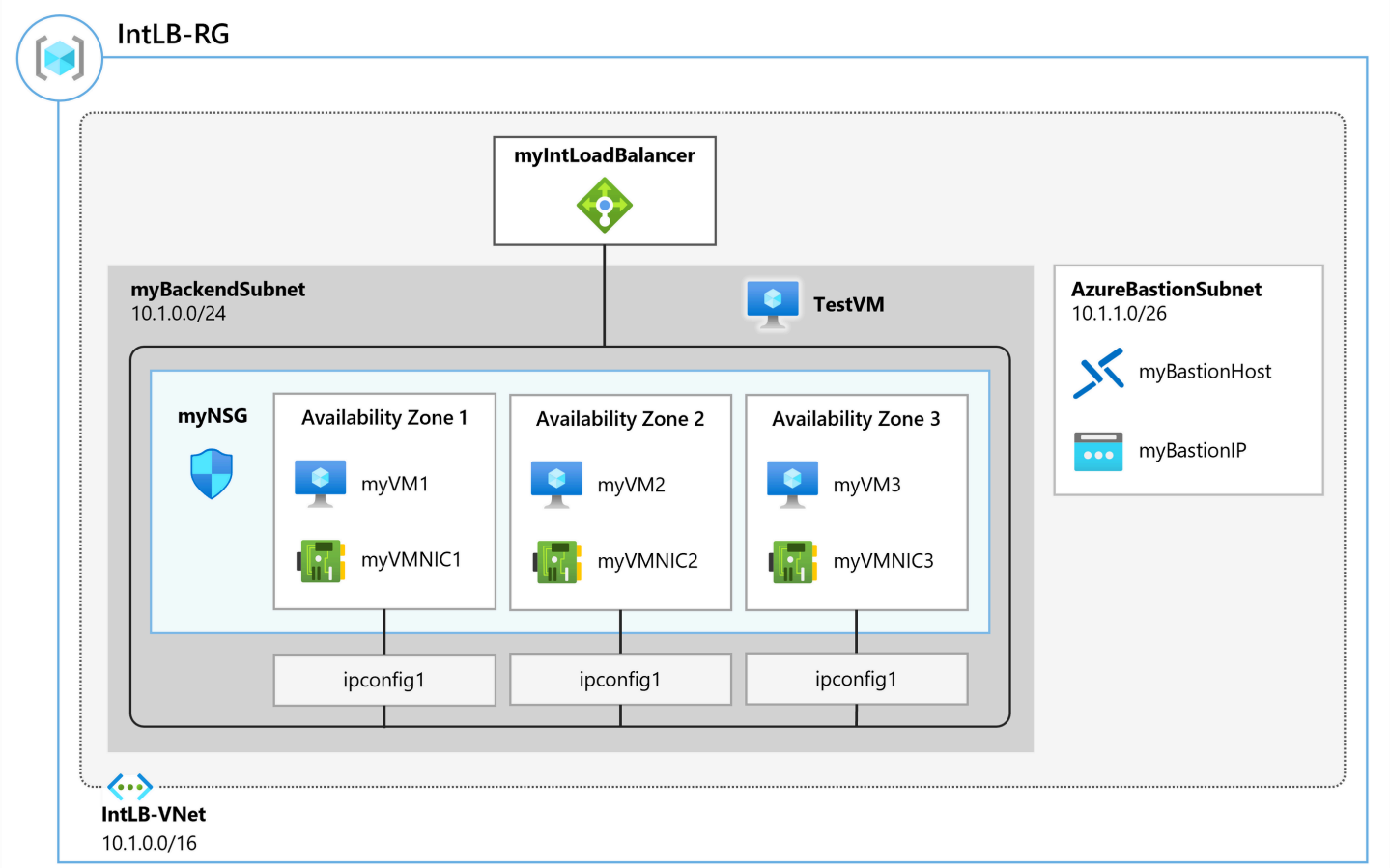
# Mastering Azure Load Balancer & Application Gateway: A Practical Guide to Diverse Traffic Architectures & Extending Traffic Distribution Beyond Web Applications

🤯 Think Traffic Management is Just for Web? Discover the Power of Azure Load Balancer and Application Gateway for Non-HTTP, TCP/UDP, and Advanced Application Delivery! 💡This guide provides clear, actionable steps to configure 🛠️ Azure Load Balancer and Application Gateway for Both Web and Non-Web Applications (TCP/UDP, Advanced Routing, and Security); It breaks down the capabilities of both services by leveraging the strengths of both Azure's foundational traffic management services, empowering you to intelligently route and secure all types of network traffic. Get ready to elevate your Azure networking game!

This lab offers a practical exploration of Azure's crucial load balancing services. *We'll first examine the Azure Load Balancer at Layer 4, demonstrating its use cases through the creation of an internal load balancer. Subsequently, we'll transition to the Azure Application Gateway, a sophisticated Layer 7 solution, illustrating its power in managing and distributing HTTP(S) traffic within your Azure environment. This lab will guide you through the practical steps of deploying an internal Azure Load Balancer. It's important to understand that the infrastructure you set up today forms the foundation for our next lab, where we'll introduce the Azure Application Gateway.* To ensure a smooth transition and avoid redundant deployments, we will retain these resources across both labs. You'll receive instructions for cleanup once both exercises are complete.




Today, I want to address a foundational yet perpetually critical concept in modern distributed systems: **Load Balancing!!!** At its core, **Load Balancing** is the strategic distribution of inbound network traffic across a cluster of backend computing resources. The fundamental objective is multifaceted: to **optimize resource utilization**, thereby ensuring that no single server bears a disproportionate share of the workload. Simultaneously, we strive to **maximize throughput**, effectively increasing the overall capacity of our infrastructure to handle concurrent

requests. Crucially, load balancing aims to **minimize response latency**, providing a consistently performant user experience.

Beyond mere efficiency, **Load Balancing** is a cornerstone of **high availability**. By intelligently distributing workloads across a redundant pool of backend resources, we inherently mitigate the risk of single points of failure. Should one or more instances within the pool become unavailable, the load balancer seamlessly redirects traffic to the remaining healthy nodes, ensuring uninterrupted service delivery.

Interestingly, as my note astutely points out, a well-orchestrated pool of commodity hardware, possessing granular resource allocation, can often exhibit superior responsiveness compared to a monolithic, high-performance single server under heavy load. This phenomenon underscores the efficiency gains inherent in parallel processing and the avoidance of resource contention bottlenecks that can plague even the most powerful single instances.

In essence, **Load Balancing** is not merely about spreading traffic; it's about architecting resilient, scalable, and performant application delivery through intelligent resource management. It is a fundamental building block for any modern, robust, and user-centric digital infrastructure.

**+ How does the Azure load balancer decide to process incoming requests?** The Azure load balancer decides how to process incoming requests based on its configured load-balancing rules and health probes. When a request arrives at the load balancer's frontend, it uses the load-balancing rule to map the incoming traffic (such as a specific port and protocol) to a backend pool of virtual machines or instances. The load balancer then distributes the traffic using a hash-based algorithm that considers factors like the source IP, destination IP, source port, destination port, and protocol type. This ensures that requests from the same client are consistently routed to the same backend instance, providing session persistence if needed.

Additionally, Azure load balancer uses health probes to monitor the status of backend instances. If a backend instance fails the health probe, the load balancer stops sending new requests to that instance until it becomes healthy again. This combination of rules, algorithms, and health checks ensures efficient and reliable distribution of incoming requests.

Load Balancer has several elements that work together to ensure an application's high availability and performance.

💥 **Listeners ->** A listener is a logical entity that checks for incoming connection requests. A listener accepts a request if the protocol, port, hostname, and IP address match the listener's configuration. You must have at least one listener. A listener can be Basic or Multi-site. A Basic listener only routes a request based on the path in the URL. A Multi-site listener can also route requests using the hostname element of the URL. Listeners also handle TLS/SSL certificates for securing your application between the user and Application Gateway.

💥**Front-end IP -> ** The front-end IP address is the address clients use to connect to your web application. A front-end IP address can be either a public or a private IP address. Azure load balancers can have multiple front-end IPs. The selection of a public or a private IP address determines which type of load balancer to create Public IP address: A public load balancer or Private IP address: An internal load balancer.

💥 **Load balancer rules ->** A load balancer rule defines how traffic is distributed to the back-end pool. The rule maps a given front-end IP and port combination to a set of back-end IP addresses and port combination. Traffic is managed using a five-tuple hash made from the following elements: Source IP: Source port: Destination IP: Destination port: Protocol type(TCP or UDP): Load Balancer allows you to load balance services on multiple ports,

multiple IP addresses, or both. You can configure different load balancing rules for each front-end IP. Multiple front-end configurations are only supported with IaaS VMs.

💥 **Back-end pool ->** The back-end pool is a group of VMs or instances in a Virtual Machine Scale Set that responds to the incoming request. To scale cost-effectively to meet high volumes of incoming traffic, computing guidelines generally recommend adding more instances to the back-end pool. Load Balancer implements automatic reconfiguration to redistribute load across the altered number of instances when you scale instances up or down.

💥 **Health probes ->** A health probe is used to determine the health status of the instances in the back-end pool. This health probe determines if an instance is healthy and can receive traffic. You can define the unhealthy threshold for your health probes. When a probe fails to respond, the load balancer stops sending new connections to the unhealthy instances. A probe failure doesn't affect existing connections. The connection continues until: The application ends the flow, Idle timeout occurs, The VM shuts down. Load Balancer allows you to configure different health probe types for endpoints: TCP, HTTP, and HTTPS.

💥 **Session Persistence** specifies how traffic from a client should be handled, which ensures that clients only communicate with a single VM during a session. Session persistence is also known as session affinity, source IP affinity, or client IP affinity. Ensures that the same pool node always handles traffic for a client. When you use session persistence, connections from the same client go to the same back-end instance within the back-end pool. You can configure one of the following session persistence options:

* None (default): The default behavior (None) is that any healthy VM can handle successive requests from a client.

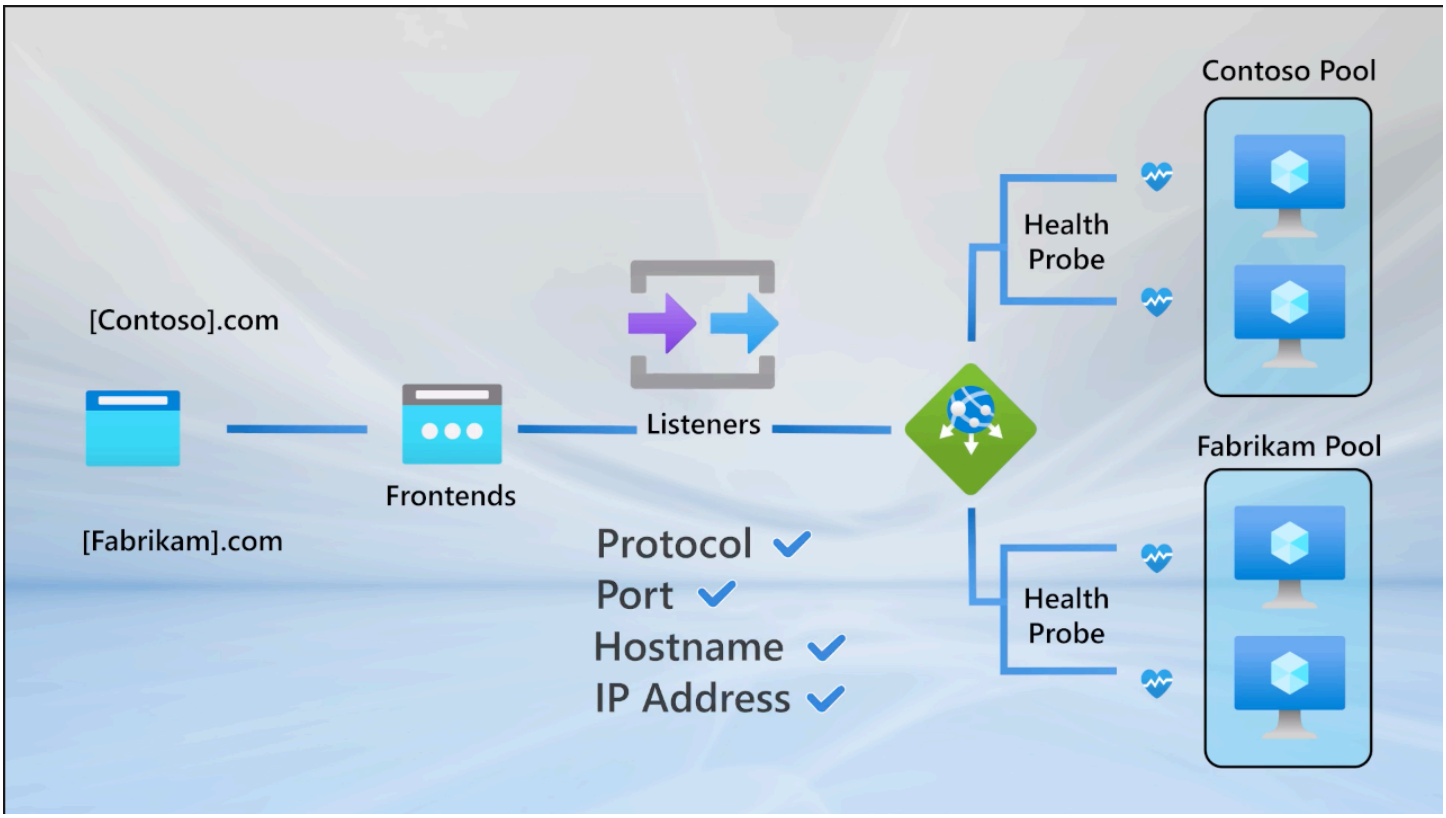
* Client IP (2-tuple)(source IP and destination IP): Specifies that the same back-end instance can handle successive requests from the same client IP address.

* Client IP and protocol (3-tuple)(source IP, destination IP, and protocol type): Specifies that the same back-end instance can handle successive requests from the same client IP address and protocol combination.

💥 **High availability ports ->** A load balancer rule configured with protocol - all(UDP&TCP) and port - 0 is called a high availability (HA) port rule. This rule enables a single rule to load balance all TCP and UDP flows that arrive on all ports of an internal standard load balancer. HA ports load-balancing rules help you with critical scenarios, such as high availability and scale for network virtual appliances (NVAs) inside virtual networks. The feature can help when a large number of ports must be load balanced.

💥 **Inbound NAT rules ->** You can use load balancing rules in combination with Network Address Translation (NAT) rules.

💥**Outbound rules ->** An outbound rule configures Source Network Address Translation (SNAT) for all VMs or instances identified by the back-end pool. This rule enables instances in the back end to communicate (outbound) to the internet or other public endpoints.




In this exercised lab, We will demonstrate the creation of an internal load balancer.

## PART 1

**Azure Load Balancer** is a high-performance, ultra-low-latency **Layer 4 load-balancing** service (inbound and outbound) for all UDP and TCP protocols, that distributes incoming traffic among healthy virtual machine instances. Load balancers uses a hash-based distribution algorithm. By default, it uses a 5-tuple (source IP, source port, destination IP, destination port, protocol type) hash to map traffic to available servers. Load balancers can either be internet-facing where it is accessible via public IP addresses, or internal where it is only accessible from a virtual network. **Azure Load Balancer** is your go-to for robust TCP/UDP traffic management too. Azure load balancers also support **Network Address Translation (NAT)** to route traffic between public and private IP addresses. Its built to handle millions of requests per second while ensuring your solution is highly available. Azure Load Balancer is zone-redundant, ensuring high availability across availability zones.  Azure Load Balancer distributes inbound traffic across a set of VMs in a back-end pool. The back-end pool can be made up of Azure infrastructure as a service (IaaS) VMs or instances in a Virtual Machine Scale Set. Azure Load Balancer is an Azure service that allows you to evenly distribute incoming network traffic across a group of Azure VMs, or across instances in a Virtual Machine Scale Set.

*+ Task 1: Create the virtual network*

*+ Task 2: Create backend servers*

*+ Task 3: Create the load balancer*

*+ Task 4: Create load balancer resources*

*+ Task 5: Test the load balancer*

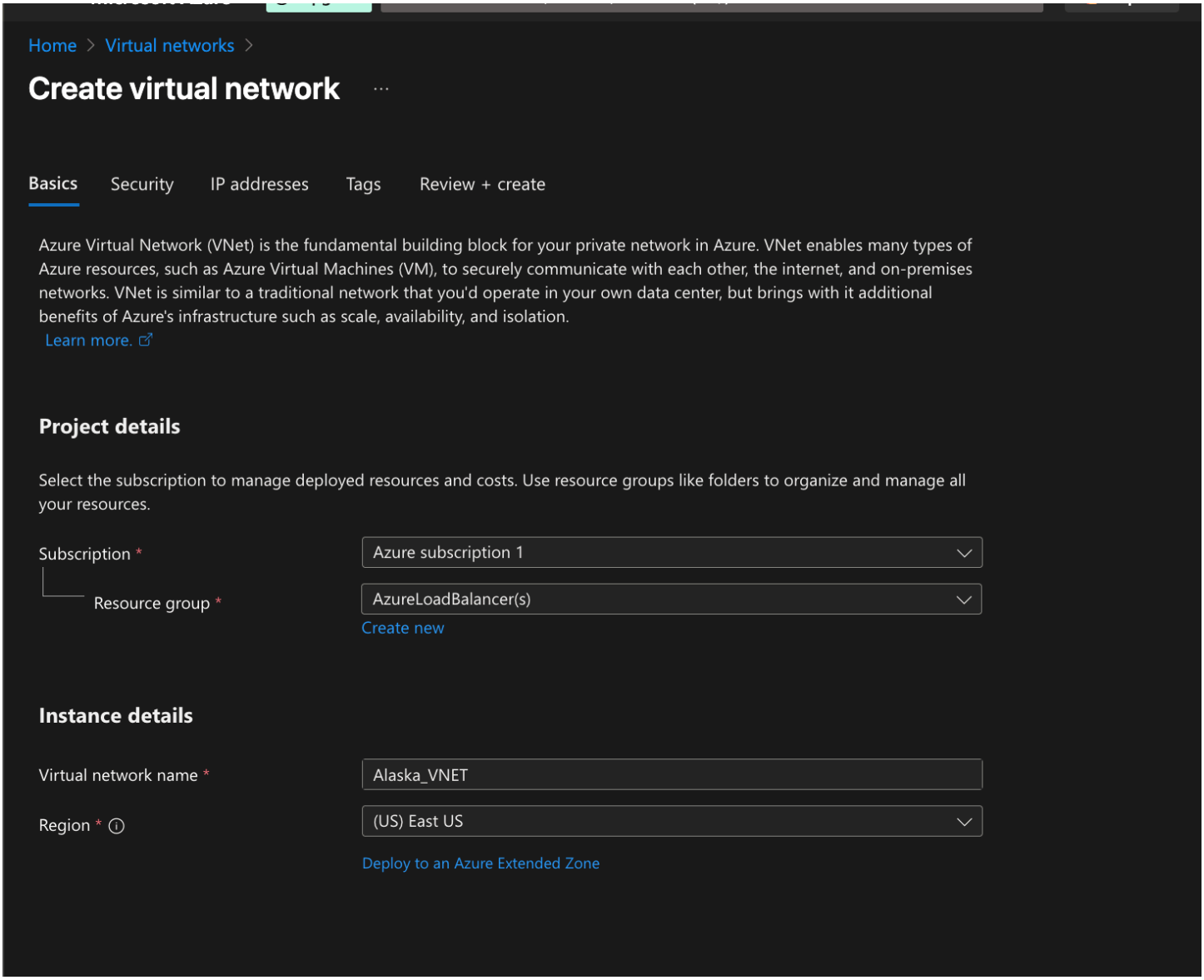
**## Task 1: Create the virtual network**

In this section, you will create a virtual network and a subnet.

A **virtual network (VNet)** in Azure acts as a logically isolated section of the Azure cloud, allowing you to securely connect Azure resources to each other, the internet, and on-premises networks. Within this VNet, you will also create a subnet, which is a range of IP addresses within the VNet. Subnets help organize and isolate resources, control traffic flow, and apply security policies. Creating a VNet and subnet is a fundamental step before deploying resources like virtual machines or load balancers, as it defines the network environment in which those resources will operate.

1. Log in to the Azure portal.
2. On the Azure portal home page, navigate to the Global Search bar and search **Virtual Networks** and select virtual networks under services. ![Azure portal home page Global Search bar results for virtual network.]
3. Select **Create** on the Virtual networks page.
4. On the **Basics** tab, We create the virtual network using the information in the table below

**Subscription | Select your subscription | Resource group: Select Create, Name: AzureLoadBalancer(s) | Name: Alaska_VNET | Region: (US) East US**



Home > Virtual networks >

# Create virtual network ...

Basics | Security | **IP addresses** | Tags | Review + create

Configure your virtual network address space with the IPv4 and IPv6 addresses and subnets you need. Learn more

Define the address space of your virtual network with one or more IPv4 or IPv6 address ranges. Create subnets to segment the virtual network address space into smaller ranges for use by your applications. When you deploy resources into a subnet, Azure assigns the resource an IP address from the subnet. Learn more

+ Add a subnet

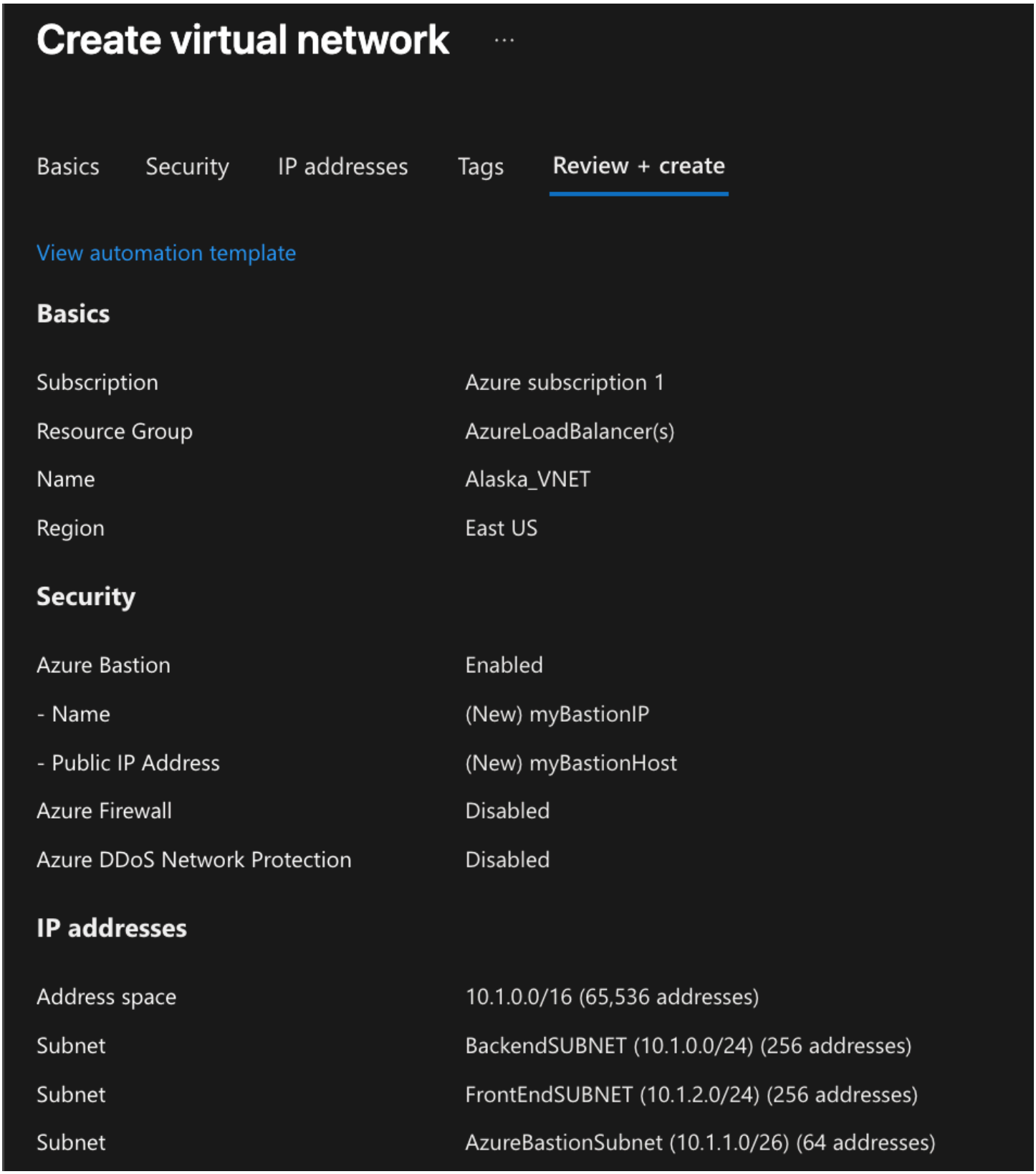
10.1.0.0/16 — Delete address space

10.1.0.0 /16

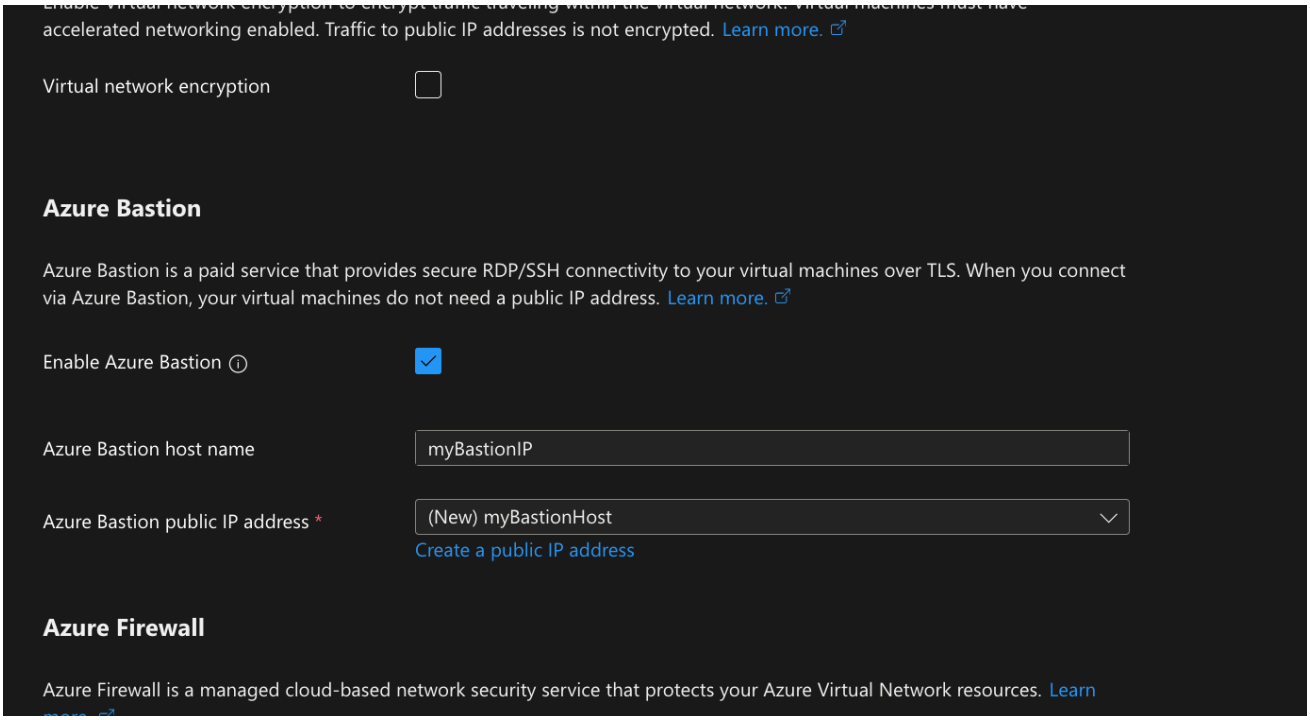
10.1.0.0 - 10.1.255.255 65,536 addresses

| Subnets | IP address range | Size | NAT gateway |
|---|---|---|---|
| BackendSUBNET | 10.1.0.0 - 10.1.0.255 | /24 (256 addresses) | - |
| FrontEndSUBNET | 10.1.2.0 - 10.1.2.255 | /24 (256 addresses) | - |
| AzureBastionSubnet | 10.1.1.0 - 10.1.1.63 | /26 (64 addresses) | - |

Add IPv4 address space




1. Select **Next : IP Addresses**
2. On the **IP Addresses** tab, in the **IPv4 address space** box, remove the default and enter **10.1.0.0/16**.
3. On the **IP Addresses** tab, select **+ Add subnet**.
4. In the **Add subnet** pane, provide a subnet name of **BackendSubnet**, and a subnet address range of **10.1.0.0/24**.
5. Select **Add**.
6. Select **Add subnet**, provide a subnet name of **FrontEndSubnet**, and a subnet address range of **10.1.2.0/24**. Select **Add**
7. Select **Next : Security**.
8. Under **BastionHost** select **Enable**, then enter the information from the table below.

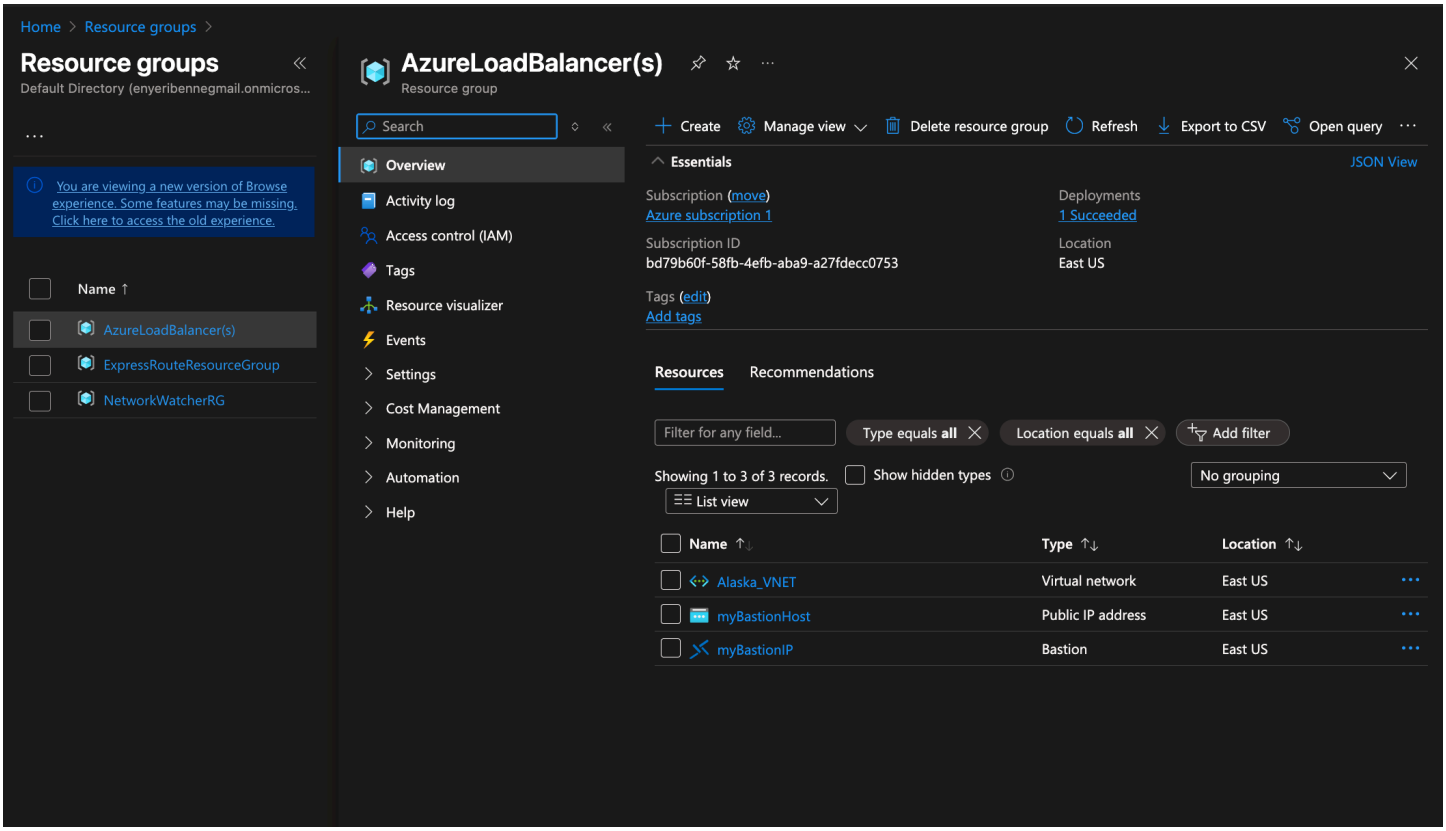



| Bastion name: myBastionHost | AzureBastionSubnet address space | **10.1.1.0/26** | Public IP address | Select **Create new** Name: **myBastionIP**

Select **Review + create**, Select **Create**.

A **Bastion Host Subnet** is created to securely manage virtual machines (VMs) in your Azure virtual network without exposing them directly to the public internet.

Why Create a **Bastion Host Subnet** In this lab? you'll deploy backend servers and a load balancer. The Bastion Host subnet allows you to Remotely access VMs deployed in the BackendSubnet for configuration and troubleshooting, even though they don't have public IPs. And to test connectivity and load balancer behavior from a secure, internal point in the network. Without a Bastion Host, you'd have to expose your VMs to the internet, which is not recommended for production or secure environments. The Bastion Host provides a secure, browser-based way to connect to your VMs directly from the Azure portal.




## Task 2: Create backend servers

In this section, we will create three VMs, that will be in the same availability set, for the backend pool of the load balancer, add the VMs to the backend pool, and then install IIS on the three VMs to test the load balancer. This section outlines the steps to set up the backend resources for an Azure load balancer. We will create three virtual machines (VMs) and place them in the same availability set. An availability set ensures that the VMs are distributed across multiple physical hardware resources within the Azure datacenter, increasing their resilience to hardware failures and planned maintenance.

These VMs will be added to the backend pool of the load balancer. The backend pool is a group of resources that receive traffic distributed by the load balancer. By adding the VMs to this pool, you enable the load balancer to route incoming requests to any of the VMs, helping to balance the load and improve application availability.

Finally, you will install Internet Information Services (IIS) on each VM. IIS is a web server, and installing it allows you to easily test the load balancer by accessing the web server on each VM through the load balancer's frontend IP address. This setup helps verify that the load balancer is correctly distributing traffic among the backend VMs.

1. In the Azure portal, select the Cloud Shell icon (top right). If necessary, configure the shell.

    + Select **PowerShell**.

    + Select **No Storage Account required** and your **Subscription**, then select **Apply**.

    + Wait for the terminal to create and a prompt to be displayed.

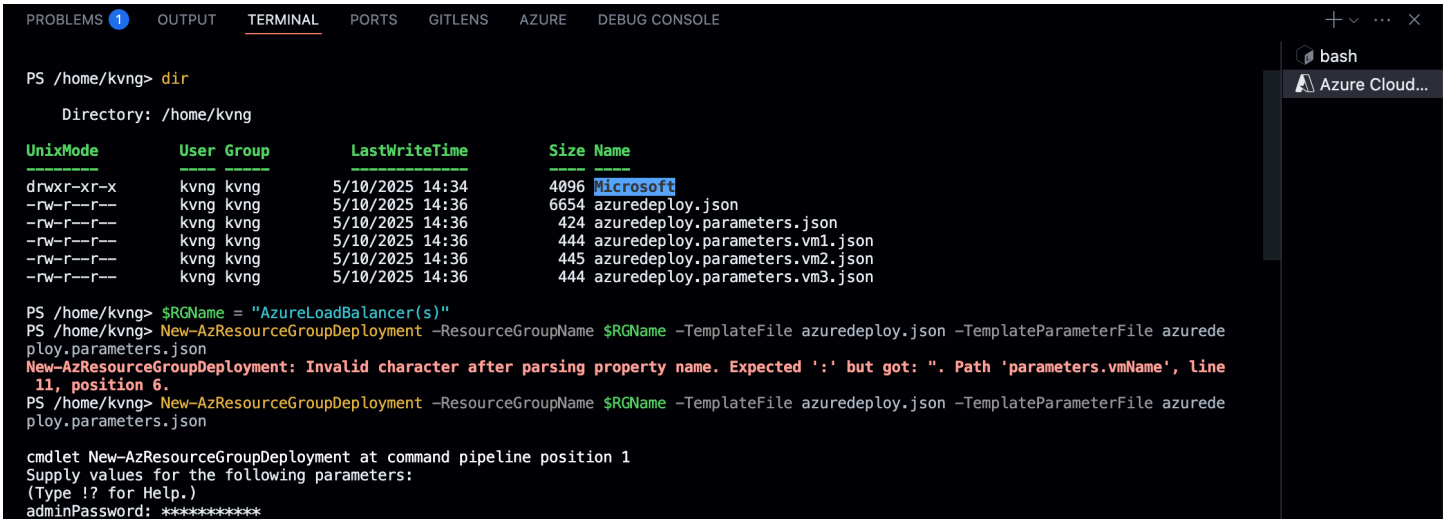
2. On the toolbar of the Cloud Shell pane, select the **Upload/Download files** icon, in the drop-down menu, select **Upload** and upload the following files azuredeploy.json, and azuredeploy.parameters.json into the Cloud Shell home directory one by one.

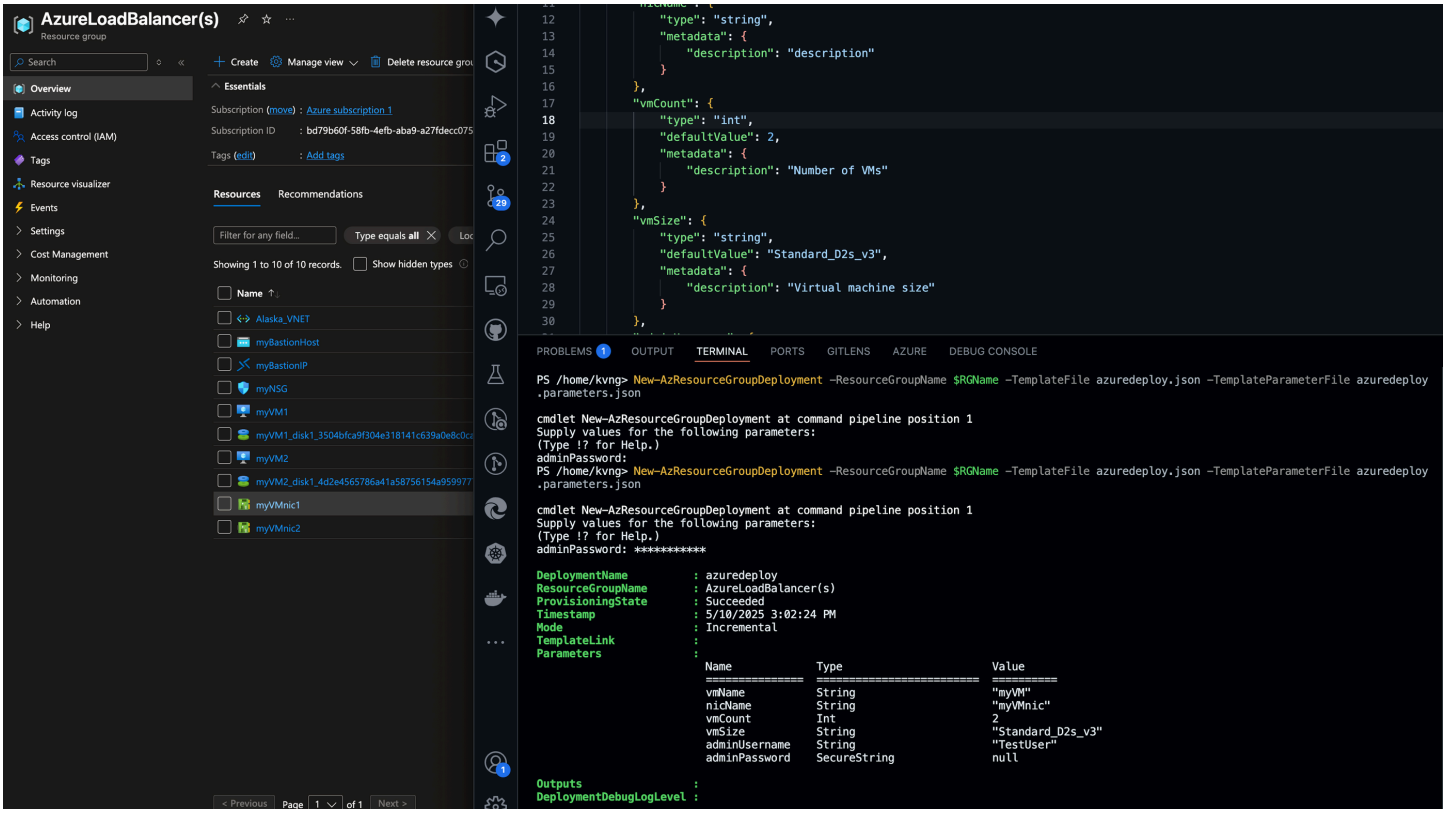
3. Deploy the following ARM templates to create the VMs needed for this exercise:

>**Note**: You will be prompted to provide an Admin password.

New-AzResourceGroupDeployment -ResourceGroupName $RGName -TemplateFile azuredeploy.json -TemplateParameterFile azuredeploy.parameters.json

It may take 5-10 min to create these three VMs. You do not have to wait until this job completes, you can continue with the next task already.




## Task 3: Create the load balancer

This section introduces the creation of an internal Standard SKU load balancer in Azure. The choice to use the Standard SKU, rather than the Basic SKU, is intentional and based on requirements for later exercises. The Standard SKU load balancer offers advanced features, improved security, and broader compatibility compared to the Basic SKU. For example, it supports availability zones, better integration with network security groups, and higher scalability. By starting with the Standard SKU, you ensure that your environment will be

compatible with more advanced scenarios and configurations that may be covered in subsequent parts of the lab or course.

1. On the Azure portal home page, select **Create a resource**.

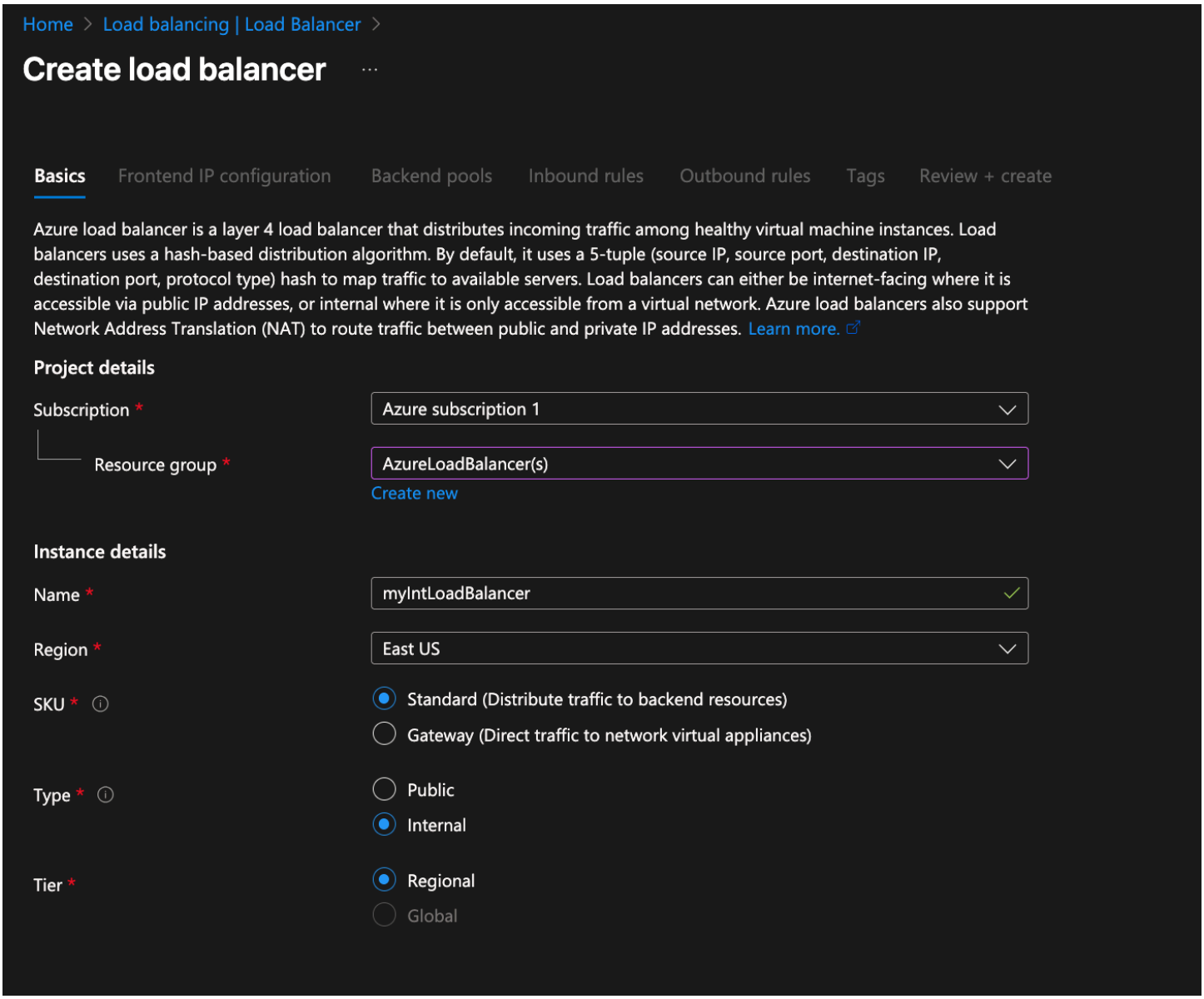
2. On the search box at the top of the page, enter **Load Balancer**, then press **Enter** (**Note:** do not select one from the list).

3. On the results page, locate and select **Load Balancer** (the one that says 'Microsoft' and 'Azure Service' under the name).

4. Select **Create**.

5. On the **Basics** tab, use the information in the table below to create the load balancer.

| **Setting** | **Value** |
| --- | --- |
| Subscription | Select your subscription |
| Resource group | **AzureLoadBalancer(s)** |
| Name | **myIntLoadBalancer** |
| Region | **(US) East US** |
| SKU | **Standard** |
| Type | **Internal** |
| Tier | **Regional** |




6. Select **Next: Frontend IP configurations**.

7. Select Add a frontend IP

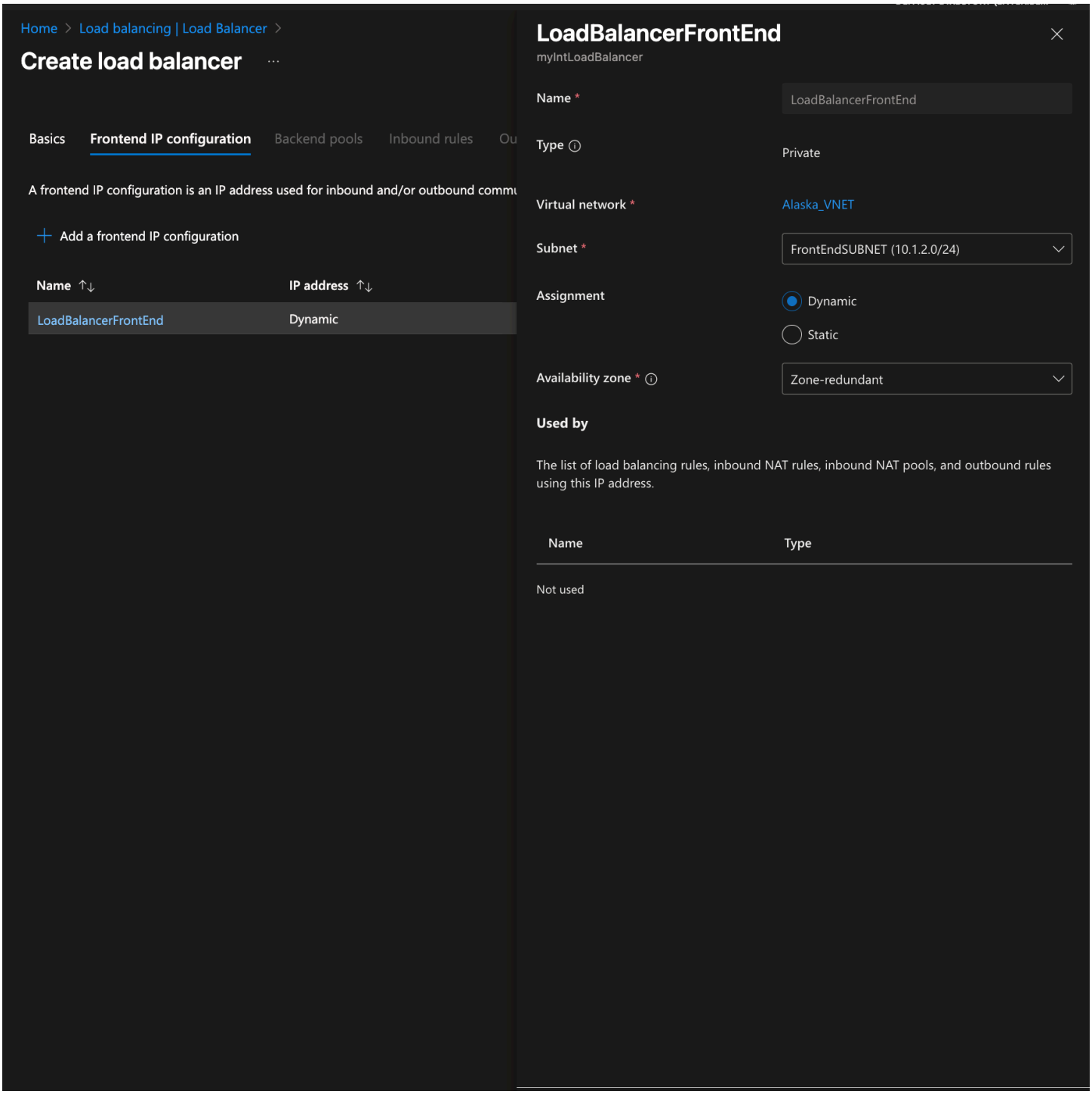
8. On the **Add frontend IP address** blade, enter the information from the table below and select **Add**.

| **Setting** | **Value** |
| --- | --- |
| Name | **LoadBalancerFrontEnd** |
| Virtual network | **Alaska_VNET** |
| Subnet | **myFrontEndSubnet** |
| Assignment | **Dynamic** |

The front-end IP address is the address clients(by client i meant the TestVM or BastionHosts) use to connect to your web application. A front-end IP address can be either a public or a private IP address. Azure load balancers can have multiple front-end IPs. The selection of a public or a private IP address determines which type of load balancer to create Public IP address: A public load balancer or Private IP address: An internal load balancer.

9. Select **Review + create**.

10. Select **Create**.




## Task 4: Create load balancer resources

In this section, you will configure load balancer settings for a backend address pool, then create a health probe and a load balancer rule.

### Create a backend pool and add VMs to the backend pool

The back-end pool is a group of VMs or instances in a Virtual Machine Scale Set that is tied to the load balancer. It ensures that incoming requests are distributed efficiently among the VMs in the pool. The Load Balancer also implements automatic reconfiguration to redistribute traffic when the number of instances in the backend pool changes due to scaling operations.

The backend address pool contains the dynamically assigned IP addresses of the virtual NICs of the VMs in the BackEndSubnet that will be connected to the load balancer.

1. On the Azure portal home page, select **All resources**, then select on **myIntLoadBalancer** from the resources list.

2. Under **Settings**, select **Backend pools**, and then select **Add**.

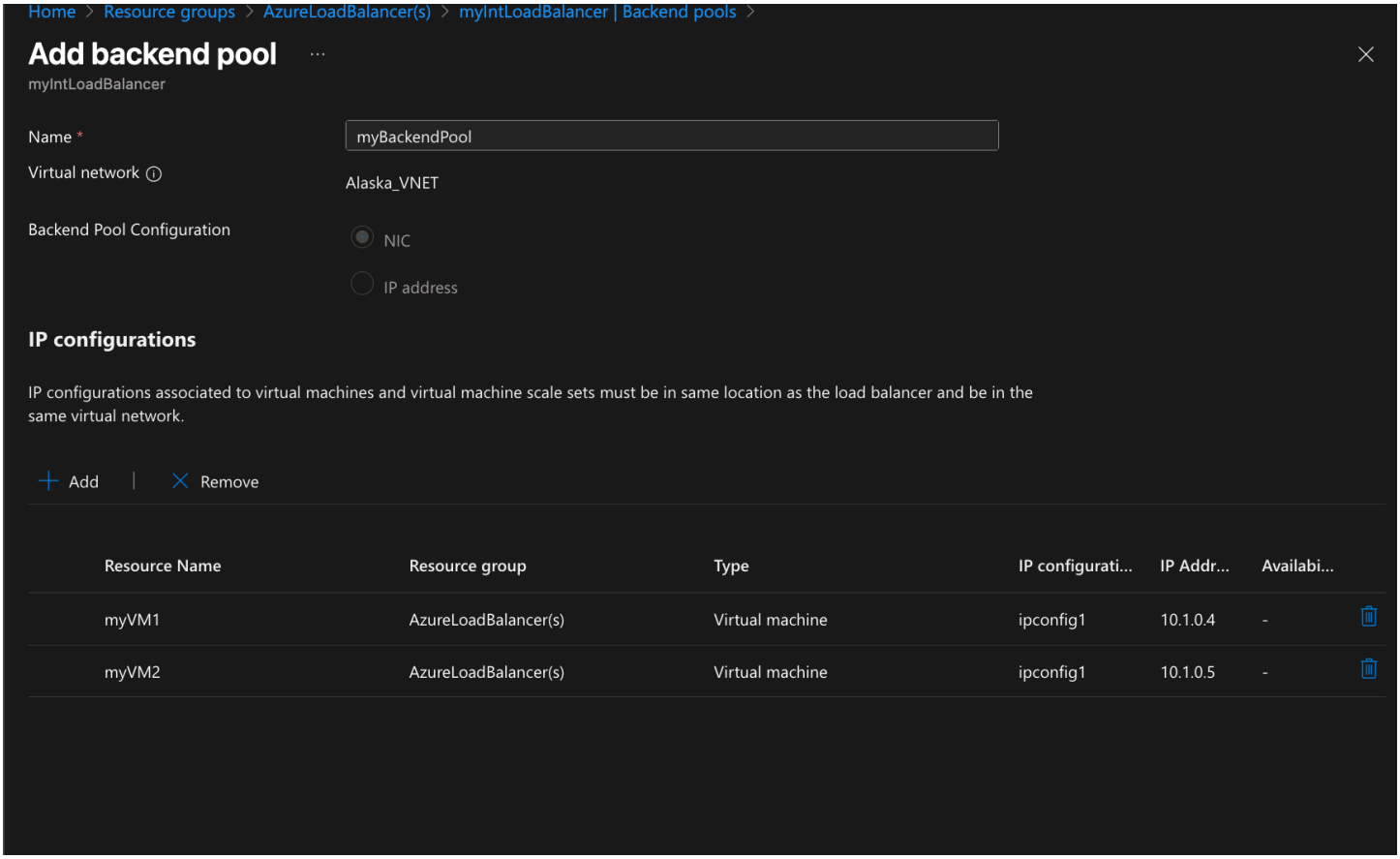
3. On the **Add backend pool** page, enter the information from the table below.

| **Setting** | **Value** |
| --- | --- |
| Name | **myBackendPool** |
| Virtual network | **Alaska_VNET** |

4. Under **Virtual machines**, select **Add**.

5. Select the checkboxes for all 3 VMs (**myVM1**, **myVM2**), then select **Add**.

6. Select **Save**.




**### Create a health probe**

A health probe is used to determine the health status of the instances in the back-end pool. This health probe determines if an instance is healthy and can receive traffic. You can define the unhealthy threshold for your health probes. When a probe fails to respond, the load balancer stops sending new connections to the unhealthy instances. A probe failure doesn't affect existing connections. The connection continues until: The application ends the flow, Idle timeout occurs, The VM shuts down. Load Balancer allows you to configure different health probe types for endpoints: TCP, HTTP, and HTTPS.

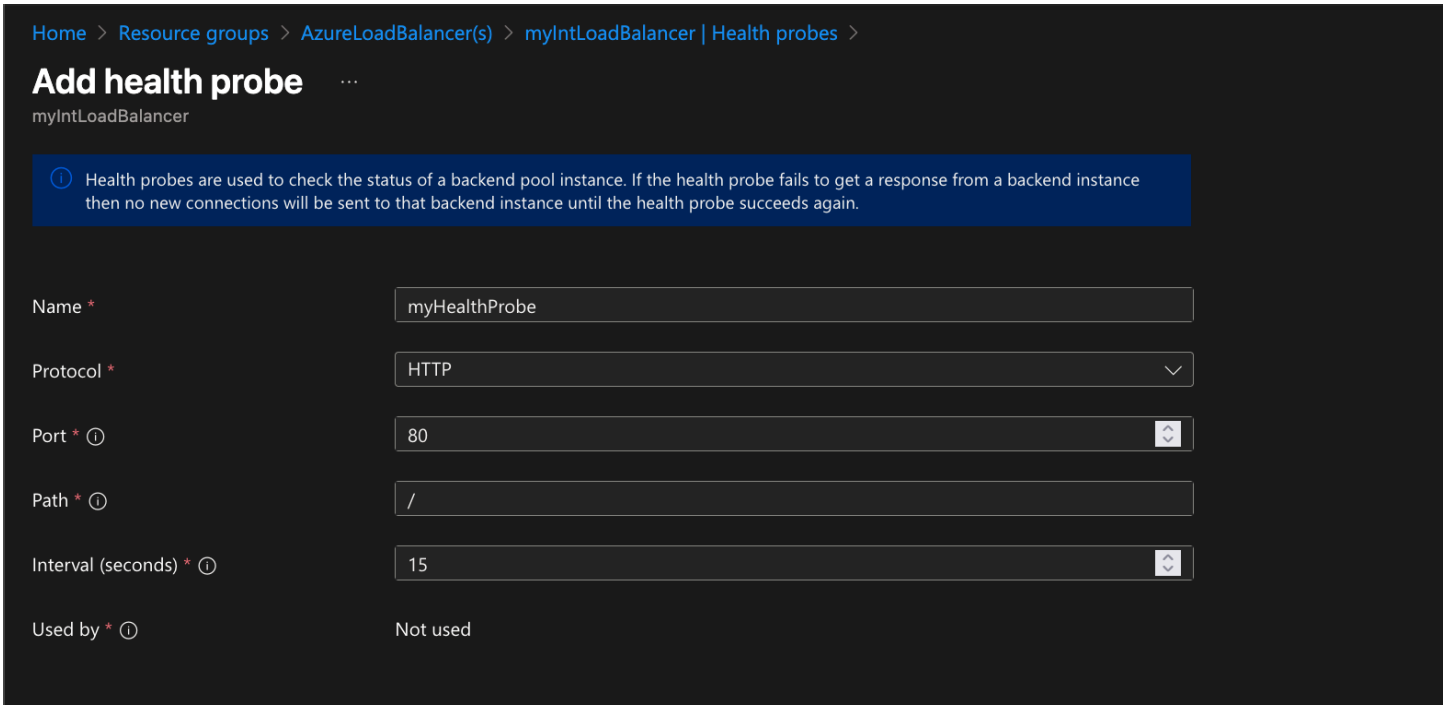
The load balancer monitors the status of your app with a health probe. The health probe adds or removes VMs from the load balancer based on their response to health checks. Here you will create a health probe to monitor the health of the VMs.

1. Under **Settings**, select **Health probes**, then select **Add**.

2. On the **Add health probe** page, enter the information from the table below.

| **Setting** | **Value** |
| --- | --- |
| Name | **myHealthProbe** |
| Protocol | **HTTP** |
| Port | **80** |
| Path | **/** |
| Interval | **15** |

3. Select **Add**.




### Create a load balancer rule

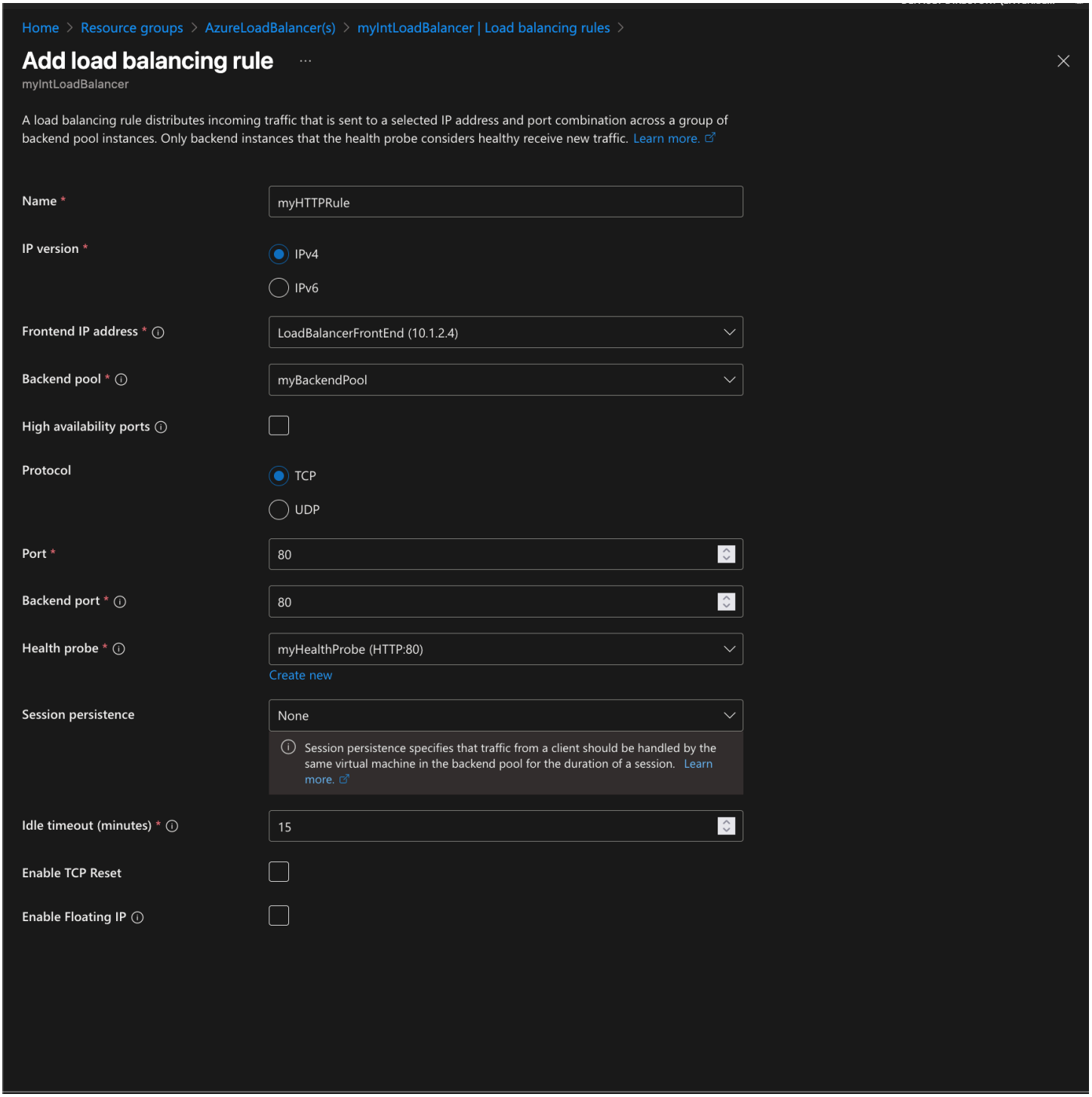
A load balancer rule defines how traffic is distributed to the back-end pool of VMS. The rule maps a given front-end IP and port combination to a set of back-end IP addresses and port combination. Traffic is managed using a 5-tuple/3-tuple/2-tuple hash made from the following elements: Source IP: Source port: Destination IP: Destination port: Protocol type(TCP or UDP): Load Balancer allows you to load balance services on multiple ports, multiple IP addresses, or both. You can configure different load balancing rules for each front-end IP. Multiple front-end configurations are only supported with IaaS VMs.You define the frontend IP configuration for the incoming traffic and the backend IP pool to receive the traffic. The source and destination port are defined in the rule. Here you will create a load balancer rule.

1. Under **Settings**, select **Load balancing rules**, then select **Add**.

2. On the **Add load balancing rule** page, enter the information from the table below.

| **Setting** | **Value** |
| --- | --- |
| Name | **myHTTPRule** |
| IP Version | **IPv4** |
| Frontend IP address | **LoadBalancerFrontEnd** |
| Backend pool | **myBackendPool** |
| Protocol | **TCP** |
| Port | **80** |

| Backend port | **80** |
| Health probe | **myHealthProbe** |
| Session persistence | **None** |
| Idle timeout (minutes) | **15** |
| Floating IP | **Disabled** |




## Task 5: Test the load balancer

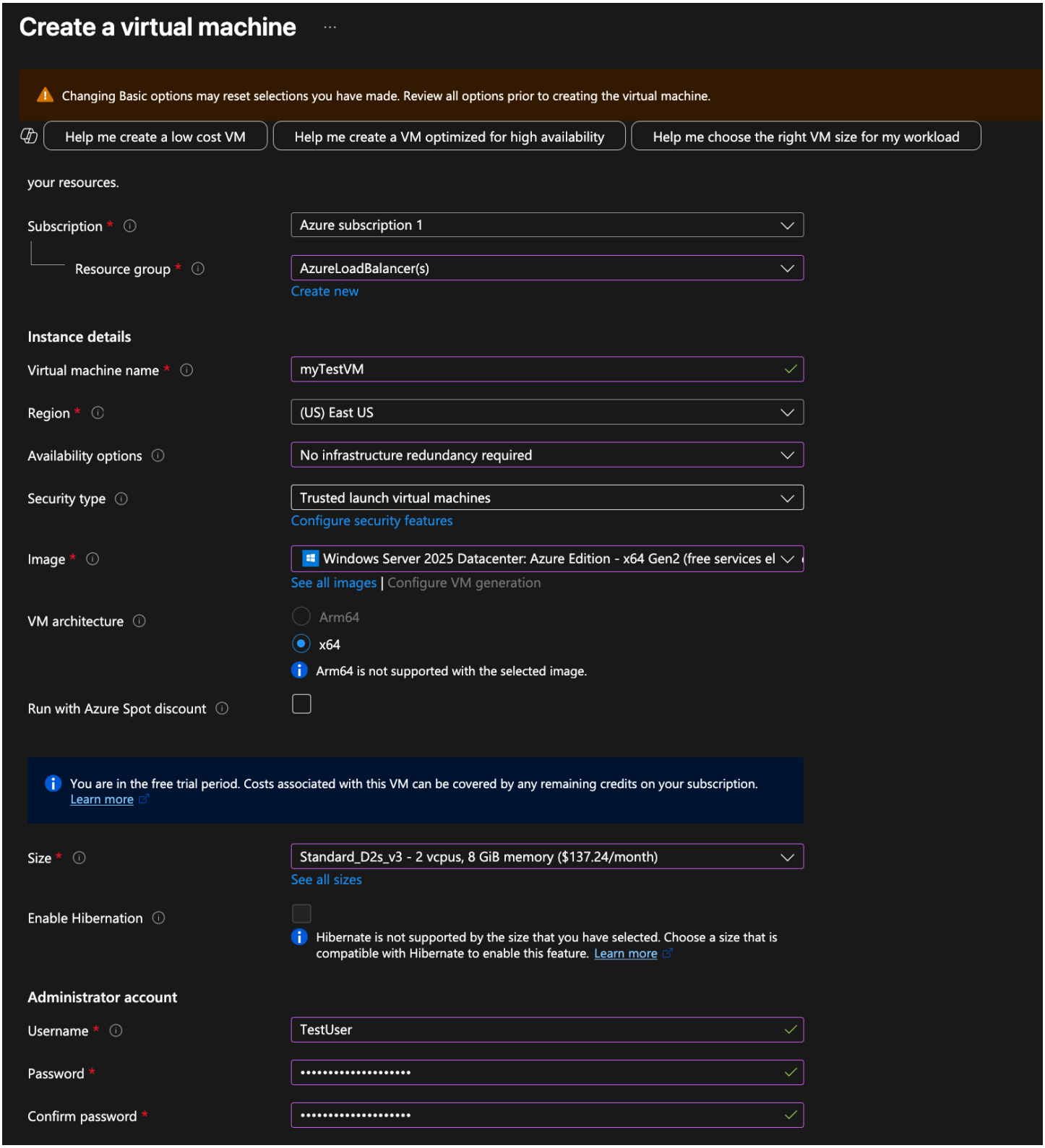
In this section, you will create a test VM, and then test the load balancer.

### Create test VM

1. On the Azure portal home page, select **Create a resource**, then **virtual**, then select **Virtual machine** (if this resource type is not listed on the page, use the search box at the

top of the page to search for it and select it).

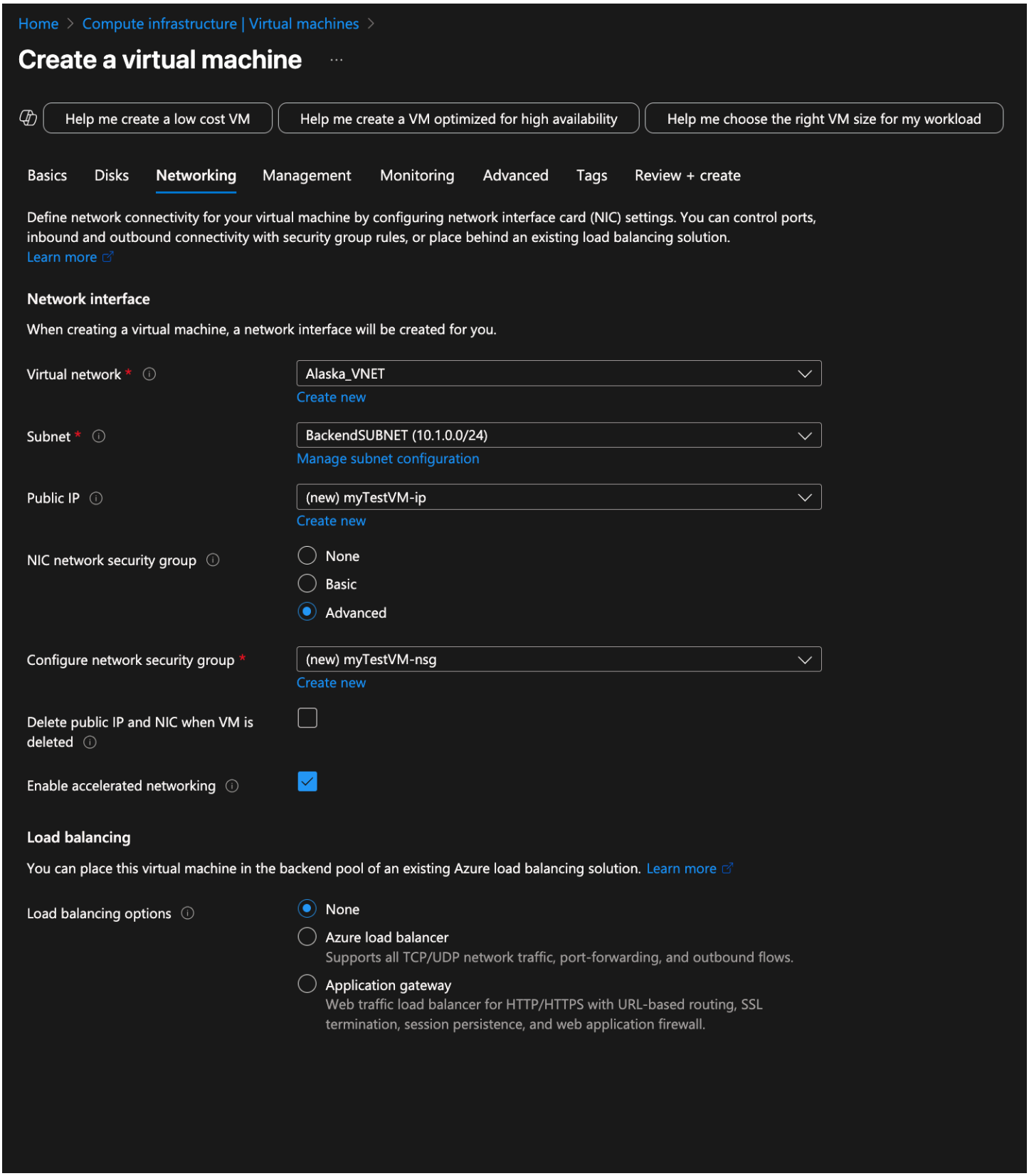
2.On the **Create a virtual machine** page, on the **Basics** tab, use the information in the table below to create the first VM.




| **Setting** | **Value** |
| --- | --- |
| Subscription | Select your subscription |
| Resource group | **AzureLoadBalancer(s)** |
| Virtual machine name | **myTestVM** |
| Region | **(US) East US** |

| Availability options | **No infrastructure redundancy required** |
| Image | **Windows Server 2025 Datacenter - Gen 2** |
| Size | **Standard_DS2_v3 - 2 vcpu, 8 GiB memory** |
| Username | **TestUser** |
| Password | **Provide a secure password** |
| Confirm password | **Provide a secure password** |

1. Select **Next : Disks**, then select **Next : Networking**.



On the **Networking** tab, use the information in the table below to configure networking settings.

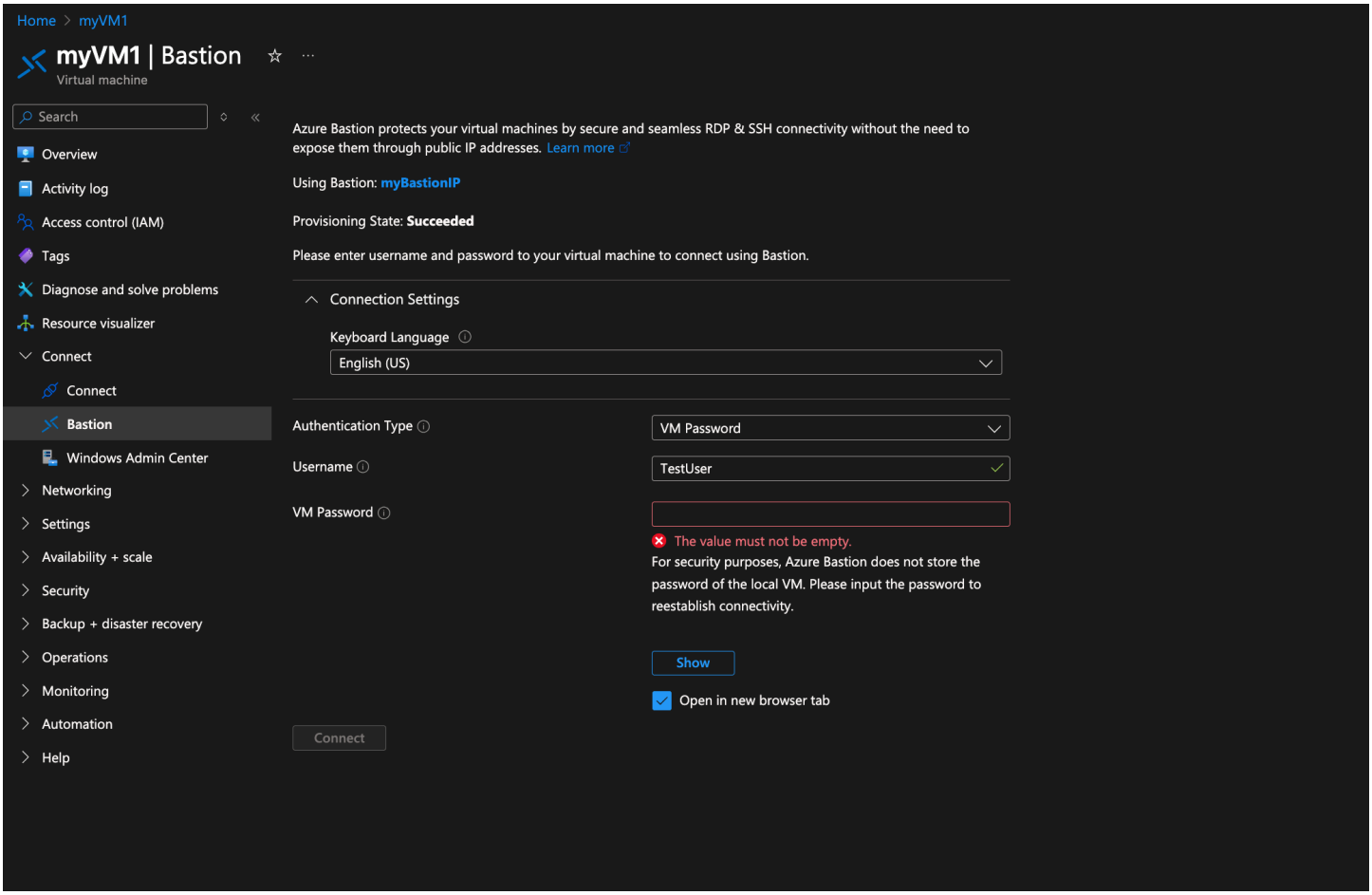
| **Setting** | **Value** |
| --- | --- |
| Virtual network | **Alaska_VNET** |
| Subnet | **myBackendSubnet** |
| Public IP | Change to **None** |
| NIC network security group | **Advanced** |
| Configure network security group | Select the existing **myNSG** |
| Load balancing options | **None** |

**Note:** If you are unable to deploy additional VMs due to quota limits or subscription restrictions, you can still test the load balancer by using one of the existing backend VMs. Connect to one of the backend VMs (e.g., **myVM1**) using Bastion or RDP, open a web browser, and access the internal load balancer's private IP address. This will allow you to verify that the load balancer is distributing traffic among the backend pool VMs. Alternatively, you can use tools like PowerShell or `curl` from within any VM in the same virtual network to test connectivity to the load balancer's frontend IP.

### Alternative ways to test the load balancer

If you are unable to deploy the test VM or prefer a different approach, you can test the internal load balancer using any VM within the same virtual network. Here are some alternative methods:

1. **Connect to an existing backend VM** (such as **myVM1**, **myVM2**, or **myVM3**) using Bastion or RDP.



2. Open a web browser on the VM and enter the internal load balancer's private IP address in the address bar. You should see the IIS default web page, confirming connectivity.

3. Alternatively, use PowerShell or Command Prompt on the VM to run the following command, replacing `<ILB_Private_IP>` with your load balancer's private IP address:

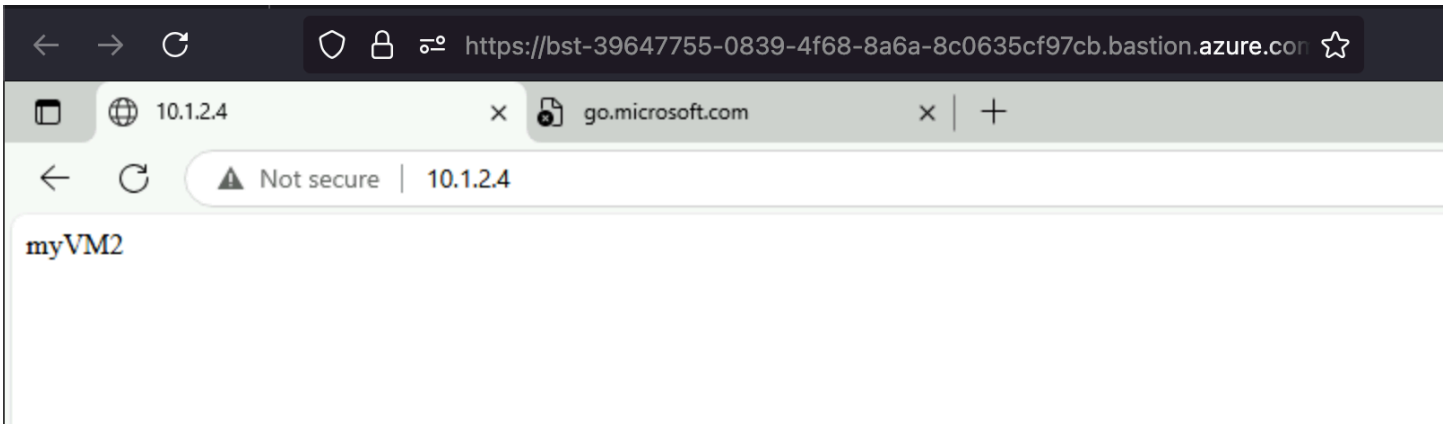
```powershell
Invoke-WebRequest -Uri http://<ILB_Private_IP>
```

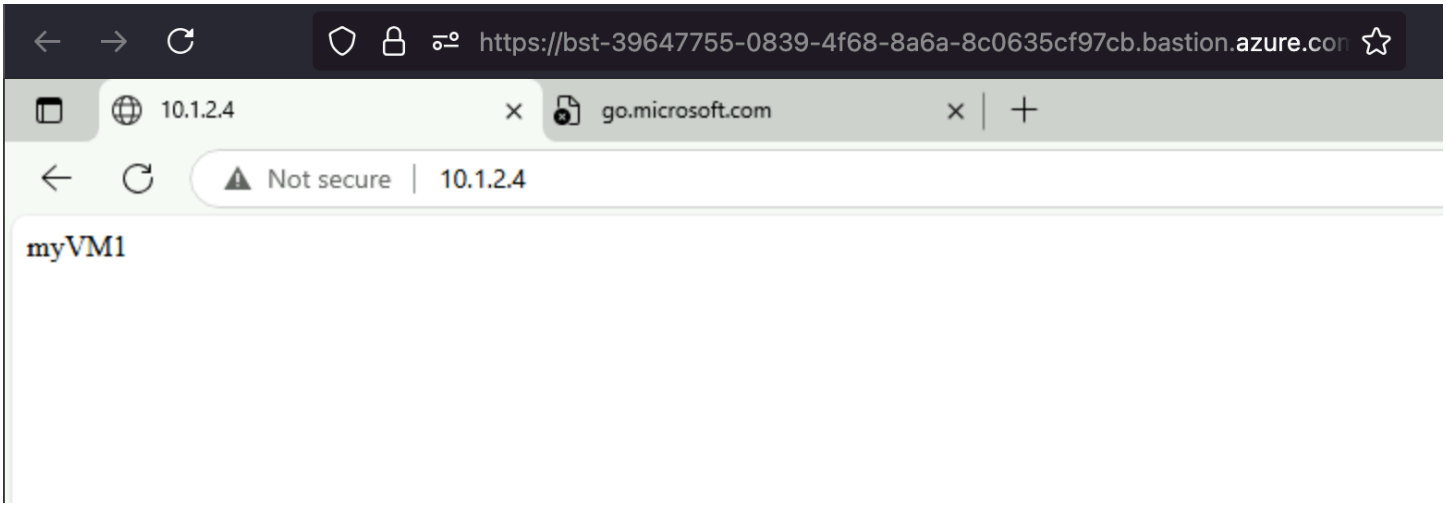
Or, if using a Linux VM, use `curl`:

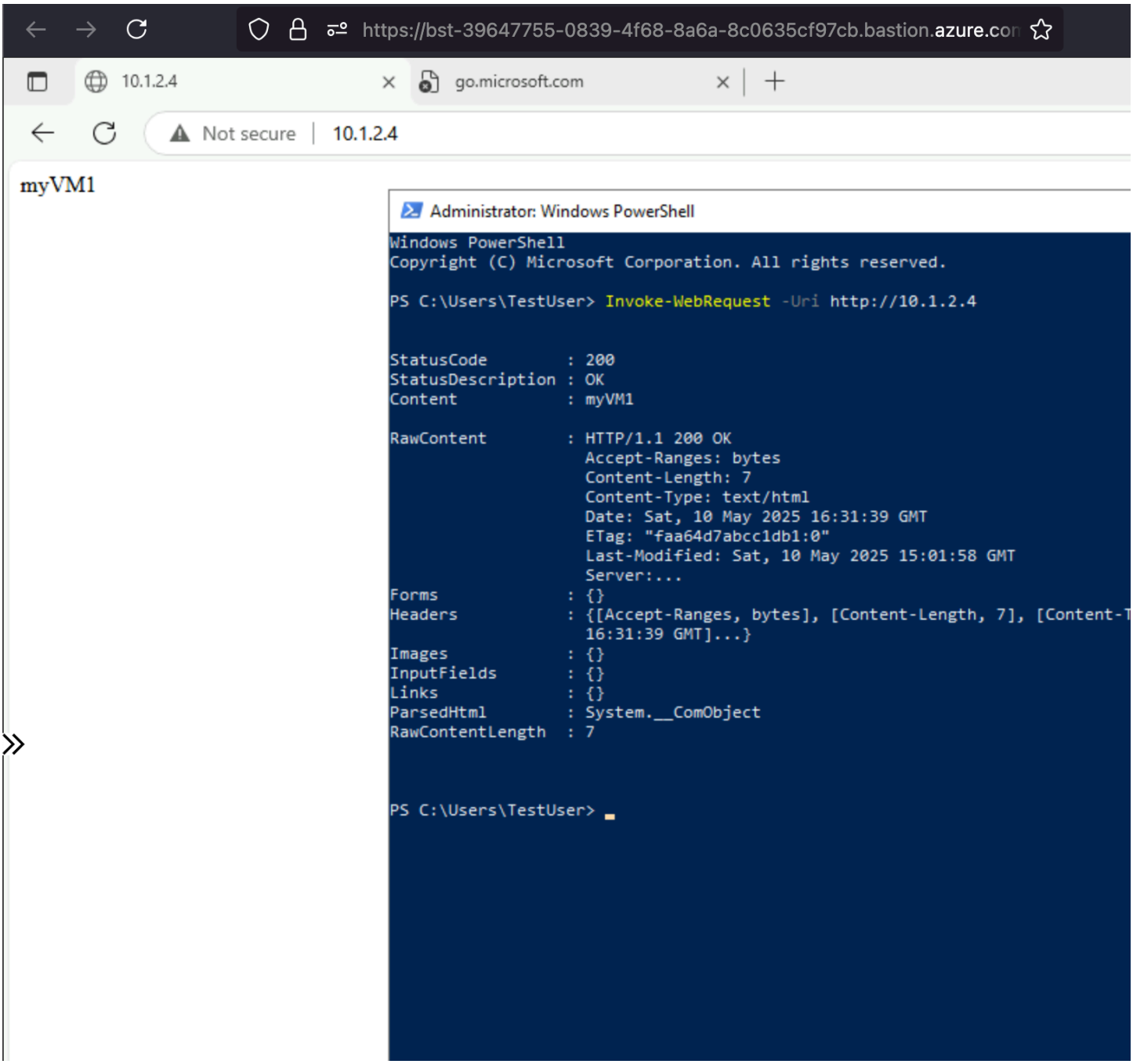
```bash
curl http://<ILB_Private_IP>
```

4. Refresh the browser or repeat the command multiple times to observe responses from different backend VMs, demonstrating load balancing.

**Note:** Ensure that network security group (NSG) rules allow HTTP traffic between VMs in the virtual network.



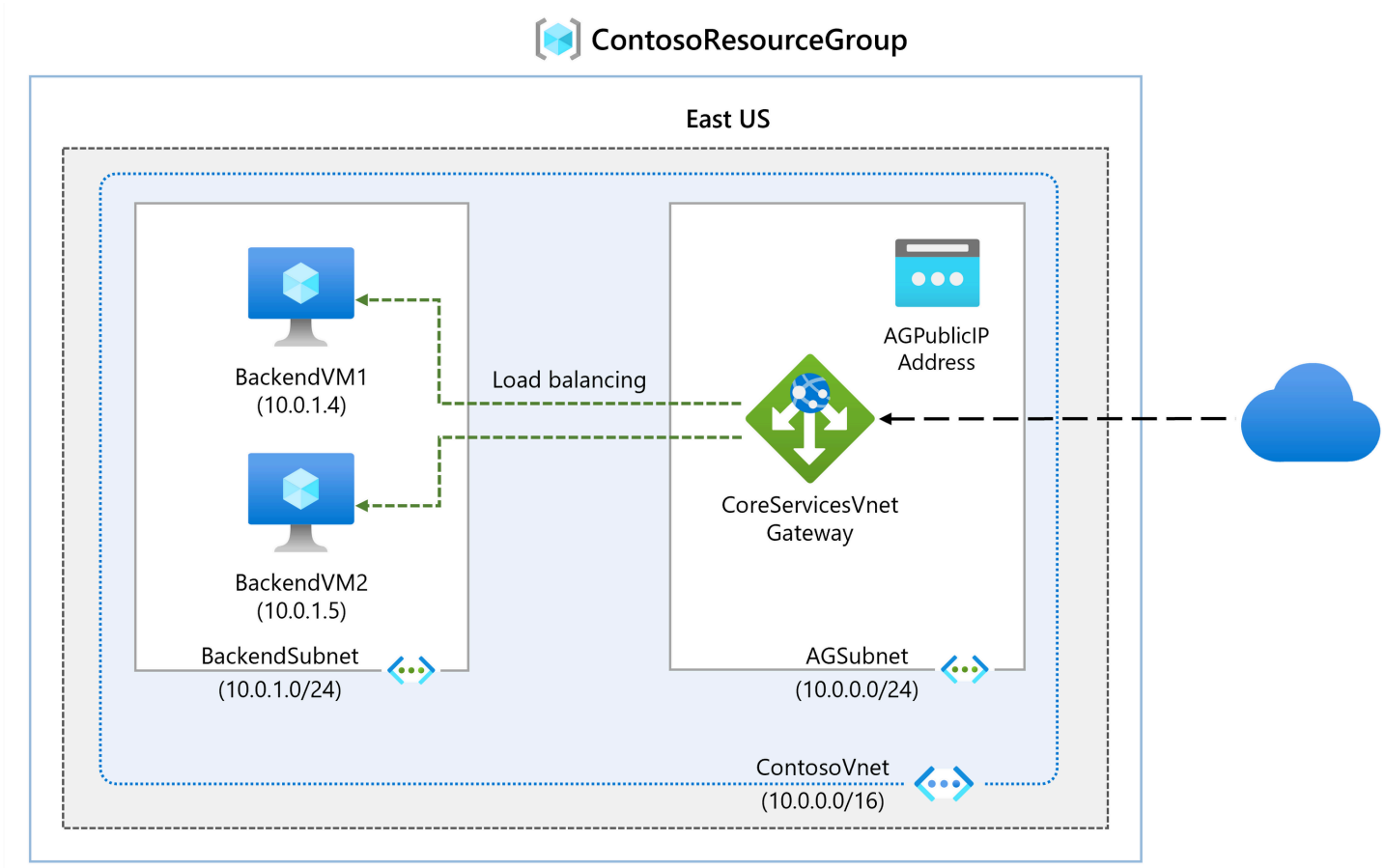






## PART 2

**Azure Application Gateway** provides Application Delivery Controller (ADC) as a service, offering various Layer 7 load-balancing capabilities. Use it to optimize web farm productivity by offloading CPU-intensive TLS/SSL termination to the Application Gateway - Also, you don't need to install certificates and configure TLS/SSL on your servers. If you need end-to-end encryption, Application Gateway can decrypt the traffic on the gateway by using your private key, then re-encrypt again with the public key of the service running in the back-end pool. Application Gateway works within a region rather than globally. Azure Application Gateway includes the following features: Support for the HTTP, HTTPS, HTTP/2, and WebSocket protocols, A web application firewall (WAF) to protect against web application vulnerabilities, End-to-end request encryption, Autoscaling to dynamically adjust capacity as your web traffic load change, Connection draining allowing graceful removal of backend pool members during planned service updates, Session stickiness to ensure client requests in the same session are routed to the same backend server(Session stickiness is especially important with e-commerce applications where you don't want a transaction to be disrupted because the load balancer bounces it around between back-end servers.). Application Gateway uses a

round-robin process to load balance requests to the servers in each back-end pool. Load-balancing works with the OSI Layer 7 routing implemented by Application Gateway routing, which means that it load balances requests based on the routing parameters (host names and paths) used by the Application Gateway rules.




**+ How does the Azure Application Gateway route requests?**

The Azure Application Gateway routes requests based on Layer 7 (application layer) information, specifically HTTP/HTTPS attributes. This sophisticated routing is achieved through several key mechanisms:

💥 **Listeners**: The Application Gateway uses listeners to check for incoming requests on specific ports and protocols (e.g., HTTP on port 80, HTTPS on port 443).

* Basic Listeners: Process requests on a single domain.

* Multi-site Listeners: Allow the Application Gateway to host multiple web applications on the same Application Gateway instance, routing requests based on the host header (e.g., www.contoso.com goes to one backend pool, www.fabrikam.com to another).

💥 **Backend Pools:** A collection of backend targets (e.g., Azure VMs, Virtual Machine Scale Sets, Azure App Service, on-premises servers, or even other public IP addresses).8

💥 **HTTP Settings:** Define parameters for how the Application Gateway connects to the backend servers, including port, protocol (HTTP/HTTPS), cookie-based affinity (session persistence), request timeout, and health probes.

💥 **Health Probes**: The Application Gateway continuously monitors the health of backend servers using configurable health probes. If a backend server fails a health probe, the Application Gateway stops sending requests to it until it becomes healthy again.

💥 **Request Routing rules ->** A request routing rule is a key component of an application gateway because it determines how to route traffic on the listener. The rule binds the listener, the backend server pool, and the backend HTTP settings. A rule specifies how to interpret the hostname and path elements in the URL of a request and direct the request to the appropriate back-end pool. When a listener accepts a request, the request routing rule forwards the request to the backend or redirects it elsewhere. If the request is forwarded to the backend, the request routing rule defines which backend server pool to forward it to. One of the most important gateway configuration settings is the routing rules.

The Azure Application Gateway has two primary methods of routing client requests: PATH-BASED and MULTI-SITES.

1. **Path-based routing / (URL based routing)** sends requests with different URL paths to different pools of back-end servers.
2. **Multiple site routing** configures more than one web application on the same Application Gateway instance. In a multiple site configuration, you register multiple DNS names (CNAMEs) for the IP address of the application gateway, specifying the name of each site. Application Gateway uses separate listeners to wait for requests for each site. Each listener passes the request to a different rule, which can route the requests to servers in a different back-end pool. Along with path-based routing and multiple site hosting, there are a few other capabilities when routing with Application Gateway.

**I. Redirection** can be used to another site, or from HTTP to HTTPS. For example, redirecting HTTP requests to a secure HTTPS shopping site.

**II. Rewrite HTTP headers**, HTTP headers allow the client and server to pass additional information with the request or the response.

**III. Custom error pages.** Application Gateway allows you to create custom error pages instead of displaying default error pages. You can use your own branding and layout using a custom error page.

**IV. A Routing Rule** also has an associated set of **HTTP settings**. These HTTP settings indicate whether (and how) traffic is encrypted between Application Gateway and the back-end servers. Other configuration information includes **V. Protocol; VI. Session stickiness; VII. Connection draining; VIII. Request timeout period**

In summary, the Application Gateway acts as a reverse proxy, inspecting the HTTP/HTTPS headers and URL paths to make intelligent routing decisions, distribute load, and apply security policies before forwarding requests to the appropriate backend servers.

The application gateway directs application web traffic to specific resources in a backend pool. You assign listeners to ports, create rules, and add resources to a backend pool. For the sake of simplicity, this article uses a simple setup with a public front-end IP, a basic listener to host a single site on the application gateway, a basic request routing rule, and two virtual machines in the backend pool. For Azure to communicate between the resources that you create, it needs a virtual network. You can either create a new virtual network or use an existing one. In this example, you'll create a new virtual network while you create the application gateway. Application Gateway instances are created in separate subnets. You create two subnets in this example: one for the application gateway, and another for the backend servers.

In this exercise, you will:

*+ Task 1: Create an application gateway*

*+ Task 2: Create virtual machines*

*+ Task 3: Add backend servers to backend pool*

*+ Task 4: Test the application gateway*

*+ Task 4: Test the application gateway*

**## Task 1: Create an application gateway**

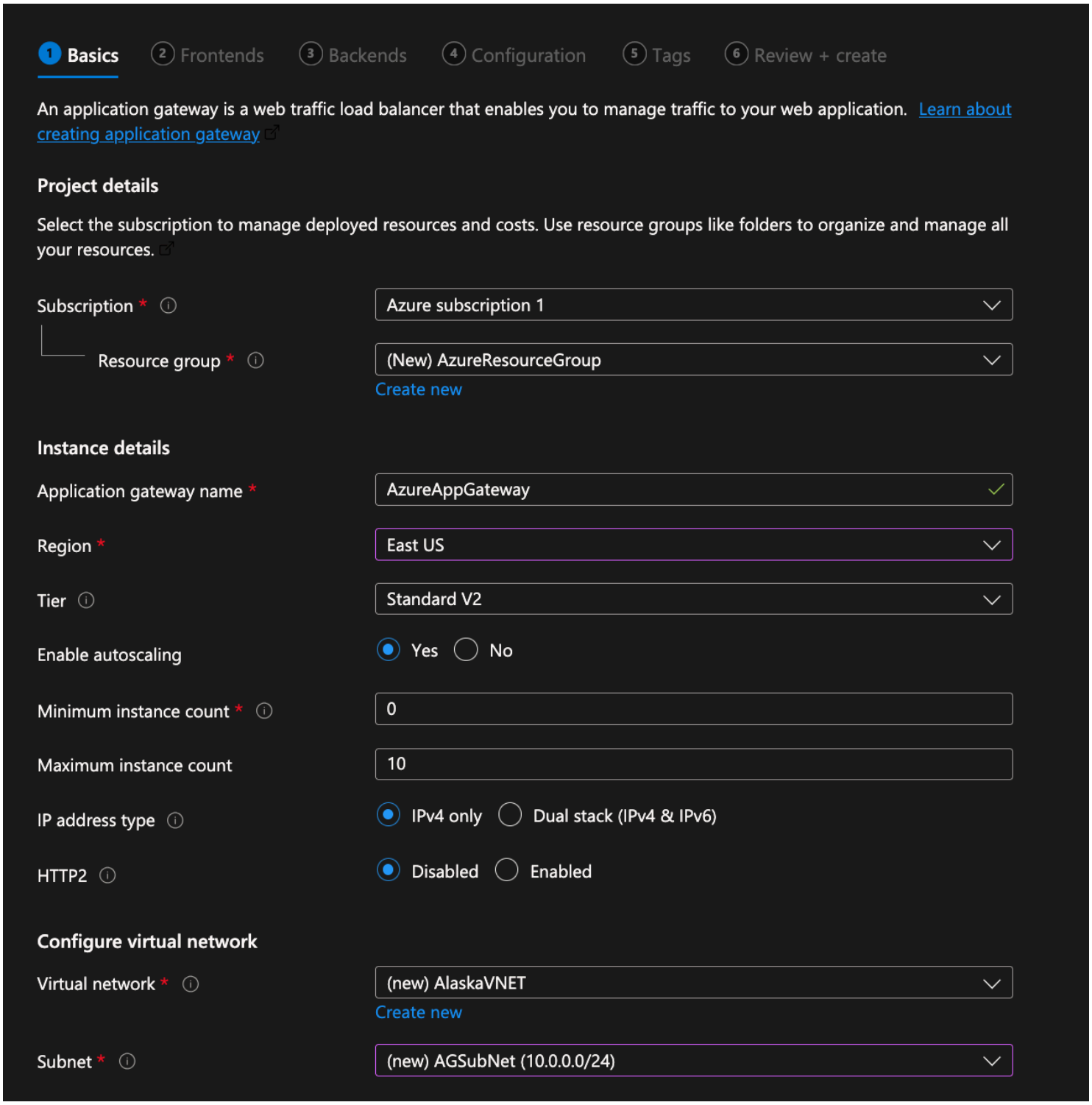
On any Azure Portal page, in **Search resources, services and docs (G+/)**, enter application gateway, and then select **Application gateways** from the results. On the Application gateways page, select **+ Create**.

On the Create application gateway **Basics** tab, enter, or select the following information:

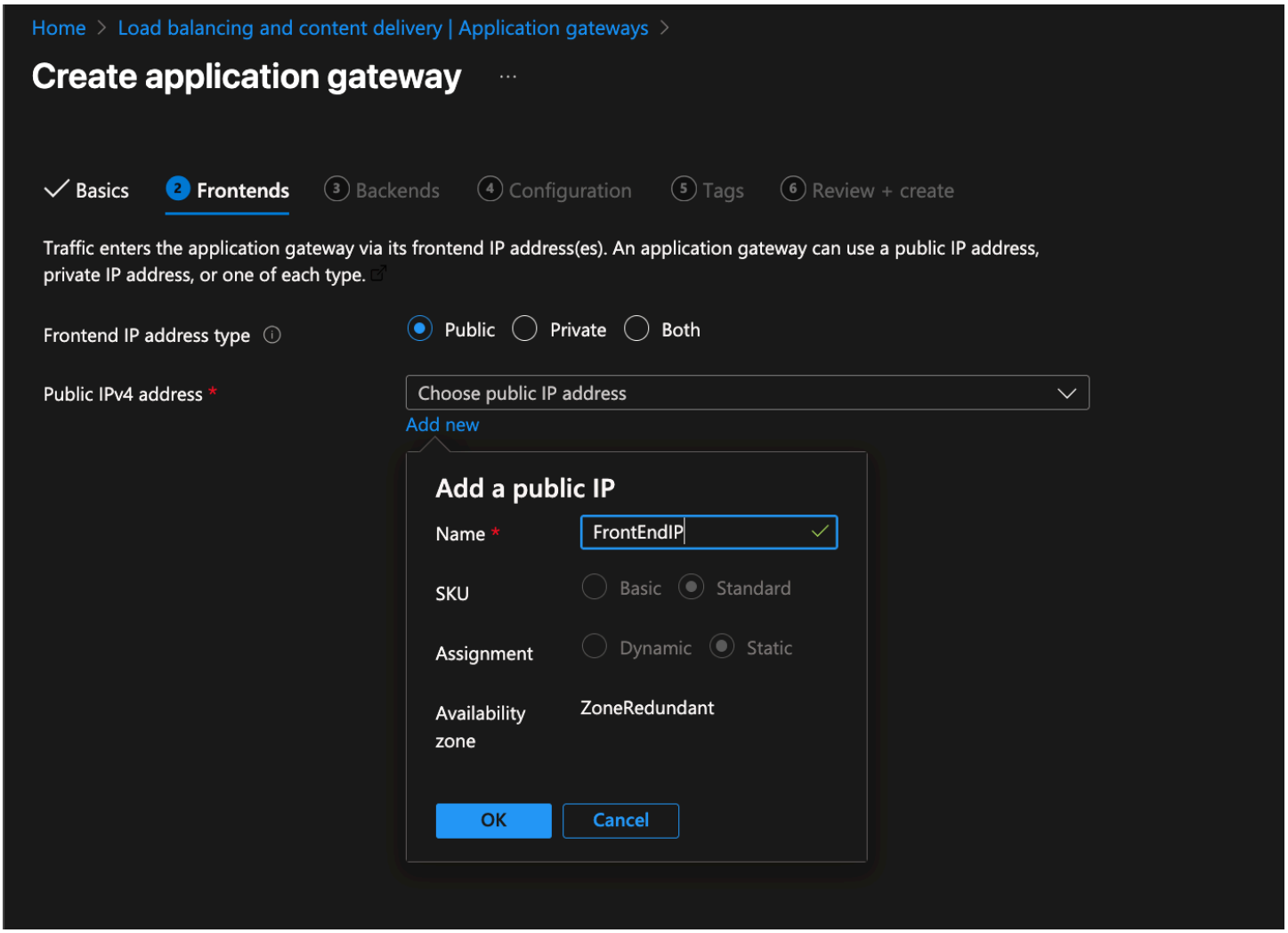
| **Setting** | **Value**
| Subscription | Select your subscription.
| Resource group | Select Create new AzureResourceGroup
| Application Gateway | AzureAppGateway
| Region | Select **East US**
| Virtual Network | Select **Create new**

Next We are going to Create virtual network and add Subnets, enter, or select the following information: It is important to note the architectural decision to allocate a dedicated subnet, **AGSubnet**, for the Application Gateway. This mirrors the assignment of the Layer 4 Load Balancer's FrontEnd-IP to the **FrontEnd Subnet**, with both subnets collectively defining our Demilitarized Zone (DMZ).

| **Setting** | **Value**
| Name | AlaskaVNet
| **ADDRESS SPACE** |
| Address range | 10.0.0.0/16
| **SUBNETS** | |
| Subnet name | Change **default** to **AGSubnet** |
| Address range | 10.0.0.0/24 |

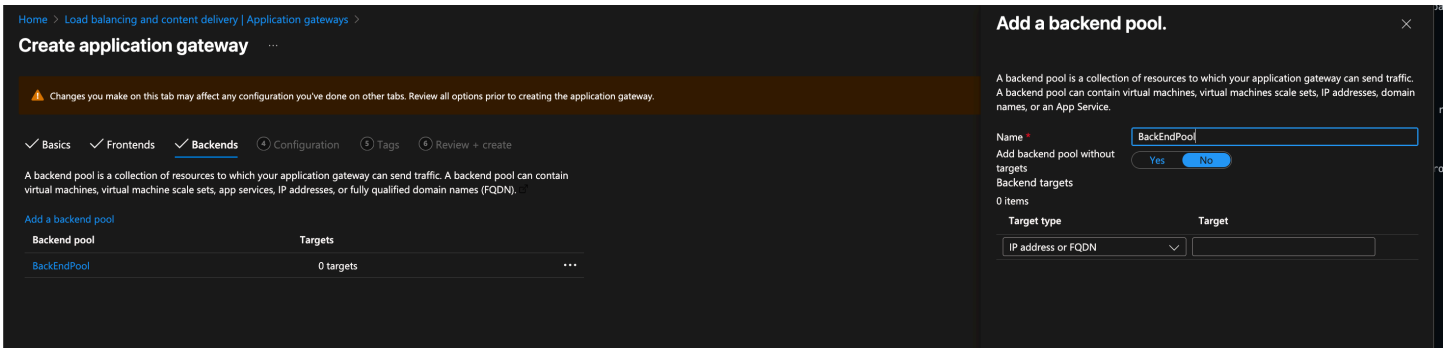



Select **OK** to return to the Create application gateway Basics tab. Accept the default values for the other settings and then select **Next: Frontends**. On the **Frontends** tab, verify **Frontend IP** address type** is set to **Public**. Select **Add new** for the **Public IP address** and enter **FRONTENDIP** for the public IP address name, and then select **OK**.




Select **Next: Backends**. On the **Backends** tab, select **Add a backend pool**. On the **Add a backend pool** window that opens, enter the following values to create an empty backend pool:

| **Setting** | **Value** |
| --- | --- |
| Name | BackendPool |
| Add backend pool without targets | Yes |




On the **Add a backend pool** window, select **Add** to save the backend pool configuration and return to the **Backends** tab. On the **Backends** tab, select **Next: Configuration**. On the **Configuration** tab, you'll connect the frontend and backend pool you created using a routing rule. On the **Routing rules** column, select **Add a routing rule**. On the **Rule name** box, enter **RoutingRule**. For **Priority** enter **100**.

On the **Listener** tab, enter or select the following information:

Accept the default values for the other settings on the **Listener** tab.

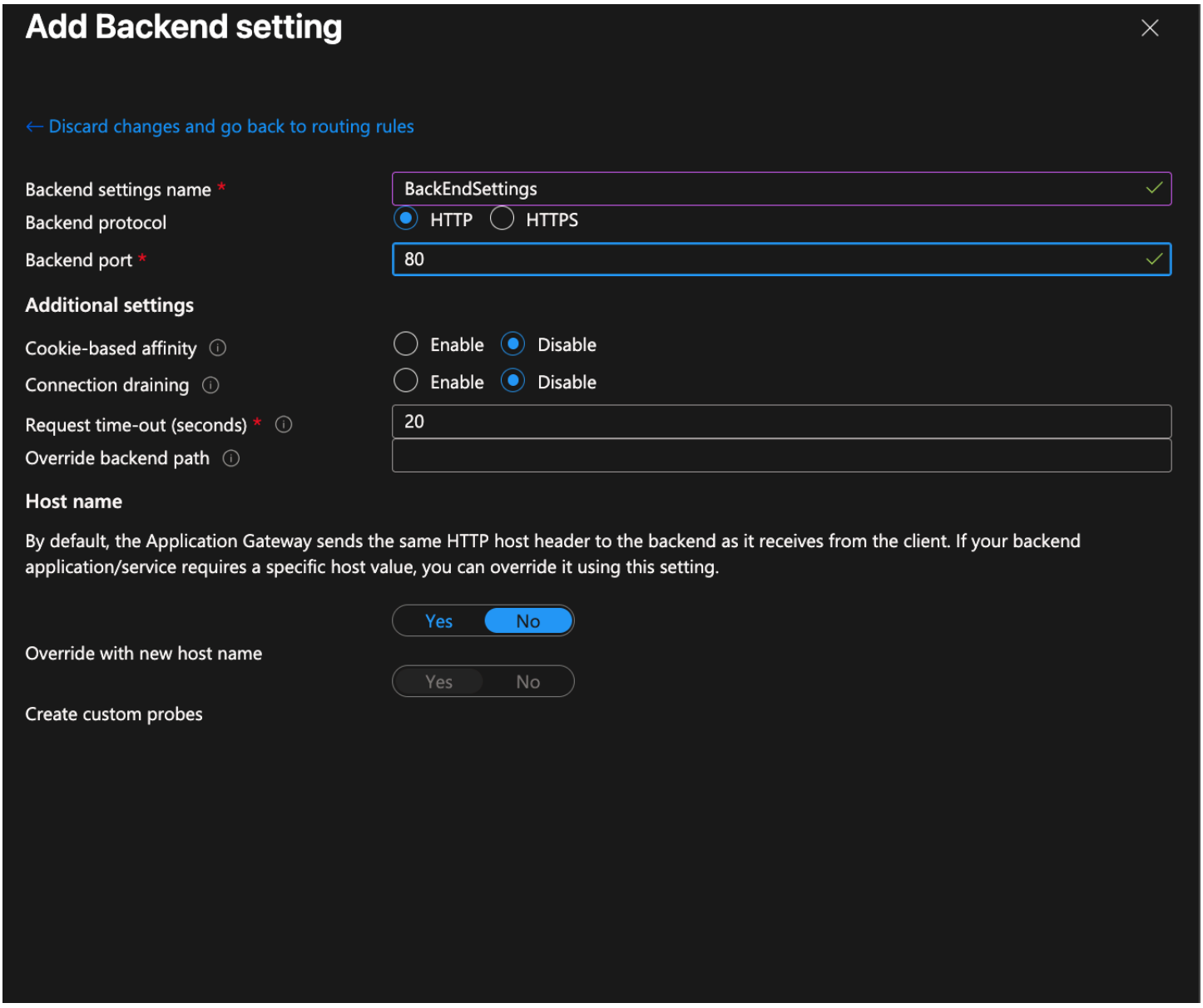
| **Setting** | **Value** |
| Listener name | Listener |
| Frontend IP | Select **Public IPv4** |

Select the **Backend targets** tab to configure the rest of the routing rule. On the **Backend targets** tab, enter or select the following information

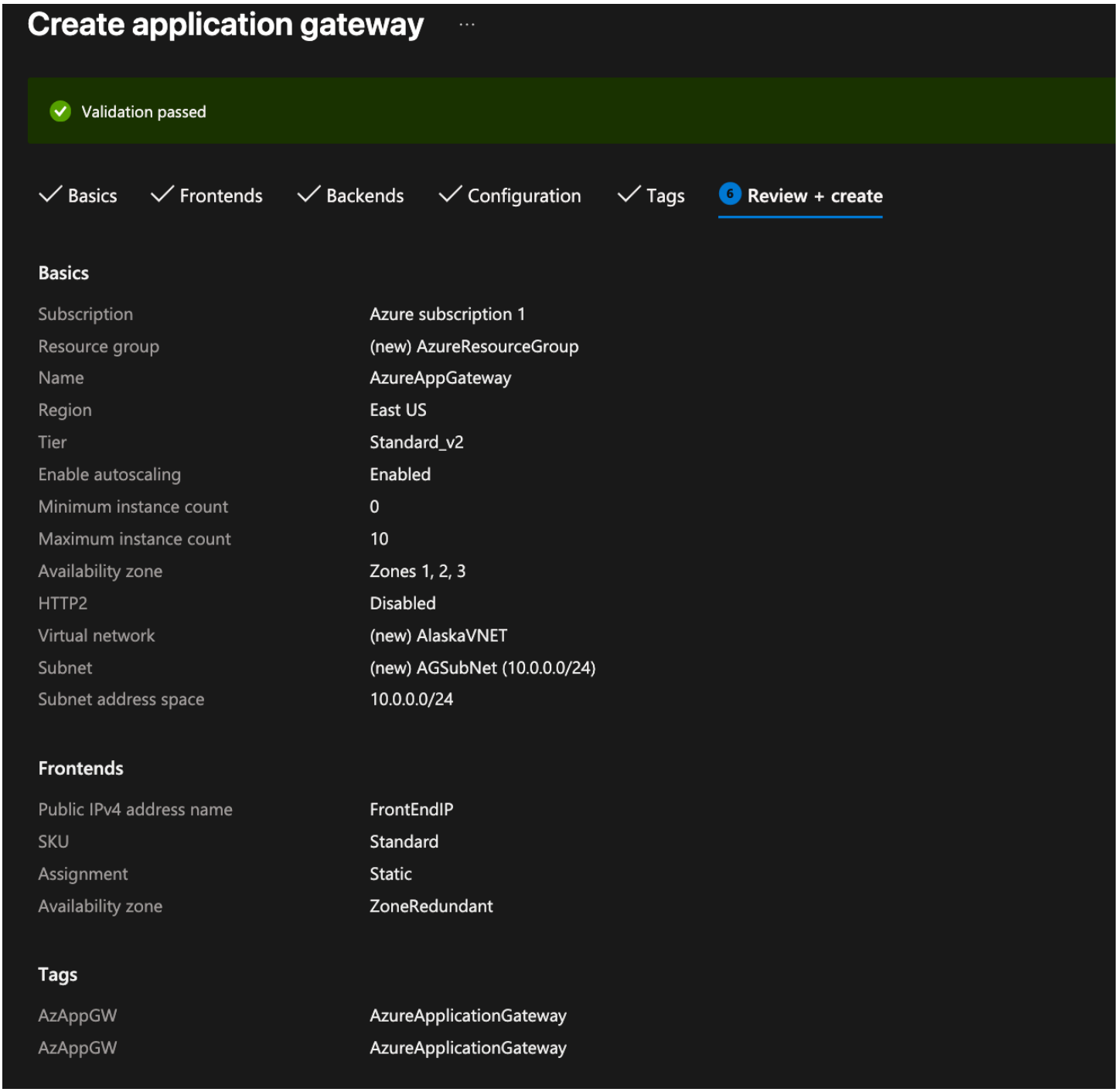
| **Setting** | **Value** |
| Target type | Backend pool |
| Backend Settings | **Add new** |

In **Add a Backend Setting**, enter or select the following information:

| **Setting** | **Value** |
| Backend setting name | HTTPSetting |
| Backend port | 80 |




Next Select **Add** to save the routing rule and return to the **Configuration** tab. Select **Next: Tags** and then **Next: Review + create**. Review the settings on the **Review + create** tab. Select **Create** to create the virtual network, the public IP address, and the application gateway. It may take several minutes for Azure to create the application gateway. Wait until the deployment finishes successfully.



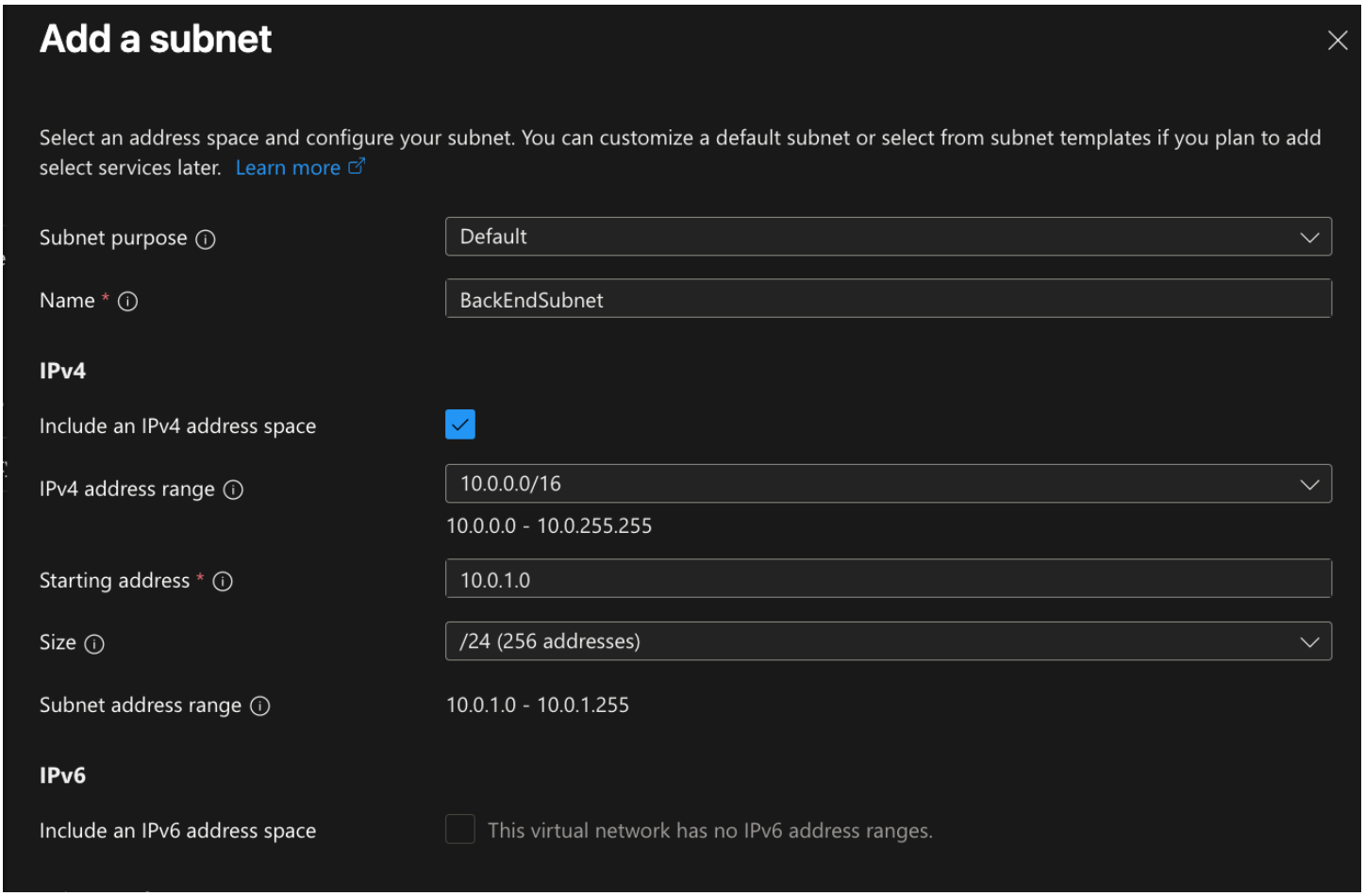

**### Add a subnet for a backend servers**

Search for and select the **AlaskaVNet**. Verify the **AGSubnet** was created. To create the **BackendSubnet**, select **Settings** and then **Subnets**. Be sure to **Add** the subnet when finished.

| **Setting** | **Value**

| Subnet name | BackendSubnet

| Address range | 10.0.1.0/24




**## Task 2: Create virtual machines**

In the Azure portal, select the Cloud Shell icon (top right). If necessary, configure the shell.

+ Select **PowerShell**.

+ Select **No Storage Account required** and your **Subscription**, then select **Apply**.

+ Wait for the terminal to create and a prompt to be displayed.

On the toolbar of the Cloud Shell pane, select **Manage files** and then **Upload**. Upload the following files: **backend.json**, **backend.parameters.json**, and **install-iis.ps1**. The files are available for download from the repository, **AZ-700-Designing-and-Implementing-Microsoft-Azure-Networking-Solutions-master\Allfiles\Exercises\M05** folder.

Deploy the following ARM templates to create the VMs needed for this exercise:

**Note**: You will be prompted to provide an Admin password.

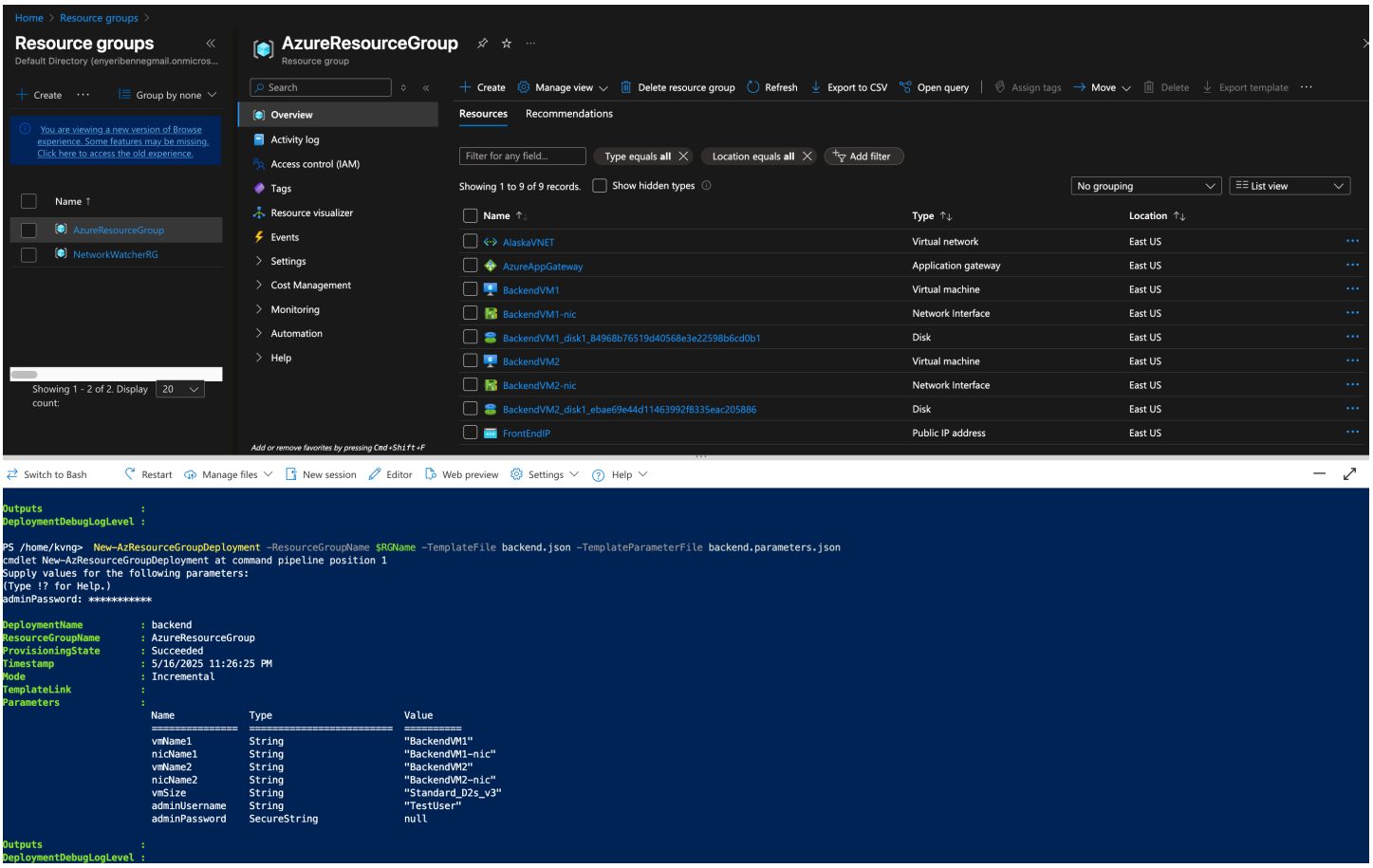
```powershell

$RGName = "AzureResourceGroup"

New-AzResourceGroupDeployment -ResourceGroupName $RGName -TemplateFile backend.json -TemplateParameterFile backend.parameters.json

```

>**Note**: Take time to review the **backend.json** file. There are two virtual machines being deployed. This will take a few minute, The command should complete successfully and list **BackendVM1** and **BackendVM2**.







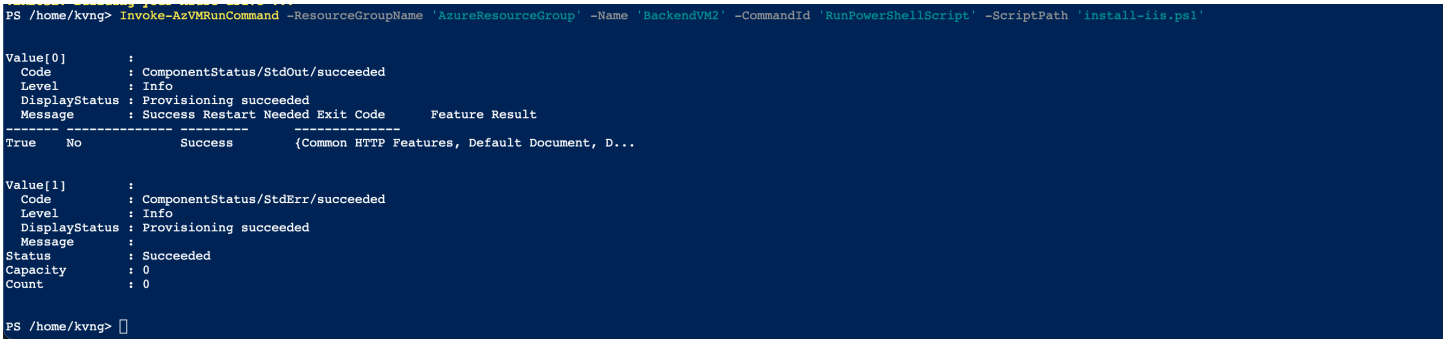
### Install IIS on each virtual machine

Each backend server needs IIS installed. This is similar to Installing an Apache Web Server on Linux VMs. Continue at the PowerShell prompt and use the provided script to install IIS on **BackendVM1**. Run the command again, this time for **BackendVM2**.

```powershell

Invoke-AzVMRunCommand -ResourceGroupName 'AzureResourceGroup' -Name 'BackendVM1' -CommandId 'RunPowerShellScript' -ScriptPath 'install-iis.ps1'
```

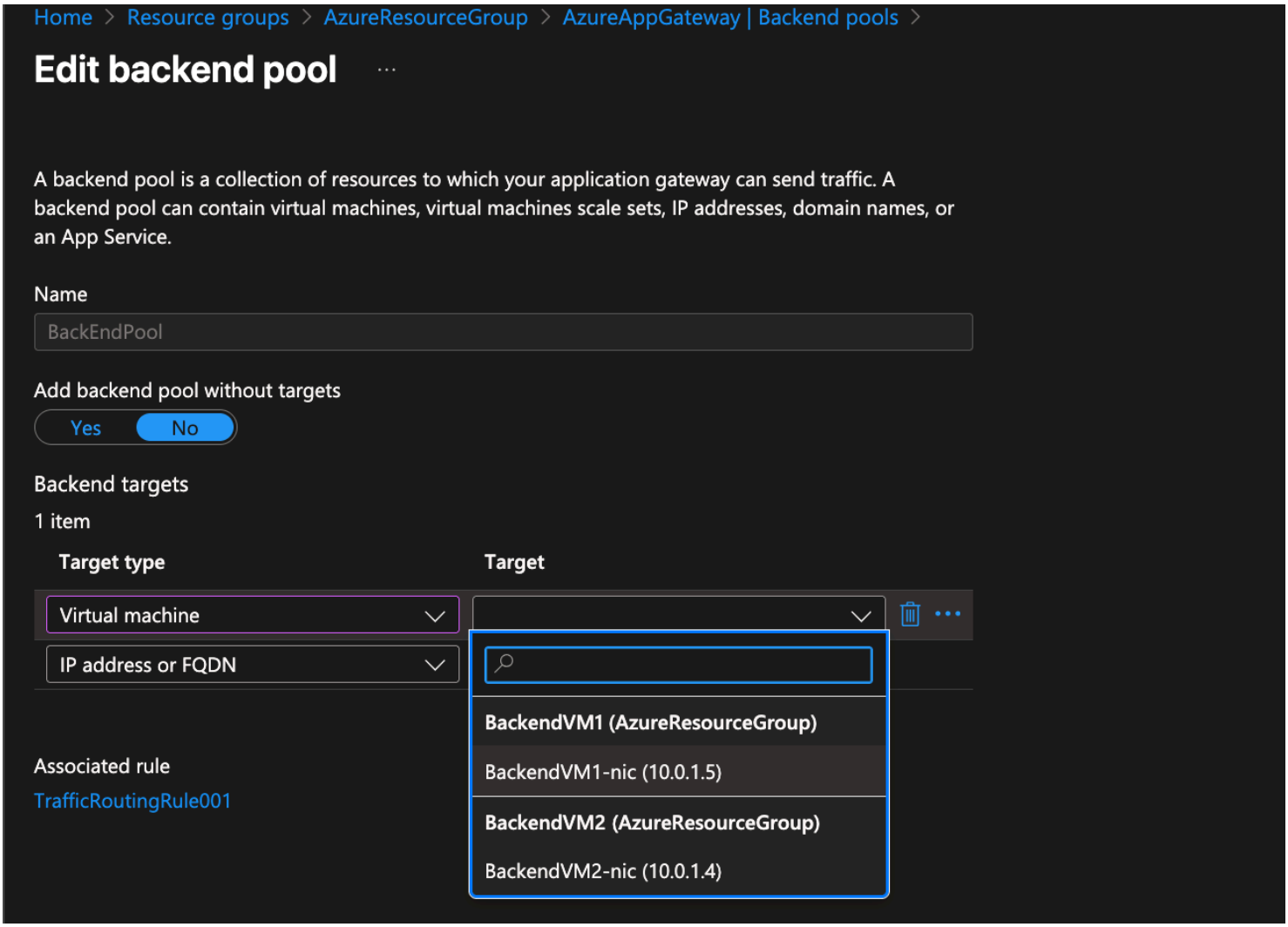
**Note**: While you wait review the PowerShell script. Notice that the IIS home page is being customized to provide the virtual machine name.




## Task 3: Add backend servers to backend pool

On the Azure portal menu, select **All resources** or search for and select All resources. Then select **AzureAppGateway**. Under **Settings**, select **Backend pools**.

On the Edit backend pool page, under **Backend targets**, in **Target type**, select **Virtual machine**.  Under **Target**, select **BackendVM1-nic.** On **Target type**, select **Virtual machine**. Under **Target**, select **BackendVM2-nic.**



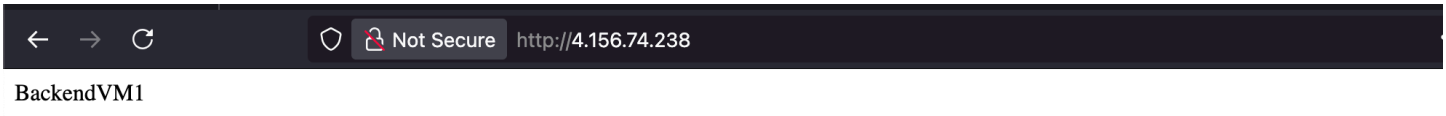

Select **Save** and wait for the targets to be added.  Check to ensure the backend servers are healthy. Select **Monitoring** and then **Backend Health**. Both targets should be healthy.

**## Task 4: Test the application gateway**

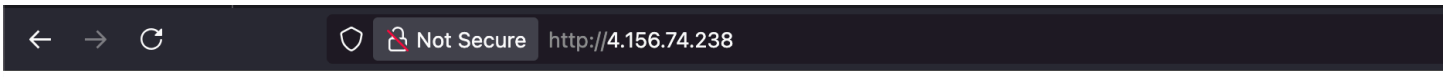
Although IIS isn't required to create the application gateway, you installed it in this exercise to verify if Azure successfully created the application gateway.

### Use IIS to test the application gateway

Find the public IP address for the application gateway on its **Overview** page. Copy the public IP address, and then paste it into the address bar of your browser to browse that IP address. Check the response.




Go ahead and refresh your browser several times. What you should see is the Application Gateway intelligently routing your traffic, with connections landing on both **BackendVM1** and **BackendVM2**. The consistent response from the **Application Gateway's Front-End IP** paired with the varied backend connections means your setup is working perfectly!



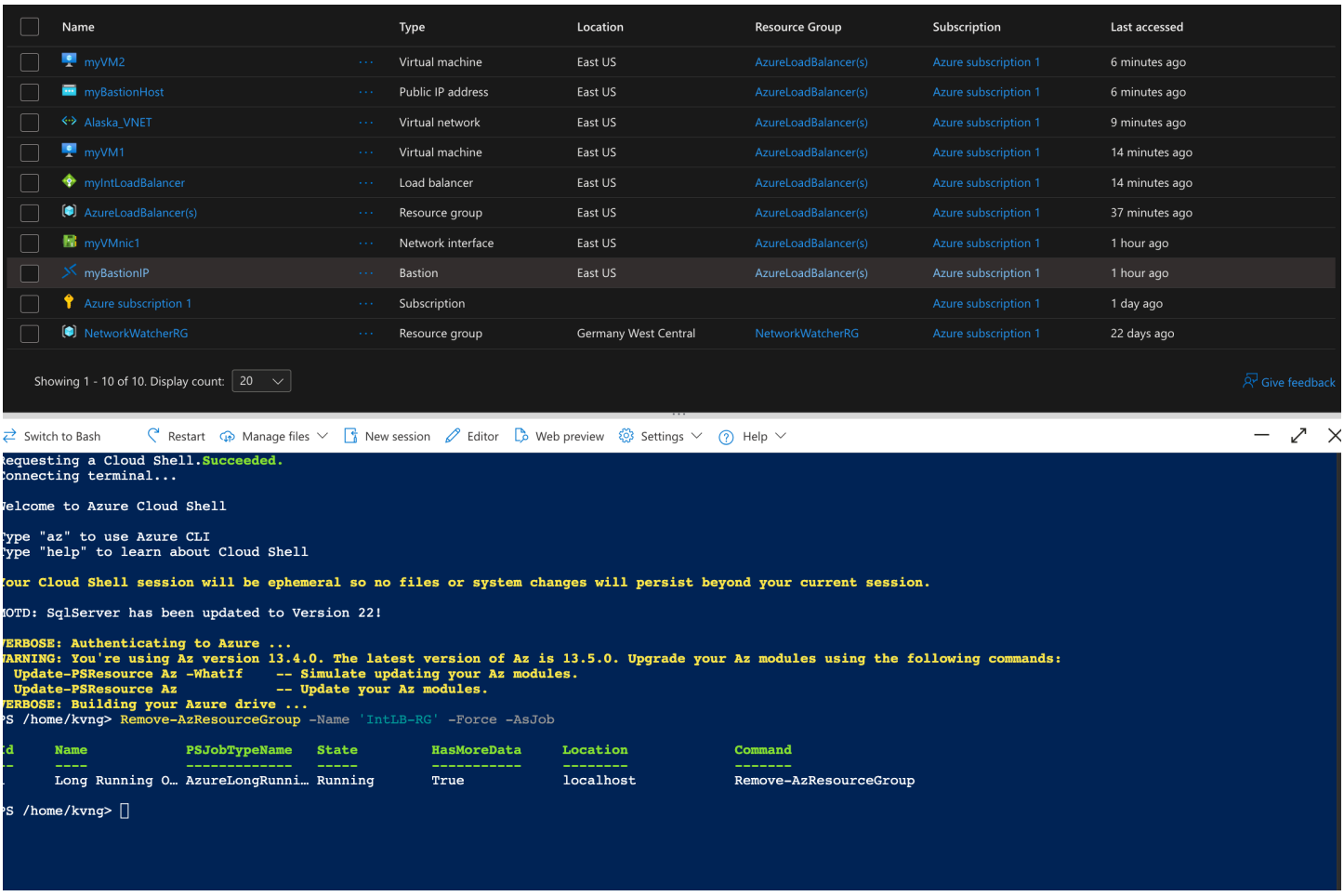

BackendVM2

## Clean up resources

>**Note**: Remember to remove any newly created Azure resources that you no longer use. Removing unused resources ensures you will not see unexpected charges.

On the Azure portal, open the **PowerShell** session within the **Cloud Shell** pane. Delete all resource groups you created throughout the labs of this module by running the following command:

powershell → Remove-AzResourceGroup -Name 'AzureResourceGroup' -Force -AsJob







**Note**: The command executes asynchronously (as determined by the -AsJob parameter), so while you will be able to run another PowerShell command immediately afterwards within the same PowerShell session, it will take a few minutes before the resource groups are actually removed.

## Further Discussion to Extend Our learning

### + What security features does Azure Application Gateway include?

The Azure Application Gateway is rich in security features, making it a robust component for protecting web applications. Its primary security features include:

### 1. Web Application Firewall (WAF):

* OWASP Core Rule Set (CRS): The WAF SKU integrates with the OWASP (Open Web Application Security Project) Core Rule Set to protect web applications from common web vulnerabilities such as SQL injection, cross-site scripting (XSS), command injection, HTTP protocol violations, and more.

* Custom Rules: Allows you to create your own WAF rules to block or allow traffic based on specific conditions, such as IP addresses, HTTP headers, or request attributes.

* Geo-filtering: Restrict access to your web applications based on the geographic location of the client IP address.

* Rate Limiting: Protects against DDoS attacks by limiting the number of requests from a specific IP address over a period.

**2. SSL/TLS Termination (End-to-End SSL):**

* The Application Gateway can decrypt SSL/TLS traffic at the gateway, allowing it to inspect traffic for WAF policies and then re-encrypt it before sending it to the backend servers. This offloads the encryption/decryption burden from the backend servers and allows for deeper inspection.

* Supports various SSL/TLS versions and cipher suites, allowing for strong encryption.

**3. End-to-End Encryption:**

* While it can terminate SSL, it can also re-encrypt the traffic to the backend, ensuring encryption throughout the entire path from client to server. This is crucial for compliance requirements.

**4. IP Configuration & Network Security Groups (NSGs):**

* The Application Gateway is deployed within a virtual network subnet, allowing you to apply Network Security Groups (NSGs) to control inbound and outbound traffic to and from the Application Gateway subnet, further enhancing network security.

**5. Role-Based Access Control (RBAC):**

* Integrates with Azure RBAC, allowing you to define granular permissions for who can manage and configure the Application Gateway.

**6. Custom Error Pages:**

* Allows you to present custom error pages to users, preventing the display of potentially sensitive server error messages.

These features collectively provide comprehensive protection against a wide range of application-layer attacks and ensure secure communication for your web applications.

**+ How are the Azure public and private load balancers different?** Azure public and private load balancers differ primarily in how they expose services and the types of traffic they handle. Azure offers both public and private load balancers. Public Load Balancers are ideal for internet-facing applications, outbound connections, and web applications. Private load balancers are better for internal-only traffic applications, backend services, and hybrid scenarios or within your Azure Environment.

1. A **public load balancer** provides a frontend with a public IP address, making applications or services accessible from the internet. It distributes incoming internet traffic to backend resources, such as virtual machines, within an Azure virtual network. Public load balancers are commonly used for scenarios where external clients need to access web

applications, APIs, or other services hosted in Azure. For example, a company hosting a public-facing website on Azure would use a public load balancer to distribute HTTP/HTTPS requests from users around the world to multiple web servers for high availability and scalability. Azure Load Balancer distributes inbound flows from the load balancer's frontend to backend pool instances. These flows are distributed according to configured load-balancing rules and health probes. The backend pool instances can be Azure virtual machines (VMs) or virtual machine scale sets.

2. A **private load balancer,** on the other hand, uses an internal (private) IP address as its frontend. It is only accessible within the Azure virtual network or through connected networks (such as via VPN or ExpressRoute). Private load balancers are ideal for distributing traffic between internal services that should not be exposed to the internet. For instance, an organization might use a private load balancer to balance requests between application servers and database servers within a secure, isolated environment, ensuring that only internal systems can communicate with these resources.

**+ When to use Azure Load Balancer:**

**Non-HTTP/HTTPS applications:** When you need to balance TCP or UDP traffic for applications like gaming servers, VPN termination, or custom protocols.

* Example: Distributing incoming RDP (TCP 3389) traffic across multiple jump box VMs.

* Example: Load balancing traffic for a custom financial application that uses a proprietary TCP protocol.

**Internal Load Balancing:** When you need to distribute traffic within your virtual network, for example, between different tiers of a multi-tier application.

* Example: Balancing traffic from web servers (front-end) to application servers (middle-tier) in a private network.

**High-performance Layer 4 load balancing:** For scenarios where minimal latency and high throughput at the transport layer are critical, and application-layer features are not required.

* Simple Load Distribution: When you just need basic load distribution without complex routing or WAF capabilities.

* Example: Providing high availability for a cluster of Redis cache servers.

**+ When to use Azure Application Gateway:**

**Web Applications (HTTP/HTTPS):** Any public-facing or internal web application that uses HTTP or HTTPS.

* Example: Hosting a WordPress website across multiple backend VMs, offloading SSL, and protecting it with a WAF.

* Example: Managing traffic for a suite of RESTful APIs, routing requests to different backend microservices based on the URL path (/api/users vs. /api/products).

**SSL/TLS Offloading:** When you want to offload the CPU-intensive task of SSL decryption from your backend web servers.

* Example: An e-commerce site where all client-side traffic is HTTPS, but you want your backend servers to only handle HTTP traffic.

**Web Application Firewall (WAF) requirements:** When you need to protect your web applications from common web-based attacks.

* Example: A critical business application that needs protection against SQL injection, XSS, and other OWASP Top 10 vulnerabilities.

**URL-based Routing:** When you need to direct requests to different backend pools based on the URL path or host header.

* Example: Routing www.mycompany.com/blog to one set of servers and www.mycompany.com/store to another.

**Multi-site Hosting:** When you want to host multiple web applications (with different domain names) on the same Application Gateway instance.

* Example: Hosting www.siteA.com and www.siteB.com on the same Application Gateway, each directing to a distinct backend pool.

**Session Affinity:** When you need to ensure that requests from a particular user always go to the same backend server (cookie-based).

* Example: A shopping cart application where a user's session state is tied to a specific backend server.

In essence, Azure Load Balancer is your go-to for network-level (L4) load balancing for any TCP/UDP traffic, while Azure Application Gateway is specialized for intelligent application-level (L7) traffic management and security for web applications. Often, they can be used together in a complex architecture, with a Load Balancer handling non-web traffic or acting as a first layer, and an Application Gateway handling specific web traffic.

**Here's a comparison of Azure Application Gateway and Azure Load Balancer, highlighting their key differences and use cases:**

| Feature/Aspect | Azure Load Balancer | Azure Application Gateway |
|---|---|---|
| **Layer of Operation** | Layer 4 (Transport Layer) - TCP/UDP | Layer 7 (Application Layer) - HTTP/HTTPS |
| **Functionality** | Distributes network traffic based on IP address and port. | Routes and load-balances HTTP/HTTPS traffic based on URL, host headers, cookies, and other L7 attributes. |
| **Health Probes** | Basic health probes (TCP, HTTP, HTTPS) | Advanced health probes (custom probes, host headers, status codes) |
| **SSL/TLS** | No direct SSL/TLS termination. Can forward encrypted traffic. | **SSL/TLS Termination (offloading)**, end-to-end SSL encryption, multi-domain SSL. |
| **Security** | Basic network security (NSGs on subnet) | **Web Application Firewall (WAF)** for L7 protection, IP filtering. |
| **Sticky Sessions** | No built-in session affinity beyond source IP hashing (for TCP). | **Cookie-based session affinity** |
| **URL-based Routing** | No | Yes (path-based routing, multi-site routing) |
| **Cost** | Generally lower | Generally higher (due to advanced features) |
| **Use Cases** | Non-web protocols (FTP, SSH, RDP, custom apps), internal traffic, high-performance L4 load balancing. | Web applications (port 80/443), microservices with HTTP APIs, applications requiring WAF, SSL offloading, URL-based routing, or multi-site hosting. |

**## Key takeaways && Final Thoughts**

- **Internal Load Balancers Enhance Internal Application Availability:** This lab demonstrates how to distribute non-web traffic across multiple backend servers *within* your Azure Virtual Network, improving resilience and performance for internal applications.
- **Virtual Networks are Foundational:** The first step highlights the necessity of a properly configured virtual network as the isolated environment for your load balancer and backend resources.
- **Backend Pools are the Target Group:** You learned how to define a backend pool, which is the set of virtual machines that will receive the load-balanced traffic.

- **Load Balancer Resources Define Behavior:**The lab showcased the creation of crucial load balancer resources:
  - **Frontend IP Configuration:** The private IP address within your VNet that the internal clients will connect to.
  - **Load-Balancing Rules:** These rules dictate how incoming traffic on a specific port and protocol is distributed to the backend pool. This lab focused on non-web traffic (TCP/UDP).
  - **Health Probes:** Essential for ensuring high availability. The probe monitors the health of the backend instances and removes unhealthy ones from the rotation.
- **Testing is Crucial for Validation:** The final step emphasizes the importance of testing the load balancer to confirm that traffic is being distributed correctly and that the health probes are functioning as expected.
- **Non-Web Traffic Load Balancing is Straightforward:** The guide illustrates that configuring a load balancer for TCP or UDP traffic follows a similar logical flow to web traffic (HTTP/HTTPS), focusing on the relevant port and protocol.
- Azure Application Gateway is a web traffic (OSI layer 7) load balancer that enables you to manage traffic to your web applications.
- Application Gateway can make routing decisions based on additional attributes of an HTTP request, for example URI path or host headers.
- Use Application Gateway for application hosted in a single region and when you need URL based routing.

**Final Thoughts:**

- **Internal Load Balancers are Vital for Modern Applications:** As organizations increasingly adopt microservices and distributed architectures within their private networks, internal load balancing becomes a critical component for ensuring application stability and scalability.
- **Azure Provides a Robust and Flexible Solution:** Azure Load Balancer offers a managed service that simplifies the process of distributing internal traffic without the overhead of managing load balancer appliances.
- **Understanding the Configuration Steps is Key:** This lab provides a practical understanding of the core components and configuration steps involved in setting up an internal load balancer in Azure. This knowledge is transferable to more complex scenarios.
- **Consider Health Probes Carefully:** The configuration of health probes is paramount for ensuring that only healthy instances receive traffic. Choosing the right probe type and configuration is crucial for the reliability of your load-balanced application.
- **Extensibility and Integration:** While this lab focused on a basic setup, Azure Load Balancer can be further integrated with other Azure services for more advanced scenarios, such as integration with Azure Monitor for detailed metrics and alerting.

Essentially, the key takeaway is that this lab provides a fundamental understanding of how to build a resilient and scalable internal application infrastructure in Azure by effectively distributing non-web traffic using the Internal Load Balancer service. It highlights the core components and the logical flow of configuration, emphasizing the importance of health monitoring and testing.