**Resource Group and Networking**

The first step in our infrastructure deployment is to create a resource group, which acts as a logical container for our Azure resources. We define a resource group using the azurerm\_resource\_group resource in Terraform, specifying its name and location.

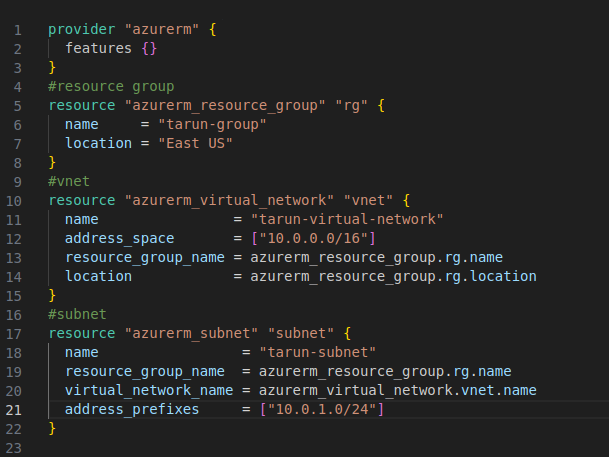
Next, we set up the networking components. We create a virtual network (azurerm\_virtual\_network) and a subnet (azurerm\_subnet) within that virtual network. These resources define the network configuration for our infrastructure, such as IP address ranges and connectivity rules. We associate the subnet with the virtual network and the resource group.

Provider: The code specifies the Azure provider and enables features.

Resource Group: The azurerm\_resource\_group resource creates an Azure resource group with the specified name and location.

Virtual Network: The azurerm\_virtual\_network resource creates a virtual network with the specified name, address space, and associated resource group and location.

Subnet: The azurerm\_subnet resource creates a subnet within the virtual network with the specified name, address prefix, and associated resource group and virtual network.



**Associating subnet with the NSG rules**

To control inbound and outbound traffic to our infrastructure, we configure a network security group (azurerm\_network\_security\_group) and define security rules using azurerm\_network\_security\_rule. These rules allow or deny specific network traffic based on protocols, ports, and IP addresses. We associate the network security group with the subnet using azurerm\_subnet\_network\_security\_group\_association.

Network Security Group (NSG): The azurerm\_network\_security\_group resource creates a network security group with the specified name, location, and associated resource group.

NSG Subnet Association: The azurerm\_subnet\_network\_security\_group\_association resource associates the subnet created earlier with the network security group.

Network Security Rule (NSG Rule): The azurerm\_network\_security\_rule resource creates a network security rule within the network security group. Two rules are created: one for SSH inbound traffic (port 22) and another for inbound traffic on port 80.



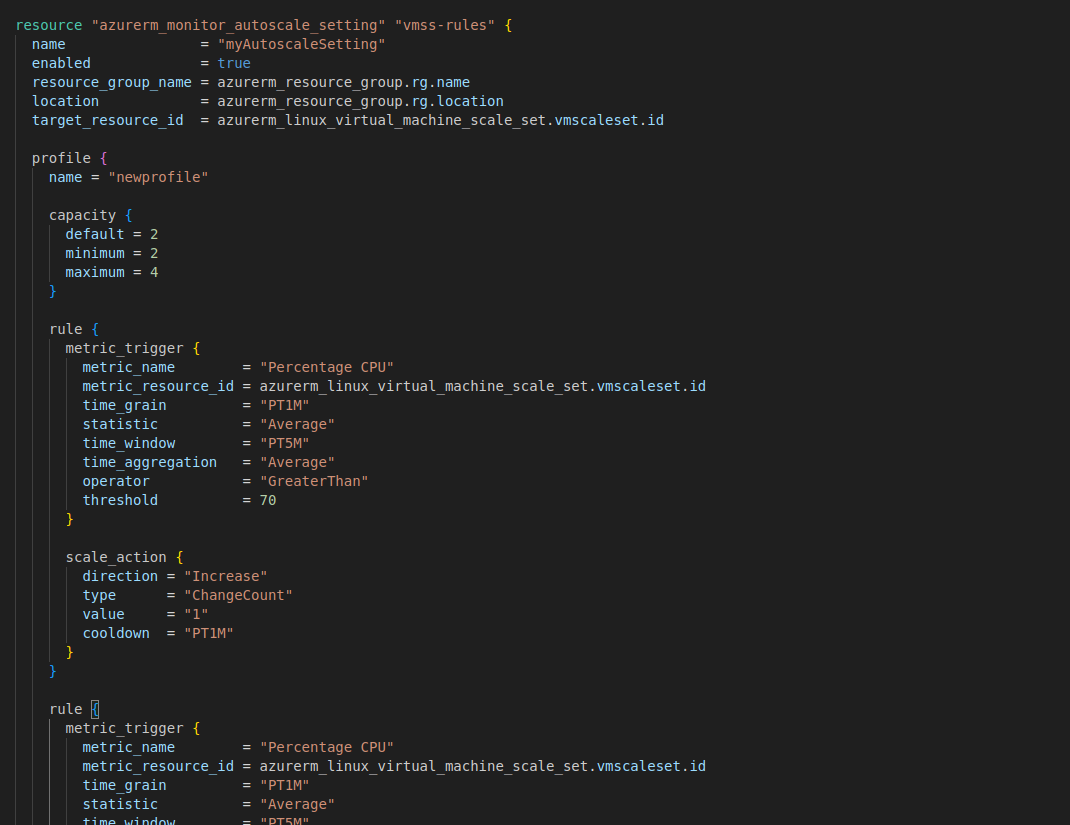
**Creating a Virtual Machine Scale Set**

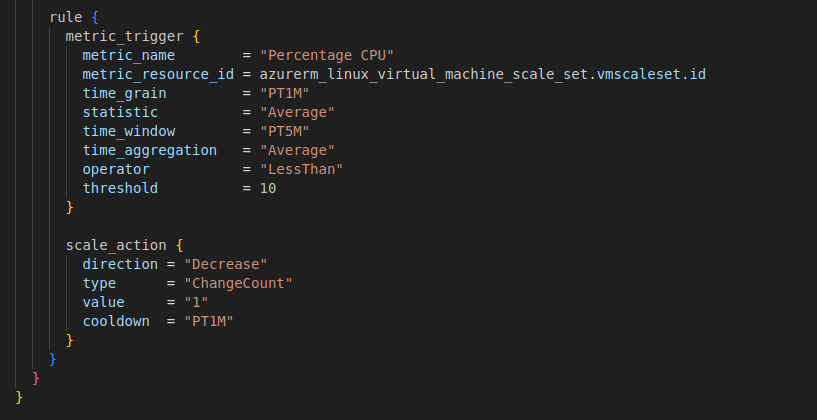
To enable automatic scaling of our application, we use a virtual machine scale set (azurerm\_linux\_virtual\_machine\_scale\_set). This resource allows us to create and manage a group of identical virtual machines that can scale up or down based on predefined rules. We specify the instance size, operating system image, and number of instances. We also configure the SSH key for remote access.

Virtual Machine Scale Set (VMSS): The azurerm\_linux\_virtual\_machine\_scale\_set resource creates a virtual machine scale set with the specified name, resource group, location, SKU, number of instances, and other configurations such as SSH key, OS image, disk, and network interface. It also associates the VMSS with the subnet and load balancer backend address pool.



Autoscale Setting: The azurerm\_monitor\_autoscale\_setting resource enables autoscaling for the virtual machine scale set based on CPU utilization thresholds. It defines scaling rules that increase or decrease the number of instances based on the average CPU percentage.



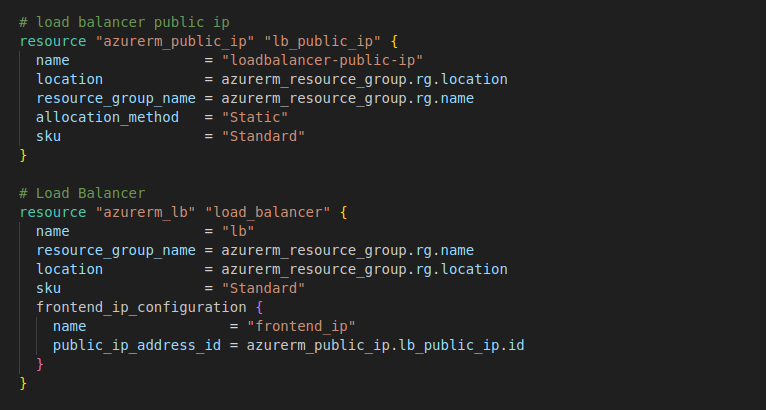


**Implementing load balancing**

Load Balancer: To distribute incoming traffic across multiple instances of our application, we set up a load balancer (azurerm\_lb) with a public IP address (azurerm\_public\_ip). The load balancer distributes traffic based on defined rules, such as TCP port forwarding. We configure a backend address pool and a probe to monitor the health of the instances. Finally, we create a rule that maps incoming requests to the backend pool.

Public IP: The azurerm\_public\_ip resource creates a public IP address with the specified name, location, allocation method (static), and SKU (Standard). This public IP address is associated with the load balancer.

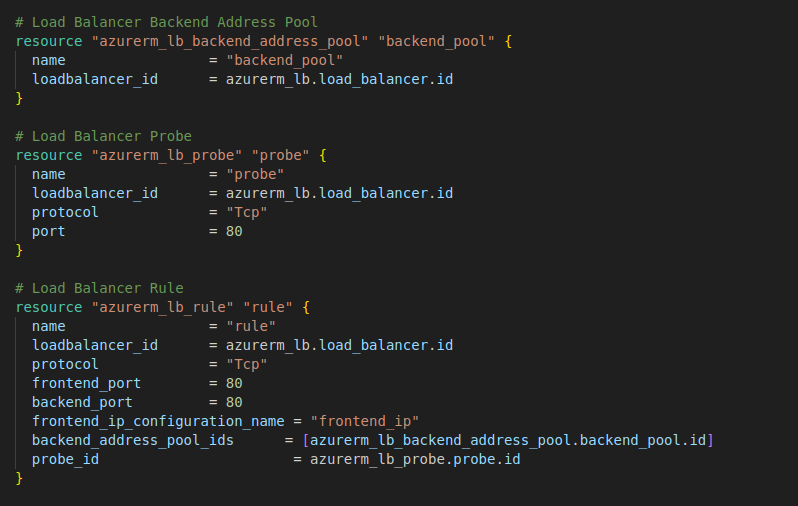
Load Balancer: The azurerm\_lb resource creates a load balancer with the specified name, resource group, location, and SKU. It also defines a frontend IP configuration that associates the load balancer with the previously created public IP.



load Balancer Backend Address Pool: The azurerm\_lb\_backend\_address\_pool resource defines a backend address pool for the load balancer, which determines the set of resources that receive traffic from the load balancer.

Load Balancer Probe: The azurerm\_lb\_probe resource defines a health probe for the load balancer, which checks the availability of backend resources by sending TCP requests to port 80.

Load Balancer Rule: The azurerm\_lb\_rule resource defines a load balancer rule that maps incoming traffic on port 80 to the backend resources in the backend address pool. It uses the previously defined frontend IP configuration and health probe.



**Creating Bastion service to access a specific VMS securely**

For secure remote access to our infrastructure, we set up Azure Bastion (azurerm\_bastion\_host). Azure Bastion provides a fully managed, browser-based SSH and RDP gateway to connect to virtual machines in the virtual network subnet securely.

Bastion Subnet: The azurerm\_subnet resource creates an additional subnet within the virtual network specifically for Azure Bastion. It has a specified name, address prefix, and associated resource group and virtual network.

Bastion Public IP: The azurerm\_public\_ip resource creates a public IP address specifically for Azure Bastion with the specified name, location, allocation method (static), and SKU.

Azure Bastion Host: The azurerm\_bastion\_host resource provisions an Azure Bastion host with the specified name, location, and associated resource group. It is configured with an IP configuration that links it to the Bastion

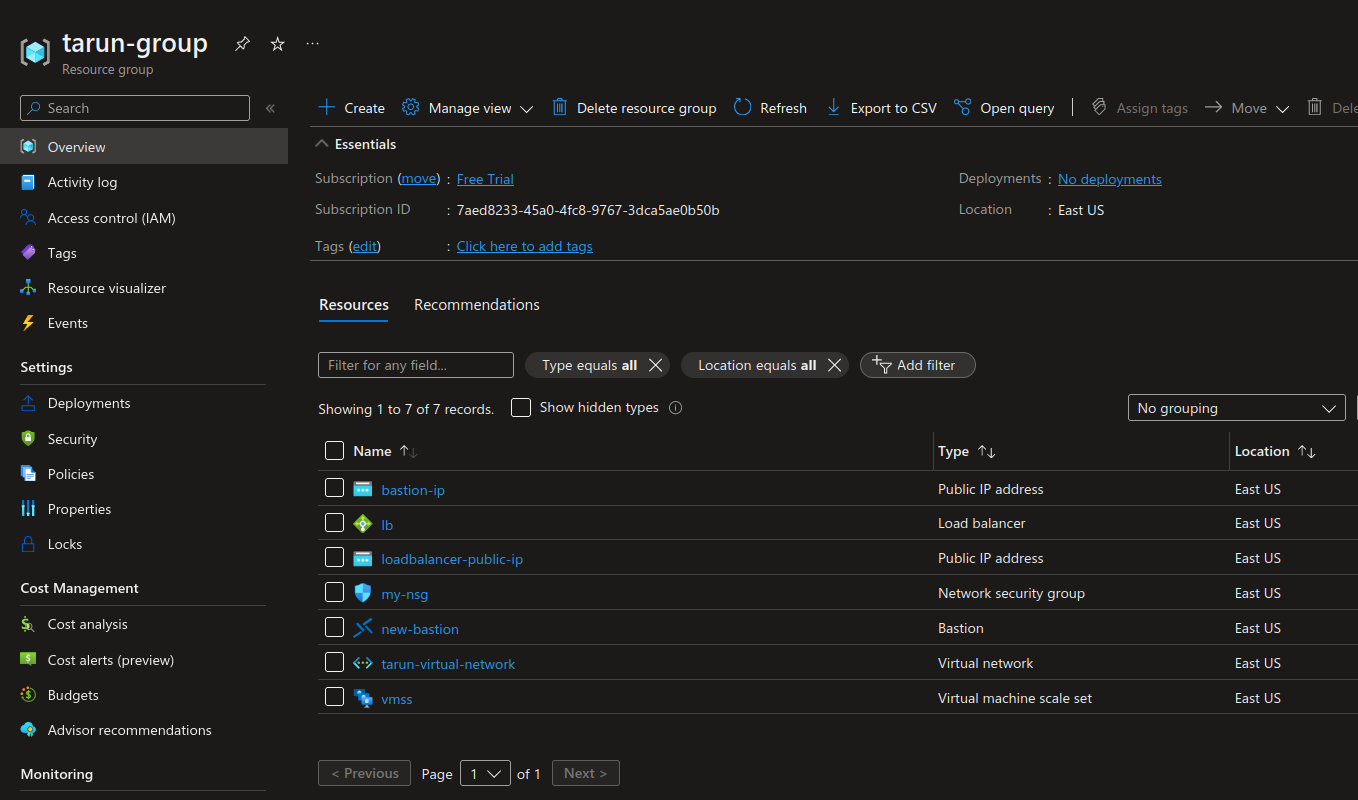


**Implement the code saving the code in a main.tf file and execute with terraform init, plan and apply command.**

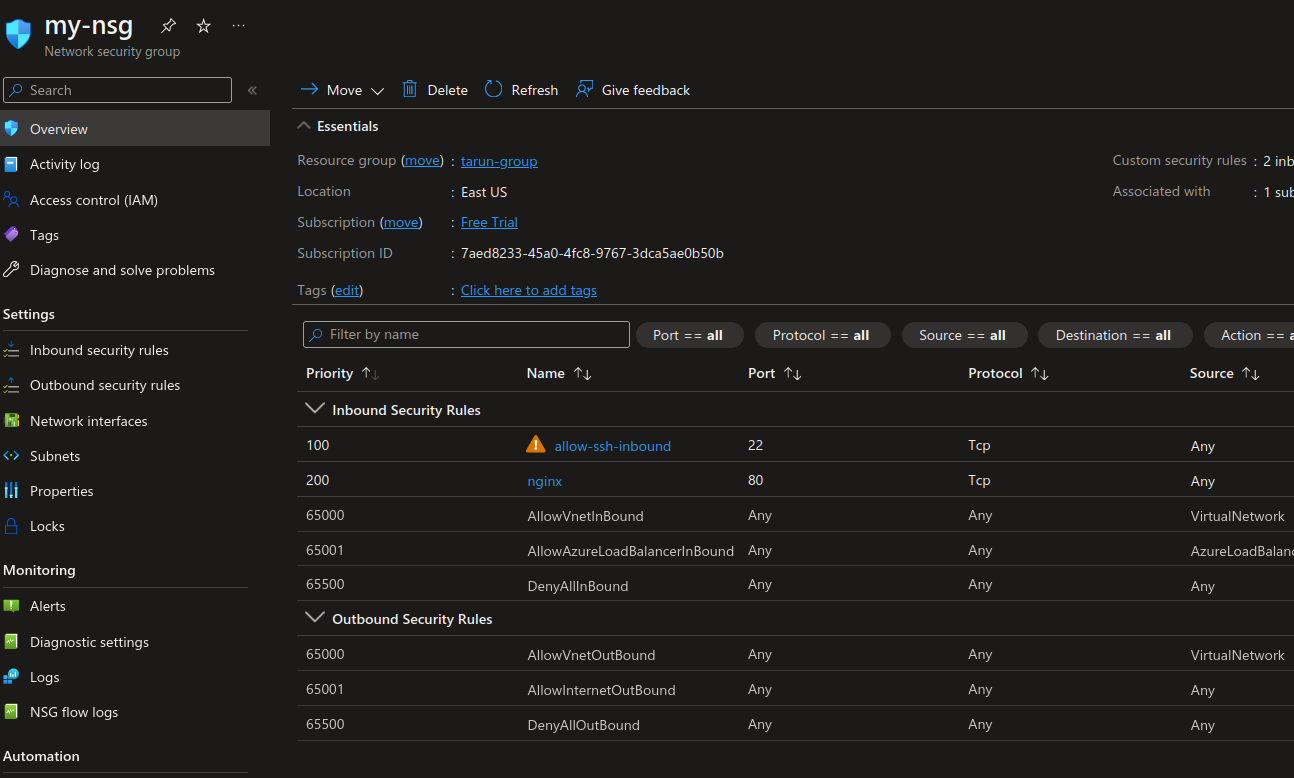
Terraform init: The terraform init command is used to initialize a Terraform working directory. It downloads the necessary provider plugins and sets up the backend configuration. During initialization, Terraform checks for any configuration files in the working directory and automatically downloads the required provider plugins specified in the configuration. This command needs to be executed only once in a new or existing Terraform project.

Terraform plan: The terraform plan command is used to create an execution plan for Terraform. It examines the current configuration and compares it with the deployed infrastructure to determine the changes that need to be made. It generates a detailed report that includes resource creation, modification, or deletion. This command allows you to review the proposed changes before actually applying them, providing an opportunity to catch any errors or unintended modifications.

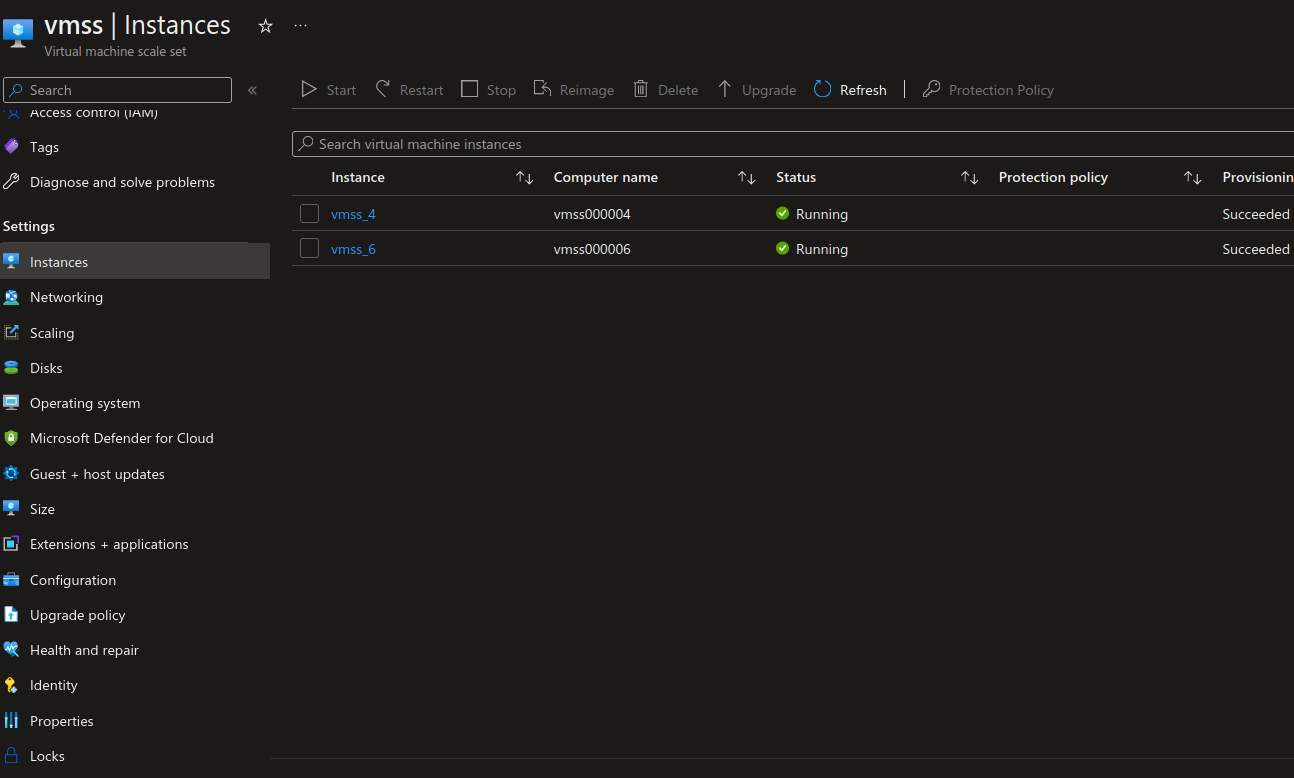
Terraform apply: The terraform apply command is used to apply the changes defined in the Terraform configuration. It creates, modifies, or deletes resources based on the execution plan generated by terraform plan. When running terraform apply, Terraform prompts for confirmation before making any modifications to the infrastructure. It also displays a summary of the changes that will be applied. Once confirmed, Terraform starts provisioning or modifying the resources as specified in the configuration.

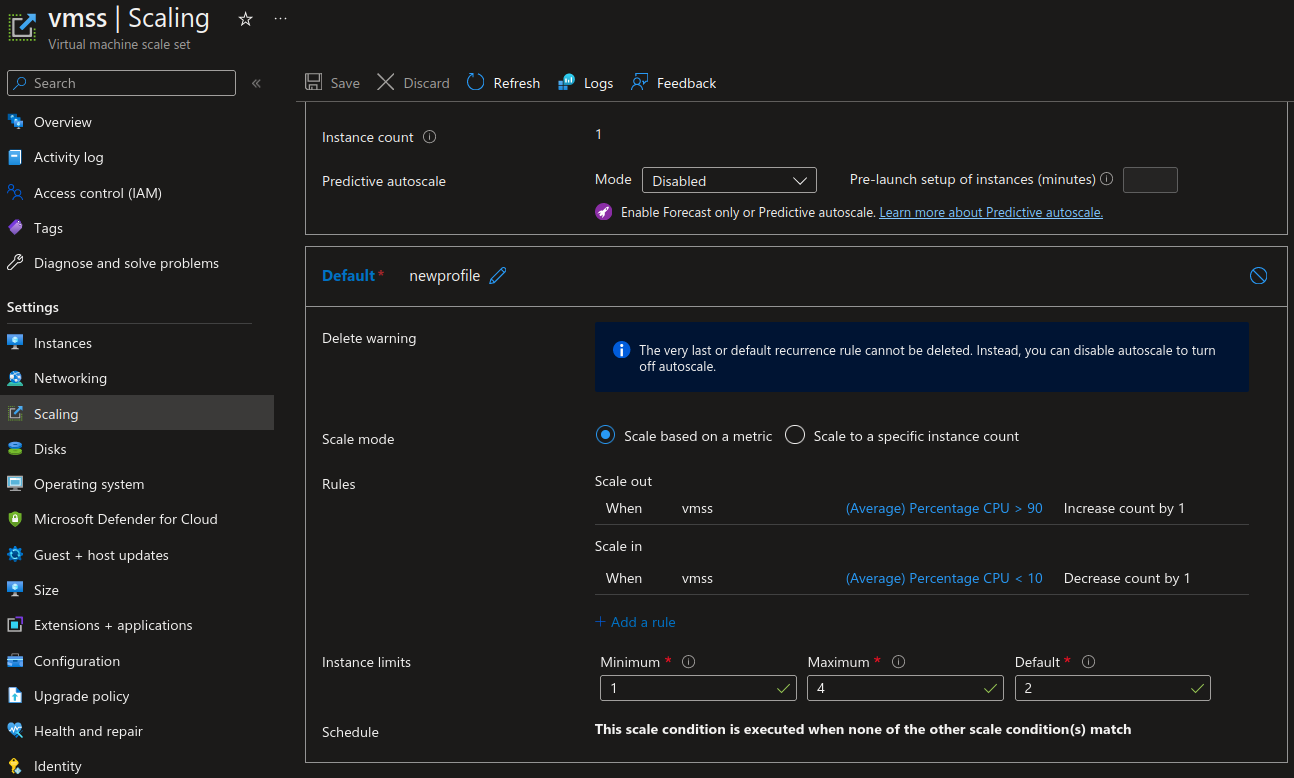
**Resource Group**

**Network Security Group**

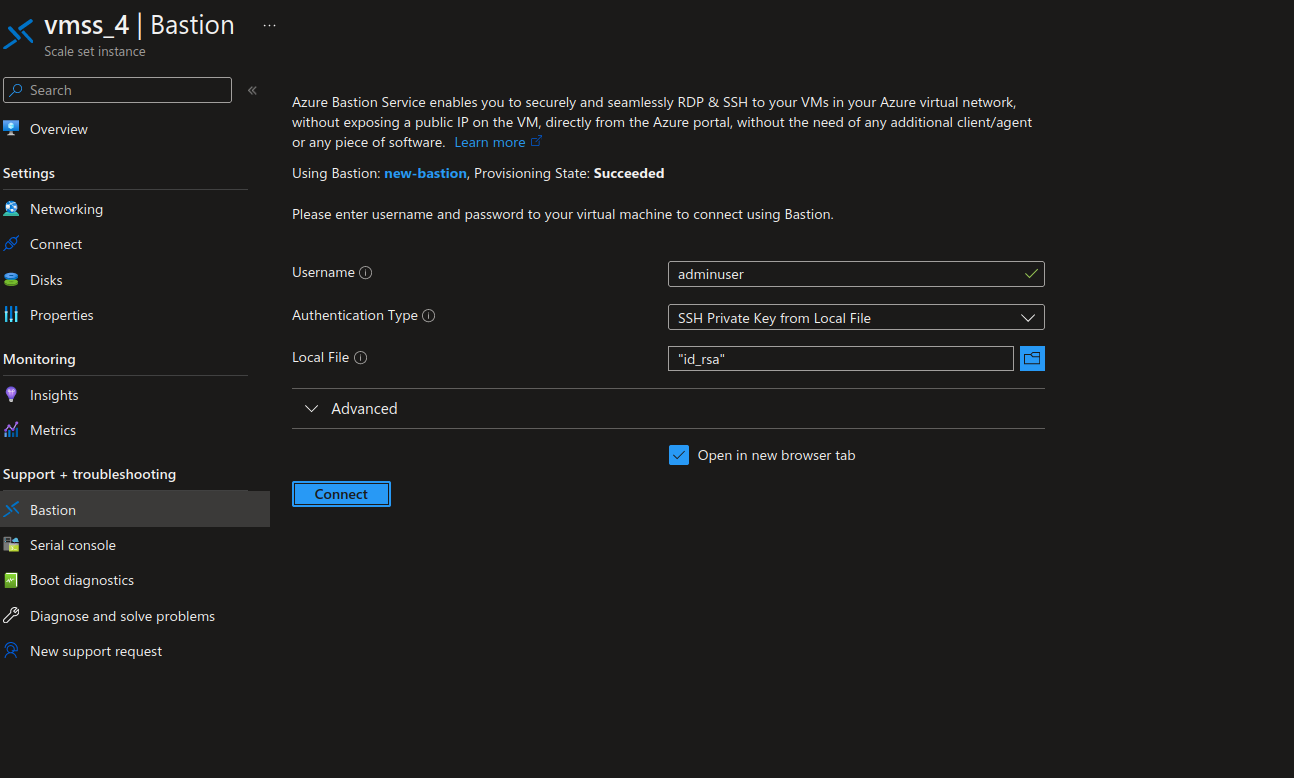


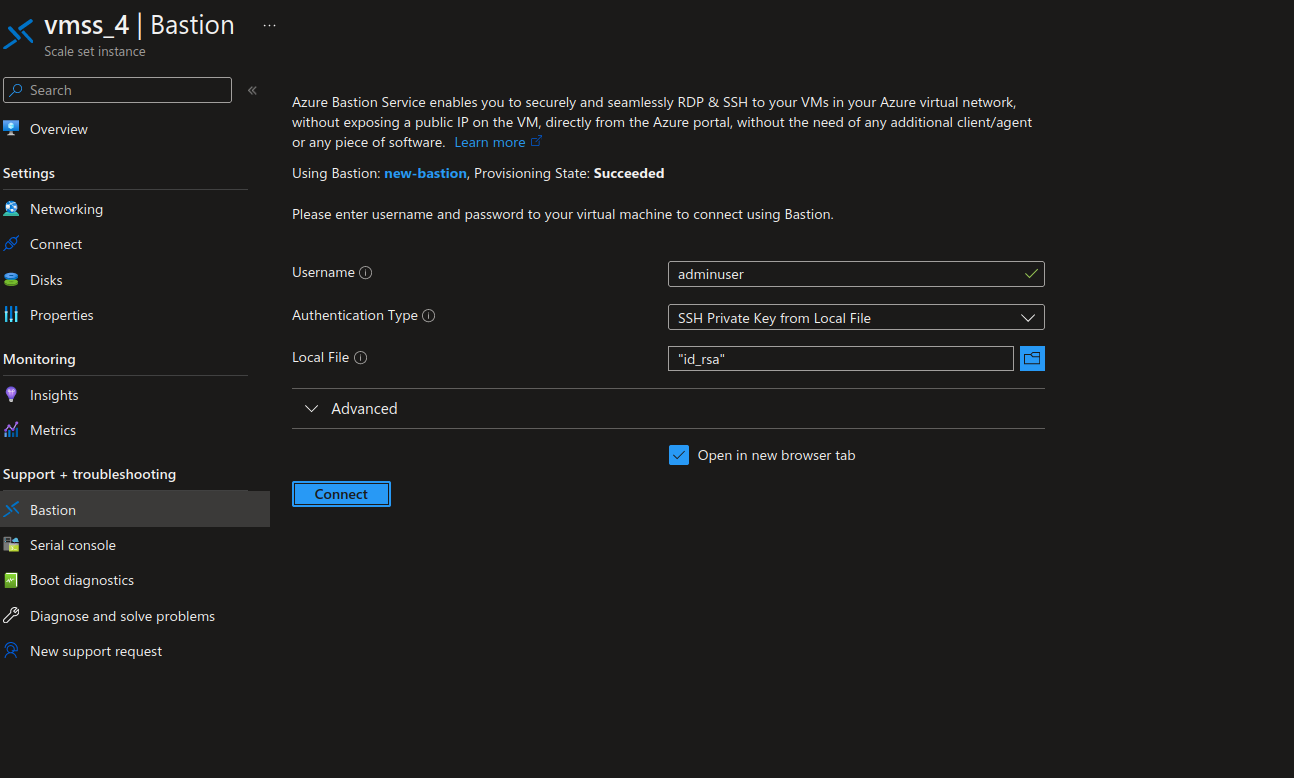
**Scale Set resources and Scaling Rules**

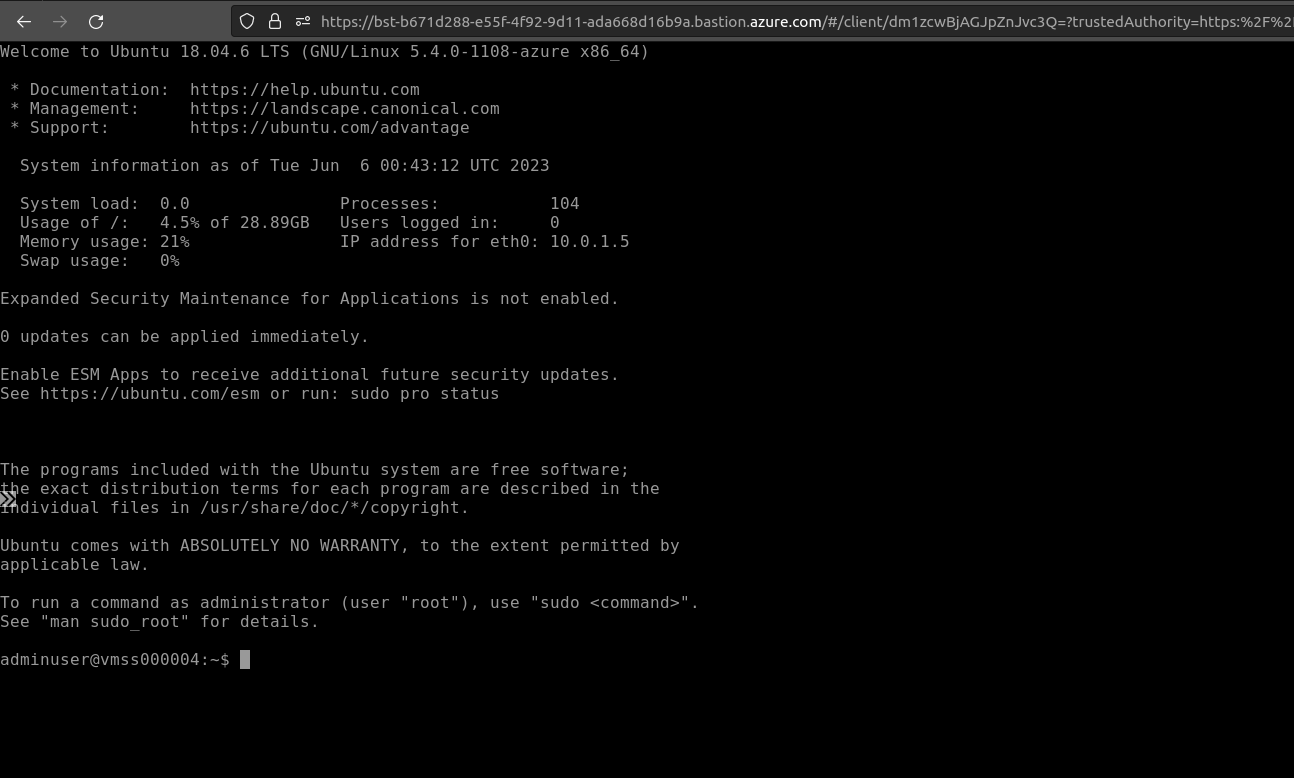
****



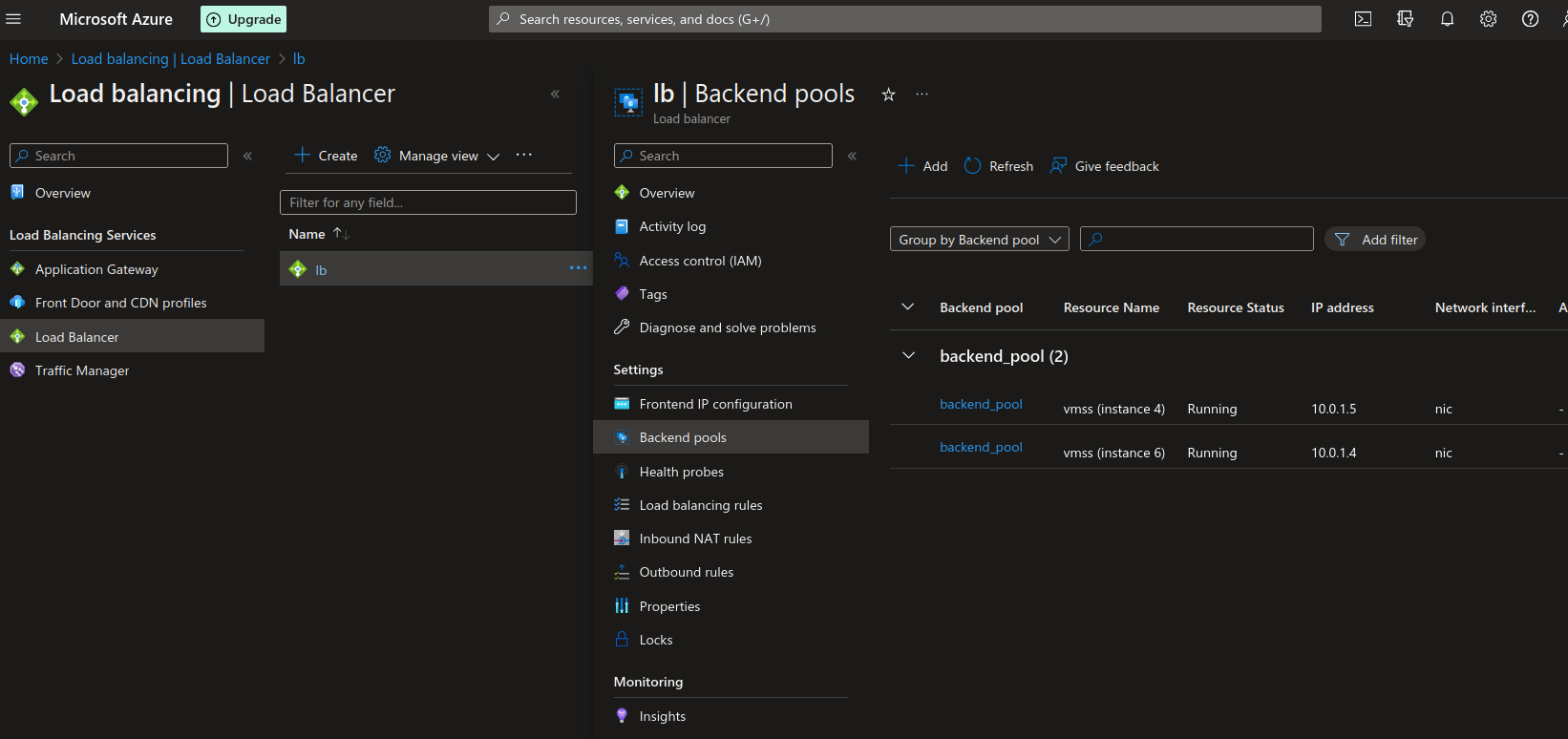
**Bastion**

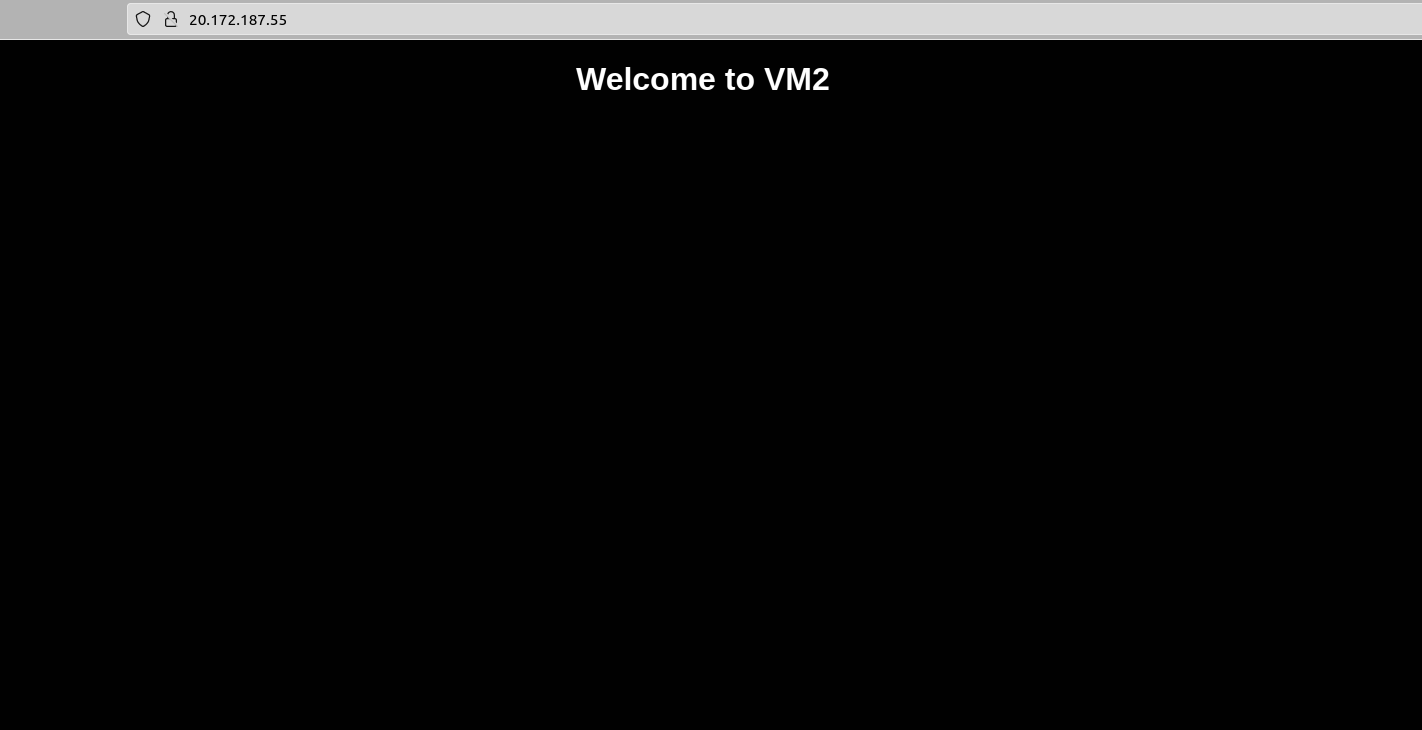






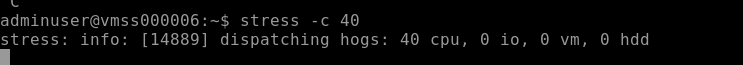
**Installing nginx on VMs in the pool with Load balancer**

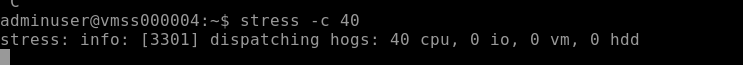




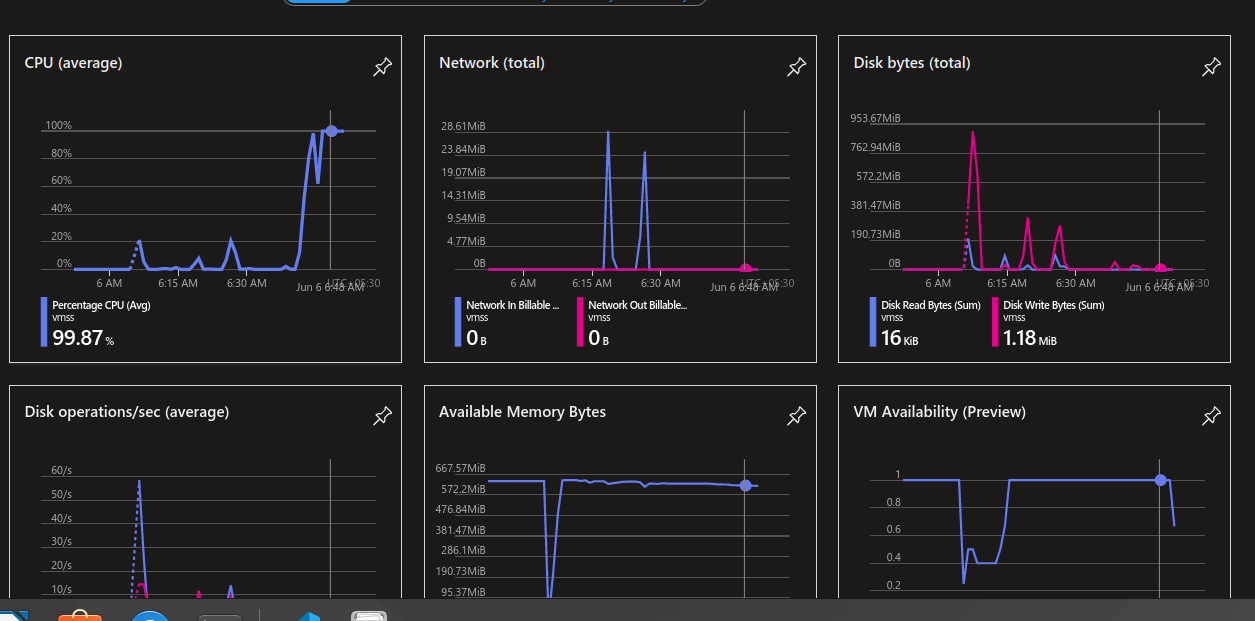
**Building stress to auto-scale**

**Using command stress -c gor cpu utilization**





Monitoring the VMs to check if stress worked



VMs added in the set the moment cpu utilization croos 90%

