>
</tr>
<tr>
<td>myScaleSet_2</td>
<td>Running</td>
<td>Yes</td>
</tr>
<tr>
<td>myScaleSet_4</td>
<td>Creating</td>
<td>Yes</td>
</tr>
<tr>
<td>myScaleSet_5</td>
<td>Creating</td>
<td>Yes</td>
</tr>
<tr>
<td>myScaleSet_6</td>
<td>Creating</td>
<td>Yes</td>
</tr>
<tr>
<td>myScaleSet_7</td>
<td>Creating</td>
<td>Yes</td>
</tr>
</tbody>
</table>

3. To create the rule, select **Add**

**Operator**
Operator used to compare the metric data against the threshold.

**Threshold**
The percentage that causes the autoscale rule to trigger an action.

**Operation**
Defines if the scale set should scale up or down when the rule applies and by what increment

**Instance count**
The percentage of VM instances should be changed when the rule triggers.
Autoscale based on a schedule

The previous examples automatically scaled a scale set in or out with basic host metrics such as CPU usage. You can also create autoscale rules based on schedules. These schedule-based rules allow you to automatically scale out the number of VM instances ahead of an anticipated increase in application demand, such as core work hours, and then automatically scale in the number of instances at a time that you anticipate less demand, such as the weekend.

1. Choose **Scaling** from the menu on the left-hand side of the scale set window. To delete the existing autoscale rules created in the previous examples, choose the trash can icon.

![Auto scale by percentage based on CPU usage](image)

2. Choose to **Add a scale condition**. Select the pencil icon next to rule name, and provide a name such as **Scale out during each work day**.

![Scale out during each work day](image)

3. Select the radio button to **Scale to a specific instance count**.

4. To scale up the number of instances, enter 10 as the instance count.

5. Choose **Repeat specific days** for the **Schedule** type.

6. Select all the work days, Monday through Friday.

7. Choose the appropriate timezone, then specify a **Start time** of 09:00.

8. Choose to **Add a scale condition** again. Repeat the process to create a schedule named **Scale in during the evening** that scales to 3 instances, repeats every weekday, and starts at 18:00.

9. To apply your schedule-based autoscale rules, select **Save**.
To see how your autoscale rules are applied, select Run history across the top of the Scaling window. The graph and events list shows when the autoscale rules trigger and the number of VM instances in your scale increases or decreases.

Next steps

In this article, you learned how to use autoscale rules to scale horizontally and increase or decrease the number of VM instances in your scale set. You can also scale vertically to increase or decrease the VM instance size. For more information, see Vertical autoscale with Virtual Machine Scale sets.

For information on how to manage your VM instances, see Manage virtual machine scale sets with Azure PowerShell.

To learn how to generate alerts when your autoscale rules trigger, see Use autoscale actions to send email and webhook alert notifications in Azure Monitor. You can also Use audit logs to send email and webhook alert notifications in Azure Monitor.
Automatically scale a virtual machine scale set with the Azure CLI 2.0

10/20/2017 • 8 min to read • Edit Online

When you create a scale set, you define the number of VM instances that you wish to run. As your application demand changes, you can automatically increase or decrease the number of VM instances. The ability to autoscale lets you keep up with customer demand or respond to application performance changes throughout the lifecycle of your app.

This article shows you how to create autoscale rules with the Azure CLI 2.0 that monitor the performance of the VM instances in your scale set. These autoscale rules increase or decrease the number of VM instances in response to these performance metrics. You can also complete these steps with Azure PowerShell or in the Azure portal.

Prerequisites

To create autoscale rules, you need an existing virtual machine scale set. You can create a scale set with the Azure portal, Azure CLI 2.0, or Azure PowerShell.

To make it easier to create the autoscale rules, define some variables for your scale set. The following example defines variables for the scale set named *myScaleSet* in the resource group named *myResourceGroup* and in the *eastus* region. Your subscription ID is obtained with `az account show`. If you have multiple subscriptions associated with your account, only the first subscription is returned. Adjust the names and subscription ID as follows:

```bash
sub=$(az account show --query id -o tsv)
resourcegroup_name="myResourceGroup"
scaleset_name="myScaleSet"
location_name="eastus"
```

Define an autoscale profile

Autoscale rules are deployed as JSON (JavaScript Object Notation) with the Azure CLI 2.0. The complete JSON that defines and deploys the autoscale rules can be found later in the article.

The start of the autoscale profile defines the default, minimum, and maximum scale set capacity. The following example sets the default, and minimum, capacity of 2 VM instances, and a maximum of 10:

```json
{
    "name": "autoscale rules",
    "capacity": {
        "minimum": "2",
        "maximum": "10",
        "default": "2"
    }
}
```

Create a rule to automatically scale out

If your application demand increases, the load on the VM instances in your scale set increases. If this increased load is consistent, rather than just a brief demand, you can configure autoscale rules to increase the number of VM instances in the scale set. When these VM instances are created and your applications are deployed, the scale set
starts to distribute traffic to them through the load balancer. You control what metrics to monitor, such as CPU or disk, how long the application load must meet a given threshold, and how many VM instances to add to the scale set.

Let’s create a rule that increases the number of VM instances in a scale set when the average CPU load is greater than 70% over a 10-minute period. When the rule triggers, the number of VM instances is increased by 20%. In scale sets with a small number of VM instances, you could set the type to ChangeCount and increase the value by 1 or 2 instances. In scale sets with a large number of VM instances, an increase of 10% or 20% VM instances may be more appropriate.

The following parameters are used for this rule:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>EXPLANATION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>metricName</td>
<td>The performance metric to monitor and apply scale set actions on.</td>
<td>Percentage CPU</td>
</tr>
<tr>
<td>timeGrain</td>
<td>How often the metrics are collected for analysis.</td>
<td>1 minute</td>
</tr>
<tr>
<td>timeAggregation</td>
<td>Defines how the collected metrics should be aggregated for analysis.</td>
<td>Average</td>
</tr>
<tr>
<td>timeWindow</td>
<td>The amount of time monitored before the metric and threshold values are compared.</td>
<td>10 minutes</td>
</tr>
<tr>
<td>operator</td>
<td>Operator used to compare the metric data against the threshold.</td>
<td>Greater Than</td>
</tr>
<tr>
<td>threshold</td>
<td>The value that causes the autoscale rule to trigger an action.</td>
<td>70%</td>
</tr>
<tr>
<td>direction</td>
<td>Defines if the scale set should scale up or down when the rule applies.</td>
<td>Increase</td>
</tr>
<tr>
<td>type</td>
<td>Indicates that the number of VM instances should be changed by a percentage amount.</td>
<td>Percent Change</td>
</tr>
<tr>
<td>value</td>
<td>How many VM instances should be scaled up or down when the rule applies.</td>
<td>20</td>
</tr>
<tr>
<td>cooldown</td>
<td>The amount of time to wait before the rule is applied again so that the autoscale actions have time to take effect.</td>
<td>5 minutes</td>
</tr>
</tbody>
</table>

The following example defines the rule to scale out the number of VM instances. The metricResourceUri uses the variables previously defined for the subscription ID, resource group name, and scale set name:
Create a rule to automatically scale in

On an evening or weekend, your application demand may decrease. If this decreased load is consistent over a period of time, you can configure autoscale rules to decrease the number of VM instances in the scale set. This scale-in action reduces the cost to run your scale set as you only run the number of instances required to meet the current demand.

Create another rule that decreases the number of VM instances in a scale set when the average CPU load then drops below 30% over a 10-minute period. The following example defines the rule to scale out the number of VM instances. The `metricResourceUri` uses the variables previously defined for the subscription ID, resource group name, and scale set name:

```
{
  "metricTrigger": {
    "metricName": "Percentage CPU",
    "metricNamespace": "",
    "metricResourceUri": "/subscriptions/'$sub'/resourceGroups/'$resourcegroup_name'/providers/Microsoft.Compute/virtualMachineScaleSets/'$scaleset_name'",
    "metricResourceLocation": "'$location_name'",
    "timeGrain": "PT1M",
    "statistic": "Average",
    "timeWindow": "PT10M",
    "timeAggregation": "Average",
    "operator": "GreaterThan",
    "threshold": 70
  },
  "scaleAction": {
    "direction": "Increase",
    "type": "PercentChangeCount",
    "value": "20",
    "cooldown": "PT5M"
  }
}
```

Apply autoscale rules to a scale set

The final step is to apply the autoscale profile and rules to your scale set. Your scale is then able to automatically
scale in or out based on the application demand. Apply the autoscale profile with `az monitor autoscale-settings create` as follows. The complete JSON uses the profile and rules noted in the previous sections.

```bash
az monitor autoscale-settings create \
--resource-group myResourceGroup \
--name autoscale \
--parameters '{"autoscale_setting_resource_name": "autoscale", 
 "enabled": true, 
 "location": "'$location_name'", 
 "notifications": [], 
 "profiles": [ 
  { 
   "name": "autoscale by percentage based on CPU usage", 
   "capacity": { 
    "minimum": "2", 
    "maximum": "10", 
    "default": "2" 
   }, 
   "rules": [ 
    { 
      "metricTrigger": { 
        "metricName": "Percentage CPU", 
        "metricNamespace": "", 
        "metricResourceUri": "/subscriptions/'$sub'/resourceGroups/'$resourcegroup_name'/providers/Microsoft.Compute/virtualMachineScaleSets/'$scaleset_name' " 
      }, 
      "scaleAction": { 
        "direction": "Increase", 
        "type": "PercentChangeCount", 
        "value": "20", 
        "cooldown": "PT5M" 
      } 
    }, 
    { 
      "metricTrigger": { 
        "metricName": "Percentage CPU", 
        "metricNamespace": "", 
        "metricResourceUri": "/subscriptions/'$sub'/resourceGroups/'$resourcegroup_name'/providers/Microsoft.Compute/virtualMachineScaleSets/'$scaleset_name' " 
      }, 
      "scaleAction": { 
        "direction": "Decrease", 
        "type": "PercentChangeCount", 
        "value": "20", 
        "cooldown": "PT5M" 
      } 
    } 
   ]
  }
]
, 
"tags": {} 
} 
```

Monitor number of instances in a scale set

To see the number and status of VM instances, view a list of instances in your scale set with `az vmss list-instances`. The status indicates if the VM instance is provisioning as the scale set automatically scales out, or is deprovisioning as the scale automatically scales in. The following example views the VM instance status for the scale set named `myScaleSet` in the resource group named `myResourceGroup`:

```
az vmss list-instances
--resource-group myResourceGroup
--name myScaleSet
--output table
```

Autoscale based on a schedule

The previous examples automatically scaled a scale set in or out with basic host metrics such as CPU usage. You can also create autoscale rules based on schedules. These schedule-based rules allow you to automatically scale out the number of VM instances ahead of an anticipated increase in application demand, such as core work hours, and then automatically scale in the number of instances at a time that you anticipate less demand, such as the weekend.

To use schedule-based autoscale rules, create a JSON profile that defines the number of VM instances to run for a fixed start and end time window. The following example defines a rule to scale out to 10 instances at 9 A.M each work day (Monday through Friday).

```
{
  "name": "Scale out during each work day",
  "capacity": {
    "minimum": "10",
    "maximum": "10",
    "default": "10"
  },
  "rules": [],
  "recurrence": {
    "frequency": "Week",
    "schedule": {
      "timeZone": "Pacific Standard Time",
      "days": [
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday"
      ],
      "hours": [9],
      "minutes": [0]
    }
  }
}
```

To scale in during the evening, create another rule that specifies a lower number of VM instances and an
The following complete example defines the rules to scale out and then scale in, then applies the autoscale profile with `az monitor autoscale-settings create`. This example overwrites the metric-based autoscale rules created in the previous examples. The `metricResourceUri` uses the variables previously defined for the subscription ID, resource group name, and scale set name:

```
az monitor autoscale-settings create \
--resource-group myResourceGroup \
--name autoscale \
--parameters '{"autoscale_setting_resource_name": "autoscale", 
"enabled": true,
"location": "'$location_name'",
"notifications": [],
"profiles": [ 
  {
    "name": "Scale out during each work day",
    "capacity": {
      "minimum": "10",
      "maximum": "10",
      "default": "10"
    },
    "rules": [],
    "recurrence": {
      "frequency": "Week",
      "schedule": {
        "timeZone": "Pacific Standard Time",
        "days": [
          "Monday",
          "Tuesday",
          "Wednesday",
          "Thursday",
          "Friday"
        ],
        "hours": [
          9
        ],
        "minutes": [
          0
        ]
      }
    }
  },
  {
    "name": "Scale in during the evening",
    "capacity": {
      "minimum": "3",
      "maximum": "3",
      "default": "3"
    },
    "rules": [],
    "recurrence": {
      "frequency": "Week",
      "schedule": {
        "timeZone": "Pacific Standard Time",
        "days": [
          "Monday",
          "Tuesday",
          "Wednesday",
          "Thursday",
          "Friday"
        ],
        "hours": [
          18
        ],
        "minutes": [
          0
        ]
      }
    }
  }
]'
```
Next steps

In this article, you learned how to use autoscale rules to scale horizontally and increase or decrease the number of VM instances in your scale set. You can also scale vertically to increase or decrease the VM instance size. For more information, see Vertical autoscale with virtual machine scale sets.

For information on how to manage your VM instances, see Manage virtual machine scale sets with Azure PowerShell.

To learn how to generate alerts when your autoscale rules trigger, see Use autoscale actions to send email and webhook alert notifications in Azure Monitor. You can also Use audit logs to send email and webhook alert notifications in Azure Monitor.
Automatically scale a virtual machine scale set with Azure PowerShell

10/20/2017 • 6 min to read • Edit Online

When you create a scale set, you define the number of VM instances that you wish to run. As your application demand changes, you can automatically increase or decrease the number of VM instances. The ability to autoscale lets you keep up with customer demand or respond to application performance changes throughout the lifecycle of your app.

This article shows you how to create autoscale rules with Azure PowerShell that monitor the performance of the VM instances in your scale set. These autoscale rules increase or decrease the number of VM instances in response to these performance metrics. You can also complete these steps with the Azure CLI 2.0 or in the Azure portal.

Prerequisites

To create autoscale rules, you need an existing virtual machine scale set. You can create a scale set with the Azure portal, Azure PowerShell, or Azure CLI 2.0.

To make it easier to create the autoscale rules, define some variables for your scale set. The following example defines variables for the scale set named `myScaleSet` in the resource group named `myResourceGroup` and in the `East US` region. Your subscription ID is obtained with `Get-AzureRmSubscription`. If you have multiple subscriptions associated with your account, only the first subscription is returned. Adjust the names and subscription ID as follows:

```
$mySubscriptionId = (Get-AzureRmSubscription).Id
$myResourceGroup = "myResourceGroup"
$myScaleSet = "myScaleSet"
$myLocation = "East US"
```

Create a rule to automatically scale out

If your application demand increases, the load on the VM instances in your scale set increases. If this increased load is consistent, rather than just a brief demand, you can configure autoscale rules to increase the number of VM instances in the scale set. When these VM instances are created and your applications are deployed, the scale set starts to distribute traffic to them through the load balancer. You control what metrics to monitor, such as CPU or disk, how long the application load must meet a given threshold, and how many VM instances to add to the scale set.

Let's create a rule with `New-AzureRmAutoscaleRule` that increases the number of VM instances in a scale set when the average CPU load is greater than 70% over a 5-minute period. When the rule triggers, the number of VM instances is increased by 20%. In scale sets with a small number of VM instances, you could leave out `-ScaleActionScaleType` and only specify `-ScaleActionValue` to increase by 1 or 2 instances. In scale sets with a large number of VM instances, an increase of 10% or 20% VM instances may be more appropriate.

The following parameters are used for this rule:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>EXPLANATION</th>
<th>VALUE</th>
</tr>
</thead>
</table>

- ScaleActionScaleType
- ScaleActionValue
<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>EXPLANATION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>-MetricName</td>
<td>The performance metric to monitor and apply scale set actions on.</td>
<td>Percentage CPU</td>
</tr>
<tr>
<td>-TimeGrain</td>
<td>How often the metrics are collected for analysis.</td>
<td>1 minute</td>
</tr>
<tr>
<td>-MetricStatistic</td>
<td>Defines how the collected metrics should be aggregated for analysis.</td>
<td>Average</td>
</tr>
<tr>
<td>-TimeWindow</td>
<td>The amount of time monitored before the metric and threshold values are compared.</td>
<td>10 minutes</td>
</tr>
<tr>
<td>-Operator</td>
<td>Operator used to compare the metric data against the threshold.</td>
<td>Greater Than</td>
</tr>
<tr>
<td>-Threshold</td>
<td>The value that causes the autoscale rule to trigger an action.</td>
<td>70%</td>
</tr>
<tr>
<td>-ScaleActionDirection</td>
<td>Defines if the scale set should scale up or down when the rule applies.</td>
<td>Increase</td>
</tr>
<tr>
<td>-ScaleActionScaleType</td>
<td>Indicates that the number of VM instances should be changed by a percentage amount.</td>
<td>Percent Change</td>
</tr>
<tr>
<td>-ScaleActionValue</td>
<td>The percentage of VM instances should be changed when the rule triggers.</td>
<td>20</td>
</tr>
<tr>
<td>-ScaleActionCooldown</td>
<td>The amount of time to wait before the rule is applied again so that the autoscale actions have time to take effect.</td>
<td>5 minutes</td>
</tr>
</tbody>
</table>

The following example creates an object named `myRuleScaleOut` that holds this scale up rule. The `-MetricResourceId` uses the variables previously defined for the subscription ID, resource group name, and scale set name:

```powershell
$myRuleScaleOut = New-AzureRmAutoscaleRule `   -MetricName "Percentage CPU" `   -MetricResourceId `/subscriptions/$mySubscriptionId/resourceGroups/$myResourceGroup/providers/Microsoft.Compute/virtualMachineScaleSets/$myScaleSet `   -TimeGrain 00:01:00 `   -MetricStatistic Average `   -TimeWindow 00:10:00 `   -Operator GreaterThan `   -Threshold 70 `   -ScaleActionDirection Increase `   -ScaleActionScaleType PercentChangeCount `   -ScaleActionValue 20 `   -ScaleActionCooldown 00:05:00
```

Create a rule to automatically scale in
On an evening or weekend, your application demand may decrease. If this decreased load is consistent over a period of time, you can configure autoscale rules to decrease the number of VM instances in the scale set. This scale-in action reduces the cost to run your scale set as you only run the number of instances required to meet the current demand.

Create a rule with `New-AzureRmAutoscaleRule` that decreases the number of VM instances in a scale set when the average CPU load then drops below 30% over a 10-minute period. When the rule triggers, the number of VM instances is decreased by 20%. The following example creates an object named `myRuleScaleIn` that holds this scale up rule.

```powershell
$myRuleScaleIn = New-AzureRmAutoscaleRule `-MetricName "Percentage CPU" `-MetricResourceId /subscriptions/$mySubscriptionId/resourceGroups/$myResourceGroup/providers/Microsoft.Compute/virtualMachineScaleSets/$myScaleSet`-Operator LessThan `-MetricStatistic Average `-Threshold 30 `-TimeGrain 00:01:00 `-TimeWindow 00:10:00 `-ScaleActionCooldown 00:05:00 `-ScaleActionDirection Decrease `-ScaleActionScaleType PercentChangeCount `-ScaleActionValue 20
```

Define an autoscale profile

To associate your autoscale rules with a scale set, you create a profile. The autoscale profile defines the default, minimum, and maximum scale set capacity, and associates your autoscale rules. Create an autoscale profile with `New-AzureRmAutoscaleProfile`. The following example sets the default, and minimum, capacity of 2 VM instances, and a maximum of 10. The scale out and scale in rules created in the preceding steps are then attached:

```powershell
$myScaleProfile = New-AzureRmAutoscaleProfile `-DefaultCapacity 2` `-MaximumCapacity 10` `-MinimumCapacity 2` `-Rules $myRuleScaleOut,$myRuleScaleIn` `-Name "autoprofile"
```

Apply autoscale rules to a scale set

The final step is to apply the autoscale profile to your scale set. Your scale is then able to automatically scale in or out based on the application demand. Apply the autoscale profile with `Add-AzureRmAutoscaleSetting` as follows:

```powershell
Add-AzureRmAutoscaleSetting ` -Location $myLocation` -Name "autosetting"` -ResourceGroup $myResourceGroup` -TargetResourceId /subscriptions/$mySubscriptionId/resourceGroups/$myResourceGroup/providers/Microsoft.Compute/virtualMachineScaleSets/$myScaleSet` -AutoscaleProfiles $myScaleProfile
```

Monitor number of instances in a scale set
To see the number and status of VM instances, view a list of instances in your scale set with `Get-AzureRmVmssVM`. The status indicates if the VM instance is provisioning as the scale set automatically scales out, or is deprovisioning as the scale automatically scales in. The following example views the VM instance status for the scale set named `myScaleSet` in the resource group named `myResourceGroup`:

```
Get-AzureRmVmssVM -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet"
```

### Autoscale based on a schedule

The previous examples automatically scaled a scale set in or out with basic host metrics such as CPU usage. You can also create autoscale rules based on schedules. These schedule-based rules allow you to automatically scale out the number of VM instances ahead of an anticipated increase in application demand, such as core work hours, and then automatically scale in the number of instances at a time that you anticipate less demand, such as the weekend.

To create autoscale rules based on a schedule rather than host metrics, use the Azure portal. Schedule-based rules cannot currently be created with Azure PowerShell.

### Next steps

In this article, you learned how to use autoscale rules to scale horizontally and increase or decrease the number of VM instances in your scale set. You can also scale vertically to increase or decrease the VM instance size. For more information, see [Vertical autoscale with Virtual Machine Scale sets](#).

For information on how to manage your VM instances, see [Manage virtual machine scale sets with Azure PowerShell](#).

To learn how to generate alerts when your autoscale rules trigger, see [Use autoscale actions to send email and webhook alert notifications in Azure Monitor](#). You can also [Use audit logs to send email and webhook alert notifications in Azure Monitor](#).
You can scale-in and scale-out in Virtual Machine Scale Sets based on performance metric thresholds, by a recurring schedule, or by a particular date. You can also configure email and webhook notifications for scale actions. This walkthrough shows an example of configuring all these objects using a Resource Manager template on a VM Scale Set.

NOTE
While this walkthrough explains the steps for VM Scale Sets, the same information applies to autoscaling Cloud Services, and App Service - Web Apps. For a simple scale in/out setting on a VM Scale Set based on a simple performance metric such as CPU, refer to the Linux and Windows documents.

Walkthrough

In this walkthrough, we use Azure Resource Explorer to configure and update the autoscale setting for a scale set. Azure Resource Explorer is an easy way to manage Azure resources via Resource Manager templates. If you are new to Azure Resource Explorer tool, read this introduction.

1. Deploy a new scale set with a basic autoscale setting. This article uses the one from the Azure QuickStart Gallery, which has a Windows scale set with a basic autoscale template. Linux scale sets work the same way.

2. After the scale set is created, navigate to the scale set resource from Azure Resource Explorer. You see the following under Microsoft.Insights node.

The template execution has created a default autoscale setting with the name ‘autoscalewad’. On the right-hand side, you can view the full definition of this autoscale setting. In this case, the default autoscale setting comes with a CPU% based scale-out and scale-in rule.
3. You can now add more profiles and rules based on the schedule or specific requirements. We create an autoscale setting with three profiles. To understand profiles and rules in autoscale, review Autoscale Best Practices.

<table>
<thead>
<tr>
<th>PROFILES &amp; RULES</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile</td>
<td>Performance/metric based</td>
</tr>
<tr>
<td>Rule</td>
<td>Service Bus Queue Message Count &gt; x</td>
</tr>
<tr>
<td>Rule</td>
<td>Service Bus Queue Message Count &lt; y</td>
</tr>
<tr>
<td>Rule</td>
<td>CPU% &gt; n</td>
</tr>
<tr>
<td>Rule</td>
<td>CPU% &lt; p</td>
</tr>
<tr>
<td>Profile</td>
<td>Weekday morning hours (no rules)</td>
</tr>
<tr>
<td>Profile</td>
<td>Product Launch day (no rules)</td>
</tr>
</tbody>
</table>

4. Here is a hypothetical scaling scenario that we use for this walk-through.

- **Load based** - I’d like to scale out or in based on the load on my application hosted on my scale set.*
- **Message Queue size** - I use a Service Bus Queue for the incoming messages to my application. I use the queue’s message count and CPU% and configure a default profile to trigger a scale action if either of message count or CPU hits the threshold.*
- **Time of week and day** - I want a weekly recurring ‘time of the day’ based profile called ‘Weekday Morning Hours’. Based on historical data, I know it is better to have certain number of VM instances to handle my application’s load during this time.*
- **Special Dates** - I added a ‘Product Launch Day’ profile. I plan ahead for specific dates so my application is ready to handle the load due marketing announcements and when we put a new product in the application.*
- **The last two profiles can also have other performance metric based rules within them. In this case, I decided not to have one and instead to rely on the default performance metric based rules. Rules are optional for the recurring and date-based profiles.**

Autoscale engine’s prioritization of the profiles and rules is also captured in the autoscaling best practices article. For a list of common metrics for autoscale, refer Common metrics for Autoscale

5. Make sure you are on the **Read/Write** mode in Resource Explorer

6. Click Edit. **Replace** the 'profiles' element in autoscale setting with the following configuration:
"profiles": [
  {
    "name": "Perf_Based_Scale",
    "capacity": {
      "minimum": "2",
      "maximum": "12",
      "default": "2"
    },
    "rules": [
      {
        "metricTrigger": {
          "metricName": "MessageCount",
          "metricNamespace": "",
          "metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/Microsoft.ServiceBus/namespaces/mySB/queues/myqueue",
          "timeGrain": "PT5M",
          "statistic": "Average",
          "timeWindow": "PT5M",
          "timeAggregation": "Average",
          "operator": "GreaterThan",
          "threshold": 10
        },
        "scaleAction": {
          "direction": "Increase",
          "type": "ChangeCount",
          "value": "1",
          "cooldown": "PT5M"
        }
      },
      {
        "metricTrigger": {
          "metricName": "MessageCount",
          "metricNamespace": "",
          "metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/Microsoft.ServiceBus/namespaces/mySB/queues/myqueue",
          "timeGrain": "PT5M",
          "statistic": "Average",
          "timeWindow": "PT5M",
          "timeAggregation": "Average",
          "operator": "LessThan",
          "threshold": 3
        },
        "scaleAction": {
          "direction": "Decrease",
          "type": "ChangeCount",
          "value": "1",
          "cooldown": "PT5M"
        }
      },
      {
        "metricTrigger": {
          "metricName": "Percentage CPU",
          "metricNamespace": "",
          "metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/Microsoft.Compute/virtualMachineScaleSets/<this_vmss_name>",
          "timeGrain": "PT5M",
          "statistic": "Average",
          "timeWindow": "PT30M",
          "timeAggregation": "Average",
          "operator": "GreaterThan",
          "threshold": 85
        },
        "scaleAction": {
          "direction": "Increase",
          "type": "ChangeCount"
        }
      }
    ]
  }
]


```json
{
  "type": "ChangeCount",
  "value": "1",
  "cooldown": "PT5M"
}
{
  "metricTrigger": {
    "metricName": "Percentage CPU",
    "metricNamespace": "",
    "metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/Microsoft.Compute/virtualMachineScaleSets/<this_vmss_name>",
    "timeGrain": "PT5M",
    "statistic": "Average",
    "timeWindow": "PT30M",
    "timeAggregation": "Average",
    "operator": "LessThan",
    "threshold": 60
  },
  "scaleAction": {
    "direction": "Decrease",
    "type": "ChangeCount",
    "value": "1",
    "cooldown": "PT5M"
  }
}
{
  "name": "Weekday_Morning_Hours_Scale",
  "capacity": {
    "minimum": "4",
    "maximum": "12",
    "default": "4"
  },
  "rules": [],
  "recurrence": {
    "frequency": "Week",
    "schedule": {
      "timeZone": "Pacific Standard Time",
      "days": [
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday"
      ],
      "hours": [
        6
      ],
      "minutes": [
        0
      ]
    }
  }
}
{
  "name": "Product_Launch_Day",
  "capacity": {
    "minimum": "6",
    "maximum": "20",
    "default": "6"
  },
  "rules": [],
  "fixedDate": {
    "timeZone": "Pacific Standard Time",
    "start": "2016-06-20T00:06:00Z",
    "end": "2016-06-21T23:59:00Z"
  }
}
```
For supported fields and their values, see [Autoscale REST API documentation](#). Now your autoscale setting contains the three profiles explained previously.

7. Finally, look at the Autoscale **notification** section. Autoscale notifications allow you to do three things when a scale-out or in action is successfully triggered.

   - Notify the admin and co-admins of your subscription
   - Email a set of users
   - Trigger a webhook call. When fired, this webhook sends metadata about the autoscaling condition and the scale set resource. To learn more about the payload of autoscale webhook, see [Configure Webhook & Email Notifications for Autoscale](#).

Add the following to the Autoscale setting replacing your `notification` element whose value is null

```json
"notifications": [  
  {  
    "operation": "Scale",  
    "email": {  
      "sendToSubscriptionAdministrator": true,  
      "sendToSubscriptionCoAdministrators": false,  
      "customEmails": [  
        "user1@mycompany.com",  
        "user2@mycompany.com"  
      ]  
    },  
    "webhooks": [  
      {  
        "serviceUri": "https://foo.webhook.example.com?token=abcd1234",  
        "properties": {  
          "optional_key1": "optional_value1",  
          "optional_key2": "optional_value2"  
        }  
      }  
    ]  
  }  
]
```

Hit **Put** button in Resource Explorer to update the autoscale setting.

You have updated an autoscale setting on a VM Scale set to include multiple scale profiles and scale notifications.

**Next Steps**

Use these links to learn more about autoscaling.

- [Troubleshoot Autoscale with Virtual Machine Scale Sets](#)
- [Common Metrics for Autoscale](#)
- [Best Practices for Azure Autoscale](#)
- [Manage Autoscale using PowerShell](#)
- [Manage Autoscale using CLI](#)
- [Configure Webhook & Email Notifications for Autoscale](#)
To run applications on virtual machine (VM) instances in a scale set, you first need to install the application components and required files. This article introduces ways to build a custom VM image for instances in a scale set, or automatically run install scripts on existing VM instances. You also learn how to manage application or OS updates across a scale set.

Build a custom VM image

When you use one of the Azure platform images to create the instances in your scale set, no additional software is installed or configured. You can automate the install of these components, however that adds to the time it takes to provision VM instances to your scale sets. If you apply many configuration changes to the VM instances, there is management overhead with those configuration scripts and tasks.

To reduce the configuration management and time to provision a VM, you can create a custom VM image that is ready to run your application as soon as an instance is provisioned in the scale set. The overall process to create a custom VM image for scale set instances are as follows:

1. To build a custom VM image for your scale set instances, you create and log in to a VM, then install and configure the application. You can use Packer to define and build a Linux or Windows VM image. Or, you can manually create and configure the VM:
   - Create a Linux VM with the Azure CLI 2.0, Azure PowerShell, or the portal.
   - Create a Windows VM with the Azure PowerShell, the Azure CLI 2.0, or the portal.
   - Log in to a Linux or Windows VM.
   - Install and configure the applications and tools needed. If you need specific versions of a library or runtime, a custom VM image allows you to define a version and
2. Capture your VM with the Azure CLI 2.0 or Azure PowerShell. This step creates the custom VM image that is used to then deploy instances in a scale set.
3. Create a scale set and specify the custom VM image created in the preceding steps.

Install an app with the Custom Script Extension

The Custom Script Extension downloads and executes scripts on Azure VMs. This extension is useful for post deployment configuration, software installation, or any other configuration / management task. Scripts can be downloaded from Azure storage or GitHub, or provided to the Azure portal at extension run-time.

The Custom Script extension integrates with Azure Resource Manager templates, and can also be run using the Azure CLI, PowerShell, Azure portal, or the Azure Virtual Machine REST API.

For more information, see the Custom Script Extension overview.

Use Azure PowerShell

PowerShell uses a hashtable to store the file to download and the command to execute. The following example:

- Instructs the VM instances to download a script from GitHub -
  
  `https://raw.githubusercontent.com/iainfoulds/azure-samples/master/automate-iis.ps1`

- Sets the extension to run an install script -
  
  `powershell -ExecutionPolicy Unrestricted -File automate-iis.ps1`

- Gets information about a scale set with
  
  `Get-AzureRmVmss`
The Custom Script Extension is applied to the *myScaleSet* VM instances in the resource group named *myResourceGroup*. Enter your own names as follows:

```powershell
# Define the script for your Custom Script Extension to run
$customConfig = @(  
    "fileUris": ["https://raw.githubusercontent.com/iainfoulds/azure-samples/master/automate-iis.ps1"],  
    "commandToExecute": "powershell -ExecutionPolicy Unrestricted -File automate-iis.ps1"
)

# Get information about the scale set
$vms = Get-AzureRmVmss `  
    -ResourceGroupName "myResourceGroup" `  
    -VMScaleSetName "myScaleSet"

# Add the Custom Script Extension to install IIS and configure basic website
$vms = Add-AzureRmVmssExtension `  
    -VirtualMachineScaleSet $vms `  
    -Name "customScript" `  
    -Publisher "Microsoft.Compute" `  
    -Type "CustomScriptExtension" `  
    -TypeHandlerVersion 1.8 `  
    -Setting $customConfig

# Update the scale set and apply the Custom Script Extension to the VM instances
Update-AzureRmVmss `  
    -ResourceGroupName "myResourceGroup" `  
    -Name "myScaleSet" `  
    -VirtualMachineScaleSet $vms
```

If the upgrade policy on your scale set is *manual*, update your VM instances with `Update-AzureRmVmssInstance`. This cmdlet applies the updated scale set configuration to the VM instances and installs your application.

**Use Azure CLI 2.0**

To use the Custom Script Extension with the Azure CLI, you create a JSON file that defines what files to obtain and commands to execute. These JSON definitions can be reused across scale set deployments to apply consistent application installs.

In your current shell, create a file named `customConfig.json` and paste the following configuration. For example, create the file in the Cloud Shell not on your local machine. You can use any editor you wish. Enter `sensible-editor cloudConfig.json` to create the file and see a list of available editors.

```json
{
    "fileUris": ["https://raw.githubusercontent.com/iainfoulds/azure-samples/master/automate_nginx.sh"],
    "commandToExecute": ".\automate_nginx.sh"
}
```

Apply the Custom Script Extension configuration to the VM instances in your scale set with `az vmss extension set`. The following example applies the *customConfig.json* configuration to the *myScaleSet* VM instances in the resource group named *myResourceGroup*. Enter your own names as follows:

```bash
az vmss extension set \  
    --publisher Microsoft.Azure.Extensions \  
    --version 2.0 \  
    --name CustomScript \  
    --resource-group myResourceGroup \  
    --vms-name myScaleSet \  
    --settings @customConfig.json
```
If the upgrade policy on your scale set is *manual*, update your VM instances with `az vmss update-instances`. This cmdlet applies the updated scale set configuration to the VM instances and installs your application.

### Install an app to a Windows VM with PowerShell DSC

**PowerShell Desired State Configuration (DSC)** is a management platform to define the configuration of target machines. DSC configurations define what to install on a machine and how to configure the host. A Local Configuration Manager (LCM) engine runs on each target node that processes requested actions based on pushed configurations.

The PowerShell DSC extension lets you customize VM instances in a scale set with PowerShell. The following example:

- Instructs the VM instances to download a DSC package from GitHub - [https://github.com/iainfoulds/azure-samples/raw/master/dsc.zip](https://github.com/iainfoulds/azure-samples/raw/master/dsc.zip)
- Sets the extension to run an install script - `configure-http.ps1`
- Gets information about a scale set with `Get-AzureRmVmss`
- Applies the extension to the VM instances with `Update-AzureRmVmss`

The DSC extension is applied to the *myScaleSet* VM instances in the resource group named *myResourceGroup*. Enter your own names as follows:

```powershell
# Define the script for your Desired Configuration to download and run
$dscConfig = @{
    "wmfVersion" = "latest";
    "configuration" = @{
        "url" = "https://github.com/iainfoulds/azure-samples/raw/master/dsc.zip";
        "script" = "configure-http.ps1";
        "function" = "WebsiteTest";
    }
};

# Get information about the scale set
$vmss = Get-AzureRmVmss ` -ResourceGroupName "myResourceGroup" ` -VMScaleSetName "myScaleSet"

# Add the Desired State Configuration extension to install IIS and configure basic website
$vmss = Add-AzureRmVmssExtension ` -VirtualMachineScaleSet $vmss ` -Publisher Microsoft.Powershell ` -Type DSC ` -TypeHandlerVersion 2.24 ` -Name "DSC" ` -Setting $dscConfig

# Update the scale set and apply the Desired State Configuration extension to the VM instances
Update-AzureRmVmss ` -ResourceGroupName "myResourceGroup" ` -Name "myScaleSet" ` -VirtualMachineScaleSet $vmss
```

If the upgrade policy on your scale set is *manual*, update your VM instances with `Update-AzureRmVmssInstance`. This cmdlet applies the updated scale set configuration to the VM instances and installs your application.

### Install an app to a Linux VM with cloud-init

**Cloud-init** is a widely used approach to customize a Linux VM as it boots for the first time. You can use cloud-init to install packages and write files, or to configure users and security. As cloud-init runs during the initial boot process, there are no additional steps or required agents to apply your configuration.
Cloud-init also works across distributions. For example, you don’t use `apt-get install` or `yum install` to install a package. Instead you can define a list of packages to install. Cloud-init automatically uses the native package management tool for the distro you select.

For more information, including an example `cloud-init.txt` file, see Use cloud-init to customize Azure VMs.

To create a scale set and use a cloud-init file, add the `--custom-data` parameter to the `az vmss create` command and specify the name of a cloud-init file. The following example creates a scale set named `myScaleSet` in `myResourceGroup` and configures VM instances with a file named `cloud-init.txt`. Enter your own names as follows:

```
az vmss create 
  --resource-group myResourceGroup 
  --name myScaleSet 
  --image UbuntuLTS 
  --upgrade-policy-mode automatic 
  --custom-data cloud-init.txt 
  --admin-username azureuser 
  --generate-ssh-keys
```

Install applications as a set scales out

Scale sets allow you to increase the number of VM instances that run your application. This scale out process can be started manually, or automatically based on metrics such as CPU or memory usage.

If you applied a Custom Script Extension to the scale set, the application is installed to each new VM instance. If the scale set is based on a custom image with the application pre-installed, each new VM instance is deployed in a usable state.

If the scale set VM instances are container hosts, you can use the Custom Script Extension to pull and run the need container images. The Custom Script extension could also register the new VM instance with an orchestrator, such as Azure Container Service.

Deploy application updates

If you update your application code, libraries, or packages, you can push the latest application state to VM instances in a scale set. If you use the Custom Script Extension, updates to your application and not automatically deployed. Change the Custom Script configuration, such as to point to an install script that has an updated version name. In a previous example, the Custom Script Extension uses a script named `automate_nginx.sh` as follows:

```
{
  "fileUris": ["https://raw.githubusercontent.com/iainfoulds/azure-samples/master/automate_nginx.sh"],
  "commandToExecute": "./automate_nginx.sh"
}
```

Any updates you make to your application are not exposed to the Custom Script Extension unless that install script changes. One approach is to include a version number that increments with your application releases. The Custom Script extension could now reference `automate_nginx_v2.sh` as follows:

```
{
  "fileUris": ["https://raw.githubusercontent.com/iainfoulds/azure-samples/master/automate_nginx_v2.sh"],
  "commandToExecute": "./automate_nginx_v2.sh"
}
```

The Custom Script Extension now runs against the VM instances to apply the latest application updates.
**Install applications with OS updates**

When new OS releases are available, you can use or build a new custom image and deploy OS upgrades to a scale set. Each VM instance is upgraded to the latest image that you specify. You can use a custom image with the application pre-installed, the Custom Script Extension, or PowerShell DSC to have your application automatically available as you perform the upgrade. You may need to plan for application maintenance as you perform this process to ensure that there are no version compatibility issues.

If you use a custom VM image with the application pre-installed, you could integrate the application updates with a deployment pipeline to build the new images and deploy OS upgrades across the scale set. This approach allows the pipeline to pick up the latest application builds, create and validate a VM image, then upgrade the VM instances in the scale set. To run a deployment pipeline that builds and deploys application updates across custom VM images, you could use Visual Studio Team Services, Spinnaker, or Jenkins.

**Next steps**

As you build and deploy applications to your scale sets, you can review the Scale Set Design Overview. For more information on how to manage your scale set, see Use PowerShell to manage your scale set.
Azure virtual machine scale sets now support virtual machines with attached data disks. Data disks can be defined in the storage profile for scale sets that have been created with Azure Managed Disks. Previously the only directly attached storage options available with VMs in scale sets were the OS drive and temp drives.

**NOTE**

When you create a scale set with attached data disks defined, you still need to mount and format the disks from within a VM to use them (just like for standalone Azure VMs). A convenient way to do this is to use a custom script extension which calls a standard script to partition and format all the data disks on a VM.

## Create a scale set with attached data disks

A simple way to create a scale set with attached disks is to use the Azure CLI `vmss create` command. The following example creates an Azure resource group, and a VM scale set of 10 Ubuntu VMs, each with 2 attached data disks, of 50 GB and 100 GB respectively.

```
az group create -l southcentralus -n dsktest
az vmss create -g dsktest -n dskvmss --image ubuntults --instance-count 10 --data-disk-sizes-gb 50 100
```

Note that the `vmss create` command defaults certain configuration values if you do not specify them. To see the available options that you can override try:

```
az vmss create --help
```

Another way to create a scale set with attached data disks is to define a scale set in an Azure Resource Manager template, include a `dataDisks` section in the `storageProfile`, and deploy the template. The 50 GB and 100 GB disk example above would be defined like this in the template:

```
"dataDisks": [
  {
    "lun": 1,
    "createOption": "Empty",
    "caching": "ReadOnly",
    "diskSizeGB": 50
  },
  {
    "lun": 2,
    "createOption": "Empty",
    "caching": "ReadOnly",
    "diskSizeGB": 100
  }
]
```

You can see a complete, ready to deploy example of a scale set template with an attached disk defined here: https://github.com/chagarw/MDPP/tree/master/101-vmss-os-data.

## Adding a data disk to an existing scale set
You can only attach data disks to a scale set which has been created with Azure Managed Disks.

You can add a data disk to a VM scale set using Azure CLI `az vmss disk attach` command. Make sure you specify a lun which is not already in use. The following CLI example adds a 50 GB drive to lun 3:

```
$vmss = Get-AzureRmVmss -ResourceGroupName myvmssrg -VMScaleSetName myvmss
$vmss = Add-AzureRmVmssDataDisk -VirtualMachineScaleSet $vmss -Lun 3 -Caching 'ReadWrite' -CreateOption Empty -DiskSizeGB 50 -StorageAccountType StandardLRS
Update-AzureRmVmss -ResourceGroupName myvmssrg -Name myvmss -VirtualMachineScaleSet $vmss
```

The following PowerShell example adds a 50 GB drive to lun 3:

```
$vms = Get-AzureRmVmss -ResourceGroupName myvmssrg -VMScaleSetName myvmss
$vms = Add-AzureRmVmssDataDisk -VirtualMachineScaleSet $vms -Lun 3 -Caching 'ReadWrite' -CreateOption Empty -DiskSizeGB 50 -StorageAccountType StandardLRS
Update-AzureRmVmss -ResourceGroupName myvmssrg -Name myvmss -VirtualMachineScaleSet $vms
```

Different VM sizes have different limits on the numbers of attached drives they support. Check the virtual machine size characteristics before adding a new disk.

You can also add a disk by adding a new entry to the `dataDisks` property in the `storageProfile` of a scale set definition and applying the change. To test this, find an existing scale set definition in the Azure Resource Explorer. Select Edit and add a new disk to the list of data disks. E.g. using the example above:

```
"dataDisks": [
  {
    "lun": 1,
    "createOption": "Empty",
    "caching": "ReadOnly",
    "diskSizeGB": 50
  },
  {
    "lun": 2,
    "createOption": "Empty",
    "caching": "ReadOnly",
    "diskSizeGB": 100
  },
  {
    "lun": 3,
    "createOption": "Empty",
    "caching": "ReadOnly",
    "diskSizeGB": 20
  }
]
```

Then select PUT to apply the changes to your scale set. This example would work as long as you are using a VM size which supports more than two attached data disks.

When you make a change to a scale set definition such as adding or removing a data disk, it applies to all newly created VMs, but only applies to existing VMs if the `upgradePolicy` property is set to “Automatic”. If it is set to “Manual”, you need to manually apply the new model to existing VMs. You can do this in the portal, using the `Update-AzureRmVmssInstance` PowerShell command, or using the `az vmss update-instances` CLI command.
Adding pre-populated data disks to an existent scale set

When you add disks to an existent scale set model, by design, the disk will always be created empty. This scenario also includes new instances created by the scale set. This behaviour is because the scaleset definition has an empty data disk. In order to create pre-populated data drives for an existent scale set model, you can choose either of next two options:

- Copy data from the instance 0 VM to the data disk(s) in the other VMs by running a custom script.
- Create a managed image with the OS disk plus data disk (with the required data) and create a new scale set with the image. This way every new VM created will have a data disk that is provided in the definition of the scale set. Since this definition will refer to an image with a data disk that has customized data, every virtual machine on the scale set will automatically come up with these changes.

The way to create a custom image can be found here: Create a managed image of a generalized VM in Azure

The user needs to capture the instance 0 VM which has the required data, and then use that vhd for the image definition.

Removing a data disk from a scale set

You can remove a data disk from a VM scale set using Azure CLI `az vmss disk detach` command. For example the following command removes the disk defined at lun 2:

```
az vmss disk detach -g dsktest -n dskvmss --lun 2
```

Similarly you can also remove a disk from a scale set by removing an entry from the `dataDisks` property in the `storageProfile` and applying the change.

Additional notes

Support for Azure Managed disks and scale set attached data disks is available in API version 2016-04-30-preview or later of the Microsoft.Compute API.

In the initial implementation of attached disk support for scale sets, you cannot attach or detach data disks to/from individual VMs in a scale set.

Azure portal support for attached data disks in scale sets is initially limited. Depending on your requirements you can use Azure templates, CLI, PowerShell, SDKs, and REST API to manage attached disks.
You can now create Azure virtual machine scale sets with a capacity of up to 1,000 VMs. In this document, a large virtual machine scale set is defined as a scale set capable of scaling to greater than 100 VMs. This capability is set by a scale set property (singlePlacementGroup=False).

Certain aspects of large scale sets, such as load balancing and fault domains behave differently to a standard scale set. This document explains the characteristics of large scale sets, and describes what you need to know to successfully use them in your applications.

A common approach for deploying cloud infrastructure at large scale is to create a set of scale units, for example by creating multiple VMs scale sets across multiple VNETs and storage accounts. This approach provides easier management compared to single VMs, and multiple scale units are useful for many applications, particularly those that require other stackable components like multiple virtual networks and endpoints. If your application requires a single large cluster however, it can be more straightforward to deploy a single scale set of up to 1,000 VMs. Example scenarios include centralized big data deployments, or compute grids requiring simple management of a large pool of worker nodes. Combined with VM scale set attached data disks, large scale sets enable you to deploy a scalable infrastructure consisting of thousands of vCPUs and petabytes of storage, as a single operation.

Placement groups

What makes a large scale set special is not the number of VMs, but the number of placement groups it contains. A placement group is a construct similar to an Azure availability set, with its own fault domains and upgrade domains. By default, a scale set consists of a single placement group with a maximum size of 100 VMs. If a scale set property called singlePlacementGroup is set to false, the scale set can be composed of multiple placement groups and has a range of 0-1,000 VMs. When set to the default value of true, a scale set is composed of a single placement group, and has a range of 0-100 VMs.

Checklist for using large scale sets

To decide whether your application can make effective use of large scale sets, consider the following requirements:

- Large scale sets require Azure Managed Disks. Scale sets that are not created with Managed Disks require multiple storage accounts (one for every 20 VMs). Large scale sets are designed to work exclusively with Managed Disks to reduce your storage management overhead, and to avoid the risk of running into subscription limits for storage accounts. If you do not use Managed Disks, your scale set is limited to 100 VMs.
- Scale sets created from Azure Marketplace images can scale up to 1,000 VMs.
- Scale sets created from custom images (VM images you create and upload yourself) can currently scale up to 300 VMs.
- Layer-4 load balancing with scale sets composed of multiple placement groups requires Azure Load Balancer Standard SKU. The Load Balancer Standard SKU provides additional benefits, such as the ability to load balance between multiple scale sets. Standard SKU also requires that the scale set has a Network Security Group associated with it, otherwise NAT pools will not work correctly. If you need to use the Azure Load Balancer Basic SKU, make sure the scale set is configured to use a single placement group, which is the default setting.
- Layer-7 load balancing with the Azure Application Gateway is supported for all scale sets.
- A scale set is defined with a single subnet - make sure your subnet has an address space large enough for all the VMs you need. By default a scale set overprovisions (creates extra VMs at deployment time or when scaling out, which you are not charged for) to improve deployment reliability and performance. Allow for an address space 20% greater than the number of VMs you plan to scale to.
If you are planning to deploy many VMs, your Compute vCPU quota limits may need to be increased.

Fault domains and upgrade domains are only consistent within a placement group. This architecture does not change the overall availability of a scale set, as VMs are evenly distributed across distinct physical hardware, but it does mean that if you need to guarantee two VMs are on different hardware, make sure they are in different fault domains in the same placement group. Fault domain and placement group ID are shown in the instance view of a scale set VM. You can view the instance view of a scale set VM in the Azure Resource Explorer.

Creating a large scale set

When you create a scale set in the Azure portal, you can allow it to scale to multiple placement groups by setting the Limit to a single placement group option to False in the Basics blade. With this option set to False, you can specify an Instance count value of up to 1,000.

You can create a large VM scale set using the Azure CLI az vmss create command. This command sets intelligent defaults such as subnet size based on the instance-count argument:

```
az group create -l southcentralus -n biginfra
az vmss create -g biginfra -n bigvmss --image ubuntults --instance-count 1000
```

Note that the vmss create command defaults certain configuration values if you do not specify them. To see the available options that you can override, try:

```
az vmss create --help
```

If you are creating a large scale set by composing an Azure Resource Manager template, make sure the template creates a scale set based on Azure Managed Disks. You can set the singlePlacementGroup property to false in the properties section of the Microsoft.Compute/virtualMachineScaleSets resource. The following JSON fragment shows the beginning of a scale set template, including the 1,000 VM capacity and the "singlePlacementGroup": false setting:
For a complete example of a large scale set template, refer to https://github.com/gbowerman/azure-myriad/blob/master/bigtest/bigbottle.json.

Converting an existing scale set to span multiple placement groups

To make an existing VM scale set capable of scaling to more than 100 VMs, you need to change the `singlePlacementGroup` property to `false` in the scale set model. You can test changing this property with the Azure Resource Explorer. Find an existing scale set, select Edit and change the `singlePlacementGroup` property. If you do not see this property, you may be viewing the scale set with an older version of the Microsoft.Compute API.

**NOTE**

You can change a scale set from supporting a single placement group only (the default behavior) to a supporting multiple placement groups, but you cannot convert the other way around. Therefore make sure you understand the properties of large scale sets before converting.
Customers with a Resource Manager template for creating a scale set not using managed disk may wish to modify it to use managed disk. This article shows how to do this, using as an example a pull request from the Azure Quickstart Templates, a community-driven repo for sample Resource Manager templates. The full pull request can be seen here: https://github.com/Azure/azure-quickstart-templates/pull/2998, and the relevant parts of the diff are below, along with explanations:

Making the OS disks managed

In the diff below, we can see that we have removed several variables related to storage account and disk properties. Storage account type is no longer necessary (Standard_LRS is the default), but we could still specify it if we wished to. Only Standard_LRS and Premium_LRS are supported with managed disk. New storage account suffix, unique string array, and sa count were used in the old template to generate storage account names. These variables are no longer necessary in the new template because managed disk automatically creates storage accounts on the customer’s behalf. Similarly, vhd container name and os disk name are no longer necessary because managed disk automatically names the underlying storage blob containers and disks.

```
"variables": {
  "storageAccountType": "Standard_LRS",
  "namingInfix": "[tolower(substring(concat(parameters('vmssName'), uniqueString(resourceGroup().id)), 0, 9))]",
  "longNamingInfix": "[tolower(parameters('vmssName'))]",
  "newStorageAccountSuffix": "[concat(variables('namingInfix'), 'sa')]",
  "uniqueStringArray": [
    "[concat(uniqueString(concat(resourceGroup().id, variables('newStorageAccountSuffix'), '0')))]",
    "[concat(uniqueString(concat(resourceGroup().id, variables('newStorageAccountSuffix'), '1')))]",
    "[concat(uniqueString(concat(resourceGroup().id, variables('newStorageAccountSuffix'), '2')))]",
    "[concat(uniqueString(concat(resourceGroup().id, variables('newStorageAccountSuffix'), '3')))]",
    "[concat(uniqueString(concat(resourceGroup().id, variables('newStorageAccountSuffix'), '4')))]"
  ],
  "saCount": "[length(variables('uniqueStringArray'))]",
  "vhdContainerName": "[concat(variables('namingInfix'), 'vhd')]",
  "osDiskName": "[concat(variables('namingInfix'), 'osdisk')]",
  "addressPrefix": "10.0.0.0/16",
  "subnetPrefix": "10.0.0.0/24",
  "virtualNetworkName": "[concat(variables('namingInfix'), 'vnet')]"
}
```

In the diff below, we can see that we updated the compute api version to 2016-04-30-preview, which is the earliest required version for managed disk support with scale sets. Note that we could still use unmanaged disks in the new api version with the old syntax if desired. In other words, if we only update the compute api version and don’t change anything else, the template should continue to work as before.
In the diff below, we can see that we are removing the storage account resource from the resources array completely. We no longer need them since managed disk creates them automatically on our behalf.

In the diff below, we can see that we are removing the depends on clause referring from the scale set to the loop that was creating storage accounts. In the old template, this was ensuring that the storage accounts were created before the scale set began creation, but this clause is no longer necessary with managed disk. We also remove the vhd containers property, and the os disk name property as these properties are automatically handled under the hood by managed disk. If we wished, we could add `{ "managedDisk": { "storageAccountType": "Premium_LRS" } }` in the "osDisk" configuration if we wanted premium OS disks. Only VMs with an uppercase or lowercase 's' in the VM sku can use premium disks.
Data disks

With the changes above, the scale set uses managed disks for the OS disk, but what about data disks? To add data disks, add the “dataDisks” property under “storageProfile” at the same level as “osDisk”. The value of the property is a JSON list of objects, each of which has properties “lun” (which must be unique per data disk on a VM), “createOption” (“empty” is currently the only supported option), and “diskSizeGB” (the size of the disk in gigabytes; must be greater than 0 and less than 1024) as in the following example:

```json
"dataDisks": [
    {
        "lun": "1",
        "createOption": "empty",
        "diskSizeGB": "1023"
    }
]
```

If you specify \( n \) disks in this array, each VM in the scale set gets \( n \) data disks. Do note, however, that these data disks are raw devices. They are not formatted. It is up to the customer to attach, partition, and format the disks before using them. Optionally, we could also specify "managedDisk": { "storageAccountType": "Premium_LRS" } in each data disk object to specify that it should be a premium data disk. Only VMs with an uppercase or lowercase ‘s’ in the VM sku can use premium disks.

To learn more about using data disks with scale sets, see this article.

Next steps
For example Resource Manager templates using scale sets, search for “vmss” in the Azure Quickstart Templates github repo.

For general information, check out the main landing page for scale sets.
Throughout the lifecycle of a virtual machine scale set, you may need to run one or more management tasks. Additionally, you may want to create scripts that automate various lifecycle-tasks. This article details some of the common Azure CLI 2.0 commands that let you perform these tasks.

To complete these management tasks, you need the latest Azure CLI 2.0 build. For information on how to install and use the latest version, see Install the Azure CLI 2.0. If you need to create a virtual machine scale set, you can create a scale set in the Azure portal.

View information about a scale set

To view the overall information about a scale set, use `az vmss show`. The following example gets information about the scale set named `myScaleSet` in the `myResourceGroup` resource group. Enter your own names as follows:

```
az vmss show --resource-group myResourceGroup --name myScaleSet
```

View VMs in a scale set

To view a list of VM instance in a scale set, use `az vmss list-instances`. The following example list all VM instances in the scale set named `myScaleSet` in the `myResourceGroup` resource group. Provide your own values for these names:

```
az vmss list-instances \ 
  --resource-group myResourceGroup \ 
  --name myScaleSet \ 
  --output table
```

To view additional information about a specific VM instance, add the `--instance-id` parameter to `az vmss get-instance-view` and specify an instance to view. The following example views information about VM instance `0` in the scale set named `myScaleSet` and the `myResourceGroup` resource group. Enter your own names as follows:

```
az vmss get-instance-view \ 
  --resource-group myResourceGroup \ 
  --name myScaleSet \ 
  --instance-id 0
```

List connection information for VMs

To connect to the VMs in a scale set, you SSH or RDP to an assigned public IP address and port number. By default, Network Address Translation (NAT) rules are added to the Azure load balancer that forwards remote connection traffic to each VM. To list the address and ports to connect to VM instances in a scale set, use `az vmss list-instance-connection-info`. The following example list connection information for VM instances in the scale set named `myScaleSet` and in the `myResourceGroup` resource group. Provide your own values for these names:
Change the capacity of a scale set

The preceding commands showed information about your scale set and the VM instances. To increase or decrease the number of instances in the scale set, you can change the capacity. The scale set creates or removes the required number of VMs, then configures the VMs to receive application traffic.

To see the number of instances you currently have in a scale set, use `az vmss show` and query on `sku.capacity`:

```
az vmss show \
--resource-group myResourceGroup \
--name myScaleSet \
--query [sku.capacity] \
--output table
```

You can then manually increase or decrease the number of virtual machines in the scale set with `az vmss scale`. The following example sets the number of VMs in your scale set to 5:

```
az vmss scale \
--resource-group myResourceGroup \
--name myScaleSet \
--new-capacity 5
```

If takes a few minutes to update the capacity of your scale set. If you decrease the capacity of a scale set, the VMs with the highest instance IDs are removed first.

Stop and start VMs in a scale set

To stop one or more VMs in a scale set, use `az vmss stop`. The `--instance-ids` parameter allows you to specify one or more VMs to stop. If you do not specify an instance ID, all VMs in the scale set are stopped. To stop multiple VMs, separate each instance ID with a space.

The following example stops instance 0 in the scale set named `myScaleSet` and the `myResourceGroup` resource group. Provide your values as follows:

```
az vmss stop --resource-group myResourceGroup --name myScaleSet --instance-ids 0
```

Stopped VMs remain allocated and continue to incur compute charges. If you instead wish the VMs to be deallocated and only incur storage charges, use `az vmss deallocate`. To deallocate multiple VMs, separate each instance ID with a space. The following example stops and deallocates instance 0 in the scale set named `myScaleSet` and the `myResourceGroup` resource group. Provide your values as follows:

```
az vmss deallocate --resource-group myResourceGroup --name myScaleSet --instance-ids 0
```

Start VMs in a scale set

To start one or more VMs in a scale set, use `az vmss start`. The `--instance-ids` parameter allows you to specify one or more VMs to start. If you do not specify an instance ID, all VMs in the scale set are started. To start multiple VMs, separate each instance ID with a space.
The following example starts instance 0 in the scale set named *myScaleSet* and the *myResourceGroup* resource group. Provide your values as follows:

```
az vmss start --resource-group myResourceGroup --name myScaleSet --instance-ids 0
```

**Restart VMs in a scale set**

To restart one or more VMs in a scale set, use `az vmss restart`. The `--instance-ids` parameter allows you to specify one or more VMs to restart. If you do not specify an instance ID, all VMs in the scale set are restarted. To restart multiple VMs, separate each instance ID with a space.

The following example restarts instance 0 in the scale set named *myScaleSet* and the *myResourceGroup* resource group. Provide your values as follows:

```
az vmss restart --resource-group myResourceGroup --name myScaleSet --instance-ids 0
```

**Remove VMs from a scale set**

To remove one or more VMs in a scale set, use `az vmss delete-instances`. The `--instance-ids` parameter allows you to specify one or more VMs to remove. If you specify * for the instance ID, all VMs in the scale set are removed. To remove multiple VMs, separate each instance ID with a space.

The following example removes instance 0 in the scale set named *myScaleSet* and the *myResourceGroup* resource group. Provide your values as follows:

```
az vmss delete-instances --resource-group myResourceGroup --name myScaleSet --instance-ids 0
```

**Next steps**

Other common tasks for scale sets include how to [deploy an application](#), and [upgrade VM instances](#). You can also use Azure CLI to [configure auto-scale rules](#).
Manage a virtual machine scale set with Azure PowerShell

Throughout the lifecycle of a virtual machine scale set, you may need to run one or more management tasks. Additionally, you may want to create scripts that automate various lifecycle-tasks. This article details some of the common Azure PowerShell cmdlets that let you perform these tasks.

To complete these management tasks, you need the latest Azure PowerShell module. For information on how to install and use the latest version, see Getting started with Azure PowerShell. If you need to create a virtual machine scale set, you can create a scale set in the Azure portal.

### View information about a scale set

To view the overall information about a scale set, use `Get-AzureRmVmss`. The following example gets information about the scale set named `myScaleSet` in the `myResourceGroup` resource group. Enter your own names as follows:

```powershell
Get-AzureRmVmss -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet"
```

### View VMs in a scale set

To view a list of VM instance in a scale set, use `Get-AzureRmVmssVM`. The following example list all VM instances in the scale set named `myScaleSet` and in the `myResourceGroup` resource group. Provide your own values for these names:

```powershell
Get-AzureRmVmssVM -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet"
```

To view additional information about a specific VM instance, add the `-InstanceId` parameter to `Get-AzureRmVmssVM` and specify an instance to view. The following example views information about VM instance 0 in the scale set named `myScaleSet` and the `myResourceGroup` resource group. Enter your own names as follows:

```powershell
Get-AzureRmVmssVM -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet" -InstanceId "0"
```

### Change the capacity of a scale set

The preceding commands showed information about your scale set and the VM instances. To increase or decrease the number of instances in the scale set, you can change the capacity. The scale set automatically creates or removes the required number of VMs, then configures the VMs to receive application traffic.

First, create a scale set object with `Get-AzureRmVmss`, then specify a new value for `sku.capacity`. To apply the capacity change, use `Update-AzureRmVmss`. The following example updates `myScaleSet` in the `myResourceGroup` resource group to a capacity of 5 instances. Provide your own values as follows:
# Get current scale set

```powershell
$vmss = Get-AzureRmVmss -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet"
```

# Set and update the capacity of your scale set

```powershell
$vmss.sku.capacity = 5
Update-AzureRmVmss -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet" -VirtualMachineScaleSet $vmss
```

If takes a few minutes to update the capacity of your scale set. If you decrease the capacity of a scale set, the VMs with the highest instance IDs are removed first.

## Stop and start VMs in a scale set

To stop one or more VMs in a scale set, use `Stop-AzureRmVmss`. The `-InstanceId` parameter allows you to specify one or more VMs to stop. If you do not specify an instance ID, all VMs in the scale set are stopped. To stop multiple VMs, separate each instance ID with a comma.

The following example stops instance 0 in the scale set named `myScaleSet` and the `myResourceGroup` resource group. Provide your values as follows:

```powershell
Stop-AzureRmVmss -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet" -InstanceId "0"
```

By default, stopped VMs are deallocated and do not incur compute charges. If you wish the VM to remain in a provisioned state when stopped, add the `-StayProvisioned` parameter to the preceding command. Stopped VMs that remain provisioned incur regular compute charges.

## Start VMs in a scale set

To start one or more VMs in a scale set, use `Start-AzureRmVmss`. The `-InstanceId` parameter allows you to specify one or more VMs to start. If you do not specify an instance ID, all VMs in the scale set are started. To start multiple VMs, separate each instance ID with a comma.

The following example starts instance 0 in the scale set named `myScaleSet` and the `myResourceGroup` resource group. Provide your values as follows:

```powershell
Start-AzureRmVmss -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet" -InstanceId "0"
```

## Restart VMs in a scale set

To restart one or more VMs in a scale set, use `Restart-AzureRmVmss`. The `-InstanceId` parameter allows you to specify one or more VMs to restart. If you do not specify an instance ID, all VMs in the scale set are restarted. To restart multiple VMs, separate each instance ID with a comma.

The following example restarts instance 0 in the scale set named `myScaleSet` and the `myResourceGroup` resource group. Provide your values as follows:

```powershell
Restart-AzureRmVmss -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet" -InstanceId "0"
```

## Remove VMs from a scale set

To remove one or more VMs in a scale set, use `Remove-AzureRmVmss`. The `-InstanceId` parameter allows you to specify one or more VMs to remove. If you do not specify an instance ID, all VMs in the scale set are removed. To remove multiple VMs, separate each instance ID with a comma.
The following example removes instance 0 in the scale set named *myScaleSet* and the *myResourceGroup* resource group. Provide your values as follows:

```
Remove-AzureRmVmss -ResourceGroupName "myResourceGroup" -VMScaleSetName "myScaleSet" -InstanceId "0"
```

**Next steps**

Other common tasks for scale sets include how to [deploy an application](#), and [upgrade VM instances](#). You can also use Azure PowerShell to [configure auto-scale rules](#).
This article describes how to vertically scale Azure Virtual Machine Scale Sets with or without reprovisioning. For vertical scaling of VMs which are not in scale sets, refer to Vertically scale Azure virtual machine with Azure Automation.

Vertical scaling, also known as scale up and scale down, means increasing or decreasing virtual machine (VM) sizes in response to a workload. Compare this with horizontal scaling, also referred to as scale out and scale in, where the number of VMs is altered depending on the workload.

Reprovisioning means removing an existing VM and replacing it with a new one. When you increase or decrease the size of VMs in a VM Scale Set, in some cases you want to resize existing VMs and retain your data, while in other cases you need to deploy new VMs of the new size. This document covers both cases.

Vertical scaling can be useful when:

- A service built on virtual machines is under-utilized (for example at weekends). Reducing the VM size can reduce monthly costs.
- Increasing VM size to cope with larger demand without creating additional VMs.

You can set up vertical scaling to be triggered based on metric based alerts from your VM Scale Set. When the alert is activated it fires a webhook that triggers a runbook which can scale your scale set up or down. Vertical scaling can be configured by following these steps:

1. Create an Azure Automation account with run-as capability.
2. Import Azure Automation Vertical Scale runbooks for VM Scale Sets into your subscription.
3. Add a webhook to your runbook.
4. Add an alert to your VM Scale Set using a webhook notification.

NOTE

Vertical autoscaling can only take place within certain ranges of VM sizes. Compare the specifications of each size before deciding to scale from one to another (higher number does not always indicate bigger VM size). You can choose to scale between the following pairs of sizes:

<table>
<thead>
<tr>
<th>VM SIZES SCALING PAIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard_A0</td>
</tr>
<tr>
<td>Standard_D1</td>
</tr>
<tr>
<td>Standard_DS1</td>
</tr>
<tr>
<td>Standard_D1v2</td>
</tr>
<tr>
<td>Standard_G1</td>
</tr>
<tr>
<td>Standard_GS1</td>
</tr>
</tbody>
</table>

Create an Azure Automation Account with run-as capability
The first thing you need to do is create an Azure Automation account that will host the runbooks used to scale the VM Scale Set instances. Recently Azure Automation introduced the "Run As account" feature which makes setting up the Service Principal for automatically running the runbooks on a user's behalf very easy. You can read more about this in the article below:

- Authenticate Runbooks with Azure Run As account

**Import Azure Automation Vertical Scale runbooks into your subscription**

The runbooks needed to vertically scale your VM Scale Sets are already published in the Azure Automation Runbook Gallery. To import them into your subscription follow the steps in this article:

- Runbook and module galleries for Azure Automation

Choose the Browse Gallery option from the Runbooks menu:

![Runbooks Gallery](image)

The runbooks that need to be imported are shown. Select the runbook based on whether you want vertical scaling with or without reprovisioning:
Add a webhook to your runbook

Once you’ve imported the runbooks you’ll need to add a webhook to the runbook so it can be triggered by an alert from a VM Scale Set. The details of creating a webhook for your Runbook are described in this article:

- Azure Automation webhooks

**NOTE**

Make sure you copy the webhook URI before closing the webhook dialog as you will need this in the next section.

Add an alert to your VM Scale Set

Below is a PowerShell script which shows how to add an alert to a VM Scale Set. Refer to the following article to get the name of the metric to fire the alert on: Azure Monitor autoscaling common metrics.
$actionEmail = New-AzureRmAlertRuleEmail -CustomEmail user@contoso.com
$actionWebhook = New-AzureRmAlertRuleWebhook -ServiceUri <uri-of-the-webhook>
$threshold = <value-of-the-threshold>
$rg = <resource-group-name>
$id = <resource-id-to-add-the-alert-to>
$location = <location-of-the-resource>
$alertName = <name-of-the-resource>
$metricName = <metric-to-fire-the-alert-on>
$timeWindow = <time-window-in-hh:mm:ss-format>
$condition = <condition-for-the-threshold> # Other valid values are LessThanOrEqual, GreaterThan, GreaterThanOrEqual
$description = <description-for-the-alert>

Add-AzureRmMetricAlertRule  -Name  $alertName `  
  -Location  $location `  
  -ResourceGroup $rg `  
  -TargetResourceId $id `  
  -MetricName $metricName `  
  -Operator $condition `  
  -Threshold $threshold `  
  -WindowSize $timeWindow `  
  -TimeAggregationOperator Average `  
  -Actions $actionEmail, $actionWebhook `  
  -Description $description

**NOTE**

It is recommended to configure a reasonable time window for the alert in order to avoid triggering vertical scaling, and any associated service interruption, too often. Consider a window of at least 20-30 minutes or more. Consider horizontal scaling if you need to avoid any interruption.

For more information on how to create alerts refer to the following articles:

- Azure Monitor PowerShell quick start samples
- Azure Monitor Cross-platform CLI quick start samples

**Summary**

This article showed simple vertical scaling examples. With these building blocks - Automation account, runbooks, webhooks, alerts - you can connect a rich variety of events with a customized set of actions.
Automatic OS image upgrade is a preview feature for Azure virtual machine scale sets that automatically upgrades all VMs to the latest OS image.

Automatic OS upgrade has the following characteristics:

- Once configured, the latest OS image published by image publishers is automatically applied to the scale set without user intervention.
- Upgrades batches of instances in a rolling manner each time a new platform image is published by the publisher.
- Integrates with application health probe (optional, but highly recommended for safety).
- Works for all VM sizes.
- Works for Windows and Linux platform images.
- You can opt out of automatic upgrades at any time (OS Upgrades can be initiated manually as well).
- The OS Disk of a VM is replaced with the new OS Disk created with latest image version. Configured extensions and custom data scripts are run, while persisted data disks are retained.

Preview notes

While in preview, the following limitations and restrictions apply:

- Automatic OS upgrades only support three OS SKUs. There is no SLA or guarantees. We recommend you do not use automatic upgrades on production critical workloads during preview.
- Support for scale sets in Service Fabric clusters is coming soon.
- Azure disk encryption (currently in preview) is not currently supported with virtual machine scale set automatic OS upgrade.
- Portal experience coming soon.

Register to use Automatic OS Upgrade

To use the automated OS upgrade feature, register the preview provider with `Register-AzureRmProviderFeature` as follows:

```powershell
Register-AzureRmProviderFeature -ProviderNamespace Microsoft.Compute -FeatureName AutoOSUpgradePreview
```

It takes approximately 10 minutes for registration state to report as Registered. You can check the current registration status with `Get-AzureRmProviderFeature`. Once registered, ensure that the `Microsoft.Compute` provider is registered with `Register-AzureRmResourceProvider` as follows:

```powershell
Register-AzureRmResourceProvider -ProviderNamespace Microsoft.Compute
```

We recommend that your applications use health probes. To register the provider feature for health probes, use `Register-AzureRmProviderFeature` as follows:

```powershell
Register-AzureRmProviderFeature -ProviderNamespace Microsoft.Compute
```
Again, it takes approximately 10 minutes for registration state to report as Registered. You can check the current registration status with `Get-AzureRmProviderFeature`. Once registered ensure that the `Microsoft.Network` provider is registered with `Register-AzureRmResourceProvider` as follows:

```
Register-AzureRmProviderFeature -ProviderNamespace Microsoft.Network -FeatureName AllowVmssHealthProbe

Register-AzureRmResourceProvider -ProviderNamespace Microsoft.Network
```

## Supported OS images

Only certain OS platform images are currently supported. You cannot currently use custom images that you have created yourself. The `version` property of the platform image must be set to `latest`.

The following SKUs are currently supported (more will be added):

<table>
<thead>
<tr>
<th>PUBLISHER</th>
<th>OFFER</th>
<th>SKU</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicrosoftWindowsServer</td>
<td>WindowsServer</td>
<td>2012-R2-Datacenter</td>
<td>latest</td>
</tr>
<tr>
<td>MicrosoftWindowsServer</td>
<td>WindowsServer</td>
<td>2016-Datacenter</td>
<td>latest</td>
</tr>
<tr>
<td>Canonical</td>
<td>UbuntuServer</td>
<td>16.04-LTS</td>
<td>latest</td>
</tr>
</tbody>
</table>

## Application Health

During an OS Upgrade, VM instances in a scale set are upgraded one batch at a time. The upgrade should continue only if the customer application is healthy on the upgraded VM instances. We recommend that the application provides health signals to the scale set OS Upgrade engine. By default, during OS Upgrades the platform considers VM power state and extension provisioning state to determine if a VM instance is healthy after an upgrade. During the OS Upgrade of a VM instance, the OS disk on a VM instance is replaced with a new disk based on latest image version. After the OS Upgrade has completed, the configured extensions are run on these VMs. Only when all the extensions on a VM are successfully provisioned, is the application considered healthy.

A scale set can optionally be configured with Application Health Probes to provide the platform with accurate information on the ongoing state of the application. Application Health Probes are Custom Load Balancer Probes that are used as a health signal. The application running on a scale set VM instance can respond to external HTTP or TCP requests indicating whether it is healthy. For more information on how Custom Load Balancer Probes work, see [Understand load balancer probes](#). An Application Health Probe is not required for automatic OS upgrades, but it is highly recommended.

If the scale set is configured to use multiple placement groups, probes using a [Standard Load Balancer](#) need to be used.

### Configuring a Custom Load Balancer Probe as Application Health Probe on a scale set

As a best practice, create a load balancer probe explicitly for scale set health. The same endpoint for an existing HTTP probe or TCP probe may be used, but a health probe may require different behavior from a traditional load-balancer probe. For example, a traditional load balancer probe may return unhealthy if the load on the instance is too high, whereas that may not be appropriate for determining the instance health during an automatic OS upgrade. Configure the probe to have a high probing rate of less than two minutes.

The load-balancer probe can be referenced in the `networkProfile` of the scale set and can be associated with either an internal or public facing load-balancer as follows:
Enforce an OS image upgrade policy across your subscription

For safe upgrades, it is highly recommended to enforce an upgrade policy. This policy can require application health probes across your subscription. The following Azure Resource Manager policy rejects deployments that do not have automated OS image upgrade settings configured:

1. Obtain the built-in Azure Resource Manager policy definition with `Get-AzureRmPolicyDefinition` as follows:

   ```powershell
   $policyDefinition = Get-AzureRmPolicyDefinition -Id
   "providers/Microsoft.Authorization/policyDefinitions/465f0161-0087-490a-9ad9-ad6217f4f43a"
   ```

2. Assign policy to a subscription with `New-AzureRmPolicyAssignment` as follows:

   ```powershell
   New-AzureRmPolicyAssignment
   -Name "Enforce automatic OS upgrades with app health checks" 
   -Scope "/subscriptions/<SubscriptionId>" 
   -PolicyDefinition $policyDefinition
   ```

Configure auto-updates

To configure automatic upgrades, ensure that the `automaticOSUpgrade` property is set to `true` in the scale set model definition. You can configure this property with Azure PowerShell or the Azure CLI 2.0.

The following example uses Azure PowerShell (4.4.1 or later) to configure automatic upgrades for the scale set named `myVMSS` in the resource group named `myResourceGroup`:

```powershell
$rgname = myResourceGroup
$vmssname = myVMSS
$vmss = Get-AzureRmVmss -ResourceGroupName $rgname -VmScaleSetName $vmssname
$vmss.UpgradePolicy.AutomaticOSUpgrade = $true
Update-AzureRmVmss -ResourceGroupName $rgname -VmScaleSetName $vmssname -VirtualMachineScaleSet $vmss
```

The following example uses the Azure CLI (2.0.20 or later) to configure automatic upgrades for the scale set named `myVMSS` in the resource group named `myResourceGroup`:

```powershell
rgname="myResourceGroup"
vmssname="myVMSS"
az vmss update --name $vmssname --resource-group $rgname --set upgradePolicy.AutomaticOSUpgrade=true
```

Check the status of an automatic OS upgrade

You can check the status of the most recent OS upgrade performed on your scale set with Azure PowerShell, Azure CLI 2.0, or the REST APIs.

**Azure PowerShell**

To following example uses Azure PowerShell (4.4.1 or later) to check the status for the scale set named `myVMSS` in
the resource group named `myResourceGroup`:

```bash
Get-AzureRmVmssRollingUpgrade -ResourceGroupName myResourceGroup -VMScaleSetName myVMSS
```

**Azure CLI 2.0**

The following example uses the Azure CLI (2.0.20 or later) to check the status for the scale set named `myVMSS` in the resource group named `myResourceGroup`:

```bash
az vmss rolling-upgrade get-latest --resource-group myResourceGroup --name myVMSS
```

**REST API**

The following example uses the REST API to check the status for the scale set named `myVMSS` in the resource group named `myResourceGroup`:

```
GET on
```

The GET call returns properties similar to the following example output:

```json
{
   "properties": {
      "policy": {
         "maxBatchInstancePercent": 20,
         "maxUnhealthyInstancePercent": 5,
         "maxUnhealthyUpgradedInstancePercent": 5,
         "pauseTimeBetweenBatches": "PT0S"
      },
      "runningStatus": {
         "code": "Completed",
         "startTime": "2017-06-16T03:40:14.0924763+00:00",
         "lastAction": "Start",
         "lastActionTime": "2017-06-22T08:45:43.1838042+00:00"
      },
      "progress": {
         "successfulInstanceCount": 3,
         "failedInstanceCount": 0,
         "inprogressInstanceCount": 0,
         "pendingInstanceCount": 0
      }
   },
   "type": "Microsoft.Compute/virtualMachineScaleSets/rollingUpgrades",
   "location": "southcentralus"
}
```

**Automatic OS Upgrade Execution**

To expand on the use of application health probes, scale set OS upgrades execute following steps:

1. If more than 20% of instances are Unhealthy, stop the upgrade; otherwise proceed.
2. Identify the next batch of VM instances to upgrade, with a batch having maximum 20% of total instance count.
3. Upgrade the OS of the next batch of VM instances.
4. If more than 20% of upgraded instances are Unhealthy, stop the upgrade; otherwise proceed.
5. If the customer has configured Application Health Probes, the upgrade waits up to 5 minutes for probes to become healthy, then immediately continues onto the next batch; otherwise, it waits 30 minutes before moving
6. If there are remaining instances to upgrade, goto step 1) for the next batch; otherwise the upgrade is complete.

The scale set OS Upgrade Engine checks for the overall VM instance health before upgrading every batch. While upgrading a batch, there may be other concurrent Planned or Unplanned maintenance happening in Azure Datacenters that may impact availability of your VMs. Hence, it is possible that temporarily more than 20% instances may be down. In such cases, at the end of current batch, the scale set upgrade stops.

Deploy with a template

You can use the following template to deploy a scale set that uses automatic upgrades Automatic rolling upgrades - Ubuntu 16.04-LTS

![Deploy to Azure]

Next steps

For more examples on how to use automatic OS upgrades with scale sets, see the GitHub repo for preview features.
This article describes how you can roll out an OS update to an Azure virtual machine scale set without any
downtime. In this context, an OS update involves changing the version or SKU of the OS or changing the URI of a
custom image. Updating without downtime means updating virtual machines one at a time or in groups (such as
one fault domain at a time) rather than all at once. By doing so, any virtual machines that are not being upgraded
can keep running.

To avoid ambiguity, let's distinguish four types of OS update you might want to perform:

- Changing the version or SKU of a platform image. For example, changing Ubuntu 14.04.2-LTS version from
  14.04.201506100 to 14.04.201507060, or changing the Ubuntu 15.10/latest SKU to 16.04.0-LTS/latest. This
  scenario is covered in this article.
- Changing the URI that points to a new version of a custom image you built. This scenario is covered in this article.
- Changing the image reference of a scale set that was created using Azure Managed Disks.
- Patching the OS from within a virtual machine (examples of this include installing a security patch and running
  Windows Update). This scenario is supported but not covered in this article.

Virtual machine scale sets that are deployed as part of an Azure Service Fabric cluster are not covered here. See
Patch Windows OS in your Service Fabric cluster for more information about patching Service Fabric.

The basic sequence for changing the OS version/SKU of a platform image or the URI of a custom image looks as
follows:

1. Get the virtual machine scale set model.
2. Change the version, SKU, image reference, or URI value in the model.
3. Update the model.
4. Do a manualUpgrade call on the virtual machines in the scale set. This step is only relevant if upgradePolicy is
   set to Manual in your scale set. If it is set to Automatic, all the virtual machines are upgraded at once, thus
   causing downtime.

With this information in mind, let's see how you could update the version of a scale set in PowerShell, and by
using the REST API. These examples cover the case of a platform image, but this article provides enough
information for you to adapt this process to a custom image.

**PowerShell**

This example updates a Windows virtual machine scale set (creating to the new version 4.0.20160229. After
updating the model, it does an update one virtual machine instance at a time.
$rgname = "myrg"
$vmssname = "myvmss"
$newversion = "4.0.20160229"
$instanceid = "1"

# get the VMSS model
$vmss = Get-AzureRmVmss -ResourceGroupName $rgname -VMScaleSetName $vmssname

# set the new version in the model data
$vmss.virtualMachineProfile.storageProfile.imageReference.version = $newversion

# update the virtual machine scale set model
Update-AzureRmVmss -ResourceGroupName $rgname -Name $vmssname -VirtualMachineScaleSet $vmss

# now start updating instances
Update-AzureRmVmssInstance -ResourceGroupName $rgname -VMScaleSetName $vmssname -InstanceId $instanceId

If you are updating the URI for a custom image instead of changing a platform image version, replace the “set the new version” line with a command that will update the source image URI. For example, if the scale set was created without using Azure Managed Disks, the update would look like this:

# set the new version in the model data
$vmss.virtualMachineProfile.storageProfile.osDisk.imageRef.uri = $newURI

If a custom image based scale set was created using Azure Managed Disks, then the image reference would be updated. For example:

# set the new version in the model data
$vmss.virtualMachineProfile.storageProfile.imageReference.id = $newImageReference

The REST API

Here are a couple of Python examples that use the Azure REST API to roll out an OS version update. Both use the lightweight azurerm library of Azure REST API wrapper functions to do a GET on the scale set model, followed by a PUT with an updated model. They also look at virtual machine instances views to identify the virtual machines by update domain.

Vmssupgrade

Vmssupgrade is a Python script that’s used to roll out an OS upgrade to a running virtual machine scale set one update domain at a time.

This script lets you choose specific virtual machines to update or specify an update domain. It supports changing a platform image version or changing the URI of a custom image.

Vmsseditor

Vmsseditor is a general-purpose editor for virtual machine scale sets that shows virtual machine status as a
heatmap where one row represents one update domain. Among other things, you can update the model for a scale set with a new version, SKU, or custom image URI, and then pick fault domains to upgrade. When you do so, all the virtual machines in that update domain are upgraded to the new model. Alternatively, you can do a rolling upgrade based on the batch size of your choice.

The following screenshot shows a model of a scale set for Ubuntu 14.04-2LTS version 14.04.201507060. Many more options have been added to this tool since this screenshot was taken.

After you click **Upgrade** and then **Get Details**, virtual machines in UD 0 start to update.
Virtual Machine Scale Sets can be used with the Azure Desired State Configuration (DSC) extension handler. Virtual machine scale sets provide a way to deploy and manage large numbers of virtual machines, and can elastically scale in and out in response to load. DSC is used to configure the VMs as they come online so they are running the production software.

Differences between deploying to Virtual Machines and Virtual Machine Scale Sets

The underlying template structure for a virtual machine scale set is slightly different from a single VM. Specifically, a single VM deploys extensions under the "virtualMachines" node. There is an entry of type "extensions" where DSC is added to the template

```
"resources": [  
  {
    "name": "Microsoft.Powershell.DSC",
    "type": "extensions",
    "location": [resourceGroup().location],
    "apiVersion": "2015-06-15",
    "dependsOn": [
      [concat('Microsoft.Compute/virtualMachines/', variables('vmName'))]
    ],
    "tags": {
      "displayName": "dscExtension"
    },
    "properties": {
      "publisher": "Microsoft.Powershell",
      "type": "DSC",
      "typeHandlerVersion": "2.20",
      "autoUpgradeMinorVersion": false,
      "forceUpdateTag": [parameters('dscExtensionUpdateTagVersion')]
    },
    "configuration": {
      "url": [concat(parameters('_artifactsLocation'), '/', variables('dscExtensionArchiveFolder'), '/', variables('dscExtensionArchiveFileName'))],
      "script": "DscExtension.ps1",
      "function": "Main"
    },
    "configurationArguments": {
      "nodeName": variables('vmName')
    },
    "protectedSettings": {
      "configurationUrlSasToken": [parameters('_artifactsLocationSasToken')]
    }
  }
]  
```

A virtual machine scale set node has a "properties" section with the "VirtualMachineProfile", "extensionProfile" attribute. DSC is added under "extensions"
Behavior for a Virtual Machine Scale Set

The behavior for a virtual machine scale set is identical to the behavior for a single VM. When a new VM is created, it is automatically provisioned with the DSC extension. If a newer version of the WMF is required by the extension, the VM reboots before coming online. Once it is online, it downloads the DSC configuration .zip and provision it on the VM. More details can be found in the [Azure DSC Extension Overview](#).

Next steps

Examine the [Azure Resource Manager template for the DSC extension](#).

Learn how the [DSC extension securely handles credentials](#).

For more information on the Azure DSC extension handler, see [Introduction to the Azure Desired State Configuration extension handler](#).

For more information about PowerShell DSC, visit the [PowerShell documentation center](#).
When you deploy an Azure virtual machine scale set through the portal, certain network properties are defaulted, for example an Azure Load Balancer with inbound NAT rules. This article describes how to use some of the more advanced networking features that you can configure with scale sets.

You can configure all of the features covered in this article using Azure Resource Manager templates. Azure CLI and PowerShell examples are also included for selected features. Use CLI 2.10, and PowerShell 4.2.0 or later.

### Accelerated Networking

Azure **Accelerated Networking** improves network performance by enabling single root I/O virtualization (SR-IOV) to a virtual machine. To use accelerated networking with scale sets, set `enableAcceleratedNetworking` to `true` in your scale set's `networkInterfaceConfigurations` settings. For example:

```
"networkProfile": {
    "networkInterfaceConfigurations": [
    {
        "name": "niconfig1",
        "properties": {
            "primary": true,
            "enableAcceleratedNetworking": true,
            "ipConfigurations": [ ...
        }
    }
    ]
}
```

### Create a scale set that references an existing Azure Load Balancer

When a scale set is created using the Azure portal, a new load balancer is created for most configuration options. If you create a scale set, which needs to reference an existing load balancer, you can do this using CLI. The following example script creates a load balancer and then creates a scale set, which references it:

```
az network lb create -g lbtest -n mylb --vnet-name myvnet --subnet mysubnet --public-ip-address-allocation Static --backend-pool-name mybackendpool

az vmss create -g lbtest -n myvmss --image Canonical:UbuntuServer:16.04-LTS:latest --admin-username negat --ssh-key-value /home/myuser/.ssh/id_rsa.pub --upgrade-policy-mode Automatic --instance-count 3 --vnet-name myvnet --subnet mysubnet --lb mylb --backend-pool-name mybackendpool
```

### Configurable DNS Settings

By default, scale sets take on the specific DNS settings of the VNET and subnet they were created in. You can however, configure the DNS settings for a scale set directly.

**Creating a scale set with configurable DNS servers**

To create a scale set with a custom DNS configuration using CLI 2.0, add the `--dns-servers` argument to the `vmss create` command, followed by space separated server ip addresses. For example:
To configure custom DNS servers in an Azure template, add a dnsSettings property to the scale set networkInterfaceConfigurations section. For example:

```
"dnsSettings":{
  "dnsServers":["10.0.0.6", "10.0.0.5"]
}
```

### Creating a scale set with configurable virtual machine domain names

To create a scale set with a custom DNS name for virtual machines using CLI 2.0, add the `--vm-domain-name` argument to the `vmss create` command, followed by a string representing the domain name.

To set the domain name in an Azure template, add a `dnsSettings` property to the scale set networkInterfaceConfigurations section. For example:

```
"networkProfile": {
  "networkInterfaceConfigurations": [
    {
      "name": "nic1",
      "properties": {
        "primary": "true",
        "ipConfigurations": [
          {
            "name": "ip1",
            "properties": {
              "subnet": {
                "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/virtualNetworks/', variables('vnetName'), '/subnets/subnet1')]"},
              "publicIPAddressconfiguration": {
                "name": "publicip",
                "properties": {
                  "idleTimeoutInMinutes": 10,
                  "dnsSettings": {
                    "domainNameLabel": "[parameters('vmssDnsName')]"
                  }
                }
              }
            }
          }
        ]
      }
    }
  ]
}
```

The output, for an individual virtual machine dns name would be in the following form:

```
<vm><vmindex>.<specifiedVmssDomainNameLabel>
```

### Public IPv4 per virtual machine

In general, Azure scale set virtual machines do not require their own public IP addresses. For most scenarios, it is more economical and secure to associate a public IP address to a load balancer or to an individual virtual machine (aka a jumpbox), which then routes incoming connections to scale set virtual machines as needed (for example, through inbound NAT rules).
However, some scenarios do require scale set virtual machines to have their own public IP addresses. An example is gaming, where a console needs to make a direct connection to a cloud virtual machine, which is doing game physics processing. Another example is where virtual machines need to make external connections to one another across regions in a distributed database.

**Creating a scale set with public IP per virtual machine**

To create a scale set that assigns a public IP address to each virtual machine with CLI 2.0, add the `--public-ip-per-vm` parameter to the `vmss create` command.

To create a scale set using an Azure template, make sure the API version of the Microsoft.Compute/virtualMachineScaleSets resource is at least **2017-03-30**, and add a `publicIpAddressConfiguration` JSON property to the scale set ipConfigurations section. For example:

```
"publicIpAddressConfiguration": {
  "name": "pub1",
  "properties": {
    "idleTimeoutInMinutes": 15
  }
}
```

Example template: *[201-vmss-public-ip-linux]*

**Querying the public IP addresses of the virtual machines in a scale set**

To list the public IP addresses assigned to scale set virtual machines using CLI 2.0, use the `az vmss list-instance-public-ips` command.

To list scale set public IP addresses using PowerShell, use the `Get-AzureRmPublicIpAddress` command. For example:

```
PS C:\> Get-AzureRmPublicIpAddress -ResourceGroupName myrg -VirtualMachineScaleSetName myvmss
```

You can also query the public IP addresses by referencing the resource id of the public IP address configuration directly. For example:

```
PS C:\> Get-AzureRmPublicIpAddress -ResourceGroupName myrg -Name myvmsspip
```

To query the public IP addresses assigned to scale set virtual machines using the Azure Resource Explorer, or the Azure REST API with version **2017-03-30** or higher.

To view public IP addresses for a scale set using the Resource Explorer, look at the `publicipaddresses` section under your scale set. For example:

https://resources.azure.com/subscriptions/_your_sub_id_/resourceGroups/_your_rg_/providers/Microsoft.Compute/virtualMachineScaleSets/_your_vmss_/publicipaddresses

```
```

Example output:
Multiple IP addresses per NIC

Every NIC attached to a VM in a scale set can have one or more IP configurations associated with it. Each configuration is assigned one private IP address. Each configuration may also have one public IP address resource associated with it. To understand how many IP addresses can be assigned to a NIC, and how many public IP addresses you can use in an Azure subscription, refer to Azure limits.

Multiple NICs per virtual machine

You can have up to 8 NICs per virtual machine, depending on machine size. The maximum number of NICs per machine is available in the VM size article. The following example is a scale set network profile showing multiple NIC entries, and multiple public IPs per virtual machine:

```json
"networkProfile": {
  "networkInterfaceConfigurations": [
    {
      "name": "nic1",
      "properties": {
        "primary": "true",
        "ipConfigurations": ["pub1"]
      }
    }
  ]
}
```
"ipConfigurations": [ 
  { 
    "name": "ip1", 
    "properties": { 
      "subnet": { 
        "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/virtualNetworks/', variables('vnetName'), '/subnets/subnet1')]"
      }, 
      "publicipaddressconfiguration": { 
        "name": "pub1", 
        "properties": { 
          "idleTimeoutInMinutes": 15 
        }
      }, 
      "loadBalancerInboundNatPools": [ 
        { 
          "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/inboundNatPools/natPool1')]"
        }, 
        "loadBalancerBackendAddressPools": [ 
          { 
            "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/backendAddressPools/addressPool1')]"
          }
        ]
      }
    }
  }, 
  { 
    "name": "nic2", 
    "properties": { 
      "primary": "false", 
      "ipConfigurations": [ 
        { 
          "name": "ip1", 
          "properties": { 
            "subnet": { 
              "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/virtualNetworks/', variables('vnetName'), '/subnets/subnet1')]"
            }, 
            "publicipaddressconfiguration": { 
              "name": "pub1", 
              "properties": { 
                "idleTimeoutInMinutes": 15 
              }
            }, 
            "loadBalancerInboundNatPools": [ 
              { 
                "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/inboundNatPools/natPool1')]"
              }, 
              "loadBalancerBackendAddressPools": [ 
                { 
                  "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/backendAddressPools/addressPool1')]"
                }
              ]
            }
          }
        }
      ]
    }
  }
]
NSG per scale set

Network Security Groups can be applied directly to a scale set, by adding a reference to the network interface configuration section of the scale set virtual machine properties.

For example:

```json
"networkProfile": {
    "networkInterfaceConfigurations": [
        {
            "name": "nic1",
            "properties": {
                "primary": "true",
                "ipConfigurations": [
                    {
                        "name": "ip1",
                        "properties": {
                            "subnet": {
                                "id": "[concat('/subscriptions/',
                                   subscription().subscriptionId,'/resourceGroups/', resourceGroup().name,
                                   '/providers/Microsoft.Network/virtualNetworks/', variables('vnetName'), '/subnets/subnet1')]"
                            }"[concat('/subscriptions/',
                                subscription().subscriptionId,'/resourceGroups/', resourceGroup().name,
                                '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/loadBalancerInboundNatPools/natPool1'])"
                        }"[concat('/subscriptions/',
                                subscription().subscriptionId,'/resourceGroups/', resourceGroup().name,
                                '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/backendAddressPools/addressPool1'])"
                    }
                ]
            }
        }
    ]
    "networkSecurityGroup": {
        "id": "[concat('/subscriptions/',
                         subscription().subscriptionId,'/resourceGroups/',
                         resourceGroup().name, '/providers/Microsoft.Network/networkSecurityGroups/', variables('nsgName'))]"
    }
}
```

Next steps

For more information about Azure virtual networks, refer to [this documentation](https://docs.microsoft.com/en-us/azure/virtual-networks/).
Customers with a Resource Manager template for creating a scale set not using managed disk may wish to modify it to use managed disk. This article shows how to do this, using as an example a pull request from the Azure Quickstart Templates, a community-driven repo for sample Resource Manager templates. The full pull request can be seen here: https://github.com/Azure/azure-quickstart-templates/pull/2998, and the relevant parts of the diff are below, along with explanations:

### Making the OS disks managed

In the diff below, we can see that we have removed several variables related to storage account and disk properties. Storage account type is no longer necessary (Standard_LRS is the default), but we could still specify it if we wished to. Only Standard_LRS and Premium_LRS are supported with managed disk. New storage account suffix, unique string array, and sa count were used in the old template to generate storage account names. These variables are no longer necessary in the new template because managed disk automatically creates storage accounts on the customer’s behalf. Similarly, vhd container name and os disk name are no longer necessary because managed disk automatically names the underlying storage blob containers and disks.

```json
"variables": {
  - "storageAccountType": "Standard_LRS",
  - "namingInfix": 
  - "longNamingInfix": 
  - "newStorageAccountSuffix": 
  - "uniqueStringArray": [
    - "saCount": 
    - "vhdContainerName": 
    - "osDiskName": 
  - "addressPrefix": "10.0.0.0/16",
  - "subnetPrefix": "10.0.0.0/24",
  - "virtualNetworkName": 
}
```

In the diff below, we can see that we updated the compute api version to 2016-04-30-preview, which is the earliest required version for managed disk support with scale sets. Note that we could still use unmanaged disks in the new api version with the old syntax if desired. In other words, if we only update the compute api version and don’t change anything else, the template should continue to work as before.
In the diff below, we can see that we are removing the storage account resource from the resources array completely. We no longer need them since managed disk creates them automatically on our behalf.

In the diff below, we can see that we are removing the depends on clause referring from the scale set to the loop that was creating storage accounts. In the old template, this was ensuring that the storage accounts were created before the scale set began creation, but this clause is no longer necessary with managed disk. We also remove the vhd containers property, and the os disk name property as these properties are automatically handled under the hood by managed disk. If we wished, we could add "managedDisk": { "storageAccountType": "Premium_LRS" } in the "osDisk" configuration if we wanted premium OS disks. Only VMs with an uppercase or lowercase 's' in the VM sku can use premium disks.
There is no explicit property in the scale set configuration for whether to use managed or unmanaged disk. The scale set knows which to use based on the properties that are present in the storage profile. Thus, it is important when modifying the template to ensure that the right properties are in the storage profile of the scale set.

### Data disks

With the changes above, the scale set uses managed disks for the OS disk, but what about data disks? To add data disks, add the "dataDisks" property under "storageProfile" at the same level as "osDisk". The value of the property is a JSON list of objects, each of which has properties "lun" (which must be unique per data disk on a VM), "createOption" ("empty" is currently the only supported option), and "diskSizeGB" (the size of the disk in gigabytes; must be greater than 0 and less than 1024) as in the following example:

```json
"dataDisks": [
  {
    "lun": "1",
    "createOption": "empty",
    "diskSizeGB": "1023"
  }
]
```

If you specify \( n \) disks in this array, each VM in the scale set gets \( n \) data disks. Do note, however, that these data disks are raw devices. They are not formatted. It is up to the customer to attach, partition, and format the disks before using them. Optionally, we could also specify "managedDisk": { "storageAccountType": "Premium_LRS" } in each data disk object to specify that it should be a premium data disk. Only VMs with an uppercase or lowercase 's' in the VM sku can use premium disks.

To learn more about using data disks with scale sets, see [this article](#).

### Next steps
For example Resource Manager templates using scale sets, search for "vmss" in the Azure Quickstart Templates github repo.

For general information, check out the main landing page for scale sets.
Problem – you’ve created an autoscaling infrastructure in Azure Resource Manager using virtual machine scale sets – for example, by deploying a template like this one: https://github.com/Azure/azure-quickstart-templates/tree/master/201-vmss-bottle-autoscale – you have your scale rules defined and it works great, except no matter how much load you put on the VMs, it doesn’t autoscale.

Troubleshooting steps

Some things to consider include:

- How many vCPUs does each VM have, and are you loading each vCPU? The preceding sample Azure Quickstart template has a do_work.php script, which loads a single vCPU. If you’re using a VM bigger than a single-vCPU VM size like Standard_A1 or D1, you’d need to run this load multiple times. Check how many vCPUs for your VMs by reviewing Sizes for Windows virtual machines in Azure.

- How many VMs in the virtual machine scale set, are you doing work on each one?

A scale-out event only takes place when the average CPU across all the VMs in a scale set exceeds the threshold value, over the time internal defined in the autoscale rules.

- Did you miss any scale events?

Check the audit logs in the Azure portal for scale events. Maybe there was a scale up and a scale down that was missed. You can filter by “Scale”.

- Are your scale-in and scale-out thresholds sufficiently different?

Suppose you set a rule to scale out when average CPU is greater than 50% over five minutes, and to scale in when average CPU is less than 50%. This setting would cause a “flapping” problem when CPU usage is close to the threshold, with scale actions constantly increasing and decreasing the size of the set. Because of this setting, the autoscale service tries to prevent “flapping”, which can manifest as not scaling. Therefore, be sure your scale-out and scale-in thresholds are sufficiently different to allow some space in between scaling.

- Did you write your own JSON template?

It is easy to make mistakes, so start with a template like the one above which is proven to work, and make small incremental changes.

- Can you manually scale in or out?

Try redeploying the virtual machine scale set resource with a different “capacity” setting to change the number of VMs manually. An example template is here: https://github.com/Azure/azure-quickstart-templates/tree/master/201-vmss-bottle-autoscale
Check your Microsoft.Compute/virtualMachineScaleSet, and Microsoft.Insights resources in the Azure Resource Explorer

The Azure Resource Explorer is an indispensable troubleshooting tool that shows you the state of your Azure Resource Manager resources. Click on your subscription and look at the Resource Group you are troubleshooting. Under the Compute resource provider, look at the virtual machine scale set you created and check the Instance View, which shows you the state of a deployment. Also, check the instance view of VMs in the virtual machine scale set. Then, go into the Microsoft.Insights resource provider and check that the autoscale rules look right.

Is the Diagnostic extension working and emitting performance data?

Update: Azure autoscale has been enhanced to use a host-based metrics pipeline, which no longer requires a diagnostics extension to be installed. The next few paragraphs no longer apply if you create an autoscaling application using the new pipeline. An example of Azure templates that have been converted to use the host pipeline is available here: https://github.com/Azure/azure-quickstart-templates/tree/master/201-vmss-bottle-autoscale.

Using host-based metrics for autoscale is better for the following reasons:

- Fewer moving parts as no diagnostics extensions need to be installed.
- Simpler templates. Just add insights autoscale rules to an existing scale set template.
- More reliable reporting and faster launching of new VMs.

The only reasons you might want to keep using a diagnostic extension is if you need memory diagnostics reporting/scaling. Host-based metrics don’t report memory.

With that in mind, only follow the rest of this article if you’re using diagnostic extensions for your autoscaling.

Autoscale in Azure Resource Manager can work (but no longer has to) by means of a VM extension called the Diagnostics Extension. It emits performance data to a storage account you define in the template. This data is then aggregated by the Azure Monitor service.

If the Insights service can’t read data from the VMs, it is supposed to send you an email. For example, you get an email if the VMs are down. Be sure to check your email, at the email address you specified when you created your Azure account.

You can also look at the data yourself. Look at the Azure storage account using a cloud explorer. For example, using the Visual Studio Cloud Explorer, log in and pick the Azure subscription you’re using. Then, look at the Diagnostics storage account name referenced in the Diagnostics extension definition in your deployment template.
You see a bunch of tables where the data from each VM is being stored. Taking Linux and the CPU metric as an example, look at the most recent rows. The Visual Studio cloud explorer supports a query language so you can run a query. For example, you can run a query for "Timestamp gt datetime'2016-02-02T21:20:00Z'" to make sure you get the most recent events. The timezone corresponds to UTC. Does the data you see in there correspond to the scale rules you set up? In the following example, the CPU for machine 20 started increasing to 100% over the last five minutes.

If the data is not there, it implies the problem is with the diagnostic extension running in the VMs. If the data is there, it implies there is either a problem with your scale rules, or with the Insights service. Check Azure Status.

Once you’ve been through these steps, if you’re still having autoscale problems, you can try the following resources:

- Read the forums on MSDN, or Stack overflow
- Log a support call. Be prepared to share the template and a view of your performance data.
Get answers to frequently asked questions about virtual machine scale sets in Azure.

**Autoscale**

**What are best practices for Azure Autoscale?**

For best practices for Autoscale, see [Best practices for autoscaling virtual machines](#).

**Where do I find metric names for autoscaling that uses host-based metrics?**

For metric names for autoscaling that uses host-based metrics, see [Supported metrics with Azure Monitor](#).

**Are there any examples of autoscaling based on an Azure Service Bus topic and queue length?**

Yes. For examples of autoscaling based on an Azure Service Bus topic and queue length, see [Azure Monitor autoscaling common metrics](#).

For a Service Bus queue, use the following JSON:

```
"metricName": "MessageCount",
"metricNamespace": "",
"metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/Microsoft.ServiceBus/namespaces/mySB/queues/myqueue"
```

For a storage queue, use the following JSON:

```
"metricName": "ApproximateMessageCount",
"metricNamespace": "",
"metricResourceUri": "/subscriptions/s1/resourceGroups/rg1/providers/Microsoft.ClassicStorage/storageAccounts/mystorage/services/queue/queues/mystoragequeue"
```

Replace example values with your resource Uniform Resource Identifiers (URIs).

**Should I autoscale by using host-based metrics or a diagnostics extension?**

You can create an autoscale setting on a VM to use host-level metrics or guest OS-based metrics.

For a list of supported metrics, see [Azure Monitor autoscaling common metrics](#).

For a full sample for virtual machine scale sets, see [Advanced autoscale configuration by using Resource Manager templates for virtual machine scale sets](#).

The sample uses the host-level CPU metric and a message count metric.

**How do I set alert rules on a virtual machine scale set?**

You can create alerts on metrics for virtual machine scale sets via PowerShell or Azure CLI. For more information, see [Azure Monitor PowerShell quick start samples](#) and [Azure Monitor cross-platform CLI quick start samples](#).

The TargetResourceId of the virtual machine scale set looks like this:

/subscriptions/yoursubscriptionid/resourceGroups/yourresourcegroup/providers/Microsoft.Compute/virtualMachineScaleSets/yourvmssname
You can choose any VM performance counter as the metric to set an alert for. For more information, see Guest OS metrics for Resource Manager-based Windows VMs and Guest OS metrics for Linux VMs in the Azure Monitor autoscaling common metrics article.

**How do I set up autoscale on a virtual machine scale set by using PowerShell?**

To set up autoscale on a virtual machine scale set by using PowerShell, see the blog post [How to add autoscale to an Azure virtual machine scale set](https://aka.ms/autoscale-vmscale-set).

**Certificates**

**How do I securely ship a certificate to the VM?** How do I provision a virtual machine scale set to run a website where the SSL for the website is shipped securely from a certificate configuration? (The common certificate rotation operation would be almost the same as a configuration update operation.) Do you have an example of how to do this?

To securely ship a certificate to the VM, you can install a customer certificate directly into a Windows certificate store from the customer's key vault.

Use the following JSON:

```json
"secrets": [
  {
    "sourceVault": {
      "id": "/subscriptions/{subscriptionid}/resourceGroups/myrg1/providers/Microsoft.KeyVault/vaults/mykeyvault1"
    },
    "vaultCertificates": [
      {
        "certificateUrl": "https://mykeyvault1.vault.azure.net/secrets/{secretname}/{secret-version}",
        "certificateStore": "certificateStoreName"
      }
    ]
  }
]
```

The code supports Windows and Linux.

For more information, see [Create or update a virtual machine scale set](https://aka.ms/vmss).

**Example of Self-signed certificate**

1. Create a self-signed certificate in a key vault.

   Use the following PowerShell commands:

   ```powershell
   Import-Module "C:\Users\mikhegn\Downloads\Service-Fabric-master\Scripts\ServiceFabricRPHelpers\ServiceFabricRPHelpers.psm1"
   Login-AzureRmAccount
   Invoke-AddCertToKeyVault -SubscriptionId <Your SubID> -ResourceGroupName KeyVault -Location westus -VaultName MikhegnVault -CertificateName VMSSCert -Password VmssCert -CreateSelfSignedCertificate -DnsName vmss.mikhegn.azure.com -OutputPath c:\users\mikhegn\desktop
   ```

   This command gives you the input for the Azure Resource Manager template.

   For an example of how to create a self-signed certificate in a key vault, see [Service Fabric cluster security scenarios](https://aka.ms/servicefabricclustersecurity).

2. Change the Resource Manager template.

   Add this property to `virtualMachineProfile`, as part of the virtual machine scale set resource:
Can I specify an SSH key pair to use for SSH authentication with a Linux virtual machine scale set from a Resource Manager template?

Yes. The REST API for `osProfile` is similar to the standard VM REST API. Include `osProfile` in your template:

```json
"osProfile": {
    "computerName": "[variables('vmName')]",
    "adminUsername": "[parameters('adminUsername')]",
    "linuxConfiguration": {
        "disablePasswordAuthentication": "true",
        "ssh": {
            "publicKeys": [
                {
                    "path": "[variables('sshKeyPath')]",
                    "keyData": "[parameters('sshKeyData')]"
                }
            ]
        }
    }
}
```

This JSON block is used in the 101-vm-sshkey GitHub quick start template.

The OS profile also is used in the grelayhost.json GitHub quick start template.

For more information, see Create or update a virtual machine scale set.

How do I remove deprecated certificates?

To remove deprecated certificates, remove the old certificate from the vault certificates list. Leave all the certificates that you want to remain on your computer in the list. This does not remove the certificate from all your VMs. It also does not add the certificate to new VMs that are created in the virtual machine scale set.

To remove the certificate from existing VMs, write a custom script extension to manually remove the certificates from your certificate store.

How do I inject an existing SSH public key into the virtual machine scale set SSH layer during provisioning? I want to store the SSH public key values in Azure Key Vault, and then use them in my Resource Manager template.

If you are providing the VMs only with a public SSH key, you don't need to put the public keys in Key Vault.
keys are not secret.

You can provide SSH public keys in plain text when you create a Linux VM:

```
"linuxConfiguration": {
  "ssh": {
    "publicKeys": [
      {
        "path": "path",
        "keyData": "publickey"
      }
    ]
  }
}
```

<table>
<thead>
<tr>
<th>LINUXCONFIGURATION ELEMENT NAME</th>
<th>REQUIRED</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssh</td>
<td>No</td>
<td>Collection</td>
<td>Specifies the SSH key configuration for a Linux OS</td>
</tr>
<tr>
<td>path</td>
<td>Yes</td>
<td>String</td>
<td>Specifies the Linux file path where the SSH keys or certificate should be located</td>
</tr>
<tr>
<td>keyData</td>
<td>Yes</td>
<td>String</td>
<td>Specifies a base64-encoded SSH public key</td>
</tr>
</tbody>
</table>

For an example, see the 101-vm-sshkey GitHub quick start template.

When I run `Update-AzureRmVmss` after adding more than one certificate from the same key vault, I see the following message:

```
Update-AzureRmVmss: List secret contains repeated instances of /subscriptions//resourceGroups/internal-rg-dev/providers/Microsoft.KeyVault/vaults/internal-keyvault-dev, which is disallowed.
```

This can happen if you try to re-add the same vault instead of using a new vault certificate for the existing source vault. The `Add-AzureRmVmssSecret` command does not work correctly if you are adding additional secrets.

To add more secrets from the same key vault, update the `$vmss.properties.osProfile.secrets[0].vaultCertificates` list.

For the expected input structure, see Create or update a virtual machine set.

Find the secret in the virtual machine scale set object that is in the key vault. Then, add your certificate reference (the URL and the secret store name) to the list associated with the vault.

**NOTE**
Currently, you cannot remove certificates from VMs by using the virtual machine scale set API.

New VMs will not have the old certificate. However, VMs that have the certificate and which are already deployed will have the old certificate.

**Can I push certificates to the virtual machine scale set without providing the password, when the certificate is in the secret store?**
You do not need to hard-code passwords in scripts. You can dynamically retrieve passwords with the permissions you use to run the deployment script. If you have a script that moves a certificate from the secret store key vault, the secret store `get certificate` command also outputs the password of the .pfx file.
How does the Secrets property of virtualMachineProfile.osProfile for a virtual machine scale set work? Why do I need the sourceVault value when I have to specify the absolute URI for a certificate by using the certificateUrl property?

A Windows Remote Management (WinRM) certificate reference must be present in the Secrets property of the OS profile.

The purpose of indicating the source vault is to enforce access control list (ACL) policies that exist in a user’s Azure Cloud Service model. If the source vault isn’t specified, users who do not have permissions to deploy or access secrets to a key vault would be able to through a Compute Resource Provider (CRP). ACLs exist even for resources that do not exist.

If you provide an incorrect source vault ID but a valid key vault URL, an error is reported when you poll the operation.

If I add secrets to an existing virtual machine scale set, are the secrets injected into existing VMs, or only into new ones?

Certificates are added to all your VMs, even preexisting ones. If your virtual machine scale set upgradePolicy property is set to manual, the certificate is added to the VM when you perform a manual update on the VM.

Where do I put certificates for Linux VMs?

To learn how to deploy certificates for Linux VMs, see Deploy certificates to VMs from a customer-managed key vault.

How do I add a new vault certificate to a new certificate object?

To add a vault certificate to an existing secret, see the following PowerShell example. Use only one secret object.

```powershell
Update-AzureRmVmss -VirtualMachineScaleSet $vmss -ResourceGroup $rg -Name $vmssName
```

What happens to certificates if you reimage a VM?

If you reimage a VM, certificates are deleted. Reimaging deletes the entire OS disk.

What happens if you delete a certificate from the key vault?

If the secret is deleted from the key vault, and then you run stop deallocate for all your VMs and then start them again, you will encounter a failure. The failure occurs because the CRP needs to retrieve the secrets from the key vault, but it cannot. In this scenario, you can delete the certificates from the virtual machine scale set model.

The CRP component does not persist customer secrets. If you run stop deallocate for all VMs in the virtual machine scale set, the cache is deleted. In this scenario, secrets are retrieved from the key vault.

You don’t encounter this problem when scaling out because there is a cached copy of the secret in Azure Service Fabric (in the single-fabric tenant model).

Why do I have to specify the exact location for the certificate URL (https://vault.azure.net:443/secrets/), as indicated in Service Fabric cluster security scenarios?

The Azure Key Vault documentation states that the Get Secret REST API should return the latest version of the secret if the version is not specified.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td><a href="https://mykeyvault.vault.azure.net/secrets/%5BSecret-name%5D/%5Bsecret-version%5D?api-version=(api-version)">https://mykeyvault.vault.azure.net/secrets/[Secret-name]/[secret-version]?api-version=(api-version)</a></td>
</tr>
</tbody>
</table>
Why do I have to specify the certificate version when I use Key Vault?

The purpose of the Key Vault requirement to specify the certificate version is to make it clear to the user what certificate is deployed on their VMs.

If you create a VM and then update your secret in the key vault, the new certificate is not downloaded to your VMs. But your VMs appear to reference it, and new VMs get the new secret. To avoid this, you are required to reference a secret version.

My team works with several certificates that are distributed to us as .cer public keys. What is the recommended approach for deploying these certificates to a virtual machine scale set?

To deploy .cer public keys to a virtual machine scale set, you can generate a .pfx file that contains only .cer files. To do this, use `X509ContentType = Pfx`. For example, load the .cer file as an `x509Certificate2` object in C# or PowerShell, and then call the method.

For more information, see `X509Certificate.Export Method (X509ContentType, String)`.

I do not see an option for users to pass in certificates as base64 strings. Most other resource providers have this option.

To emulate passing in a certificate as a base64 string, you can extract the latest versioned URL in a Resource Manager template. Include the following JSON property in your Resource Manager template:

```
"certificateUrl": "[reference(resourceId(parameters('vaultResourceGroup'), 'Microsoft.KeyVault/vaults/secrets', parameters('vaultName'), parameters('secretName')), '2015-06-01').secretUriWithVersion]"
```

Do I have to wrap certificates in JSON objects in key vaults?

In virtual machine scale sets and VMs, certificates must be wrapped in JSON objects.

We also support the content type application/x-pkcs12. For instructions on using application/x-pkcs12, see PFX certificates in Azure Key Vault.

We currently do not support .cer files. To use .cer files, export them into .pfx containers.

Compliance and Security

Are virtual machine scale sets PCI-compliant?

Virtual machine scale sets are a thin API layer on top of the CRP. Both components are part of the compute platform in the Azure service tree.

From a compliance perspective, virtual machine scale sets are a fundamental part of the Azure compute platform. They share a team, tools, processes, deployment methodology, security controls, just-in-time (JIT) compilation, monitoring, alerting, and so on, with the CRP itself. Virtual machine scale sets are Payment Card Industry (PCI)-compliant because the CRP is part of the current PCI Data Security Standard (DSS) attestation.

For more information, see the Microsoft Trust Center.

Does Azure Managed Service Identity work with VM scale sets?


Extensions
**How do I delete a virtual machine scale set extension?**

To delete a virtual machine scale set extension, use the following PowerShell example:

```
$vmss = Get-AzureRmVmss -ResourceGroupName "resource_group_name" -VMScaleSetName "vmssName"
$vmss=Remove-AzureRmVmssExtension -VirtualMachineScaleSet $vmss -Name "extensionName"
Update-AzureRmVmss -ResourceGroupName "resource_group_name" -VMScaleSetName "vmssName" -VirtualMachineScaleSet $vmss
```

You can find the extensionName value in `$vmss`.

**Is there a virtual machine scale set template example that integrates with Operations Management Suite?**


**Extensions seem to run in parallel on virtual machine scale sets. This causes my custom script extension to fail. What can I do to fix this?**

To learn about extension sequencing in virtual machine scale sets, see [Extension sequencing in Azure virtual machine scale sets](https://docs.microsoft.com/en-us/azure/virtual-machines/extensions/extension-sequencing).

**How do I reset the password for VMs in my virtual machine scale set?**

To reset the password for VMs in your virtual machine scale set, use VM access extensions. Use the following PowerShell example:

```
$vmssName = "myvmss"
$vmssResourceGroup = "myvmssrg"
$publicConfig = @"{UserName" = "newuser"
$privateConfig = @"{Password" = "********"
$extName = "VMAccessAgent"
$publisher = "Microsoft.Compute"
$vmss = Get-AzureRmVmss -ResourceGroupName $vmssResourceGroup -VMScaleSetName $vmssName
$vmss = Add-AzureRmVmssExtension -VirtualMachineScaleSet $vmss -Name $extName -Publisher $publisher -Setting $publicConfig -ProtectedSetting $privateConfig -Type $extName -TypeHandlerVersion "2.0" -AutoUpgradeMinorVersion $true
Update-AzureRmVmss -ResourceGroupName $vmssResourceGroup -Name $vmssName -VirtualMachineScaleSet $vmss
```

**How do I add an extension to all VMs in my virtual machine scale set?**

If update policy is set to **automatic**, redeploying the template with the new extension properties updates all VMs.

If update policy is set to **manual**, first update the extension, and then manually update all instances in your VMs.

**If the extensions associated with an existing virtual machine scale set are updated, are existing VMs affected? (That is, will the VMs not match the virtual machine scale set model?) Or are they ignored? When an existing machine is service-healed or reimaged, are the scripts that are currently configured on the virtual machine scale set executed, or are the scripts that were configured when the VM was first created used?**

If the extension definition in the virtual machine scale set model is updated and the upgradePolicy property is set to **automatic**, it updates the VMs. If the upgradePolicy property is set to **manual**, extensions are flagged as not matching the model.

If an existing VM is service-healed, it appears as a reboot, and the extensions are not rerun. If it is reimaged, it's like replacing the OS drive with the source image. Any specialization from the latest model, such as extensions, are run.

**How do I join a virtual machine scale set to an Azure AD domain?**

To join a virtual machine scale set to an Azure Active Directory (Azure AD) domain, you can define an extension.
To define an extension, use the JsonADDomainExtension property:

```
"extensionProfile": {
  "extensions": [
    {
      "name": "joindomain",
      "properties": {
        "publisher": "Microsoft.Compute",
        "type": "JsonADDomainExtension",
        "typeHandlerVersion": "1.3",
        "settings": {
          "Name": "[parameters('domainName')]",
          "OUPath": "[variables('ouPath')]",
          "User": "[variables('domainAndUsername')]",
          "Restart": "true",
          "Options": "[variables('domainJoinOptions')]
        },
        "protectedsettings": {
          "Password": "[parameters('domainJoinPassword')]
        }
      }
    }
  ]
}
```

My virtual machine scale set extension is trying to install something that requires a reboot. For example, 
"commandToExecute": "powershell.exe -ExecutionPolicy Unrestricted Install-WindowsFeature –Name FS-Resource-Manager –IncludeManagementTools"

If your virtual machine scale set extension is trying to install something that requires a reboot, you can use the Azure Automation Desired State Configuration (Automation DSC) extension. If the operating system is Windows Server 2012 R2, Azure pulls in the Windows Management Framework (WMF) 5.0 setup, reboots, and then continues with the configuration.

How do I turn on antimalware in my virtual machine scale set?

To turn on antimalware on your virtual machine scale set, use the following PowerShell example:

```
$rgname = 'autolap'
$vmssname = 'autolapbr'
$location = 'eastus'

# Retrieve the most recent version number of the extension.
$allVersions= (Get-AzureRmVMExtensionImage -Location $location -PublisherName "Microsoft.Azure.Security" -Type "IaaSAntimalware").Version
$versionString = $allVersions[($allVersions.count)-1].Split(".")[0] + "." + $allVersions[($allVersions.count)-1].Split(".")[1]

$VMSS = Get-AzureRmVmss -ResourceGroupName $rgname -VMScaleSetName $vmssname
echo $VMSS
Add-AzureRmVmssExtension -VirtualMachineScaleSet $VMSS -Name "IaaSAntimalware" -Publisher "Microsoft.Azure.Security" -Type "IaaSAntimalware" -TypeHandlerVersion $versionString
Update-AzureRmVmss -ResourceGroupName $rgname -Name $vmssname -VirtualMachineScaleSet $VMSS
```

I need to execute a custom script that's hosted in a private storage account. The script runs successfully when the storage is public, but when I try to use a Shared Access Signature (SAS), it fails. This message is displayed: "Missing mandatory parameters for valid Shared Access Signature". Link+SAS works fine from my local browser.

To execute a custom script that’s hosted in a private storage account, set up protected settings with the storage account key and name. For more information, see Custom Script Extension for Windows.

Networking
Is it possible to assign a Network Security Group (NSG) to a scale set, so that it will apply to all the VM NICs in the set?

Yes. A Network Security Group can be applied directly to a scale set by referencing it in the networkInterfaceConfigurations section of the network profile. Example:

```json
"networkProfile": {
  "networkInterfaceConfigurations": [
    {
      "name": "nic1",
      "properties": {
        "primary": "true",
        "ipConfigurations": [
          {
            "name": "ip1",
            "properties": {
              "subnet": {
                "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/virtualNetworks/', variables('vnetName'), '/subnets/subnet1')]
              }
            },
          }
        ],
        "loadBalancerInboundNatPools": [
          {
            "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/inboundNatPools/natPool1')]
          },
          {
            "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/loadBalancers/', variables('lbName'), '/backendAddressPools/addressPool1')]
          }
        ],
        "networkSecurityGroup": {
          "id": "[concat('/subscriptions/', subscription().subscriptionId,'/resourceGroups/', resourceGroup().name, '/providers/Microsoft.Network/networkSecurityGroups/', variables('nsgName'))]"
        }
      }
    }
  ]
}
```

How do I do a VIP swap for virtual machine scale sets in the same subscription and same region?

If you have two virtual machine scale sets with Azure Load Balancer front-ends, and they are in the same subscription and region, you could deallocate the public IP addresses from each one, and assign to the other. See VIP Swap: Blue-green deployment in Azure Resource Manager for example. This does imply a delay though as the resources are deallocated/allocated at the network level. A faster option is to use Azure Application Gateway with two backend pools, and a routing rule. Alternatively, you could host your application with Azure App service which provides support for fast switching between staging and production slots.

How do I specify a range of private IP addresses to use for static private IP address allocation?

IP addresses are selected from a subnet that you specify. The allocation method of virtual machine scale set IP addresses is always “dynamic,” but that doesn’t mean that these IP addresses can change. In this case, “dynamic” only means that you do not specify the IP address in a PUT request. Specify the static set by using the subnet.

How do I deploy a virtual machine scale set to an existing Azure virtual network?
To deploy a virtual machine scale set to an existing Azure virtual network, see [Deploy a virtual machine scale set to an existing virtual network](#).

**How do I add the IP address of the first VM in a virtual machine scale set to the output of a template?**

To add the IP address of the first VM in a virtual machine scale set to the output of a template, see [ARM: Get VMSS's private IPs](#).

**Can I use scale sets with Accelerated Networking?**

Yes. To use accelerated networking, set `enableAcceleratedNetworking` to true in your scale set's `networkInterfaceConfigurations` settings. E.g.

```json

"networkProfile": {
  "networkInterfaceConfigurations": [
    {
      "name": "niconfig1",
      "properties": {
        "primary": true,
        "enableAcceleratedNetworking": true,
        "ipConfigurations": [
          {
          }
        ]
    }
  ]
}

```

**How can I configure the DNS servers used by a scale set?**

To create a VM scale set with a custom DNS configuration, add a `dnsSettings` JSON packet to the scale set `networkInterfaceConfigurations` section. Example:

```json

"dnsSettings": {
  "dnsServers": ["10.0.0.6", "10.0.0.5"]
}

```

**How can I configure a scale set to assign a public IP address to each VM?**

To create a VM scale set that assigns a public IP address to each VM, make sure the API version of the Microsoft.Compute/virtualMachineScaleSets resource is 2017-03-30, and add a `publicipaddressconfiguration` JSON packet to the scale set `ipConfigurations` section. Example:

```json

"publicipaddressconfiguration": {
  "name": "pub1",
  "properties": {
    "idleTimeoutInMinutes": 15
  }
}

```

**Can I configure a scale set to work with multiple Application Gateways?**

Yes. You can add the resource id’s for multiple Application Gateway backend address pools to the `applicationGatewayBackendAddressPools` list in the `ipConfigurations` section of your scale set network profile.

**Scale**

**In what case would I create a virtual machine scale set with fewer than two VMs?**

One reason to create a virtual machine scale set with fewer than two VMs would be to use the elastic properties of
a virtual machine scale set. For example, you could deploy a virtual machine scale set with zero VMs to define your infrastructure without paying VM running costs. Then, when you are ready to deploy VMs, increase the “capacity” of the virtual machine scale set to the production instance count.

Another reason you might create a virtual machine scale set with fewer than two VMs is if you’re concerned less with availability than in using an availability set with discrete VMs. Virtual machine scale sets give you a way to work with undifferentiated compute units that are fungible. This uniformity is a key differentiator for virtual machine scale sets versus availability sets. Many stateless workloads do not track individual units. If the workload drops, you can scale down to one compute unit, and then scale up to many when the workload increases.

**How do I change the number of VMs in a virtual machine scale set?**

To change the number of VMs in a virtual machine scale set in the Azure portal, from the VM scale set properties section, click on the “Scaling” blade and use the slider bar. For other ways to change the instance count, see Change the instance count of a virtual machine scale set.

**How do I define custom alerts for when certain thresholds are reached?**

You have some flexibility in how you handle alerts for specified thresholds. For example, you can define customized webhooks. The following webhook example is from a Resource Manager template:

```json
{
  "type": "Microsoft.Insights/autoscaleSettings",
  "apiVersion": "[variables('insightsApi')]",
  "name": "autoscale",
  "location": "[parameters('resourceLocation')]",
  "dependsOn": [
    "[concat('Microsoft.Compute/virtualMachineScaleSets/', parameters('vmSSName'))]"
  ],
  "properties": {
    "name": "autoscale",
    "targetResourceUri": "[concat('/subscriptions/',subscription().subscriptionId, '/resourceGroups/',
    resourceGroup().name, '/providers/Microsoft.Compute/virtualMachineScaleSets/', parameters('vmSSName'))]",
    "enabled": true,
    "notifications": [
      {
        "operation": "Scale",
        "email": {
          "sendToSubscriptionAdministrator": true,
          "sendToSubscriptionCoAdministrators": true,
          "customEmails": [
            "youremail@address.com"
          ]
        },
        "webhooks": [
          {
            "serviceUri": "https://events.pagerduty.com/integration/0b75b57246814149b4d87fa6e1273687/enqueue",
            "properties": {
              "key1": "custommetric",
              "key2": "scalevmss"
            }
          }
        ]
      }
    ]
  }
}
```

In this example, an alert goes to Pagerduty.com when a threshold is reached.

**Patching and operations**

**How do I create a scale set in an existing resource group?**
Can we move a scale set to another resource group?
Yes, you can move scale set resources to a new subscription or resource group.

How to I update my virtual machine scale set to a new image? How do I manage patching?
To update your virtual machine scale set to a new image, and to manage patching, see Upgrade a virtual machine scale set.

Can I use the reimage operation to reset a VM without changing the image? (That is, I want reset a VM to factory settings rather than to a new image.)
Yes, you can use the reimage operation to reset a VM without changing the image. However, if your virtual machine scale set references a platform image with version = latest, your VM can update to a later OS image when you call reimage.

For more information, see Manage all VMs in a virtual machine scale set.

Is it possible to integrate scale sets with Azure OMS (Operations Management Suite)?
Yes, you can by installing the OMS extension on the scale set VMs. Here is an Azure CLI example:

```
az vmss extension set --name MicrosoftMonitoringAgent --publisher Microsoft.EnterpriseCloud.Monitoring --resource-group Team-03 --vmss-name nt01 --settings "{'workspaceId': '<your workspace ID here>'}" --protected-settings "{'workspaceKey': '<your workspace key here'}"
```

You can find the required workspaceId and workspaceKey in the OMS portal. On the Overview page, click the Settings tile. Click the Connected Sources tab at the top.

Note: if your scale set upgradePolicy is set to Manual, you will need to apply the extension to the all VMs in the set by calling upgrade on them. In CLI this would be az vmss update-instances.

Troubleshooting

How do I turn on boot diagnostics?
To turn on boot diagnostics, first, create a storage account. Then, put this JSON block in your virtual machine scale set virtualMachineProfile, and update the virtual machine scale set:

```
"diagnosticsProfile": {
  "bootDiagnostics": {
    "enabled": true,
    "storageUri": "http://yourstorageaccount.blob.core.windows.net"
  }
}
```

When a new VM is created, the InstanceView property of the VM shows the details for the screenshot, and so on. Here’s an example:
Virtual machine properties

How do I get property information for each VM without making multiple calls? For example, how would I get the fault domain for each of the 100 VMs in my virtual machine scale set?

To get property information for each VM without making multiple calls, you can call `ListVMInstanceViews` by doing a REST API `GET` on the following resource URI:

/subscriptions/<subscription_id>/resourceGroups/<resource_group_name>/providers/Microsoft.Compute/virtualMachineScaleSets/<scaleset_name>/virtualMachines?$expand=instanceView&$select=instanceView

Can I pass different extension arguments to different VMs in a virtual machine scale set?

No, you cannot pass different extension arguments to different VMs in a virtual machine scale set. However, extensions can act based on the unique properties of the VM they are running on, such as on the machine name. Extensions also can query instance metadata on `http://169.254.169.254` to get more information about the VM.

Why are there gaps between my virtual machine scale set VM machine names and VM IDs? For example: 0, 1, 3...

There are gaps between your virtual machine scale set VM machine names and VM IDs because your virtual machine scale set `overprovision` property is set to the default value of `true`. If overprovisioning is set to `true`, more VMs than requested are created. Extra VMs are then deleted. In this case, you gain increased deployment reliability, but at the expense of contiguous naming and contiguous Network Address Translation (NAT) rules.

You can set this property to `false`. For small virtual machine scale sets, this doesn't significantly affect deployment reliability.

What is the difference between deleting a VM in a virtual machine scale set and deallocating the VM? When should I choose one over the other?

The main difference between deleting a VM in a virtual machine scale set and deallocating the VM is that `deallocating` doesn’t delete the virtual hard disks (VHDS). There are storage costs associated with running `stop deallocate`. You might use one or the other for one of the following reasons:

- You want to stop paying compute costs, but you want to keep the disk state of the VMs.
- You want to start a set of VMs more quickly than you could scale out a virtual machine scale set.
  - Related to this scenario, you might have created your own autoscale engine and want a faster end-to-end scale.
- You have a virtual machine scale set that is unevenly distributed across fault domains or update domains. This might be because you selectively deleted VMs, or because VMs were deleted after overprovisioning. Running `stop deallocate` followed by `start` on the virtual machine scale set evenly distributes the VMs across fault domains or update domains.
This article provides syntax and options for Azure command-line interface (CLI) commands you’d commonly use to create and manage Azure resources in the Azure Resource Manager deployment model. You access these commands by running the CLI in Resource Manager (arm) mode. This is not a complete reference, and your CLI version may show slightly different commands or parameters. For a general overview of Azure resources and resource groups, see Azure Resource Manager Overview.

NOTE
This article shows Resource Manager mode commands in the Azure CLI, sometimes called Azure CLI 1.0. To work in the Resource Manager model, you can also try the Azure CLI 2.0, our next generation multi-platform CLI. Find out more about the old and new Azure CLIs.

To get started, first install the Azure CLI and connect to your Azure subscription.

For current command syntax and options at the command line in Resource Manager mode, type `azure help` or, to display help for a specific command, `azure help [command]`. Also find CLI examples in the documentation for creating and managing specific Azure services.

Optional parameters are shown in square brackets (for example, `[parameter]`). All other parameters are required.

In addition to command-specific optional parameters documented here, there are three optional parameters that can be used to display detailed output such as request options and status codes. The `-v` parameter provides verbose output, and the `-vv` parameter provides even more detailed verbose output. The `--json` option outputs the result in raw JSON format.

### Setting the Resource Manager mode

Use the following command to enable Azure CLI Resource Manager mode commands.

```bash
azure config mode arm
```

**NOTE**
The CLI’s Azure Resource Manager mode and Azure Service Management mode are mutually exclusive. That is, resources created in one mode cannot be managed from the other mode.

**azure account: Manage your account information**

Your Azure subscription information is used by the tool to connect to your account.

#### List the imported subscriptions

`account list [options]`

#### Show details about a subscription
account show [options] [subscriptionNameOrId]

Set the current subscription

account set [options] <subscriptionNameOrId>

Remove a subscription or environment, or clear all of the stored account and environment info

account clear [options]

Commands to manage your account environment

account env list [options]
account env show [options] [environment]
account env add [options] [environment]
account env set [options] [environment]
account env delete [options] [environment]

azure ad: Commands to display Active Directory objects

Commands to display active directory applications

ad app create [options]
ad app delete [options] <object-id>

Commands to display active directory groups

ad group list [options]
ad group show [options]

Commands to provide an active directory sub group or member info

ad group member list [options] [objectId]

Commands to display active directory service principals

ad sp list [options]
ad sp show [options]
ad sp create [options] <application-id>
ad sp delete [options] <object-id>

Commands to display active directory users

ad user list [options]
ad user show [options]

azure availset: commands to manage your availability sets

Creates an availability set within a resource group
Lists the availability sets within a resource group

```
availset list [options] <resource-group>
```

Gets one availability set within a resource group

```
availset show [options] <resource-group> <name>
```

Deletes one availability set within a resource group

```
availset delete [options] <resource-group> <name>
```

azure config: commands to manage your local settings

**List Azure CLI configuration settings**

```
config list [options]
```

Delete a config setting

```
config delete [options] <name>
```

Update a config setting

```
config set <name> <value>
```

Sets the Azure CLI working mode to either `arm` or `asm`

```
config mode [options] <modename>
```

azure feature: commands to manage account features

**List all features available for your subscription**

```
feature list [options]
```

Shows a feature

```
feature show [options] <providerName> <featureName>
```

Registers a previewed feature of a resource provider

```
feature register [options] <providerName> <featureName>
```
azure group: Commands to manage your resource groups

Creates a resource group

```
  group create [options] <name> <location>
```

Set tags to a resource group

```
  group set [options] <name> <tags>
```

Deletes a resource group

```
  group delete [options] <name>
```

Lists the resource groups for your subscription

```
  group list [options]
```

Shows a resource group for your subscription

```
  group show [options] <name>
```

Commands to manage resource group logs

```
  group log show [options] [name]
```

Commands to manage your deployment in a resource group

```
  group deployment create [options] [resource-group] [name]
  group deployment list [options] [resource-group] [state]
  group deployment show [options] [resource-group] [deployment-name]
  group deployment stop [options] [resource-group] [deployment-name]
```

Commands to manage your local or gallery resource group template

```
  group template list [options]
  group template show [options] <name>
  group template download [options] [name] [file]
  group template validate [options] <resource-group>
```

azure hdinsight: Commands to manage your HDInsight clusters

Commands to create or add to a cluster configuration file

```
  hdinsight config create [options] <configFilePath> <overwrite>
  hdinsight config add-config-values [options] <configFilePath>
  hdinsight config add-script-action [options] <configFilePath>
```

Example: Create a configuration file that contains a script action to run when creating a cluster.
Command to create a cluster in a resource group

hdinsight cluster create [options] <clusterName>

Example: Create a Storm on Linux cluster

azure hdinsight cluster create -g myarmgroup -l westus -y Linux --clusterType Storm --version 3.2 --
defaultStorageAccountName mystorageaccount --defaultStorageAccountKey <defaultStorageAccountKey> --
defaultStorageContainer mycontainer --userName admin --password <clusterPassword> --sshUserName sshuser --
sshPassword <sshPassword> --workerNodeCount 1 myNewCluster01

info: Executing command hdinsight cluster create
+ Submitting the request to create cluster...
info: hdinsight cluster create command OK

Example: Create a cluster with a script action

azure hdinsight cluster create -g myarmgroup -l westus -y Linux --clusterType Hadoop --version 3.2 --
defaultStorageAccountName mystorageaccount --defaultStorageAccountKey <defaultStorageAccountKey> --
defaultStorageContainer mycontainer --userName admin --password <clusterPassword> --sshUserName sshuser --
sshPassword <sshPassword> --workerNodeCount 1 -configurationPath "C:\myFiles\configFile.config" myNewCluster01

info: Executing command hdinsight cluster create
+ Submitting the request to create cluster...
info: hdinsight cluster create command OK

Parameter options:
-h, --help                   output usage information
-v, --verbose                use verbose output
-vv                          more verbose with debug output
--json                       use json output
-g --resource-group <resource-group>  The name of the resource group
-c --clusterName <clusterName> HDInsight cluster name
-l --location <location>     Data center location for the cluster
-y --osType <osType>        HDInsight operating system 'Windows' or 'Linux'
--version <version>         HDInsight cluster version
--clusterType <clusterType> HDInsight cluster type.
Hadoop | HBase | Spark | Storm
--defaultStorageAccountName <storageAccountName> Storage account url to use for default HDInsight storage
--defaultStorageAccountKey <storageAccountKey> Key to the storage account to use for default HDInsight storage
--defaultStorageContainer <storageContainer> Container in the storage account to use for HDInsight storage
--headNodeSize <headNodeSize> (Optional) Head node size for the cluster
--workerNodeCount <workerNodeCount> Number of worker nodes to use for the cluster
--workerNodeSize <workerNodeSize> (Optional) Worker node size for the cluster
--zookeeperNodeSize <zookeeperNodeSize> (Optional) Zookeeper node size for the cluster
--userName <userName>       Cluster username
--password <password>       Cluster password
--sshUserName <sshUserName> SSH username (only for Linux clusters)
--sshPassword <sshPassword> SSH password (only for Linux clusters)
--sshPublicKey <sshPublicKey> SSH public key (only for Linux clusters)
--rdpUserName <rdpUserName> RDP username (only for Windows clusters)
--rdpPassword <rdpPassword> RDP password (only for Windows clusters)
--rdpAccessExpiry <rdpAccessExpiry> RDP access expiry.
--virtualNetworkId <virtualNetworkId> (Optional) Virtual network ID for the cluster. Value is a GUID for Windows cluster and ARM resource ID for Linux cluster
--subnetName <subnetName>   (Optional) Subnet for the cluster
--additionalStorageAccounts <additionalStorageAccounts> (Optional) Additional storage accounts. Can be multiple.
In the format of 'accountName#accountKey'. For example, --additionalStorageAccounts "acc1#key1;acc2#key2"
--hiveMetastoreServerName <hiveMetastoreServerName> (Optional) SQL Server name for the external metastore for Hive
--hiveMetastoreDatabaseName <hiveMetastoreDatabaseName> (Optional) Database name for the external metastore for Hive
--hiveMetastoreUserName <hiveMetastoreUserName> (Optional) Database username for the external metastore for Hive
--hiveMetastorePassword <hiveMetastorePassword> (Optional) Database password for the external metastore for Hive
--oozieMetastoreServerName <oozieMetastoreServerName> (Optional) SQL Server name for the external metastore for Oozie
--oozieMetastoreDatabaseName <oozieMetastoreDatabaseName> (Optional) Database name for the external metastore for Oozie
--oozieMetastoreUserName <oozieMetastoreUserName> (Optional) Database username for the external metastore for Oozie
--oozieMetastorePassword <oozieMetastorePassword> (Optional) Database password for the external metastore for Oozie
--configurationPath <configurationPath> (Optional) HDInsight cluster configuration file path
-s, --subscription <id> Tags to set to the cluster. Can be multiple.
In the format of 'name=value'. Name is required and value is optional. For example, --tags tag1=value1;tag2

Command to delete a cluster
Command to show cluster details

   hdinsight cluster show [options] <clusterName>

Command to list all clusters (in a specific resource group, if provided)

   hdinsight cluster list [options]

Command to resize a cluster

   hdinsight cluster resize [options] <clusterName> <targetInstanceCount>

Command to enable HTTP access for a cluster

   hdinsight cluster enable-http-access [options] <clusterName> <userName> <password>

Command to disable HTTP access for a cluster

   hdinsight cluster disable-http-access [options] <clusterName>

Command to enable RDP access for a cluster

   hdinsight cluster enable-rdp-access [options] <clusterName> <rdpUserName> <rdpPassword> <rdpExpiryDate>

Command to disable HTTP access for a cluster

   hdinsight cluster disable-rdp-access [options] <clusterName>

azure insights: Commands related to monitoring Insights (events, alert rules, autoscale settings, metrics)

Retrieve operation logs for a subscription, a correlationId, a resource group, resource, or resource provider

   insights logs list [options]

azure location: Commands to get the available locations for all resource types

List the available locations

   location list [options]
azure network: Commands to manage network resources

Commands to manage virtual networks

network vnet create [options] <resource-group> <name> <location>

Creates a virtual network. In the following example we create a virtual network named newvnet for resource group myresourcegroup in the West US region.

```
azure network vnet create myresourcegroup newvnet "west us"
info:   Executing command network vnet create
+ Looking up virtual network "newvnet"
+ Creating virtual network "newvnet"
Loading virtual network state
data:   Id:
   /subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/newvnet
data:   Name:                 newvnet
data:   Type:                 Microsoft.Network/virtualNetworks
data:   Location:             westus
data:   Tags:
data:   Provisioning state:   Succeeded
data:   Address prefixes:
data:     10.0.0.0/8
data:   DNS servers:
data:   Subnets:
data:   info:    network vnet create command OK
```

Parameter options:

- `--help`: output usage information
- `--verbose`: use verbose output
- `--json`: use json output
- `--resource-group <resource-group>`: the name of the resource group
- `--name <name>`: the name of the virtual network
- `--location <location>`: the location
- `--address-prefixes <address-prefixes>`: the comma separated list of address prefixes for this virtual network
  For example `-a 10.0.0.0/24,10.0.1.0/24.
  Default value is 10.0.0.0/8
- `--dns-servers <dns-servers>`: the comma separated list of DNS servers IP addresses
- `--tags <tags>`: the tags set on this virtual network.
  Can be multiple. In the format "name=value".
  Name is required and value is optional.
  For example, -t tag1=value1;tag2
- `--subscription <subscription>`: the subscription identifier

network vnet set [options] <resource-group> <name>

Updates a virtual network configuration within a resource group.
azure network vnet set myresourcegroup newvnet

info: Executing command network vnet set
+ Looking up virtual network "newvnet"
+ Updating virtual network "newvnet"
+ Loading virtual network state

data: Id: /subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/newvnet

data: Name: newvnet

data: Type: Microsoft.Network/virtualNetworks

data: Location: westus

data: Tags: 

data: Provisioning state: Succeeded

data: Address prefixes: 10.0.0.0/8

data: DNS servers: 

data: Subnets: 

data: info: network vnet set command OK

Parameter options:

-h, --help output usage information

-v, --verbose use verbose output

--json use json output

-g, --resource-group <resource-group> the name of the resource group

-n, --name <name> the name of the virtual network

-a, --address-prefixes <address-prefixes> the comma separated list of address prefixes for this virtual network.

For example -a 10.0.0.0/24,10.0.1.0/24.
The address prefixes in this list should not overlap between them.
The address prefixes in this list should not overlap with existing address prefixes in the vnet.

-d, --dns-servers [dns-servers] the comma separated list of DNS servers IP addresses.

This list will be appended to the current list of DNS server IP addresses.

-t, --tags [tags] the tags set on this virtual network.

Can be multiple. In the format of "name=value".
Name is required and value is optional. For example, -t tag1=value1;tag2.
This list will be appended to the current list of tags

--no-tags remove all existing tags

-s, --subscription <subscription> the subscription identifier

network vnet list [options] <resource-group>

The command lists all virtual networks in a resource group.
C:\>azure network vnet list myresourcegroup

info: Executing command network vnet list
+ Listing virtual networks
data: ID
   Name Location Address prefixes DNS servers
data: -------------------------------------------------------------------
------ -------- -------- ----------------  -----------
data: /subscriptions/###############################/resourceGroups/
wvnet newvnet westus 10.0.0.0/8
info: network vnet list command OK

Parameter options:

-h, --help output usage information
-v, --verbose use verbose output
--json use json output
-g, --resource-group <resource-group> the name of the resource group
-s, --subscription <subscription> the subscription identifier

network vnet show [options] <resource-group> <name>

The command shows the virtual network properties in a resource group.

azure network vnet show -g myresourcegroup -n newvnet

info: Executing command network vnet show
+ Looking up virtual network "newvnet"
data: Id:
   /subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/newvnet
data: Name: newvnet
data: Type: Microsoft.Network/virtualNetworks
data: Location: westus
data: Tags:
data: Provisioning state: Succeeded
data: Address prefixes:
data: 10.0.0.0/8
data: DNS servers:
data: Subnets:
data: info: network vnet show command OK

network vnet delete [options] <resource-group> <name>

The command removes a virtual network.

azure network vnet delete myresourcegroup newvnetX

info: Executing command network vnet delete
+ Looking up virtual network "newvnetX"
Delete virtual network newvnetX? [y/n] y
+ Deleting virtual network "newvnetX"
info: network vnet delete command OK
Parameter options:

- `--help`  
  output usage information
- `--verbose`  
  use verbose output
- `--json`  
  use json output
- `--resource-group <resource-group>`  
  the name of the resource group
- `--name <name>`  
  the name of the virtual network
- `--quiet`  
  quiet mode, do not ask for delete confirmation
- `--subscription <subscription>`  
  the subscription identifier

Commands to manage virtual network subnets

### network vnet subnet create [options] <resource-group> <vnet-name> <name>

Adds another subnet to an existing virtual network.

```bash
azure network vnet subnet create -g myresourcegroup --vnet-name newvnet -n subnet --address-prefix 10.0.1.0/24
```

Parameter options:

- `--help`  
  output usage information
- `--verbose`  
  use verbose output
- `--json`  
  use json output
- `--resource-group <resource-group>`  
  the name of the resource group
- `--vnet-name <vnet-name>`  
  the name of the virtual network
- `--name <name>`  
  the name of the subnet
- `--address-prefix <address-prefix>`  
  the address prefix
- `--network-security-group-id <network-security-group-id>`  
  the network security group identifier.  
  e.g. /subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Network/networkSecurityGroups/<nsg-name>
- `--network-security-group-name <network-security-group-name>`  
  the network security group name
- `--subscription <subscription>`  
  the subscription identifier

### network vnet subnet set [options] <resource-group> <vnet-name> <name>

Sets a specific virtual network subnet within a resource group.
C:\>azure network vnet subnet set -g myresourcegroup --vnet-name newvnet -n subnet1

info: Executing command network vnet subnet set
+ Looking up the subnet "subnet1"
+ Setting subnet "subnet1"
+ Looking up the subnet "subnet1"
data: Id:
/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/newvnet/subnets/subnet1
data: Name: subnet1
data: Type: Microsoft.Network/virtualNetworks/subnets
data: Provisioning state: Succeeded
data: Address prefix: 10.0.1.0/24
info: network vnet subnet set command OK

network vnet subnet list [options] <resource-group> <vnet-name>

Lists all the virtual network subnets for a specific virtual network within a resource group.

azure network vnet subnet set -g myresourcegroup --vnet-name newvnet -n subnet1

info: Executing command network vnet subnet set
+ Looking up the subnet "subnet1"
+ Setting subnet "subnet1"
+ Looking up the subnet "subnet1"
data: Id:
/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/newvnet/subnets/subnet1
data: Name: subnet1
data: Type: Microsoft.Network/virtualNetworks/subnets
data: Provisioning state: Succeeded
data: Address prefix: 10.0.1.0/24
info: network vnet subnet set command OK

network vnet subnet show [options] <resource-group> <vnet-name> <name>

Displays virtual network subnet properties

azure network vnet subnet show -g myresourcegroup --vnet-name newvnet -n subnet1

info: Executing command network vnet subnet show
+ Looking up the subnet "subnet1"
data: Id:
/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/newvnet/subnets/subnet1
data: Name: subnet1
data: Type: Microsoft.Network/virtualNetworks/subnets
data: Provisioning state: Succeeded
data: Address prefix: 10.0.1.0/24
info: network vnet subnet show command OK

Parameter options:
Removes a subnet from an existing virtual network.

```
azure network vnet subnet delete -g myresourcegroup --vnet-name newvnet -n subnet1
```

Info: Executing command network vnet subnet delete
+ Looking up the subnet "subnet1"
Delete subnet "subnet1"? [y/n] y
+ Deleting subnet "subnet1"
Info: network vnet subnet delete command OK

Parameter options:

```
-network vnet subnet delete [options] <resource-group> <vnet-name> <subnet-name>
```

Commands to manage load balancers

```
network lb create [options] <resource-group> <name> <location>
```

Creates a load balancer set.

```
azure network lb create -g myresourcegroup -n mylb -l westus
```

Info: Executing command network lb create
+ Looking up the load balancer "mylb"
+ Creating load balancer "mylb"
+ Looking up the load balancer "mylb"
Data: Id:
/subscriptions/xxxxxxxxxxxxxxxxxxxxxxxxx/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb
Data: Name: mylb
Data: Type: Microsoft.Network/loadBalancers
Data: Location: westus
Data: Provisioning state: Succeeded
Info: network lb create command OK

Parameter options:
network lb list [options] <resource-group>

Lists Load balancer resources within a resource group.

azure network lb list myresourcegroup

info:    Executing command network lb list
+ Getting the load balancers
data:    Name  Location
data:    ----  --------
data:    mylb  westus
info:    network lb list command OK

Parameter options:

-network lb list [options] <resource-group> <name>

Displays load balancer information of a specific load balancer within a resource group

azure network lb show myresourcegroup mylb -v

info:    Executing command network lb show
verbose:    Looking up the load balancer "mylb"
data:    Id:
  /subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb
data:    Name:                         mylb
data:    Type:                         Microsoft.Network/loadBalancers
data:    Location:                     westus
data:    Provisioning state:           Succeeded
info:    network lb show command OK

Parameter options:
network lb delete [options] <resource-group> <name>

Delete load balancer resources.

```
azure network lb delete myresourcegroup mylb
```

```
info: Executing command network lb delete
+ Looking up the load balancer "mylb"
Delete load balancer "mylb"? [y/n] y
+ Deleting load balancer "mylb"
info: network lb delete command OK
```

Parameter options:

 Commands to manage probes of a load balancer

```
network lb probe create [options] <resource-group> <lb-name> <name>
```

Create the probe configuration for health status in the load balancer. Keep in mind to run this command, your load balancer requires a frontend-ip resource (Check out command "azure network frontend-ip" to assign an ip address to load balancer).

```
azure network lb probe create -g myresourcegroup --lb-name mylb -n mylbprobe --protocol tcp --port 80 --i 300
```

```
info: Executing command network lb probe create
+ Looking up the load balancer "mylb"
+ Updating load balancer "mylb"
info: network lb probe create command OK
```

Parameter options:
network lb probe set [options] <resource-group> <lb-name> <name>

Updates an existing load balancer probe with new values for it.

```bash
> azure network lb probe set -g myresourcegroup -l mylb -n mylbprobe -p mylbprobe1 -p TCP -o 443 -i 300

info:    Executing command network lb probe set
+ Looking up the load balancer "mylb"
+ Updating load balancer "mylb"
info:    network lb probe set command OK
```

Parameter options

- **-h, --help**
  output usage information
- **-v, --verbose**
  use verbose output
- **--json**
  use json output
- **-g, --resource-group <resource-group>**
  the name of the resource group
- **-l, --lb-name <lb-name>**
  the name of the load balancer
- **-n, --name <name>**
  the name of the probe
- **-p, --protocol <protocol>**
  the probe protocol
- **-o, --port <port>**
  the probe port
- **-f, --path <path>**
  the probe path
- **-i, --interval <interval>**
  the probe interval in seconds
- **-c, --count <count>**
  the number of probes
- **-s, --subscription <subscription>**
  the subscription identifier

network lb probe list [options] <resource-group> <lb-name>

List the probe properties for a load balancer set.

```bash
> C:\> azure network lb probe list -g myresourcegroup -l mylb

info:    Executing command network lb probe list
+ Looking up the load balancer "mylb"
data:    Name       Protocol  Port  Path  Interval  Count
data:    ---------  --------  ----  ----  --------  -----  
data:    mylbprobe  Tcp       443         300       2
info:    network lb probe list command OK
```

Parameter options:
network lb probe delete [options] <resource-group> <lb-name> <name>

Removes the probe created for the load balancer.

```
azure network lb probe delete -g myresourcegroup -l mylb -n mylbprobe
```

info: Executing command network lb probe delete
+ Looking up the load balancer "mylb"
Delete a probe "mylbprobe?" [y/n] y
+ Updating load balancer "mylb"
info: network lb probe delete command OK

Commands to manage frontend ip configurations of a load balancer

network lb frontend-ip create [options] <resource-group> <lb-name> <name>

Creates a frontend IP configuration to an existing load balancer set.

```
azure network lb frontend-ip create -g myresourcegroup --lb-name mylb -n myfrontendip -o Dynamic -e subnet -m newvnet
```

info: Executing command network lb frontend-ip create
+ Looking up the load balancer "mylb"
+ Looking up the subnet "subnet"
+ Creating frontend IP configuration "myfrontendip"
+ Looking up the load balancer "mylb"
data: Id:
/subscriptions/################################/resourceGroups/Myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb
/frontendIPConfigurations/myfrontendip
data: Name: myfrontendip
data: Type: Microsoft.Network/loadBalancers/frontendIPConfigurations
data: Provisioning state: Succeeded
data: Private IP allocation method: Dynamic
data: Private IP address: 10.0.1.4
data: Subnet:
id=/subscriptions/################################/resourceGroups/myresourcegroup/providers/Microsoft.Network/virtualNetworks/newvnet/subnets/subnet
data: Public IP address:
data: Inbound NAT rules
data: Outbound NAT rules
data: Load balancing rules
data:
info: network lb frontend-ip create command OK

network lb frontend-ip set [options] <resource-group> <lb-name> <name>

Updates an existing configuration of a frontend IP. The command below adds a public IP called mypubip5 to an existing load balancer frontend IP named myfrontendip.
azure network lb frontend-ip set -g myresourcegroup --lb-name mylb -n myfrontendip -i mypubip5

info: Executing command network lb frontend-ip set
+ Looking up the load balancer "mylb"
+ Looking up the public ip "mypubip5"
+ Updating load balancer "mylb"
+ Looking up the load balancer "mylb"

data: Id:
/subscriptions/###################################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb/frontendIPConfigurations/myfrontendip

   data: Name: myfrontendip
   data: Type: Microsoft.Network/loadBalancers/frontendIPConfigurations
   data: Provisioning state: Succeeded
   data: Private IP allocation method: Dynamic
   data: Private IP address: 
   data: Subnet: 
   data: Public IP address: 
/subscriptions/###################################/resourceGroups/myresourcegroup/providers/Microsoft.Network/publicIPAddresses/mypubip5

   data: Inbound NAT rules
   data: Outbound NAT rules
   data: Load balancing rules
   data: 

info: network lb frontend-ip set command OK

Parameter options:

- -h, --help         output usage information
- -v, --verbose      use verbose output
- --json             use json output
- -g, --resource-group <resource-group>  the name of the resource group
- -l, --lb-name <lb-name>  the name of the load balancer
- -n, --name <name>     the name of the frontend ip configuration
- -a, --private-ip-address <private-ip-address>  the private ip address
- -o, --private-ip-allocation-method <private-ip-allocation-method>  the private ip allocation method [Static, Dynamic]
- -u, --public-ip-id <public-ip-id>  the public ip identifier.
  e.g. /subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Network/publicIPAddresses/<public-ip-name>
- -i, --public-ip-name <public-ip-name>  the public ip name.
  This public ip must exist in the same resource group as the lb.
  Please use public-ip-id if that is not the case.
- -b, --subnet-id <subnet-id>  the subnet id.
  e.g. /subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Network/VirtualNetworks/<vnet-name>/subnets/<subnet-name>
- -e, --subnet-name <subnet-name>  the subnet name
- -m, --vnet-name <vnet-name>  the virtual network name.
  This virtual network must exist in the same resource group as the lb.
  Please use subnet-id if that is not the case.
- -s, --subscription <subscription>  the subscription identifier

network lb frontend-ip list [options] <resource-group> <lb-name>

Lists all the frontend IP resources configured for the load balancer.
azure network lb frontend-ip list -g myresourcegroup -l mylb

info: Executing command network lb frontend-ip list
+ Looking up the load balancer "mylb"
data: Name         Provisioning state  Private IP allocation method  Subnet
data: -----------  ------------------  ----------------------------  ------
data: myprivateip  Succeeded           Dynamic
info: network lb frontend-ip list command OK

Parameter options:

-h, --help                             output usage information
-v, --verbose                          use verbose output
--json                                 use json output
-g, --resource-group <resource-group>  the name of the resource group
-l, --lb-name <lb-name>                the name of the load balancer
-s, --subscription <subscription>      the subscription identifier

network lb frontend-ip list [options] <resource-group> <lb-name> <name>

Deletes the frontend IP object associated to load balancer

network lb frontend-ip delete -g myresourcegroup -l mylb -n myfrontendip
info: Executing command network lb frontend-ip delete
+ Looking up the load balancer "mylb"
Delete frontend ip configuration "myfrontendip"? [y/n] y
+ Updating load balancer "mylb"

Parameter options:

-h, --help                             output usage information
-v, --verbose                          use verbose output
--json                                 use json output
-g, --resource-group <resource-group>  the name of the resource group
-l, --lb-name <lb-name>                the name of the load balancer
-n, --name <name>                      the name of the frontend ip configuration
-q, --quiet                            quiet mode, do not ask for delete confirmation
-s, --subscription <subscription>      the subscription identifier

Commands to manage backend address pools of a load balancer

network lb address-pool create [options] <resource-group> <lb-name> <name>

Create a backend address pool for a load balancer.
azure network lb address-pool create -g myresourcegroup --lb-name mylb -n myaddresspool

info: Executing command network lb address-pool create
+ Looking up the load balancer "mylb"
+ Updating load balancer "mylb"
+ Looking up the load balancer "mylb"
data:  Id: /subscriptions/###############################/resourceGroups/myresourgroup/providers/Microsoft.Network/loadBalancers/mylb/backendAddressPools/myaddresspool
data:  Name:                      myaddresspool
data:  Type:                      Microsoft.Network/loadBalancers/backendAddressPools
data:  Provisioning state:        Succeeded
data:  Backend IP configurations:
data:  Load balancing rules:
data:
info: network lb address-pool create command OK

Parameter options:
-h, --help                         output usage information
-v, --verbose                      use verbose output
--json                             use json output
-g, --resource-group <resource-group>  the name of the resource group
-l, --lb-name <lb-name>            the name of the load balancer
-n, --name <name>                  the name of the backend address pool
-s, --subscription <subscription>  the subscription identifier

network lb address-pool list [options] <resource-group> <lb-name>

List backend IP address pool range for a specific resource group

azure network lb address-pool list -g myresourcegroup -l mylb

info: Executing command network lb address-pool list
+ Looking up the load balancer "mylb"
data:  Name  Provisioning state
   mybackendpool  Succeeded
info: network lb address-pool list command OK

Parameter options:
-h, --help                         output usage information
-v, --verbose                      use verbose output
--json                             use json output
-g, --resource-group <resource-group>  the name of the resource group
-l, --lb-name <lb-name>            the name of the load balancer
-s, --subscription <subscription>  the subscription identifier

network lb address-pool delete [options]

Removes the backend IP pool range resource from load balancer.
azure network lb address-pool delete -g myresourcegroup -l mylb -n mybackendpool

info: Executing command network lb address-pool delete
+ Looking up the load balancer "mylb"
Delete backend address pool "mybackendpool"? [y/n] y
+ Updating load balancer "mylb"
info: network lb address-pool delete command OK

Parameter options:

- `-h, --help` output usage information
- `-v, --verbose` use verbose output
- `--json` use json output
- `-g, --resource-group <resource-group>` the name of the resource group
- `-l, --lb-name <lb-name>` the name of the load balancer
- `-n, --name <name>` the name of the backend address pool
- `-q, --quiet` quiet mode, do not ask for delete confirmation
- `-s, --subscription <subscription>` the subscription identifier

Commands to manage load balancer rules

```
network lb rule create [options] <resource-group> <lb-name> <name>
```

Create load balancer rules.

You can create a load balancer rule configuring the frontend endpoint for the load balancer and the backend address pool range to receive the incoming network traffic. Settings also include the ports for frontend IP endpoint and ports for the backend address pool range.

The following example shows how to create a load balancer rule, the frontend endpoint listening to port 80 TCP and load balancing network traffic sending to port 8080 for the backend address pool range.

```
azure network lb rule create -g myresourcegroup -l mylb -n mylbrule -p tcp -f 80 -b 8080 -i 10
```

info: Executing command network lb rule create
+ Looking up the load balancer "mylb"
+ Updating load balancer "mylb"
+ Loading rule state
data: Id:
/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb/loadBalancingRules/mylbrule
data: Name: mylbrule
data: Type: Microsoft.Network/loadBalancers/loadBalancingRules
data: Provisioning state: Succeeded
data: Frontend IP configuration:
/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb/frontendIPConfigurations/myfrontendip
data: Backend address pool:
id=/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb/backendAddressPools/mybackendpool
data: Protocol: Tcp
data: Frontend port: 80
data: Backend port: 8080
data: Enable floating IP: false
data: Idle timeout in minutes: 10
data: Probes
data:
info: network lb rule create command OK
Updates an existing load balancer rule set in a specific resource group. In the following example, we changed the rule name from mylbrule to mynewlbrule.

```
azure network lb rule set -g myresourcegroup -l mylb -n mylbrule -r mynewlbrule -p tcp -f 80 -b 8080 -i 10 -t myfrontendip -o mybackendpool
```

Parameter options:

- **-h, --help**  
  output usage information
- **-v, --verbose**  
  use verbose output
- **--json**  
  use json output
- **-g, --resource-group <resource-group>**  
  the name of the resource group
- **-l, --lb-name <lb-name>**  
  the name of the load balancer
- **-n, --name <name>**  
  the name of the rule
- **-r, --new-rule-name <new-rule-name>**  
  new rule name
- **-p, --protocol <protocol>**  
  the rule protocol
- **-f, --frontend-port <frontend-port>**  
  the frontend port
- **-b, --backend-port <backend-port>**  
  the backend port
- **-e, --enable-floating-ip <enable-floating-ip>**  
  enable floating point ip
- **-i, --idle-timeout <idle-timeout>**  
  the idle timeout in minutes
- **-a, --probe-name [probe-name]**  
  the name of the probe defined in the same load balancer
- **-t, --frontend-ip-name [frontend-ip-name]**  
  the name of the frontend ip configuration in the same load balancer
- **-a, --backend-address-pool <backend-address-pool>**  
  name of the backend address pool defined in the same load balancer
- **-s, --subscription <subscription>**  
  the subscription identifier

```
network lb rule list [options] <resource-group> <lb-name>
```

Lists all load balancer rules configured for a load balancer in a specific resource group.
azure network lb rule list -g myresourcegroup -l mylb

**info:** Executing command network lb rule list
+ Looking up the load balancer "mylb"

**data:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Provisioning state</th>
<th>Protocol</th>
<th>Frontend port</th>
<th>Backend port</th>
<th>Enable floating IP</th>
<th>Idle timeout in minutes</th>
<th>Backend address pool</th>
<th>Probe data</th>
</tr>
</thead>
<tbody>
<tr>
<td>mynewlbrule</td>
<td>Succeeded</td>
<td>Tcp</td>
<td>80</td>
<td>8080</td>
<td>false</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb/backendAddressPools/mybackendpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**info:** network lb rule list command OK

### Parameter options:

- **-h, --help**
  output usage information
- **-v, --verbose**
  use verbose output
- **--json**
  use json output
- **-g, --resource-group <resource-group>**
  the name of the resource group
- **-l, --lb-name <lb-name>**
  the name of the load balancer
- **-n, --name <name>**
  the name of the rule
- **-q, --quiet**
  quiet mode, do not ask for delete confirmation
- **-s, --subscription <subscription>**
  the subscription identifier

### deletes a load balancer rule.

azure network lb rule delete -g myresourcegroup -l mylb -n mynewlbrule

**info:** Executing command network lb rule delete
+ Looking up the load balancer "mylb"
Delete load balancing rule mynewlbrule? [y/n] y
+ Updating load balancer "mylb"
**info:** network lb rule delete command OK

### Parameter options:

- **-h, --help**
  output usage information
- **-v, --verbose**
  use verbose output
- **--json**
  use json output
- **-g, --resource-group <resource-group>**
  the name of the resource group
- **-l, --lb-name <lb-name>**
  the name of the load balancer
- **-n, --name <name>**
  the name of the rule
- **-q, --quiet**
  quiet mode, do not ask for delete confirmation
- **-s, --subscription <subscription>**
  the subscription identifier

### commands to manage load balancer inbound NAT rules

network lb inbound-nat-rule create [options] <resource-group> <lb-name> <name>

Creates an inbound NAT rule for load balancer.

In the following example we created a NAT rule from frontend IP (which was previously defined using the "azure network frontend-ip" command) with an inbound listening port and outbound port that the load balancer uses to send the network traffic.
azure network lb inbound-nat-rule create -g myresourcegroup -l mylb -n myinboundnat -p tcp -f 80 -b 8080 -i myfrontendip

info: Executing command network lb inbound-nat-rule create
+ Looking up the load balancer "mylb"
+ Updating load balancer "mylb"
+ Looking up the load balancer "mylb"
data: Id:/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb/inboundNatRules/myinboundnat
data: Name: myinboundnat
data: Type: Microsoft.Network/loadBalancers/inboundNatRules
data: Provisioning state: Succeeded
data: Frontend IP Configuration:
  id=/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/loadBalancers/mylb/frontendIPConfigurations/myfrontendip
data: Backend IP configuration
data: Protocol: Tcp
data: Frontend port: 80
data: Backend port: 8080
info: Enable floating IP: false
info: network lb inbound-nat-rule create command OK

Parameter options:

-h, --help
-v, --verbose
--json
--resource-group <resource-group>
--lb-name <lb-name>
--name <name>
--protocol <protocol>
--frontend-port <frontend-port>
--backend-port <backend-port>
--enable-floating-ip <enable-floating-ip>
--frontend-ip <frontend-ip>
--vm-id <vm-id>
--vm-name <vm-name>
--subscription <subscription>

Updates an existing inbound nat rule. In the following example, we changed the inbound listening port from 80 to 81.
azure network lb inbound-nat-rule set -g group-1 -l mylb -n myinboundnat -p tcp -f 81 -b 8080 -i myfrontendip

info: Executing command network lb inbound-nat-rule set
+ Looking up the load balancer "mylb"
+ Updating load balancer "mylb"
+ Looking up the load balancer "mylb"

data: Id: /subscriptions/###############################/resourceGroups/group-1/providers/Microsoft.Network/loadBalancers/mylb/inboundNatRules/myinboundnat

data: Name: myinboundnat

data: Type: Microsoft.Network/loadBalancers/inboundNatRules

data: Provisioning state: Succeeded

data: Frontend IP Configuration: id=/subscriptions/###############################/resourceGroups/group-1/providers/Microsoft.Network/loadBalancers/mylb/frontendIPConfigurations/myfrontendip

data: Backend IP configuration

data: Protocol: Tcp

data: Frontend port: 81

data: Backend port: 8080

data: Enable floating IP: false

info: network lb inbound-nat-rule set command OK

Parameter options:

```
-h, --help  output usage information
-v, --verbose use verbose output
--json use json output
-g, --resource-group <resource-group> the name of the resource group
-l, --lb-name <lb-name> the name of the load balancer
-n, --name <name> the name of the inbound NAT rule
-p, --protocol <protocol> the rule protocol [tcp,udp]
-f, --frontend-port <frontend-port> the frontend port [0-65535]
-b, --backend-port <backend-port> the backend port [0-65535]
-e, --enable-floating-ip <enable-floating-ip> enable floating point ip [true,false]
-i, --frontend-ip <frontend-ip> the name of the frontend ip configuration
-m, --vm-id [vm-id] the VM id.

-vm-name <vm-name> the VM name.
This virtual machine must exist in the same resource group as the lb.
Please use vm-id if that is not the case
-s, --subscription <subscription> the subscription identifier
```

network lb inbound-nat-rule list [options] <resource-group> <lb-name>

Lists all inbound nat rules for load balancer.

azure network lb inbound-nat-rule list -g myresourcegroup -l mylb

info: Executing command network lb inbound-nat-rule list
+ Looking up the load balancer "mylb"

data: Name Provisioning state Protocol Frontend port Backend port Enable floating IP Idle timeout in minutes Backend IP configuration

data: myinboundnat Succeeded Tcp 81 8080 false 4

info: network lb inbound-nat-rule list command OK

Parameter options:
network lb inbound-nat-rule delete [options] <resource-group> <lb-name> <name>

Deletes NAT rule for the load balancer in a specific resource group.

azure network lb inbound-nat-rule delete -g myresourcegroup -l mylb -n myinboundnat

info: Executing command network lb inbound-nat-rule delete
+ Looking up the load balancer "mylb"
Delete inbound NAT rule "myinboundnat"? [y/n] y
+ Updating load balancer "mylb"
info: network lb inbound-nat-rule delete command OK

Parameter options:

- **-h, --help** output usage information
- **-v, --verbose** use verbose output
- **--json** use json output
- **-g, --resource-group <resource-group>** the name of the resource group
- **-l, --lb-name <lb-name>** the name of the load balancer
- **-n, --name <name>** the name of the inbound NAT rule
- **-q, --quiet** quiet mode, do not ask for delete confirmation
- **-s, --subscription <subscription>** the subscription identifier

Commands to manage public ip addresses

network public-ip create [options] <resource-group> <name> <location>

Creates a public ip resource. You will create the public ip resource and associate to a domain name.

azure network public-ip create -g myresourcegroup -n mytestpublicip1 -l eastus -d azureclitest -a "Dynamic"

info: Executing command network public-ip create
+ Looking up the public ip "mytestpublicip1"
+ Creating public ip address "mytestpublicip1"
+ Looking up the public ip "mytestpublicip1"
data: Id:
/subscriptions/####################################resourceGroups/myresourcegroup/providers/Microsoft.Network/publicIPAddresses/mytestpublicip1
data: Name: mytestpublicip1
data: Type: Microsoft.Network/publicIPAddresses
data: Location: eastus
data: Provisioning state: Succeeded
data: Allocation method: Dynamic
data: Idle timeout: 4
data: Domain name label: azureclitest
data: FQDN: azureclitest.eastus.cloudapp.azure.com
info: network public-ip create command OK

Parameter options:
Updates the properties of an existing public IP resource. In the following example we changed the public IP address from Dynamic to Static.

```
azure network public-ip set -g group-1 -n mytestpublicip1 -d azureclitest -a "Static"
```

Parameter options:

- `-h`, `--help` output usage information
- `-v`, `--verbose` use verbose output
- `--json` use json output
- `--resource-group <resource-group>` the name of the resource group
- `--name <name>` the name of the public IP
- `--location <location>` the location
- `--domain-name-label <domain-name-label>` the domain name label.

This set DNS to `<domain-name-label>.<location>.cloudapp.azure.com`

- `-a`, `--allocation-method <allocation-method>` the allocation method [Static][Dynamic]
- `-i`, `--idletimeout <idletimeout>` the idle timeout in minutes
- `-f`, `--reverse-fqdn <reverse-fqdn>` the reverse fqdn
- `-t`, `--tags <tags>` the list of tags.

Can be multiple. In the format of "name=value".
Name is required and value is optional.
For example, `-t tag1=value1;tag2`

```
-s, --subscription <subscription> the subscription identifier
```
network public-ip list [options] Lists all public IP resources within a resource group.

```bash
azure network public-ip list -g myresourcegroup

info: Executing command network public-ip list
+ Getting the public ip addresses
```

```
data: Name          Location  Allocation  IP Address  Idle timeout  DNS Name
----------  --------  ----------  ------------  ------------  -----------------------------------
--------
data: mypubip5      westus    Dynamic                   4             "domain
name".westus.cloudapp.azure.com

data: myPublicIP    eastus    Dynamic                   4             "domain
name".eastus.cloudapp.azure.com

data: mytestpublicip eastus    Dynamic                   4             "domain
name".eastus.cloudapp.azure.com

data: mytestpublicip1 eastus   Static (Static IP address) 4
azureclitest.eastus.cloudapp.azure.com

Parameter options:

```
-h, --help                             output usage information
-v, --verbose                          use verbose output
--json                                 use json output
-g, --resource-group <resource-group>  the name of the resource group
-s, --subscription <subscription>      the subscription identifier
```

network public-ip show [options]

Displays public ip properties for a public ip resource within a resource group.

```bash
azure network public-ip show -g myresourcegroup -n mytestpublicip

info: Executing command network public-ip show
+ Looking up the public ip "mytestpublicip1"
```

```
data: Id:
/subscriptions/###############################/resourceGroups/myresourcegroup/providers/Microsoft.Network/publicIPAddresses/mytestpublicip

data: Name:                  mytestpublicip

data: Type:                  Microsoft.Network/publicIPAddresses

data: Location:              eastus

data: Provisioning state:    Succeeded

data: Allocation method:     Static

data: Idle timeout:          4

data: IP Address:            (static IP address)

data: Domain name label:     azureclitest

data: FQDN:                  azureclitest.eastus.cloudapp.azure.com

info: network public-ip show command OK
```

Parameter options:
network public-ip delete [options] <resource-group> <name>

Deletes public ip resource.

```
 azure network public-ip delete -g group-1 -n mypublicipname
 info: Executing command network public-ip delete
 + Looking up the public ip "mypublicipname"
 Delete public ip address "mypublicipname"? [y/n] y
 + Deleting public ip address "mypublicipname"
 info: network public-ip delete command OK
```

Parameter options:

- `-h, --help` output usage information
- `-v, --verbose` use verbose output
- `--json` use json output
- `-g, --resource-group <resource-group>` the name of the resource group
- `-n, --name <name>` the name of the public IP
- `-s, --subscription <subscription>` the subscription identifier

Commands to manage network interfaces

network nic create [options] <resource-group> <name> <location>

Creates a resource called network interface (NIC) which can be used for load balancers or associate to a Virtual Machine.

```
 azure network nic create -g myresourcegroup -l eastus -n testnic1 --subnet-name subnet-1 --subnet-vnet-name myvnet
 info: Executing command network nic create
 + Looking up the network interface "testnic1"
 + Looking up the subnet "subnet-1"
 + Creating network interface "testnic1"
 + Looking up the network interface "testnic1"
 data: Id:                     /subscriptions/c4a17ddf-aa84-491c-b6f9-b90d882299f7/resourceGroups/group-1/providers/Microsoft.Network/networkInterfaces/testnic1
 data: Name:                   testnic1
 data: Type:                   Microsoft.Network/networkInterfaces
 data: Location:               eastus
 data: Provisioning state:     Succeeded
 data: IP configurations:
 data:     Name:                         NIC-config
 data:     Provisioning state:           Succeeded
 data:     Private IP address:           10.0.0.5
 data:     Private IP Allocation Method: Dynamic
 data:     Subnet:                       /subscriptions/c4a17ddf-aa84-491c-b6f9-b90d882299f7/resourceGroups/group-1/providers/Microsoft.Network/virtualNetworks/myVNET/subnets/Subnet-1
```

Parameter options:
network nic set [options] <resource-group> <name>

network nic list [options] <resource-group>
network nic show [options] <resource-group> <name>
network nic delete [options] <resource-group> <name>

**Commands to manage network security groups**

network nsg create [options] <resource-group> <name> <location>

network nsg set [options] <resource-group> <name>

network nsg list [options] <resource-group>

network nsg show [options] <resource-group> <name>

network nsg delete [options] <resource-group> <name>

**Commands to manage network security group rules**

network nsg rule create [options] <resource-group> <nsg-name> <name>

network nsg rule set [options] <resource-group> <nsg-name> <name>

network nsg rule list [options] <resource-group> <nsg-name>

network nsg rule show [options] <resource-group> <nsg-name> <name>

network nsg rule delete [options] <resource-group> <nsg-name> <name>

**Commands to manage traffic manager profile**
network traffic-manager profile create [options] <resource-group> <name>
network traffic-manager profile set [options] <resource-group> <name>
network traffic-manager profile list [options] <resource-group>
network traffic-manager profile show [options] <resource-group> <name>
network traffic-manager profile delete [options] <resource-group> <name>
network traffic-manager profile is-dns-available [options] <resource-group> <relative-dns-name>

Commands to manage traffic manager endpoints

network traffic-manager profile endpoint create [options] <resource-group> <profile-name> <name> <endpoint-location>
network traffic-manager profile endpoint set [options] <resource-group> <profile-name> <name>
network traffic-manager profile endpoint delete [options] <resource-group> <profile-name> <name>

Commands to manage virtual network gateways

network gateway list [options] <resource-group>

azure provider: Commands to manage resource provider registrations

List currently registered providers in Resource Manager

provider list [options]

Show details about the requested provider namespace

provider show [options] <namespace>

Register provider with the subscription

provider register [options] <namespace>

Unregister provider with the subscription

provider unregister [options] <namespace>

azure resource: Commands to manage your resources

Creates a resource in a resource group

resource create [options] <resource-group> <name> <resource-type> <location> <api-version>

Updates a resource in a resource group without any templates or parameters

resource set [options] <resource-group> <name> <resource-type> <properties> <api-version>

Lists the resources
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource list [options] [resource-group]</td>
<td>Gets one resource within a resource group or subscription</td>
</tr>
<tr>
<td>resource show [options] &lt;resource-group&gt; &lt;name&gt; &lt;resource-type&gt; &lt;api-version&gt;</td>
<td></td>
</tr>
<tr>
<td>resource delete [options] &lt;resource-group&gt; &lt;name&gt; &lt;resource-type&gt; &lt;api-version&gt;</td>
<td>Deletes a resource in a resource group</td>
</tr>
</tbody>
</table>

**azure role: Commands to manage your Azure roles**

**Get all available role definitions**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role list [options]</td>
<td></td>
</tr>
</tbody>
</table>

**Get an available role definition**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role show [options] [name]</td>
<td></td>
</tr>
</tbody>
</table>

**Commands to manage your role assignment**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role assignment create [options] [objectId] [upn] [mail] [spn] [role] [scope] [resource-group] [resource-type] [resource-name]</td>
<td></td>
</tr>
<tr>
<td>role assignment list [options] [objectId] [upn] [mail] [spn] [role] [scope] [resource-group] [resource-type] [resource-name]</td>
<td></td>
</tr>
<tr>
<td>role assignment delete [options] [objectId] [upn] [mail] [spn] [role] [scope] [resource-group] [resource-type] [resource-name]</td>
<td></td>
</tr>
</tbody>
</table>

**azure storage: Commands to manage your Storage objects**

**Commands to manage your Storage accounts**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storage account list [options]</td>
<td></td>
</tr>
<tr>
<td>storage account show [options] &lt;name&gt;</td>
<td></td>
</tr>
<tr>
<td>storage account create [options] &lt;name&gt;</td>
<td></td>
</tr>
<tr>
<td>storage account set [options] &lt;name&gt;</td>
<td></td>
</tr>
<tr>
<td>storage account delete [options] &lt;name&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**Commands to manage your Storage account keys**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storage account keys list [options] &lt;name&gt;</td>
<td></td>
</tr>
<tr>
<td>storage account keys renew [options] &lt;name&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**Commands to show your Storage connection string**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storage account connectionstring show [options] &lt;name&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**Commands to manage your Storage containers**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
Commands to manage shared access signatures of your Storage container

```
storage container sas create [options] [container] [permissions] [expiry]
```

Commands to manage stored access policies of your Storage container

```
storage container policy create [options] [container] [name]
storage container policy show [options] [container] [name]
storage container policy list [options] [container]
storage container policy set [options] [container] [name]
storage container policy delete [options] [container] [name]
```

Commands to manage your Storage blobs

```
storage blob list [options] [container] [prefix]
storage blob show [options] [container] [blob]
storage blob delete [options] [container] [blob]
storage blob upload [options] [file] [container] [blob]
storage blob download [options] [container] [blob] [destination]
```

Commands to manage your blob copy operations

```
storage blob copy start [options] [sourceUri] [destContainer]
storage blob copy show [options] [container] [blob]
storage blob copy stop [options] [container] [blob] [copyid]
```

Commands to manage shared access signature of your Storage blob

```
storage blob sas create [options] [container] [blob] [permissions] [expiry]
```

Commands to manage your Storage file shares

```
storage share create [options] [share]
storage share show [options] [share]
storage share delete [options] [share]
storage share list [options] [prefix]
```

Commands to manage your Storage files

```
storage file list [options] [share] [path]
storage file delete [options] [share] [path]
storage file upload [options] [source] [share] [path]
storage file download [options] [share] [path] [destination]
```

Commands to manage your Storage file directory
Commands to manage your Storage queues

- storage directory create [options] [share] [path]
- storage directory delete [options] [share] [path]

Commands to manage shared access signatures of your Storage queue

- storage queue sas create [options] [queue] [permissions] [expiry]

Commands to manage stored access policies of your Storage queue

- storage queue policy create [options] [queue] [name]
- storage queue policy show [options] [queue] [name]
- storage queue policy list [options] [queue]
- storage queue policy set [options] [queue] [name]
- storage queue policy delete [options] [queue] [name]

Commands to manage your Storage logging properties

- storage logging show [options]
- storage logging set [options]

Commands to manage your Storage metrics properties

- storage metrics show [options]
- storage metrics set [options]

Commands to manage your Storage tables

- storage table create [options] [table]
- storage table list [options] [prefix]
- storage table show [options] [table]
- storage table delete [options] [table]

Commands to manage shared access signatures of your Storage table

- storage table sas create [options] [table] [permissions] [expiry]

Commands to manage stored access policies of your Storage table

- storage table policy create [options] [table] [name]
- storage table policy show [options] [table] [name]
- storage table policy list [options] [table]
- storage table policy set [options] [table] [name]
- storage table policy delete [options] [table] [name]
azure tag: Commands to manage your resource manager tag

Add a tag

tag create [options] <name> <value>

Remove an entire tag or a tag value

tag delete [options] <name> <value>

Lists the tag information

tag list [options]

Get a tag

tag show [options] [name]

azure vm: Commands to manage your Azure Virtual Machines

Create a VM

vm create [options] <resource-group> <name> <location> <os-type>

Create a VM with default resources

vm quick-create [options] <resource-group> <name> <location> <os-type> <image-urn> <admin-username> <admin-password>

TIP
Starting with CLI version 0.10, you can provide a short alias such as "UbuntuLTS" or "Win2012R2Datacenter" as the image-urn for some popular Marketplace images. Run azure help vm quick-create for options. Additionally, starting with version 0.10, azure vm quick-create uses premium storage by default if it’s available in the selected region.

List the virtual machines within an account

vm list [options]

Get one virtual machine within a resource group

vm show [options] <resource-group> <name>

Delete one virtual machine within a resource group

vm delete [options] <resource-group> <name>

Shutdown one virtual machine within a resource group
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vm stop [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>Restart one virtual machine within a resource group</td>
</tr>
<tr>
<td><code>vm restart [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>Start one virtual machine within a resource group</td>
</tr>
<tr>
<td><code>vm start [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>Shutdown one virtual machine within a resource group and releases the compute resources</td>
</tr>
<tr>
<td><code>vm deallocate [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>List available virtual machine sizes</td>
</tr>
<tr>
<td><code>vm sizes [options]</code></td>
<td>Capture the VM as OS Image or VM Image</td>
</tr>
<tr>
<td><code>vm capture [options] &lt;resource-group&gt; &lt;name&gt; &lt;vhd-name-prefix&gt;</code></td>
<td>Set the state of the VM to Generalized</td>
</tr>
<tr>
<td><code>vm generalize [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>Get instance view of the VM</td>
</tr>
<tr>
<td><code>vm get-instance-view [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>Enable you to reset Remote Desktop Access or SSH settings on a Virtual Machine and to reset the password for the account that has administrator or sudo authority</td>
</tr>
<tr>
<td><code>vm reset-access [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>Update VM with new data</td>
</tr>
<tr>
<td><code>vm set [options] &lt;resource-group&gt; &lt;name&gt;</code></td>
<td>Commands to manage your Virtual Machine data disks</td>
</tr>
<tr>
<td><code>vm disk attach-new [options] &lt;resource-group&gt; &lt;vm-name&gt; &lt;size-in-gb&gt; [vhd-name]</code></td>
<td>Commands to manage VM resource extensions</td>
</tr>
<tr>
<td><code>vm disk detach [options] &lt;resource-group&gt; &lt;vm-name&gt; &lt;lun&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>vm disk attach [options] &lt;resource-group&gt; &lt;vm-name&gt; [vhd-url]</code></td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| `vm extension set [options] <resource-group> <vm-name> <name> <publisher-name> <version>` | Set VM extension
| `vm extension get [options] <resource-group> <vm-name>` | Get VM extension
| `vm docker create [options] <resource-group> <name> <location> <os-type>` | Create VM docker
| `vm image list-publishers [options] <location>` | List VM images publishers
| `vm image list-offers [options] <location> <publisher>` | List VM images offers
| `vm image list-skus [options] <location> <publisher> <offer>` | List VM images skus
| `vm image list [options] <location> <publisher> [offer] [sku]` | List VM images

**Commands to manage your Docker Virtual Machine**

- `vm docker create [options] <resource-group> <name> <location> <os-type>`

**Commands to manage VM images**

- `vm image list-publishers [options] <location>`
- `vm image list-offers [options] <location> <publisher>`
- `vm image list-skus [options] <location> <publisher> <offer>`
- `vm image list [options] <location> <publisher> [offer] [sku]`