Microsoft Azure, formerly known as Windows Azure, is a cloud service from Microsoft that offers **Infrastructure as a Service**, (**IaaS**), **Platform as a Service** (**PaaS**), and **Software as a Service**(**SaaS**) through its more than 600 services.

Preparing Ansible to work with Azure

Ansible interacts with the Azure resource manager's REST APIs to manage infrastructure components using Python SDK provided by Microsoft Azure, which requires credentials of an authorized user to interact with Azure REST APIs.

Ansible packages cloud modules to interact with Azure resource manager. These modules require Azure Python SDK to interact with the Azure resource manager's APIs. We will now prepare our host to run Ansible using Azure cloud modules and put it together so that it can create and manage Azure resources:

1. We should start by installing Azure SDK on a host running Ansible:

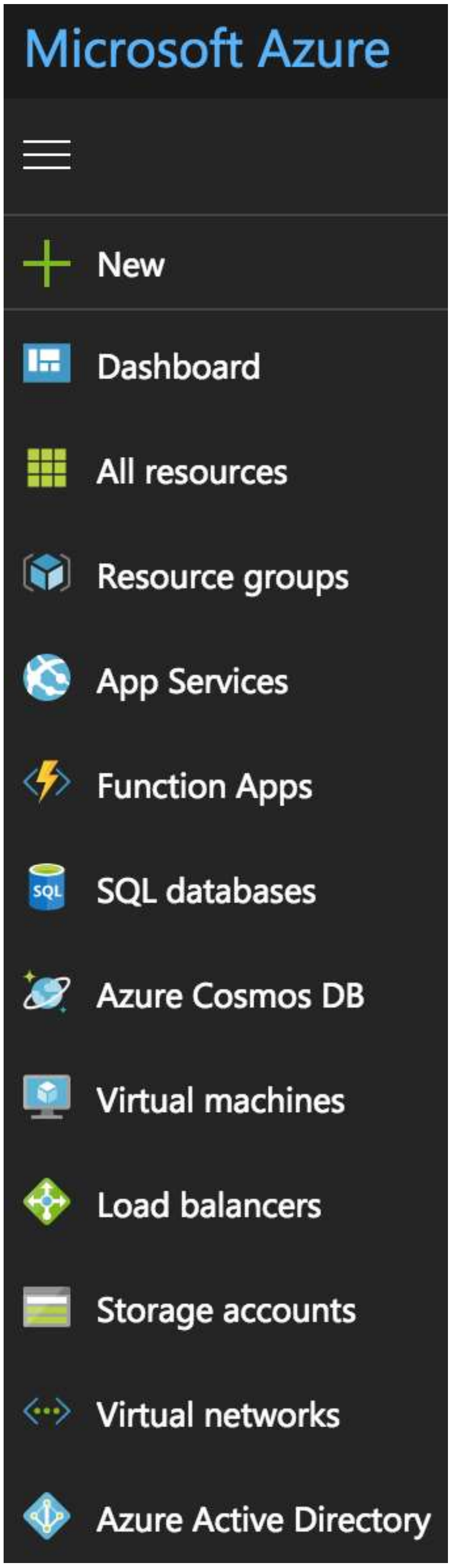
```
$ pip install ansible[azure]
```

2. We need to set up credentials for the Azure resource manager to interact with Azure APIs. Azure provides two ways to authenticate with Azure:

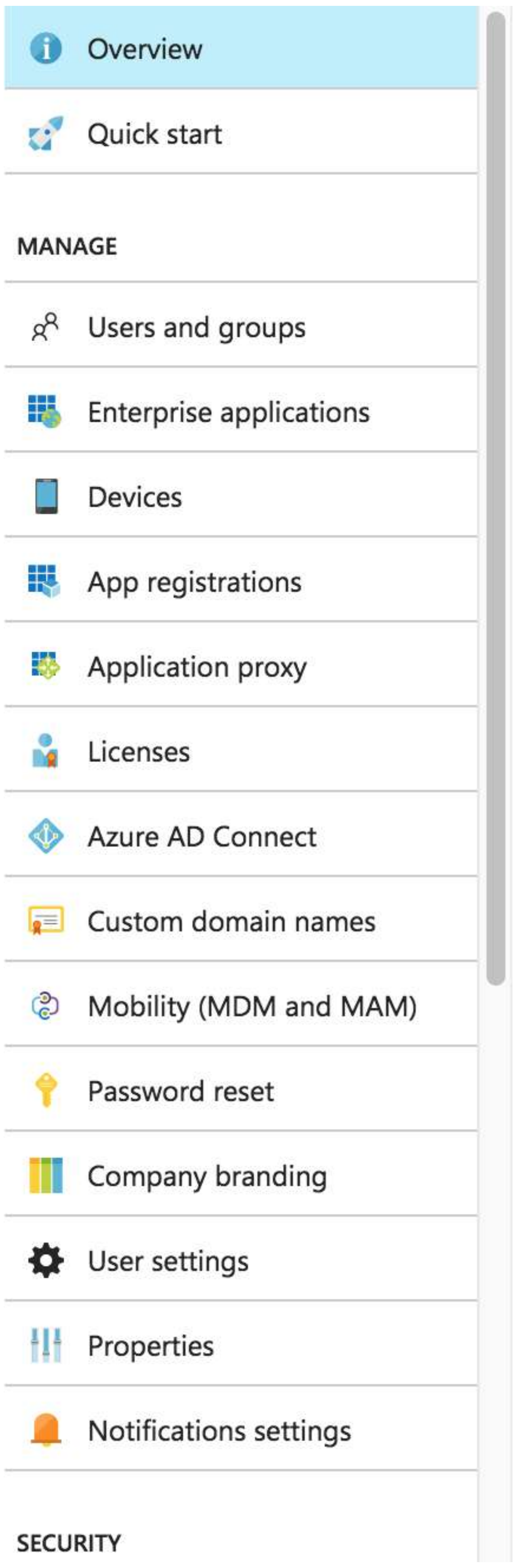
- Active Directory username and password
- Service principle credentials

Let's ensure that we have the correct permissions to create our application for Ansible:

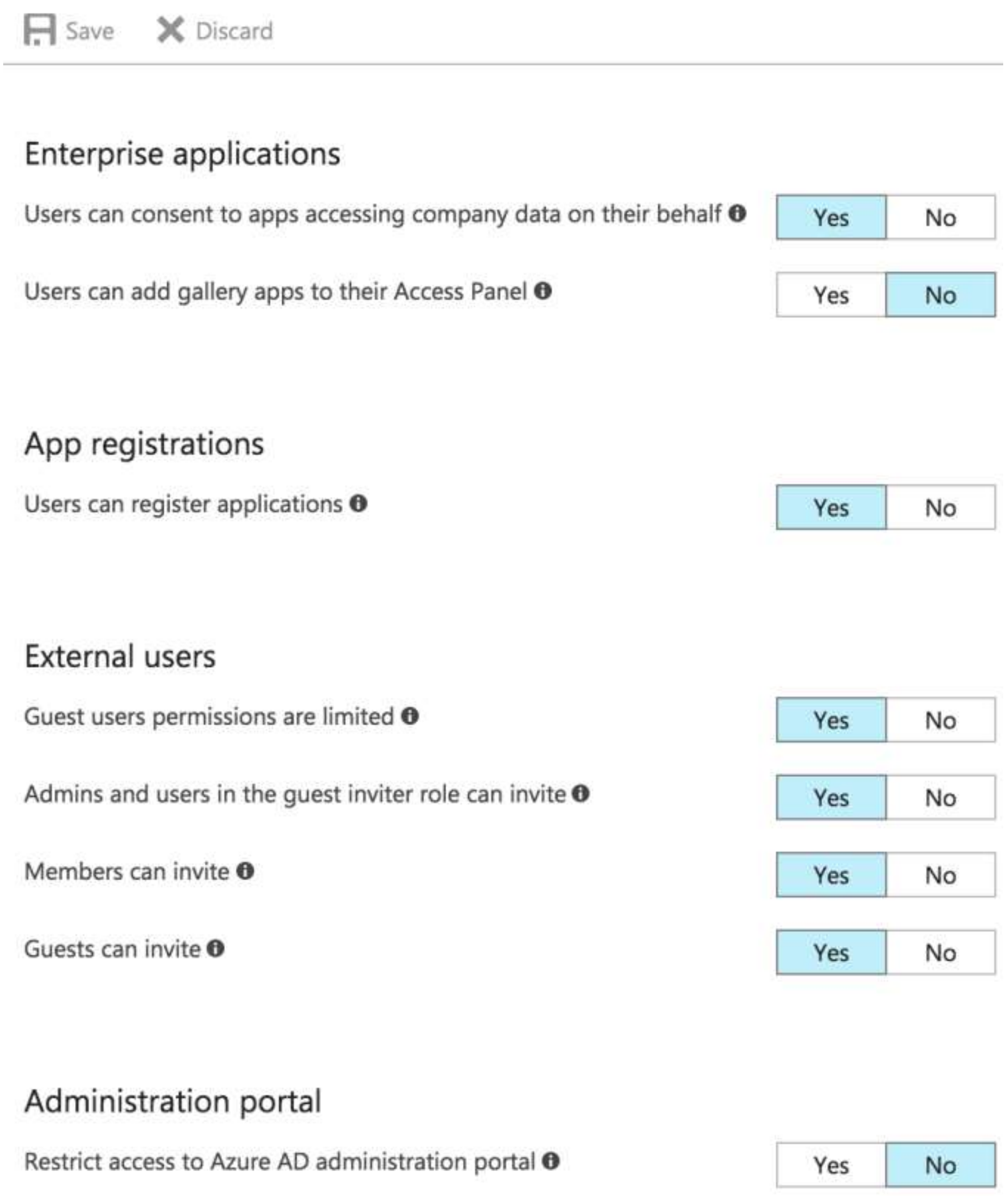
1. Log into the Azure portal and select Azure Active Directory:


Microsoft Azure
New
Dashboard
All resources
Resource groups
App Services
Function Apps
SQL databases
Azure Cosmos DB
Virtual machines
Load balancers
Storage accounts
Virtual networks
Azure Active Directory

2. Select User settings, as shown in the following screenshot:


Overview
Quick start
MANAGE
Users and groups
Enterprise applications
Devices
App registrations
Application proxy
Licenses
Azure AD Connect
Custom domain names
Mobility (MDM and MAM)
Password reset
Company branding
User settings
Properties
Notifications settings
SECURITY

3. Check whether the App Registrations setting is set to Yes. If it’s set to Yes, then anyone other than the admin can register an application and we can proceed. If it’s set to No, either we need to be the admin in order to register an application or we should get it enabled by the admin of the Azure account:


Save Discard

Enterprise applications

Users can consent to apps accessing company data on their behalf [✓ Yes] No

Users can add gallery apps to their Access Panel Yes [✓ No]

App registrations

Users can register applications [✓ Yes] No

External users

Guest users permissions are limited [✓ Yes] No

Admins and users in the guest inviter role can invite [✓ Yes] No

Members can invite [✓ Yes] No

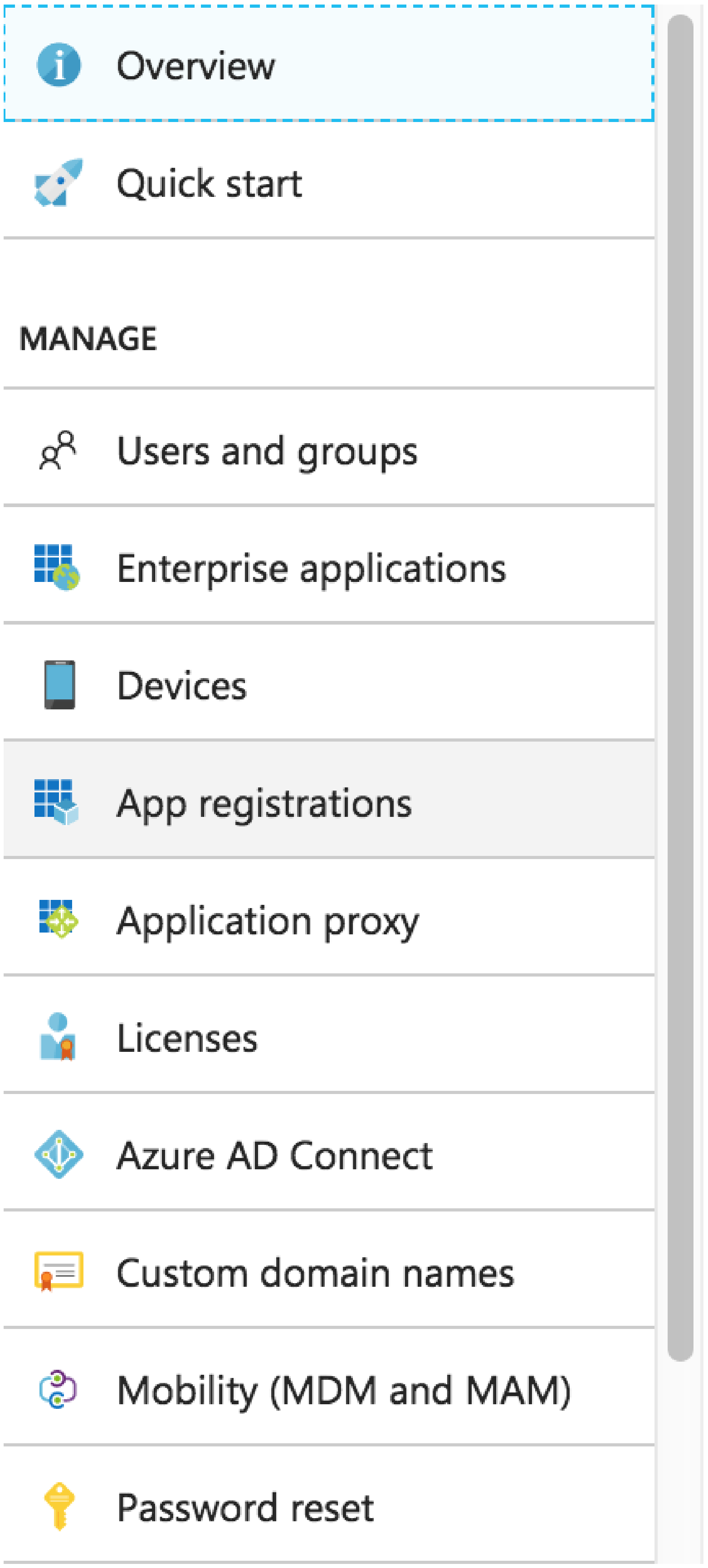
Guests can invite [✓ Yes] No

Administration portal

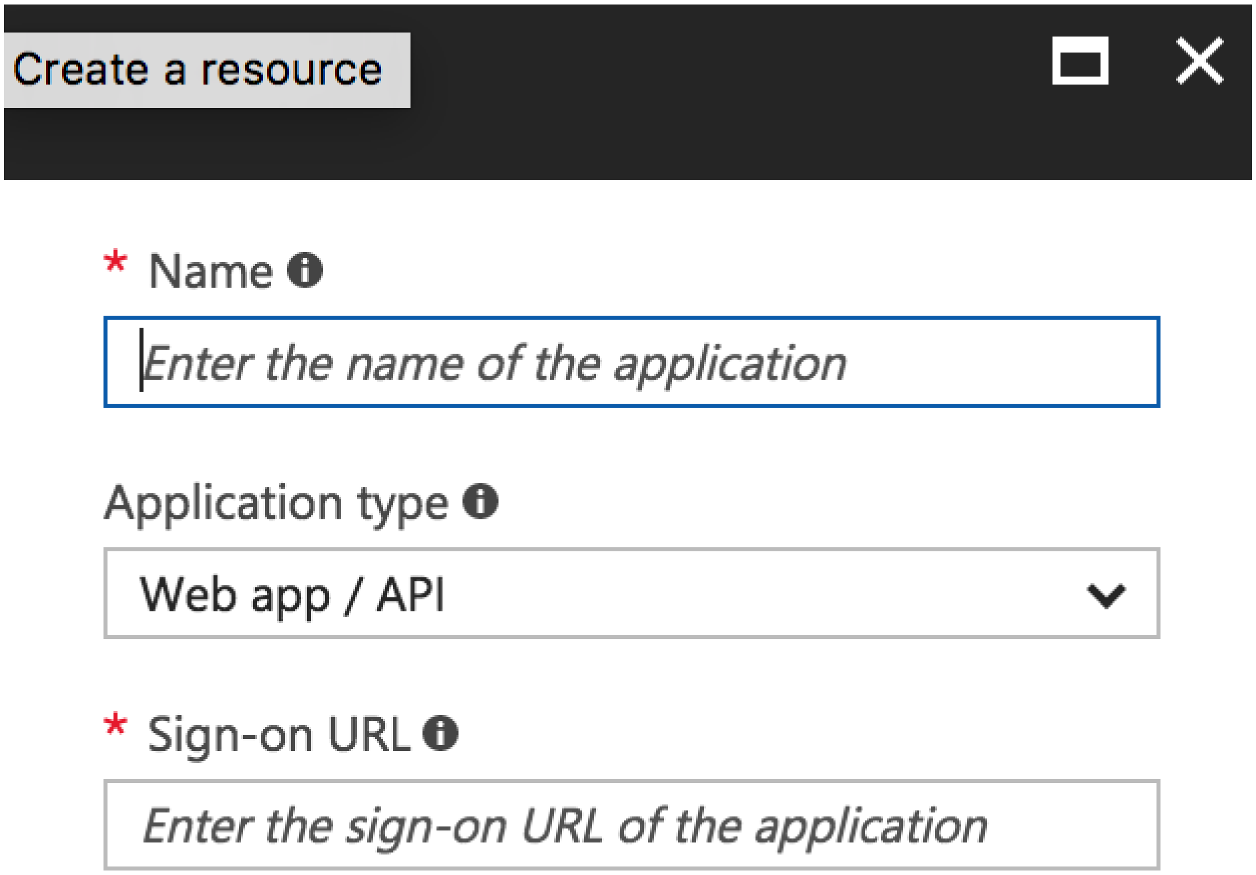
Restrict access to Azure AD administration portal Yes [✓ No]

Once we have the required permissions, we will register an application.

1. Select Azure Active Directory and select App Registrations:


Overview
Quick start
MANAGE
Users and groups
Enterprise applications
Devices
App registrations
Application proxy
Licenses
Azure AD Connect
Custom domain names
Mobility (MDM and MAM)
Password reset

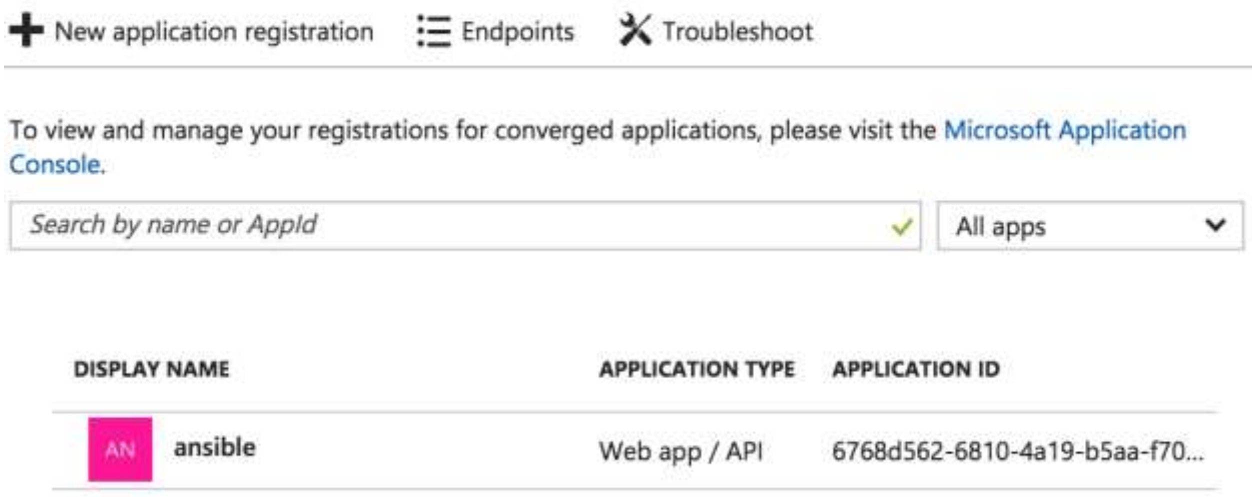
2. Select New Application Registrations and we will see a dialog box, as shown in the following screenshot:



Name: *Enter the name of the application*
Application type: Web app / API
Sign-on URL: *Enter the sign-on URL of the application*

3. Enter a name and URL for the application.

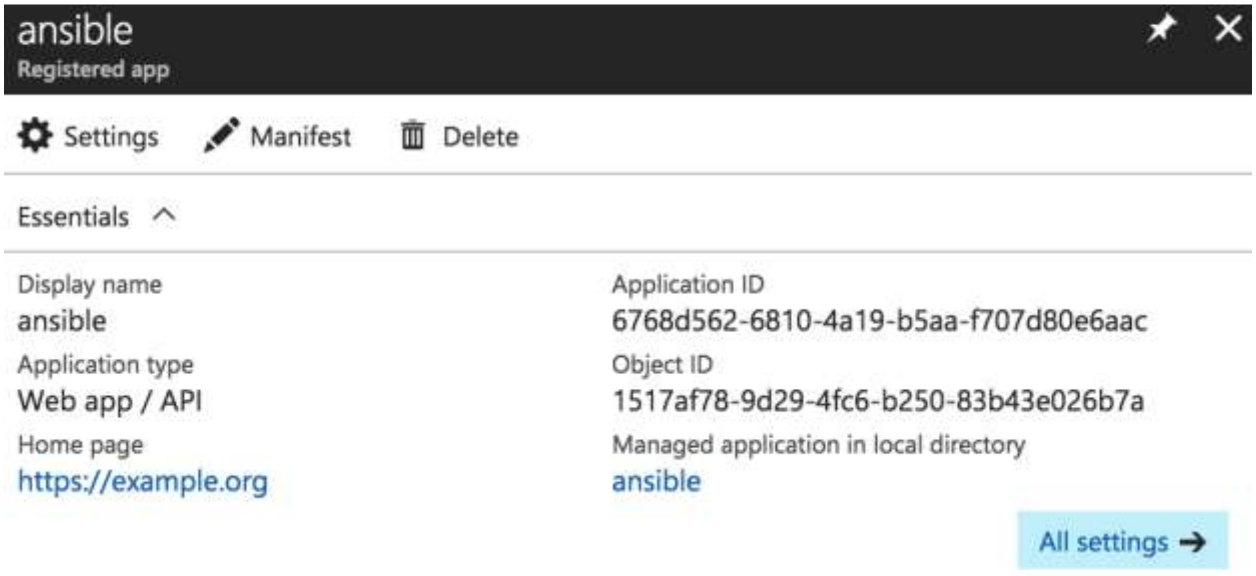
4. We will need the application ID and authorization key for our registered application. Navigate to App Registrations and select our application:



To view and manage your registrations for converged applications, please visit the Microsoft Application Console.

DISPLAY NAME	APPLICATION TYPE	APPLICATION ID
AN ansible	Web app / API	6768d562-6810-4a19-b5aa-f70...

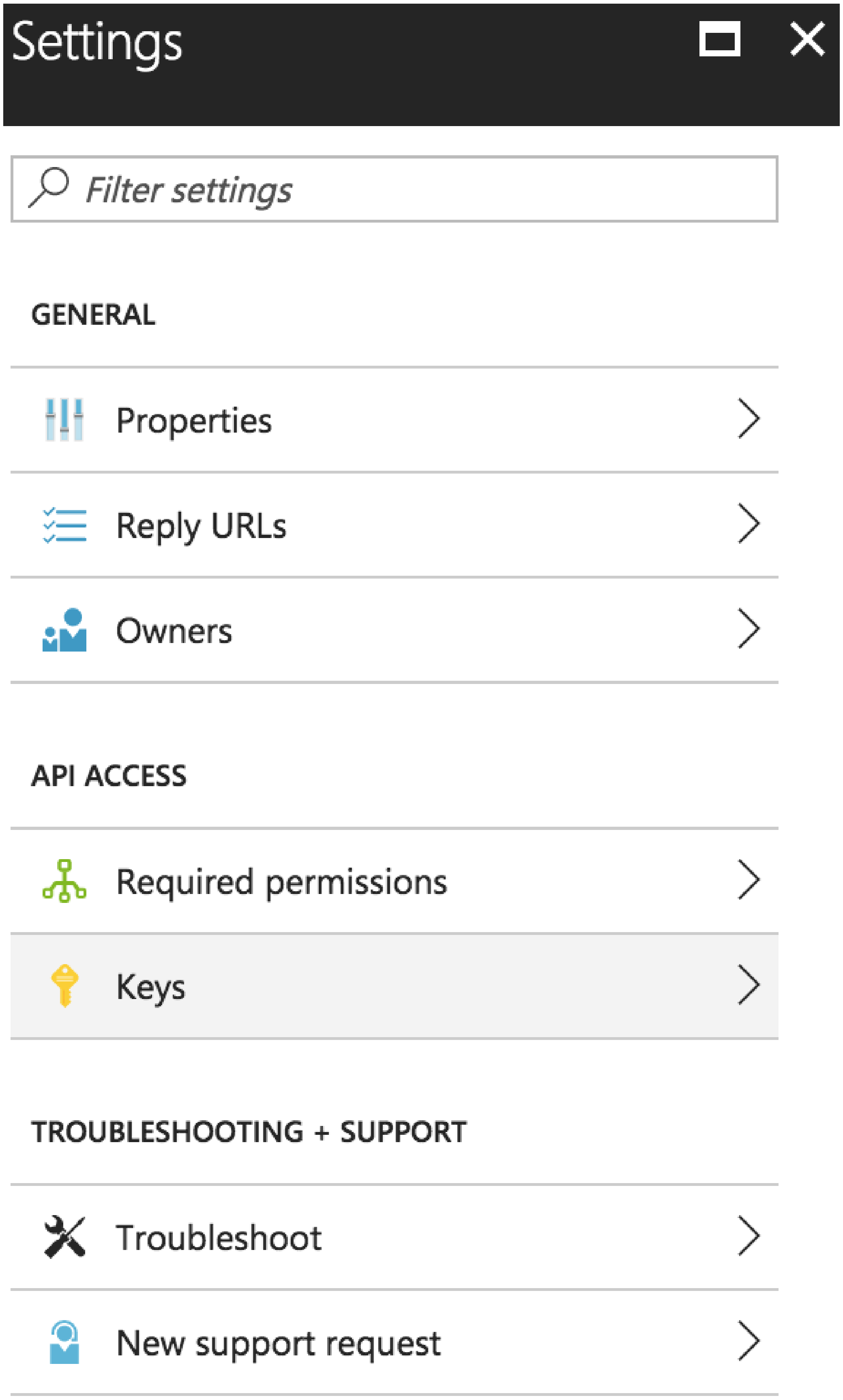
5. We can copy the Application ID:



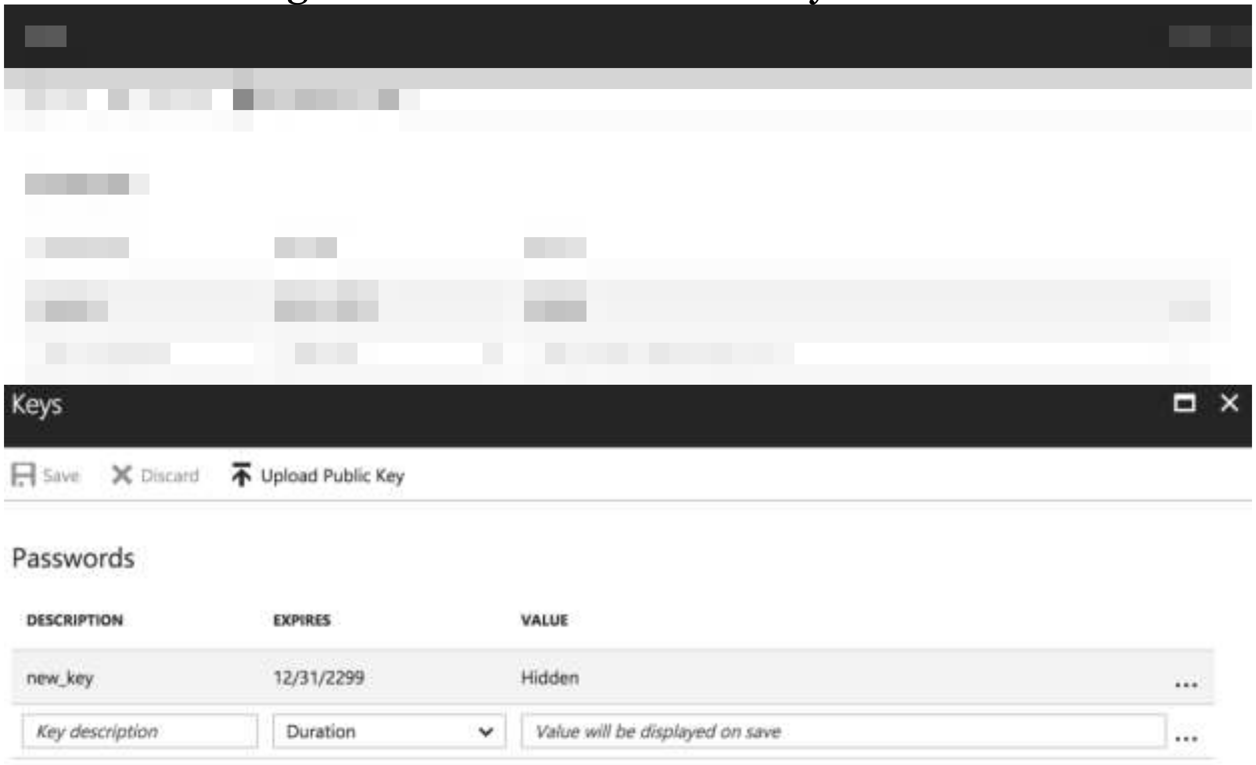
Display name
ansible
Application type
Web app / API
Home page
https://example.org

Application ID
6768d562-6810-4a19-b5aa-f707d80e6aac
Object ID
1517af78-9d29-4fc6-b250-83b43e026b7a
Managed application in local directory
ansible

6. Let’s generate an authorization key. Navigate to Keys:


Settings
Filter settings
GENERAL
Properties
Reply URLs
Owners
API ACCESS
Required permissions
Keys
TROUBLESHOOTING + SUPPORT
Troubleshoot
New support request

7. Enter the key’s name, select the expiration for this key, and note down the auto-generated authentication key:



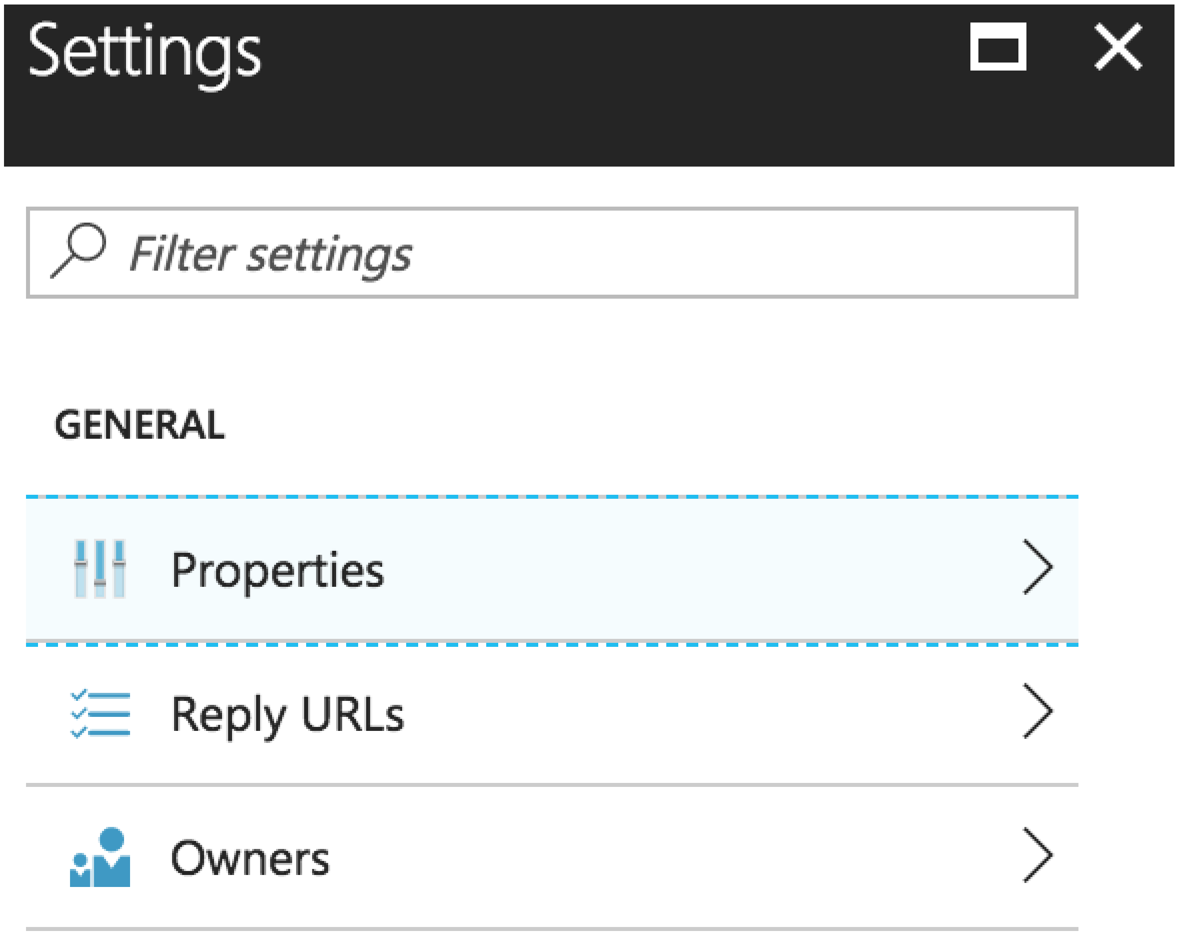
Keys

Passwords

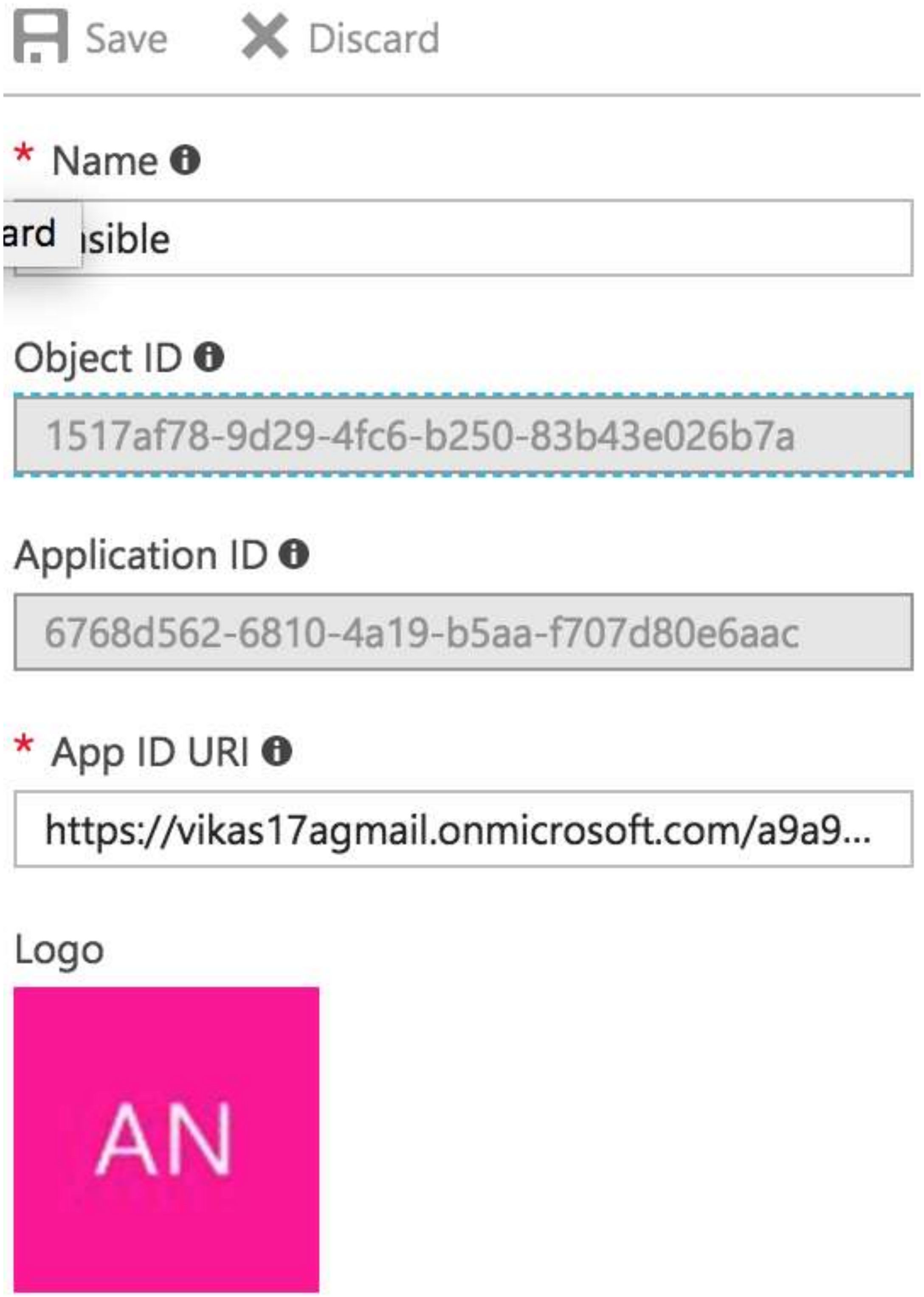
DESCRIPTION	EXPIRES	VALUE
new_key	12/31/2299	Hidden
Key description	Duration	Value will be displayed on save

While accessing the Azure resource manager, it also requires the tenant ID. Let’s look for the tenant ID.

1. Navigate to Azure Active Directory and select Properties to get the tenant ID:



2. The Application ID here is our tenant ID:



* Name

ard ısible

Object ID

1517af78-9d29-4fc6-b250-83b43e026b7a

Application ID

6768d562-6810-4a19-b5aa-f707d80e6aac

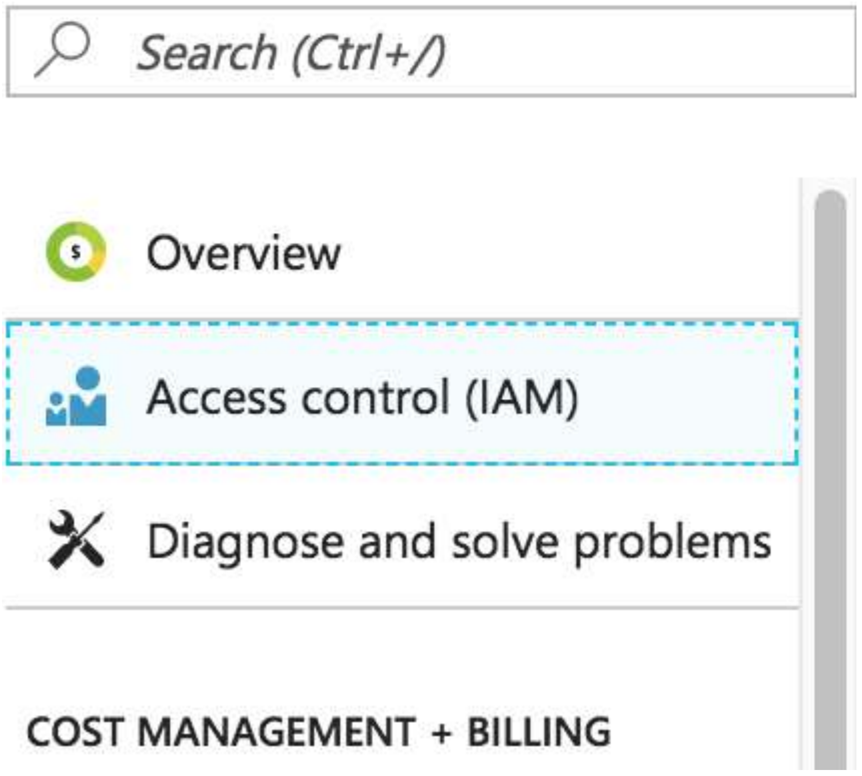
* App ID URI

https://vikas17agmail.onmicrosoft.com/a9a9...

Logo

Now our application is ready, but it needs access to create new resources. We will now assign a role to our application using Access Control (IAM).

1. Select Subscriptions and navigate to the current Azure Subscription | Access Control (IAM):



2. Add a new IAM control, using the role from a predefined list and by selecting Application under Resources. We can select roles or create one depending on what we want to achieve through Ansible automation. To keep this step simple, we are using the Owner role.

3. Once we have all the required credentials, we can create a credential file inside **.azure** in the **home** directory:

```
$ vim ~/.azure/credential
[default]
subscription_id=xxxxxx-xxxxx-xxxxxx-xxxx
client_id=xxxxxx-xxxx-xxxx-xxxxx
```

```
secret=xxxxxxxxxx
tenant=xxxxx-xxxx-xxx-xxx-xxx
```

Creating an Azure virtual machine

Before we jump into creating a Linux VM, we should know the following terms with respect to Azure:

1. **Resource groups**: These are logical containers where Azure resources are deployed. We can deploy resources into a specific resource group for a specific use case. For example, we can have resource groups named production for all the production resources and staging for all the resources required for staging.
2. **Image**: Azure Marketplace has various images for creating virtual machines. We can select an image of our choice, based on the use case, and create our virtual machine. In this, we will be using an Ubuntu Server image. There are four parameters linked to an image:

- **Offer** defines the distribution type. For example, Ubuntu, Debian, or CoreOS.
- **Publisher** refers to the organization that created the image. For example, canonical, credativ, or CoreOS.
- **SKU** defines the instance of the image offered. For example, 16.04-LTS for Ubuntu Server.
- **Version** defines the version of an image SKU. When defining an image, we can set the version to Latest to select the latest SKU of an image.

3. Storage account: The Azure Storage Account is a Microsoft-managed cloud service which provides scalable, highly available, and redundant storage. Azure Storage offers three kinds of storage:

- **Blob storage**: This stores our files in the same way that they are stored on local computers. For example, images, PDFs, and so on. The storage can consist of a **virtual hard disk** (**VHD**) format, large log files, and so on. This kind of storage is used for creating disks attached to a virtual machine.
- **File storage**: This is the network file share, which can be accessed through standard **Server Message Block** (**SMB**) protocol.
- **Queue storage**: This is a messaging storage system. A single queue can store millions of messages, and each message can be up to 64 KB in size.

4. Location: This is a region where we can deploy our resources in Azure Cloud. All of these will be deploying resources in the westus region. We will define this location as azure_region in group_vars for the playbook:

```
azure_region: westus
```

To do this, follow the steps below:

1. Create a resource group for deploying resources:

```
- name: Create resource group
  azure_rm_resourcegroup:
     name: example
     location: "{{azure_region}}"
     tags:
        env: testing
  tags:
    - recipe4
```

2. Create a storage account for our VM disk:

```
- name: Create a storage account
  azure_rm_storageaccount:
     resource_group: example
     name: examplestorage01
     type: Standard_RAGRS
     location: "{{azure_region}}"
     tags:
       - env: testing
  tags:
    - recipe4
```

3. Let’s create our first VM in Azure Cloud:

```
- name: Create VM
  azure_rm_virtualmachine:
    resource_group: example
    name: first_vm
    location: "{{azure_region}}"
    vm_size: Standard_D4
    storage_account: examplestorage01
    admin_username: cookbook
    ssh_public_keys:
      - path: /home/admin/.ssh/authorized_keys
         key_data: "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDq8ddP3LGDr586Njl9lqScZvakv4DvGPsK9PN
Cw+MWaLZsSovUECLm1v3IxfBhbGUrbQMFAbff0Piie9+6aas5vSFaqn1LMhEyVNjJka
Faztg/FiYbhcSzb4zc7hrKyZriUyou2gj68o9113g38wh0tK6TSjfQ+DrN2HiV8bo4j
LYmGnh+A3O6HMWR1ceCclN5c3g4RRjrLzSC9YolufMDLzs4CWxjEDLufYwsPqafOrvc
XUlLeAzfjYrG8Re82sH6uE8Zw1WHRDk9hhRZU8s5jFCtepLeHL0jgftMXHGEP7F/cFX
Zb9KzdO1sqIie7OMfQ44hAPAcA1KexEPt6gb1"
     image:
       offer: UbuntuServer
       publisher: Ubuntu
       sku: '16.04-LTS'
       version: latest
  tags:
    - recipe4
```

1. **In *step 1:*** we created a resource group for deploying resources in our defined location. We will be using this resource group name in subsequent tasks to deploy all of the resources in this resource group.

2. **In *step 2:*** we created a storage account, which will be required for the OS disk of our virtual machine. Azure offers multiple storage types depending upon the use case and availability- **Locally Redundant Storage** (**LRS**), **Geo-Redundant Storage** (**GRS**), **Zone-Redundant Storage** (**ZRS**), or **Read Access Geo-Redundant Storage** (**RAGRS**). We are using **Standard_RAGRS**.
3. **In *step 3:*** we created our first virtual machine in Azure Cloud. This task will take care of setting up an admin user and deploying its public keys at the required places. We have set the admin_username as cookbook. Once our virtual machine is ready, we can connect to our virtual machine using the SSH protocol and the private key associated with the public key we used in the task.

Managing network interfaces

An Azure virtual machine can be attached to multiple network interface cards. With a network interface card, the virtual machine can access the internet and communicate with other resources both inside and outside Azure Cloud. In the, *Creating an Azure virtual machine*, while creating a virtual machine, it creates a default NIC card for the VM with default configurations. In this, we will see how to create, delete, and manage a NIC with custom settings.

Before we move ahead and create a NIC, we should be aware of the following term:

1. **Virtual network**: This is a logical separation of the network within the Azure Cloud. A virtual network can be assigned a CIDR block. Any virtual network has the following attributes associated with it:

- Name
- Address Space
- Subnet

To do this, follow the steps below:

1. Create a virtual network:

```
- name: Create Virtual Network
  azure_rm_virtualnetwork:
    name: vnet01
    resource_group: example
    address_prefixes_cidr:
       - "10.2.0.0/16"
       - "172.1.0.0/16"
    tags:
       env: testing
  state: present
  tags:
    - recipe2
```

2. Create a subnet:

```
- name: Create subnet
  azure_rm_subnet:
    name: subnet01
    virtual_network_name: my_first_subnet
    resource_group: example
    address_prefix_cidr: "10.2.0.0/24"
    state: present
  tags:
    - recipe2
```

3. Create a Network Interface Card:

```
- name: Create network interface card
  azure_rm_networkinterface:
    name: nic01
    resource_group: example
    virtual_network_name: vnet01
    subnet_name: subnet01
    public_ip: no
    state: present
  register: network_interface
  tags:
    - recipe2
```

4. Access the private IP address:

```
- name: Show private ip
  debug:
    msg:
"{{network_interface.ip_configuration.private_ip_address}}"
  tags:
    - recipe2
```

1. **In *step 1:*** we created a virtual network with the name vnet01 within the same resource group we used in the first, *Creating an Azure virtual machine*. Since we are using a resource group, the resources can pick the default location of the resource group. We have defined the CIDR network addresses as 10.2.0.0/24 and 172.1.0.0/16. We can also define multiple network addresses using YAML syntax.

2. **In *step 2:*** we created a subnet using the **azure_rm_subnet** module inside the virtual network **vnet01**.

3. **In *step 3:*** we created a network interface card using **azure_rm_networkinterface** and named it **nic01**. We specified the **public_ip** as no, which will ensure that Azure will not allocate any public IP address to the NIC, but also that NIC will be associated with a private IP from the subnet address space.

4. **In *step 4:*** we use the debug module to print the private IP address allocated to the network interface using the variable registered in *step 3*.

Working with public IP addresses

In this, we will create a public IP address and associate it with the network interface.

Azure allocates the public IP address using one of two methods, static or dynamic. An IP address allocated with the static method will not change, irrespective of the power cycle of the virtual machine; whereas, an IP address allocated with the dynamic method is subject to change. In this, we will create a public IP address allocated with the Static method.

To do this, follow the steps below:

1. Create a public IP:

```
- name: Create Public IP address
  azure_rm_publicipaddress:
    resource_group: example
    name: pip01
    allocation_method: Static
    domain_name: test
    state: present
  register: publicip
  tag:
    - recipe3
```

2. Display a public IP:

```
- name: Show Public IP address
  debug:
     msg: "{{ publicip.ip_address }}"
  tag:
    - recipe3
```

Using public IP addresses with network interfaces and virtual machines

In this, we will create a virtual machine and network interface using the public IP we created. After creating a virtual machine with the public network interface, we will log into it using SSH.

1. Create a NIC with the existing public IP address:

```
- name: Create network interface card using existing public ip
address
  azure_rm_networkinterface:
    name: nic02
    resource_group: example
    virtual_network_name: vnet01
    subnet_name: subnet01
    public_ip_address_name: pip01
    state: present
  register: network_interface02
  tags:
    - recipe4
```

2. Create a virtual machine with the existing network interface:

```
- name: Create VM using existing virtual network interface card
  azure_rm_virtualmachine:
    resource_group: example
    name: first_vm
    location: "{{azure_region}}"
    vm_size: Standard_D4
    storage_account: examplestorage01
    admin_username: cookbook
    ssh_public_keys:
      - path: /home/admin/.ssh/authorized_keys
        key_data: "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDq8ddP3LGDr586Njl9lqScZvakv4DvGPsK9PN
Cw+MWaLZsSovUECLm1v3IxfBhbGUrbQMFAbff0Piie9+6aas5vSFaqn1LMhEyVNjJka
Faztg/FiYbhcSzb4zc7hrKyZriUyou2gj68o9113g38wh0tK6TSjfQ+DrN2HiV8bo4j
LYmGnh+A3O6HMWR1ceCclN5c3g4RRjrLzSC9YolufMDLzs4CWxjEDLufYwsPqafOrvc
XUlLeAzfjYrG8Re82sH6uE8Zw1WHRDk9hhRZU8s5jFCtepLeHL0jgftMXHGEP7F/cFX
Zb9KzdO1sqIie7OMfQ44hAPAcA1KexEPt6gb1"
     image:
       offer: UbuntuServer
       publisher: Ubuntu
       sku: '16.04-LTS'
       version: latest
     network_interfaces: nic02
  tags:
    - recipe4
```

2. Log into the VM using the public IP from the recipe, *Working with public IP addresses*:

```
$ ssh -i ~/.ssh/cookpook.pem cookbook@13.33.23.24
```

In the first two steps, we created a NIC with an existing public IP, created in the recipe *Working with public IP addresses*, and a virtual machine using that NIC.

In *step 3*, we logged into the virtual machine created with the public NIC.

Managing an Azure network security group

In Azure, a network security group is an **access control list** (**ACL**), which allows and denies network traffic to subnets or an individual NIC. In this recipe, we will create a network security group with some basic rules for allowing web and SSH traffic and denying the rest of the traffic.

Since a network security group is the property of the network and not the virtual machine, we can use subnets to group our virtual machines and keep them in the same network security group for the same ACL.

1. To Create a network security group:

```
- name: Create network security group
  azure_rm_securitygroup:
    resource_group: example
    name: mysg01
    purge_rules: yes
    rules:
        - name: 'AllowSSH'
          protocol: TCP
          source_address_prefix: *
          destination_port_range: 22
          access: Allow
          priority: 100
          direction: Inbound
        - name: 'AllowHTTP'
          protocol: TCP
          source_address_prefix: *
          destination_port_range: 80
          priority: 101
          direction: Inbound
        - name: 'AllowHTTPS'
          protocol: TCP
          source_address_prefix: *
          destination_port_range: 443
          priority: 102
          direction: Inbound
        - name: 'DenyAll'
          protocol: TCP
          source_address_prefix: *
          destination_port_range: *
          priority: 103
          direction: Inbound
  tags:
     - recipe5
```

2. Attach the subnet to the security group:

```
- name: Create subnet
  azure_rm_subnet:
    name: subnet01
    virtual_network_name: my_first_subnet
    resource_group: example
    address_prefix_cidr: "10.2.0.0/24"
    state: present
    security_group_name: mysg01
  tags:
     - recipe5
```

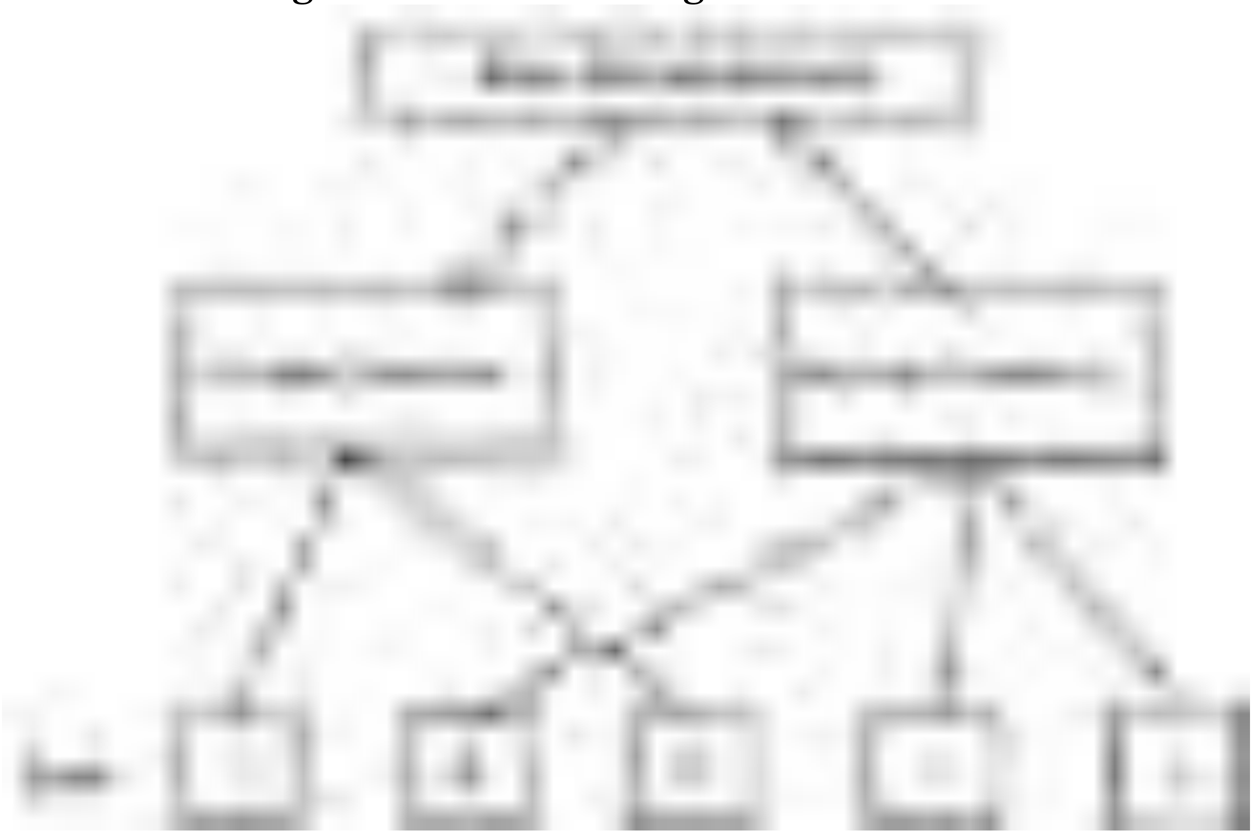
3. Attach the NIC card to the security group:

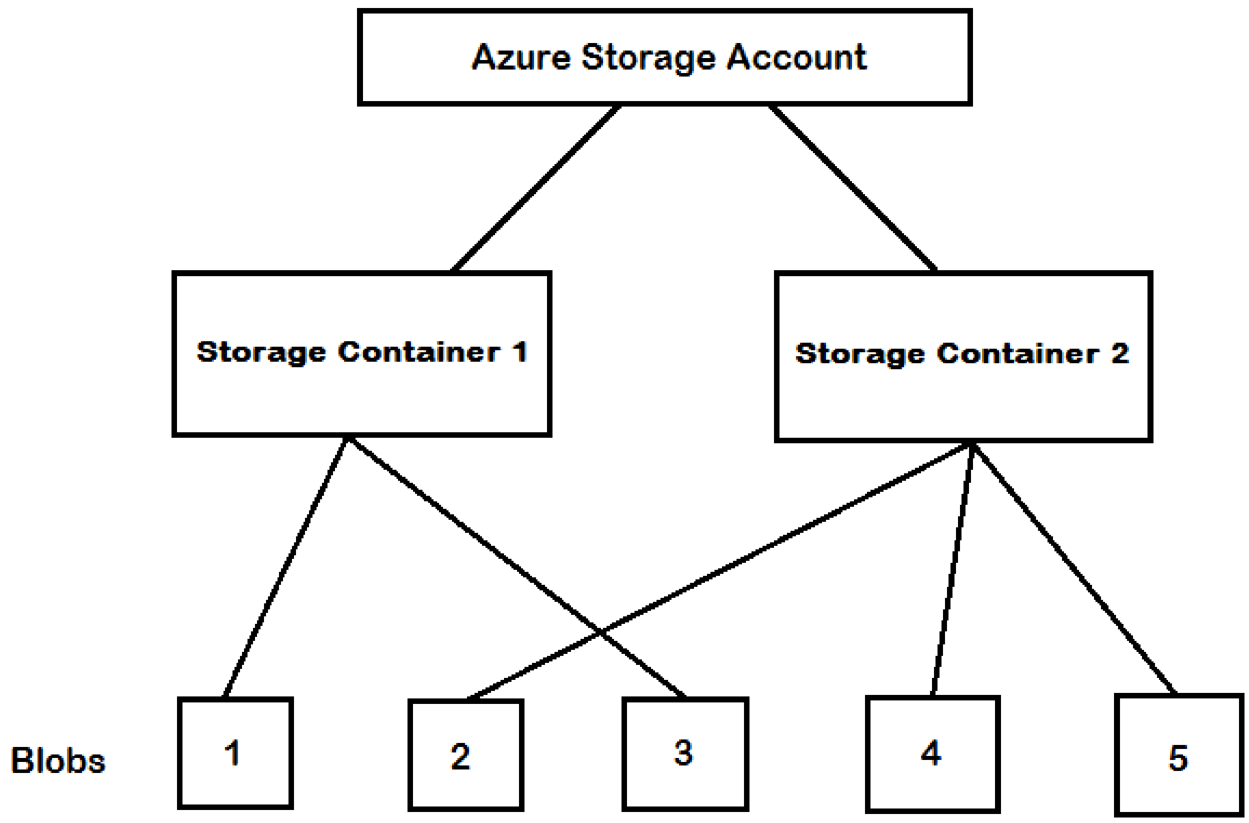
```
- name: Create network interface card using existing public ip
address and   security group
  azure_rm_networkinterface:
    name: nic02
    resource_group: example
    virtual_network_name: vnet01
```

```
      subnet_name: subnet01
      public_ip_address_name: pip01
      security_group_name: mysg01
      state: present
    register: network_interface02
    tags:
       - recipe5
```

Working with Azure Blob storage

The Azure Storage Account offers Blob storage, which operates in the same way as files stored on our local systems, such as pictures, videos, PDFs, and so on. In this recipe, we will learn how to use the Azure Storage Account for storing blobs.





- The **Azure Storage Account** is an access point through which Azure storage can be accessed.
- The **Storage Container** is a set of specific storage blobs. Every blob needs to be part of a container.
- The **Blobs** is a file that could be anything, such as a PDF, image, video, and so on.

1. Create a storage container:

```
- name: Create storage container
  azure_rm_storageblob:
     resource_group: example
     storage_account_name: examplestorage01
     container: cookbook
     state: present
```

2. Add a blob to the container:

```
- name: Upload a file to existing container
  azure_rm_storageblob:
     resource_group: example
     storage_account_name: examplestorage01
     container: cookbook
     blob: myfile.png
     src: /tmp/myfile.png
     public_access: blob
     content_type: 'application/image'
```

3. Download the uploaded file:

```
- name: Download blob object
  azure_rm_storageblob:
     resource_group: example
     storage_account_name: examplestorage01
     container: cookbook
     blob: myfile.png
     dest: /tmp/download_file.png
```

1. **In *step 1:*** we created an Azure Storage Container and named it **cookbook**.
2. **In *step 2:*** we uploaded an image file with the PNG format, located at **/tmp/myfile.png** on the host running the Ansible task, to the cookbook container. We also set the **public_access** parameter as a **blob**. We granted this blob the public read permission.
3. **In *step 3:*** we downloaded the same file we just uploaded to the **/tmp/download_file.png** location on the Ansible host.

Using a dynamic inventory

In this, we will use dynamic inventory to target hosts and build an Ansible inventory using the Azure Resource Manager API. Dynamic inventory enables us to leverage its dynamic nature by not hardcoding the host's addresses in a static inventory file.

Ansible can target the hosts present in a group, and by default, the Azure inventory creates the following groups:

- **azure (all hosts):** This will contain all hosts, and can be accessed using hosts: azure.
- **location name:** This will group hosts by their location-for example, westus, eastus, and so on. This can be accessed using hosts: westus.
- **name of resource group:** This will group hosts deployed in a resource group. It can be accessed using hosts: example.
- **security group name:** This will group hosts deployed with the same security groups. For example, mysg01. It can be accessed using hosts: mysg01.
- **tag key:** This will group hosts by their respective tag key-for example, env. It can be accessed by hosts: env.
- **tag key_value:** This will group hosts by their tag's key and value. For example, a host may have the tag key envand the value testing. It can be accessed using hosts: tag_env_testing.

We can define a few other parameters during configuration to restrict hosts in the inventory,. We could define specific groups and filters, such as resource groups, using the following inventory configuration file:

```
[azure]

# Controls resource groups to include in inventory.
#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.
 include_powerstate=yes # Control grouping with the following
boolean flags.
```

```
group_by_resource_group=yes
group_by_location=yes
group_by_security_group=yes
group_by_tag=yes
```

1. Use Ansible dynamic inventory with the following ad hoc command:

```
#Target all hosts in azure group
$ ansible -i azure_rm.py azure -m ping

#Target all hosts in example resource group
$ ansible -i azure_rm.py example -m ping
```

2. Use ansible-playbook with dynamic inventory:

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

# Content of playbook.yml
$ cat playbook.yml
---
- host: tag_env_preprod
  tasks:
     - name: Test Dynamic Inventory
       shell:  /bin/uname -a
```

3. Use Azure tags to target hosts in Ansible:

```
$ ansible tag_env_preprod -m ping
```

Deploying a sample application

We will now deploy a simple phonebook application in Azure Cloud. In this application, we will create a resource group, a virtual network (phonebook-vnet01), a subnet (phonebook-net01), a security group (phonebook) allowing HTTP and SSH traffic, a network interface with a public IP, and finally a virtual machine.

We create and save our playbook as phonebook.yml:

```
#Playbook for deploying phonebook application in Azure Cloud
---
```

```
- hosts: tag_Name_first_vm
  gather_facts: no
  roles:
    - phonebook
```

We should note that the hosts in this playbook are set as tag_Name_first_vm, which will create an inventory for our phonebook application at runtime. We can execute our playbook using the following command:

```
$ ansible-playbook -i azure-rm.py --become phonebook.yml
```

We used a command-line inventory argument for our playbook, since we are creating our resources in the playbook itself. After successful completion of this playbook, we will be able to access our application on port 8000 and the host IP of the virtual machine created in this playbook, which will be displayed in the last task using the debug module.