# Deep-Dive Addendum: IaC & CI/CD Recipes

This addendum provides detailed insights and practical recipes for implementing Infrastructure as Code (IaC) and Continuous Integration/Continuous Delivery (CI/CD) within your enterprise data platform. These practices are crucial for ensuring maintainability, collaboration, consistency, and automated delivery of your data solutions.

## 5.1. Project Structure & Infrastructure as Code (IaC)

A well-organized project structure and the adoption of Infrastructure as Code (IaC) are crucial for maintainability, collaboration, and consistent deployments.

**Mono-repo Skeleton:** A mono-repo approach centralizes all project components, enhancing discoverability and simplifying dependency management.

```
data-ingestion-platform/
├── .github/              # GitHub Actions CI/CD workflows
│   └── workflows/
│       ├── ci.yml        # Continuous Integration pipeline
│       └── release.yml   # Release/Deployment pipeline
├── data/                 # Persistent Docker volumes for all services
│   ├── postgres/
│   ├── mongodb/
│   ├── minio/
│   ├── spark-events/
│   ├── grafana/
│   ├── openmetadata_mysql/
│   └── openmetadata_elasticsearch/
├── src/                  # Core Python application logic (e.g., FastAPI, common utils)
│   └── common/
│       └── utils.py
│   └── models/           # Pydantic/Avro schemas for data contracts
│       └── financial_transaction.py
│       └── insurance_claim.py
├── fastapi_app/          # FastAPI ingestion service
│   ├── Dockerfile
│   ├── requirements.txt
│   └── app/
│       └── main.py       # Entry point for FastAPI app
│   └── tests/
│       ├── unit/
│       │   └── test_api.py
```

```
│       └── integration/
│           └── test_data_flow.py # Integration tests
├── pyspark_jobs/          # Apache Spark transformation jobs (PySpark)
│   ├── __init__.py
│   ├── batch_transformations.py
│   └── streaming_consumer.py
│   └── tests/
│       └── unit/
│           └── test_spark_logic.py
├── airflow_dags/          # Apache Airflow DAG definitions
│   ├── data_ingestion_dag.py
│   └── data_transformation_dag.py
├── terraform_infra/       # Infrastructure as Code for cloud deployments
│   ├── modules/           # Reusable Terraform modules
│   │   ├── s3_data_lake/
│   │   ├── msk_kafka/
│   │   └── rds_postgres/
│   ├── environments/      # Environment-specific Terraform configurations
│   │   ├── dev/
│   │   │   └── main.tf
│   │   │   └── variables.tf
│   │   ├── staging/
│   │   │   └── main.tf
│   │   │   └── variables.tf
│   │   └── prod/
│   │       └── main.tf
│   │       └── variables.tf
├── observability/         # Grafana dashboards, Grafana Alloy configurations, Prometheus
rules
│   ├── alloy-config.river
│   ├── dashboards/
│   │   └── health_dashboard.json
│   └── grafana_dashboards_provisioning/
│   └── grafana_datasources_provisioning/
├── openmetadata_ingestion_scripts/ # Python scripts for OpenMetadata connectors
├── runbooks/              # Operational Runbooks library
│   ├── kafka_consumer_lag.md
│   └── spark_job_hang.md
├── conceptual_code/       # Contains conceptual snippets from document for quick
reference
├── docker-compose.yml     # Central Docker Compose file for local environment
├── docker-compose.test.yml  # Docker Compose file for integration testing
└── README.md
```

## 5.2. Security Best Practices & Secrets Management

Security is paramount, especially when handling sensitive financial and insurance data. This section, while broader than just IaC, is included here due to its strong ties to how infrastructure is provisioned and applications are deployed securely via CI/CD.

**Data Encryption:**

- **In Transit:** All data moving between services within the platform, and especially data ingested via the FastAPI API, should be encrypted using HTTPS/TLS. For Kafka, configure SSL/TLS (e.g., KAFKA_PROTOCOL: SSL in production).
- **At Rest:** Data stored in all persistence layers (PostgreSQL, MongoDB, MinIO/S3) must be encrypted. Locally, this relies on the host's disk encryption. In cloud environments, managed services (e.g., RDS, S3, DocumentDB) offer encryption at rest.

**Secure Credential Management:** Hardcoding sensitive information (passwords, API keys, tokens) is a critical vulnerability.

- **Local Development:** Use .env files (added to .gitignore) for environment variables or Docker secrets. Docker secrets are safer as they are mounted as files and not directly exposed as environment variables.
  *Example .env (.gitignore it!):*
  KAFKA_BROKER="localhost:9092"
  POSTGRES_USER="user"
  POSTGRES_PASSWORD="password"

- **Production (Cloud):** Employ dedicated, enterprise-grade secrets management solutions.

**Cloud Secrets Management Comparison:**

| Solution | Type | Strengths | Weaknesses | Usage Tips |
|---|---|---|---|---|
| **AWS Secrets Manager** | Cloud-Native | Fully managed, automated rotation, granular IAM policies, integrates with other AWS services. | AWS lock-in. | Use for most AWS-native applications. Implement rotation policies for databases. |
| **HashiCorp Vault** | Vendor-Neutral | Strong audit logging, dynamic secrets (on-demand credentials), robust access controls, supports multiple | Requires self-management (or Vault Enterprise), steeper learning curve. | Run Vault Agent as a sidecar in Kubernetes/ECS to inject secrets. Implement transit encryption. |

| | | | | |
|---|---|---|---|---|
| | | backends. | | |
| **Doppler** | SaaS | Centralized secrets management for multiple environments, easy integration with CI/CD. | SaaS dependency, potential for vendor lock-in. | Good for smaller teams or those prioritizing ease of use and cross-environment consistency. |
| **SOPS (Secrets OPerationS)** | Open-Source | Encrypts secrets in Git (YAML, JSON), works well with GitOps, easy CLI. | Less dynamic than Vault, requires key management (KMS, GPG). | Ideal for encrypting static configuration secrets in Git repos (e.g., Kubernetes manifests). |

**Real-World Usage Tips:**

- **Vault Agent Sidecar:** In containerized environments (Kubernetes, ECS), run a Vault Agent as a sidecar container. It can pull secrets from Vault and render them to a shared volume, making them available to the main application container as files (more secure than env vars).
- **Rotation Policies:** Implement automated secret rotation for database credentials, API keys, etc., to minimize the window of compromise.
- **Least Privilege:** Ensure IAM roles/policies for services accessing secrets managers adhere strictly to the principle of least privilege.

# 5.3. CI/CD: Automating Quality and Delivery

A robust CI/CD pipeline is essential for automating the software development lifecycle, ensuring code quality, consistency, and rapid, reliable deployments.

- **Version Control:** All code (FastAPI, PySpark, Airflow DAGs, Dockerfiles, IaC) resides in a Git repository.
- **Automated Build & Test (Continuous Integration - CI):**
  - **Trigger:** On every code commit/pull request.
  - **Steps:** Linting (Black, Flake8), static analysis (SonarQube), unit tests (pytest), Docker image builds.
- **Automated Deployment (Continuous Delivery/Deployment - CD):**
  - **Development/Staging Environments:** Automatically deploy validated artifacts for further testing.
  - **Production Deployment:** Controlled process with manual approval gates, canary deployments, or blue/green strategies.
- **Infrastructure as Code (IaC):** Manage infrastructure (e.g., cloud resources via Terraform) as code within the Git repository and deploy via CI/CD.

**Conceptual GitHub Actions Release Workflow (.github/workflows/release.yml):**
This workflow demonstrates building, publishing, testing on staging, and conditionally promoting to production.

```yaml
# .github/workflows/release.yml
name: Release Pipeline

on:
  push:
    branches:
      - release # Trigger on pushes to a 'release' branch, or tag pushes
  workflow_dispatch: # Allows manual trigger from GitHub UI
    inputs:
      version:
        description: 'Release Version (e.g., v1.0.0)'
        required: true

jobs:
  build-and-publish-images:
    runs-on: ubuntu-latest
    outputs:
      fastapi_image: ${{ steps.build_fastapi.outputs.image_name }}
      pyspark_image: ${{ steps.build_pyspark.outputs.image_name }}
    steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Set up Docker BuildX
      uses: docker/setup-buildx-action@v2
    - name: Log in to Docker Hub (or ECR)
      uses: docker/login-action@v2
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_TOKEN }}
        # For ECR: registry: ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.${{ secrets.AWS_REGION
}}.amazonaws.com
    - name: Build and push FastAPI Ingestor image
      id: build_fastapi
      uses: docker/build-push-action@v4
      with:
        context: ./fastapi_app
        push: true
        tags: yourusername/fastapi-ingestor:${{ github.sha }} # Use Git SHA for unique tag
        # For ECR: tags: ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.${{ secrets.AWS_REGION
}}.amazonaws.com/fastapi-ingestor:${{ github.sha }}
```

```
      outputs: type=string,name=image_name
    - name: Build and push PySpark Job image (base for running jobs)
      id: build_pyspark
      uses: docker/build-push-action@v4
      with:
        context: ./pyspark_jobs # Assuming a Dockerfile here for PySpark environment
        push: true
        tags: yourusername/pyspark-job-runner:${{ github.sha }}
        outputs: type=string,name=image_name

  deploy-to-staging:
    needs: build-and-publish-images
    runs-on: ubuntu-latest
    environment: staging # Links to GitHub Environments
    steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Configure AWS Credentials (for IaC deployment)
      uses: aws-actions/configure-aws-credentials@v3
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: us-east-1
    - name: Set up Terraform
      uses: hashicorp/setup-terraform@v2
      with:
        terraform_version: 1.5.0 # Or desired version
    - name: Terraform Init (Staging)
      run: terraform -chdir=./terraform_infra/environments/staging init
    - name: Terraform Apply (Staging)
      run: terraform -chdir=./terraform_infra/environments/staging apply -auto-approve \
            -var="fastapi_image_tag=${{ needs.build-and-publish-images.outputs.fastapi_image
}}" \
            -var="pyspark_image_tag=${{
needs.build-and-publish-images.outputs.pyspark_image }}"
      env:
        TF_VAR_environment: staging # Pass environment variable to Terraform
    - name: Run End-to-End Smoke Tests on Staging
      # This would involve:
      # 1. Waiting for staging deployment to complete
      # 2. Triggering data generation against staging API Gateway
      # 3. Verifying data in S3/Delta Lake or triggering a Spark job
      # 4. Checking Grafana/CloudWatch for basic health metrics
```

```
    run: |
     echo "Running smoke tests on staging environment using deployed API and data lake."
     # Example: python scripts/run_smoke_tests.py --env staging --api-url ${{
secrets.STAGING_API_URL }}
     sleep 60 # Simulate test execution
     echo "Staging smoke tests passed."

 promote-to-production:
  needs: deploy-to-staging
  runs-on: ubuntu-latest
  environment: production # Links to GitHub Environments, requires manual approval
  if: success() && github.ref == 'refs/heads/release' # Only promote if staging passed and on
release branch
  steps:
  - name: Checkout code
   uses: actions/checkout@v3
  - name: Configure AWS Credentials (for IaC deployment)
   uses: aws-actions/configure-aws-credentials@v3
   with:
    aws-access-key-id: ${{ secrets.AWS_PROD_ACCESS_KEY_ID }} # Use production specific
credentials
    aws-secret-access-key: ${{ secrets.AWS_PROD_SECRET_ACCESS_KEY }}
    aws-region: us-east-1
  - name: Set up Terraform
   uses: hashicorp/setup-terraform@v2
   with:
    terraform_version: 1.5.0
  - name: Terraform Init (Production)
   run: terraform -chdir=./terraform_infra/environments/prod init
  - name: Terraform Apply (Production)
   run: terraform -chdir=./terraform_infra/environments/prod apply -auto-approve \
       -var="fastapi_image_tag=${{ needs.build-and-publish-images.outputs.fastapi_image
}}" \
       -var="pyspark_image_tag=${{
needs.build-and-publish-images.outputs.pyspark_image }}"
   env:
    TF_VAR_environment: prod
```

# 9.3. Hybrid Testing with LocalStack/ECS-Local

For "hybrid" testing, LocalStack or ECS-Local allows you to interact with local AWS-compatible APIs before full cloud cutover. This is a critical part of a robust CI/CD pipeline, enabling faster feedback loops and reduced cloud spend during development and testing phases.

**LocalStack:** A cloud service emulator that runs in your local environment.

- **Benefit:** Test cloud service integrations (S3, Lambda, SQS, SNS) without deploying to actual AWS, saving costs and speeding up feedback.
- **Usage:**
  - Run LocalStack (e.g., via Docker Compose).
  - Configure your Python boto3 clients to point to LocalStack's endpoint URL (e.g., s3 = boto3.client('s3', endpoint_url='http://localhost:4566')).
  - Test your application logic that interacts with these AWS services locally.

**ECS-Local:** A tool that allows you to test ECS task definitions locally without deploying to AWS.

- **Benefit:** Validate your ECS task definitions, Docker images, and container configurations in a local environment before pushing to Amazon ECS.
- **Usage:** Define your ECS task definitions as you would for AWS. Use the ecs-local CLI to run these tasks locally as Docker containers.

# Appendix I: AWS IaC Snippets

This appendix provides conceptual Terraform Infrastructure as Code (IaC) snippets for deploying various components of the data platform on AWS. These snippets demonstrate how the local Docker Compose setup can be translated into production-grade cloud infrastructure, forming a core part of your automated deployment pipeline.

## AWS Account and Core Networking Setup:

- **Prerequisites:** Active AWS account, AWS CLI configured, basic familiarity with AWS Console.
- **IAM (Identity and Access Management):** Create necessary IAM roles and policies with least privilege for all services and components (e.g., Lambda execution role, EMR instance profile, MWAA execution role).
- **VPC (Virtual Private Cloud):** Design and create a VPC with public and private subnets. Deploy a NAT Gateway in the public subnet for private subnet resources to access the internet. Configure appropriate Route Tables and Network ACLs.
- **Security Groups:** Create security groups for each service to control inbound and outbound traffic.

## Amazon S3 (Data Lake Storage - Replaces MinIO):

```
# S3 Data Lake Module (terraform_infra/modules/s3_data_lake/main.tf)
resource "aws_s3_bucket" "raw_data_bucket" {
  bucket = "${var.project_name}-raw-${var.environment}-${var.aws_region}"
  tags = {
    Environment = var.environment
    Project     = var.project_name
```

```
    ManagedBy   = "Terraform"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration"
"raw_data_bucket_encryption" {
  bucket = aws_s3_bucket.raw_data_bucket.id
  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

resource "aws_s3_bucket" "curated_data_bucket" {
  bucket = "${var.project_name}-curated-${var.environment}-${var.aws_region}"
  tags = {
    Environment = var.environment
    Project     = var.project_name
    ManagedBy   = "Terraform"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration"
"curated_data_bucket_encryption" {
  bucket = aws_s3_bucket.curated_data_bucket.id
  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

# Output bucket ARNs
output "raw_bucket_arn" {
  value = aws_s3_bucket.raw_data_bucket.arn
}

output "curated_bucket_arn" {
  value = aws_s3_bucket.curated_data_bucket.arn
}
```

## Amazon MSK (Managed Apache Kafka - Replaces Apache Kafka):

```
# MSK Kafka Cluster Module (terraform_infra/modules/msk_kafka/main.tf)
resource "aws_msk_cluster" "main" {
  cluster_name         = "${var.project_name}-kafka-${var.environment}"
  kafka_version        = "2.8.1" # Or latest stable
  number_of_broker_nodes = var.number_of_broker_nodes

  broker_node_group_info {
    instance_type = var.broker_instance_type
    ebs_volume_info {
      provisioned_throughput = 0 # For smaller clusters, adjust for higher IOPS
      volume_size          = var.broker_ebs_volume_size # GB
    }
    client_subnets = var.subnet_ids
    security_groups = [var.security_group_id]
  }

  encryption_info {
    encryption_in_transit {
      client_broker = "TLS"
      in_cluster    = true
    }
    # key_arn = aws_kms_key.kafka_kms.arn # Optional: for KMS encryption at rest
  }

  open_monitoring {
    prometheus {
      jmx_exporter {
        enabled_in_broker = true
      }
      node_exporter {
        enabled_in_broker = true
      }
    }
  }

  tags = {
    Environment = var.environment
    Project     = var.project_name
  }
}
```

```
# Output MSK broker endpoints
output "bootstrap_brokers_tls" {
  value = aws_msk_cluster.main.bootstrap_brokers_tls
}
```

## AWS Lambda + Amazon API Gateway (FastAPI Replacement):

```
# Lambda API Ingestor Module (terraform_infra/modules/lambda_api_ingestor/main.tf)
resource "aws_ecr_repository" "fastapi_repo" {
  name = "${var.project_name}/fastapi-ingestor"
}

# IAM Role for Lambda function
resource "aws_iam_role" "lambda_exec_role" {
  name = "${var.project_name}-lambda-fastapi-exec-role-${var.environment}"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "lambda.amazonaws.com"
      }
    }]
  })
}

resource "aws_iam_role_policy_attachment" "lambda_basic_exec" {
  role       = aws_iam_role.lambda_exec_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
}

resource "aws_iam_role_policy_attachment" "lambda_vpc_access" {
  role       = aws_iam_role.lambda_exec_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AWSLambdaVPCAccessExecutionRole"
}

# Policy to allow Lambda to publish to MSK (example)
resource "aws_iam_policy" "lambda_msk_publish" {
  name   = "${var.project_name}-lambda-msk-publish-policy-${var.environment}"
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
```

```
      Action = [
        "kafka-action:DescribeCluster",
        "kafka-action:GetBootstrapBrokers",
        "kafka-action:GetTopicPartitions",
        "kafka-action:ListTopics",
        "kafka-action:Produce"
      ]
      Effect   = "Allow"
      Resource = var.msk_cluster_arn
    }]
  })
}

resource "aws_iam_role_policy_attachment" "lambda_msk_publish_attach" {
  role       = aws_iam_role.lambda_exec_role.name
  policy_arn = aws_iam_policy.lambda_msk_publish.arn
}

resource "aws_lambda_function" "fastapi_ingestor_lambda" {
  function_name = "${var.project_name}-fastapi-ingestor-${var.environment}"
  package_type  = "Image"
  image_uri     = "${aws_ecr_repository.fastapi_repo.repository_url}:${var.fastapi_image_tag}"
  role          = aws_iam_role.lambda_exec_role.arn
  timeout       = 30 # seconds
  memory_size   = 512 # MB
  vpc_config {
    subnet_ids         = var.subnet_ids
    security_group_ids = [var.security_group_id]
  }
  environment {
    variables = {
      KAFKA_BROKER_ADDRESSES = var.msk_bootstrap_brokers_tls # From MSK output
      KAFKA_TOPIC            = var.kafka_topic_name
      # ... other FastAPI env vars
    }
  }
  tags = {
    Environment = var.environment
    Project     = var.project_name
  }
}

resource "aws_apigatewayv2_api" "http_api" {
```

```
  name          = "${var.project_name}-fastapi-http-api-${var.environment}"
  protocol_type = "HTTP"
}

resource "aws_apigatewayv2_integration" "lambda_integration" {
  api_id           = aws_apigatewayv2_api.http_api.id
  integration_type = "AWS_PROXY"
  integration_method = "POST"
  integration_uri  = aws_lambda_function.fastapi_ingestor_lambda.invoke_arn
}

resource "aws_apigatewayv2_route" "ingest_financial" {
  api_id    = aws_apigatewayv2_api.http_api.id
  route_key = "POST /ingest-financial-transaction"
  target    = "integrations/${aws_apigatewayv2_integration.lambda_integration.id}"
}

resource "aws_apigatewayv2_route" "ingest_insurance" {
  api_id    = aws_apigatewayv2_api.http_api.id
  route_key = "POST /ingest-insurance-claim"
  target    = "integrations/${aws_apigatewayv2_integration.lambda_integration.id}"
}

resource "aws_apigatewayv2_stage" "default" {
  api_id      = aws_apigatewayv2_api.http_api.id
  name        = "$default"
  auto_deploy = true
}

resource "aws_lambda_permission" "apigateway_lambda_permission" {
  statement_id  = "AllowAPIGatewayInvoke"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.fastapi_ingestor_lambda.function_name
  principal     = "apigateway.amazonaws.com"
  # The /*/* part is to allow all API Gateway methods
  # to invoke the Lambda
  source_arn    = "${aws_apigatewayv2_api.http_api.execution_arn}/*/*"
}

output "api_gateway_url" {
  value = aws_apigatewayv2_api.http_api.api_endpoint
}
```

## Amazon RDS for PostgreSQL (Relational Database - Replaces local PostgreSQL):

```
# RDS PostgreSQL Module (terraform_infra/modules/rds_postgres/main.tf)
resource "aws_db_instance" "main" {
  identifier          = "${var.project_name}-postgres-${var.environment}"
  engine              = "postgres"
  engine_version      = "15.3"
  instance_class      = var.instance_class
  allocated_storage   = var.allocated_storage_gb
  storage_type        = "gp2" # Or gp3 for higher performance
  db_name             = var.db_name
  username            = var.db_username
  password            = var.db_password # Use AWS Secrets Manager in production!
  port                = 5432
  vpc_security_group_ids = [var.security_group_id]
  db_subnet_group_name = var.db_subnet_group_name # Must be created separately
  skip_final_snapshot  = var.skip_final_snapshot
  multi_az            = var.multi_az_enabled # True for production
  publicly_accessible  = false

  tags = {
    Environment = var.environment
    Project     = var.project_name
  }
}

output "rds_endpoint" {
  value = aws_db_instance.main.address
}
```

## Amazon DocumentDB (MongoDB Compatible Database - Replaces local MongoDB):

Creation steps via Console or AWS CLI. Terraform resources aws_docdb_cluster, aws_docdb_cluster_instance would be used.

## Amazon EMR or AWS Glue (Spark Replacement):

### Option A: Amazon EMR (Managed Spark Clusters) - Conceptual EMR Cluster Definition:

```
# EMR Cluster Module
```

```
resource "aws_emr_cluster" "spark_cluster" {
  name          = "${var.project_name}-spark-cluster-${var.environment}"
  release_label = "emr-6.9.0" # Or latest stable
  applications  = ["Spark"]

  ec2_attributes {
    subnet_id                         = var.subnet_id
    instance_profile                  = aws_iam_instance_profile.emr_profile.arn
    emr_managed_master_security_group = var.master_sg_id
    emr_managed_slave_security_group  = var.slave_sg_id
  }

  master_instance_group {
    instance_type = var.master_instance_type
    instance_count = 1
  }

  core_instance_group {
    instance_type = var.core_instance_type
    instance_count = var.core_instance_count
  }

  configurations_json = jsonencode([
    {
      Classification = "spark-defaults",
      Properties = {
        "spark.jars.packages" =
"io.delta:delta-core_2.12:2.4.0,org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0",
        "spark.sql.extensions" = "io.delta.sql.DeltaSparkSessionExtension",
        "spark.sql.catalog.spark_catalog" = "org.apache.spark.sql.delta.catalog.DeltaCatalog",
        "spark.hadoop.fs.s3a.endpoint" = "s3.${var.aws_region}.amazonaws.com" # Ensure S3 is
used
      }
    },
    # ... other configurations for Kafka connectivity etc.
  ])

  step_concurrency_level = 1 # For sequential steps

  tags = {
    Environment = var.environment
    Project     = var.project_name
  }
```

```
}

# Add steps (e.g., PySpark job execution) via aws_emr_cluster_step resource
```

**Option B: AWS Glue (Serverless Spark ETL) - Conceptual Glue ETL Job Definition:**

```
# Glue ETL Job Module
resource "aws_glue_job" "spark_transform_job" {
  name          = "${var.project_name}-spark-transform-${var.environment}"
  role_arn      = var.glue_execution_role_arn
  command {
    name            = "glueetl"
    script_location = "s3://${var.glue_scripts_bucket}/pyspark_jobs/data_transformer_spark.py"
    python_version  = "3"
  }
  default_arguments = {
    "--extra-jars"           = "s3://delta-lake/delta-core_2.12-2.4.0.jar" # Or from a public Maven
repo
    "--additional-python-modules" = "delta-spark==2.4.0"
    "--job-bookmark-option"  = "job-bookmark-enable" # To track processed data
    "--TempDir"              = "s3://${var.glue_temp_bucket}/temp/"
    "--source_kafka_topic"   = var.kafka_topic_name
    "--kafka_broker_address" = var.msk_bootstrap_brokers_tls
    "--raw_delta_path"       = "s3a://${var.raw_bucket_name}/"
    "--curated_delta_path"   = "s3a://${var.curated_bucket_name}/"
  }
  glue_version  = "4.0" # Or desired version (Spark 3.3)
  number_of_workers = var.number_of_glue_workers # DPUs * 2 for worker type Standard
  worker_type       = "G.1X" # Or G.2X, Standard
  timeout           = 60 # minutes
  tags = {
    Environment = var.environment
    Project     = var.project_name
  }
}

# You would then create aws_glue_trigger resources to schedule or event-drive this job.
```

## Amazon MWAA (Managed Workflows for Apache Airflow):

Creation via Console or Terraform resources aws_mwaa_environment.

## AWS Observability (ADOT, X-Ray, CloudWatch):

Managed services automatically integrate or can be configured via Lambda layers and ECS task definitions.

### Amazon Managed Grafana:

Workspace creation and data source linking.

### Data Lineage & Cataloging (Spline, OpenMetadata):

Deployment on EC2/ECS with RDS/OpenSearch for backends. OpenMetadata ingestion workflows configured to pull metadata from Glue Data Catalog, MSK, Spline, and CloudWatch.