PRJ-NET-038: Secure Service Exposure with AWS PrivateLink

Certification: AWS Certified Advanced Networking – Specialty
Domain: Network Security and Private Connectivity

---

1. Project Overview

This project demonstrates how to use **AWS PrivateLink** to securely expose a service running in one VPC (the "Service Provider") to consumers in another VPC (the "Service Consumer") without traversing the public internet. PrivateLink provides private, unidirectional connectivity from consumer VPCs to services, powered by an architecture that does not require VPC peering, internet gateways, NAT gateways, or Transit Gateway.

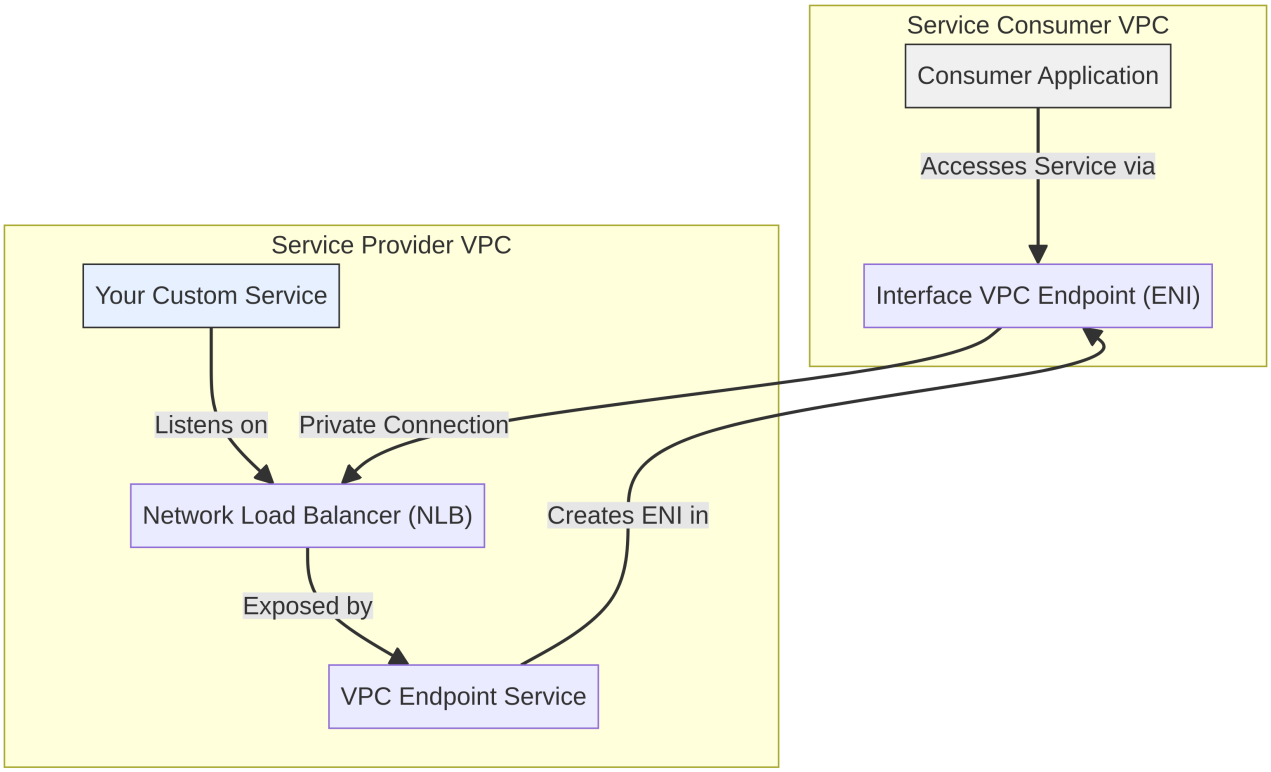
This is the same technology that AWS uses to privately expose its own services (like S3, DynamoDB, and Kinesis) inside your VPC. We will build a custom service, expose it using a **VPC Endpoint Service**, and then consume it from a different VPC by creating an **Interface VPC Endpoint**. This pattern is ideal for SaaS providers or large enterprises that need to offer services to customers or other internal teams in a secure and scalable way, without the complexity of network routing.

Key Objectives

- Understand the provider/consumer model of AWS PrivateLink.
- Deploy a **Network Load Balancer (NLB)** in front of a custom service.
- Create a **VPC Endpoint Service** to expose the NLB to other accounts.
- Create an **Interface VPC Endpoint** in a consumer VPC to connect to the service.
- Verify that the consumer application can access the provider service privately using the endpoint's DNS name.
- Manage access control for the endpoint service using a whitelist.

2. Architecture

The architecture creates a secure, private connection between a service provider and a service consumer, even if they are in different AWS accounts.



Architectural Components:

1. **Service Provider VPC:**
    - This VPC hosts the custom application or service (e.g., a fleet of EC2 instances running a REST API).
    - A **Network Load Balancer (NLB)** is placed in front of the service. AWS PrivateLink requires the use of an NLB.
    - A **VPC Endpoint Service** is created and configured to point to the NLB. This service is what gets exposed to consumers. It has a unique service name (e.g., `com.amazonaws.vpce.us-east-1.vpce-svc-xxxxxxxxxxxxxxxxx`).
2. **Service Consumer VPC:**
    - This VPC contains the application that needs to consume the service.

- The consumer creates an **Interface VPC Endpoint** for the provider's service. This action creates one or more **Elastic Network Interfaces (ENIs)** inside the consumer's own VPC subnets.
- These ENIs get private IP addresses from the consumer's VPC CIDR range.

3. **Private Connectivity:**
   - When the consumer application sends traffic to the DNS name of the Interface VPC Endpoint, it resolves to the private IP address of the ENI within its own VPC.
   - The traffic is then securely and privately routed by the AWS backbone from the consumer's ENI directly to the provider's Network Load Balancer, and then to the underlying service.
   - Crucially, the traffic **never leaves the AWS network**. The IP addresses of the provider and consumer VPCs can even overlap, as there is no direct network routing between them.

---

3. Prerequisites

- An AWS account with administrative permissions. Using two separate accounts (one for the provider, one for the consumer) is recommended to fully simulate the real-world use case.
- Two VPCs, which can be in the same or different accounts.

---

4. Step-by-Step Implementation Guide

Step 4.1: Set Up the Service Provider VPC

1. **Launch a Service:**
   - In the **Provider VPC**, launch an EC2 instance. Install a simple web server on it (e.g., Nginx) that listens on port 80.
   - This will simulate your custom service.

2. **Create a Network Load Balancer (NLB):**
    - Go to the **EC2 Console -> Load Balancers -> Create Load Balancer**.
    - Choose **Network Load Balancer**.
    - **Name:** `my-service-nlb`
    - **Scheme:** Internal
    - **VPC:** Select the Provider VPC.
    - **Listener:** TCP, Port 80.
    - **Target Group:** Create a new target group that includes the EC2 instance you launched.
    - Create the NLB.
3. **Create a VPC Endpoint Service:**
    - Go to the **VPC Console -> Endpoint Services -> Create endpoint service**.
    - **Name:** `my-custom-app-service`
    - **Load balancer type:** Network
    - **Available load balancers:** Select the `my-service-nlb` you just created.
    - **Acceptance required:** Check this box. This means you must manually approve connection requests from consumers.
    - Create the service. Once created, note down its **Service name**.

Step 4.2: Set Up the Service Consumer VPC

1. **(If using a different account) Whitelist the Consumer Account:**
    - In the **Provider Account**, select the endpoint service you created.
    - Go to the **Allow principals** tab and add the ARN of the consumer AWS account (e.g., `arn:aws:iam::123456789012:root`).
2. **Create an Interface VPC Endpoint:**
    - Log in to the **Consumer Account** (or stay in the same account if not using two).
    - Go to the **VPC Console -> Endpoints -> Create endpoint**.

- **Name:** `my-app-endpoint`
- **Service category:** Find service by name.
- **Service name:** Paste the service name you copied from the provider's endpoint service.
- Click **Verify service**.
- **VPC:** Select the **Consumer VPC**.
- **Subnets:** Choose the subnets where you want to create ENIs for the endpoint.
- **Security Group:** Select a security group that allows outbound traffic on TCP port 80 to the provider service.
- Create the endpoint.

Step 4.3: Accept the Connection and Test

1. **Accept the Connection Request:**
   - Switch back to the **Provider Account**.
   - Go to the **Endpoint Services** console and select your service.
   - In the **Endpoint connections** tab, you will see a connection request from the consumer.
   - Select it and choose **Actions -> Accept endpoint connection request**.
2. **Test the Connection:**
   - In the **Consumer Account**, go to the **Endpoints** console and select the endpoint you created.
   - In the **Details** tab, copy one of the **DNS names** for the endpoint.
   - Launch an EC2 instance in the **Consumer VPC**.
   - SSH into this instance.
   - Use `curl` to access the service using the endpoint's DNS name:

```
curl <endpoint_dns_name>
```

    - You should receive the default Nginx welcome page, served from the EC2 instance in the **Provider VPC**. The connection was made entirely over the AWS private network.

---

5. Key Concepts and Considerations

- **Unidirectional Access:** PrivateLink is unidirectional. The consumer can initiate connections to the provider, but the provider cannot initiate connections to the consumer.
- **No IP Overlap Issues:** Because there is no routing involved, the CIDR ranges of the provider and consumer VPCs can overlap without any issues.
- **Scalability:** The architecture is highly scalable. The provider can add more instances behind their NLB, and the consumer can create endpoints in many VPCs, all without any network configuration changes.
- **Security:** Traffic is encrypted in transit within the AWS network, and you can control access at multiple levels: NLB security groups, endpoint policies, and whitelisting principals on the endpoint service.

---

6. Cleanup

1. In the **Consumer Account**, delete the **Interface VPC Endpoint**.
2. In the **Provider Account**, delete the **VPC Endpoint Service**.
3. Delete the **Network Load Balancer**.
4. Terminate the EC2 instances in both the provider and consumer VPCs.

Business Context

The Problem

Cloud networks are complex and vulnerable to misconfigurations that expose resources to the internet. Organizations lack visibility into network traffic and struggle

to implement proper network segmentation. Manual network management leads to security gaps and compliance violations.

The Solution

Secure network architecture with automated traffic inspection, network segmentation, and centralized management. Implements VPC design best practices, transit gateway for multi-VPC connectivity, and network firewall for deep packet inspection. Provides complete network visibility and control.

Business Value

- **Zero Trust Architecture:** Micro-segmentation prevents lateral movement of threats
- **Network Visibility:** Complete traffic logging and analysis for security and troubleshooting
- **Centralized Management:** Single pane of glass for multi-account network control
- **Compliance Ready:** Network logs and flow records for audit requirements

Risk Mitigation

Prevents unauthorized network access, blocks malicious traffic, contains breaches through segmentation, and ensures compliance with network security requirements.

GRC Mapping

Compliance Frameworks

- **NIST CSF:** PR.AC-5 (Network segregation), PR.PT-4 (Network protection), DE.CM-1 (Network monitoring)
- **ISO 27001:** A.13.1 (Network security management), A.13.2 (Information transfer)
- **CIS Controls:** Control 12 (Network Infrastructure Management), Control 13 (Network Monitoring)
- **Zero Trust Architecture:** NIST SP 800-207

Security Controls Implemented

- Network segmentation and micro-segmentation
- Stateful firewall and deep packet inspection
- VPC flow logs and traffic analysis
- Network access control lists (NACLs)
- Private connectivity (VPN/Direct Connect)

Audit Evidence

- VPC flow logs and network traffic records
- Firewall rules and policy configurations
- Network topology diagrams
- Security group and NACL configurations

Regulatory Alignment

- **PCI DSS:** Requirement 1 (Firewall configuration), Requirement 2 (Network segmentation)
- **HIPAA:** § 164.312(e) (Transmission security)
- **GDPR:** Article 32 (Network security measures)
- **SOC 2:** CC6.6 (Logical access - network security)