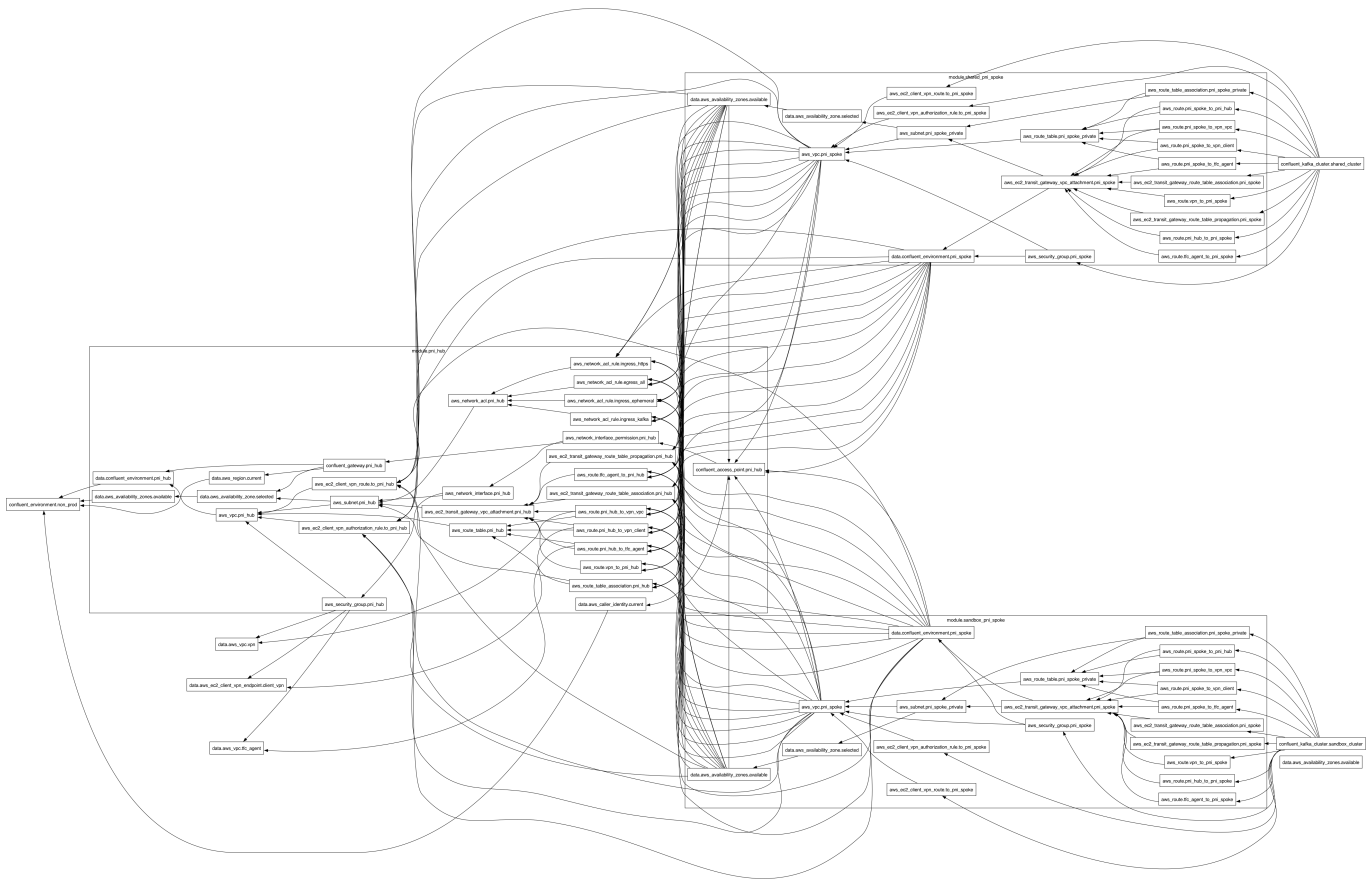


IaC Confluent Cloud AWS Private Network Interface (PNI), Infrastructure and Networking Example



Terraform-managed, hub-and-spoke Private Network Interface (PNI) connectivity between AWS VPCs and Confluent Cloud Enterprise Kafka clusters, deployed via Terraform Cloud.

Table of Contents

- [1.0 Overview](#)
- [2.0 Architecture](#)
- [3.0 Module Structure](#)
- [4.0 Prerequisites](#)
 - [4.1 Client VPN, Centralized DNS Server, and Transit Gateway](#)
 - [4.1.1 Key Features Required for Confluent PNI to Work](#)
 - [4.1.1.1 Hub-and-Spoke Network Architecture via Transit Gateway](#)
 - [4.1.1.2 Client VPN Integration](#)
 - [4.1.1.3 Cross-VPC Routing](#)
 - [4.1.1.4 Security & Observability](#)
 - [4.2 Terraform Cloud Agent](#)
 - [4.2.1 Key Features Required for Confluent PNI to Work \(TFC Agent Configuration\)](#)
 - [4.2.1.1 Custom DHCP Options for DNS Resolution](#)
 - [4.2.1.2 Transit Gateway Connectivity](#)
 - [4.2.1.3 Security Group Configuration for Kafka Traffic](#)

- [4.2.1.4 AWS VPC Endpoints for Private Service Access](#)
 - [4.2.1.5 ECS Fargate Deployment Pattern](#)
 - [4.2.1.6 IAM Permissions for Infrastructure Management](#)
 - [4.2.1.7 Network Architecture Summary](#)
- [5.0 Configuration](#)
 - [5.1 Key Input Variables](#)
 - [5.2 CIDR Allocations](#)
- [6.0 Deployment](#)
 - [6.1 Create](#)
 - [6.2 Destroy](#)
- [7.0 Outputs](#)
- [8.0 Security Design](#)
- [9.0 How PNI Differs from PrivateLink](#)
- [10.0 Resources](#)

1.0 Overview

This repo provisions a production-grade private networking topology that connects AWS workload VPCs to Confluent Cloud Enterprise Kafka clusters via **Private Network Interface (PNI)**, Confluent's next-gen model with customer-owned ENIs deployed directly in your VPCs.

The architecture follows a **Hub-and-Spoke** pattern:

- A **PNI Hub VPC** owns the ENIs registered with Confluent Cloud and serves as the centralized private connectivity anchor.
- **PNI Spoke VPCs** (sandbox, shared) each host a Confluent Cloud Enterprise Kafka cluster and peer connectivity through the hub via AWS Transit Gateway.
- All VPCs are stitched together through an existing **AWS Transit Gateway (TGW)**, enabling a VPN-connected developer/operator to reach Confluent Cloud endpoints without traversing the public internet.

2.0 Architecture

```

flowchart TD
    subgraph VPN ["VPN VPC (existing)"]
        CVE["AWS Client VPN Endpoint"]
    end

    subgraph TFC_AGENT ["TFC Agent VPC (existing)"]
        AGENT["Terraform Cloud Agent"]
    end

    TGW["AWS Transit Gateway\n+ Route Table"]

    subgraph PNI_HUB ["PNI Hub VPC – 10.3.0.0/20"]
        direction TB
        SG["Security Group\n(port 443, 9092 ingress only)"]
    end

```

```

    NACL["Network ACL\n(443, 9092, ephemeral)"]
    ENI1["ENI - AZ-a"]
    ENI2["ENI - AZ-b"]
    ENI3["ENI - AZ-c"]
    TGW_ATT_HUB["TGW Attachment"]
    GW["confluent_gateway\n(AWS PNI Gateway)"]
    AP["confluent_access_point\n(PNI Access Point)"]
    SG --> ENI1 & ENI2 & ENI3
    ENI1 & ENI2 & ENI3 --> AP
    AP --> GW
end

subgraph SANDBOX_SPOKE["Sandbox PNI Spoke VPC - 10.0.0.0/20"]
    TGW_ATT_SB["TGW Attachment"]
    SK_CLUSTER["confluent_kafka_cluster\nsandbox_cluster\n(Enterprise,
HIGH availability)"]
end

subgraph SHARED_SPOKE["Shared PNI Spoke VPC - 10.1.0.0/20"]
    TGW_ATT_SH["TGW Attachment"]
    SH_CLUSTER["confluent_kafka_cluster\nshared_cluster\n(Enterprise,
HIGH availability)"]
end

subgraph CONFLUENT_CLOUD["Confluent Cloud"]
    ENV["Environment: non-prod\n(Stream Governance: ESSENTIALS)"]
    GW --> ENV
    SK_CLUSTER --> ENV
    SH_CLUSTER --> ENV
end

CVE -->|"routes via TGW"| TGW
AGENT -->|"routes via TGW"| TGW
TGW <-->|"attach + propagate"| TGW_ATT_HUB
TGW <-->|"attach + propagate"| TGW_ATT_SB
TGW <-->|"attach + propagate"| TGW_ATT_SH
TGW_ATT_HUB --> PNI_HUB
TGW_ATT_SB --> SANDBOX_SPOKE
TGW_ATT_SH --> SHARED_SPOKE

```

3.0 Module Structure

```

.
├── main.tf                # Root: environment, cluster
resources, module calls
├── data.tf                # Data sources: VPCs, VPN endpoint,
AZs
├── variables.tf           # Root-level input variables
├── outputs.tf             # PNI gateway & access point IDs
├── versions.tf            # Provider version pins (AWS 6.33,

```

```

Confluent 2.62, TFE 0.73)
├─ provider.tf                # Provider configuration
├─ deploy.sh                  # Bootstrap / teardown script (AWS
SSO + TF env vars)
├─ modules/
│   └─ aws-vpc-confluent-pni-hub/ # PNI Hub module
│       └─ setup-confluent-pni-hub.tf # Gateway, access point,
VPC, subnets, ENIs, SG
│           └─ setup-aws-vpc-tgw-private_routing.tf # TGW attachment,
associations, routes
│               └─ setup-aws-vpc-security_group_rules.tf # NACL rules
│               └─ setup-aws-network-permissions.tf # ENI permissions for
Confluent's AWS account
│                   └─ data.tf # Module-level data
sources
│   └─ variables.tf # Module inputs
│   └─ outputs.tf # Gateway/access point
IDs exposed to root
│   └─ versions.tf # Module provider
constraints
└─ aws-vpc-confluent-pni-spoke/ # PNI Spoke module
(sandbox, shared)
    └─ main.tf
    └─ data.tf
    └─ variables.tf
    └─ outputs.tf
    └─ versions.tf

```

4.0 Prerequisites

This project assumes you have the following prerequisites in place:

- Client VPN, Centralized DNS Server, and Transit Gateway
- Terraform Cloud Agent

4.1 Client VPN, Centralized DNS Server, and Transit Gateway

```

%%{init: {'theme': 'base', 'themeVariables': { 'primaryColor': '#1a73e8',
'primaryTextColor': '#fff', 'primaryBorderColor': '#1557b0', 'lineColor':
'#5f6368', 'secondaryColor': '#34a853', 'tertiaryColor': '#fbbc04'}}}%

```

flowchart TB

```

    subgraph USERS["👤 Remote Users"]
        VPNClient["VPN Client
(OpenVPN/AWS Client)"]
    end

    subgraph AWS["☁️ AWS Cloud"]
        subgraph VPN_VPC["Client VPN VPC
var.vpn_vpc_cidr"]

```

```

        VPNEndpoint["AWS Client VPN
Endpoint"]
        VPNSubnets["VPN Subnets
(Multi-AZ)"]
        VPNSG["Security Group
client-vpn-sg"]
        VPNResolver["Route53 Outbound
Resolver Endpoint"]
        VPNEndpoint --> VPNSubnets
        VPNSubnets --> VPNSG
        VPNSubnets --> VPNResolver
    end

    subgraph TGW["Transit Gateway
signalroom-tgw"]
        TGWCore["TGW Core
ASN: 64512"]
        TGWRouteTable["Custom Route
Tables"]
        TGWCore --> TGWRouteTable
    end

    subgraph DNS_VPC["DNS VPC (Centralized)
var.dns_vpc_cidr"]
        R53Inbound["Route53 Inbound
Resolver Endpoint"]
        R53PHZ["Private Hosted Zones
*.aws.confluent.cloud"]
        R53Inbound --> R53PHZ
    end

    subgraph TFC_VPC["TFC Agent VPC
var.tfc_agent_vpc_cidr"]
        TFCAgent["Terraform Cloud
Agent"]
    end

    subgraph WORKLOAD_VPCs["Workload VPCs"]
        subgraph WL1["Workload VPC 1"]
            VPCE1["VPC Endpoint
(PrivateLink)"]
        end
        subgraph WL2["Workload VPC N..."]
            VPCEN["VPC Endpoint
(PrivateLink)"]
        end
    end

    ACM["ACM Certificates
(Server & Client)"]
    CWLogs["CloudWatch Logs
VPN & Flow Logs"]
end

```

```

    subgraph CONFLUENT["▲ Confluent Cloud"]
        PrivateLinkService["PrivateLink Service
Endpoint"]
        Kafka["Kafka Cluster
(Private)"]
        PrivateLinkService --> Kafka
    end

    %% Connections
    VPNClient -->|"Mutual TLS
Authentication"| VPNEndpoint
    ACM -. ->|"Certificate Auth"| VPNEndpoint

    VPN_VPC -->|"TGW Attachment"| TGW
    DNS_VPC -->|"TGW Attachment"| TGW
    TFC_VPC -->|"TGW Attachment"| TGW
    WL1 -->|"TGW Attachment"| TGW
    WL2 -->|"TGW Attachment"| TGW

    VPNResolver -->|"DNS Forwarding
Rule"| R53Inbound
    R53PHZ -->|"Returns Private
Endpoint IPs"| VPCE1

    VPCE1 -->|"AWS PrivateLink"| PrivateLinkService
    VPCEN -->|"AWS PrivateLink"| PrivateLinkService

    VPNEndpoint -. ->|"Logs"| CWLogs
    TGW -. ->|"Flow Logs"| CWLogs

    %% Styling
    classDef userStyle fill:#4285f4,stroke:#1557b0,stroke-
width:2px,color:#fff
    classDef vpcStyle fill:#e8f0fe,stroke:#1a73e8,stroke-width:2px
    classDef tgwStyle fill:#fef7e0,stroke:#f9ab00,stroke-width:3px
    classDef dnsStyle fill:#e6f4ea,stroke:#34a853,stroke-width:2px
    classDef confluentStyle fill:#f3e8fd,stroke:#9334e6,stroke-width:2px
    classDef serviceStyle fill:#fff,stroke:#5f6368,stroke-width:1px

    class USERS userStyle
    class VPN_VPC,TFC_VPC,WORKLOAD_VPCs,WL1,WL2 vpcStyle
    class TGW tgwStyle
    class DNS_VPC dnsStyle
    class CONFLUENT confluentStyle

```

4.1.1 Key Features Required for Confluent PNI to Work

4.1.1.1 Hub-and-Spoke Network Architecture via Transit Gateway

- Transit Gateway serves as the central routing hub connecting all VPCs
- Disabled default route table association/propagation for explicit routing control
- DNS support enabled on the TGW (`dns_support = "enable"`)

- Custom route tables for fine-grained traffic control between VPCs

4.1.1.2 Client VPN Integration

- Mutual TLS authentication using ACM certificates (server + client)
- Split tunnel configuration for routing only Confluent traffic through VPN
- Authorization rules controlling which CIDRs VPN clients can access
- Routes added to VPN endpoint for all workload VPC CIDRs via Transit Gateway

4.1.1.3 Cross-VPC Routing

- TGW attachments for: VPN VPC, DNS VPC, TFC Agent VPC, and all Workload VPCs
- Route tables in each VPC with routes to other VPCs via TGW
- Workload VPC CIDRs aggregated and distributed to VPN client routes

4.1.1.4 Security & Observability

- Dedicated security groups per component (VPN endpoint, etc.)
- VPC Flow Logs and TGW Flow Logs to CloudWatch
- VPN connection logging for audit trails
- IAM roles with least-privilege for flow log delivery

4.2 Terraform Cloud Agent

```
%%{init: {'theme': 'base', 'themeVariables': { 'primaryColor': '#1a73e8',
'primaryTextColor': '#fff', 'primaryBorderColor': '#1557b0', 'lineColor':
'#5f6368', 'secondaryColor': '#34a853', 'tertiaryColor': '#fbbc04'}}}%
```

```
flowchart TB
```

```
    subgraph TERRAFORM_CLOUD["^ Terraform Cloud (HCP)"]
        TFC["Terraform Cloud
```

```
API & Workspaces"]
```

```
        AgentPool["Agent Pool
(signalroom)"]
```

```
    end
```

```
    subgraph AWS["^ AWS Cloud"]
```

```
        subgraph TFC_AGENT_VPC["TFC Agent VPC
var.vpc_cidr"]
```

```
            subgraph PUBLIC_SUBNETS["Public Subnets (Multi-AZ)"]
                IGW["Internet
Gateway"]
```

```
                NAT1["NAT Gateway
AZ-1"]
```

```
                NAT2["NAT Gateway
AZ-2"]
```

```
            end
```

```
            subgraph PRIVATE_SUBNETS["Private Subnets (Multi-AZ)"]
                subgraph ECS["ECS Fargate Cluster"]
```

```

        TFCAgent1["TFC Agent
Container"]
        TFCAgent2["TFC Agent
Container"]
    end

    subgraph AWS_ENDPOINTS["AWS VPC Endpoints"]
        VPCE_SM["Secrets Manager
Endpoint"]
        VPCE_CW["CloudWatch Logs
Endpoint"]
        VPCE_ECR["ECR API/DKR
Endpoints"]
        VPCE_S3["S3 Gateway
Endpoint"]
    end

    CONFLUENT_SG["Confluent PrivateLink
Security Group"]
    end

    DHCP["DHCP Options
(Custom DNS)"]
    TFC_AGENT_SG["TFC Agent
Security Group"]
    end

    subgraph TGW["Transit Gateway
signalroom-tgw"]
        TGWCore["TGW Core"]
        TGWRT["Route Table"]
    end

    subgraph DNS_VPC["DNS VPC (Centralized)
var.dns_vpc_cidr"]
        R53Inbound["Route53 Inbound
Resolver"]
        PHZ["Private Hosted Zones
*.aws.confluent.cloud"]
    end

    subgraph CLIENT_VPN_VPC["Client VPN VPC
var.client_vpn_vpc_cidr"]
        VPNEndpoint["Client VPN
Endpoint"]
    end

    subgraph WORKLOAD_VPCs["Workload VPCs
(Confluent PrivateLink)"]
        subgraph WL1["Workload VPC 1"]
            VPCE1["PrivateLink
Endpoint"]
        end
        subgraph WL2["Workload VPC N"]

```

```

        VPCEN["PrivateLink
Endpoint"]
        end
    end

    SecretsManager["AWS Secrets Manager
(TFC Agent Token)"]
    CloudWatch["CloudWatch Logs"]
    ECR_Registry["ECR Registry
(hashicorp/tfc-agent)"]
    end

    subgraph CONFLUENT["▲ Confluent Cloud"]
        PrivateLinkSvc["PrivateLink
Service"]
        Kafka["Kafka Cluster
(Private)"]
    end

    %% External Connections
    TFC <-->|"HTTPS/443
via NAT"| TFCAgent1
    TFC <-->|"HTTPS/443
via NAT"| TFCAgent2
    AgentPool -.->|"Agent Registration"| TFCAgent1

    %% Internal VPC Connections
    TFCAgent1 --> TFC_AGENT_SG
    TFCAgent2 --> TFC_AGENT_SG
    TFCAgent1 --> VPCE_SM
    TFCAgent2 --> VPCE_CW

    VPCE_SM -.->|"Private DNS"| SecretsManager
    VPCE_CW -.->|"Private DNS"| CloudWatch
    VPCE_ECR -.->|"Private DNS"| ECR_Registry

    NAT1 --> IGW
    NAT2 --> IGW
    TFCAgent1 -->|"0.0.0.0/0"| NAT1
    TFCAgent2 -->|"0.0.0.0/0"| NAT2

    %% DHCP & DNS Flow
    DHCP -->|"DNS Servers:
VPC + Centralized"| TFCAgent1
    TFCAgent1 -->|"DNS Query:
*.confluent.cloud"| R53Inbound

    %% Transit Gateway Connections
    TFC_AGENT_VPC -->|"TGW Attachment"| TGW
    DNS_VPC -->|"TGW Attachment"| TGW
    CLIENT_VPN_VPC -->|"TGW Attachment"| TGW
    WL1 -->|"TGW Attachment"| TGW
    WL2 -->|"TGW Attachment"| TGW

```

```

%% Route Propagation
TGWCore --> TGWRT

%% DNS Resolution
R53Inbound --> PHZ
PHZ -->|"Returns Private IPs"| VPCE1

%% PrivateLink Connections
VPCE1 -->|"AWS PrivateLink"| PrivateLinkSvc
VPCEN -->|"AWS PrivateLink"| PrivateLinkSvc
PrivateLinkSvc --> Kafka

%% TFC Agent to Workload VPCs
TFC_AGENT_SG -->|"HTTPS/443
Kafka/9092"| CONFLUENT_SG
CONFLUENT_SG -->|"via TGW"| VPCE1
CONFLUENT_SG -->|"via TGW"| VPCEN

%% Styling
classDef tfcStyle fill:#5c4ee5,stroke:#3d32a8,stroke-
width:2px,color:#fff
classDef vpcStyle fill:#e8f0fe,stroke:#1a73e8,stroke-width:2px
classDef tgwStyle fill:#fef7e0,stroke:#f9ab00,stroke-width:3px
classDef dnsStyle fill:#e6f4ea,stroke:#34a853,stroke-width:2px
classDef confluentStyle fill:#f3e8fd,stroke:#9334e6,stroke-width:2px
classDef endpointStyle fill:#fce8e6,stroke:#ea4335,stroke-width:1px
classDef ecsStyle fill:#fff3e0,stroke:#ff9800,stroke-width:2px

class TERRAFORM_CLOUD tfcStyle
class TFC_AGENT_VPC,CLIENT_VPN_VPC,WORKLOAD_VPCs,WL1,WL2 vpcStyle
class TGW tgwStyle
class DNS_VPC dnsStyle
class CONFLUENT confluentStyle
class AWS_ENDPOINTS,VPCE_SM,VPCE_CW,VPCE_ECR,VPCE_S3 endpointStyle
class ECS ecsStyle

```

4.2.1 Key Features Required for Confluent PNI to Work (TFC Agent Configuration)

4.2.1.1 Custom DHCP Options for DNS Resolution

- DHCP Options Set configured with **dual DNS servers**: VPC default DNS (`cidrhost(vpc_cidr, 2)`) AND centralized DNS VPC resolver IPs
- Region-aware domain name configuration (`ec2.internal` for us-east-1, `{region}.compute.internal` for others)
- Associates TFC Agent VPC with custom DHCP options to route Confluent domain queries to the central DNS infrastructure

4.2.1.2 Transit Gateway Connectivity

- TFC Agent VPC attached to shared Transit Gateway with DNS support enabled
- Explicit route table association and route propagation (not using TGW defaults)

- Routes added from private subnets to: DNS VPC, and Client VPN VPC
- Flattened route map pattern (`for_each`) ensures routes are created for every workload VPC CIDR

4.2.1.3 Security Group Configuration for Kafka Traffic

- **TFC Agent Security Group** with egress rules for:
 - HTTPS (443) and Kafka (9092) to each workload VPC CIDR
 - DNS (UDP/TCP 53) to DNS VPC CIDR specifically
 - General HTTPS/HTTP for Terraform Cloud API and package downloads

4.2.1.4 AWS VPC Endpoints for Private Service Access

- **Interface endpoints** with private DNS enabled for: Secrets Manager, CloudWatch Logs, ECR API, ECR DKR
- **S3 Gateway endpoint** (required for ECR image layer pulls)
- Dedicated security group for VPC endpoints allowing HTTPS from within VPC
- Eliminates NAT Gateway dependency for AWS service calls

4.2.1.5 ECS Fargate Deployment Pattern

- TFC Agents run in private subnets with `assign_public_ip = false`
- NAT Gateways per AZ for outbound internet (Terraform Cloud API communication)
- Agent token stored in Secrets Manager, fetched via VPC Endpoint
- Container health checks and deployment circuit breaker for reliability

4.2.1.6 IAM Permissions for Infrastructure Management

- Task role with Transit Gateway, VPC, Route53 Resolver, and Client VPN management permissions
- Execution role with Secrets Manager access for agent token retrieval
- KMS permissions scoped to Secrets Manager service for encryption/decryption

4.2.1.7 Network Architecture Summary

- **Hub-and-spoke model:** TGW connects TFC Agent VPC → DNS VPC → Workload VPCs

5.0 Configuration

All sensitive values are passed as environment variables (never stored in `.tfvars`). The `deploy.sh` script handles setting `TF_VAR_*` exports automatically after AWS SSO authentication.

5.1 Key Input Variables

Variable	Description
<code>tgw_id</code>	Existing Transit Gateway ID
<code>tgw_rt_id</code>	Transit Gateway Route Table ID
<code>vpn_vpc_id</code>	VPN VPC ID

Variable	Description
vpn_vpc_rt_ids	Comma-separated VPN VPC route table IDs
vpn_endpoint_id	AWS Client VPN Endpoint ID
vpn_target_subnet_ids	Comma-separated VPN associated subnet IDs
tfc_agent_vpc_id	Terraform Cloud Agent VPC ID
tfc_agent_vpc_rt_ids	Comma-separated TFC Agent VPC route table IDs
eni_number_per_subnet	Number of ENIs per subnet (default: 17)
aws_region	AWS region for all resources

5.2 CIDR Allocations

Network	CIDR
PNI Hub VPC	10.3.0.0/20
Sandbox Spoke VPC	10.0.0.0/20
Shared Spoke VPC	10.1.0.0/20

All VPCs use 3 subnets across 3 AZs with /4 new bits of sub-netting.

6.0 Deployment

6.1 Create

```
./deploy.sh create --profile=<SSO_PROFILE_NAME> \
    --confluent-api-key=<CONFLUENT_API_KEY> \
    --confluent-api-secret=<CONFLUENT_API_SECRET> \
    --tfe-token=<TFE_TOKEN> \
    --tgw-id=<TGW_ID> \
    --tgw-rt-id=<TGW_RT_ID> \
    --tfc-agent-vpc-id=<TFC_AGENT_VPC_ID> \
    --tfc-agent-vpc-rt-ids=<TFC_AGENT_VPC_RT_IDS> \
    --vpn-vpc-id=<VPN_VPC_ID> \
    --vpn-vpc-rt-ids=<VPN_VPC_RT_IDS> \
    --vpn-endpoint-id=<VPN_ENDPOINT_ID> \
    --vpn-target-subnet-ids=<VPN_TARGET_SUBNET_IDS> \
    --pni-hub-vpc-cidr=<PNI_HUB_VPC_CIDR>
```

Here's the argument table for `deploy.sh create` command:

Argument	Required	Placeholder	Description
----------	----------	-------------	-------------

Argument	Required	Placeholder	Description
<code>--profile</code>	✓	<code><SSO_PROFILE_NAME></code>	AWS SSO profile name used to authenticate via <code>aws sso login</code> and derive the AWS region and account ID for the session.
<code>--confluent-api-key</code>	✓	<code><CONFLUENT_API_KEY></code>	Confluent Cloud API key used by Terraform to authenticate against the Confluent provider.
<code>--confluent-api-secret</code>	✓	<code><CONFLUENT_API_SECRET></code>	Confluent Cloud API secret paired with the API key for Terraform provider authentication.
<code>--tfe-token</code>	✓	<code><TFE_TOKEN></code>	Terraform Enterprise/Cloud (TFE/TFC) API token used to authenticate the Terraform Cloud Agent with the TFC control plane.
<code>--tgw-id</code>	✓	<code><TGW_ID></code>	AWS Transit Gateway ID (e.g., <code>tgw-xxxxxxxxxxxxxxxxxx</code>) that serves as the central hub routing traffic between the TFC Agent VPC, VPN VPC, and the PrivateLink Network Interconnect (PNI) Hub VPC.
<code>--tgw-rt-id</code>	✓	<code><TGW_RT_ID></code>	AWS Transit Gateway Route Table ID (e.g., <code>tgw-rtb-xxxxxxxxxxxxxxxxxx</code>) to which routes are added for inter-VPC traffic through the Transit Gateway.
<code>--tfc-agent-vpc-id</code>	✓	<code><TFC_AGENT_VPC_ID></code>	VPC ID of the VPC hosting the Terraform Cloud Agent (ECS Fargate). This VPC is attached to the Transit Gateway for connectivity to the Confluent Private Network.
<code>--tfc-agent-vpc-rt-ids</code>	✓	<code><TFC_AGENT_VPC_RT_IDS></code>	Comma-separated list of route table IDs within the TFC Agent VPC. Routes to the PNI Hub VPC CIDR are injected into these tables to enable PrivateLink reachability.
<code>--vpn-vpc-id</code>	✓	<code><VPN_VPC_ID></code>	VPC ID of the VPC where the Client VPN endpoint is deployed, allowing operator/admin connectivity into the private network.

Argument	Required	Placeholder	Description
<code>--vpn-vpc-rt-ids</code>	✓	<code><VPN_VPC_RT_IDS></code>	Comma-separated list of route table IDs within the VPN VPC. Routes to the PNI Hub VPC CIDR are added here so VPN-connected clients can reach Confluent Private Network.
<code>--vpn-endpoint-id</code>	✓	<code><VPN_ENDPOINT_ID></code>	AWS Client VPN Endpoint ID (e.g., <code>cvpn-endpoint-xxxxxxxxxxxxxxxxxx</code>). Used to associate target subnets and configure VPN routing rules.
<code>--vpn-target-subnet-ids</code>	✓	<code><VPN_TARGET_SUBNET_IDS></code>	Comma-separated list of subnet IDs within the VPN VPC to which the Client VPN endpoint is associated, determining which AZs VPN traffic can ingress through.
<code>--pni-hub-vpc-cidr</code>	✓	<code><PNI_HUB_VPC_CIDR></code>	CIDR block of the Confluent PrivateLink Network Interconnect (PNI) Hub VPC (e.g., <code>10.x.x.x/xx</code>). This CIDR is injected as a route destination into the TFC Agent and VPN VPC route tables so traffic destined for Confluent endpoints is properly directed through the Transit Gateway.

All 13 arguments are required, the script exits with code 85 if any are missing.

The script will:

1. Authenticate to AWS SSO and export temporary credentials.
2. Export all `TF_VAR_*` environment variables.
3. Run `terraform init`, `terraform plan`, prompt for confirmation, then `terraform apply`.
4. Generate a Terraform graph visualization at [docs/images/terraform-visualization.png](#).

6.2 Destroy

```
./deploy.sh destroy --profile=<SSO_PROFILE_NAME> \
    --confluent-api-key=<CONFLUENT_API_KEY> \
    --confluent-api-secret=<CONFLUENT_API_SECRET> \
    --tfe-token=<TFE_TOKEN> \
    --tgw-id=<TGW_ID> \
    --tgw-rt-id=<TGW_RT_ID> \
    --tfc-agent-vpc-id=<TFC_AGENT_VPC_ID> \
    --tfc-agent-vpc-rt-ids=<TFC_AGENT_VPC_RT_IDS> \
    --vpn-vpc-id=<VPN_VPC_ID> \
    --vpn-vpc-rt-ids=<VPN_VPC_RT_IDS> \
```

```
--vpn-endpoint-id=<VPN_ENDPOINT_ID> \
--vpn-target-subnet-ids=<VPN_TARGET_SUBNET_IDS> \
--pni-hub-vpc-cidr=<PNI_HUB_VPC_CIDR>
```

Here's the argument table for `deploy.sh destroy` command:

Argument	Required	Placeholder	Description
<code>--profile</code>	✓	<code><SSO_PROFILE_NAME></code>	AWS SSO profile name used to authenticate via <code>aws sso login</code> and derive the AWS region and account ID for the session.
<code>--confluent-api-key</code>	✓	<code><CONFLUENT_API_KEY></code>	Confluent Cloud API key used by Terraform to authenticate against the Confluent provider.
<code>--confluent-api-secret</code>	✓	<code><CONFLUENT_API_SECRET></code>	Confluent Cloud API secret paired with the API key for Terraform provider authentication.
<code>--tfe-token</code>	✓	<code><TFE_TOKEN></code>	Terraform Enterprise/Cloud (TFE/TFC) API token used to authenticate the Terraform Cloud Agent with the TFC control plane.
<code>--tgw-id</code>	✓	<code><TGW_ID></code>	AWS Transit Gateway ID (e.g., <code>tgw-xxxxxxxxxxxxxxxxxxxx</code>) that serves as the central hub routing traffic between the TFC Agent VPC, VPN VPC, and the PrivateLink Network Interconnect (PNI) Hub VPC.
<code>--tgw-rt-id</code>	✓	<code><TGW_RT_ID></code>	AWS Transit Gateway Route Table ID (e.g., <code>tgw-rtb-xxxxxxxxxxxxxxxxxxxx</code>) to which routes are added for inter-VPC traffic through the Transit Gateway.
<code>--tfc-agent-vpc-id</code>	✓	<code><TFC_AGENT_VPC_ID></code>	VPC ID of the VPC hosting the Terraform Cloud Agent (ECS Fargate). This VPC is attached to the Transit Gateway for connectivity to the Confluent Private Network.
<code>--tfc-agent-vpc-rt-ids</code>	✓	<code><TFC_AGENT_VPC_RT_IDS></code>	Comma-separated list of route table IDs within the TFC Agent VPC. Routes to the PNI Hub VPC CIDR are injected into these tables to enable PrivateLink reachability.

Argument	Required	Placeholder	Description
<code>--vpn-vpc-id</code>	✓	<code><VPN_VPC_ID></code>	VPC ID of the VPC where the Client VPN endpoint is deployed, allowing operator/admin connectivity into the private network.
<code>--vpn-vpc-rt-ids</code>	✓	<code><VPN_VPC_RT_IDS></code>	Comma-separated list of route table IDs within the VPN VPC. Routes to the PNI Hub VPC CIDR are added here so VPN-connected clients can reach Confluent Private Network.
<code>--vpn-endpoint-id</code>	✓	<code><VPN_ENDPOINT_ID></code>	AWS Client VPN Endpoint ID (e.g., <code>cvpn-endpoint-xxxxxxxxxxxxxxxxxx</code>). Used to associate target subnets and configure VPN routing rules.
<code>--vpn-target-subnet-ids</code>	✓	<code><VPN_TARGET_SUBNET_IDS></code>	Comma-separated list of subnet IDs within the VPN VPC to which the Client VPN endpoint is associated, determining which AZs VPN traffic can ingress through.
<code>--pni-hub-vpc-cidr</code>	✓	<code><PNI_HUB_VPC_CIDR></code>	CIDR block of the Confluent PrivateLink Network Interconnect (PNI) Hub VPC (e.g., <code>10.x.x.x/xx</code>). This CIDR is injected as a route destination into the TFC Agent and VPN VPC route tables so traffic destined for Confluent endpoints is properly directed through the Transit Gateway.

All 13 arguments are required, the script exits with code 85 if any are missing.

Destroy runs `terraform destroy -auto-approve` and regenerates the visualization.

7.0 Outputs

Output	Description
<code>confluent_pni_hub_gateway_id</code>	ID of the <code>confluent_gateway</code> resource (PNI Hub)
<code>confluent_pni_hub_access_point_id</code>	ID of the <code>confluent_access_point</code> resource
<code>confluent_environment_id</code>	ID of the Confluent Cloud environment where clusters are provisioned

Output	Description
<code>confluent_sandbox_cluster_id</code>	ID of the sandbox Kafka cluster (<code>confluent_kafka_cluster</code>)
<code>confluent_shared_cluster_id</code>	ID of the shared Kafka cluster (<code>confluent_kafka_cluster</code>)

8.0 Security Design

Security Group (PNI ENIs): Ingress-only on ports `443` (HTTPS/REST/Schema Registry) and `9092` (Kafka), sourced from the PNI Hub VPC CIDR, TFC Agent VPC CIDR, VPN VPC CIDR, and Client VPN CIDR. **No egress rules are defined**, which causes Terraform to revoke AWS's default `0.0.0.0/0` egress — intentionally mirroring PrivateLink's unidirectional behavior and preventing Confluent-initiated connections into the customer network.

Network ACL: Allows TCP ingress on `443`, `9092`, and ephemeral ports `1024–65535`. Allows all egress.

ENI Permissions: `aws_network_interface_permission` grants Confluent's AWS account `INSTANCE-ATTACH` permission on each customer-owned ENI. This is the core PNI handshake — Confluent attaches its broker VMs to your ENIs without your traffic ever leaving the AWS backbone.

9.0 How PNI Differs from PrivateLink

Aspect	PrivateLink	PNI
ENI ownership	Confluent's account	Customer's account
DNS	Requires PHZ + VPC associations	Confluent manages DNS
Connectivity model	VPC Interface Endpoint	ENIs registered via <code>confluent_access_point</code>
Egress control	Unidirectional by design	Explicit empty egress on SG required
Port 53 (DNS) rules	Required in SG	Not needed

10.0 Resources

- [Confluent PNI Documentation](#)
- [Confluent PNI FAQ](#)
- [AWS Multi-VPC ENI Attachment](#)
- [confluent_gateway Terraform resource](#)
- [confluent_access_point Terraform resource](#)