# PRJ-SAP-014: Hybrid Cloud Connectivity — Lab Simulation Approach

**Certification:** AWS Certified Solutions Architect – Professional**Domain:** Designing for Hybrid Environments**Author:** Mo**Template Version:** v9

## 1. Business Context and GRC Mapping

For modern enterprises, transitioning to the cloud is rarely an overnight process. A hybrid cloud model—where on-premises data centers securely connect to cloud environments—is a strategic reality for most large organizations.

While AWS Direct Connect (DX) is the enterprise standard for this, **provisioning physical fiber in a lab environment is not feasible**. This project demonstrates how to simulate a full hybrid cloud environment using a secondary AWS VPC acting as the "on-premises" data center. We establish secure, scalable, and highly available network connectivity using AWS Site-to-Site VPN and AWS Transit Gateway, fully replicating the routing behaviors of a real enterprise setup.

### Standardized GRC Mapping

| GRC Category | Control/Objective | Implementation Details |
|---|---|---|
| **Governance** | Network Architecture Standardization | AWS Transit Gateway provides a centralized, hub-and-spoke routing architecture, simplifying network management and ensuring consistent static routing policies across all connected VPCs and on-premises environments. |
| **Risk Management** | Secure Data Transmission | IPsec with AES-128 and SHA-1 HMAC for Site-to-Site VPN ensures all data traversing the public internet between the simulated on-premises data center and AWS is encrypted, mitigating the risk of data interception. |
| **Compliance** | Network Isolation | The architectural patterns (static routing, Transit Gateway |

| | | |
|---|---|---|
| | | attachments, private subnets with VPC Endpoints) mirror the isolation strategies required for compliance frameworks handling sensitive data. |

---

# 2. Architecture Overview

The simulation uses a secondary VPC ( `192.168.0.0/16` ) as the on-premises data center, running a `libreswan` IPsec VPN daemon on an EC2 instance with an Elastic IP. This connects via an encrypted IPsec Site-to-Site VPN to an AWS Transit Gateway, which routes traffic to two application VPCs (VPC-A `10.1.0.0/16` and VPC-B `10.2.0.0/16` ).

**Key design decisions:**

- **Static routing** is used instead of BGP, which is simpler and sufficient for this lab topology.
- **NAT Traversal (NAT-T)** is required because the router EC2 instance sits behind an Internet Gateway (NAT). All ESP traffic is encapsulated in UDP port 4500.
- **SSM VPC Interface Endpoints** are deployed in VPC-A and VPC-B so that private instances can be accessed via Session Manager without internet gateways or SSH keys.
- A **Lambda custom resource** adds the TGW static route at deploy time, working around a known CloudFormation limitation where the VPN attachment ID is not natively exposed.

---

# 3. Comprehensive Deployment Guide

## Prerequisites and IAM Permissions

Before beginning the deployment, ensure you have the following configured:

1. **AWS Account** with administrative access.
2. **AWS CLI or CloudShell** — no local installation required if using CloudShell.
3. **IAM Permissions** — the deploying user or role must have:

- `ec2:*` (VPC, Subnets, Route Tables, Internet Gateways, Elastic IPs, EC2 Instances, VPC Endpoints, Security Groups, Customer Gateways, VPN Connections, Transit Gateways)
- `ssm:StartSession` , `ssm:DescribeInstanceInformation`

- `iam:CreateRole`, `iam:PutRolePolicy`, `iam:AttachRolePolicy`, `iam:PassRole`
- `lambda:CreateFunction`, `lambda:InvokeFunction`, `lambda:DeleteFunction`

## Step 3.1: Deploy the CloudFormation Stack

Save the v9 template (Appendix A) as `prj-sap-014-hybrid-simulation-v9.yml` and deploy:

Bash

```
aws cloudformation create-stack \
  --stack-name PRJ-SAP-014-Hybrid-Simulation \
  --template-body file://prj-sap-014-hybrid-simulation-v9.yml \
  --capabilities CAPABILITY_NAMED_IAM \
  --region us-east-1
```

Monitor progress:

Bash

```
aws cloudformation wait stack-create-complete \
  --stack-name PRJ-SAP-014-Hybrid-Simulation \
  --region us-east-1
```

The stack provisions: On-Prem VPC and Router, Transit Gateway, VPC-A and VPC-B with test instances, Customer Gateway, Site-to-Site VPN, Lambda custom resource for TGW static route, and SSM VPC Endpoints in both application VPCs.

## Step 3.2: Download the VPN Configuration (Get the PSKs)

1. Navigate to **VPC Console → Site-to-Site VPN Connections**.
2. Select `PRJ-SAP-014-Site-to-Site-VPN`.
3. Click **Download Configuration** → Vendor: **Generic**, Platform: **Generic**, Software: **Vendor Agnostic**.
4. Open the file and note the **Pre-Shared Keys** for Tunnel 1 and Tunnel 2.

## Step 3.3: Configure the On-Prem Router

Copy the `OnPremRouterInstanceId` from the CloudFormation Outputs tab, then connect via SSM:

Bash

```
aws ssm start-session --target <OnPremRouterInstanceId> --region us-east-1
```

Run the auto-configuration script with the `VpnConnectionId` from the Outputs:

Bash

```
sudo /usr/local/bin/configure-vpn.sh <VpnConnectionId> us-east-1
```

The script automatically:

- Fetches the router's private IP and public EIP from instance metadata
- Retrieves the two AWS VPN tunnel endpoint IPs via the AWS API
- Writes `/etc/ipsec.d/aws-tunnels.conf` with `left=<private-IP>`, `leftid=<EIP>`, and `encapsulation=yes`
- Adds persistent OS static routes for `10.1.0.0/16` and `10.2.0.0/16`

## Step 3.4: Update the Pre-Shared Keys

Bash

```
sudo nano /etc/ipsec.d/aws-tunnels.secrets
```

Replace `REPLACE_WITH_TUNNEL1_PSK` and `REPLACE_WITH_TUNNEL2_PSK` with the actual PSKs from the downloaded configuration file. Keep the double quotes around each PSK. Save and exit.

## Step 3.5: Start the IPsec Service

Bash

```
sudo systemctl restart ipsec
```

The `ipsec-tunnel-up.service` (installed by the template) automatically runs `ipsec auto --up Tunnel1` five seconds after ipsec starts, ensuring Tunnel 1 initiates the IKE handshake and refreshes the NAT-T mapping with AWS.

Verify the tunnel is established:

Bash

```
sudo ipsec status | grep -E 'STATE_QUICK_I2|ESTABLISHED|loaded'
```

You should see `STATE_QUICK_I2 (sent QI2, IPsec SA established)` for Tunnel 1.

### Step 3.6: Verify End-to-End Connectivity

Open a second CloudShell tab and connect to the VPC-A test instance:

```bash
aws ssm start-session --target <VpcATestInstanceId> --region us-east-1
```

Ping the On-Prem router's private IP (find it in the CloudFormation Outputs as `OnPremRouterPrivateIp`):

```bash
ping -c 10 192.168.1.x
```

**0% packet loss confirms end-to-end hybrid cloud connectivity is operational.**

---

## 4. Advanced Troubleshooting (Lessons from the Field)

The following failure modes were encountered and resolved during the development of this simulation. They are documented here for reference.

### Failure 1 — VPC Instances Unreachable via SSM ( `TargetNotConnected` )

**Symptom:** `An error occurred (TargetNotConnected)` when running `aws ssm start-session` against VPC-A or VPC-B instances.

**Root Cause:** The instances are in private subnets with no Internet Gateway or NAT Gateway. The SSM agent cannot reach the `ssm.us-east-1.amazonaws.com` endpoint over the public internet.

**Resolution (v6):** The template deploys three VPC Interface Endpoints ( `ssm` , `ssmmessages` , `ec2messages` ) into each application VPC, with `PrivateDnsEnabled: true` and a Security Group allowing HTTPS/443 from the VPC CIDR. The SSM agent resolves the endpoint via private DNS automatically.

---

### Failure 2 — TGW Static Route Creation Fails ( `AWS::EC2::VPNConnectionRoute` )

**Symptom:** CloudFormation fails with: *"Static routes for vpn-xxx must be added through the Transit Gateway API."*

**Root Cause:** `AWS::EC2::VPNConnectionRoute` only works when the VPN is attached to a Virtual Private Gateway (VGW). When attached to a Transit Gateway, this resource is unsupported. Additionally, the TGW VPN Attachment ID is not natively exposed as a CloudFormation return value (a known AWS limitation open since 2019), so `AWS::EC2::TransitGatewayRoute` cannot reference it directly.

**Resolution (v5):** A Lambda-backed Custom Resource runs inline Python code that: (1) calls `ec2:DescribeTransitGatewayAttachments` to find the VPN attachment ID, (2) calls `ec2:DescribeTransitGatewayRouteTables` to find the default route table, and (3) calls `ec2:CreateTransitGatewayRoute` to add the static route `192.168.0.0/16 → VPN attachment`. On stack deletion, the Lambda calls `ec2:DeleteTransitGatewayRoute` to clean up.

---

## Failure 3 — IPsec Tunnel "eroute in use" Error

**Symptom:** `cannot install eroute -- it is in use for "Tunnel1" #3` when bringing up Tunnel 2.

**Root Cause:** Both tunnels share identical traffic selectors (`leftsubnet=192.168.0.0/16`, `rightsubnet=10.0.0.0/8`). The Linux kernel XFRM subsystem only permits one IPsec policy per unique traffic selector pair. Whichever tunnel establishes Phase 2 first claims the eroute exclusively.

**Resolution (v9):** Tunnel 2 is set to `auto=add` instead of `auto=start`. Only Tunnel 1 initiates automatically. Tunnel 2 is loaded into the kernel policy table but not initiated, serving as a standby that activates via DPD if Tunnel 1 fails.

---

## Failure 4 — NAT-T Return Path Silent Drop (The Core Issue)

**Symptom:** Ping from On-Prem router to VPC-A shows 100% packet loss. `ipsec whack --trafficstatus` shows `ESPout` increasing but `ESPin=0`. A `tcpdump -i eth0 esp -n` on the router shows zero incoming ESP packets. Meanwhile, a `tcpdump` on the VPC-A instance confirms it is receiving ICMP requests and sending replies.

**Root Cause (identified in v9):** The libreswan directive `left=%defaultroute` resolves to the EC2 instance's private subnet IP (e.g., `192.168.1.30`), not the Elastic IP. The `leftid=$MY_EIP` sets the IKE identity to the EIP correctly, but the XFRM source IP template is set to the private IP. This creates a mismatch: AWS establishes the NAT-T UDP 4500 mapping for the EIP but the XFRM state expects ESP packets to arrive addressed to the private IP. AWS drops the return ESP packets because the source/destination IP in the XFRM state does not match the actual packet headers after NAT translation.

**Resolution (v9):** The `configure-vpn.sh` script now fetches `MY_PRIVATE_IP` from instance metadata (`http://169.254.169.254/latest/meta-data/local-ipv4`) and uses `left=$MY_PRIVATE_IP` explicitly. Additionally, `encapsulation=yes` is added to both tunnel configurations to force

NAT-T (UDP 4500 encapsulation) unconditionally, regardless of libreswan's NAT detection heuristics. This ensures the XFRM state, the IKE negotiation, and the actual ESP packet flow are all consistent.

---

## Appendix A: CloudFormation Template (v9)

The full v9 template is provided in the accompanying file `prj-sap-014-hybrid-simulation-v9.yml`. Deploy it with:

Bash

```
aws cloudformation create-stack \
  --stack-name PRJ-SAP-014-Hybrid-Simulation \
  --template-body file://prj-sap-014-hybrid-simulation-v9.yml \
  --capabilities CAPABILITY_NAMED_IAM \
  --region us-east-1
```

To update an existing stack using a change set:

Bash

```
aws cloudformation create-change-set \
  --stack-name PRJ-SAP-014-Hybrid-Simulation \
  --template-body file://prj-sap-014-hybrid-simulation-v9.yml \
  --capabilities CAPABILITY_NAMED_IAM \
  --change-set-name v9-nat-t-root-cause-fix \
  --region us-east-1
```