Deploy the ASA Virtual Auto Scale Solution on Azure

Auto Scale Solution for the on Azure

Overview

The auto scale solution enables allocation of resources to match performance requirements and reduce costs. If the demand for resources increases, the system ensures that resources are allocated as required. If the demand for resources decreases, resources are deallocated to reduce costs.

The auto scale for Azure is a complete serverless implementation which makes use of serverless infrastructure provided by Azure (Logic App, Azure Functions, Load Balancers, Security Groups, Virtual Machine Scale Set, etc.).

Some of the key features of the auto scale for Azure implementation include:

- Azure Resource Manager (ARM) template-based deployment.
- Support for scaling metrics based on CPU.

Note See Auto Scale Logic, on page 23 for more information.

- Support for deployment and multi-availability zones.

- Support for Load Balancers and multi-availability zones.
- Support for enabling and disabling the auto scale feature.
- Cisco provides an auto scale for Azure deployment package to facilitate the deployment.

The auto scale solution on Azure supports two types of use cases configured using different topologies:

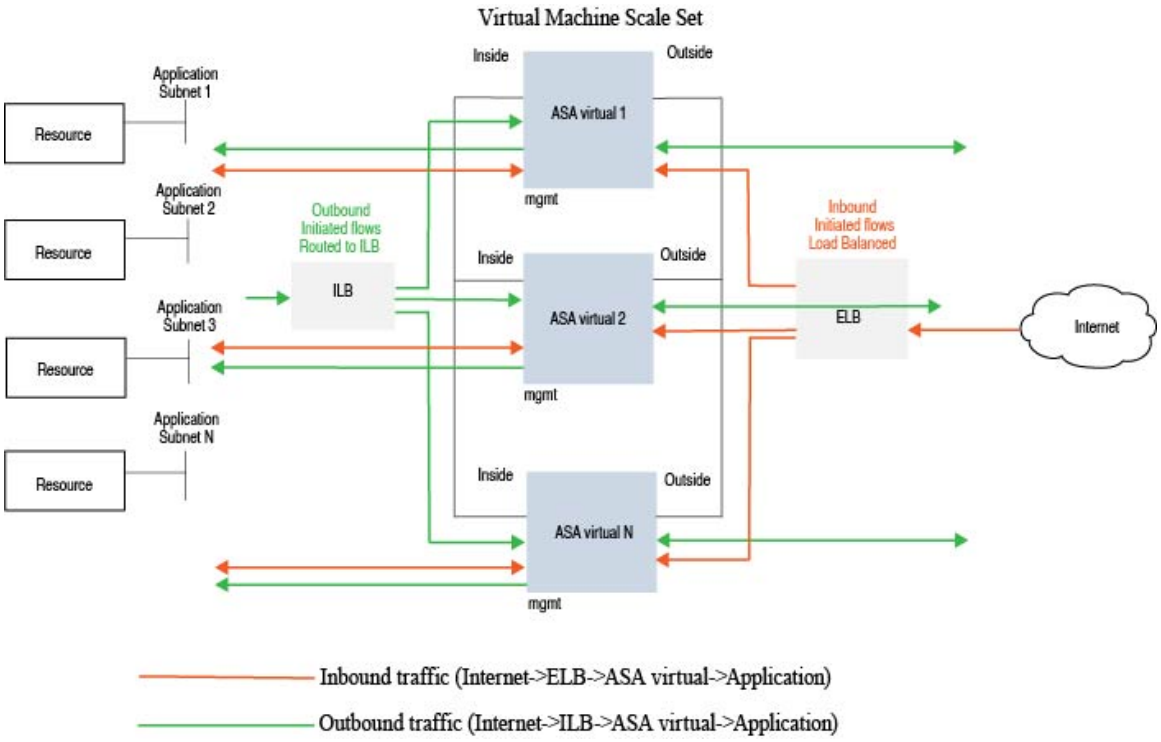
- Auto scale using Sandwich Topology – The scale set is sandwiched between an Azure Internal load balancer (ILB) and an Azure External load balancer (ELB).
- Auto scale with Azure Gateway load balancer (GWLB) – The Azure GWLB is integrated with Secure Firewall, public load balancer, and internal servers - to simplify deployment, management, and scaling of firewalls.

Auto Scale using Sandwich Topology Use Case

The ASA Virtual auto scale for Azure is an automated horizontal scaling solution that positions an ASA Virtual scale set sandwiched between an Azure Internal load balancer (ILB) and an Azure External load balancer (ELB).

- The ELB distributes traffic from the Internet to ASA Virtual instances in the scale set; the firewall then forwards traffic to application.
- The ILB distributes outbound Internet traffic from an application to ASA Virtual instances in the scale set; the firewall then forwards traffic to Internet.
- A network packet will never pass through both (internal & external) load balancers in a single connection.
- The number of ASA Virtual instances in the scale set will be scaled and configured automatically based on load conditions.

Figure 1: ASA Virtual Auto Scale using Sandwich Topology Use Case



Auto Scale with Azure Gateway Load Balancer Use Case

The Azure Gateway Load Balancer (GWLB) ensures that internet traffic to and from an Azure VM, such as an application server, is inspected by Secure Firewall without requiring any routing changes. This integration of the Azure GWLB with Secure Firewall simplifies deployment, management, and scaling of firewalls. This integration also reduces operational complexity and provides a single entry and exit point for traffic at the firewall. The applications and infrastructure can maintain visibility of source IP address, which is critical in some environments.

In the Azure GWLB Auto Scale use case, the uses only two interfaces: Management and one data interface.

Note

- Network Address Translation (NAT) is not required if you are deploying the Azure GWLB.
- Only IPv4 is supported.

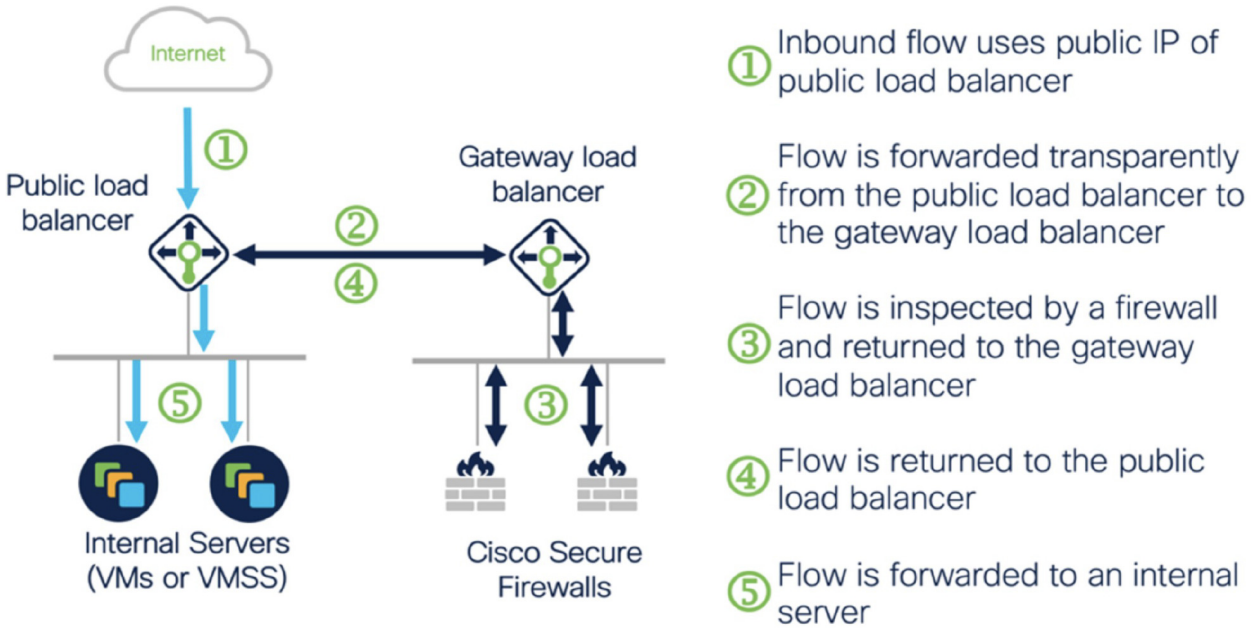
Licensing

Both PAYG and BYOL are supported.

BYOL is supported.

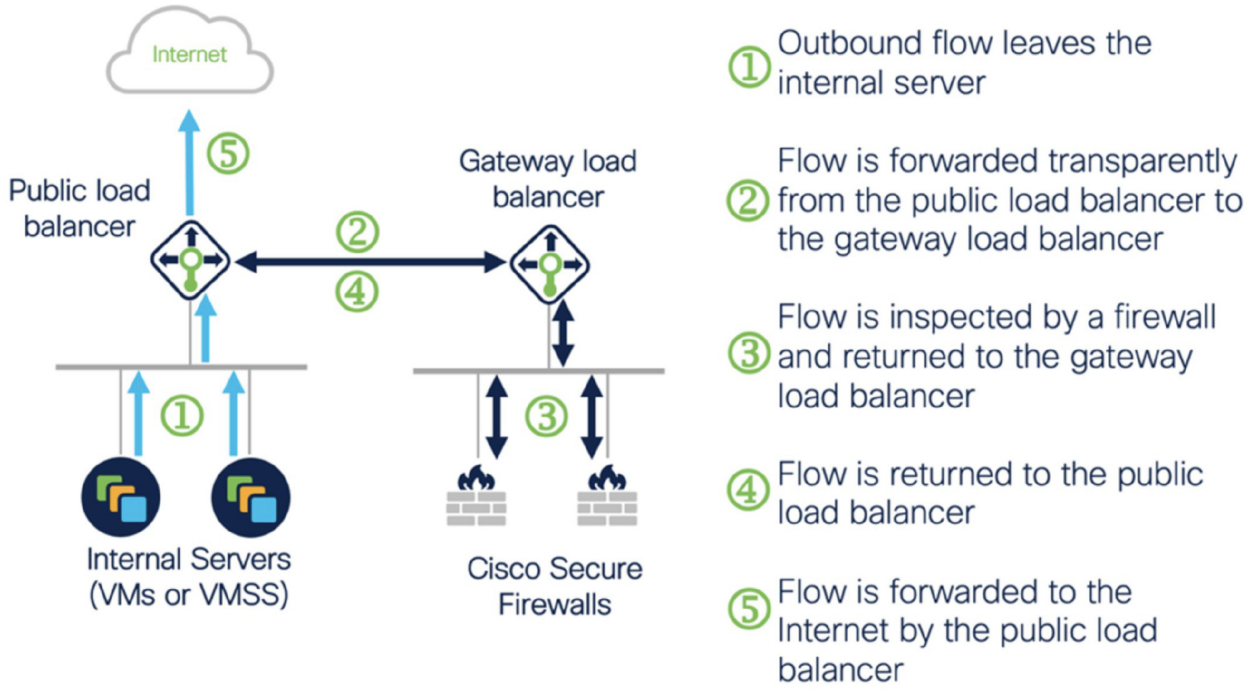
Inbound Traffic Use Case and Topology

The following diagram displays the traffic flow for inbound traffic.



Outbound Traffic Use Case and Topology

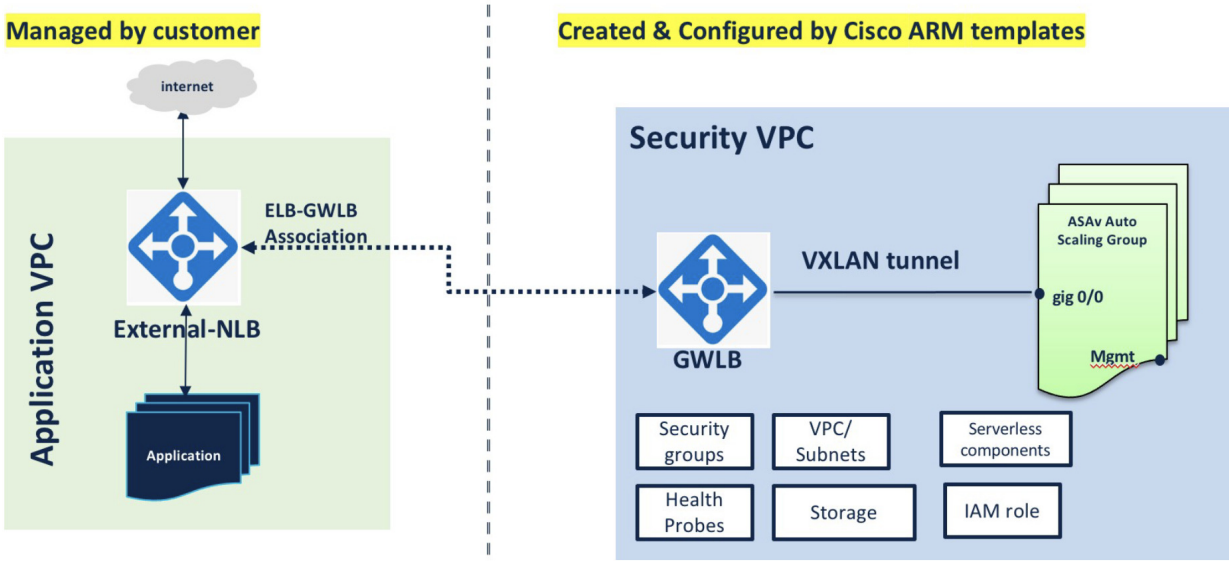
The following diagram displays the traffic flow for outbound traffic.



Traffic Flow between the Application VNet and Security VNet

In the diagram shown below, traffic is redirected from the existing topology to the firewalls for inspection by the external load balancer. The traffic is then routed to the newly created GWLB. Any traffic that is routed to the ELB is forwarded to the GWLB.

The GWLB then forwards the VXLAN-encapsulated traffic to a instance. You have to create two associations as the GWLB uses two separate VXLAN tunnels for ingress and egress traffic. The decapsulates the VXLAN-encapsulated traffic, inspects it, and routes the traffic to the GWLB. The GWLB then forwards the traffic to the ELB.



Scope

This document covers the detailed procedures to deploy the serverless components for the auto scale for Azure solution.

Important

- Read the entire document before you begin your deployment.
- Make sure the prerequisites are met before you start deployment.
- Make sure you follow the steps and order of execution as described herein.

Download the Deployment Package

The auto scale for Azure solution is an Azure Resource Manager (ARM) template-based deployment which makes use of the serverless infrastructure provided by Azure (Logic App, Azure Functions, Load Balancers, Virtual Machine Scale Set, and so on).

Download the files required to launch the auto scale for Azure solution. Deployment scripts and templates for your version are available in the repository.

Attention Note that Cisco-provided deployment scripts and templates for auto scale are provided as open source examples, and are not covered within the regular Cisco TAC support scope. Check GitHub regularly for updates and ReadMe instructions.

See Build Azure Functions from Source Code, on page 26 for instructions on how to build the *ASM_Function.zip* package.

Auto Scale Solution Components

The following components make up the auto scale for Azure solution.

Azure Functions (Function App)

The Function App is a set of Azure functions. The basic functionality includes:

- Communicate/Probe Azure metrics periodically.
- Monitor the load and trigger Scale In/Scale Out operations.

These functions are delivered in the form of compressed Zip package (see Build the Azure Function App Package, on page 10). The functions are as discrete as possible to carry out specific tasks, and can be upgraded as needed for enhancements and new release support.

Orchestrator (Logic App)

The Auto Scale Logic App is a workflow, i.e. a collection of steps in a sequence. Azure functions are independent entities and cannot communicate with each other. This orchestrator sequences the execution of these functions and exchanges information between them.

- The Logic App is used to orchestrate and pass information between the auto scale Azure functions.
- Each step represents an auto scale Azure function or built-in standard logic.
- The Logic App is delivered as a JSON file.
- The Logic App can be customized via the GUI or JSON file.

Virtual Machine Scale Set (VMSS)

The VMSS is a collection of homogeneous virtual machines, such as devices.

- The VMSS is capable of adding new identical VMs to the set.
- New VMs added to the VMSS are automatically attached with Load Balancers, Security Groups, and network interfaces.
- The VMSS has a built–in auto scale feature which is disabled for for Azure.
- You should not add or delete instances in the VMSS manually.

Azure Resource Manager (ARM) Template

ARM templates are used to deploy the resources required by the auto scale for Azure solution.

ASA virtual auto scale for Azure - The ARM template provides input for the Auto Scale Manager components including:

- Azure Function App
- Azure Logic App
- The Virtual Machine Scale Set (VMSS)
- Internal/External load balancers.
- Security Groups and other miscellaneous components needed for deployment.

ASA virtual auto scale with Azure GWLB - The ARM template provides input for the Auto Scale Manager components including:

- Azure Function App
- Azure Logic App
- Virtual Machine (VM) or Virtual Machine Scale Set (VMSS)
- Networking Infrastructure
- Gateway load balancer
- Security Groups and other miscellaneous components needed for deployment

Important The ARM template has limitations with respect to validating user input, hence it is your responsibility to validate input during deployment.

Prerequisites

- Ensure that you have Owner role in the Azure subscription.
- Create the Azure Resource Group. Ensure that the Azure Virtual Network along with the necessary subnets are created.
  - Interfaces for NLB-based cluster : Management, Diagnostic, Inside, Outside, CCL and the function app.
  - Interfaces for GWLB-based cluster : Management, Diagnostic, Data, CCL and the function app.
- On the Management Center:
  - Ensure that Management Center Virtual is licensed correctly.
  - Create the access control policy.
  - Create the Security Zone (SZ) object for the interfaces. For NLB based cluster, create the SZ for inside and outside interfaces. For GWLB-based cluster, create the SZ for the data interface.

- Create a separate user name and password for the azure function to add the Threat Defense Virtual instances to the Management Center Virtual and configure the instances.

- Install the Azure CLI on your local system.
- Download the Azure Clustering Autoscale repository from GitHub to your local computer and run the command **python3 make.py build** to create the Azure functions zip file.

Azure Resources

Resource Group

An existing or newly created Resource Group is required to deploy all the components of this solution.

Note Record the Resource Group name, the Region in which it is created, and the Azure Subscription ID for later use.

Networking

Make sure a virtual network is available or created. An auto scale deployment with sandwich topology does not create, alter, or manage any networking resources. However, note that auto scale deployment with the Azure GWLB creates networking infrastructure.

The requires network interfaces, thus your virtual network requires subnets for:

1. Management traffic
2. Inside traffic
3. Outside traffic

The following ports should be open in the Network Security Group to which the subnets are connected:

- SSH(TCP/22)

  Required for the Health probe between the Load Balancer and .

  Required for communication between the Serverless functions and .

- Application-specific protocol/ports

  Required for any user applications (for example, TCP/80, etc.).

Note Record the virtual network name, the virtual network CIDR, the names of the subnets, and the Gateway IP addresses of the outside and inside subnets.

Prepare the ASA Configuration File

Prepare an ASA Virtual configuration file and store in a http/https server accessible by the ASA Virtual instance. This is a standard ASA configuration file format. A scaled-out ASA Virtual will download this file and update its configuration.

The ASA configuration file should have the following (at a minimum):

- Set DHCP IP assignment to all the interfaces.
- GigabitEthernet0/1 should be the 'inside' interface.
- GigabitEthernet0/0 should be the 'outside' interface.

Note Auto scale deployment using sandwich topology requires two data interfaces. However, auto scale deployment with Azure GWLB requires only one data interface.

- Set the gateway to the inside and outside interface.
- Enable SSH on the inside and outside interface from Azure utility IP (for health probe).
- Create a NAT configuration to forward traffic from the outside to the inside interface.
- Create an access policy to allow desired traffic.
- License the configuration. PAYG billing is not supported.

Note There is no need to specifically configure the Management interface.

The following is a sample ASA configuration file for the ASA Virtual auto scale for Azure solution.

```
ASA Version 9.13(1)
!
interface GigabitEthernet0/1
nameif inside
security-level 100
ip address dhcp setroute
!
interface GigabitEthernet0/0
nameif outside
security-level 0
ip address dhcp setroute
!
route outside 0.0.0.0 0.0.0.0 10.12.3.1 2
!
route inside 0.0.0.0 0.0.0.0 10.12.2.1 3
!
ssh 168.63.129.0 255.255.255.0 outside
!
ssh 168.63.129.0 255.255.255.0 inside
!
object network webserver
host 10.12.2.5
```

```
object service myport
service tcp source range 1 65535 destination range 1 65535
access-list outowebaccess extended permit object myport any any log disable
access-group outowebaccess in interface outside
object service app
service tcp source eq www
nat (inside,outside) source static webserver interface destination static interface any
service app app
object network obj-any
subnet 0.0.0.0 0.0.0.0
nat (inside,outside) source dynamic obj-any interface destination static obj-any obj-any
configure terminal
dns domain-lookup management
policy-map global_policy
class inspection_default
inspect icmp
call-home
profile License
destination transport-method http
destination address http https://tools.cisco.com/its/service/oddce/services/DDCEService
license smart
feature tier standard
throughput level 2G
license smart register idtoken <TOKEN>
: end
```

The following is a sample ASA configuration file for the ASA Virtual auto scale with Azure GWLB solution.

```
interface G0/0
nameif outside
ip address dhcp setroute
no shut
!s
sh 168.63.129.0 255.255.255.0 outside
route outside 0.0.0.0 0.0.0.0 192.168.2.1 2
nve 1
encapsulation vxlan
source-interface outside
peer ip 192.168.2.100
!i
nterface vni1
proxy paired
nameif GWLB-backend-pool
internal-port 2000
internal-segment-id 800
external-port 2001
external-segment-id 801
vtep-nve 1
!s
ame-security-traffic permit intra-interface
```

Build the Azure Function App Package

The auto scale solution requires that you build an archive file: *ASM_Function.zip*. which delivers a set of discrete Azure functions in the form of a compressed ZIP package.

See Build Azure Functions from Source Code, on page 26 for instructions on how to build the *ASM_Function.zip* package.

These functions are as discrete as possible to carry out specific tasks, and can be upgraded as needed for enhancements and new release support.

Input Parameters

The following table defines the template parameters and provides an example. Once you decide on these values, you can use these parameters to create the device when you deploy the ARM template into your Azure subscription. See Deploy the Auto Scale ARM Template, on page 14.In the Auto scale with Azure GWLB solution, networking infrastructure is also created due to which additional input parameters have to be configured in the template. The parameter descriptions are self-explanatory.

Table 1: Template Parameters

Parameter Name	Allowed Values/Type	Description	Resource Creation Type
resourceNamePrefix	String* (3-10 characters)	All the resources are created with name containing this prefix. Note: Use only lowercase letters. Example:	New
virtualNetworkRg	String	The virtual network resource group name. Example: cisco-virtualnet-rg	Existing
virtualNetworkName	String	The virtual network name (already created). Example: cisco-virtualnet	Existing
mgmtSubnet	String	The management subnet name (already created). Example: cisco-mgmt-subnet	Existing
insideSubnet	String	The inside Subnet name (already created). Example: cisco-inside-subnet	Existing
internalLbIp	String	The internal load balancer IP address for the inside subnet (already created). Example: 1.2.3.4	Existing
outsideSubnet	String	The outside subnet name (already created). Example: cisco-outside-subnet	Existing
softwareVersion	String	The Version (selected from drop-down during deployment).	Existing

Parameter Name	Allowed Values/Type	Description	Resource Creation Type
vmSize	String	Size of instance (selected from drop-down during deployment).	N/A
scalingPolicy	POLICY-1 / POLICY-2	**POLICY-1**: Scale-Out will be triggered when the average load of any goes beyond the Scale Out threshold for the configured duration. **POLICY-2**: Scale-Out will be triggered when average load of all the devices in the auto scale group goes beyond the Scale Out threshold for the configured duration. In both cases Scale-In logic remains the same: Scale-In will be triggered when average load of all the devices comes below the Scale In threshold for the configured duration.	N/A
scalingMetricsList	String	Metrics used in making the scaling decision. Allowed: CPU Default: CPU	N/A
	String	The Scale-In threshold in percent. Default: 10 When the metric goes below this value the Scale-In will be triggered. See Auto Scale Logic, on page 23.	N/A
	String	The Scale-Out threshold in percent. Default: 80 When the metric goes above this value, the Scale-Out will be triggered. The ‘’ should always be **greater** than the ‘’. See Auto Scale Logic, on page 23.	N/A

Parameter Name	Allowed Values/Type	Description	Resource Creation Type
	Integer	The minimum instances available in the scale set at any given time. Example: 2	N/A
	Integer	The maximum instances allowed in the Scale set. Example: 10 **Note** The Auto Scale logic will not check the range of this variable, hence fill this carefully.	N/A
metricsAverageDuration	Integer	Select from the drop-down. This number represents the time (in minutes) over which the metrics are averaged out. If the value of this variable is 5 (i.e. 5min), when the Auto Scale Manager is scheduled it will check the past 5 minutes average of metrics and based on this it will make a scaling decision. **Note** Only numbers 1, 5, 15, and 30 are valid due to Azure limitations.	N/A
initDeploymentMode	BULK / STEP	Primarily applicable for the first deployment, or when the Scale Set does not contain any instances. BULK: The Auto Scale Manager will try to deploy " number of instances in parallel at one time. STEP: The Auto Scale Manager will deploy the " number of devices one by one at each scheduled interval.	
*Azure has restrictions on the naming convention for new resources. Review the limitations or simply use all lowercase. **Do not use spaces or any other special characters**.			

Deploy the Auto Scale Solution

Deploy the Auto Scale ARM Template

auto scale for Azure using Sandwich Topology - Use the ARM template to deploy the resources required by the auto scale for Azure. Within a given resource group, the ARM template deployment creates the following:

- Virtual Machine Scale Set (VMSS)
- External Load Balancer
- Internal Load Balancer
- Azure Function App
- Logic App
- Security groups (For Data and Management interfaces)

auto scale with Azure GWLB - Use the ARM template to deploy the resources required by the auto scale with Azure GWLB solution. Within a given resource group, the ARM template deployment creates the following:

- Virtual Machine (VM) or Virtual Machine Scale Set (VMSS)
- Gateway Load Balancer
- Azure Function App
- Logic App
- Networking Infrastructure
- Security Groups and other miscellaneous components needed for deployment

Before you begin

- Download the ARM templates from the GitHub repository ().

Procedure

Step 1 If you need to deploy the instances in multiple Azure zones, edit the ARM template based on the zones available in the Deployment region.

Example:

```
"zones": [
    "1",
    "2",
    "3"
```

```
        ],
```

This example shows the “Central US” region which has 3 zones.

Step 2 Edit the traffic rules required in External Load Balancer. You can add any number of rules by extending this ‘json’ array.

Example:

```
       {
        "type": "Microsoft.Network/loadBalancers",
        "name": "[variables('elbName')]",
        "location": "[resourceGroup().location]",
        "apiVersion": "2018-06-01",
        "sku": {
          "name": "Standard"
        },
        "dependsOn": [
          "[concat('Microsoft.Network/publicIPAddresses/', variables('elbPublicIpName'))]"

        ],
        "properties": {
          "frontendIPConfigurations": [
            {
              "name": "LoadBalancerFrontEnd",
                "properties": {
                  "publicIPAddress": {
                    "id": "[resourceId('Microsoft.Network/publicIPAddresses/',
variables('elbPublicIpName'))]"
                  }
                }
            }
          ],
          "backendAddressPools": [
            {
              "name": "backendPool"
            }
          ],
          "loadBalancingRules": [
            {
              "properties": {
                "frontendIPConfiguration": {
                 "Id": "[concat(resourceId('Microsoft.Network/loadBalancers', variables('elbName')),
 '/frontendIpConfigurations/LoadBalancerFrontend')]"
                },
                "backendAddressPool": {
                "Id": "[concat(resourceId('Microsoft.Network/loadBalancers', variables('elbName')),
 '/backendAddressPools/BackendPool')]"
                },
                "probe": {
                "Id": "[concat(resourceId('Microsoft.Network/loadBalancers', variables('elbName')),
 '/probes/lbprobe')]"
                },
                "protocol": "TCP",
                "frontendPort": "80",
                "backendPort": "80",
                "idleTimeoutInMinutes": "[variables('idleTimeoutInMinutes')]"
              },
              "Name": "lbrule"
            }
          ],
```

Note
You can also edit this from the Azure portal post-deployment if you prefer not to edit this file.

Step 3 Log in to the Microsoft Azure portal using your Microsoft account username and password.

Step 4 Click **Resource groups** from the menu of services to access the Resource Groups blade. You will see all the resource groups in your subscription listed in the blade.

Create a new resource group or select an existing, empty resource group; for example, *_AutoScale*.

Step 5 Click **Create a resource (+)** to create a new resource for template deployment. The Create Resource Group blade appears.

Step 6 In **Search the Marketplace**, type **Template deployment (deploy using custom templates)**, and then press **Enter**.

Step 7 Click **Create**.

Step 8 There are several options for creating a template. Choose **Build your own template in editor**.

Step 9 In the **Edit template** window, delete all the default content and copy the contents from the updated *azure__autoscale.json* and click **Save**.

Step 10 In next section, fill all the parameters. Refer to Input Parameters, on page 11 for details about each parameter, then click **Purchase**.

Note
You can also click **Edit Parameters** and edit the JSON file or upload pre-filled contents.

The ARM template has limited input validation capabilities, hence it is your responsibility to validate the input.

Step 11 When a template deployment is successful, it creates all the required resources for the auto scale for Azure solution. See the resouses in the following figure. The Type column describes each resource, including the Logic App, VMSS, Load Balancers, Public IP address, etc.

Deploy the Azure Function App

When you deploy the ARM template, Azure creates a skeleton Function App, which you then need to update and configure manually with the functions required for the Auto Scale Manager logic.

Before you begin

- Build the *ASM_Function.zip* package. See Build Azure Functions from Source Code, on page 26.

Procedure

Step 1 Go to the Function App you created when you deployed the ARM template, and verify that no functions are present. In a browser go to this URL:

https://<Function App Name>.scm.azurewebsites.net/DebugConsole

Step 2 In the file explorer navigate to **site/wwwroot**.

Step 3 Drag-and-drop the **ASM_Function.zip** to the right side corner of the file explorer.

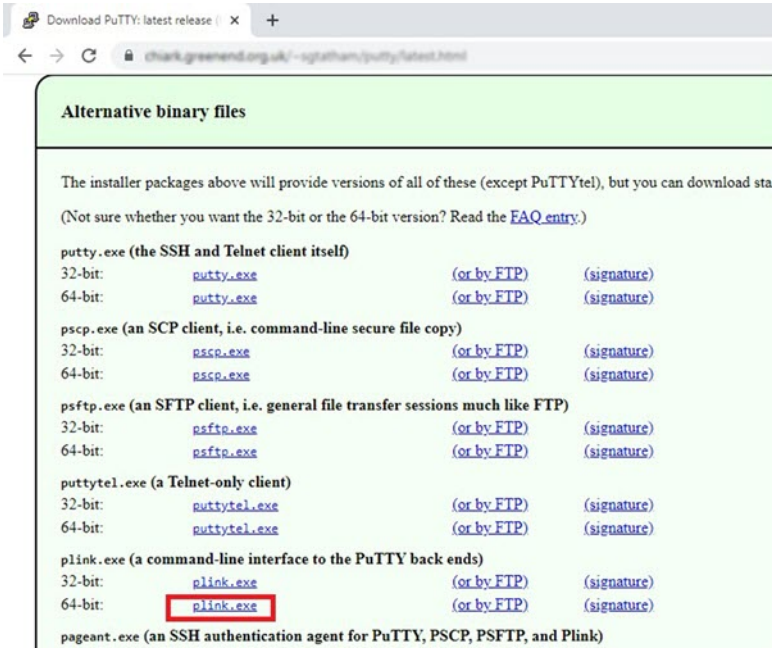
Step 4 Once the upload is successful, all of the serverless functions should appear.

Step 5 Download the PuTTY SSH client.

Azure functions need to access the via an SSH connection. However, the opensource libraries used in the serverless code do not support the SSH key exchange algorithms used by the . Hence you need to download a pre-built SSH client.

Download the PuTTY command-line interface to the PuTTY back end (*plink.exe*) from www.putty.org.

Figure 2: Download PuTTY



Step 6 Rename the SSH client executable file **plink.exe** to .

Step 7 Drag-and-drop the to the right side corner of the file explorer, to the location where **ASM_Function.zip** was uploaded in the previous step.

Step 8 Verify the SSH client is present with the function application. Refresh the page if necessary.

Fine Tune the Configuration

There are a few configurations available to fine tune the Auto Scale Manager or to use in debugging. These options are not exposed in the ARM template, but you can edit them under the Function App.

Before you begin

Note This can be edited at any time. Follow this sequence to edit the configurations.

- Disable the Function App.
- Wait for existing scheduled task to finish.
- Edit and save the configuration.
- Enable the Function App.

Procedure

Step 1 In the Azure portal, search for and select the function application.

Step 2 Configurations passed via the ARM template can also be edited here. Variable names may appear different from the ARM template, but you can easily identify the purpose of these variables from their name.

Most of the options are self-explanatory from the name. For example:

- Configuration Name: “DELETE_FAULTY_” (Default value : YES)

  During Scale-Out, a new instance is launched and . In case the fails, based on this option, Auto Scale Manager will decide to keep that instance or delete it. (YES : Delete faulty / NO : Keep the instance).

- In the Function App settings, all the variables (including variables containing a secure string like ‘password’) can be seen in clear text format by users that have access to the Azure subscription.

  If users have any security concerns with this (for example, if an Azure subscription is shared among users with lower privilages within the organization), a user can make use of Azure’s *Key Vault* service to protect passwords. Once this is configured, instead of providing a clear text ‘password’ in function settings, a user has to provide a secure identifier generated by the key vault where the password is stored.

  Note
  Search the Azure documentation to find the best practices to secure your application data.

Configure the IAM Role in the Virtual Machine Scale Set

Azure Identity and Access Management (IAM) is used as a part of Azure Security and Access Control to manage and control a user’s identity. Managed identities for Azure resources provides Azure services with an automatically managed identity in Azure Active Directory.

This allows the Function App to control the Virtual Machine Scale Sets (VMSS) without explicit authentication credentials.

Procedure

Step 1 In the Azure portal, go to the VMSS.

Step 2 Click **Access control (IAM)**.

Step 3 Click **Add** to add a role assignment

Step 4 From the **Add role assignment** drop-down, choose **Contributor**.

Step 5 From the **Assign access to** drop-down, choose **Function App**.

Step 6 Select the function application.

Step 7 Click **Save**.

Note
You should also verify that there are no instances launched yet.

Update Security Groups

The ARM template creates two security groups, one for the Management interface, and one for data interfaces. The Management security group will allow only traffic required for management activities. However, the data interface security group will allow all traffic.

Procedure

Fine tune the security group rules based on the topology and application needs of your deployments.

Note
The data interface security group should allow, at a minimum, SSH traffic from the load balancers.

Update the Azure Logic App

The Logic App acts as the orchestrator for the Autoscale functionality. The ARM template creates a skeleton Logic App, which you then need to update manually to provide the information necessary to function as the auto scale orchestrator.

Procedure

Step 1 From the repository, retrieve the file *LogicApp.txt* to the local system and edit as shown below.

Important
Read and understand all of these steps before proceeding.

These manual steps are not automated in the ARM template so that only the Logic App can be upgraded independently later in time.

a) Required: Find and replace all the occurrences of "SUBSCRIPTION_ID" with your subscription ID information.
b) Required: Find and replace all the occurrences of "RG_NAME" with your resource group name.
c) Required: Find and replace all of the occurrences of "FUNCTIONAPPNAME" to your function app name.

   The following example shows a few of these lines in the *LogicApp.txt* file:

```
  "AutoScaleManager": {
      "inputs": {
          "function": {
              "id":
"/subscriptions/SUBSCRIPTION_ID/resourceGroups/RG_NAME/providers/Microsoft.Web/sites/FUNCTIONAPPNAME/functions/AutoScaleManager"

          }
.
.
                      },
                      "Deploy_Changes_to_": {
                          "inputs": {
                              "body": "@body('AutoScaleManager')",
                              "function": {
                                  "id":
"/subscriptions/SUBSCRIPTION_ID/resourceGroups/RG_NAME/providers/Microsoft.Web/sites/FUNCTIONAPPNAME/functions/DeployConfiguration"

                              }
.
.
                      "DeviceDeRegister": {
                          "inputs": {
                              "body": "@body('AutoScaleManager')",
                              "function": {
                                  "id":
"/subscriptions/SUBSCRIPTION_ID/resourceGroups/RG_NAME/providers/Microsoft.Web/sites/FUNCTIONAPPNAME/functions/DeviceDeRegister"

                              }
                          },
                          "runAfter": {
                              "Delay_For_connection_Draining": [
```

d) (Optional) Edit the trigger interval, or leave the default value (5). This is the time interval at which the Autoscale functionality is periodically triggered. The following example shows these lines in the *LogicApp.txt* file:

```
        "triggers": {
            "Recurrence": {
                "conditions": [],
                "inputs": {},
                "recurrence": {
                    "frequency": "Minute",
                    "interval": 5
                },
```

e) (Optional) Edit the time to drain, or leave the default value (5). This is the time interval to drain existing connections from the before deleting the device during the Scale-In operation. The following example shows these lines in the *LogicApp.txt* file:

```
         "actions": {
```

```
"Branch_based_on_Scale-In_or_Scale-Out_condition": {
    "actions": {
        "Delay_For_connection_Draining": {
            "inputs": {
                "interval": {
                    "count": 5,
                    "unit": "Minute"
                }
```

f) (Optional) Edit the cool down time, or leave the default value (10). This is the time to perform NO ACTION after the Scale-Out is complete. The following example shows these lines in the *LogicApp.txt* file:

```
"actions": {
    "Branch_based_on_Scale-Out_or_Invalid_condition": {
        "actions": {
            "Cooldown_time": {
                "inputs": {
                    "interval": {
                        "count": 10,
                        "unit": "Second"
                    }
```

Note
These steps can also be done from the Azure portal. Consult the Azure documentation for more information.

Step 2 Go to the **Logic App code view**, delete the default contents and paste the contents from the edited *LogicApp.txt* file, and click **Save**.

Step 3 When you save the Logic App, it is in a 'Disabled' state. Click **Enable** when you want to start the Auto Scale Manager.

Step 4 Once enabled, the tasks start running. Click the 'Running' status to see the activity.

Step 5 Once the Logic App starts, all the deployment-related steps are complete.

Step 6 Verify in the VMSS that instances are being created.

Figure 3: Instances Running

In this example, three instances are launched because was set to '3' and 'initDeploymentMode' was set to 'BULK' in the ARM template deployment.

Upgrade the

The upgrade is supported only in the form of an image upgrade of virtual machine scale set (VMSS). Hence, you upgrade the through the Azure REST API interface.

Note You can use any REST client to upgrade the .

Before you begin

- Obtain the new image version available in market place (example:).

- Obtain the SKU used to deploy original scale set (example: -azure-byol).
- Obtain the Resource Group and the virtual machine scale set name.

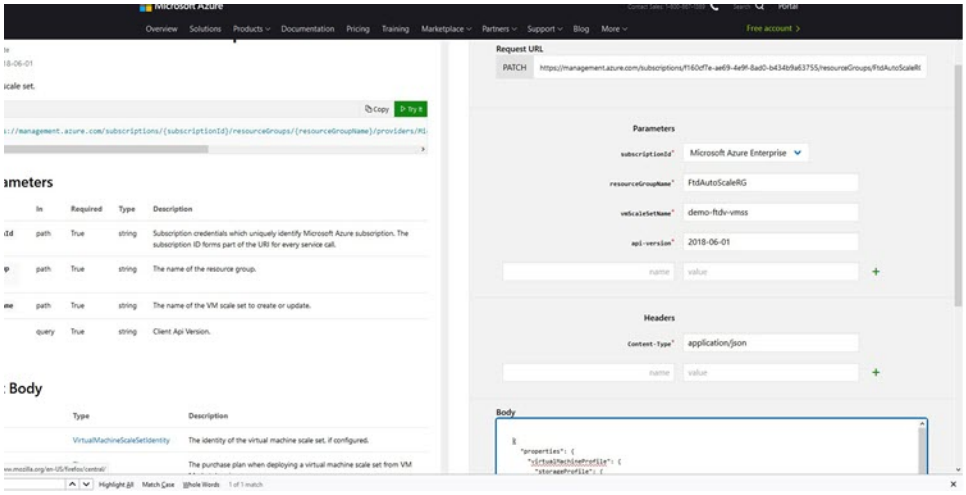
Procedure

Step 1 In a browser go to the following URL:

https://docs.microsoft.com/en-us/rest/api/compute/virtualmachinescalesets/update#code-try-0

Step 2 Enter the details in the Parameters section.

Figure 4: Upgrade the



Step 3 Enter the JSON input containing the new image version, SKU, and trigger RUN in the **Body** section.

```
{
 "properties": {
        "virtualMachineProfile": {
              "storageProfile": {
                      "imageReference": {
                           "publisher": "cisco",
                           "offer": "cisco-",
                           "sku": "-azure-byol",
                            "version": "650.32.0"
                      }
                  },
            }
      }
}
```

Step 4 A successful response from Azure means that the VMSS has accepted the change.

The new image will be used in the new instances which will get launched as part of Scale-Out operation.

- Existing instances will continue to use the old software image while they exist in a scale set.

- You can override the above behavior and upgrade the existing instances manually. To do this, click the **Upgrade** button in the VMSS. It will reboot and upgrade the selected instances. You must reregister and reconfigure these upgraded instances manually. **Note that this method is NOT recommended.**

Auto Scale Logic

Scale-Out Logic

- **POLICY-1**: Scale-Out will be triggered when the average load of **any** goes beyond the Scale-Out threshold for the configured duration.
- **POLICY-2**: Scale-Out will be triggered when average load of **all** of the devices go beyond Scale-Out threshold for the configured duration.

Scale-In Logic

- If the CPU utilization of **all** of the devices goes below the configured Scale-In threshold for the configured duration.

Notes

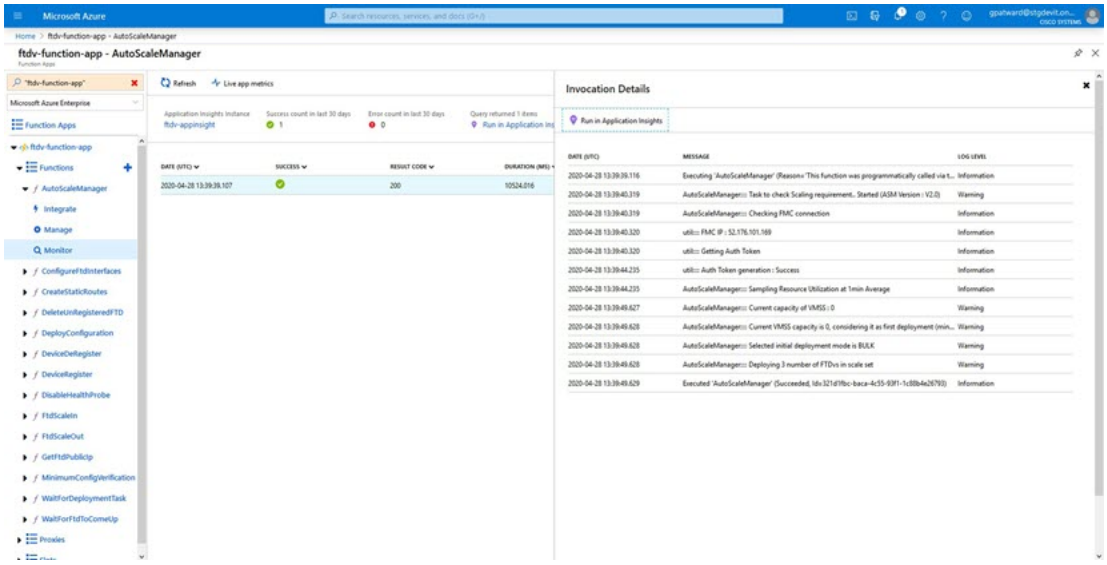
- Scale-In/Scale-Out occurs in steps of 1 (i.e. only 1 will be scaled in/out at a time).

Auto Scale Logging and Debugging

Each component of the serverless code has its own logging mechanism. In addition, logs are published to application insight.

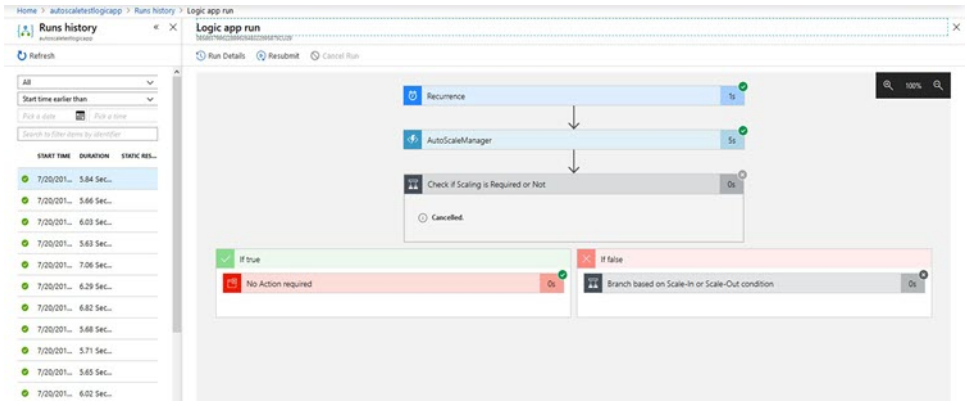
- Logs of individual Azure functions can be viewed.

Figure 5: Azure Function Logs



- Similar logs for each run of the Logic App and its individual components can be viewed.

Figure 6: Logic App Run Logs



- If needed, any running task in the Logic App can be stopped/terminated at any time. However, currently running devices getting launched/terminated will be in an inconsistent state.
- The time taken for each run/individual task can be seen in the Logic App.
- The Function App can be upgraded at any time by uploading a new zip. Stop the Logic App and wait for all tasks to complete before upgrading the Function App.

Auto Scale Guidelines and Limitations

Be aware of the following guidelines and limitations when deploying the auto scale for Azure:

- Scaling decisions are based on CPU utilization.

- The Management interface is configured to have public IP address.
- Only IPv4 is supported.
- The ARM template has limited input validation capabilities, hence it is your responsibility to provide the correct input validation.
- The Azure administrator can see sensitive data (such as admin login credentials and passwords) in plain text format inside Function App environment. You can use the *Azure Key Vault* service to secure sensitive data.
- Any changes in configuration won't be automatically reflected on already running instances. Changes will be reflected on upcoming devices only. Any such changes should be manually pushed to already existing devices.
- If you are facing issues while manually updating the configuration on existing instances, we recommend removing these instances from the Scaling Group and replacing them with new instances.

Troubleshooting

The following are common error scenarios and debugging tips for the auto scale for Azure:

- Unable to SSH into the : Check if a complex password is passed to the via the template; check if Security Groups allow SSH connections.
- Load Balancer Health check failure: Check if the responds to SSH on data interfaces; check Security Group settings.
- Traffic issues: Check Load Balancer rules, NAT rules / Static routes configured in ; check Azure virtual network / subnets / gateway details provided in the template and Security Group rules.
- Logic App failed to access VMSS: Check if the IAM role configuration in VMSS is correct.
- Logic App runs for very long time: Check SSH access on scaled-out devices; check the state of the devices in Azure VMSS.
- Azure Function throwing error related to subscription ID : Verify that you have a default subscription selected in your account.
- Failure of Scale-In operation: Sometimes, Azure takes a considerably long time to delete an instance in such situations, Scale-in operation may time out and report an error; but eventually the instance, will get deleted.
- Before doing any configuration change, make sure to disable the logic application and wait for all the running tasks to complete.

The following are troubleshooting tips if you encounter any issues during auto scale with Azure GWLB deployment:

- Check the ELB-GWLB association.
- Check the health probe status in the GWLB.
- Check VXLAN configuration by verifying the traffic flow at the physical and logical interfaces of the .
- Check security group rules.

Build Azure Functions from Source Code

System Requirements

- Microsoft Windows desktop/laptop.
- Visual Studio (tested with Visual studio 2019 version 16.1.3)

Note Azure functions are written using C#.

- The "Azure Development" workload needs to be installed in Visual Studio.

Build with Visual Studio

1. Download the 'code' folder to the local machine.
2. Navigate to the folder ".
3. Open the project file '.csproj' in Visual Studio.
4. Use Visual Studio standard procedure to Clean and Build.
5. Once the build is compiled successfully, navigate to the **\bin\Release\netcoreapp2.1** folder.
6. Select all the contents, click **Send to** > **Compressed (zipped) folder**, and save the ZIP file as *ASM_Function.zip*.