MOVINGDATA: TECHNICAL ARCHITECTURE DOSSIER

Project: Intelligent Multi-Cloud Data Pipeline (NetApp "Data in Motion" Hackathon Submission)
Authors: MovingData Team
Repository: ‘MovingData‘
Document Version: 1.0
Last Updated: 2025-11-09

TABLE OF CONTENTS

EXECUTIVE SUMMARY

MovingData demonstrates an enterprise-grade control plane that keeps sensitive data encrypted end-to-end while orchestrating replication across heterogeneous object stores. The platform blends deterministic policies with machine-learning predictions to migrate content between emulated AWS S3 (MinIO), Azure Blob Storage (Azurite), and Google Cloud Storage (FakeGCS). The mission control dashboard provides an operational cockpit for analysts to validate integrity, latency, and cost signals in real time.

Key capabilities showcased in the hackathon submission include:

- **Secure-by-default transfers** - Every payload is encrypted via location-specific Fernet keys before leaving the API process. MinIO additionally applies SSE-KMS using a static key alias (‘netapp-key‘).
- **Cross-cloud parity** - Replicas are created and verified across MinIO, Azurite, and FakeGCS through unified storage client abstractions. Checksums, retries, and conflict detection guarantee deterministic outcomes.
- **Event-driven decisioning** - Access patterns, stream anomalies, and operational metrics flow into Redpanda (Kafka API). These signals feed the predictive tiering engine and drive alerting thresholds.
- **Observability-first design** - FastAPI exposes health, file inventory, policy metadata, and consistency status endpoints. Streamlit visualizations surface telemetry for demo audiences without requiring direct CLI access.

SYSTEM GOALS & NON-FUNCTIONAL REQUIREMENTS

The hackathon brief emphasized a multi-cloud, data-in-motion workload with strict security assurances. MovingData addresses the following goals:

- **Confidentiality** - Encrypt in-flight data with distinct Fernet keys per provider and enforce RBAC gates prior to decryption.
- **Integrity & Consistency** - Maintain consistent object replicas despite heterogeneous APIs; detect and heal drifts automatically.
- **Availability** - Provide resilient background simulations and manual overrides through RESTful operations; degrade gracefully when optional services (e.g., stream sandbox) are offline.
- **Auditability** - Persist metadata describing encryption policies, access events, and synchronization attempts in MongoDB for evidence generation.
- **Extensibility** - Modular orchestrator and storage clients simplify replacement of emulators with production cloud SDKs.

Non-functional requirements shaped the implementation:

- **Demo-ready within minutes** - ‘docker compose up‘ must bootstrap all dependencies without external credentials.
- **Secure defaults** - Pre-generated Fernet keys, RBAC roles, and SSE-KMS configuration allow immediate demonstrations without manual key provisioning.
- **Synthetic load generation** - Optional background simulator exercises telemetry, predictive models, and alerting logic for hackathon judges.

END-TO-END ARCHITECTURE

High-Level Data Flow

1. **Ingress** - Users interact through the FastAPI control plane (‘app/api/server.py‘) or via the Streamlit dashboard (‘app/ui/dashboard.py‘).
2. **Encryption & Authorization** - The security manager (‘app/security/policies.py‘) validates caller roles and encrypts payloads destined for storage providers.
3. **Replication** - The orchestrator mover (‘app/orchestrator/mover.py‘) streams ciphertext between providers using the storage client abstractions.
4. **Metadata Persistence** - MongoDB stores file inventories, access events, and synchronization metadata. Collections are lazily created during API startup.
5. **Telemetry** - Access events and streaming anomalies publish into Redpanda. The streaming consumer module can push high-temperature alerts back to the API.
6. **Visualization** - Streamlit polls ‘/health‘, ‘/files‘, ‘/consistency/status‘, and streaming metrics to render dashboards in real time.

Component Inventory

Layer	Module	Responsibilities
API	‘app/api/server.py‘	REST endpoints, Kafka producers, Mongo orchestration, simulation threads
Orchestrator	‘app/orchestrator/*‘	Migration, predictive analytics, rules engine, consistency management
Security	‘app/security/policies.py‘	Fernet key management, RBAC enforcement, policy reporting
Storage Clients	‘app/storage_clients/*.py‘	Thin wrappers around MinIO/S3, Azurite, and FakeGCS REST APIs
Streaming	‘app/streaming/*‘	Kafka producers/consumers for IoT-style telemetry
UI	‘app/ui/dashboard.py‘, ‘ui/stream_dashboard.py‘	Streamlit dashboards for operators
Infrastructure	‘infra/docker-compose.yml‘, ‘infra/stream-compose.yml‘	Containerized topology for demos

CONTROL PLANE API (FASTAPI)

Startup Lifecycle

During ‘@app.on_event("startup")‘, the server:

1. Connects to MongoDB and initializes collections (‘files‘, ‘access_events‘, ‘sync_status‘).
2. Ensures Kafka topics exist via ‘orchestrator.stream_consumer.ensure_topic‘ and builds a ‘KafkaProducer‘ instance.
3. Seeds the predictive model (‘TierPredictor‘) with bootstrap training data or centroid defaults.
4. Parses replica endpoint overrides and instantiates the ‘ConsistencyManager‘ for drift detection.
5. Launches an optional synthetic load thread when ‘ENABLE_SYNTHETIC_LOAD‘ is enabled, generating access events and periodic migrations.

Shutdown handlers flush Kafka producers, close Mongo connections, and stop the simulator thread gracefully.

REST Endpoints

The API surface enables full lifecycle control:

Endpoint	Method	Purpose
‘/health‘	GET	Aggregated status including uptime, Mongo/Kafka connectivity, simulator counters, and active alerts.
‘/files‘	GET	Returns known files with storage location, size, confidence scores, and policy metadata.
‘/policy/{file_id}‘	GET	Introspects the encryption/RBAC policy applied to the file’s current location.
‘/streaming/metrics‘	GET	Reports producer/consumer health, event throughput, and last anomaly timestamps.
‘/predictive/train‘	POST	Accepts labeled records or auto-label requests to refine the tiering model (supports hist-gradient boosting or centroid fallback).
‘/ingest_event‘	POST	Records a raw access event from external systems and publishes to Kafka; updates rolling statistics.
‘/move‘	POST	Securely migrates an object between providers, enforcing RBAC and encryption per source/destination.
‘/seed‘	POST	Ensures buckets/containers exist and uploads demo seed files from ‘data/seeds/‘.
‘/simulate/burst‘	POST	Triggers a bounded simulation burst with configurable pace, file sampling, and Kafka streaming.
‘/storage_test‘	POST	Performs cross-provider read/write checks to validate credentials and encryption plumbing.

‘/consistency/status‘ | GET | Surfaces replica parity metrics, conflict counters, and last repair timestamps.
‘/consistency/resync‘ | POST | Initiates targeted resynchronization against configured replica endpoints.

MongoDB Schemas & Collections

MongoDB acts as the metadata backbone:

- **‘files‘** - Documents describing each tracked object: size (KB), latest tier, confidence, costs, policy snapshot, and last move timestamps.
- **‘access_events‘** - Raw event stream for analytics, including latency metrics, temperature readings, and flags for high-temperature or network anomalies.
- **‘sync_status‘** - Replica verification results, retry counters, conflicts detected, and external endpoint health.

Indexes are created dynamically to optimize queries by file ID and timestamps. The ‘Collation‘ ensures case-insensitive matching where required.

Kafka Telemetry Pipeline

‘KafkaProducer‘ instances target the ‘access-events‘ topic (default). Each event includes:

- File identifier and event type (read/write).
- Latency, size, and client metadata for predictive features.
- Derived fields such as estimated egress cost and alert flags.

The orchestrator consumer (‘orchestrator.stream_consumer‘) subscribes to the same topic to maintain moving averages, update predictive feature windows, and push anomalies back into the API.

ORCHESTRATION SERVICES

Mover

‘app/orchestrator/mover.py‘ encapsulates secure transfers. Key behaviors:

- Lazily ensures required buckets/containers exist when seeding.
- Retrieves ciphertext from the source provider, decrypts using source policy, and re-encrypts with the destination key.
- Deletes the source object post-transfer to simulate a true "move" while guaranteeing destination integrity via checksums.
- Supports principal role overrides so that human operators can provide explicit RBAC context when invoking ‘/move‘.

Predictive Tiering

‘app/orchestrator/predictive.py‘ implements the ‘TierPredictor‘ class:

- Prefers ‘HistGradientBoostingClassifier‘ when scikit-learn is available; otherwise falls back to centroid-based nearest neighbor logic.
- Normalizes feature vectors using defaults defined in ‘FEATURE_DEFAULTS‘ within the API server.
- Emits confidence scores and rationale metadata so policies can override when confidence is low or costs exceed thresholds.
- Offers auto-labeling heuristics that bucket files into hot/warm/cold tiers based on recent activity and SLA pressures.

Rules Engine

‘app/orchestrator/rules.py‘ codifies deterministic overrides:

- Forces **hot tier** retention when latency SLOs are breached or high-temperature alerts persist.
- Promotes to **warm tier** when access frequency drops but confidence remains high.
- Falls back to **cold tier** when access is sporadic and storage cost pressures dominate.
- Integrates with alert thresholds defined in environment variables, ensuring that operations teams can tune behavior without code changes.

Consistency Verification

‘app/orchestrator/consistency.py‘ provides the ‘ConsistencyManager‘:

- Periodically polls configured replica endpoints (including optional external REST mirrors) to compare checksums.
- Applies exponential backoff via ‘with_retry‘ helper for flaky networks.
- Flags conflicts in MongoDB and surfaces them through ‘/consistency/status‘ so Streamlit can visualize drift.
- Supports manual resynchronization workflows triggered by ‘/consistency/resync‘.

Streaming Consumers

‘app/orchestrator/stream_consumer.py‘ and ‘app/streaming/consumer‘ coordinate Kafka subscriptions:

- Maintain connection health, last processed offsets, and consumer lag metrics.
- Transform raw events into aggregated statistics for predictive features (EMA, growth rates, percentile latency calculations).
- Bubble anomalies (e.g., sustained high temperature) back to the API for alerting and policy overrides.

SECURITY & COMPLIANCE

Encryption Domains

Three logical locations-‘s3‘, ‘azure‘, and ‘gcs‘-map to their respective emulators. Each domain receives a dedicated Fernet key. Payloads remain encrypted even during transient storage in orchestrator memory, minimizing plaintext exposure.

Role-Based Access Control

‘AdaptiveSecurityManager.authorize‘ ensures that the caller possesses at least one allowed role before encryption/decryption. Defaults include:

- S3: ‘analytics‘, ‘engineering‘, ‘system‘
- Azure: ‘operations‘, ‘engineering‘, ‘system‘
- GCS: ‘compliance‘, ‘analytics‘, ‘system‘

Operators can override roles via environment variables (‘S3_ALLOWED_ROLES‘, etc.) without redeploying the API.

Key Management & Rotation

Keys can be rotated through environment variables (‘S3_ENCRYPTION_KEY‘, etc.). When set, the manager records that the source was ‘env:VARIABLE‘, aiding audit trails. Invalid key lengths trigger immediate startup failures, preventing silent misconfiguration.

Auditability

‘describe_policy‘ generates structured snapshots covering algorithm, key ID, key source, and allowed roles. These snapshots are persisted alongside file metadata so auditors can verify encryption lineage per object.

STORAGE CLIENT ABSTRACTIONS

- **‘storage_clients/s3_client.py‘** - Wraps ‘boto3‘-compatible MinIO operations (bucket ensure, put/get/delete) with retry semantics and SSE headers.
- **‘storage_clients/azure_client.py‘** - Uses Azure Blob SDK equivalents to manage containers and block blobs against Azurite.
- **‘storage_clients/gcs_client.py‘** - Talks to FakeGCS via HTTP, normalizing responses to match other clients.

These thin adapters isolate provider-specific credentials and APIs, enabling future swap-outs with managed cloud services.

OBSERVABILITY & ALERTING

- Alert thresholds for cost (‘ALERT_COST_THRESHOLD‘), latency (‘ALERT_LATENCY_P95_MS‘), stream throughput, and predictive confidence are environment-configurable.
- ‘/health‘ consolidates metrics such as Mongo connection status, Kafka reachability, simulator counters, and outstanding alerts.
- Consistency metrics track last verification time, conflict counts, and retry statistics for each replica endpoint.
- Streamlit surfaces sparkline charts, tabular inventories, and anomaly banners to make drift and SLA breaches obvious during demos.

STREAMLIT MISSION CONTROL

‘app/ui/dashboard.py‘ consumes the API to render:

- File inventory table with tier, size, replica status, encryption policy summary, and estimated monthly cost.
- Alert panel summarizing latency, cost, or anomaly breaches.
- Real-time stream metrics (events per minute, connected devices, high-temperature alerts).
- Manual controls to trigger seeding, migrations, and simulation bursts for live demonstrations.

An optional ‘ui/stream_dashboard.py‘ focuses on the streaming sandbox, plotting IoT sensor data and anomaly detectors when the secondary stream API is enabled via ‘infra/stream-compose.yml‘.

INFRASTRUCTURE AS CODE

Local Compose Topology

‘infra/docker-compose.yml‘ orchestrates the full stack:

1. **Redpanda** (‘redpandadata/redpanda‘) - Kafka-compatible broker without ZooKeeper.
2. **MinIO** - Provides S3 API with SSE-KMS enabled via ‘netapp-key‘ alias.
3. **Azurite** - Emulates Azure Blob Storage.
4. **FakeGCS** - Simulates Google Cloud Storage (HTTP on port 4443).
5. **MongoDB** - Metadata store.
6. **API** - Builds from ‘app/Dockerfile‘, mounts source volume for hot reload.
7. **UI** - Lightweight Python container running Streamlit.

Volumes persist emulator data between restarts. Ports are published for local testing and UI access.

Environment Configuration

Key environment variables (see [Appendix](#reference-environment-variables)) control endpoints, credentials, and alert thresholds. Compose injects sane defaults so newcomers can start demos without extra setup.

Extending to Cloud Providers

To graduate from emulators to managed services:

- Swap MinIO credentials for AWS IAM keys and update ‘S3_ENDPOINT‘ to an AWS region endpoint.
- Configure Azure storage connection strings and enable HTTPS.
- Point GCS client to ‘https://storage.googleapis.com‘ with OAuth credentials.
- Harden KMS integration by replacing MinIO static key with external KMS URIs (AWS KMS, Azure Key Vault, Google Cloud KMS).
- Add infrastructure automation (Terraform or Pulumi) to provision buckets and IAM roles consistently.

RUNBOOKS & OPERATIONAL PLAYBOOKS

1. **Bootstrap Demo**
CODE BLOCK:

```
cd infra
docker compose up --build
```

Wait for API (port 8001) and Streamlit (port 8501) to report healthy.

2. **Seed Initial Data**
CODE BLOCK:

```
curl -X POST http://localhost:8001/seed
```

Seeds ‘data/seeds/*.txt‘ into MinIO with encryption.

3. **Trigger Predictive Move**
CODE BLOCK:

```
curl -X POST http://localhost:8001/move \
    -H ’Content-Type: application/json’ \
    -d ’{"file_id": "file_001.txt", "target": "gcs", "principal_roles":
["analytics"]}’
```

4. **Inspect Consistency**
CODE BLOCK:

```
curl http://localhost:8001/consistency/status
```

5. **Kick Off Simulation Burst**
CODE BLOCK:

```
curl -X POST http://localhost:8001/simulate/burst \
    -H 'Content-Type: application/json' \
    -d '{"events": 250, "stream_events": true, "include_moves": true, "pace_ms":
25}'
```

TESTING STRATEGY

- Unit-level logic (encryption, RBAC, predictive scoring) can be exercised via pytest when run locally.
- Integration testing leverages the Compose stack: seed data, move objects, and validate metrics via REST endpoints.
- Observability smoke tests query `/health`, `/streaming/metrics`, and `/consistency/status` to ensure instrumentation is live.
- Future iterations should add CI workflows that spin up minimal services (MinIO + Mongo) to validate migrations in headless mode.

ROADMAP & FUTURE ENHANCEMENTS

- Integrate native AWS/Azure/GCP SDKs with credential management and IAM policies.
- Add Kubernetes deployment manifests (Helm or Kustomize) for scalable clusters.
- Implement reinforcement learning feedback loops for adaptive tiering.
- Expand anomaly detection with multivariate forecasting models.
- Introduce cost governance dashboards correlating with real billing APIs.

APPENDIX

Reference Environment Variables

Variable	Default	Description
`KAFKA_BOOTSTRAP`	`redpanda:9092`	Kafka bootstrap server for producers/consumers
`TOPIC_ACCESS`	`access-events`	Topic for telemetry events

‘MONGO_URL‘ | ‘mongodb://mongo:27017‘ | Metadata persistence
‘S3_ENDPOINT‘ | ‘http://minio:9000‘ | MinIO API endpoint
‘S3_ACCESS_KEY‘ | ‘minio‘ | MinIO credential
‘S3_SECRET_KEY‘ | ‘minio12345‘ | MinIO credential
‘S3_BUCKET‘ | ‘netapp-bucket‘ | Primary S3 bucket
‘AZURITE_BLOB_URL‘ | ‘http://azurite:10000/devstoreaccount1‘ | Azurite endpoint
‘GCS_ENDPOINT‘ | ‘http://fakegcs:4443‘ | FakeGCS endpoint
‘GCS_BUCKET‘ | ‘netapp-gcs‘ | Target GCS bucket
‘ENABLE_SYNTHETIC_LOAD‘ | ‘1‘ | Toggle background simulator
‘ALERT_COST_THRESHOLD‘ | ‘0.08‘ | Storage cost alert threshold (USD/GB)
‘ALERT_LATENCY_P95_MS‘ | ‘180‘ | Latency SLO (milliseconds)
‘ALERT_STREAM_EVENTS_PER_MIN‘ | ‘15‘ | Streaming throughput minimum
‘ALERT_CONFIDENCE_THRESHOLD‘ | ‘0.9‘ | Predictive confidence floor
‘S3_ENCRYPTION_KEY‘ | demo Fernet key | Override for S3 encryption
‘AZURE_ENCRYPTION_KEY‘ | demo Fernet key | Override for Azure encryption
‘GCS_ENCRYPTION_KEY‘ | demo Fernet key | Override for GCS encryption
‘S3_ALLOWED_ROLES‘ | ‘analytics,engineering,system‘ | RBAC overrides
‘AZURE_ALLOWED_ROLES‘ | ‘operations,engineering,system‘ | RBAC overrides
‘GCS_ALLOWED_ROLES‘ | ‘compliance,analytics,system‘ | RBAC overrides

Sample API Sequences

1. **Onboard New Dataset**
1. Call ‘/seed‘ to ensure storage primitives exist and load baseline files.
2. Submit historical access records to ‘/predictive/train‘ to prime the tiering model.
3. Stream real-time events via ‘/ingest_event‘; monitor Streamlit for anomalies.
4. Use ‘/move‘ to promote/demote tiers based on combined ML + rules recommendations.
5. Periodically hit ‘/consistency/status‘ and ‘/consistency/resync‘ to maintain replica health.

2. **Respond to Latency Incident**
1. ‘/health‘ reveals rising latency alerts and declining predictive confidence.
2. ‘/files‘ identifies impacted objects with warm/cold tiers.
3. ‘/move‘ promotes critical files to ‘s3‘ hot storage using emergency operator roles.
4. ‘/simulate/burst‘ is disabled (set ‘ENABLE_SYNTHETIC_LOAD=0‘) to reduce background noise.
5. ‘/streaming/metrics‘ confirms throughput recovery; update postmortem with Mongo audit records.

End of Document