Here’s a structured breakdown to help you **successfully instantiate Azure Private Link Service (PLS)** — *not to be confused with Azure Private Link* — for **exposing on-prem assets (like services or APIs) to customers over private connectivity**:

### **✅ Benefits of Azure Private Link Service**

Azure Private Link Service allows you to expose services running **behind a standard load balancer in your Azure VNet** to customers via **private endpoints in their own VNets**, even if your service originates from **on-prem via hybrid architectures**.

* **Private Access for Customers**  
   Customers connect to your service using private IP addresses over Azure backbone — no public internet exposure.
* **Enhanced Security Posture**  
   Reduces attack surface and supports granular NSG and NSX policies.
* **Controlled Exposure**  
   You control who can connect via approved private endpoints, reducing the risk of unauthorized access.
* **Hybrid Scenario Support**  
   Supports NVA or ExpressRoute-connected backends, so on-prem services can be made available via Azure PLS.
* **Scalable Multi-Tenant Design**  
   Ideal for ISVs or partners exposing multi-tenant services securely to multiple customers.

### **⚠️ Constraints**

Key limitations you must consider when deploying PLS:

* **Standard Load Balancer Required**  
   PLS only works with a Standard SKU Load Balancer (no Basic SKU support).
* **TCP/UDP Only**  
   Only supports TCP and UDP protocols — **no HTTP-layer routing (Layer 7)**.
* **No Direct On-Prem Hosting**  
   The service itself must be hosted in an Azure VNet (e.g., as an NVA or proxy to on-prem), not purely on-prem.
* **Inbound Connectivity Only**  
   Customers can initiate traffic, but not vice versa — your service can’t initiate outbound traffic to customer VNet.
* **Manual DNS Integration**  
   Customer must manually configure DNS or use Private DNS Zones — it’s not automatic.

### **🔍 Considerations for Successful Setup**

* **On-Prem Integration via ExpressRoute/VPN**  
   If the actual service is on-prem, integrate using **Azure VPN Gateway or ExpressRoute** to route traffic from Azure Load Balancer to the on-prem service.
* **NVA or Proxy in Azure**  
   For security and routing, deploy a **Network Virtual Appliance (NVA)** or **Azure Firewall** in a spoke VNet to proxy requests to on-prem assets.
* **Subnet Planning**  
   The PLS must be fronted by a **Standard Load Balancer with a dedicated subnet**, and the backend pool should point to reachable IPs (e.g., NVA or on-prem via gateway).
* **IP Whitelisting and NSGs**  
   Restrict access to approved customer VNets using **Private Endpoint connection approval** and **NSG/UDR rules**.
* **Monitoring and Auditing**  
   Integrate with **Azure Monitor, Network Watcher**, and **Diagnostic Logs** for visibility into PLS operations.
* **Cost Implications**  
   Private Link Services and Standard Load Balancer have **per-hour and per-GB egress costs** — evaluate based on expected usage.

### **🔗 Dependencies and Required Resources**

|  |  |
| --- | --- |
| **Resource/Dependency** | **Description** |
| **Azure VNet (Hub/Spoke)** | Required to host the Standard Load Balancer and proxy or NVA if on-prem is involved |
| **Standard Load Balancer (Internal)** | Mandatory to front-end your Private Link Service |
| **Backend Pool (e.g., NVA, VM, App)** | Must include reachable targets (e.g., on-prem routed via VPN Gateway or ExpressRoute) |
| **Private Link Service** | Configured with the Load Balancer frontend and backend |
| **NSG/UDR** | To restrict or route traffic between the PLS, backend, and customer endpoints |
| **ExpressRoute/VPN Gateway** | Needed if exposing on-prem resources via Azure PLS |
| **Private Endpoint (Customer Side)** | Customers will create a Private Endpoint connection to your PLS |
| **Private DNS Zone (Customer Side)** | For internal DNS resolution of the service exposed via PLS |

Let me know if you'd like this in a **Visio-style diagram**, **Terraform template**, or **step-by-step deployment guide**.

##### **You said:**

yes

###### **ChatGPT said:**

Great — since you'd like a **visual diagram**, a **Terraform template**, and a **step-by-step deployment guide**, here’s the full breakdown:

## **📊 Architecture Diagram: Azure Private Link Service for On-Prem Exposure**

Here’s a description of the architecture (you can visualize or diagram this in Visio, draw.io, or Lucidchart):

java

CopyEdit

Customer VNet  
└── Private Endpoint ─────┐  
 │  
 ▼  
 Azure Private Link Service (PLS)  
 ▲  
 Standard Load Balancer (Internal - TCP/UDP Only)  
 └── Backend Pool (e.g., Azure NVA, Azure VM)  
 |  
 Azure VPN Gateway / ExpressRoute  
 |  
 On-Prem Service (e.g., API, DB)

## **🛠️ Terraform Template Snippet (PLS & Dependencies)**

Assumes you already have the VPN/ExpressRoute & NVA in place.

hcl

CopyEdit

resource "azurerm\_resource\_group" "example" {  
 name = "rg-private-link-service"  
 location = "eastus"  
}  
  
resource "azurerm\_virtual\_network" "example" {  
 name = "vnet-pls"  
 address\_space = ["10.10.0.0/16"]  
 location = azurerm\_resource\_group.example.location  
 resource\_group\_name = azurerm\_resource\_group.example.name  
}  
  
resource "azurerm\_subnet" "lb" {  
 name = "subnet-lb"  
 resource\_group\_name = azurerm\_resource\_group.example.name  
 virtual\_network\_name = azurerm\_virtual\_network.example.name  
 address\_prefixes = ["10.10.1.0/24"]  
 enforce\_private\_link\_service\_network\_policies = true  
}  
  
resource "azurerm\_lb" "internal" {  
 name = "internal-lb"  
 location = azurerm\_resource\_group.example.location  
 resource\_group\_name = azurerm\_resource\_group.example.name  
 sku = "Standard"  
 frontend\_ip\_configuration {  
 name = "internal-fe"  
 subnet\_id = azurerm\_subnet.lb.id  
 private\_ip\_address\_allocation = "Dynamic"  
 }  
}  
  
resource "azurerm\_private\_link\_service" "pls" {  
 name = "pls-onprem"  
 location = azurerm\_resource\_group.example.location  
 resource\_group\_name = azurerm\_resource\_group.example.name  
 load\_balancer\_frontend\_ip\_configuration\_ids = [  
 azurerm\_lb.internal.frontend\_ip\_configuration[0].id  
 ]  
 ip\_configuration {  
 name = "pls-config"  
 private\_ip\_address = "10.10.1.100"  
 subnet\_id = azurerm\_subnet.lb.id  
 }  
 visibility {  
 subscription\_ids = ["<customer-subscription-id>"]  
 }  
 auto\_approval {  
 subscription\_ids = ["<customer-subscription-id>"]  
 }  
}

## **🧭 Step-by-Step Deployment Guide**

### **1. Prepare Your On-Prem Connection**

* Ensure ExpressRoute or VPN Gateway is set up.
* On-prem target (e.g., API, app, DB) must be routable from Azure.

### **2. Deploy VNet + Subnet for Load Balancer**

* Create a VNet (hub or spoke) and subnet for the Standard Load Balancer.
* Enable enforce\_private\_link\_service\_network\_policies = true.

### **3. Deploy NVA or Jump Host (Optional)**

* If routing traffic to on-prem, deploy an Azure VM or NVA in a backend subnet.

### **4. Create Internal Standard Load Balancer**

* Configure backend pool with the NVA or VM IP.
* Add a health probe and TCP load balancing rule.

### **5. Create Azure Private Link Service**

* Associate with LB frontend.
* Define auto-approval rules for customer subscriptions.
* Set up visibility scope (subscription, tenant, or all).

### **6. Customer: Create Private Endpoint**

* Customers create Private Endpoint to your PLS resource ID.
* DNS setup (manual or using Private DNS Zones) is required.

### **7. Test and Monitor**

* Validate traffic flow: Customer VNet → PLS → Load Balancer → On-Prem.
* Use NSG, UDR, and diagnostic logs to troubleshoot and monitor.

## **🧠 Optional Enhancements**

* **TLS Termination**: Use App Gateway + PLS if TLS/HTTPS required.
* **Rate Limiting/Inspection**: Insert Azure Firewall or 3rd-party NVA.
* **Auditing**: Enable diagnostic logs for PLS, NSGs, and Load Balancer.

NAT VMs and Load Balancers serve different but complementary roles in Azure Private Link Service architectures:

## **NAT VMs (Network Address Translation VMs)**

NAT VMs are needed when you have resources in private subnets that require outbound internet access but shouldn't be directly accessible from the internet. They act as a gateway for outbound traffic while maintaining security:

* **Outbound internet access**: Private resources can reach external services (updates, APIs, etc.) through the NAT VM
* **Security boundary**: Prevents direct inbound connections to private resources
* **Centralized control**: Single point for managing and monitoring outbound traffic
* **IP address management**: Provides a consistent public IP for outbound connections

## **Load Balancers in Private Link Service**

Load Balancers are essential components of Azure Private Link Service for several reasons:

* **Traffic distribution**: Distributes incoming private endpoint connections across multiple backend instances
* **High availability**: Ensures service continuity if backend instances fail
* **Scalability**: Handles increased load by spreading traffic across multiple targets
* **Health monitoring**: Continuously checks backend health and routes traffic only to healthy instances
* **Single entry point**: Provides a unified interface for the Private Link Service

## **Why Both Together?**

In a typical Private Link Service setup:

1. **Private Link Service** connects to a Standard Load Balancer (required)
2. **Load Balancer** distributes traffic to backend VMs in private subnets
3. **NAT VMs** provide outbound internet access for those same backend VMs when needed

The Load Balancer handles inbound private connectivity, while NAT VMs handle outbound internet connectivity. This separation of concerns provides both secure private access through Private Link and controlled outbound access through NAT, creating a robust and secure architecture.

Without the Load Balancer, you couldn't create the Private Link Service. Without NAT VMs, your private backend resources would be isolated from external services they might need to access.

Here's a **design scope** for implementing **Azure Private Link Service (PLS)**—not to be confused with Azure Private Endpoint or Azure Private Link itself. This scope helps align technical, security, and operational aspects when exposing services privately to customers or partners over Azure’s backbone network.

## **🔧 Design Scope: Azure Private Link Service (PLS)**

### **1. Objective**

Establish a secure and private connectivity channel from customer or partner networks (on-premises or Azure) to your service hosted in Azure via **Azure Private Link Service**.

### **2. Use Cases**

* Enable private access to a **partner-hosted service** (e.g., APIs, databases, custom apps).
* Allow **on-premise clients** to consume Azure services over **ExpressRoute or VPN**.
* Avoid exposing services to the public internet for **regulatory compliance** or **security reasons**.

### **3. In-Scope Components**

* **Azure Load Balancer (Standard SKU)** – Fronts the backend pool.
* **Backend Virtual Machines or Services** – Hosting the private application.
* **Private Link Service Resource** – Created and associated with the Load Balancer frontend.
* **DNS Configuration** – Private DNS Zone or custom DNS forwarding.
* **Network Security Groups (NSGs) / Firewall Rules** – For traffic control.
* **ExpressRoute / VPN Gateway** – (Optional) for on-prem private connectivity.

### **4. Out-of-Scope**

* Azure Private Endpoints (consumer side)
* Services hosted in public Azure PaaS offerings (use Private Endpoint instead)

### **5. Design Considerations**

#### **✅ Benefits**

* **Private exposure**: No public IP needed to access the service.
* **Security isolation**: Access controlled using NSGs, Azure Firewall, and RBAC.
* **Data exfiltration protection**: Only approved consumers can connect.
* **Auditability**: Monitor traffic via NSG/Firewall logs or Azure Monitor.

#### **⚠️ Constraints**

* Only **Standard Load Balancer** is supported.
* Requires **manual approval** or whitelisting of each consumer connection.
* **DNS resolution** for PLS is not automatic—must configure manually.
* The consumer **must use Azure Private Endpoint** to connect.

#### **📌 Key Decisions**

* Choose between **automatic or manual approval** for consumer connections.
* Whether to **expose multiple services** behind a single PLS.
* Select **regions and availability zones** to deploy the service for redundancy.

### **6. Deployment Architecture**

plaintext

CopyEdit

[Consumer VNet/On-Prem]  
 |  
[Private Endpoint]  
 |  
[Azure Backbone]  
 |  
[Private Link Service]  
 |  
[Standard Load Balancer]  
 |  
[Backend Pool (VMs/AKS/App)]

### **7. Security and Governance**

* Use **Private DNS Zones** for name resolution.
* Apply **NSGs and Azure Firewall** to restrict access to backend.
* Enforce **RBAC** and **resource locks** to prevent accidental deletion.
* Enable **activity logs** and diagnostics for monitoring.

### **8. Dependencies**

* Azure Virtual Network and Subnet (for backend pool).
* Standard Load Balancer with frontend and health probes.
* DNS integration (Azure Private DNS or custom DNS forwarder).
* Approved network connectivity (e.g., ExpressRoute/PE).
* Permissions to create PLS (Contributor or Network Contributor).

### **9. Deliverables**

* Terraform or ARM template to deploy PLS architecture.
* DNS configuration guide for consumers.
* Connection request/approval workflow.
* Access logs and monitoring setup.
* Runbook for operations and troubleshooting.

### **10. Validation Criteria**

* Consumer can access backend via private IP without NAT.
* DNS resolution correctly points to Private Endpoint IP.
* No public ingress or egress occurs.
* Health probes show backend as healthy in the Load Balancer.

Great question. Here's a clear breakdown of **why NAT VMs and Load Balancers are used** with **Azure Private Link Service (PLS)** when connecting from **Azure Data Factory (ADF)** to **on-premises data assets**, and **how this works privately**—step by step.

## **🔍 Why NAT VMs + Load Balancer for Private Link Service (PLS)**

### **✅ Use Case Context**

You want **Azure Data Factory (ADF)** (hosted in Azure) to **privately access on-premises data assets** (like SQL Server, Oracle, file share, etc.) **over a private link**, instead of via public IPs or VPN connections.

### **🚧 Problem**

* **ADF is a Microsoft-managed service** and runs in a multi-tenant environment.
* ADF **can only reach private networks using Private Endpoints**, but it **cannot directly connect to on-prem networks** without a **bridge**.
* **Private Link Service (PLS)** helps expose **on-prem resources** to Azure **via Azure's backbone network**, but it expects a **backend reachable via Azure Load Balancer**.

### **💡 Solution: Use NAT VMs + Standard Load Balancer + PLS**

#### **🧱 Components:**

1. **NAT VMs** (or Jumpbox VMs):
   1. Have **private IPs** in your Azure VNet.
   2. Can establish outbound connections to **on-prem data assets** via **ExpressRoute or VPN**.
   3. Act as a **proxy or relay** for traffic coming from Azure PLS.
2. **Standard Load Balancer** (SLB):
   1. Fronts the NAT VMs and enables ADF to connect to a **single IP/port interface**.
   2. Required by Azure Private Link Service—it cannot point directly to a VM without a Load Balancer.
   3. Handles **inbound traffic** from Private Endpoint to NAT VMs.
3. **Azure Private Link Service (PLS)**:
   1. Exposes the SLB frontend IP **as a private IP** over the Azure backbone.
   2. **ADF creates a Private Endpoint** to connect to this PLS.

### **🔄 How It Works Privately: Step-by-Step**

plaintext

CopyEdit

[ADF (Azure)]  
 |  
 [Private Endpoint (PE)]  
 |  
 [Azure Backbone Network]  
 |  
 [Private Link Service (PLS)]  
 |  
 [Standard Load Balancer (SLB)]  
 |  
 [NAT VM(s)]  
 |  
 [ExpressRoute or VPN Gateway]  
 |  
 [On-prem Data Assets]

#### **🔁 Request Flow**

1. **ADF Integration Runtime** uses **Private Endpoint** to connect to the PLS in your VNet.
2. PLS routes the connection to the **SLB frontend IP**.
3. SLB load balances the request to one of the backend **NAT VMs**.
4. NAT VM uses **on-prem network connectivity (via ExpressRoute or VPN)** to reach the actual **on-prem data source**.
5. Data flows back via the same path—**all private, no public internet**.

## **🔒 How It's Private and Secure**

* **No public IPs** are involved anywhere.
* **Private Endpoint** connects securely over Azure’s private backbone, isolated from internet traffic.
* **NSGs and UDRs** can be used to lock down access at every hop.
* **PLS requires explicit approval** for PE connections—no accidental exposure.
* **DNS resolution** for the service happens via **Azure Private DNS Zone**, ensuring routing stays private.

## **📌 Summary**

|  |  |
| --- | --- |
| **Component** | **Role** |
| NAT VM | Relay traffic from Azure to on-prem over private network |
| Standard LB | Required by PLS; load balances incoming connections to NAT VMs |
| Private Link Service | Exposes NAT VM network access as a private service to ADF |
| Private Endpoint | Lets ADF access the service securely over Azure backbone |
| ExpressRoute/VPN | Enables private connection between Azure VNet and on-prem network |

Let me know if you'd like:

* A **Terraform or Bicep template**
* A **Visio/diagram**
* Or a **PowerShell/ADF linked service example** for the setup

##### **You said:**

terraform template

###### **ChatGPT said:**

Here's a **Terraform template** to deploy an **Azure Private Link Service (PLS)** using:

* Standard Load Balancer
* NAT VM backend
* Network configuration (VNet, subnet, NSGs)
* Output for consumer Private Endpoint (to be configured separately)

This is a **producer-side setup**—you expose your **on-prem route via NAT VMs** to consumers like **Azure Data Factory**, who connect via **Private Endpoint**.

## **🧱 Terraform Template: Azure Private Link Service with NAT VM**

hcl

CopyEdit

provider "azurerm" {  
 features {}  
}  
  
resource "azurerm\_resource\_group" "rg" {  
 name = "rg-pls-demo"  
 location = "East US"  
}  
  
resource "azurerm\_virtual\_network" "vnet" {  
 name = "vnet-pls"  
 address\_space = ["10.10.0.0/16"]  
 location = azurerm\_resource\_group.rg.location  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
}  
  
resource "azurerm\_subnet" "subnet" {  
 name = "subnet-pls"  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
 virtual\_network\_name = azurerm\_virtual\_network.vnet.name  
 address\_prefixes = ["10.10.1.0/24"]  
}  
  
resource "azurerm\_network\_interface" "nic\_natvm" {  
 name = "nic-natvm"  
 location = azurerm\_resource\_group.rg.location  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
  
 ip\_configuration {  
 name = "internal"  
 subnet\_id = azurerm\_subnet.subnet.id  
 private\_ip\_address\_allocation = "Dynamic"  
 }  
}  
  
resource "azurerm\_network\_security\_group" "nsg" {  
 name = "nsg-pls"  
 location = azurerm\_resource\_group.rg.location  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
}  
  
resource "azurerm\_network\_interface\_security\_group\_association" "nsg\_association" {  
 network\_interface\_id = azurerm\_network\_interface.nic\_natvm.id  
 network\_security\_group\_id = azurerm\_network\_security\_group.nsg.id  
}  
  
resource "azurerm\_windows\_virtual\_machine" "natvm" {  
 name = "natvm"  
 location = azurerm\_resource\_group.rg.location  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
 size = "Standard\_B2s"  
 admin\_username = "azureuser"  
 admin\_password = "Password123!"  
 network\_interface\_ids = [  
 azurerm\_network\_interface.nic\_natvm.id  
 ]  
  
 os\_disk {  
 caching = "ReadWrite"  
 storage\_account\_type = "Standard\_LRS"  
 }  
  
 source\_image\_reference {  
 publisher = "MicrosoftWindowsServer"  
 offer = "WindowsServer"  
 sku = "2019-Datacenter"  
 version = "latest"  
 }  
}  
  
resource "azurerm\_lb" "lb" {  
 name = "pls-lb"  
 location = azurerm\_resource\_group.rg.location  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
 sku = "Standard"  
  
 frontend\_ip\_configuration {  
 name = "LoadBalancerFrontEnd"  
 subnet\_id = azurerm\_subnet.subnet.id  
 private\_ip\_address\_allocation = "Dynamic"  
 }  
}  
  
resource "azurerm\_lb\_backend\_address\_pool" "bepool" {  
 name = "pls-be-pool"  
 loadbalancer\_id = azurerm\_lb.lb.id  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
}  
  
resource "azurerm\_lb\_probe" "healthprobe" {  
 name = "tcp-healthprobe"  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
 loadbalancer\_id = azurerm\_lb.lb.id  
 protocol = "Tcp"  
 port = 3389  
}  
  
resource "azurerm\_lb\_rule" "lbrule" {  
 name = "pls-lb-rule"  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
 loadbalancer\_id = azurerm\_lb.lb.id  
 protocol = "Tcp"  
 frontend\_port = 3389  
 backend\_port = 3389  
 frontend\_ip\_configuration\_name = "LoadBalancerFrontEnd"  
 backend\_address\_pool\_id = azurerm\_lb\_backend\_address\_pool.bepool.id  
 probe\_id = azurerm\_lb\_probe.healthprobe.id  
}  
  
resource "azurerm\_lb\_backend\_address\_pool\_address" "natvm\_backend" {  
 name = "natvm-be"  
 backend\_address\_pool\_id = azurerm\_lb\_backend\_address\_pool.bepool.id  
 virtual\_network\_id = azurerm\_virtual\_network.vnet.id  
 ip\_address = azurerm\_network\_interface.nic\_natvm.private\_ip\_address  
}  
  
resource "azurerm\_private\_link\_service" "pls" {  
 name = "pls-natvm"  
 location = azurerm\_resource\_group.rg.location  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
  
 load\_balancer\_frontend\_ip\_configuration\_ids = [  
 azurerm\_lb.lb.frontend\_ip\_configuration[0].id  
 ]  
  
 nat\_ip\_configuration {  
 name = "nat-ip-1"  
 subnet\_id = azurerm\_subnet.subnet.id  
 primary = true  
 private\_ip\_address\_version = "IPv4"  
 }  
  
 visibility {  
 subscriptions = ["<consumer-subscription-id>"] # or "\*" for all if open  
 }  
  
 auto\_approval {  
 subscriptions = ["<consumer-subscription-id>"]  
 }  
  
 enable\_proxy\_protocol = false  
}  
  
output "pls\_id" {  
 value = azurerm\_private\_link\_service.pls.id  
}

## **🧾 Notes:**

* Replace <consumer-subscription-id> with the actual ADF consumer's subscription.
* You may adjust the **TCP port** (3389) depending on the **protocol you're relaying** (SQL, SFTP, etc.).
* NAT VM should be configured with **route tables and/or a proxy forwarder** (e.g., NGINX, HAProxy, or custom TCP relay script) to access on-prem.
* You must have **ExpressRoute or VPN Gateway** configured in your actual setup (not shown here).

## **➕ Optional Add-ons**

Let me know if you need:

* A companion **consumer-side Private Endpoint Terraform**
* NAT VM **proxy forwarding script** (Python, PowerShell, or HAProxy)
* DNS configuration to support name resolution for ADF

Would you like this exported in a .tf file or zipped with modules?

##### **You said:**

zipped with modules

###### **ChatGPT said:**

Your Azure Private Link Service Terraform package is ready for download:

📦 Download ZIP – azure\_pls\_terraform\_package.zip

This includes:

* main.tf (entry point)
* Modular folders for network, nat\_vm, and pls

Let me know if you need help deploying it, generating variables/terraform.tfvars, or adding consumer-side Private Endpoint templates.

##### **You said:**

yes, both

###### **ChatGPT said:**

Your updated Terraform package now includes:

* main.tf
* terraform.tfvars
* Modular code for:
  + network
  + nat\_vm
  + pls

📦 Download Updated ZIP – azure\_pls\_terraform\_package.zip

Let me know if you also want:

* The **consumer-side Private Endpoint Terraform**
* A sample **proxy script** for the NAT VM to forward traffic to on-prem resources