whitehorses

member of the human network

WHITEBOOK

Azure Application Gateway connection to Linux Virtual Machine over SSH (port 443)

November 2021

Auteur:
Wilbert Spaanderman
INTEGRATIESPECIALIST

Introduction

For my client a new infrastructure is created in Azure portal. An IaaC (Infrastructure as a Code) approach is used for this via Terraform. This infrastructure contains vNets, subnets, securities and lots more of the azure artefacts and all of them created via Terraform. During this setup we had to deal with extra policies what we had to follow enforced by the Azure cloud admins. To test the Terraform securities among the subnets we decided to create an Application Gateway in a public subnet and a Linux Virtual Machine (VM) in a private subnet. The securities should allow the Application Gateway to set a connection to the VM. One of the policies that is set by the Azure cloud admins is that all the traffic should be encrypted. Also, the traffic inside Azure itself. Therefore, the Application Gateway can only call the VM via SSH over port 443.

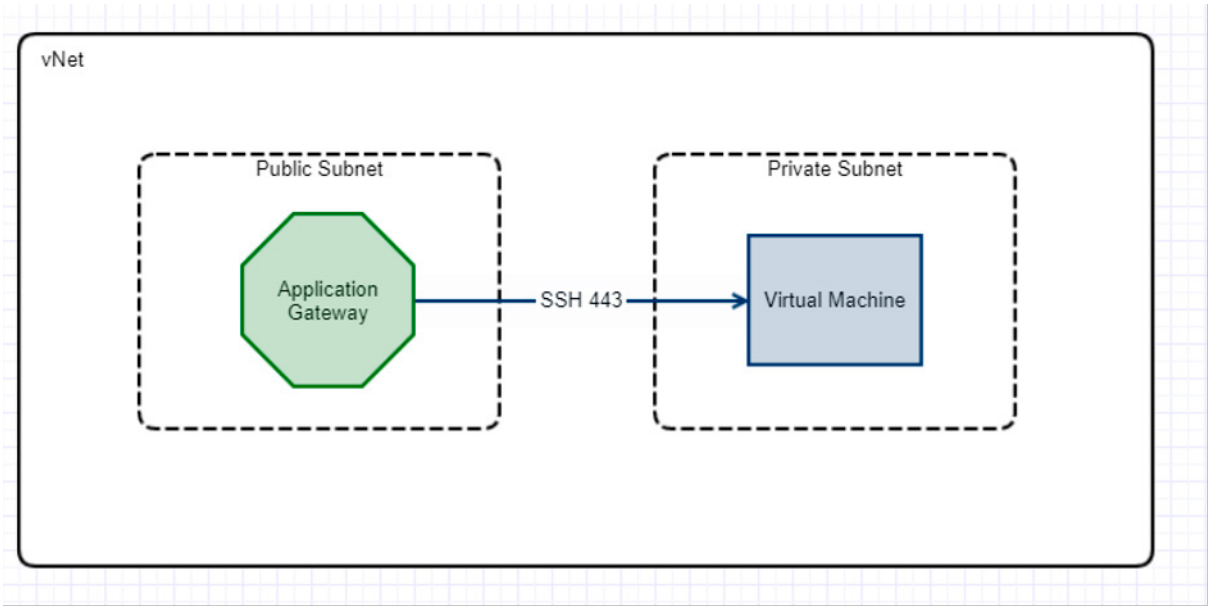

Figure 1: Setup of the infrastructure in Azure

This setup was for testing purposes only and should be made quick and dirty. However, during this Whitebook, you will notice that there are many steps involved in setting up such a simple infrastructure. This Whitebook will go through the following steps in detail:

1. Creation of the Virtual Machine
2. Creation of the Application Gateway including creation of keys
3. Installing an Apache https listener in VM with contains:
    a. Install Apache itself
    b. Add listener on port 443
    c. Test listener
    d. Remove Firewall
4. Short summary

Creation of the Virtual Machine

Prerequisites: Resource groups, vNet, private subnet.

The first step was to create a Linux Red Hat Enterprise 7.8 VM and install on the private Subnet. The first thought was to use the cheap B1ls size here. This small VM costs only € 2.96 per month. However, this is very unstable and there were some problems using 'yum' as installer! That is why we chose 'Standard_DS1_v2'.

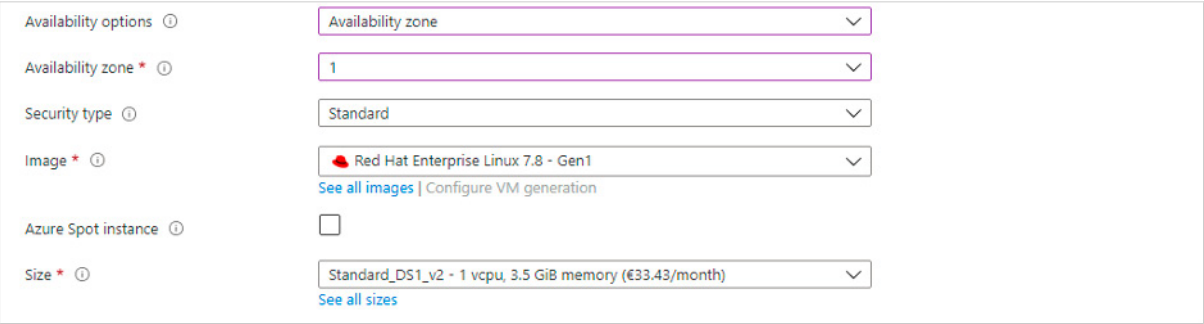

Figure 2: Printscreen of creating a VM

The ports 22 (for SSH connection via bastion) and 443 are allowed for inbound traffic:

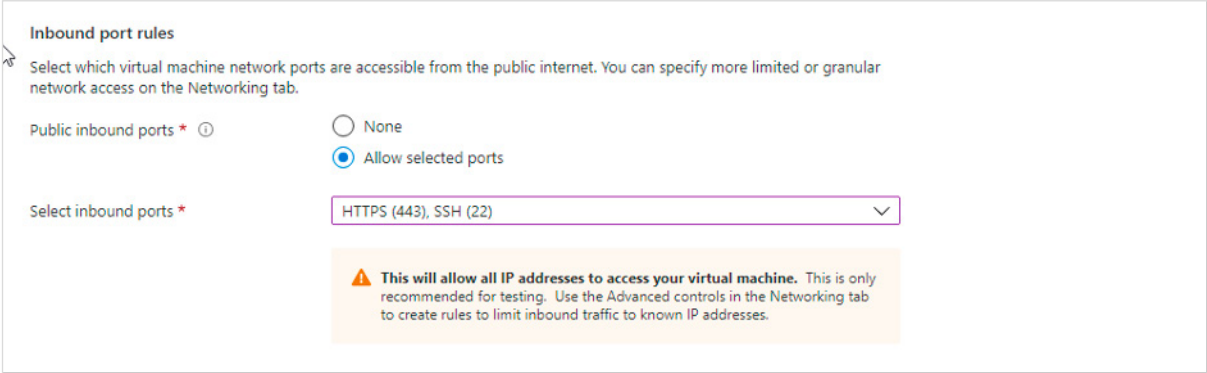

Figure 3: Printscreen of Allowed ports in VM

The rest of the setup is a next, next, finish action. The VM did not get a public IP. This is forbidden by the admin team. Creating a VM is very fast, and it runs in a minute.

Creation of the Application Gateway

Prerequisites: Resource Group, vNet, public subnet

The Azure Application Gateway serves two goals:

1. To secure the inbound traffic via WAF
2. Load balancing between available back ends

For testing the vNet and subnets a gateway is installed on the public subnet. How this is done is specified below.

Basics

The Application Gateway uses the version 1 WAF tier. It contains only one instance count, and the firewall mode is set on 'Prevention'. The rest of the basics is just standard.

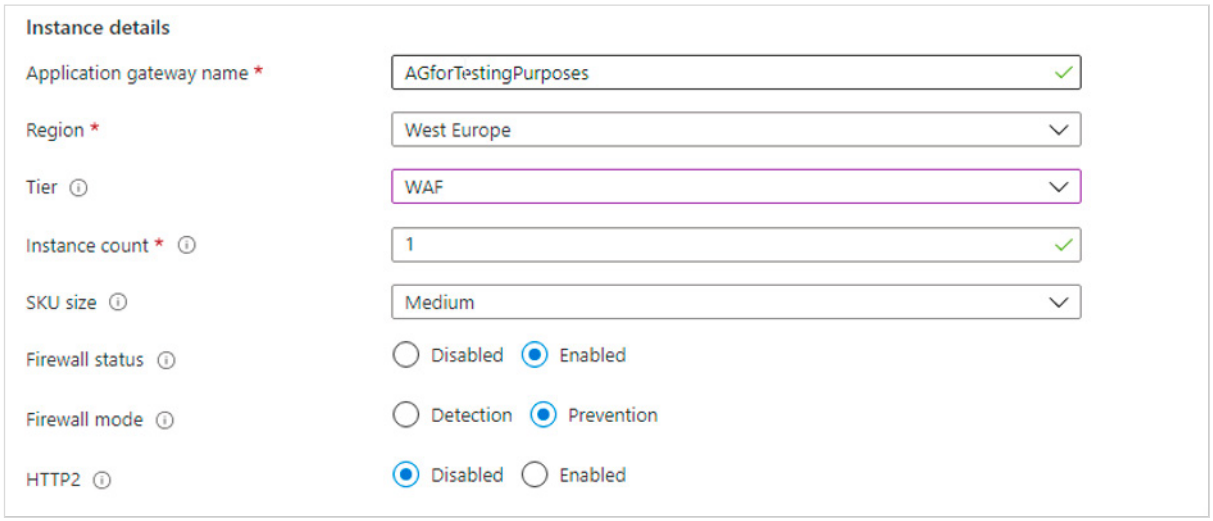

Figure 4: Printscreen of creating the basics for GW

Frontends

In 'Frontends' a new public IP address is created.

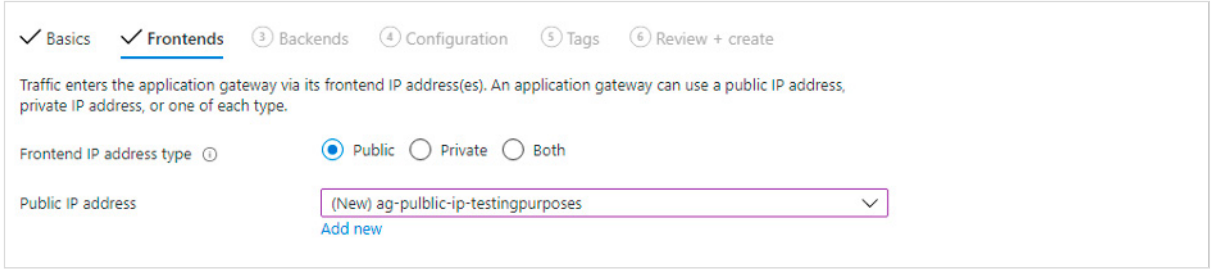

Figure 5: Printscreen of creating a Frontend

Backends

In 'Backends' a new backend pool is created. This backend pool contains the VM created in previous step.

Figure 6: Printscreen of creating a Backend

Create self-signed certificate

The next step is to configure the Application Gateway. In that step a certificate is needed. Azure use the PKCS#12/PFX format for certificates. PFX is not supported by the certificate tool 'Keystore explorer', but it can be created via OpenSSL. Because this gateway is only used for testing purposes, and will be disposed after the test, a self-signed certificate is used.
First: create a private key and certificate with:

```
openssl req -x509 -newkey rsa:4096 -keyout cert.pem -out cert.pem
```

Important I: I have noticed that different versions of OpenSSL behave differently on this command. OpenSSL 3.0.0 in Windows command prompt creates a PEM file called 'cert. pem' containing the private key and a file 'cert' (without extension) with the certificate itself. Before you execute the next command be sure that both the private key and the certificate are stored in the PEM file.
Second: the PEM is converted to a PKCS#12/PFX format:

```
openssl pkcs12 -export -in cert.pem -inkey cert.pem -out cert.pfx
```

Finally, a public certificate was required. This is possible with:

```
openssl x509 -pubkey -outform der -in cert.pem -out cert.cer
```

Important II: Depending on the OpenSSL version the “cert.cer” can contain two keys. One human readable public key and one with strange characters. If this is the case remove the ‘human readable’ public key, because this will give some troubles later on in Azure.

Figure 7: CER file

Configuration

In the configuration part a routing rule is added. Due to the enforced policy that all traffic should be encrypted, this rule contains a listener that listen on the public IP address on port 443. The PFX certificate from the previous step is added here too.

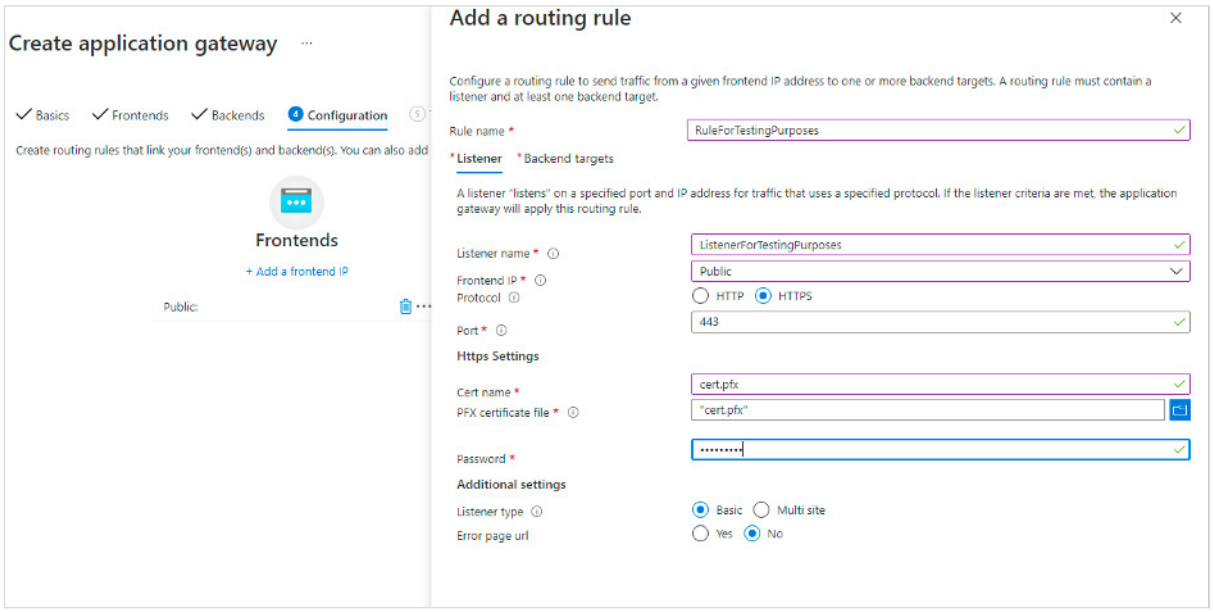

Figure 8: Printscreen of creating a routing rule

On the backend side a new 'HTTP setting' is created. Again, all the traffic also inside Azure should be encrypted so HTTPS on port 443 is used. Here the public certificate is needed from the previous step.

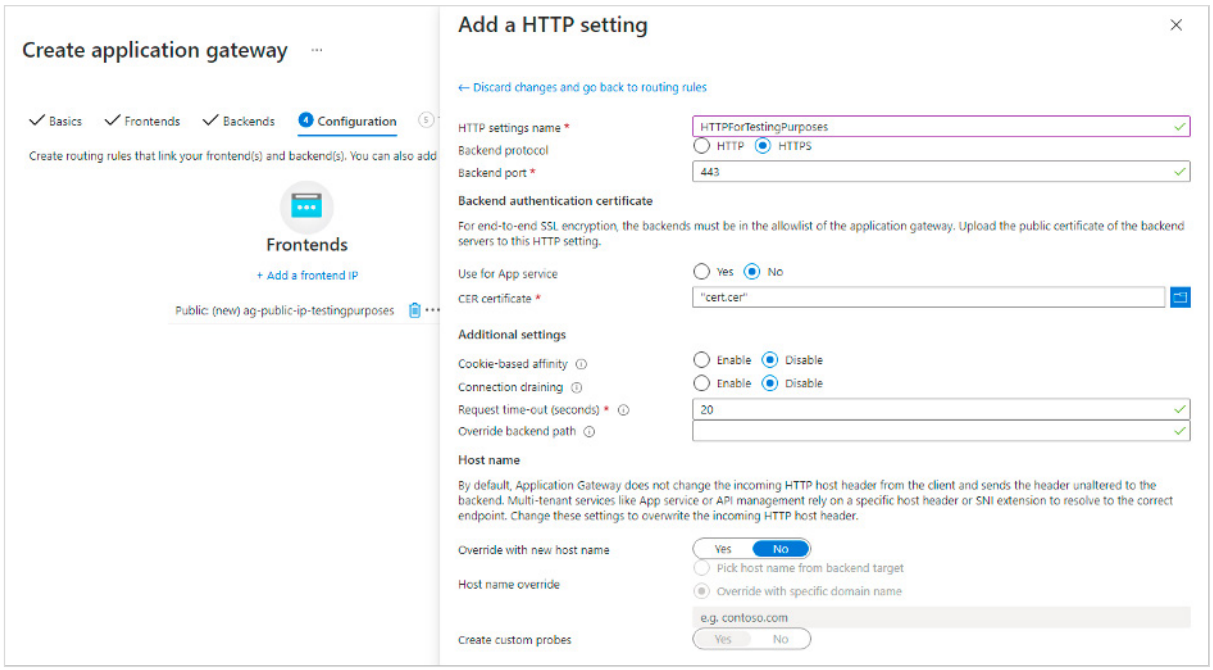

Figure 9: Printscreen of creating a HTTP setting

The rest of the creation is a next, next, finish action. After 15 minutes the Application Gateway is installed.

Remark: Ones an Application Gateway is installed in a subnet, that subnet cannot be used for Virtual Machines anymore. This is forbidden by Azure itself.

Installing Apache https listener in VM

Install Apache

Now both the VM and the Application Gateway are created, the connection health can be checked in the Application Gateway. However, at this moment it will pop up with the error *'Cannot connect to backend server. Check whether any NSG/UDR/Firewall is blocking access to the server. Check if application is running on correct port.'*

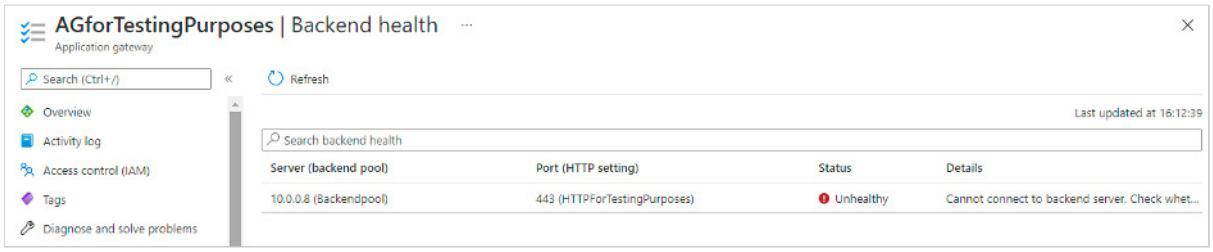

Figure 10: Unhealthy status between GW and VM

This message is thrown because there is no listener installed on the VM that listens to port 443. When you login into the VM and execute the command: *netstat –plnt*, it will show all the ports that it is listening on. By default, 443 is not among them.
To create a listener an Apache server is installed on the VM. This can be easily done with yum:

```
yum install httpd
```

Then start Apache with:

```
systemctl start httpd
```

If Apache should be started after every reboot, then run the following command:

```
systemctl enable httpd.service
```

To verify if Apache is running perform 'curl http://localhost:80'. When everything is right a default HTML response will be given.

Add listener on port 443

Now Apache is installed port 443 can be opened. Port 443 is used for encrypted traffic and

therefore it needs the certificate and private key from the PEM file created in the chapter above. The certificate must be stored in a folder called ssl_certs with filename 'cert.crt' and the private key in a folder called ssl_keys with filename 'private.key'. Both folders must be created under '/ect/httpd'.

Important: When the PEM file is created on a Windows machine it contains both carriage return and line feeds. Linux only recognize line feeds. So before copy pasting it removes the carriage returns first.
To make Apache aware of the certificate and private key the following code is added to *'/etc/httpd/conf/httpd.conf'*:

```
<VirtualHost *:443>
  ServerName azure-eu.cloudi.com
   SSLEngine on
   SSLCertificateFile /etc/httpd/ssl_certs/cert.crt
   SSLCertificateKeyFile /etc/httpd/ssl_keys/private.key
</VirtualHost>
```

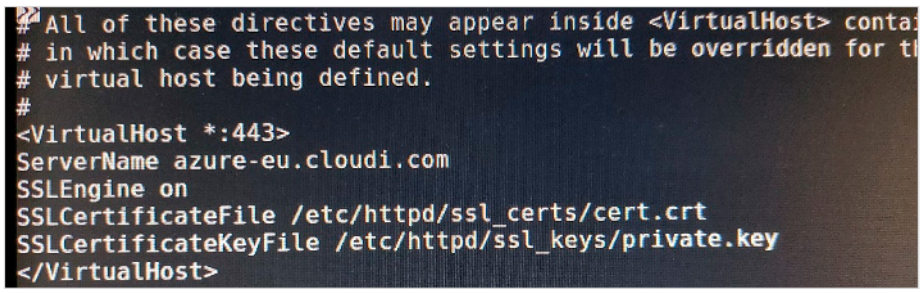

Figure 11: Printscreen of adding virtualhost to the Apache conf

Apache needs also a SSL module that is not installed by default. So, install the module SSL first.

```
yum install mod_ssl
```

Restart Apache and it should work:

```
systemctl restart httpd.service
```

Test 443 listener

To test if Apache is listening to port check 'netstat –plnt' and it shows that there is now a listener available on port 443.

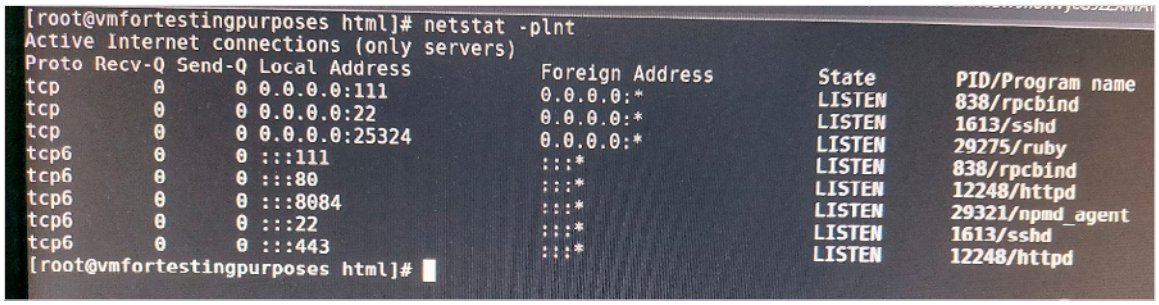

Figure 12: result from netstat -plnt

With 'curl https://localhost:443 -k -v' it is now possible to request the default index.html. The '-k' is needed here because of the self-signed certificate. The –v shows the HTTP status. Maybe unexpected but the HTTP status is a HTTP 403 forbidden.

Apache returns a 403 HTTP status code because of a missing HTML file. In the httpd.conf file it is mentioned that there should be an index.html in '/var/www/html'.

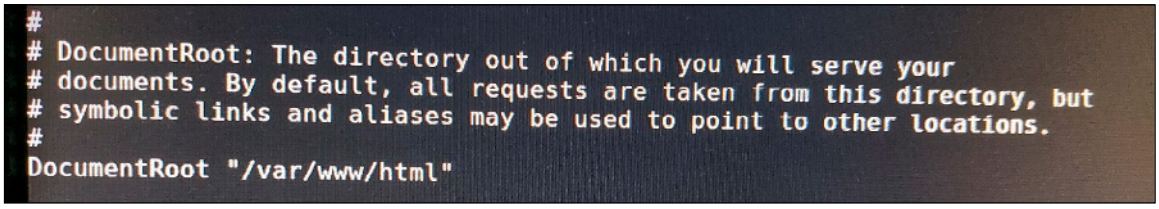

Figure 13: Printscreen from the conf file

The index.html is not created there yet. After creating an index.html file in /var/www/html' with some basic HTML code, like *<html> Hello World </html>*, the curl command will return a 200 HTTP status code.

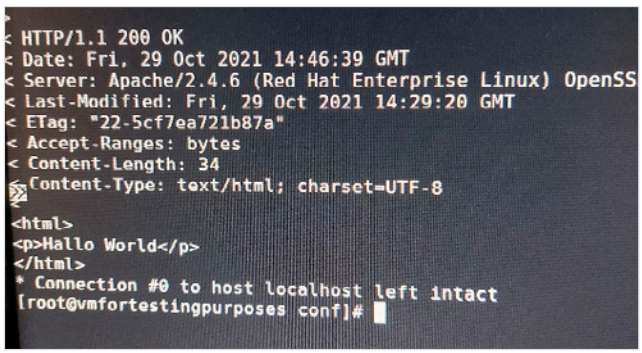

Figure 14: result from the curl command

Remove the firewall

The last step is removing the firewall running on the VM. By default, the Linux Red Hat VM has a firewall running. This is blocking all incoming traffic on HTTPS.
This can be done in two ways:

1. Change the firewall by allowing HTTPS traffic: firewall-cmd --add-service=http –permanent. And restart the firewall after: firewall-cmd –reload or/and systemctl restart firewalld.service
2. Stop the firewall: systemctl stop firewalld.service

After changing the firewall, the backend health of the Application Gateway will give a Healthy status and the process is done.

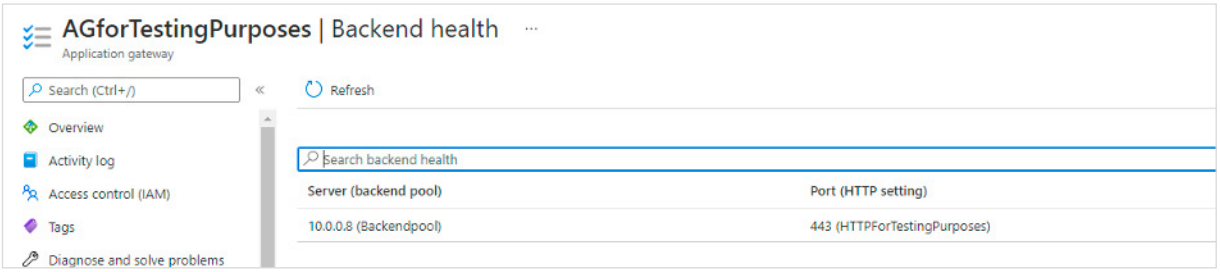

Figure 15: Healthy status of the connection between GW and VM

Since the Application Gateway does have a public endpoint, it is now possible to perform the curl command from a local machine to the online Application Gateway. For example, '*curl https://90.1.2.3.4 -k*'. This command will return now the index.html created in the Virtual Machine.

Conclusion

Now the SSH connection between the Application Gateway and the Linux Virtual Machine is created successfully, it is straightforward to test the securities among the subnets. During the creation of the Application Gateway and the Linux Virtual Machine everything was allowed in the security section. Now we can start to play with denials and approvals in the security section. For example, denying all inbound traffic from the vNet to private subnet. This will result in a bad connection between the Application Gateway and the Virtual Machine.

Priority	Name	Port	Protocol	Source	Destination
1020	AllowBastionAccess	22,3389	Any	[redacted]	Any
4000	DenyALL	Any	Any	VirtualNetwork	Any

Figure 16: Denial all inbound traffic from vNet to private subnet

Priority	Name	Port	Protocol	Source	Destination
1020	AllowBastionAccess	22,3389	Any	[redacted]	Any
3998	AllowPublicSubnet	Any	Any	[redacted]	Any
4000	DenyALL	Any	Any	VirtualNetwork	Any

Figure 17: Bad connection between GW and VM

Since it is a private subnet, we want to deny all inbound traffic from the vNet. However, the Application Gateway should be an exception in this rule. One way on how to create this exception is to allow all traffic from the public subnet (which contains the Application Gateway) to the private subnet. And give that rule a higher priority.

Server (backend pool)	Port (HTTP setting)	Status
10.0.0.8 (Backendpool)	443 (HTTPForTestingPurposes)	Healthy

Figure 18: Added exception rule for public subnet

Server (backend pool)	Port (HTTP setting)	Status
10.0.0.8 (Backendpool)	443 (HTTPForTestingPurposes)	Healthy

Figure 19: Healthy status after added the exception for the public subnet