Deep-Dive Addendum: Cloud Migration + Terraform Snippets

This addendum guides you through the process of migrating your locally developed enterprise data platform to a cloud environment, specifically AWS. It emphasizes a structured approach, service-by-service replacements, and provides conceptual Terraform Infrastructure as Code (IaC) snippets to automate the cloud deployment.

9.1. Overview of AWS Service Replacements

The local development environment, powered by Docker Compose, provides a powerful simulation of a production-grade data platform. When migrating to AWS, each open-source component has a corresponding managed AWS service that offers scalability, reliability, and reduced operational overhead.

Local Component (Docker Compose)	Primary AWS Replacement	Alternative AWS Services/Notes
FastAPI Ingestor	AWS Lambda + Amazon API Gateway	AWS Fargate (for containerized APIs), EC2 instances for fine-grained control
Apache Kafka	Amazon MSK (Managed Streaming for Kafka)	Amazon Kinesis (managed stream service, different API)
MinIO (S3 Compatible)	Amazon S3 (Simple Storage Service)	
Apache Spark	AWS Glue (Serverless ETL)	Amazon EMR (Managed Hadoop/Spark clusters), AWS Fargate for Spark on EKS (advanced)
PostgreSQL	Amazon RDS for PostgreSQL	Amazon Aurora PostgreSQL-Compatible Edition (high performance, scalability)
MongoDB	Amazon DocumentDB (MongoDB Compatible)	Amazon DynamoDB (NoSQL key-value/document, different API)
Apache Airflow	Amazon MWAA (Managed Workflows for Apache Airflow)	AWS Step Functions (serverless orchestration for simpler workflows)
Grafana	Amazon Managed Grafana	CloudWatch Dashboards (native AWS monitoring)
OpenTelemetry/Grafana Alloy	AWS Distro for OpenTelemetry (ADOT)	CloudWatch Agent, Fluent Bit/Fluentd (for log collection)

Spline	AWS Glue Data Catalog (limited lineage)	Third-party tools like OpenMetadata, manually configured lineage in Glue Data Catalog (limited)
OpenMetadata	AWS Glue Data Catalog + AWS OpenSearch/RDS	Custom deployment on EC2/ECS with managed databases for persistence
cAdvisor	Amazon CloudWatch Container Insights	

9.2. Step-by-Step AWS Migration Guide with IaC Examples

This guide outlines a phased approach to migrating your data platform to AWS, emphasizing the use of Terraform for automated infrastructure provisioning.

Phase 1: Core Networking & IAM Setup

Before deploying any services, establish the foundational network and security components. These are typically set up once per AWS account/region or per environment.

- **Create a VPC:** A Virtual Private Cloud (VPC) provides an isolated network environment.
    - Define public subnets (for NAT Gateway, Bastion hosts if needed) and private subnets (for most data services).
    - Configure Route Tables, Internet Gateway (for public subnets), and NAT Gateway (for private subnets to access the internet).
- **Establish IAM Roles and Policies:** Create specific IAM roles with the least privilege necessary for each AWS service (e.g., a role for Lambda to access MSK, a role for Glue to access S3).

Conceptual Terraform for VPC and Subnets:

```
# terraform_infra/modules/vpc/main.tf
resource "aws_vpc" "main" {
  cidr_block = var.vpc_cidr_block
  enable_dns_hostnames = true
  enable_dns_support   = true
  tags = {
    Name        = "${var.project_name}-${var.environment}-vpc"
    Environment = var.environment
  }
}

resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id
  tags = {
```

```
    Name = "${var.project_name}-${var.environment}-igw"
  }
}

resource "aws_subnet" "public" {
  count             = length(var.public_subnet_cidrs)
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.public_subnet_cidrs[count.index]
  availability_zone = data.aws_availability_zones.available.names[count.index]
  map_public_ip_on_launch = true
  tags = {
    Name        = "${var.project_name}-${var.environment}-public-subnet-${count.index}"
    Environment = var.environment
  }
}

resource "aws_subnet" "private" {
  count             = length(var.private_subnet_cidrs)
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.private_subnet_cidrs[count.index]
  availability_zone = data.aws_availability_zones.available.names[count.index]
  tags = {
    Name        = "${var.project_name}-${var.environment}-private-subnet-${count.index}"
    Environment = var.environment
  }
}

# ... (NAT Gateway, Route Tables, Security Groups would follow)
```

Phase 2: Data Lake (S3) and Core Databases (RDS, DocumentDB)

Start with the data persistence layers, as they are fundamental for storing data.

- **Amazon S3:** Replaces MinIO. Create S3 buckets for raw, curated, and any other data zones. Enable versioning, encryption, and cross-region replication for DR.
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*
- **Amazon RDS for PostgreSQL:** Replaces local PostgreSQL. Deploy a managed PostgreSQL instance for structured data, application metadata, and Airflow metastore. Configure multi-AZ for high availability.
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*
- **Amazon DocumentDB:** Replaces local MongoDB. Provision a DocumentDB cluster for MongoDB-compatible workloads.

Phase 3: Streaming & Ingestion Layer (MSK, Lambda + API Gateway)

Migrate the real-time ingestion and streaming components.

- **Amazon MSK:** Replaces Apache Kafka. Provision a managed Kafka cluster. Configure Kafka topics, retention policies, and security settings (TLS, IAM authentication).
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*
- **AWS Lambda + Amazon API Gateway:** Replaces FastAPI. Containerize your FastAPI application into a Docker image, push to ECR, and deploy as a Lambda function triggered by API Gateway. This provides a scalable, serverless ingestion endpoint.
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*

Phase 4: Data Processing (Glue or EMR)

Transition your Spark batch and streaming jobs to a managed AWS service.

- **AWS Glue:** A serverless ETL service that supports Spark. Ideal for transient jobs and where you prefer to avoid managing clusters.
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*
- **Amazon EMR:** Managed clusters for Hadoop, Spark, Hive, etc. Offers more control over the cluster environment.
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*

Phase 5: Orchestration, Governance & Observability (MWAA, Managed Grafana, OpenMetadata)

Deploy the control plane and monitoring solutions.

- **Amazon MWAA:** Replaces Apache Airflow. Deploy a managed Airflow environment. Migrate your DAGs to an S3 bucket connected to MWAA.
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*
- **Amazon Managed Grafana:** Replaces local Grafana. Create a Grafana workspace and connect it to CloudWatch, Prometheus (via ADOT), and other data sources.
    - *See Terraform Snippet in IaC & CI/CD Addendum, Appendix I.*
- **Data Lineage & Cataloging (OpenMetadata):** Deploy OpenMetadata on EC2/ECS with managed databases (RDS/OpenSearch). Configure connectors to pull metadata from Glue Data Catalog, MSK, and other sources.

Phase 6: CI/CD Pipeline Update

Modify your CI/CD pipelines (e.g., GitHub Actions) to deploy to AWS using Terraform.

- **Update Terraform Configuration:** Ensure your Terraform code points to the correct AWS regions, account IDs, and uses appropriate variables for different environments (dev, staging, prod).
- **Integrate AWS CLI/Terraform in CI/CD:** Configure GitHub Actions (or your chosen CI/CD tool) with AWS credentials to run terraform apply for deploying infrastructure and docker push to ECR.
    - *See Conceptual GitHub Actions Workflow in IaC & CI/CD Addendum, Section 5.3.*

Cloud Migration Considerations:

- **Cost Management:** AWS services are pay-as-you-go. Monitor costs closely using AWS Cost Explorer and implement budget alerts. Optimize resource sizing.
- **Security:** Leverage AWS IAM for granular access control, VPC security groups, network ACLs, and AWS Key Management Service (KMS) for encryption. Integrate AWS Secrets Manager.
- **Monitoring & Logging:** Utilize CloudWatch Logs, Metrics, and Alarms. Implement AWS Distro for OpenTelemetry (ADOT) for comprehensive tracing and metric collection.
- **Data Migration:** For existing data, plan a data migration strategy (e.g., AWS DataSync, S3 Transfer Acceleration, AWS Database Migration Service (DMS)).
- **Testing in Cloud:** Thoroughly test each migrated component and the end-to-end data pipelines in the AWS environment before going live.
- **Progressive Rollout:** Consider a phased rollout (e.g., migrating one critical pipeline first) rather than a "big bang" approach.
- **DR Planning in Cloud:** AWS provides services (Multi-AZ, Cross-Region Replication, Snapshots) that simplify DR. Integrate these into your DR strategy, defining cloud-native RPOs and RTOs.