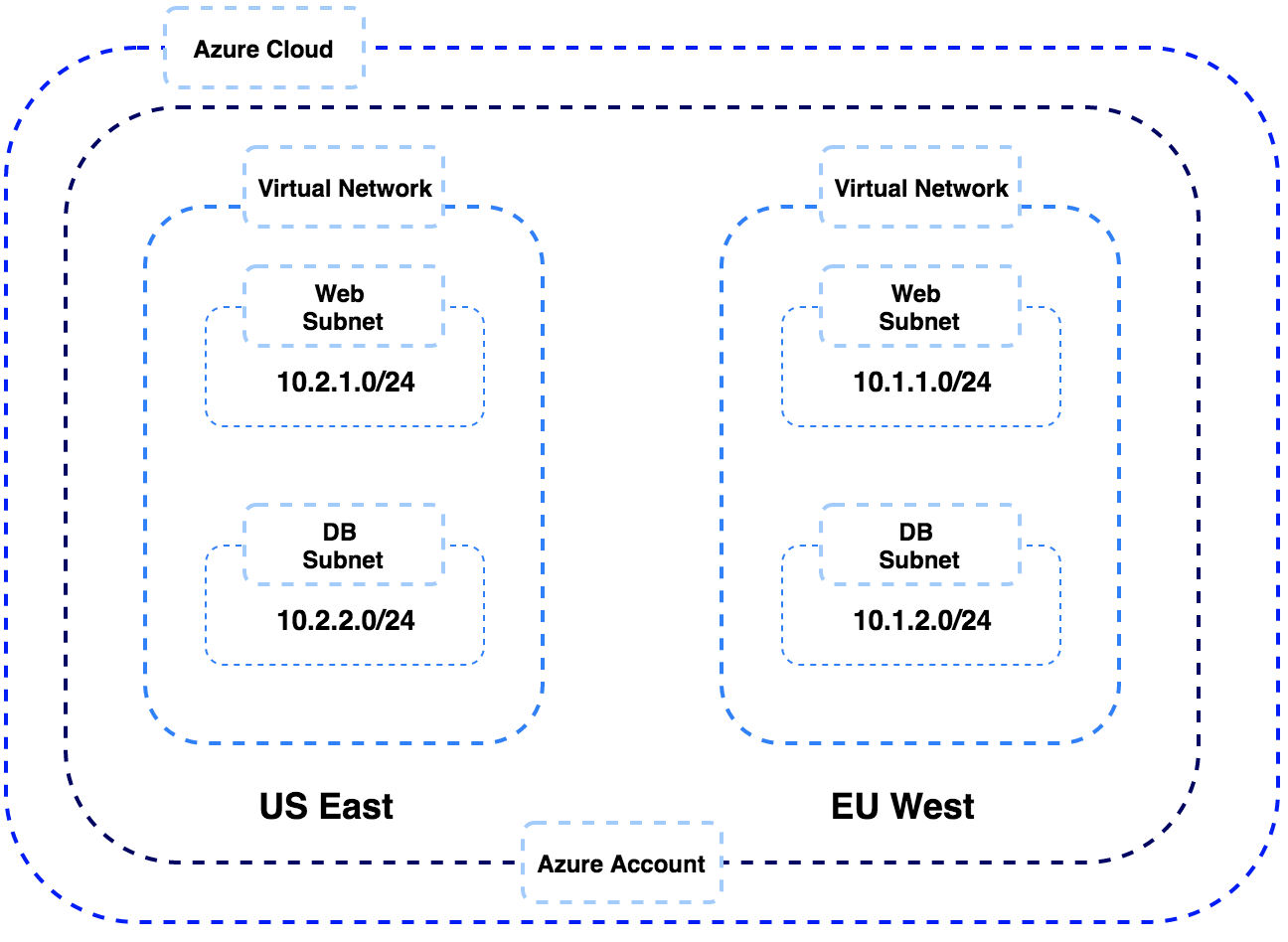
8.Deploying and operating Azure networking resources with Ansible

In the previous chapter we explored how to provision network resources on AWS cloud and how to use Ansible as the orchestration engine to deploy these resources on AWS. In this chapter we will look into the other major cloud provider which is Microsoft and its Azure cloud offering.

Azure provides multiple networking services to facilitate the deployment of highly scalable cloud solutions on Azure cloud. Ansible provides multiple modules to interact with multiple services in Azure cloud and is an excellent tool to automate cloud deployments on Azure cloud. We will explore the basic networking constructs available in Azure and we will outline how to use several modules in ansible to build and validate the below basic Network setup in Azure cloud.



The main recipes covered in this chapter is shown below

* Installing Azure SDK
* Authenticating to your Azure Account
* Building Ansible Inventory.
* Deploy Resource Groups  
  Deploying Virtual Networks.
* Deploying Subnets.
* Building User Defined Routes
* Deploying Network Security Groups.
* Deployment Validation using Ansible.
* Decommissioning Azure Resources with Ansible

# Technical Requirements

In order to start working with Azure we need to create an Account, you can setup a Free account using the below URL to setup a Free Azure Account

<https://azure.microsoft.com/en-au/free/>

Below is the GitHub code used in this chapter

[https://github.com/PacktPublishing/Network-Automation-Cookbook/tree/master/ch8\_a](https://github.com/PacktPublishing/Network-Automation-Cookbook/tree/master/ch7_aws)zure

Here are the Software releases that this chapter is based on

· Ansible Machine Running CentOS 7.7

· Ansible 2.8.4

· Python 3.7.4.

# Installing Azure SDK

In this recipe we outline how to install the required python libraries needed to start interacting with Azure orchestration system using ansible. This step is mandatory since the python library must be installed on the ansible control machine in order for all the ansible Azure modules to work.

## Getting Ready

You need to have sudo access on the machine in order to install the aws python library as well as have python already installed and python PIP package which we will use to install the aws package.

## How to do it..

1. Install the boto3 package as shown below

$ sudo pip3 install 'ansible[azure]'

1. Create a new folder ch8\_azure to host all the code for this chapter

$ mkdir ch8\_azure

## How it works..

The default installation of ansible doesn’t include all the python modules needed to run the ansible azure modules, that is why our first step is to install these required python libraries. We install all these packages using the python pip. We can verify all the azure modules installed as shown below

$ pip list | grep azure

azure-cli-core 2.0.35

azure-cli-nspkg 3.0.2

azure-common 1.1.11

azure-graphrbac 0.40.0

azure-keyvault 1.0.0a1

**<< --- Output Omitted for brevity -- >>**

As outlined above multiple python packages are installed in order to start to interact with Azure API from ansible. With this step we are ready to build our playbooks to build our infrastructure in Azure.

**See Also..**

For more information regarding how to start interacting with Azure cloud using ansible please refer to the below URL .

<https://docs.ansible.com/ansible/latest/scenario_guides/guide_azure.html>

# Building Ansible Inventory

In this recipe, we will outline how to build an Ansible inventory to describe the network infrastructure setup that we will build across the Azure public cloud. This is a mandatory step in order to define all our Virtual Networks across all the regions that we will deploy our infrastructure in.

## How to do it..

1. Create the hosts file inside ch8\_azure directory with the below data

$ cat hosts

[az\_net]

eu\_az\_net

us\_az\_net

[eu]

eu\_az\_net

[us]

us\_az\_net

1. Create the ansible.cfg file with the below content

$ cat ansible.cfg

[defaults]

inventory=./hosts

retry\_files\_enabled=False

gathering=explicit

host\_key\_checking=False

action\_warnings=False

1. Create group\_vars folder and create the eu.yml and us.yml files as shown below

$ cat group\_var/eu.yml

---

region: westeurope

$ cat group\_var/us.yml

---

region: eastus

## How it works..

We created the hosts ansible inventory file and we declare the different virtual networks that we will provision in Azure cloud. We also create two groups which describe the location of each virtual network.

In short, we created the below groups to define and group our virtaul\_networks

* **Az\_net**, this group all our virtual networks across Azure cloud.
* **EU**, this group list all virtual networks in EU region (this will map to a specific region in Azure cloud as we will outline)
* **US**, this group list all virtual networks in US region (this will map to a specific region in Azure cloud as we will outline )

We use this regional grouping in order to specify the exact region were this virtual networks will be provisioned across the Azure cloud. We declare the exact region in a variable called ***region*** and we define this variable in both the eu.yml and us.yml files under the group\_vars directory.

We are going to use this variable in subsequent recipes in order to deploy our resources in these respective azure regions.

# Authenticating to your AWS Account

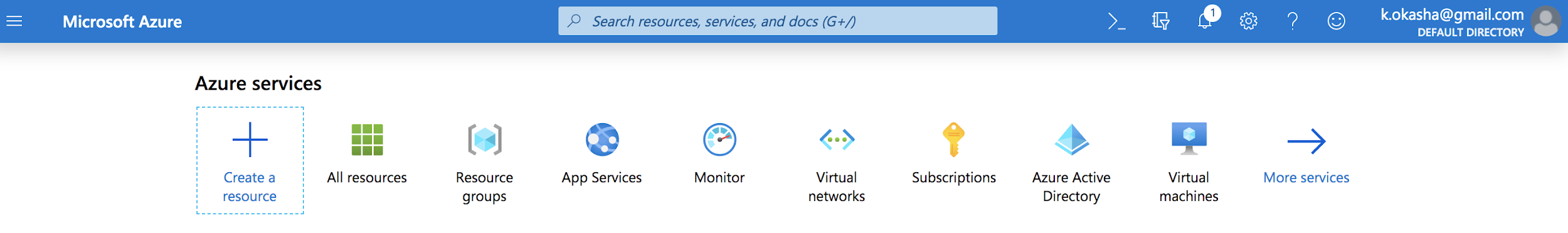
In this recipe, we will outline how to create the required credentials to programmatically authenticate to our Azure account from Ansible and how to secure these credentials using ansible vault. This is a mandatory step in order to be able to run any ansible module in all the next recipes.

## Getting Ready

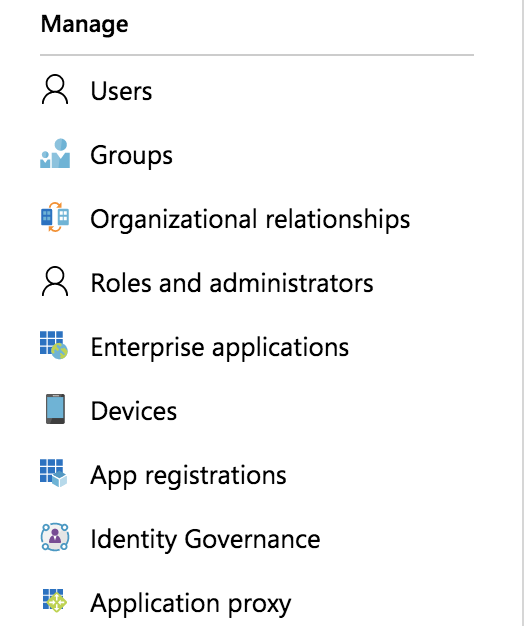
The Ansible controller must have internet access and the ansible inventory must be setup as outlined in the previous recipe. Also, the user performing these steps must have administrative access to the Azure portal in order to create the required resources to enable programmatic interaction with the Azure APIs.

## How to do it..

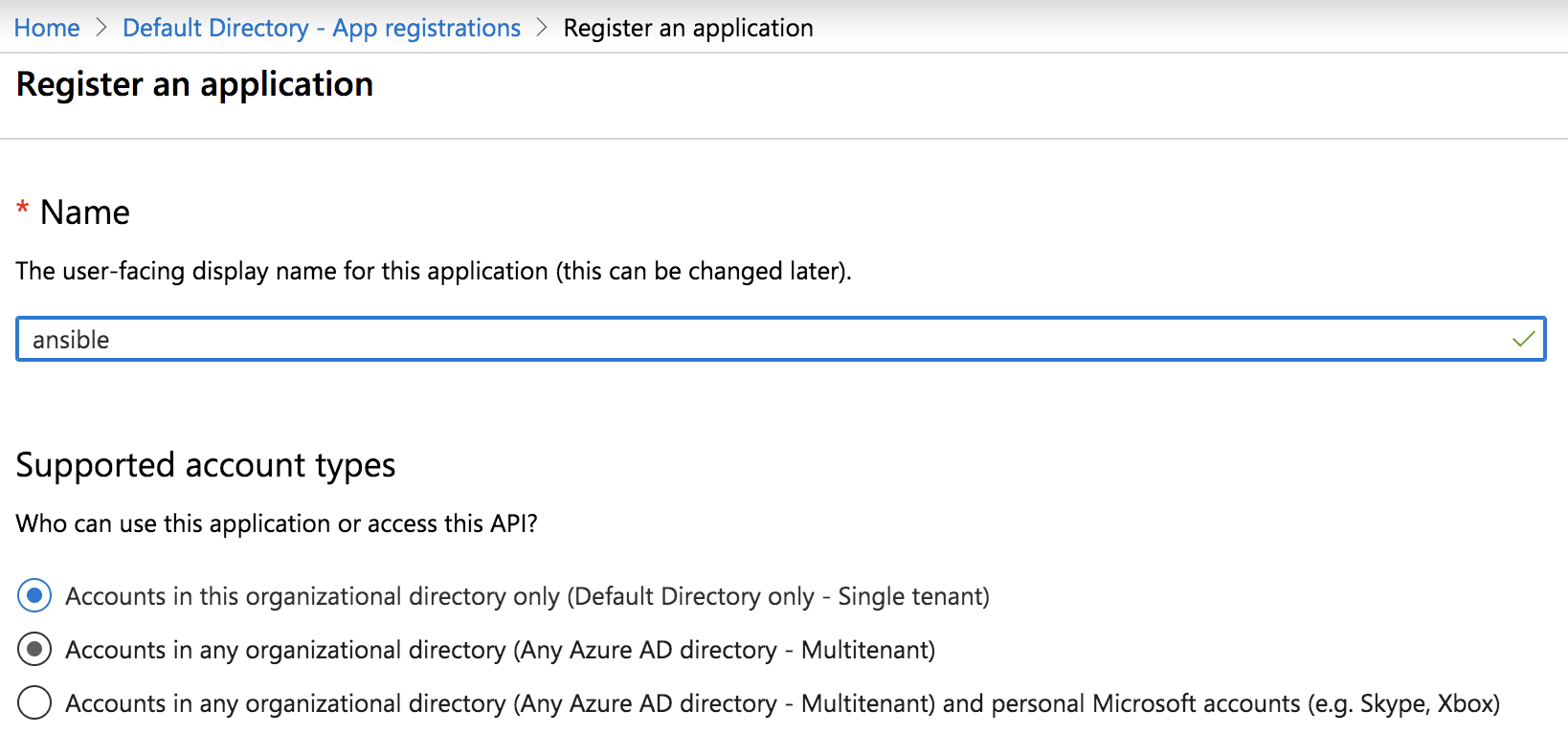
1. Login to Azure portal with the account admin rights.
2. In the Home page select Azure Active Directory.



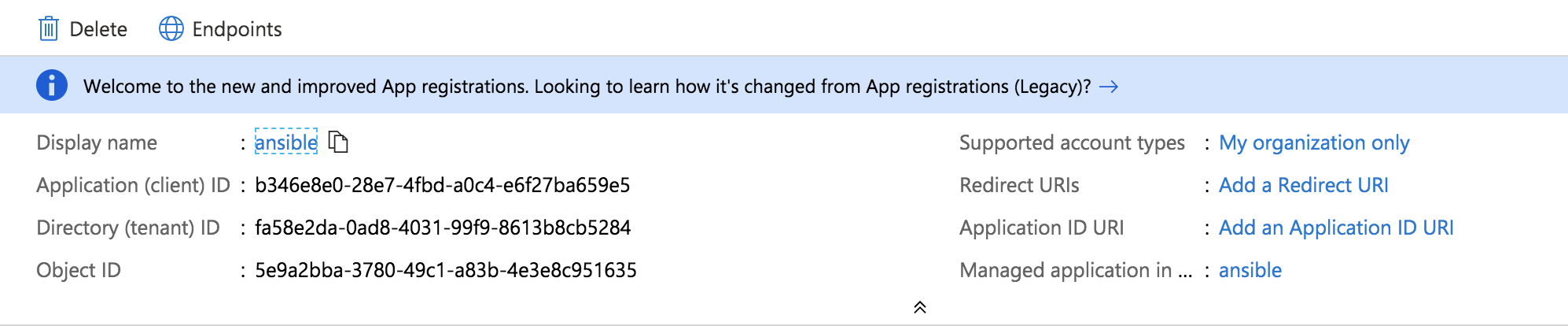
1. From the Left panel select App registration



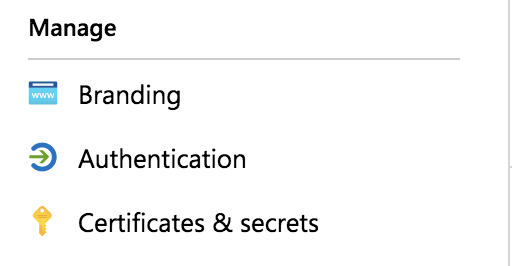
1. Click the New Registration and supply the below information to create a new application



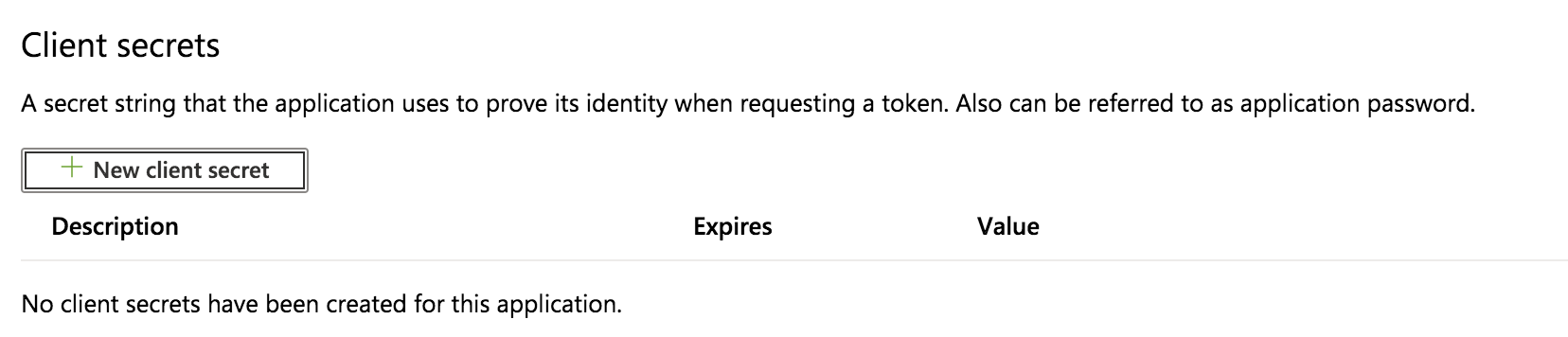
1. Once you click the register button, the new application will be created and its information is displayed as shown below ( we need the client\_id and tenant\_id data)



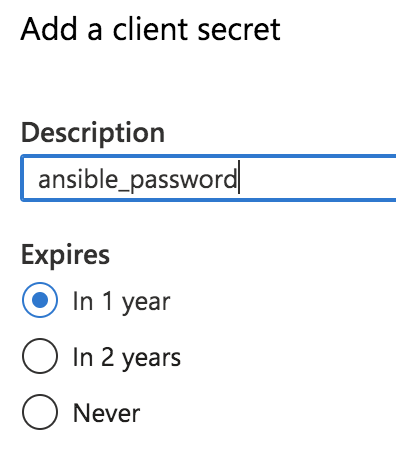
1. We select the Certificates & secrets on the left panel



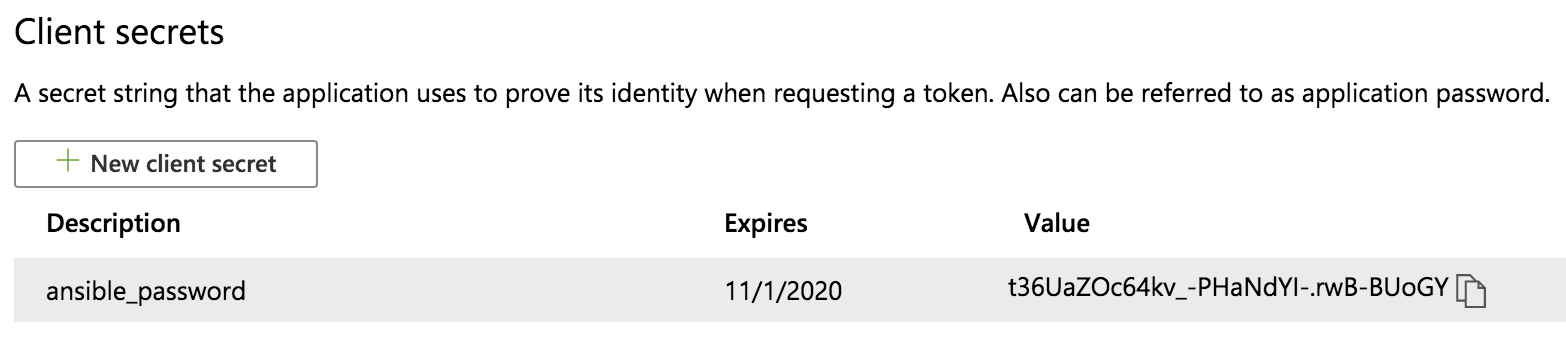
1. We click on the New Client Secret



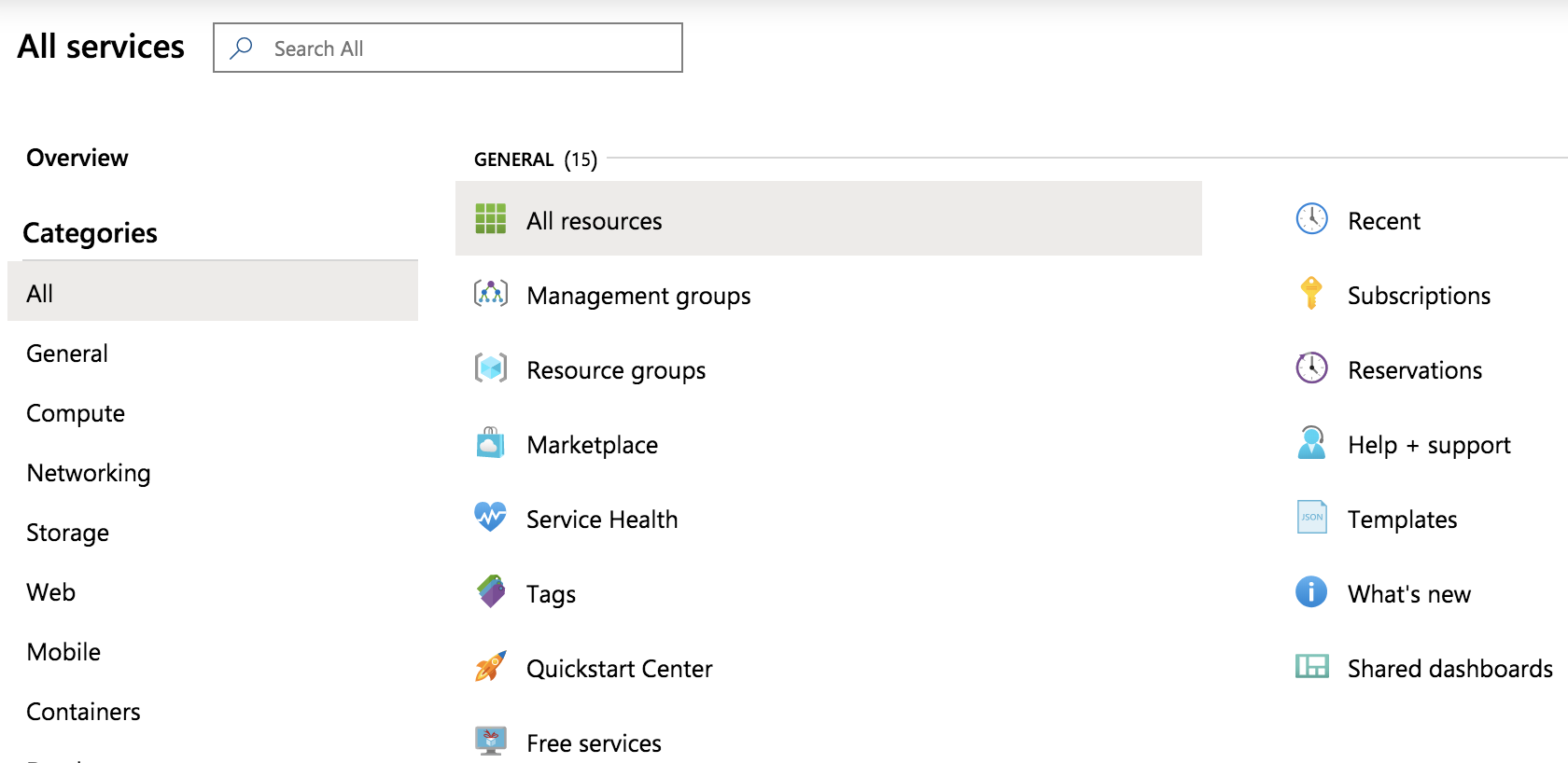
1. We specify a name for the password for this App and select its Expiry



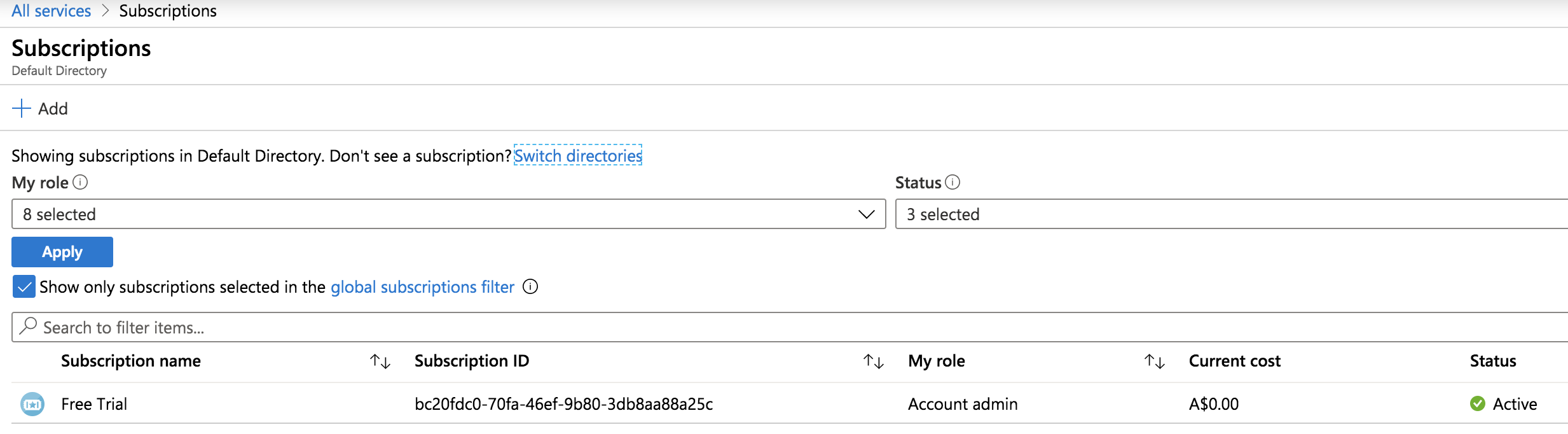
1. Once created we record the secret which is displayed (this is the only time that we will be able to see this password in plain text)



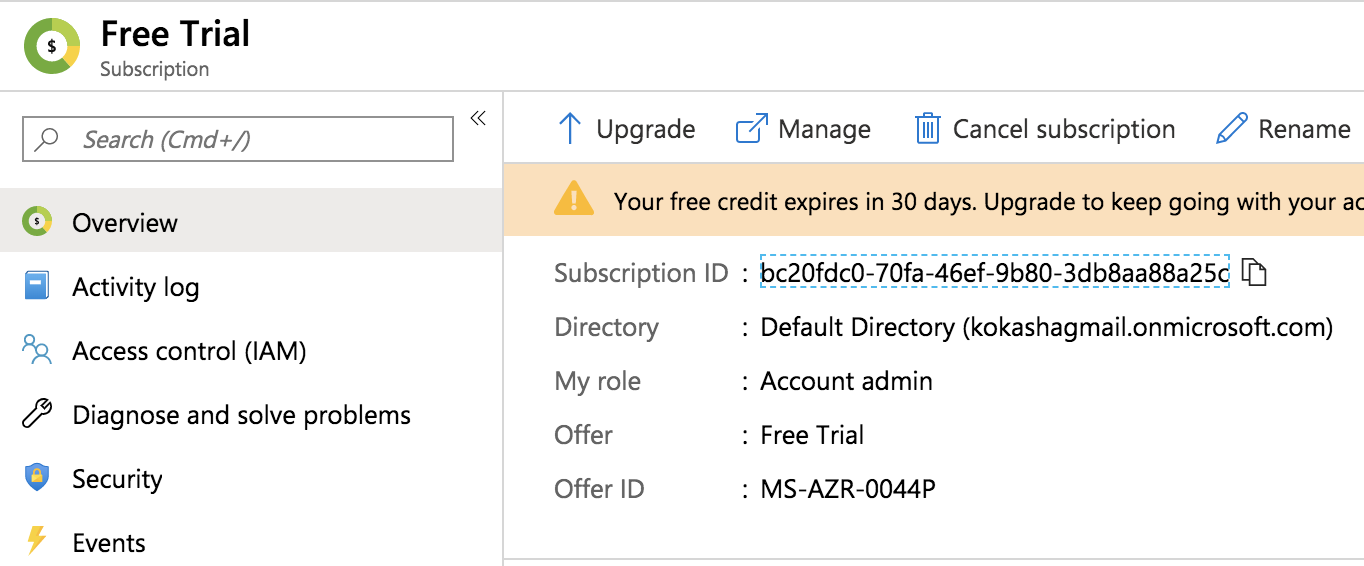
1. Go to All Services and select Subscriptions



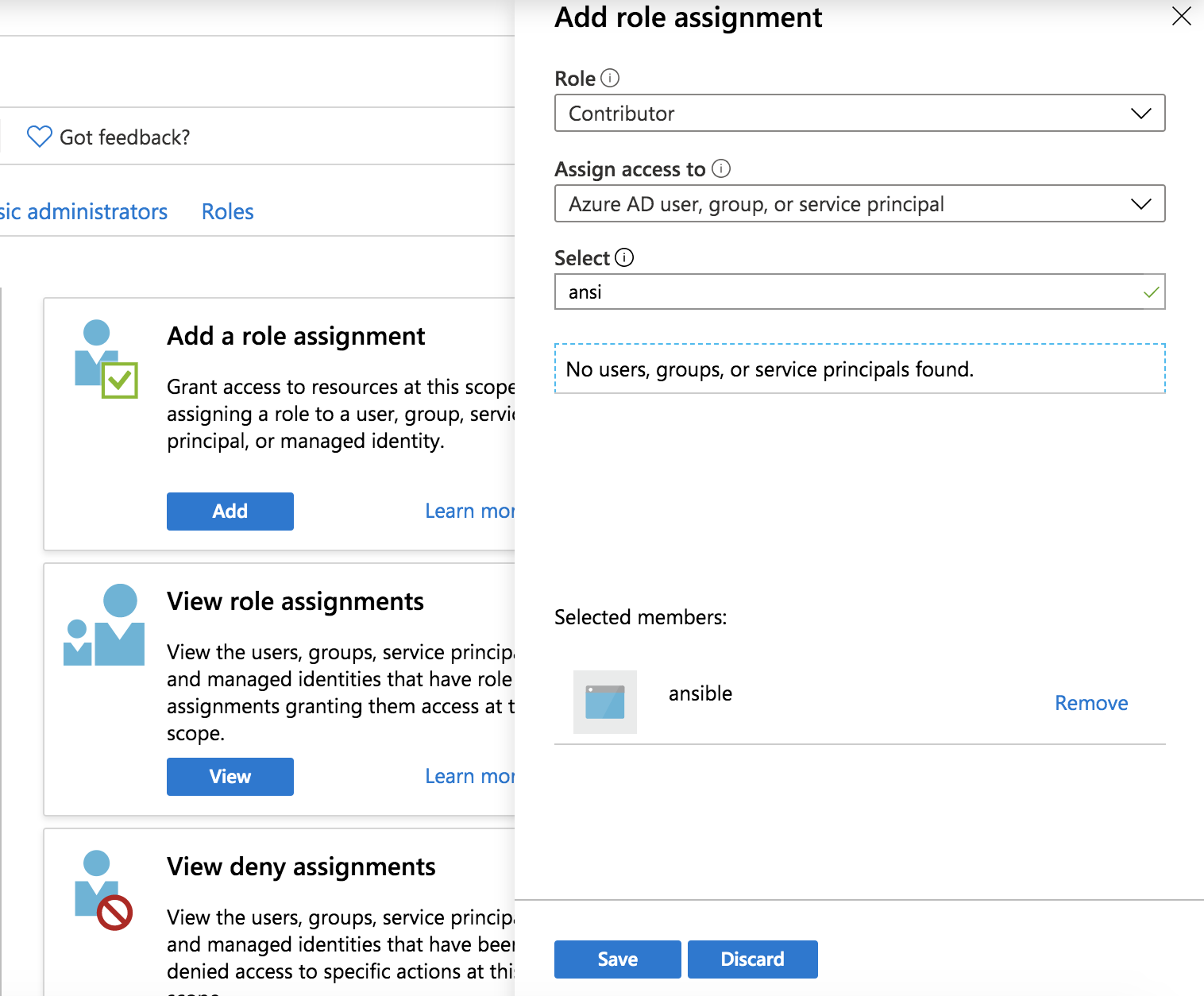
1. Click on the Subscription name ( Free Trial in my case)



1. Record the subscription ID since we need it in the authentication and click on the Access control (IAM) tab on the left



1. Click on Add role assignment and assign the Contributor role to the ansible App that we created



1. On the ansible control node create a new file which will hold our ansible vault password

$ echo ‘AzureV@uLT2019’ > .vault\_pass

1. Create a new file called ***secret.yml*** using ansible vault as shown below

$ ansible-vault create azure\_secret.yml --vault-password-file=.vault\_pass

1. Populate the secret.yml file with the data that we get from azure portal for client\_id, teneant\_id , subscription\_id and the secret we created for our app.

---

tenant\_id: XXX-XXXXXXXX

client\_id: XXX-XXXX

subscription\_id: XXX-XXXXX

secret: XXX-XXXX

## How it works..

In Order to have a programmatic access to Azure API (this is how ansible communicate with Azue cloud to provision resources), we need to create a construct known as service principle in our Azure account. This service principle is similar to a user but with only API access to the Azure account. We create this service principle and call it ansible and we provide assign to it the contributor role in access management in order to be able to create resources in our account. In order to authenticate to the Azure API using this service principle we need to supply the below four components

* Client\_id
* Tenant\_id
* Subscription\_id
* Service principal password

We locate all this information in azure portal using the steps outlined and we create a new file called secrets.yml which we encrypt using ansible-vault and we place all these variables in this file.

We will use these paramters in all subsequent recipes to authenticate to our Azure account to create the required infrastructure.

## See Also..

For more information regarding how to create a new service principle please check the below URL.  
<https://docs.microsoft.com/en-au/azure/active-directory/develop/howto-create-service-principal-portal>

For more information regarding the Azure built-in Roles which can be assigned to users/Apps please check the below URL

<https://docs.microsoft.com/en-au/azure/role-based-access-control/built-in-roles>

# Creating a resource Group

In this recipe we will outline how to deploy Resource groups in Azure. Resource groups is part of Azure Resource Manager deployment model and it is the prefered method to deploy resources in Azure cloud since it allows us to group similar resources (like VM, VM NIC, VM IP address, etc..) which share the same lifecycle in a single container which is the resource group. We are going to deploy all related resources that we will provision using the resource groups.

## Getting Ready

The Ansible control machine must be connected to the Internet with reachability to Azure Public API endpoints and the Azure Account should be configured as outlined in the previous recipes.

## How to do it..

1. Update eu.yml and us.yml files under group\_vars with the below data

$ cat group\_vars/eu.yml

rg\_name: "rg\_{{ inventory\_hostname }}"

$ cat group\_vars/eu.yml

rg\_name: "rg\_{{ inventory\_hostname }}"

1. Create new playbook pb\_build\_azure\_net.yml with the below content

---

- name: Build Azure Network Infrastructure

hosts: all

connection: local

gather\_facts: no

tasks:

- name: Create Resource group

azure\_rm\_resourcegroup:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

location: "{{ region }}"

subscription\_id: "{{ subscription\_id }}"

name: "{{ rg\_name }}"

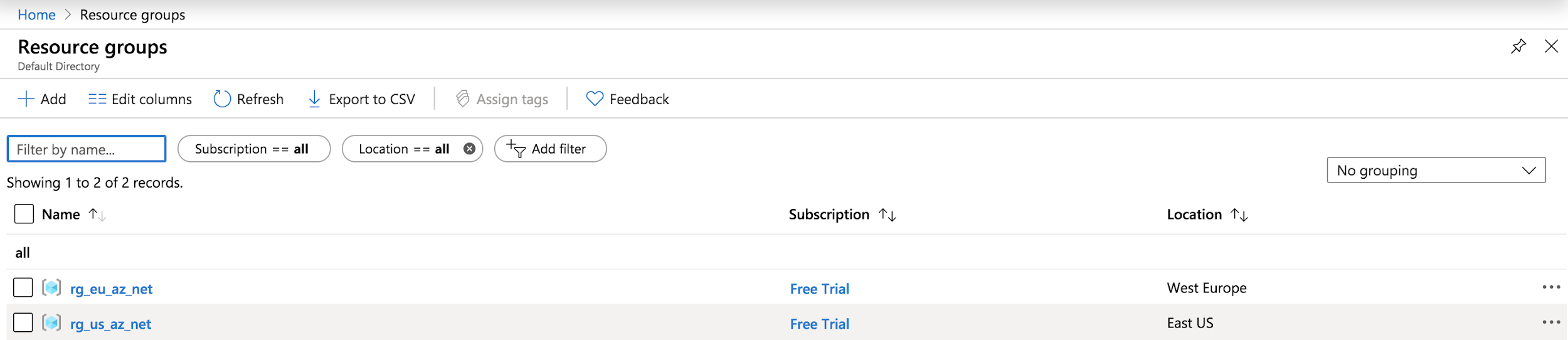
state: "{{ state | default('present') }}"

## How it works..

We declare the name of the resource group that we will deploy in each region in the YAML file which describes each region. We use the ***rg\_name*** parameter to hold the name of the resource group. We use the ansible module ***azure\_rm\_resourcegroup*** to create the resource group on Azure and it takes the following parameters to authenticate to the Azure API as well as to deploy the resource group

* Region, this parameter describe on which region we will deploy this resource group
* Tenant, secret, client\_id, subscription\_id, all these parameters are used to authenticate to our Account
* Name, is the name of our resource group

Once we run our playbook we can see that the resource groups is provisned on Azure portal as shown below



**See Also..**

For more information regarding the azure resource module in Ansible and all the other parameters supported by this module please check the below URL.

<https://docs.ansible.com/ansible/latest/modules/azure_rm_resourcegroup_module.html>

# Creating Virtual Networks

Virtual Networks in Azure Cloud is our Virtual Data Center which groups all our infrastructure in a similar manner to a physical Data Center, we can have multiple Virtual Networks in the same region and across different regions and we deploy our infrastructure inside these Virtual Networks. In this recipe we will outline how to define and provision a virtual network in Azure.

## Getting Ready

The Ansible control machine must be connected to the Internet with reachability to Azure Public API endpoints and the Azure Account should be configured as outlined in the previous recipes. Also the Resource group should be provisioned as outlined in the previous recipe

## How to do it..

1. Update ***eu.yml*** and ***us.yml*** files under group\_vars with the virtual Network name and CIDR address

$ cat group\_vars/eu.yml

vnet\_name: "vn\_{{ inventory\_hostname }}"

vnet\_cidr: 10.1.0.0/16

$ cat group\_vars/us.yml

vnet\_name: "vn\_{{ inventory\_hostname }}"

vnet\_cidr: 10.2.0.0/16

1. Update the playbook pb\_build\_azure\_net.yml with the task to create the virtual networks

- name: Create Virtual Networks

azure\_rm\_virtualnetwork:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

location: "{{ region }}"

subscription\_id: "{{ subscription\_id }}"

resource\_group: "{{ rg\_name}}"

name: "{{ vnet\_name }}"

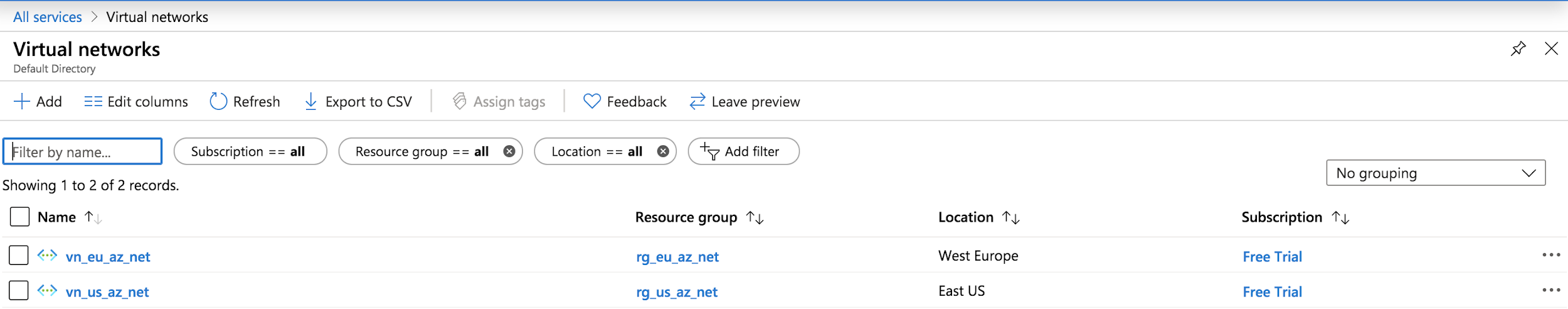
address\_prefixes\_cidr: "{{ vnet\_cidr }}"

state: "{{ state | default('present') }}"

## How it works..

In order to Create the virtual network, we need to supply the Virtual Network name along with the CIDR IP range that this Virtual Network will take. We define these two parameters in the Region YAML files as vnet\_name and vnet\_cidr. We use the ansible module ***azure\_rm\_virtualnetwork*** to create all the required virtual networks and we supply these parameters along with the resource\_group and all the required parameters for authentication.

Once we run the playbook we can see that Virtual Networks created as shown below



## See Also..

For more information regarding the azure Virtual Network module in Ansible and all the other parameters supported by this module please check the below URL.

<https://docs.ansible.com/ansible/latest/modules/azure_rm_virtualnetwork_module.html>

# Creating Subnets

Subnets is the networking construct within Azure cloud in order to segment the virtual network, it is used to provide us with the tool to segregate our virtual network into distinct routing and security domains which we can control in order to provide differentiated routing and security behaviour within each subnet. In this recipe we will outline how to define and provision subnets within Azure cloud

## Getting Ready

The Ansible control machine must be connected to the Internet with reachability to Azure Public API endpoints and the Azure Account should be configured as outlined in the previous recipes. Also the Resource group and Virtual Networks should be provisioned as outlined in the previous recipe

## How to do it..

1. Update ***eu.yml*** and ***us.yml*** files under group\_vars with the subnet information

$ cat group\_vars/eu.yml

subnets:

- name: web\_tier

cidr: 10.1.1.0/24

- name: db\_tier

cidr: 10.1.2.0/24

$ cat group\_vars/us.yml

subnets:

- name: web\_tier

cidr: 10.2.1.0/24

- name: db\_tier

cidr: 10.2.2.0/24

1. Update the playbook pb\_build\_azure\_net.yml with the task to create the subnets

- name: Create Subnets

azure\_rm\_subnet:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

subscription\_id: "{{ subscription\_id }}"

resource\_group: "{{ rg\_name}}"

name: "{{ item.name}}"

virtual\_network\_name: "{{ vnet\_name }}"

address\_prefix\_cidr: "{{ item.cidr }}"

state: "{{ state | default('present') }}"

loop: "{{ subnets }}"

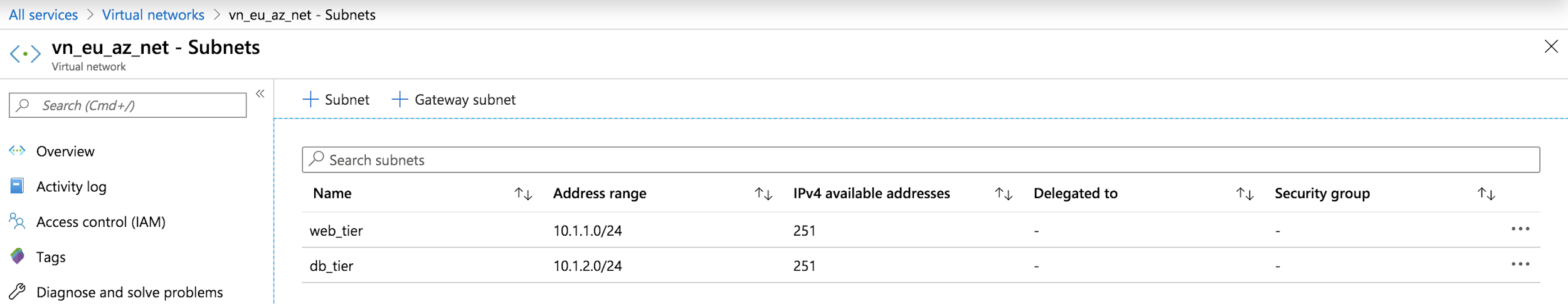
loop\_control:

label: "{{ item.name }}"

## How it works..

In order to Create the subnets within the virtual network we need to supply the Virtual Network and the CIDR prefix for the subnet (which must be within the CIDR of the virtual network). We define these the subnets data structure which include the name and CIDR for each subnet we want to provision. We use the ansible module ***azure\_rm\_subnet*** to create all the required subnets and we loop over the subnets data structure to supply the required parameters.

Once we run the playbook we can see that Subnets created within each virtual network created as shown below



## See Also..

For more information regarding the azure Subnets module in Ansible and all the other parameters supported by this module please check the below URL.

https://docs.ansible.com/ansible/latest/modules/azure\_rm\_subnet\_module.html

# Building User Defined Routes

In this recipe, we will outline how to control routing within a subnet using user defined routers. This user defined routes object can be associated with a specific subnet and we can define custom routes to adjust forwarding behaviour within a subnet in Azure cloud.

## Getting Ready

The Ansible control machine must be connected to the Internet with reachability to Azure Public API endpoints and the Azure Account should be configured as outlined in the previous recipes. Also the Resource group , Virtual Networks and Subnets should be provisioned as outlined in the previous recipes.

## How to do it..

1. Update ***eu.yml*** and ***us.yml*** files under group\_vars with the route\_tables data as shown below

**$ cat group\_vars/eu.yml group\_vars/us.yml**

route\_tables:

- name: db\_tier\_rt

subnet: db\_tier

routes:

- name: Default Route

prefix: 0.0.0.0/0

nh: none

1. Update the playbook pb\_build\_azure\_net.yml with the below task to create the custom route table

- name: Create Custom Route Table

azure\_rm\_routetable:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

subscription\_id: "{{ subscription\_id }}"

resource\_group: "{{ rg\_name}}"

name: "{{ item.name}}"

state: "{{ state | default('present') }}"

loop: "{{ route\_tables }}"

tags: routing

1. Update the playbook pb\_build\_azure\_net.yml with the below task to provision the routes within the custom route table

- name: Provision Routes

azure\_rm\_route:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

subscription\_id: "{{ subscription\_id }}"

resource\_group: "{{ rg\_name}}"

route\_table\_name: "{{ item.0.name }}"

name: "{{ item.1.name}}"

address\_prefix: "{{ item.1.prefix }}"

next\_hop\_type: "{{ item.1.nh }}"

state: "{{ state | default('present') }}"

with\_subelements:

- "{{ route\_tables }}"

- routes

tags: routing

1. Update the playbook pb\_build\_azure\_net.yml with the below task to associate the custom route with the subnet

- name: Attach Route Table to Subnet

azure\_rm\_subnet:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

subscription\_id: "{{ subscription\_id }}"

resource\_group: "{{ rg\_name}}"

name: "{{ item.subnet}}"

virtual\_network\_name: "{{ vnet\_name }}"

route\_table: "{{ item.name }}"

state: "{{ state | default('present') }}"

loop: "{{ route\_tables }}"

loop\_control:

label: "{{ item.name }}"

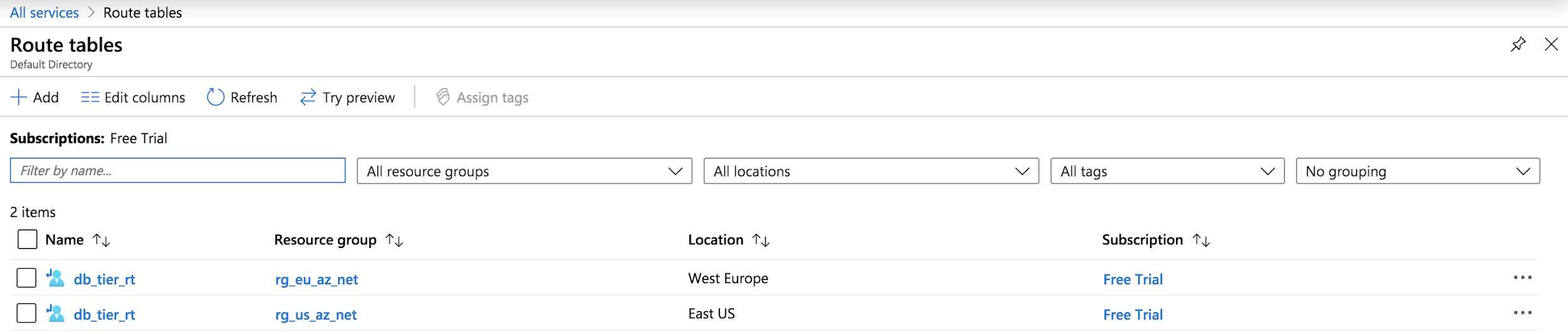
tags: routing

## How it works..

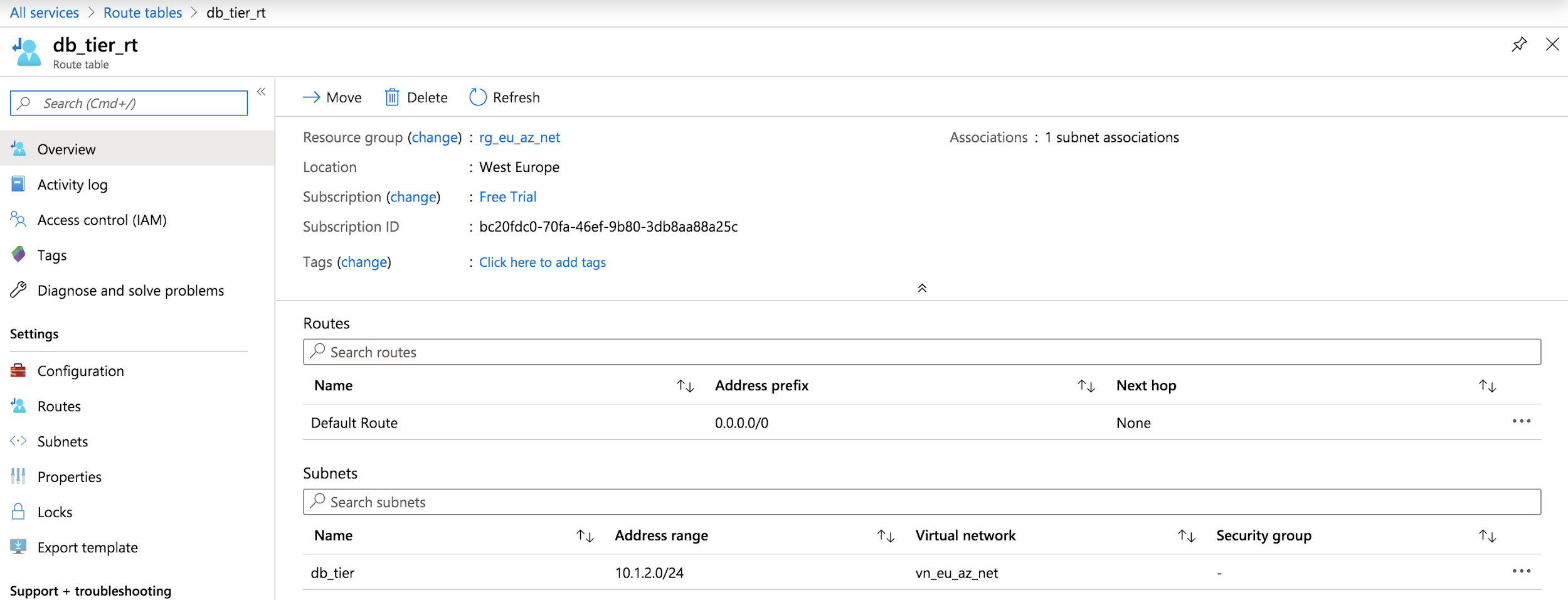
In our setup we have two subnets (web and DB) and we need to provide different routing treatment for the DB subnet in order to not have public internet access. We use the custom route table in order to achieve this behaviour by creating a new custom route table and install a default route with next-hop of none in order to drop all traffic destined to the internet.

We define our route\_tables that we will deploy in the route\_tables variable and include it in each region definition. We use the ***azure\_rm\_routetable*** ansible module to create the route table in the specific resource group and we use the ***azure\_rm\_route*** module to create the required routes within the route table. Finally, we attached the route\_table to the specific subnet using the ***azure\_rm\_subnet*** module in order to modify the default routing behaviour for this subnet.

The below diagram outlines the new Route table that was created



The below diagram outlines the exact details for one of the route tables and what is the custom routes and which subnet this custom route is attached to



## See Also..

For more information regarding the azure route table modules in Ansible and all the other parameters supported by this module please check the below URL.

<https://docs.ansible.com/ansible/latest/modules/azure_rm_routetable_module.html#azure-rm-routetable-module>

<https://docs.ansible.com/ansible/latest/modules/azure_rm_route_module.html#azure-rm-route-module>

# Deploying Network Security Groups

Security in cloud environments is critical and Azure cloud provide different tools and services to help build a secure cloud environment for the application. In this recipe we will look into one of these services which is Network Security groups. NSG is a stateful firewall which can be attached to a Virtual Machine or Subnet in order to restrict traffic flowing through the VM or subnet. In this recipe we will outline how to define and provision NSG on Azure cloud.

## Getting Ready

The Ansible control machine must be connected to the Internet with reachability to Azure Public API endpoints and the Azure Account should be configured as outlined in the previous recipes. Also the Resource group , Virtual Networks and Subnets should be provisioned as outlined in the previous recipes.

## How to do it..

1. Update ***eu.yml*** and ***us.yml*** files under group\_vars with the ACL data as shown below

**$ cat group\_vars/eu.yml group\_vars/us.yml**

acls:

- name: Inbound\_Web\_Tier

subnet: web\_tier

rules:

- name: Allow\_HTTP\_Internet

destination\_address\_prefix: 10.1.1.0/24

direction: Inbound

access: Allow

protocol: Tcp

destination\_port\_range:

- 80

- 443

priority: 101

1. Update the playbook ***pb\_build\_azure\_net.yml*** with the below task to create the security group and populate all its rules

- name: Create new Security Group

azure\_rm\_securitygroup:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

subscription\_id: "{{ subscription\_id }}"

resource\_group: "{{ rg\_name}}"

name: "{{ item.name }}"

purge\_rules: yes

rules: "{{ item.rules }}"

loop: "{{ acls }}"

Tags: security

1. Update the playbook ***pb\_build\_azure\_net.yml*** with the below task to attach the security group with the corresponding subnet

- name: Attach Security Group to Subnet

azure\_rm\_subnet:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

subscription\_id: "{{ subscription\_id }}"

resource\_group: "{{ rg\_name}}"

name: "{{ item.subnet}}"

virtual\_network\_name: "{{ vnet\_name }}"

security\_group: "{{ item.name }}"

state: "{{ state | default('present') }}"

loop: "{{ acls }}"

tags: security

## How it works..

Azure provides default Network security groups which is attached to subnets in order to provide basic security control to compute resources deployed in these subnets. The default policy includes these main rules for Inbound

* Allow Inbound traffic between VirtualNetwork CIDR Ranges (Inter Subnet communication)
* Allow Inbound traffic from Azure Load balances
* Deny any Other Traffic

On the Outbound direction below are the default rules

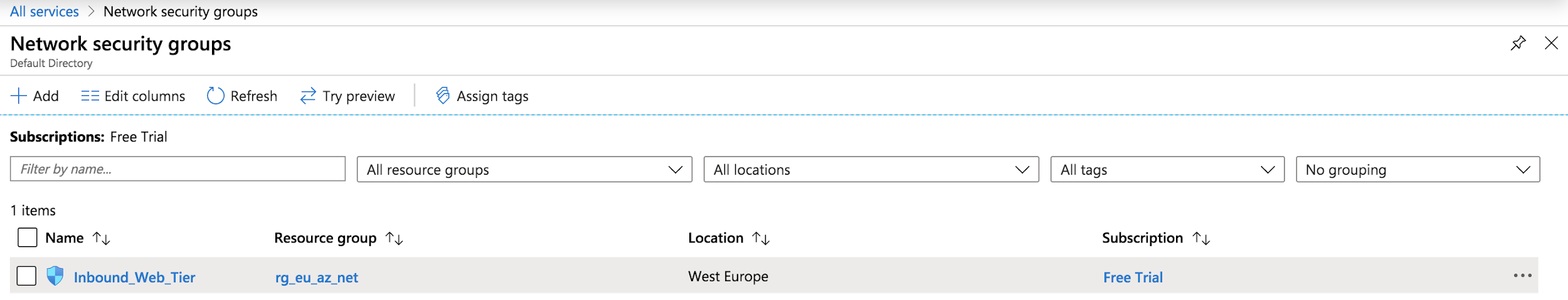
* Allow Outbound traffic between VirtualNetwork CIDR (Inter Subnet communication)
* Allow OUtbound traffic to Internet
* Deny any Other Traffic

Azure NSG provides a mechanism to augment the default NSG applied by Azure by defining custom NSG which get appended to the default NSG. The resultant NSG is evaluated based on Priority ( precedence value) value for each rule and once a rule is matched the rule is applied to the traffic traversing the subnet.

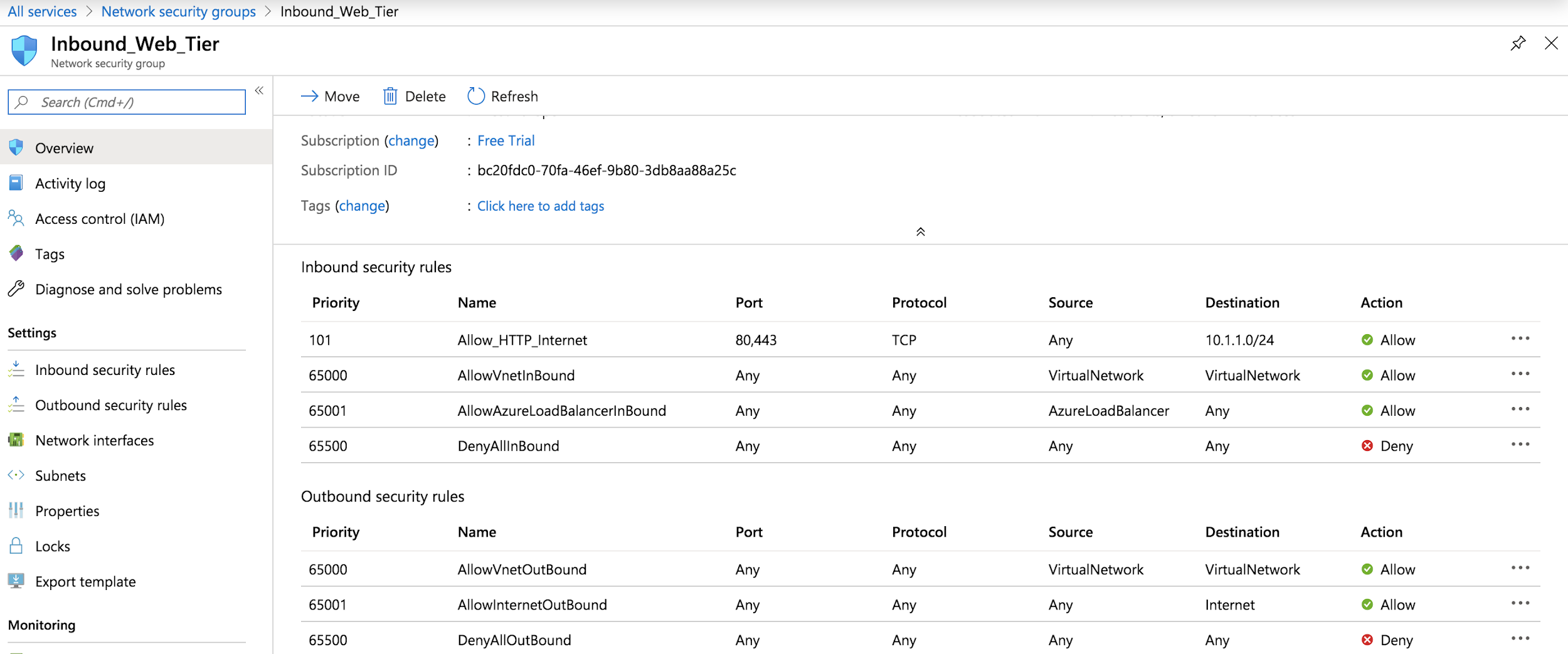
Since we are deploying Web application in the Web\_tier subnet we need to allow inbound HTTP and HTTPs traffic to this subnet so we create an ACL definition to create a custom NSG and we define the required parameters in order to allow this traffic in the inbound direction.

We use the azure\_rm\_securitygroup to loop over all our Custom ACLs and create the NSG and respective rules and we use the azure\_rm\_subnet to attach the security group to the subnet.

The below diagram outline the new NSG which is defined



The below diagram outline the resultant NSG rules that is defined (custom + default) for Inbound and Outbound direction



## See Also..

For more information regarding the azure route table modules in Ansible and all the other parameters supported by this module please check the below URL.

<https://docs.ansible.com/ansible/latest/modules/azure_rm_securitygroup_module.html#azure-rm-securitygroup-module>

# Deployment Validation using Ansible

Ansible provides multiple modules to collect operational state of the different objects we created in Azure and we can use these modules to validate the current state of our network in Azure cloud. This provides a programmatic approach for validating a deployment without the need to login via GUI to the portal to check the status of the different components within the infrastructure in Azure. In this recipe we will outline how to use multiple modules to validate the Resource groups and Virtual Networks that we have deployed.

## Getting Ready

The Ansible control machine must be connected to the Internet with reachability to Azure Public API endpoints and the Azure Account should be configured as outlined in the previous recipes.

## How to do it..

1. Create the new file ~/.azure/credentials to host all the credentials to authenticate to Azure as shown below

$ cat ~/.azure/credentials

[default]

subscription\_id=XXX-XXXX-XXXX

client\_id=XXX-XXXX-XXXX

secret=XXX-XXXX-XXXX

tenant=XXX-XXXX-XXXX

1. Create a new playbook pb\_validate\_azure\_net.yml to validate our deployment and include the below tasks to collect resource group facts and validate it.

$ cat pb\_validate\_azure\_net.yml

- name: Build Azure Network Infrastructure

hosts: all

connection: local

gather\_facts: no

tasks:

- name: Get Resource Facts

azure\_rm\_resourcegroup\_facts:

name: "{{ rg\_name }}"

register: rg\_facts

tags: rg\_facts

- name: Validate Resource Group is Deployed

assert:

that:

- rg.name == rg\_name

- rg.properties.provisioningState == 'Succeeded'

- rg.location == region

loop: "{{ azure\_resourcegroups }}"

loop\_control:

loop\_var: rg

tags: rg\_facts

1. Update the ***pb\_validate\_azure\_net.yml*** playbook to include tasks to collect virtual Networks facts and validate its state.

- name: Validate Virtual Network is Deployed

azure\_rm\_virtualnetwork\_facts:

resource\_group: "{{ rg\_name }}"

register: vnet\_facts

tags: vnet\_facts

- name: Validate Virtual Networks are Deployed

assert:

that:

- vnet.name == vnet\_name

- vnet.properties.provisioningState == 'Succeeded'

- vnet.properties.addressSpace.addressPrefixes | length == 1

- vnet.properties.addressSpace.addressPrefixes[0] == vnet\_cidr

loop: "{{ azure\_virtualnetworks }}"

loop\_control:

loop\_var: vnet

tags: vnet\_facts

**How it works..**

In this recipe we outline an alternate method to authenticate to azure cloud other that the way we outlined in the previous recipes. We create the ~/.azure/credentials file and we put into it the same information needed to authenticate to Azure API (tenant\_id, client\_id, etc…). Since we have this information in the file we don’t need to include these parameters in our ansible modules.

In order to validate our deployment, ansible provides multiple facts modules to collect the operational state of multiple objects in Azure cloud. In this example we are just outlining two of these modules to collect the facts for Resource Groups and Virtual Networks. We use the azure\_rm\_resourcegroup\_facts module to collect resource group facts and azure\_rm\_virtualnetwork\_facts to collect virtual network facts. All Azure Facts modules register the data retrieved by these modules as ansible facts that is why we don’t need to register the data returned by the module into a custom defined variable.

The ***azure\_rm\_resourcegroup\_facts*** Module saved the output into ***azure\_resourcegroups*** ansible\_fact and we use the assert module to loop over all the resource groups within this variable and confirm that it is created with the correct parameters

Below is a snippet of the azure\_resourcegroups

ok: [eu\_az\_net] => {

"azure\_resourcegroups": [

{

"id": "/subscriptions/bc20fdc0-70fa-46ef-9b80-3db8aa88a25c/resourceGroups/rg\_eu\_az\_net",

"location": "westeurope",

"name": "rg\_eu\_az\_net",

"properties": {

"provisioningState": "Succeeded"

}

}

]

}

We use the exact similar technique to collect facts for virtual networks deployed using the ***azure\_rm\_virtualnetwork\_facts*** and use the assert module to validate its state.

## See Also..

For more information regarding the multiple modules for fact collection for the different network resources in Azure please check the below links.

* **Resource Groups Facts**

<https://docs.ansible.com/ansible/latest/modules/azure_rm_resourcegroup_info_module.html#azure-rm-resourcegroup-info-module>

* **Virtual Networks Facts**

<https://docs.ansible.com/ansible/latest/modules/azure_rm_virtualnetwork_info_module.html#azure-rm-virtualnetwork-info-module>

* **Subnets Facts**

<https://docs.ansible.com/ansible/latest/modules/azure_rm_subnet_info_module.html#azure-rm-subnet-info-module>

* **Route Table Facts**

<https://docs.ansible.com/ansible/latest/modules/azure_rm_routetable_info_module.html#azure-rm-routetable-info-module>

* **Security Groups Facts** <https://docs.ansible.com/ansible/latest/modules/azure_rm_securitygroup_info_module.html#azure-rm-securitygroup-info-module>

# Decommissioning Azure Resources Using Ansible

Similar to creating resources at scale using automation we can destroy these resources once we decide we don’t need them. This is simplified with ansible and the Resource groups implemented by Azure since now with a single API call with the correct parameters we can decommission all the resources within a resource group that we have defined. In this recipe we will outline how to perform this action to destroy all the resources we have provisioned this far.

## Getting Ready

The Ansible control machine must be connected to the Internet with reachability to Azure Public API endpoints and the Azure Account should be configured as outlined in the previous recipes.

## How to do it..

1. Create a new playbook pb\_destroy\_azure\_net.yml and add the below task to delete all resource groups.

$ cat pb\_destroy\_azure\_net.yml

---

- name: Build Azure Network Infrastructure

hosts: all

connection: local

gather\_facts: no

vars:

state: absent

vars\_files:

- azure\_secret.yml

tasks:

- name: Create Resource group

azure\_rm\_resourcegroup:

tenant: "{{ tenant\_id }}"

client\_id: "{{ client\_id }}"

secret: "{{ secret }}"

location: "{{ region }}"

subscription\_id: "{{ subscription\_id }}"

name: "{{ rg\_name }}"

force\_delete\_nonempty: yes

state: "{{ state | default('present') }}"

## How it works..

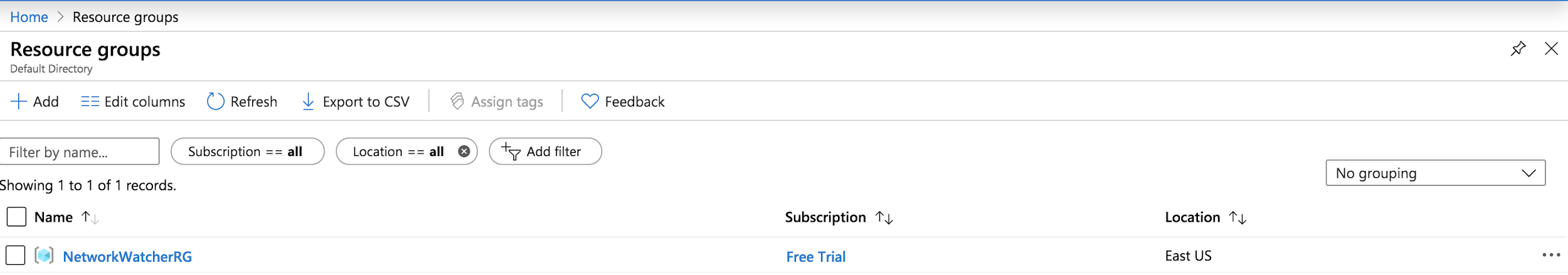
We use the azure\_rm\_resourcegroup ansible module to destroy all the resources within the resource group as well as deleting the resource group itself. We supply two important parameters to the module in order to perform the delete function

Set the state to absent

Include the parameter ***force\_delete\_nonempty*** and set it to yes.

With these parameters set all the resources within the resource group (Virtual Networks, subnets, etc…) will be deleted along with the resource group itself.

The below output shows that our two resource groups are not present



The below output also confirms that all the virtual networks were deleted after running the playbook

