Deploying Azure Firewall with Terraform
A Real-World Hub-and-Spoke VPN Architecture

Document Type:
Technical Report / Infrastructure Deployment Guide

Author:
Waseem John
Sr. Tech Support Engineer, SEE
Boxborough, MA, USA
waseemjohn@microsoft.com

Date:
July 2025

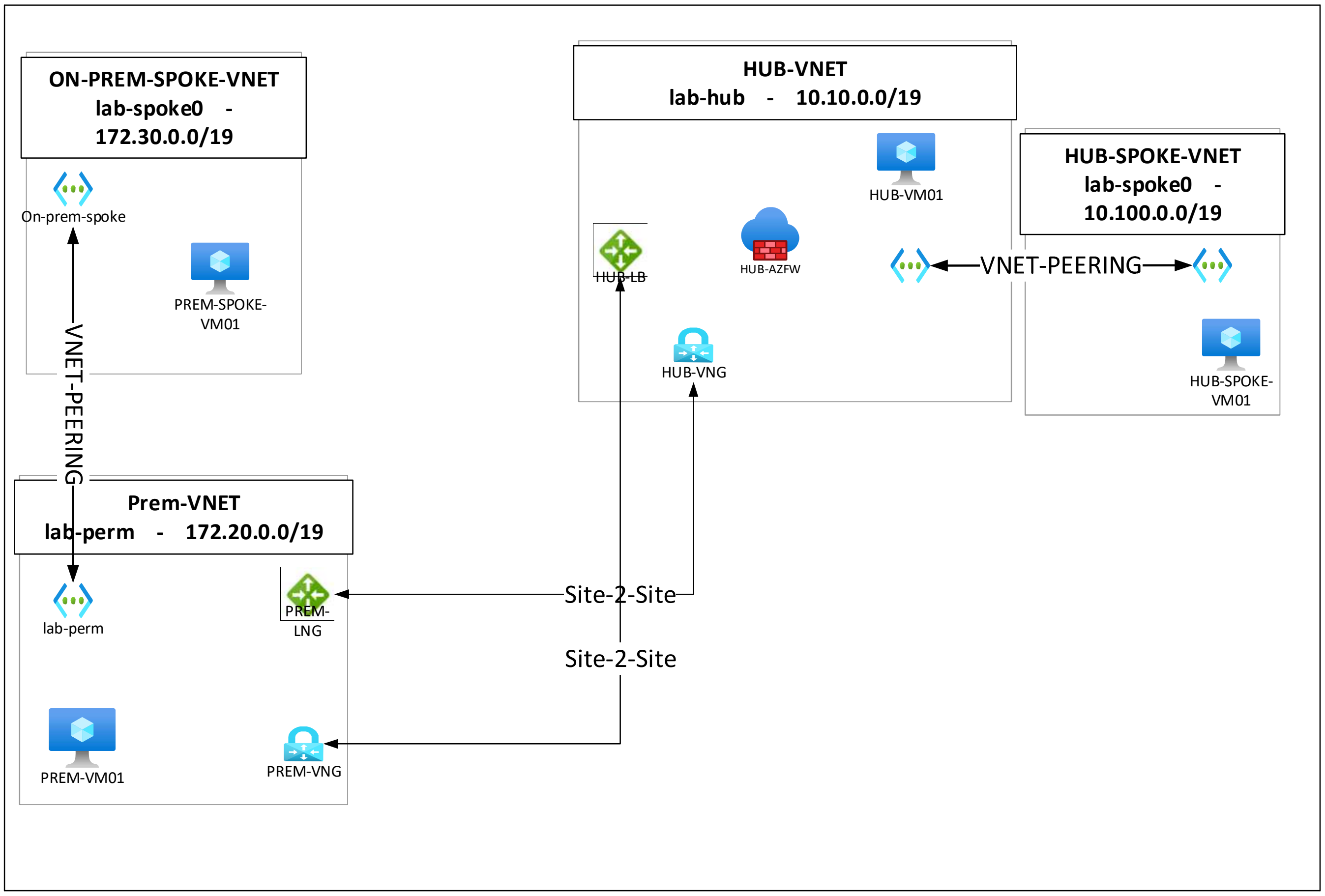
Abstract:
This technical report presents a real-world implementation of a secure, scalable Azure Firewall deployment using Terraform. The architecture follows a hub-and-spoke model with site-to-site VPN connectivity, centralized traffic inspection, and advanced routing strategies. It includes practical Terraform code, deployment steps, and best practices for enterprise-grade cloud security.

Keywords:
Azure Firewall, Terraform, VPN Gateway, Hub-and-Spoke, Cloud Networking, Infrastructure as Code, Network Security

Azure Firewall, Hub-and-Spoke VPN Deployment with Terraform

This article outlines the implementation of an Azure Firewall solution designed to securely connect two sites using a Site-to-Site IPsec VPN, provisioned through Terraform. The architecture follows a hub-and-spoke model: the Azure Hub hosts both the Azure Firewall and VPN Gateway, while the second site simulates an on-premises network with its own VPN Gateway. Additionally, a spoke virtual network is connected to the hub to represent an additional Azure VNet or branch office scenario. Each network segment—including the hub, spoke, and on-premises site—contains deployed Virtual Machines (VMs).

All inter-site and outbound internet traffic is routed through the Azure Firewall to enable centralized inspection and policy enforcement. The report also highlights recent enhancements to Azure Firewall, including advanced threat protection capabilities and performance optimizations. Best practices, cost considerations, and insights from real-world deployments are also discussed to support informed decision-making.



Network Architecture and Deployment Steps Terraform Code and Network Layout:

Environment One: Hub Virtual Network (VNet)

- **Address Space:** 10.10.0.0/19
- The Hub VNet serves as the central point for connectivity and security enforcement. It includes the following subnets:
  - **AppSubnet (Hub-App):** 10.10.0.0/24 – Hosts application virtual machines.

    - **WebSubnet (Hub-Web):** 10.10.1.0/24 – Hosts web-tier virtual machines.
    - **GatewaySubnet (Hub):** 10.10.7.0/26 – Dedicated to the VPN gateway. Azure mandates the name GatewaySubnet and recommends a subnet size of /27 or larger.
    - **AzureFirewallSubnet (Hub):** 10.10.30.0/24 – Reserved for Azure Firewall. This subnet must be named AzureFirewallSubnet and be at least /26 in size.

Environment Two: On-Premises Virtual Network (Prem VNet)

- **Address Space:** 172.20.0.0/19
- This VNet simulates an on-premises environment and mirrors the subnet structure of the Hub:
    - **AppSubnet (Prem-App):** 172.20.0.0/24
    - **WebSubnet (Prem-Web):** 172.20.1.0/24
    - **GatewaySubnet (Prem):** 172.20.7.0/26 – Used for the on-premises VPN gateway.

Environment Three: Hub Spoke Virtual Network

- **Address Space:** 10.100.0.0/19
- This spoke VNet represents an additional Azure environment, potentially in a different region. It includes:
    - **AppSubnet (Spoke-App):** 10.100.0.0/24
    - **WebSubnet (Spoke-Web):** 10.100.1.0/24
- This spoke does not include a gateway subnet, as it leverages the Hub's VPN gateway via VNet peering.

Environment Four (Optional): Prem Spoke Virtual Network

- **Address Space:** 172.30.0.0/19
- Defined for completeness, this VNet includes:
    - **Subnets:** 172.30.0.0/24, 172.30.1.0/24
- Although provisioned in code, this environment is not actively used in the current scenario.

Regional Deployment and Peering Configuration

Each environment is deployed in a designated Azure region—for example, the Hub in West US 2, the Prem VNet in Canada Central, and the Hub Spoke in East US. Peering is established between the Hub and its spokes to enable traffic flow and centralized security inspection.

In Terraform, bidirectional peering is configured with the following settings:

- **allow_virtual_network_access = true**
- **allow_forwarded_traffic = true**

These settings ensure that traffic from the spoke can be routed through the Hub and inspected by the Azure Firewall. Although use_remote_gateways = false is set on the spoke peering (meaning the spoke does not directly use the Hub's VPN gateway), internet-bound traffic from the spoke is still routed through the Hub Firewall for centralized control.

Step 1 – Deploy Resource Groups and Virtual Networks:

Using Terraform, we first create Azure Resource Groups for each major component (Hub, Prem, Hub-Spoke, Prem-Spoke). Then we create the Virtual Networks and their subnets as listed above, in each respective resource group. Key points to remember:

- The GatewaySubnet must be named exactly "GatewaySubnet" in each VNet to host a VPN gateway, and sized at least /27.
- The AzureFirewallSubnet in the Hub must be named exactly "AzureFirewallSubnet" and sized at least /26. This is required to ensure Azure can allocate enough IPs for scaling the firewall (a /26 provides 64 addresses for the firewall instances).
- Ensure none of the address ranges overlap between Hub, Prem, and spokes, otherwise the VPN routing will not work correctly.

Step 2 – Deploy VPN Gateways for Site-to-Site Connection:

Next, we create the Virtual Network Gateways (VPN gateways) in the Hub and Prem VNets. In Terraform, we define azurerm_virtual_network_gateway resources (commented in our code as "vpn_gateway_one" for Hub and "vpn_gateway_two" for Prem). Each gateway resource includes:

- **type = "Vpn" and vpn_type = "RouteBased"** (required for site-to-site IPsec with dynamic routing).
- An SKU, e.g., VpnGw1 or VpnGw2 depending on performance needs (our example uses VpnGw1 which supports BGP, needed for our setup).
- **enable_bgp = true** if using BGP (we enabled BGP in our code), along with a local asn (Autonomous System Number) for each gateway's BGP config. In the code, we set hub_asn = 65451 and prem_asn = 65401 in local variables and used those in the gateway BGP settings.
- An ip_configuration that links the gateway to the VNet: referencing the GatewaySubnet ID and a Public IP resource for the gateway. We allocate a Static Public IP for each VPN gateway (Terraform azurerm_public_ip with sku = "Standard" for zone-redundancy) – e.g., one in West US 2 for the Hub gateway, and one in Canada Central for the Prem gateway (our code defines these but had them commented out as "public_ip_one" and "public_ip_two").

After applying Terraform, Azure will provision the two VPN gateways. Each will have a public IP address (say, Hub VPN = X.X.X.X, Prem VPN = Y.Y.Y.Y). These gateways will terminate the IPsec VPN tunnel.

Step 3 – Configure the Site-to-Site IPsec Tunnel:

With the VPN gateways in place, we establish the connection between them:

- Create Local Network Gateway representations for each side. In Azure, a Local Network Gateway (LNG) is an object that represents the remote VPN device and its address space. In Terraform, we define azurerm_local_network_gateway for the Hub (pointing to the Prem gateway's public IP and prem network prefixes) and another for the Prem (pointing to the Hub gateway's IP and hub/spoke prefixes). In our code, for example, the Hub's LNG (local_network_gateway_two2) uses gateway_address = the Prem VPN's public IP and includes an address_space list of the Prem VNet prefixes and the Hub's spoke prefixes that will come via Prem. (We included on-prem branch ranges like 10.100.0.0/19 in the Hub LNG to let the Hub know about the Spoke network on the other side as well.) Similarly, the Prem LNG (local_network_gateway_two) would list the Hub VNet (10.10.0.0/19) and any networks behind Hub (like 10.100.0.0/19).

- Create the Connection resource to link the two VPN gateways. In Terraform, we use azurerm_virtual_network_gateway_connection with type = "IPsec". We reference the Hub VNet gateway ID, the Prem Local Network Gateway ID, and specify a shared_key (the pre-shared key for the IPsec tunnel). Our configuration uses a variable or local value for the shared key (e.g., "123456" for demonstration). We set enable_bgp = true to use BGP over the tunnel (so the gateways exchange routes automatically). In the code, we defined two connection resources: "hub-to-prem" and (commented) "prem-to-hub" for completeness. Azure actually only requires one connection object on the Azure side when connecting two Azure VPN gateways; if both gateways are Azure, a single connection with the local and remote gateways defined will bring up the tunnel in both directions. After Terraform applies, Azure will establish the IPsec tunnel using the shared key and the endpoints.

At this stage, we have a VPN tunnel connecting the Hub VNet and the Prem VNet. If BGP is enabled, the gateways will exchange routes. For example, the Hub gateway will learn the 172.20.0.0/19 network (Prem) and the Prem gateway will learn the 10.10.0.0/19 (Hub) as well as 10.100.0.0/19 (Spoke) if BGP is set up correctly. (If not using BGP, we would manually specify the address spaces in the Local Network Gateways, which we did, so static route mapping is achieved by that.)

Step 4 – Deploy Azure Firewall in the Hub:

With connectivity in place, we now add the Azure Firewall to the Hub to secure traffic. Using Terraform, we define an azurerm_firewall resource in the Hub resource group and VNet. Key steps:

- Allocate a Public IP for the firewall (Standard SKU). Our code shows a resource (commented as azurerm_public_ip.azfw-pip) for this. The firewall will use this IP for outbound SNAT and (optionally) inbound DNAT.
- Create an Azure Firewall Policy (optional but recommended) to hold firewall rules. In code, we have an azurerm_firewall_policy named "hub-azfw-policy" with sku = "Premium", indicating we intend to use the Premium firewall features (more on that later). Even if using Standard, a policy can be used. We would define rule collections (application rules, network rules, NAT rules) within this policy. (Our Terraform shows just creation of an empty policy in Premium tier as a placeholder.) Deploy the Firewall resource itself, referencing the subnet and public IP. In Terraform, we attach the firewall to the Hub VNet's AzureFirewallSubnet (using subnet_id = azurerm_subnet.AzureFirewallSubnet.id) and the public IP (public_ip_address_id = azurerm_public_ip.azfw-pip.id). We also attach the firewall to the policy (via firewall_policy_id). In our config, we set sku_name = "AZFW_VNet" and sku_tier = "Premium" to get the Premium SKU firewall. Azure Firewall will automatically provision in a highly available way across availability zones (if supported in region) and can scale out as needed without further user action.

After deployment, the firewall receives a private IP in the 10.10.30.0/24 subnet (Azure will pick one, often the first usable like 10.10.30.4). In our later steps, we'll use that IP for routing. Important: There should be no NSG on the AzureFirewallSubnet and no user-defined route on it, unless configuring forced tunneling (which we are not doing for the firewall itself). By default, the firewall will have system routes to know about the Hub and learned BGP routes for the Prem.

Step 5 – Configure Routing so All Traffic Flows Through the Firewall:

This is crucial for ensuring “all traffic is protected through Azure Firewall,” as required:

- **User-Defined Routes (UDRs) in Hub and Spoke:** We create route tables and assign them to subnets to direct traffic. In our Terraform code (in the gwfw.tf snippet), we planned route tables for Hub, Spoke, and Prem subnets (named by region). The general approach:
    - For any spoke or application subnet where VMs reside (Hub’s app subnet, Spoke’s app subnet, Prem’s app subnet), add a default route: destination 0.0.0.0/0 with Next hop type “Virtual Appliance” and Next hop IP = Azure Firewall’s private IP (e.g., 10.10.30.4). This forces all internet-bound traffic from those subnets to go to the firewall first. In our Terraform, for example, we had a route named “From-Spoke” in the spoke’s route table sending 0.0.0.0/0 to 10.14.30.4 (the firewall IP in an earlier address plan), and similarly “From-Hub” and “From-Prem” routes in their respective tables. All those point to the firewall IP. We then associate these route tables with the corresponding subnets (Terraform azurerm_subnet_route_table_association).
    - **Route on the Hub’s GatewaySubnet:** We want any traffic coming in from the VPN (Prem site) destined for Azure to also pass through the firewall. By default, the VPN gateway will send traffic to the target VNet directly. To intercept it, we put a UDR on the Hub’s GatewaySubnet that directs traffic for Azure subnets to the firewall. For instance, we could route the Spoke’s range (10.100.0.0/19) or even all Azure-bound traffic (10.0.0.0/8, depending on scenario) to the firewall’s IP. Microsoft’s hub-and-spoke guidance suggests if you want full firewall inspection, you disable BGP route propagation and use manual routes like this. In our simpler setup, since the Prem gateway knows about 10.100.0.0/19 via BGP or config, the traffic will come into the Hub gateway. We then rely on the fact that the Spoke’s own UDR (0/0 to firewall) will catch any traffic the gateway forwards to the Spoke’s subnet, or alternatively, the Hub gateway subnet UDR could send spoke-bound traffic to the firewall (then firewall to spoke). Our Terraform code did not explicitly include a GatewaySubnet UDR (it’s often done via Azure Portal in such scenarios), but it’s a consideration for a real deployment.
- **On-Premises device configuration:** On the actual on-premises side (if the second site were a physical network), you’d configure routes so that Azure-bound traffic goes into the VPN tunnel. In our case, the “Prem” is an Azure VNet, so the Azure VPN gateway there automatically knows to send traffic for 10.10.0.0/19 and 10.100.0.0/19 to the tunnel (via BGP or config). For internet traffic from on-prem, we have two choices: either let it break out locally (not filtered by Azure Firewall), or force it through the VPN to Azure so that the Azure Firewall can inspect and egress it. We demonstrated the latter by creating a “From-Prem” 0.0.0.0/0 route on the Prem subnet pointing to the Azure Firewall. This effectively hairpins the on-prem VM’s internet traffic over the VPN into Azure Firewall, which then goes out to the internet. This setup requires careful design to avoid asymmetric routing, as discussed later.

With these routes in place, the Azure Firewall becomes the central transit point. For example:

- The Hub VM’s internet traffic goes to firewall (via UDR) and then out to Internet (SNAT’d to firewall’s public IP).

- The Spoke VM’s internet traffic goes across the VNet peering to the Hub firewall (because its UDR sends to firewall IP which resides in the Hub; Azure peering allows this cross-VNet next-hop) and then out.
- The Prem VM’s internet traffic goes over the VPN to Hub firewall (due to the prem UDR) and then out to Internet from Azure.
- Any cross-site traffic (Hub VM to Prem VM, or Spoke to Prem) will similarly traverse the firewall. If Hub VM sends to Prem, it goes to firewall (if destination in 172.20.0.0/19 might not match 0/0, but we would also include specific routes if needed; in hub, if BGP is on, it might learn route to 172.20 and go to VPN directly unless we override). To ensure even site-to-site traffic goes through firewall, we could add a route for 172.20.0.0/19 on Hub subnets pointing to firewall. The firewall would then need a network rule to allow that traffic and would forward it to the VPN gateway. In our scenario, since we enabled BGP, the hub might learn 172.20 and try to send directly to gateway, but we could disable that via setting “Use Remote Gateway” to false on hub (which we did) and instead rely on UDRs. This is an advanced point – essentially, we are forcing all traffic through firewall at some point either via direct default route or by controlling propagation.

Step 6 – Deploy VMs and Test Traffic:

We have three VMs in our scenario (as per Terraform code):

- A Hub VM (Windows, in Hub-App subnet) with a public IP for RDP in our code.
- A Prem VM (Windows, in Prem-App subnet) with a public IP in code.
- A HubSpoke VM (Windows, in Spoke-App subnet) with a public IP in code.
- A PremSpoke VM (Windows, in Spoke-App subnet) with a public IP in code.

Each VM has a Network Interface associated with the respective subnet and we applied basic Network Security Groups to allow RDP from a specific client IP for testing. Once the VMs are up, we can test connectivity:

Conclusion and Benefits

To conclude, deploying an Azure Firewall with a hub-and-spoke VPN architecture not only enhances security but also centralizes traffic management, ensuring a robust and scalable network infrastructure. By following the outlined steps and best practices, you can achieve a secure and efficient deployment that meets your organization's needs.

Additionally, this documentation can be incredibly beneficial for your environment. By testing the functionality in a lab environment before going to production, you can thoroughly review and validate the changes. This approach allows you to identify and address any potential issues, ensuring a smooth and successful implementation in the real environment. Your efforts in preparing this documentation provide a valuable resource for others to follow, helping to streamline the deployment process and improve overall network security.

Source Code

The complete Terraform source code used in this deployment is available on GitHub:

https://github.com/waseemoncloud/azfw_vpngw/tree/main