Microsoft Azure Guide

Important

Red Hat Ansible Automation Platform will soon be available on Microsoft Azure. Sign up to preview the experience.

Ansible includes a suite of modules for interacting with Azure Resource Manager, giving you the tools to easily create and orchestrate infrastructure on the Microsoft Azure Cloud.

Requirements

Using the Azure Resource Manager modules requires having specific Azure SDK modules installed on the host running Ansible.

```
$ pip install 'ansible[azure]'
```

If you are running Ansible from source, you can install the dependencies from the root directory of the Ansible repo.

```
$ pip install .[azure]
```

You can also directly run Ansible in Azure Cloud Shell, where Ansible is pre-installed.

Authenticating with Azure

Using the Azure Resource Manager modules requires authenticating with the Azure API. You can choose from two authentication strategies:

- Active Directory Username/Password
- Service Principal Credentials

Follow the directions for the strategy you wish to use, then proceed to Providing Credentials to Azure Modules for instructions on how to actually use the modules and authenticate with the Azure API.

Using Service Principal

There is now a detailed official tutorial describing how to create a service principal.

After stepping through the tutorial you will have:

- Your Client ID, which is found in the "client id" box in the "Configure" page of your application in the Azure portal
- Your Secret key, generated when you created the application. You cannot show the key after creation. If you lost the key, you must create a new one in the "Configure" page of your application.
- And finally, a tenant ID. It's a UUID (for example, ABCDEFGH-1234-ABCD-1234-ABCDEFGHIJKL) pointing to the AD containing your application. You will find it in the URL from within the Azure portal, or in the "view endpoints" of any given URL.

Using Active Directory Username/Password

To create an Active Directory username/password:

- Connect to the Azure Classic Portal with your admin account
- Create a user in your default AAD. You must NOT activate Multi-Factor Authentication
- Go to Settings - Administrators
- Click on Add and enter the email of the new user.
- Check the checkbox of the subscription you want to test with this user.
- Login to Azure Portal with this new user to change the temporary password to a new one. You will not be able to use the temporary password for OAuth login.

Providing Credentials to Azure Modules

The modules offer several ways to provide your credentials. For a CI/CD tool such as Ansible AWX or Jenkins, you will most likely want to use environment variables. For local development you may wish to store your credentials in a file within your home directory. And of course, you can always pass credentials as parameters to a task within a playbook. The order of precedence is parameters, then environment variables, and finally a file found in your home directory.

Using Environment Variables

To pass service principal credentials via the environment, define the following variables:

- AZURE_CLIENT_ID
- AZURE_SECRET
- AZURE_SUBSCRIPTION_ID
- AZURE_TENANT

To pass Active Directory username/password via the environment, define the following variables:

- AZURE_AD_USER
- AZURE_PASSWORD
- AZURE_SUBSCRIPTION_ID

To pass Active Directory username/password in ADFS via the environment, define the following variables:

- AZURE_AD_USER
- AZURE_PASSWORD
- AZURE_CLIENT_ID
- AZURE_TENANT
- AZURE_ADFS_AUTHORITY_URL

"AZURE_ADFS_AUTHORITY_URL" is optional. It's necessary only when you have own ADFS authority like https://yourdomain.com/adfs.

Storing in a File

When working in a development environment, it may be desirable to store credentials in a file. The modules will look for credentials in `$HOME/.azure/credentials`. This file is an ini style file. It will look as follows:

```
[default]
subscription_id=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
client_id=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
secret=xxxxxxxxxxxxxxxxx
tenant=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Note

If your secret values contain non-ASCII characters, you must URL Encode them to avoid login errors.

It is possible to store multiple sets of credentials within the credentials file by creating multiple sections. Each section is considered a profile. The modules look for the [default] profile automatically. Define AZURE_PROFILE in the environment or pass a profile parameter to specify a specific profile.

Passing as Parameters

If you wish to pass credentials as parameters to a task, use the following parameters for service principal:

- client_id
- secret
- subscription_id
- tenant

Or, pass the following parameters for Active Directory username/password:

- ad_user
- password
- subscription_id

Or, pass the following parameters for ADFS username/password:

- ad_user
- password
- client_id
- tenant
- adfs_authority_url

"adfs_authority_url" is optional. It's necessary only when you have own ADFS authority like https://yourdomain.com/adfs.

Other Cloud Environments

To use an Azure Cloud other than the default public cloud (for example, Azure China Cloud, Azure US Government Cloud, Azure Stack), pass the "cloud_environment" argument to modules, configure it in a credential profile, or set the "AZURE_CLOUD_ENVIRONMENT" environment variable. The value is either a cloud name as defined by the Azure Python SDK (for example, "AzureChinaCloud", "AzureUSGovernment"; defaults to "AzureCloud") or an Azure metadata discovery URL (for Azure Stack).

Creating Virtual Machines

There are two ways to create a virtual machine, both involving the azure_rm_virtualmachine module. We can either create a storage account, network interface, security group and public IP address and pass the names of these objects to the module as parameters, or we can let the module do the work for us and accept the defaults it chooses.

Creating Individual Components

An Azure module is available to help you create a storage account, virtual network, subnet, network interface, security group and public IP. Here is a full example of creating each of these and passing the names to the `azure.azcollection.azure_rm_virtualmachine` module at the end:

```yaml
- name: Create storage account
  azure.azcollection.azure_rm_storageaccount:
    resource_group: Testing
    name: testaccount001
    account_type: Standard_LRS

- name: Create virtual network
  azure.azcollection.azure_rm_virtualnetwork:
    resource_group: Testing
    name: testvn001
    address_prefixes: "10.10.0.0/16"

- name: Add subnet
  azure.azcollection.azure_rm_subnet:
    resource_group: Testing
    name: subnet001
    address_prefix: "10.10.0.0/24"
    virtual_network: testvn001

- name: Create public ip
  azure.azcollection.azure_rm_publicipaddress:
    resource_group: Testing
    allocation_method: Static
    name: publicip001

- name: Create security group that allows SSH
  azure.azcollection.azure_rm_securitygroup:
    resource_group: Testing
    name: secgroup001
    rules:
      - name: SSH
        protocol: Tcp
        destination_port_range: 22
        access: Allow
        priority: 101
        direction: Inbound

- name: Create NIC
  azure.azcollection.azure_rm_networkinterface:
    resource_group: Testing
    name: testnic001
    virtual_network: testvn001
    subnet: subnet001
    public_ip_name: publicip001
    security_group: secgroup001

- name: Create virtual machine
  azure.azcollection.azure_rm_virtualmachine:
    resource_group: Testing
    name: testvm001
    vm_size: Standard_D1
    storage_account: testaccount001
    storage_container: testvm001
    storage_blob: testvm001.vhd
    admin_username: admin
    admin_password: Password!
    network_interfaces: testnic001
    image:
      offer: CentOS
      publisher: OpenLogic
      sku: '7.1'
      version: latest
```

Each of the Azure modules offers a variety of parameter options. Not all options are demonstrated in the above example. See each individual module for further details and examples.

Creating a Virtual Machine with Default Options

If you simply want to create a virtual machine without specifying all the details, you can do that as well. The only caveat is that you will need a virtual network with one subnet already in your resource group. Assuming you have a virtual network already with an existing subnet, you can run the following to create a VM:

```
azure.azcollection.azure_rm_virtualmachine:
  resource_group: Testing
  name: testvm10
  vm_size: Standard_D1
  admin_username: chouseknecht
  ssh_password_enabled: false
  ssh_public_keys: "{{ ssh_keys }}"
  image:
    offer: CentOS
    publisher: OpenLogic
    sku: '7.1'
    version: latest
```

Creating a Virtual Machine in Availability Zones

If you want to create a VM in an availability zone, consider the following:

- Both OS disk and data disk must be a 'managed disk', not an 'unmanaged disk'.
- When creating a VM with the `azure.azcollection.azure_rm_virtualmachine` module, you need to explicitly set the `managed_disk_type` parameter to change the OS disk to a managed disk. Otherwise, the OS disk becomes an unmanaged disk.
- When you create a data disk with the `azure.azcollection.azure_rm_manageddisk` module, you need to explicitly specify the `storage_account_type` parameter to make it a managed disk. Otherwise, the data disk will be an unmanaged disk.
- A managed disk does not require a storage account or a storage container, unlike an unmanaged disk. In particular, note that once a VM is created on an unmanaged disk, an unnecessary storage container named "vhds" is automatically created.
- When you create an IP address with the `azure.azcollection.azure_rm_publicipaddress` module, you must set the `sku` parameter to `standard`. Otherwise, the IP address cannot be used in an availability zone.

Dynamic Inventory Script

If you are not familiar with Ansible's dynamic inventory scripts, check out :ref:`Intro to Dynamic Inventory <intro_dynamic_inventory>`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\scenario_guides\[ansible-devel][docs][docsite][rst][scenario_guides]guide_azure.rst`, line 291); ***backlink***

Unknown interpreted text role "ref".

The Azure Resource Manager inventory script is called azure_rm.py. It authenticates with the Azure API exactly the same as the Azure modules, which means you will either define the same environment variables described above in Using Environment Variables, create a `$HOME/.azure/credentials` file (also described above in Storing in a File), or pass command line parameters. To see available command line options execute the following:

```
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/inventory/azure_rm.py
$ ./azure_rm.py --help
```

As with all dynamic inventory scripts, the script can be executed directly, passed as a parameter to the ansible command, or passed directly to ansible-playbook using the -i option. No matter how it is executed the script produces JSON representing all of the hosts found in your Azure subscription. You can narrow this down to just hosts found in a specific set of Azure resource groups, or even down to a specific host.

For a given host, the inventory script provides the following host variables:

```
{
  "ansible_host": "XXX.XXX.XXX.XXX",
  "computer_name": "computer_name2",
  "fqdn": null,
  "id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/providers/Microsoft.Compute/virtualMachines/object-nam
  "image": {
    "offer": "CentOS",
    "publisher": "OpenLogic",
    "sku": "7.1",
    "version": "latest"
  },
  "location": "westus",
  "mac_address": "00-00-5E-00-53-FE",
  "name": "object-name",
  "network_interface": "interface-name",
  "network_interface_id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/providers/Microsoft.Network/networkI
  "network_security_group": null,
  "network_security_group_id": null,
  "os_disk": {
    "name": "object-name",
    "operating_system_type": "Linux"
  },
  "plan": null,
  "powerstate": "running",
  "private_ip": "172.26.3.6",
  "private_ip_alloc_method": "Static",
  "provisioning_state": "Succeeded",
  "public_ip": "XXX.XXX.XXX.XXX",
  "public_ip_alloc_method": "Static",
  "public_ip_id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/providers/Microsoft.Network/publicIPAddresse
  "public_ip_name": "object-name",
  "resource_group": "galaxy-production",
  "security_group": "object-name",
  "security_group_id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/providers/Microsoft.Network/networkSecu
  "tags": {
    "db": "mysql"
  },
  "type": "Microsoft.Compute/virtualMachines",
  "virtual_machine_size": "Standard_DS4"
}
```

Host Groups

By default hosts are grouped by:

- azure (all hosts)
- location name
- resource group name
- security group name
- tag key
- tag key_value
- os_disk operating_system_type (Windows/Linux)

You can control host groupings and host selection by either defining environment variables or creating an azure_rm.ini file in your current working directory.

NOTE: An .ini file will take precedence over environment variables.

NOTE: The name of the .ini file is the basename of the inventory script (in other words, 'azure_rm') with a '.ini' extension. This allows you to copy, rename and customize the inventory script and have matching .ini files all in the same directory.

Control grouping using the following variables defined in the environment:

- AZURE_GROUP_BY_RESOURCE_GROUP=yes
- AZURE_GROUP_BY_LOCATION=yes
- AZURE_GROUP_BY_SECURITY_GROUP=yes
- AZURE_GROUP_BY_TAG=yes
- AZURE_GROUP_BY_OS_FAMILY=yes

Select hosts within specific resource groups by assigning a comma separated list to:

- AZURE_RESOURCE_GROUPS=resource_group_a,resource_group_b

Select hosts for specific tag key by assigning a comma separated list of tag keys to:

- AZURE_TAGS=key1,key2,key3

Select hosts for specific locations by assigning a comma separated list of locations to:

- AZURE_LOCATIONS=eastus,eastus2,westus

Or, select hosts for specific tag key:value pairs by assigning a comma separated list key:value pairs to:

- AZURE_TAGS=key1:value1,key2:value2

If you don't need the powerstate, you can improve performance by turning off powerstate fetching:

- AZURE_INCLUDE_POWERSTATE=no

A sample azure_rm.ini file is included along with the inventory script in here. An .ini file will contain the following:

```
[azure]
# Control which resource groups are included. By default all resources groups are included.
# Set resource_groups to a comma separated list of resource groups names.
#resource_groups=

# Control which tags are included. Set tags to a comma separated list of keys or key:value pairs
#tags=

# Control which locations are included. Set locations to a comma separated list of locations.
#locations=

# Include powerstate. If you don't need powerstate information, turning it off improves runtime performance.
# Valid values: yes, no, true, false, True, False, 0, 1.
include_powerstate=yes

# Control grouping with the following boolean flags. Valid values: yes, no, true, false, True, False, 0, 1.
group_by_resource_group=yes
group_by_location=yes
group_by_security_group=yes
group_by_tag=yes
group_by_os_family=yes
```

Examples

Here are some examples using the inventory script:

```
# Download inventory script
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/inventory/azure_rm.py

# Execute /bin/uname on all instances in the Testing resource group
$ ansible -i azure_rm.py Testing -m shell -a "/bin/uname -a"

# Execute win_ping on all Windows instances
$ ansible -i azure_rm.py windows -m win_ping

# Execute ping on all Linux instances
$ ansible -i azure_rm.py linux -m ping

# Use the inventory script to print instance specific information
$ ./azure_rm.py --host my_instance_host_name --resource-groups=Testing --pretty

# Use the inventory script with ansible-playbook
$ ansible-playbook -i ./azure_rm.py test_playbook.yml
```

Here is a simple playbook to exercise the Azure inventory script:

```
- name: Test the inventory script
  hosts: azure
  connection: local
  gather_facts: no
  tasks:
    - debug:
        msg: "{{ inventory_hostname }} has powerstate {{ powerstate }}"
```

You can execute the playbook with something like:

```
$ ansible-playbook -i ./azure_rm.py test_azure_inventory.yml
```

Disabling certificate validation on Azure endpoints

When an HTTPS proxy is present, or when using Azure Stack, it may be necessary to disable certificate validation for Azure endpoints in the Azure modules. This is not a recommended security practice, but may be necessary when the system CA store cannot be altered to include the necessary CA certificate. Certificate validation can be controlled by setting the "cert_validation_mode" value in a credential profile, via the "AZURE_CERT_VALIDATION_MODE" environment variable, or by passing the "cert_validation_mode" argument to any Azure module. The default value is "validate"; setting the value to "ignore" will prevent all certificate validation. The module argument takes precedence over a credential profile value, which takes precedence over the environment value.