Thought for 3m 50s

Gotcha—here’s a clean, end-to-end way to connect Azure Data Factory (ADF) to an HDFS cluster privately using a **Managed Private Endpoint (MPE)** to a **Private Link Service (PLS)**. This pattern works whether HDFS runs on Azure VMs or on-prem (reachable via VPN/ExpressRoute).

**What you’ll build (at a glance)**

ADF (Managed VNet IR)

│

└─► Managed Private Endpoint ──► Private Link Service ──► \*Internal\* Standard Load Balancer

└─► HDFS NameNode (WebHDFS TCP : 9870 or 50070)

(via direct backend or via a tiny “forwarder” VM to on-prem)

* ADF talks only over private IPs inside its **managed VNet**.
* PLS fronts an internal **Standard** Load Balancer that sends TCP to your WebHDFS endpoint. (PLS requires Standard LB.) [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)
* ADF’s HDFS connector uses **WebHDFS** (HTTP/HTTPS). Typical ports: Hadoop 3.x → 9870, older Hadoop 2.x → 50070. Use the WebHDFS URL in the linked service. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)
* You’ll approve the MPE request on the PLS side before traffic flows. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)

**Prereqs**

* An HDFS cluster with **WebHDFS** enabled (HTTP/HTTPS). Default WebHDFS ports are **9870** (Hadoop 3) or **50070** (Hadoop 2). If using Kerberos/WebHDFS over HTTPS, ensure TLS is set up as usual. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)
* If HDFS is **on-prem**, set up **VPN/ExpressRoute** from an Azure VNet (“provider VNet”) to your on-prem network.
* An ADF instance where you can enable **Managed Virtual Network** + **Managed Private Endpoints**. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)

**Step 1 — Expose WebHDFS behind an *internal* Standard Load Balancer**

1. In your **provider VNet**, create a **Standard Internal Load Balancer** (ILB).
2. Backend choices:
   * **HDFS on Azure VMs**: put the **NameNode VM(s)** in the backend pool, open WebHDFS on the VM(s).
   * **HDFS on-prem**: add a tiny Linux “**forwarder**” VM to the backend pool. It will DNAT traffic to your on-prem NameNode over your VPN/ER.
3. Create:
   * **Health probe** on the WebHDFS TCP port (9870 or 50070).
   * **LB rule**: Frontend :9870 → Backend :9870 (or :50070 if Hadoop 2).

**CLI snippets (examples)**

# Create internal Standard LB

az network lb create \

-g rg-net \

-n hdfs-ilb \

--sku Standard \

--vnet-name vnet-provider \

--subnet subnet-pls \

--frontend-ip-name fe \

--backend-pool-name be \

--private-ip-address 10.10.1.10 \

--frontend-ip-zone 1 2 3

# Health probe on 9870

az network lb probe create \

-g rg-net \

--lb-name hdfs-ilb \

-n hp-9870 \

--protocol Tcp \

--port 9870

# LB rule 9870->9870

az network lb rule create \

-g rg-net \

--lb-name hdfs-ilb \

-n rule-9870 \

--protocol Tcp \

--frontend-port 9870 \

--backend-port 9870 \

--frontend-ip-name fe \

--backend-pool-name be \

--probe-name hp-9870

(Reference quickstarts and CLI commands for internal LBs, probes, and rules.) [Microsoft Learn+2Microsoft Learn+2](https://learn.microsoft.com/en-us/azure/load-balancer/quickstart-load-balancer-standard-internal-cli?utm_source=chatgpt.com)

**If HDFS is on-prem: forwarder VM (DNAT) example**

On the forwarder VM (Linux), enable IP forwarding and DNAT 9870 to your on-prem NameNode:

# enable IP forwarding

sudo sysctl -w net.ipv4.ip\_forward=1

sudo sh -c 'echo "net.ipv4.ip\_forward = 1" >> /etc/sysctl.conf'

# DNAT 9870 -> on-prem NameNode:9870

sudo iptables -t nat -A PREROUTING -p tcp --dport 9870 -j DNAT --to-destination 172.16.10.50:9870

sudo iptables -t nat -A POSTROUTING -p tcp -d 172.16.10.50 --dport 9870 -j MASQUERADE

This is the same pattern Microsoft shows when fronting an on-prem DB via PLS (we’re just using the WebHDFS port instead of SQL). [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/tutorial-managed-virtual-network-on-premise-sql-server)

**Step 2 — Create a Private Link Service (PLS) pointing at the ILB**

Create the PLS in the same subnet as the ILB (PLS requires a Standard LB frontend). Save its **service alias** (you’ll paste it in ADF). [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)

az network private-link-service create \

-g rg-net \

-n hdfs-pls \

--vnet-name vnet-provider \

--subnet subnet-pls \

--lb-name hdfs-ilb \

--lb-frontend-ip-configs fe \

-l eastus

# Get/share the PLS service alias for consumers (ADF will use this)

az network private-link-service show \

-g rg-net \

-n hdfs-pls \

--query "alias" -o tsv

(Official PLS CLI docs / quickstart for LB+PLS.) [Microsoft Learn+1](https://learn.microsoft.com/en-us/cli/azure/network/private-link-service?view=azure-cli-latest&utm_source=chatgpt.com)

Tip: Configure **auto-approval** for your subscription/tenant if you control both sides, or be ready to approve the request manually in **Private Link Center**. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/private-link/manage-private-endpoint?utm_source=chatgpt.com)

**Step 3 — Enable Managed VNet in ADF and create a Managed Private Endpoint**

1. In ADF Studio → **Manage** → **Integration runtimes** → create/enable **Managed Virtual Network** on your IR. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)
2. In ADF Studio → **Manage** → **Security** → **Managed private endpoints** → **+ New** → **Private Link Service**.
3. Paste the **service alias** from Step 2 and submit. The MPE will show **Pending**.
4. Go to the PLS (provider) and **approve** the connection. After approval, ADF will show **Approved** and the MPE will have a **private IP**. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)

DNS note (important): ADF Managed VNet doesn’t let you bring your own DNS resolver. The simplest approach is to use the **MPE private IP** directly in your HDFS linked service URL (next step). This avoids any custom DNS plumbing. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)

**Step 4 — Create the HDFS linked service (via WebHDFS over the MPE)**

In ADF Studio → **Manage** → **Linked services** → **HDFS**:

* **URL**: http://<MPE-PRIVATE-IP>:9870/webhdfs/v1/  
  (Use :50070 for Hadoop 2.x, or https://... if you front WebHDFS with TLS.)
* **Authentication**: choose what your cluster uses (Anonymous/Basic/Windows/Kerberos).
* **Connect via Integration Runtime**: choose your **Managed VNet IR**. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)

**Example (JSON)**

{

"name": "ls\_hdfs\_webhdfs",

"type": "LinkedService",

"properties": {

"type": "Hdfs",

"typeProperties": {

"url": "http://10.20.0.5:9870/webhdfs/v1/",

"authenticationType": "Anonymous"

},

"connectVia": { "referenceName": "AutoResolveIntegrationRuntime", "type": "IntegrationRuntimeReference" }

}

}

Test connection should succeed only **after** the MPE shows **Approved**. If you use Kerberos or Basic, supply the relevant properties/certificates supported by the HDFS connector. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)

**Step 5 — Use it in a pipeline (quick check)**

Create a **Copy activity**:

* Source: HDFS linked service, a path like /data/incoming/.
* Sink: Azure Blob/ADLS Gen2 linked service (private, ideally via its own MPE).  
  Run a pipeline and verify data moves over private IPs only. (ADF MPEs route traffic within Microsoft’s backbone.) [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)

**Variations & tips**

* **HDFS HA (2 NameNodes)**:  
  Put both NameNodes in the ILB backend pool. WebHDFS on the standby typically redirects; ensure your health probe targets a port that reflects “active” health or use a small reverse proxy that follows redirects. Keep the LB rule on the WebHDFS TCP port. (General pattern follows the same PLS → ILB approach.) [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)
* **HTTPS/TLS**:  
  If you terminate TLS on the NameNode(s), expose **:9871** (or your chosen port) on the ILB/PLS and set the ADF URL to https://<MPE-IP>:9871/webhdfs/v1/. The HDFS connector supports HTTPS. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)
* **On-prem routing**:  
  The forwarder VM approach is a Microsoft-documented pattern for PLS to on-prem (they show it for SQL; the idea is identical here with WebHDFS). Keep firewall rules open from the forwarder to your on-prem NameNode WebHDFS port. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/tutorial-managed-virtual-network-on-premise-sql-server)
* **Why not SHIR?**  
  You can avoid deploying a Self-Hosted IR by using **Managed VNet + MPE + PLS**—supported for WebHDFS/HDFS. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)
* **Security**:  
  Use NSGs on the ILB subnet; allow the health probe and WebHDFS port; least-privileged on the forwarder VM; restrict PLS visibility/auto-approve to your subscriptions. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)

**Quick checklist (troubleshooting)**

* PLS shows **Healthy** and bound to the **Standard ILB** frontend. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)
* MPE in ADF shows **Approved**. If not, approve in **Private Link Center**. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/private-link/manage-private-endpoint?utm_source=chatgpt.com)
* ILB **Health probe** is green (verify port 9870/50070 reachable from ILB to the backend). [Microsoft Learn](https://learn.microsoft.com/en-us/azure/private-link/create-private-link-service-cli?utm_source=chatgpt.com)
* From the **forwarder VM** (if used), curl http://<onprem-nn>:9870/webhdfs/v1/?op=LISTSTATUS succeeds.
* In ADF, the HDFS **linked service URL** uses **MPE private IP** and correct port/path. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)

**References**

* **ADF managed VNet & Managed Private Endpoints** (concepts, approvals). [Microsoft Learn+1](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)
* **Private Link Service** (requires Standard LB, creation steps). [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint)
* **HDFS connector (WebHDFS)**: protocol, ports, auth, URL shape. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/connector-hdfs)
* **CLI** for LB + probe + rule + PLS creation. [Microsoft Learn+3Microsoft Learn+3Microsoft Learn+3](https://learn.microsoft.com/en-us/azure/load-balancer/quickstart-load-balancer-standard-internal-cli?utm_source=chatgpt.com)
* **On-prem via PLS pattern** (tutorial uses SQL; same flow with your WebHDFS port). [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/tutorial-managed-virtual-network-on-premise-sql-server)

Got it—here’s a clear, step-by-step way to set up **FQDNs for a Managed Private Endpoint (MPE) that targets a Private Link Service (PLS)** so Azure Data Factory (ADF) can reach an **HDFS** endpoint privately by name (not IP). I’ll show both the **recommended single-hostname pattern** (via Knox/HTTPFS) and the **direct WebHDFS pattern** (multiple hostnames), plus working CLI/Bicep examples and the HDFS linked service JSON.

**What you’re wiring up (quick picture)**

* **Producer VNet**: Your HDFS entry point (ideally **Knox** or **HTTPFS**, or else NameNode/DataNodes for pure **WebHDFS**) sits behind an **Internal Load Balancer (ILB)**. A **Private Link Service (PLS)** fronts that ILB.
* **Consumer (ADF)**: You create an **MPE** *to that PLS* inside ADF’s **Managed VNet**. In that MPE you list the **FQDNs** ADF should map to the MPE’s private IP(s).  
  Azure’s ADF MPE supports specifying --fqdns so name lookups inside the Managed VNet resolve **to the Private Endpoint**, not the public IP. [Microsoft Learn+1](https://learn.microsoft.com/en-us/azure/templates/microsoft.datafactory/2018-06-01/factories/managedvirtualnetworks/managedprivateendpoints?utm_source=chatgpt.com)

In ADF Studio’s UI, when the target type is **Private Link Service**, there’s a field to add the **FQDN(s)** of your on-prem/partner host—this is explicitly documented (example shown for on-prem SQL, same idea for HDFS). [Azure Documentation](https://docs.azure.cn/en-us/data-factory/tutorial-managed-virtual-network-on-premise-sql-server?utm_source=chatgpt.com)

**Decide your HDFS access pattern (important)**

**Option A — Recommended (1 FQDN): Knox or HTTPFS gateway**

* You expose **one** hostname (e.g., hdfs-gw.contoso.corp) that ADF calls over HTTP(S).
* Your gateway talks to NameNodes/DataNodes on your side—ADF only needs that single FQDN.

**Option B — Direct WebHDFS (many FQDNs): NameNode + DataNodes**

* WebHDFS issues redirects to **DataNode** HTTP endpoints (client follows Location to a DataNode). That means ADF must resolve and reach **every host** it may be redirected to, not just the NameNode. [hadoop.apache.org](https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/WebHDFS.html)
* Practically, you must include **NameNode HA hostnames** (e.g., nn1, nn2) **and all DataNode hostnames** (e.g., dn1 … dnN) in the MPE’s FQDN list.

Port notes: older Hadoop defaults were **50070/50075**; newer Hadoop 3.x defaults are **9870 (NN HTTP)** and **9864 (DN HTTP)**. Pick the right one for your cluster.

**Step-by-step**

**1) Prereqs on the producer side (your VNet)**

1. Put your HDFS entry point (Knox/HTTPFS or NameNode) **behind a Standard ILB**.
2. Create a **Private Link Service** pointing to that ILB and expose TCP ports you need (e.g., 443/8443 for HTTPS via Knox, or 9870/9864 for WebHDFS).  
   *(PLS creation is a standard Azure Networking task; no ADF specifics here.)*

**2) Enable ADF Managed VNet (if not already)**

* In ADF Studio → **Manage** → **Managed private endpoints** → **Enable Managed Virtual Network**. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)

**3) Create the Managed Private Endpoint to your PLS and add FQDNs**

You can do this in the **UI** or with **Azure CLI**.

**UI (ADF Studio)**

* **Manage** → **Managed private endpoints** → **+ New** → Type: **Private Link Service** → pick your PLS (or paste alias/resource ID).
* In **Fully qualified domain names**, enter:
  + **Option A (gateway)**: hdfs-gw.contoso.corp
  + **Option B (WebHDFS)**: nn-vip.contoso.corp, nn1.contoso.corp, nn2.contoso.corp, dn1.contoso.corp, dn2.contoso.corp, …
* Create. Ask the PLS owner to **Approve** the request in their resource. (MPEs start in **Pending**; owner approves/rejects.) [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)

**Azure CLI (same result)**

# variables

SUB="<sub-id>"

RG="rg-data"

ADF="adf-prod"

MPE="mpe-hdfs-pls"

PLS\_ID="/subscriptions/<sub-id>/resourceGroups/rg-core/providers/Microsoft.Network/privateLinkServices/pls-hdfs"

# Option A: single-FQDN (Knox / HTTPFS)

az datafactory managed-private-endpoint create \

--factory-name $ADF --resource-group $RG --name $MPE \

--private-link-resource-id $PLS\_ID \

--group-id "privatelinkservice" \

--fqdns hdfs-gw.contoso.corp

# Option B: WebHDFS (multiple FQDNs: NN + all DNs)

az datafactory managed-private-endpoint create \

--factory-name $ADF --resource-group $RG --name $MPE \

--private-link-resource-id $PLS\_ID \

--group-id "privatelinkservice" \

--fqdns nn-vip.contoso.corp nn1.contoso.corp nn2.contoso.corp \

dn1.contoso.corp dn2.contoso.corp dn3.contoso.corp

--fqdns is the key flag that registers those hostnames so **DNS inside the Managed VNet** resolves them to the **MPE private IP**. (Exactly as the CLI docs show.)

ARM/Bicep also supports fqdns on the MPE resource:

resource mpe 'Microsoft.DataFactory/factories/managedVirtualNetworks/managedPrivateEndpoints@2018-06-01' = {

name: '${adf.name}/default/mpe-hdfs-pls'

properties: {

privateLinkResourceId: pls.id

groupId: 'privatelinkservice'

fqdns: [

'hdfs-gw.contoso.corp' // or nn/dn list for WebHDFS

]

}

}

(See the MPE schema—fqdns is a first-class property.) [Microsoft Learn](https://learn.microsoft.com/en-us/azure/templates/microsoft.datafactory/2018-06-01/factories/managedvirtualnetworks/managedprivateendpoints?utm_source=chatgpt.com)

**4) Approve the MPE**

* PLS owner opens the PLS → **Private endpoint connections** → Approve.
* Back in ADF Studio, the MPE status turns **Approved**. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)

**5) Create the HDFS linked service in ADF using your FQDN**

For **Knox/HTTPFS (Option A)** (one FQDN):

{

"name": "ls\_hdfs\_knox",

"properties": {

"type": "Hdfs",

"typeProperties": {

"url": "https://hdfs-gw.contoso.corp:8443", // or 443

"authenticationType": "Anonymous" // or Basic/Kerberos per your setup

},

"connectVia": { "referenceName": "AutoResolveIntegrationRuntime", "type": "IntegrationRuntimeReference" }

}

}

For **WebHDFS (Option B)** (many FQDNs—NN + DNs):

{

"name": "ls\_hdfs\_webhdfs",

"properties": {

"type": "Hdfs",

"typeProperties": {

"url": "http://nn-vip.contoso.corp:9870" // Hadoop 3.x default; use https if enabled

},

"connectVia": { "referenceName": "AutoResolveIntegrationRuntime", "type": "IntegrationRuntimeReference" }

}

}

The HDFS connector expects a WebHDFS/HTTP(S) endpoint; default ports in many docs are 50070/50075 (Hadoop 2.x), while Hadoop 3.x typically uses 9870 (NN) and 9864 (DN). Match whatever your cluster uses.

**Testing**

1. In ADF Studio → **Manage → Managed private endpoints** → select your MPE → **Test connection**.
2. In your HDFS linked service → **Test connection**.
   * If the MPE and FQDNs are correct, ADF resolves your hostname(s) to the MPE’s private IP and reaches the service privately. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)

**Tips & gotchas**

* **Prefer a single FQDN** (Knox/HTTPFS) so you don’t have to maintain a long FQDN list for DataNodes. WebHDFS clients follow redirects to DataNodes; include those hostnames if you choose direct WebHDFS. [hadoop.apache.org](https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/WebHDFS.html)
* **TLS/Certificates**: If you use HTTPS, make sure the certificate CN/SAN includes the **exact FQDN** you configured (e.g., hdfs-gw.contoso.corp), or ADF’s SSL validation may fail.
* **No custom DNS in ADF’s Managed VNet**: Listing FQDNs on the MPE is how ADF learns to resolve your private names to the endpoint IP inside its Managed VNet; you don’t link your own private DNS zones to ADF. (UI flow calling for FQDNs and the --fqdns flag are the supported mechanism.) [Azure Documentation](https://docs.azure.cn/en-us/data-factory/tutorial-managed-virtual-network-on-premise-sql-server?utm_source=chatgpt.com)
* **Approval lifecycle**: MPEs start **Pending** and require approval by the PLS owner; only after approval will name resolution/traffic work. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)

**Complete, copy-paste example (CLI)**

# 0) Inputs

SUB="<sub-id>"

RG="rg-data"

ADF="adf-prod"

PLS\_ID="/subscriptions/<sub-id>/resourceGroups/rg-core/providers/Microsoft.Network/privateLinkServices/pls-hdfs"

# 1) Ensure ADF’s Managed VNet is enabled (do this once in the UI)

# 2) Create MPE with a single FQDN (Knox gateway)

az datafactory managed-private-endpoint create \

--subscription $SUB --resource-group $RG --factory-name $ADF \

--name mpe-hdfs-knox \

--private-link-resource-id $PLS\_ID \

--group-id "privatelinkservice" \

--fqdns hdfs-gw.contoso.corp

# 3) (PLS owner approves the connection)

# 4) HDFS linked service URL points to that FQDN/port (8443 or 443)

# Paste the JSON shown above into your ADF linked service authoring pane.

(Use the multi-FQDN variant if you’re doing direct WebHDFS.)

**References**

* ADF Managed VNet & MPE overview and approval flow. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/data-factory/managed-virtual-network-private-endpoint?utm_source=chatgpt.com)
* Creating MPEs via CLI (supports --fqdns).
* MPE ARM/Bicep schema includes fqdns. [Microsoft Learn](https://learn.microsoft.com/en-us/azure/templates/microsoft.datafactory/2018-06-01/factories/managedvirtualnetworks/managedprivateendpoints?utm_source=chatgpt.com)
* UI flow: add **FQDN** when MPE targets a **Private Link Service**. [Azure Documentation](https://docs.azure.cn/en-us/data-factory/tutorial-managed-virtual-network-on-premise-sql-server?utm_source=chatgpt.com)
* HDFS connector basics & ports; Hadoop 3.x port defaults; WebHDFS redirect behavior to DataNodes.

Excellent question 👍 — this is one of the most **confusing** parts of using **Private Link Service (PLS)** with **on-prem resources**. Let’s break it down carefully.

**🔎 Does PLS to on-prem *always* need a NAT (forwarder) VM?**

**Yes, in most cases.**  
Because **PLS only fronts an Azure *internal* Standard Load Balancer (ILB)**. That ILB can only send traffic to backends inside an Azure VNet.

👉 So, if your **actual service endpoint is *not* in Azure** (e.g., an on-prem SQL/HDFS/Oracle server behind ExpressRoute or VPN), the ILB has no way to “directly” forward packets to that on-prem private IP. The trick is to add a **forwarder/NAT VM** in the ILB backend pool, and that VM then relays (NATs) the traffic to the on-prem target.

Microsoft shows this pattern explicitly in their tutorial *“Connect to on-premises SQL Server using Private Link Service”* — they deploy a **forwarder VM** that NATs from the ILB frontend to the SQL private IP on-prem.

**⚙️ Why the NAT VM is needed**

1. **PLS design restriction**:
   * A Private Link Service requires a **Standard Internal LB** frontend.
   * That LB can only point at NICs in Azure subnets, not arbitrary IPs on-prem.
2. **On-prem targets live outside Azure**:
   * Even if you have ExpressRoute/VPN, the ILB cannot directly target “remote” IPs as backend members.
   * The NAT VM acts as an Azure-local backend, so the ILB can forward traffic to *something*.
3. **The NAT VM rewrites traffic**:
   * Client (ADF, VM, AKS, etc.) → PLS → ILB frontend → NAT VM.
   * NAT VM DNATs/forwards packets → On-prem private IP over ExpressRoute/VPN.
   * Return traffic comes back the same way, NATed back.

**📘 Example: SQL Server on-prem via PLS**

**Without NAT VM (❌ Not possible)**

* On-prem SQL = 10.50.1.10:1433.
* You try to make an ILB rule with backend 10.50.1.10.
* Fails: ILB only supports Azure NICs, not arbitrary IPs.

**With NAT VM (✅ Works)**

1. Deploy a small Azure Linux VM (ForwarderVM) in the provider VNet.
2. Put that VM’s NIC into the ILB backend pool.
3. On the VM, enable IP forwarding and DNAT:

# enable IP forwarding

sudo sysctl -w net.ipv4.ip\_forward=1

sudo sh -c 'echo "net.ipv4.ip\_forward=1" >> /etc/sysctl.conf'

# DNAT port 1433 from ILB -> on-prem SQL

sudo iptables -t nat -A PREROUTING -p tcp --dport 1433 -j DNAT --to-destination 10.50.1.10:1433

sudo iptables -t nat -A POSTROUTING -p tcp -d 10.50.1.10 --dport 1433 -j MASQUERADE

1. Client (e.g., ADF Managed Private Endpoint) connects to sql-pls.privatelinkservice → hits ILB frontend:1433 → load-balanced to ForwarderVM:1433 → DNAT → on-prem SQL:1433.

**📘 Example: HDFS WebHDFS on-prem via PLS**

* On-prem HDFS NameNode = 172.16.10.50:9870.
* You create an ILB + PLS in Azure.
* Backend pool contains ForwarderVM.
* On the forwarder, DNAT port 9870 to 172.16.10.50:9870 over ExpressRoute/VPN.
* From ADF: http://hdfs-pls.privatelinkservice:9870/webhdfs/v1/ works.
* Return packets traverse ExpressRoute back to the Forwarder, NATed, and then back to ADF.

**⚖️ Are there cases where you don’t need NAT VM?**

Yes — **only if your service endpoint actually runs on Azure compute** (VM, App, SQL MI, etc.) and has an Azure NIC you can put in the ILB backend pool.

👉 For example:

* Azure SQL Managed Instance behind PLS = **no NAT VM needed**.
* HDFS cluster on Azure VMs = **no NAT VM needed** (ILB can directly backend pool the VMs’ NICs).
* HDFS cluster on-prem = **NAT VM required**, since ILB can’t send to on-prem NICs.

**✅ Summary**

* **If service is on Azure**: No NAT VM needed (ILB can point directly).
* **If service is on-prem via ExpressRoute/VPN**: You *must* use a **NAT/forwarder VM** in Azure.

This NAT VM acts as a bridge: it sits in the ILB backend, receives traffic from PLS, and forwards it over private connectivity to your on-prem system. That’s the only way today because **ILBs cannot have non-Azure IPs as backends**.

Do you want me to also draw you a **simple ASCII diagram** showing the traffic flow (Client → PLS → ILB → Forwarder → On-prem), so it’s visually clear why the forwarder is needed?

**You said:**

write me the IP forwarding shell script for the linux VNAT vm for HDFS on-prem cluster to use HDFS gateway for managed private endpoint for data factory

**ChatGPT said:**

Perfect — let’s write a clean **Linux forwarder (VNAT) shell script** that will run on your **Azure VM** (the one sitting in the ILB backend pool). This VM will forward traffic from Azure **Private Link Service (PLS)** to your **on-prem HDFS Gateway** (Knox/HTTPFS).

That way, when ADF connects through its **Managed Private Endpoint (MPE)**, it will reach the HDFS gateway via private IP, while the VM transparently forwards the traffic.

**📝 Script: setup\_hdfs\_forwarder.sh**

#!/bin/bash

#

# setup\_hdfs\_forwarder.sh

#

# Configure a Linux VM as a VNAT forwarder for Azure Private Link Service

# to reach an on-prem HDFS Gateway (Knox/HTTPFS or WebHDFS).

#

# Usage:

# sudo ./setup\_hdfs\_forwarder.sh <ONPREM\_GATEWAY\_IP> <GATEWAY\_PORT>

#

# Example:

# sudo ./setup\_hdfs\_forwarder.sh 172.16.10.50 8443

# -> forwards ILB:8443 -> 172.16.10.50:8443

set -e

if [ $# -ne 2 ]; then

echo "Usage: $0 <ONPREM\_GATEWAY\_IP> <GATEWAY\_PORT>"

exit 1

fi

ONPREM\_IP=$1

ONPREM\_PORT=$2

echo "[INFO] Enabling IP forwarding..."

sysctl -w net.ipv4.ip\_forward=1

if ! grep -q "net.ipv4.ip\_forward" /etc/sysctl.conf; then

echo "net.ipv4.ip\_forward=1" >> /etc/sysctl.conf

else

sed -i 's/^net.ipv4.ip\_forward.\*/net.ipv4.ip\_forward=1/' /etc/sysctl.conf

fi

echo "[INFO] Flushing existing NAT rules..."

iptables -t nat -F

echo "[INFO] Adding DNAT and SNAT rules for $ONPREM\_IP:$ONPREM\_PORT..."

# DNAT: redirect inbound traffic on the given port to the on-prem HDFS gateway

iptables -t nat -A PREROUTING -p tcp --dport $ONPREM\_PORT -j DNAT --to-destination ${ONPREM\_IP}:${ONPREM\_PORT}

# SNAT/MASQUERADE: ensure return packets go back via this VM

iptables -t nat -A POSTROUTING -p tcp -d $ONPREM\_IP --dport $ONPREM\_PORT -j MASQUERADE

echo "[INFO] Persisting iptables rules..."

# For Ubuntu/Debian:

if command -v netfilter-persistent >/dev/null 2>&1; then

netfilter-persistent save

# For RHEL/CentOS:

elif command -v service >/dev/null 2>&1 && systemctl is-active firewalld >/dev/null 2>&1; then

firewall-cmd --permanent --direct --passthrough ipv4 -t nat -A PREROUTING -p tcp --dport $ONPREM\_PORT -j DNAT --to-destination ${ONPREM\_IP}:${ONPREM\_PORT}

firewall-cmd --permanent --direct --passthrough ipv4 -t nat -A POSTROUTING -p tcp -d $ONPREM\_IP --dport $ONPREM\_PORT -j MASQUERADE

firewall-cmd --reload

else

echo "[WARN] Could not persist iptables rules automatically. Please persist manually."

fi

echo "[SUCCESS] VNAT forwarding configured."

echo "Traffic arriving on this VM's NIC:$ONPREM\_PORT will be forwarded to $ONPREM\_IP:$ONPREM\_PORT"

**🔧 How it works**

* **Enable IP forwarding**: lets the VM act as a router.
* **PREROUTING (DNAT)**: any traffic arriving at the VM on port GATEWAY\_PORT gets rewritten to the on-prem HDFS Gateway (ONPREM\_IP:GATEWAY\_PORT).
* **POSTROUTING (MASQUERADE)**: rewrites source IP so return packets flow back via the forwarder VM.
* **Persistence**: uses netfilter-persistent (Ubuntu/Debian) or firewalld passthrough (RHEL/CentOS).

**🔗 Example for HDFS Gateway**

If your **on-prem HDFS Knox/HTTPFS gateway** runs on 172.16.10.50:8443:

scp setup\_hdfs\_forwarder.sh azureuser@forwardervm:/home/azureuser/

ssh azureuser@forwardervm

sudo ./setup\_hdfs\_forwarder.sh 172.16.10.50 8443

* Now, any traffic hitting the **ILB frontend (port 8443)** → goes to **Forwarder VM** → NATed to 172.16.10.50:8443.
* ADF MPE can be configured with FQDN hdfs-gw.contoso.corp:8443, resolving to the PLS/MPE private IP.

**⚠️ Notes & Best Practices**

* **One port per rule**: If you need both HTTP (9870) and HTTPS (8443), run the script twice or extend it.
* **Security**: Lock the VM subnet with NSGs so only the ILB/PLS can send traffic.
* **HA**: For production, deploy at least two forwarder VMs in the ILB backend pool for redundancy.
* **Monitoring**: Collect syslog/iptables logs, and add Azure Monitor alerts if forwarding breaks.

👉 Do you want me to extend this script so it can **support multiple HDFS ports at once** (e.g., 9870 for WebHDFS + 8443 for Knox), or keep it **single-port per run** for simplicity?

**You said:**

yes for 9870 for wbhdfs+8443

**ChatGPT said:**

Great 👍 let’s extend the VNAT forwarder so it can handle **both WebHDFS (9870)** and **Knox/HTTPS (8443)** at once.

This way, your **ADF Managed Private Endpoint (MPE)** to **Private Link Service (PLS)** can transparently reach both services without needing to re-run the script.

**📝 Script: setup\_hdfs\_forwarder\_multi.sh**

#!/bin/bash

#

# setup\_hdfs\_forwarder\_multi.sh

#

# Configure a Linux VM as a VNAT forwarder for Azure Private Link Service

# to reach an on-prem HDFS Gateway (Knox/HTTPFS and WebHDFS).

#

# Usage:

# sudo ./setup\_hdfs\_forwarder\_multi.sh <ONPREM\_GATEWAY\_IP>

#

# Example:

# sudo ./setup\_hdfs\_forwarder\_multi.sh 172.16.10.50

# -> forwards ILB:8443 -> 172.16.10.50:8443 (Knox/HTTPS)

# -> forwards ILB:9870 -> 172.16.10.50:9870 (WebHDFS)

set -e

if [ $# -ne 1 ]; then

echo "Usage: $0 <ONPREM\_GATEWAY\_IP>"

exit 1

fi

ONPREM\_IP=$1

PORTS=(8443 9870) # Knox + WebHDFS

echo "[INFO] Enabling IP forwarding..."

sysctl -w net.ipv4.ip\_forward=1

if ! grep -q "net.ipv4.ip\_forward" /etc/sysctl.conf; then

echo "net.ipv4.ip\_forward=1" >> /etc/sysctl.conf

else

sed -i 's/^net.ipv4.ip\_forward.\*/net.ipv4.ip\_forward=1/' /etc/sysctl.conf

fi

echo "[INFO] Flushing existing NAT rules..."

iptables -t nat -F

for PORT in "${PORTS[@]}"; do

echo "[INFO] Adding DNAT and SNAT rules for $ONPREM\_IP:$PORT..."

# DNAT: redirect inbound traffic on this VM port -> on-prem gateway

iptables -t nat -A PREROUTING -p tcp --dport $PORT -j DNAT --to-destination ${ONPREM\_IP}:${PORT}

# SNAT/MASQUERADE: ensure return packets go back via this VM

iptables -t nat -A POSTROUTING -p tcp -d $ONPREM\_IP --dport $PORT -j MASQUERADE

done

echo "[INFO] Persisting iptables rules..."

# Ubuntu/Debian

if command -v netfilter-persistent >/dev/null 2>&1; then

netfilter-persistent save

# RHEL/CentOS with firewalld

elif command -v firewall-cmd >/dev/null 2>&1; then

for PORT in "${PORTS[@]}"; do

firewall-cmd --permanent --direct --passthrough ipv4 -t nat -A PREROUTING -p tcp --dport $PORT -j DNAT --to-destination ${ONPREM\_IP}:${PORT}

firewall-cmd --permanent --direct --passthrough ipv4 -t nat -A POSTROUTING -p tcp -d $ONPREM\_IP --dport $PORT -j MASQUERADE

done

firewall-cmd --reload

else

echo "[WARN] Could not persist iptables rules automatically. Please persist manually."

fi

echo "[SUCCESS] VNAT forwarding configured."

echo "Traffic arriving on this VM will be forwarded to $ONPREM\_IP on ports: ${PORTS[\*]}"

**🔗 Example Usage**

If your on-prem HDFS Gateway runs on 172.16.10.50:

scp setup\_hdfs\_forwarder\_multi.sh azureuser@forwardervm:/home/azureuser/

ssh azureuser@forwardervm

sudo ./setup\_hdfs\_forwarder\_multi.sh 172.16.10.50

Now:

* ILB:8443 → **Knox Gateway** (172.16.10.50:8443)
* ILB:9870 → **WebHDFS/NameNode UI** (172.16.10.50:9870)

Your **ADF Managed Private Endpoint** can use the **FQDN** (e.g., hdfs-gw.contoso.corp) mapped to the **PLS private IP**, and depending on the service it connects to, the VNAT forwards appropriately.

**⚠️ Extra Notes**

* If you also expose **HDFS DataNodes (50070, 50075, etc.)** via gateway, you can add those ports to the PORTS=(...) list.
* For **high availability**, run this on **two VNAT VMs** in the ILB backend pool.
* Make sure your **on-prem firewall** allows incoming traffic from this VNAT VM’s private IP.
* You can test with curl -vk https://<FQDN>:8443/gateway/default/webhdfs/v1/?op=LISTSTATUS