IaC Confluent Cloud AWS Private Linking with Cluster Linking Example

```
%%{init: {'theme': 'default', 'themeVariables': {'background': '#ffffff',
'primaryColor': '#dbeafe', 'lineColor': '#64748b', 'textColor':
'#1e293b'}, 'flowchart': {'curve': 'basis'}}}%%

graph TB
    %% ============================================================
    %% STYLING — Light backgrounds, bold borders, dark text
    %% ============================================================
    classDef confluentCloud fill:#dbeafe,stroke:#1d4ed8,stroke-
width:2px,color:#1e3a8a
    classDef vpcBox fill:#dcfce7,stroke:#16a34a,stroke-
width:2px,color:#14532d
    classDef networking fill:#eff6ff,stroke:#2563eb,stroke-
width:1.5px,color:#1e3a8a
    classDef dns fill:#ede9fe,stroke:#7c3aed,stroke-
width:1.5px,color:#4c1d95
    classDef secrets fill:#fce7f3,stroke:#db2777,stroke-
width:1.5px,color:#831843
    classDef tfCloud fill:#f3e8ff,stroke:#9333ea,stroke-
width:2px,color:#581c87
    classDef dataFlow fill:#d1fae5,stroke:#059669,stroke-
width:1.5px,color:#064e3b
    classDef module fill:#ffedd5,stroke:#ea580c,stroke-
width:2px,color:#7c2d12,stroke-dasharray:5

    %% ============================================================
    %% TERRAFORM CLOUD
    %% ============================================================
    subgraph TFC["Terraform Cloud - signalroom org"]
        direction LR
        TFC_WORKSPACE["Workspace
iac-cc-aws-privatelink-
cluster-linking-example"]
        TFC_AGENT_POOL["Agent Pool
signalroom-iac-tfc-agents-pool"]
        TFC_WORKSPACE --- TFC_AGENT_POOL
    end
    TFC_WORKSPACE:::tfCloud
    TFC_AGENT_POOL:::tfCloud

    %% ============================================================
    %% CONFLUENT CLOUD
    %% ============================================================
    subgraph CC["Confluent Cloud"]
        direction TB
```

```
        CC_ENV["Environment: non-prod
Stream Governance: Essentials"]:::confluentCloud

        subgraph CC_SR["Schema Registry"]
            SR_CLUSTER["Schema Registry Cluster
src_api service account
DeveloperRead + DeveloperWrite"]:::confluentCloud
        end

        CC_PLATT["PrivateLink Attachment
AWS region"]:::confluentCloud

        subgraph CC_SANDBOX["Sandbox Cluster - Enterprise / HIGH"]
            direction TB
            SANDBOX_CLUSTER["sandbox_cluster
Enterprise, High Availability"]:::confluentCloud
            SANDBOX_TOPIC["Topic: dev-stock_trades"]:::dataFlow
            SANDBOX_DATAGEN["DataGen Source Connector
STOCK_TRADES to AVRO"]:::dataFlow
            SANDBOX_SA["Service Accounts
app_manager, app_producer
app_consumer, app_connector"]:::confluentCloud
            SANDBOX_DATAGEN -->|produces| SANDBOX_TOPIC
        end

        subgraph CC_SHARED["Shared Cluster - Enterprise / HIGH"]
            direction TB
            SHARED_CLUSTER["shared_cluster
Enterprise, High Availability"]:::confluentCloud
            MIRROR_TOPIC["Mirror Topic
dev-stock_trades"]:::dataFlow
            SHARED_SA["Service Accounts
app_manager, app_consumer"]:::confluentCloud
        end

        subgraph CC_CL["Cluster Linking - Bidirectional"]
            CL_LINK["Bidirectional Cluster Link
sandbox to shared
sandbox_cluster_linking_app_manager
shared_cluster_linking_app_manager"]:::confluentCloud
        end

        CC_ENV --> CC_PLATT
        CC_ENV --> SANDBOX_CLUSTER
        CC_ENV --> SHARED_CLUSTER
        CC_ENV --> SR_CLUSTER
        SANDBOX_TOPIC -.->|mirror via cluster link| MIRROR_TOPIC
        CL_LINK --- SANDBOX_CLUSTER
        CL_LINK --- SHARED_CLUSTER
    end

    %% ============================================================
    %% AWS ACCOUNT
    %% ============================================================
```

```
    subgraph AWS["AWS Account"]
        direction TB

        TGW["Transit Gateway
Central hub for all VPC routing"]:::networking
        TGW_RT["TGW Route Table
Associations + Propagations"]:::networking
        TGW --- TGW_RT

        subgraph SANDBOX_VPC["module: sandbox_vpc_privatelink - VPC
10.0.0.0/20"]
            direction TB
            SBX_SUBNETS["3x Private Subnets
across AZs"]:::networking
            SBX_RT["Route Tables
to TFC Agent VPC
to VPN Client VPC
to VPN VPC, DNS VPC"]:::networking
            SBX_SG["Security Group
Ingress: 443, 9092, 53"]:::networking
            SBX_VPCE["VPC Endpoint
Interface type, PrivateLink"]:::networking
            SBX_TGW_ATT["TGW Attachment"]:::networking
            SBX_R53_ASSOC["PHZ Association"]:::dns
            SBX_PLATTC["PL Attachment Connection
confluent_private_link_
attachment_connection"]:::networking
            SBX_SUBNETS --> SBX_VPCE
            SBX_SG --> SBX_VPCE
        end

        subgraph SHARED_VPC["module: shared_vpc_privatelink - VPC
10.1.0.0/20"]
            direction TB
            SHR_SUBNETS["3x Private Subnets
across AZs"]:::networking
            SHR_RT["Route Tables
to TFC Agent VPC
to VPN Client VPC
to VPN VPC, DNS VPC"]:::networking
            SHR_SG["Security Group
Ingress: 443, 9092, 53"]:::networking
            SHR_VPCE["VPC Endpoint
Interface type, PrivateLink"]:::networking
            SHR_TGW_ATT["TGW Attachment"]:::networking
            SHR_R53_ASSOC["PHZ Association"]:::dns
            SHR_PLATTC["PL Attachment Connection
confluent_private_link_
attachment_connection"]:::networking
            SHR_SUBNETS --> SHR_VPCE
            SHR_SG --> SHR_VPCE
        end

        TFC_AGENT_VPC["TFC Agent VPC
```

```
pre-existing"]:::vpcBox
        VPN_VPC["VPN VPC
pre-existing"]:::vpcBox
        DNS_VPC["DNS VPC
pre-existing"]:::vpcBox

        subgraph R53["Route53 - Centralized DNS"]
            direction TB
            R53_PHZ["Private Hosted Zone
region.aws.private.confluent.cloud"]:::dns
            R53_WILDCARD["Wildcard CNAME
*.dns_domain"]:::dns
            R53_ZONAL["Zonal CNAMEs
*.az-id.dns_domain"]:::dns
            R53_RESOLVER["System Resolver Rule
confluent_private domain"]:::dns
            R53_PHZ --> R53_WILDCARD
            R53_PHZ --> R53_ZONAL
        end

        subgraph SM["AWS Secrets Manager"]
            direction LR
            SM_SR["Schema Registry
API credentials"]:::secrets
            SM_SANDBOX["Sandbox Cluster
manager, consumer, producer"]:::secrets
            SM_SHARED["Shared Cluster
manager, consumer"]:::secrets
        end

        SBX_TGW_ATT ---|attach| TGW
        SHR_TGW_ATT ---|attach| TGW
        TFC_AGENT_VPC ---|attach| TGW
        VPN_VPC ---|attach| TGW
        DNS_VPC ---|attach| TGW

        R53_PHZ -.->|associate| TFC_AGENT_VPC
        R53_PHZ -.->|associate| DNS_VPC
        R53_PHZ -.->|associate| VPN_VPC
        R53_PHZ -.->|associate| SBX_R53_ASSOC
        R53_PHZ -.->|associate| SHR_R53_ASSOC

        R53_RESOLVER -.->|rule assoc| TFC_AGENT_VPC
        R53_RESOLVER -.->|rule assoc| DNS_VPC
        R53_RESOLVER -.->|rule assoc| VPN_VPC
        R53_RESOLVER -.->|rule assoc| SANDBOX_VPC
        R53_RESOLVER -.->|rule assoc| SHARED_VPC
    end

    %% ============================================================
    %% CROSS-BOUNDARY CONNECTIONS
    %% ============================================================
    SBX_VPCE ==>|AWS PrivateLink| CC_PLATT
    SHR_VPCE ==>|AWS PrivateLink| CC_PLATT
```

```
    SBX_PLATTC -.->|registers connection| CC_PLATT
    SHR_PLATTC -.->|registers connection| CC_PLATT

    TFC_AGENT_POOL -.->|runs on| TFC_AGENT_VPC

    subgraph API_KEY_ROT["API Key Rotation Module"]
        direction LR
        AKR["iac-confluent-api_key_rotation-tf_module
Auto-rotates keys per day_count schedule"]:::module
    end

    AKR -.->|manages keys for| SANDBOX_SA
    AKR -.->|manages keys for| SHARED_SA
    AKR -.->|manages keys for| SR_CLUSTER
    AKR -.->|manages keys for| CL_LINK

    AKR -.->|stores credentials| SM
```

This Terraform configuration demonstrates how to build a fully private, production-grade connectivity architecture between AWS and Confluent Cloud using AWS PrivateLink and Confluent Cluster Linking. It addresses a key architectural constraint: ***Confluent PrivateLink attachments share a non-unique DNS namespace, while AWS Route 53 prevents associating multiple Private Hosted Zones (PHZs) with the same domain name across overlapping VPC associations. As a result, separate PHZs cannot be created per cluster and distributed across interconnected VPCs***.

To resolve this, **the repo implements a centralized PHZ shared across all participating VPCs**, using wildcard and zonal CNAME records to route traffic to the appropriate interface endpoints. This ensures deterministic DNS resolution and enables scalable multi-cluster connectivity across the network fabric.

The configuration provisions a non-production Confluent Cloud environment with two Enterprise-tier, highly available Kafka clusters:

- a sandbox cluster that ingests simulated stock trade events via a DataGen source connector
- a shared cluster that receives those events through bidirectional Cluster Linking with automatic mirror topic synchronization

On the AWS side, the deployment creates two dedicated multi-AZ VPCs with private subnets, each connected to Confluent Cloud through PrivateLink interface endpoints so all Kafka traffic remains off the public internet. A Transit Gateway hub integrates these VPCs with existing VPN and DNS infrastructure, while Route 53 Resolver rules ensure seamless name resolution across all spoke networks.

Additional capabilities include Schema Registry with Stream Governance Essentials, automated API key rotation with credentials stored in AWS Secrets Manager, and agent-based execution via Terraform Cloud. The result is a complete, reproducible reference architecture for securely operating multiple Confluent Cloud clusters over PrivateLink at scale on AWS.

Below is the Terraform resource visualization of the infrastructure that's created:

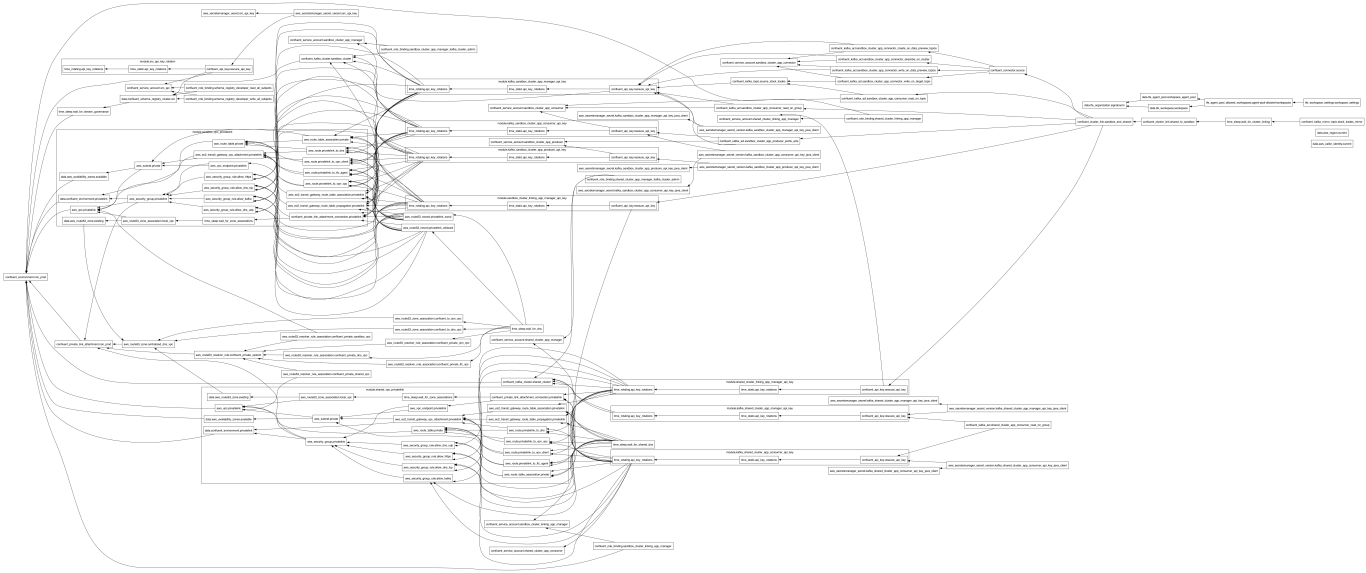

Table of Contents

1.0 Prerequisites

This project assumes you have the following prerequisites in place:

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

1.1 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
```

1.1.1 Key Features Required for Confluent PrivateLink to Work

1.1.1.1 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

1.1.1.2 Centralized DNS Resolution (Critical for PrivateLink)

- **Dedicated DNS VPC** with Route53 Inbound Resolver endpoints
- **Private Hosted Zones** for `*.aws.confluent.cloud` domain
- DNS forwarding rules route Confluent queries from all VPCs to the central DNS VPC
- Route53 Outbound Resolver in VPN VPC forwards to DNS VPC resolver IPs

1.1.1.3 DNS Forwarding Chain (as documented in your outputs)

1. Client queries `lkc-xxxxx.us-east-1.aws.private.confluent.cloud`
2. VPN VPC's default DNS forwards to Route53 Outbound Resolver
3. Outbound Resolver forwards to DNS VPC Inbound Resolver
4. DNS VPC checks Private Hosted Zones → returns VPC Endpoint private IPs

1.1.1.4 VPC Endpoints (AWS PrivateLink)

- VPC Endpoints in workload VPCs connecting to Confluent's PrivateLink service
- Security groups allowing traffic from authorized sources (VPN clients, TFC agents)

1.1.1.5 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

1.1.1.6 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

1.1.1.7 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

1.2 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
```

1.2.1 Key Features Required for Confluent PrivateLink to Work (TFC Agent Configuration)

1.2.1.1 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

1.2.1.2 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, Client VPN VPC, and all Workload VPCs containing PrivateLink endpoints
- Flattened route map pattern (`for_each`) ensures routes are created for every workload VPC CIDR

1.2.1.3 Security Group Configuration for Kafka/PrivateLink 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
- **Confluent PrivateLink Security Group** allowing inbound from TFC Agent SG on ports 443 and 9092

1.2.1.4 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

1.2.1.5 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

1.2.1.6 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

1.2.1.7 Network Architecture Summary

- **Hub-and-spoke model**: TGW connects TFC Agent VPC → DNS VPC → Workload VPCs
- **DNS resolution chain**: TFC Agent → Custom DHCP → Centralized DNS VPC → Private Hosted Zones → PrivateLink Endpoint IPs
- **Traffic flow**: TFC Agent → TGW → Workload VPC → PrivateLink Endpoint → Confluent Cloud Kafka

2.0 Project's Architecture Overview

2.1 Key Features Required for Confluent PrivateLink to Work (Confluent Cloud Configuration)

2.1.1 Confluent Private Link Attachment (Environment-Level)

- Single `confluent_private_link_attachment` resource created at the environment level for AWS region
- Provides the `vpc_endpoint_service_name` that AWS VPC Endpoints connect to
- Provides the `dns_domain` (e.g., `*.aws.private.confluent.cloud`) for DNS configuration
- Multiple VPCs can share the same PrivateLink attachment via separate VPC Endpoints

2.1.2 AWS VPC Endpoint Configuration

- Interface VPC Endpoints (`vpc_endpoint_type = "Interface"`) in each workload VPC
- **Critical**: `private_dns_enabled = false` — DNS handled via centralized Private Hosted Zones instead
- Security groups allowing inbound on ports 443 (HTTPS), 9092 (Kafka), and 53 (DNS) from TFC Agent VPC, VPN VPC, VPN Client CIDR, and local VPC CIDR
- Endpoints deployed across multiple AZs (3 subnets) for high availability

2.1.3 Confluent Private Link Attachment Connection

- `confluent_private_link_attachment_connection` links the AWS VPC Endpoint ID to the Confluent PrivateLink attachment
- Creates the bidirectional connection between AWS and Confluent Cloud
- Depends on Route53 zone associations being complete first (`time_sleep` for propagation)

2.1.4 Centralized Private Hosted Zone (PHZ) Strategy

- Single PHZ created for the Confluent DNS domain, associated with **all VPCs** that need access
- **Zonal CNAME records**: `*.{availability-zone-id}.{dns_domain}` → AZ-specific VPC Endpoint DNS
- **Wildcard CNAME record**: `*.{dns_domain}` → Primary VPC Endpoint DNS

2.1.5 Route53 SYSTEM Resolver Rule

- `rule_type = "SYSTEM"` tells Route53 to use Private Hosted Zones for the Confluent domain
- Rule associated with every VPC that needs Confluent access

2.1.6 Transit Gateway Routing

- Each PrivateLink VPC attached to TGW with DNS support enabled
- Route table association AND route propagation configured
- Routes added from PrivateLink VPCs back to all consumer VPCs

2.1.7 Multi-Cluster Architecture with Cluster Linking

- Two Enterprise Kafka clusters (Sandbox and Shared) in the same environment
- Bidirectional Cluster Link with mirror topics for data replication

2.1.8 Service Account & API Key Management

- Separate service accounts per role with API key rotation

- ACLs granting specific permissions per service account
- API keys stored in AWS Secrets Manager

2.1.9 DNS Propagation Timing

- `time_sleep` resources ensuring DNS propagates before dependent resources (1-2 minutes)

2.1.10 Schema Registry Integration

- Stream Governance (Essentials) enabled at environment level with AVRO support

3.0 Let's Get Started

3.1 Deploy the Infrastructure

The deploy.sh script handles authentication and Terraform execution:

```
./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-cidr=<TFC_AGENT_VPC_CIDR> \
  --dns-vpc-id=<DNS_VPC_ID> \
  --vpn-vpc-id=<VPN_VPC_ID> \
  --vpn-vpc-cidr=<VPN_VPC_CIDR> \
  --vpn-client-vpc-cidr=<VPN_CLIENT_VPC_CIDR> \
  [--dns-vpc-cidr=<DNS_VPC_CIDR>] \     `# Default: 10.255.0.0/24`
  [--day-count=<DAY_COUNT>]             `# Default: 30 (API key rotation
interval)`
```

Argument	Required	Default	Description
`create`	✅	—	The command to execute. `create` deploys the infrastructure via `terraform apply`.
`--profile`	✅	—	The AWS SSO profile name. Passed directly to `aws sso login` and `aws2-wrap` for authentication, and used to resolve `AWS_REGION`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`, which are then exported as `TF_VAR_aws_region`, `TF_VAR_aws_access_key_id`, `TF_VAR_aws_secret_access_key`, and `TF_VAR_aws_session_token` for Terraform, respectively.

Argument	Required	Default	Description
`--confluent-api-key`	✅	—	Confluent Cloud API key. Exported as `TF_VAR_confluent_api_key` for Terraform.
`--confluent-api-secret`	✅	—	Confluent Cloud API secret. Exported as `TF_VAR_confluent_api_secret` for Terraform.
`--tfe-token`	✅	—	Terraform Enterprise/Cloud API token. Exported as `TF_VAR_tfe_token` — used for authenticating the TFC Agent or remote backend.
`--tgw-id`	✅	—	AWS Transit Gateway ID. Exported as `TF_VAR_tgw_id` for routing between VPCs.
`--tgw-rt-id`	✅	—	AWS Transit Gateway Route Table ID. Exported as `TF_VAR_tgw_rt_id` for associating route entries.
`--tfc-agent-vpc-id`	✅	—	VPC ID where the Terraform Cloud Agent resides. Exported as `TF_VAR_tfc_agent_vpc_id`.
`--tfc-agent-vpc-cidr`	✅	—	CIDR block of the TFC Agent VPC. Exported as `TF_VAR_tfc_agent_vpc_cidr` — used for security group and routing rules.
`--dns-vpc-id`	✅	—	VPC ID hosting the DNS infrastructure (Route 53 Resolver endpoints). Exported as `TF_VAR_dns_vpc_id`.
`--vpn-vpc-id`	✅	—	VPC ID where the AWS Client VPN endpoint is deployed. Exported as `TF_VAR_vpn_vpc_id`.
`--vpn-vpc-cidr`	✅	—	CIDR block of the VPN VPC. Exported as `TF_VAR_vpn_vpc_cidr`.
`--vpn-client-vpc-cidr`	✅	—	Client-side CIDR range assigned to VPN clients. Exported as `TF_VAR_vpn_client_vpc_cidr`.
`--dns-vpc-cidr`	❌	`10.255.0.0/24`	CIDR block of the DNS VPC. Exported as `TF_VAR_dns_vpc_cidr`.
`--day-count`	❌	`30`	API key rotation interval in days. Exported as `TF_VAR_day_count`.

3.1.1 Handling DNS Resolution Errors

If you encounter DNS resolution errors during the apply process, simply re-run the `deploy.sh` script with the `create` command.

3.1.2 Cluster Linking Error

```
|
| Error: error creating Cluster Link: 400 Bad Request: A cluster link
already exists with the provided link name: Cluster Link
_fA8DRTZSvGrLkTur7e8-Q already exists.
|
|   with confluent_cluster_link.shared_to_sandbox,
|   on setup-confluent-cluster_linking.tf line 113, in resource
"confluent_cluster_link" "shared_to_sandbox":
|  113: resource "confluent_cluster_link" "shared_to_sandbox" {
|
|
```

If you see the above error, it indicates that the previous failed attempt left the cluster link in place. To resolve, delete the existing cluster link via the Confluent CLI:

```
confluent kafka link delete bidirectional_between_sandbox_and_shared --
cluster <SANDBOX_CLUSTER_ID> --environment <NON_PROD_ENV_ID> --force
```

Then re-run the `deploy.sh` script with the `create` command.

3.2 Teardown the Infrastructure

```
./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-cidr=<TFC_AGENT_VPC_CIDR> \
  --dns-vpc-id=<DNS_VPC_ID> \
  --vpn-vpc-id=<VPN_VPC_ID> \
  --vpn-vpc-cidr=<VPN_VPC_CIDR> \
  --vpn-client-vpc-cidr=<VPN_CLIENT_VPC_CIDR>
```

Argument	Required	Default	Description
`destroy`	✅	—	The command to execute. `destroy` tears it down via `terraform destroy` and force-deletes associated AWS Secrets Manager secrets.

Argument	Required	Default	Description
`--profile`	✅	—	The AWS SSO profile name. Passed directly to aws sso login and aws2-wrap for authentication, and used to resolve AWS_REGION, AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN, which are then exported as TF_VAR_aws_region, TF_VAR_aws_access_key_id, TF_VAR_aws_secret_access_key, and TF_VAR_aws_session_token for Terraform, respectively.
`--confluent-api-key`	✅	—	Confluent Cloud API key. Exported as `TF_VAR_confluent_api_key` for Terraform.
`--confluent-api-secret`	✅	—	Confluent Cloud API secret. Exported as `TF_VAR_confluent_api_secret` for Terraform.
`--tfe-token`	✅	—	Terraform Enterprise/Cloud API token. Exported as `TF_VAR_tfe_token` — used for authenticating the TFC Agent or remote backend.
`--tgw-id`	✅	—	AWS Transit Gateway ID. Exported as `TF_VAR_tgw_id` for routing between VPCs.
`--tgw-rt-id`	✅	—	AWS Transit Gateway Route Table ID. Exported as `TF_VAR_tgw_rt_id` for associating route entries.
`--tfc-agent-vpc-id`	✅	—	VPC ID where the Terraform Cloud Agent resides. Exported as `TF_VAR_tfc_agent_vpc_id`.
`--tfc-agent-vpc-cidr`	✅	—	CIDR block of the TFC Agent VPC. Exported as `TF_VAR_tfc_agent_vpc_cidr` — used for security group and routing rules.
`--dns-vpc-id`	✅	—	VPC ID hosting the DNS infrastructure (Route 53 Resolver endpoints). Exported as `TF_VAR_dns_vpc_id`.
`--vpn-vpc-id`	✅	—	VPC ID where the AWS Client VPN endpoint is deployed. Exported as `TF_VAR_vpn_vpc_id`.
`--vpn-vpc-cidr`	✅	—	CIDR block of the VPN VPC. Exported as `TF_VAR_vpn_vpc_cidr`.
`--vpn-client-vpc-cidr`	✅	—	Client-side CIDR range assigned to VPN clients. Exported as `TF_VAR_vpn_client_vpc_cidr`.

3.2.1 Handling Cluster Link Deletion Error(s)

If you encounter a Cluster Link deletion error during the destroy process, you may see an error message similar to the following:

```
| Error: error deleting Cluster Link "lkc-
m1ox27/bidirectional_between_sandbox_and_shared": 401 Unauthorized: Not
authorized: the authenticated user didn't have the right access to the
resource: Cluster authorization failed.
|
|
|
```

Navigate to the Terraform directory:

```
cd terraform
```

Remove the unreachable resources from the Terraform state:

```
terraform state rm 'confluent_cluster_link.sandbox_and_shared'
terraform state rm 'confluent_cluster_link.shared_to_sandbox'
```

Navigate back to the root directory:

```
cd ..
```

Rerun the `deploy.sh` script with the `destroy` command.

3.2.2 Handling DNS Resolution Errors

If you encounter DNS resolution errors during the destroy process, you may see error messages similar to the following:

```
| Error: error deleting Kafka ACLs "lkc-y08xyo/TOPIC#dev-
stock_trades#LITERAL#User:sa-81j9gv0#*#DESCRIBE#ALLOW": Delete
"https://lkc-y08xyo.us-east-
1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-y08xyo/acls?
host=%2A&operation=DESCRIBE&pattern_type=LITERAL&permission=ALLOW&principa
l=User%3Asa-81j9gv0&resource_name=dev-stock_trades&resource_type=TOPIC":
dial tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no such
host
|
|
|

|
```

```
│ Error: error deleting Kafka ACLs "lkc-y08xyo/TOPIC#dev-
stock_trades#LITERAL#User:sa-zmrv663#*#READ#ALLOW": Delete "https://lkc-
y08xyo.us-east-1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-
y08xyo/acls?
host=%2A&operation=READ&pattern_type=LITERAL&permission=ALLOW&principal=Us
er%3Asa-zmrv663&resource_name=dev-stock_trades&resource_type=TOPIC": dial
tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no such host
│
│
╵

╷
│ Error: error deleting Kafka ACLs "lkc-
y08xyo/GROUP#sandbox_aws_privatelink_example_#LITERAL#User:sa-
zmrv663#*#READ#ALLOW": Delete "https://lkc-y08xyo.us-east-
1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-y08xyo/acls?
host=%2A&operation=READ&pattern_type=LITERAL&permission=ALLOW&principal=Us
er%3Asa-
zmrv663&resource_name=sandbox_aws_privatelink_example_&resource_type=GROUP
": dial tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no
such host
│
│
╵

╷
│ Error: error deleting Kafka ACLs "lkc-
y08xyo/TOPIC#sandbox_aws_privatelink_example_#LITERAL#User:sa-
w7xq219#*#WRITE#ALLOW": Delete "https://lkc-y08xyo.us-east-
1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-y08xyo/acls?
host=%2A&operation=WRITE&pattern_type=LITERAL&permission=ALLOW&principal=U
ser%3Asa-
w7xq219&resource_name=sandbox_aws_privatelink_example_&resource_type=TOPIC
": dial tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no
such host
│
│
╵

╷
│ Error: error deleting Kafka ACLs "lkc-y08xyo/CLUSTER#kafka-
cluster#LITERAL#User:sa-w7xq219#*#DESCRIBE#ALLOW": Delete "https://lkc-
y08xyo.us-east-1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-
y08xyo/acls?
host=%2A&operation=DESCRIBE&pattern_type=LITERAL&permission=ALLOW&principa
l=User%3Asa-w7xq219&resource_name=kafka-cluster&resource_type=CLUSTER":
dial tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no such
host
│
│
╵

╷
│ Error: error deleting Kafka ACLs "lkc-y08xyo/TOPIC#dev-
stock_trades#LITERAL#User:sa-w7xq219#*#WRITE#ALLOW": Delete "https://lkc-
y08xyo.us-east-1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-
y08xyo/acls?
host=%2A&operation=WRITE&pattern_type=LITERAL&permission=ALLOW&principal=U
```

```
ser%3Asa-w7xq219&resource_name=dev-stock_trades&resource_type=TOPIC": dial
tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no such host
│
│
╵

╷
│ Error: error deleting Kafka ACLs "lkc-
y08xyo/TOPIC#sandbox_aws_privatelink_example_#LITERAL#User:sa-
w7xq219#*#CREATE#ALLOW": Delete "https://lkc-y08xyo.us-east-
1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-y08xyo/acls?
host=%2A&operation=CREATE&pattern_type=LITERAL&permission=ALLOW&principal=
User%3Asa-
w7xq219&resource_name=sandbox_aws_privatelink_example_&resource_type=TOPIC
": dial tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no
such host
│
│
╵

╷
│ Error: error deleting Kafka ACLs "lkc-y08xyo/TOPIC#dev-
stock_trades#LITERAL#User:sa-81j9gv0#*#READ#ALLOW": Delete "https://lkc-
y08xyo.us-east-1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-
y08xyo/acls?
host=%2A&operation=READ&pattern_type=LITERAL&permission=ALLOW&principal=Us
er%3Asa-81j9gv0&resource_name=dev-stock_trades&resource_type=TOPIC": dial
tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no such host
│
│
╵

╷
│ Error: error deleting Kafka ACLs "lkc-y08xyo/TOPIC#dev-
stock_trades#LITERAL#User:sa-81j9gv0#*#WRITE#ALLOW": Delete "https://lkc-
y08xyo.us-east-1.aws.private.confluent.cloud:443/kafka/v3/clusters/lkc-
y08xyo/acls?
host=%2A&operation=WRITE&pattern_type=LITERAL&permission=ALLOW&principal=U
ser%3Asa-81j9gv0&resource_name=dev-stock_trades&resource_type=TOPIC": dial
tcp: lookup lkc-y08xyo.us-east-1.aws.private.confluent.cloud: no such host

Operation failed: failed running terraform apply (exit 1)
```

Navigate to the Terraform directory:

```
cd terraform
```

Remove the unreachable resources from the Terraform state:

```
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_connector_describe_on_cluster'
terraform state rm
```

```
'confluent_kafka_acl.sandbox_cluster_app_connector_write_on_target_topic'
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_connector_create_on_data_preview_
topics'
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_connector_write_on_data_preview_t
opics'
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_producer_prefix_acls["DESCRIBE"]'
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_producer_prefix_acls["READ"]'
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_producer_prefix_acls["WRITE"]'
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_consumer_read_on_topic'
terraform state rm
'confluent_kafka_acl.sandbox_cluster_app_consumer_read_on_group'
terraform state rm 'confluent_kafka_topic.source_stock_trades'
```

Navigate back to the root directory:

```
cd ..
```

Rerun the `deploy.sh` script with the `destroy` command.

4.0 Resources

4.1 Terminology

- **PHZ**: Private Hosted Zone - AWS Route 53 Private Hosted Zone is a DNS service that allows you to create and manage private DNS zones within your VPCs.
- **TFC**: Terraform Cloud - A service that provides infrastructure automation using Terraform.
- **VPC**: Virtual Private Cloud - A virtual network dedicated to your AWS account.
- **AWS**: Amazon Web Services - A comprehensive cloud computing platform provided by Amazon.
- **CC**: Confluent Cloud - A fully managed event streaming platform based on Apache Kafka.
- **PL**: PrivateLink - An AWS service that enables private connectivity between VPCs and services.
- **IaC**: Infrastructure as Code - The practice of managing and provisioning computing infrastructure through machine-readable definition files.

4.2 Related Documentation

- AWS PrivateLink Overview in Confluent Cloud
- Use AWS PrivateLink for Serverless Products on Confluent Cloud
- GitHub Sample Project for Confluent Terraform Provider PrivateLink Attachment
- Geo-replication with Cluster Linking on Confluent Cloud
- Use the Confluent Cloud Console with Private Networking
- IP Filtering on Confluent Cloud
- AWS/Azure PrivateLink Networking Course

- Hands On: Configuring a PrivateLink Cluster