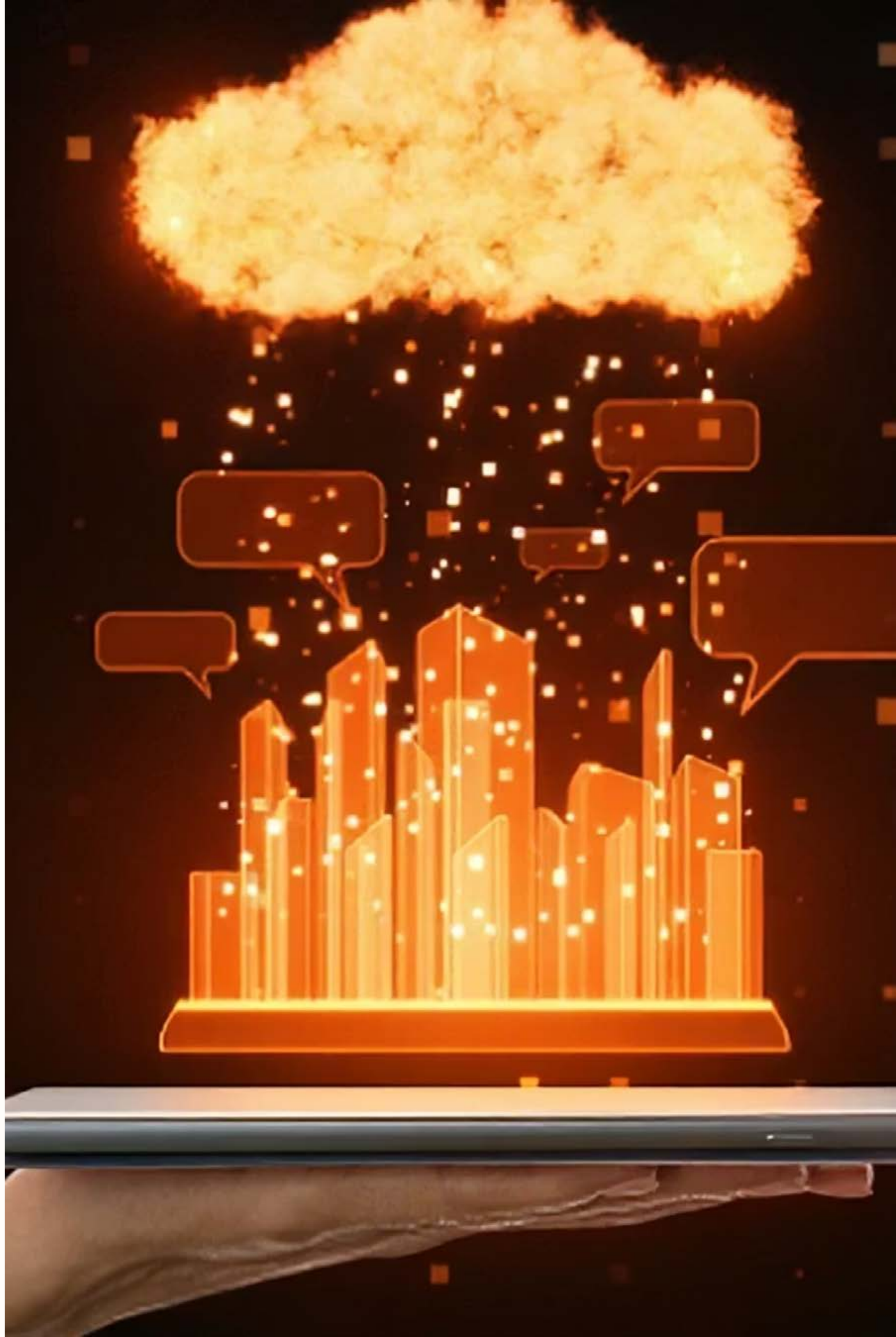
Kube-Native ETL at Scale: Optimizing PySpark + Airflow Workflows in Cloud-Native Environments

Presented by: Sruthi Erra Hareram

Independent Researcher, Canada

Conf42.com Incident Management 2025



Agenda: Scaling ETL in Kubernetes Environments

1. **Cloud-Native ETL Architecture**

   Container-based pipelines and architectural patterns

2. **PySpark Performance Optimization**

   Memory management, join strategies, and execution planning

3. **Airflow Orchestration Techniques**

   Dynamic DAG creation, resilience patterns, and monitoring

4. **Real-World Case Studies**

   Telecom and media deployments with 5TB+ daily processing

5. **Implementation Roadmap**

   Actionable patterns for your organization

The ETL Scaling Challenge

Scaling ETL processes presents significant challenges for enterprises, hindering effective data leverage due to:

- Inefficient resource allocation and slow infrastructure provisioning.
- Limited pipeline visibility and complex dependency management.
- Brittle, monolithic processing jobs.
- Escalating data volumes and high operational overhead.

These systemic challenges lead to delays in data availability, increased operational costs, and a reactive approach to data management. Overcoming them necessitates a fundamental transformation towards more agile, automated, and scalable architectures for truly data-driven decision-making.

Cloud-Native ETL Architecture

Containerization Benefits

Isolates dependencies, ensures reproducibility across environments, and enables precise resource allocation for ETL pipelines.

Kubernetes as Compute Fabric

Provides dynamic scaling of worker pods, namespace isolation for multi-tenancy, and robust resource governance.

Infrastructure-as-Code (IaC)

Leverages Terraform and Helm for declarative deployment, fostering automated GitOps workflows.

PySpark on Kubernetes: Core Optimization Techniques

Memory Management

- Right-sized executor memory allocation (4-8GB optimal)
- Strategic caching with `persist()` at materialization points
- Memory fraction tuning (0.6 for execution, 0.2 for storage)

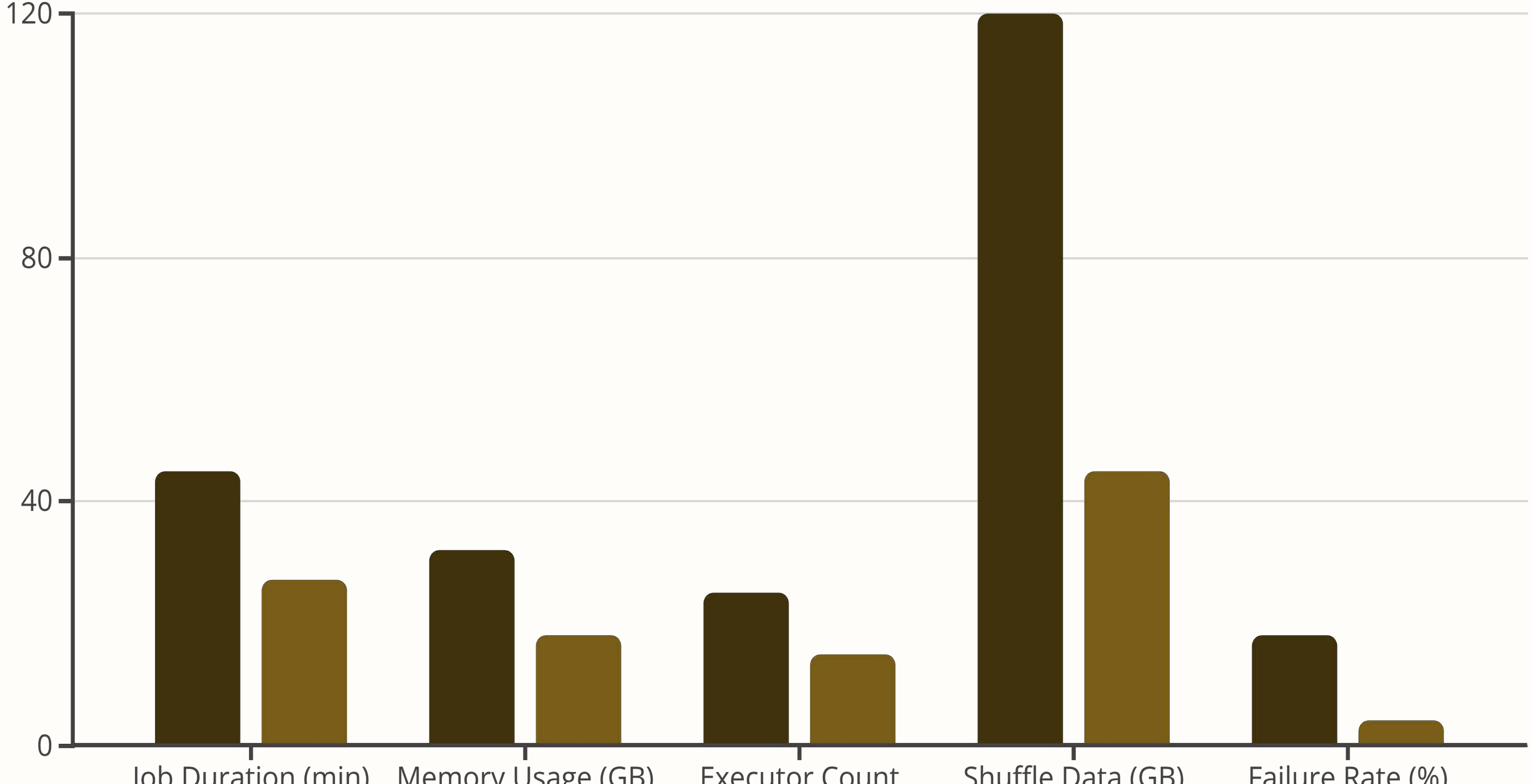
Join Optimization

- Broadcast hash joins for dimension tables (<100MB)
- Sort-merge joins for large fact tables
- Skew handling with salting techniques

Fault Tolerance

- Strategic checkpointing for lineage truncation
- Dynamic partition discovery with predicate pushdown
- K8s-aware restart policies with idempotent writers

Performance Benchmarks: Before & After Optimization



Airflow on Kubernetes: Modernizing Orchestration

Cloud Composer/K8s-native Benefits

- **Scheduler Scalability:** Horizontally scaled for 1,000+ daily DAG runs
- **Worker Isolation:** Task-specific resource profiles via Pod templates
- **Security:** Workload identity for IAM integration and secrets management
- **Observability:** Native integration with Cloud Logging/Monitoring

Managed Airflow environments provide 99.9% scheduler uptime with drastically reduced operational overhead.

Advanced Airflow Patterns for Resilient ETL

1 Dynamic DAG Generation

Programmatically generate DAGs from configuration stores (GCS/S3) to support hundreds of pipeline variations with minimal code duplication.

2 Intelligent Branching

Implement data-aware routing with `BranchPythonOperator` to conditionally execute specific pipeline segments based on quality checks or volume thresholds.

3 Backfill-Safe Design

Employ logical timestamps and idempotent operations to enable safe historical reprocessing without data corruption or duplication.

4 SLA Monitoring

Custom SLA callbacks tied to monitoring systems (Datadog, Prometheus) for proactive alerting and anomaly detection.

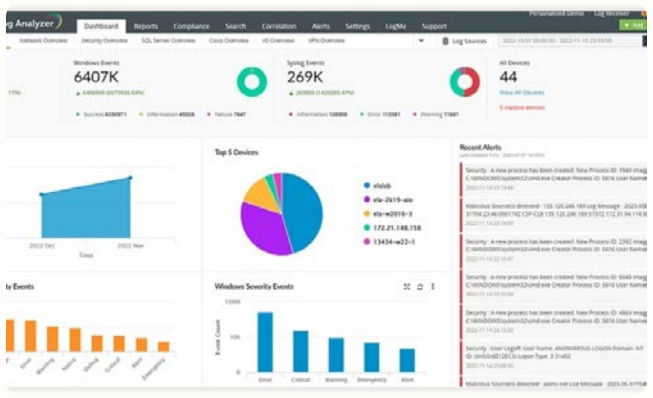
Spark Executor Sizing: The Art of Resource Allocation

Finding the optimal Spark executor configuration requires balancing:

Monitoring & Observability: The Reliability Foundation

Real-time Monitoring

Leverage Prometheus for cluster and Spark metrics, crucial for detecting resource saturation and performance bottlenecks.

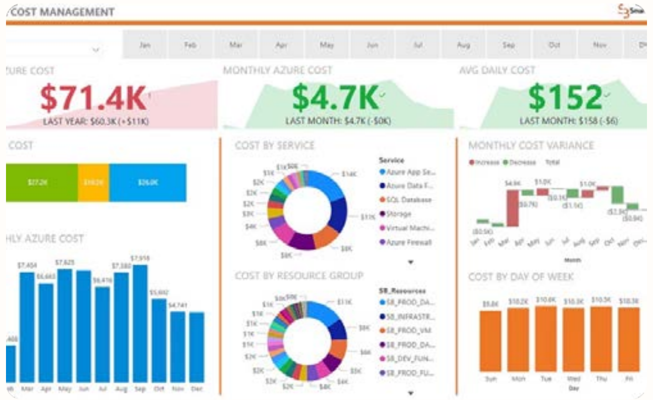


Structured Logging & Tracing

Implement JSON logs with correlation IDs and OpenTelemetry for end-to-end tracing, identifying bottlenecks and lineage.

Data Quality & Alerting

Integrate automated data profiling (e.g., Great Expectations) to proactively identify anomalies and configure alerts for critical KPIs.



Cost Monitoring

Track resource consumption and associated cloud costs for individual Spark applications and Airflow DAGs to identify inefficiencies and optimize budget.

Modernization Journey: From Legacy Batch to Cloud-Native ETL

1 Phase 1: Assessment

Inventory existing jobs, performance profiling, and dependency mapping. Identify migration candidates based on business impact.

2 Phase 2: Containerization

Refactor monolithic scripts into modular components with clear interfaces. Package with Docker using multi-stage builds for efficiency.

3 Phase 3: Orchestration

Implement Airflow DAGs with proper dependency management. Set up environment-specific configurations using Airflow variables.

4 Phase 4: Optimization

Performance tuning, resource right-sizing, and implementation of retry/recovery mechanisms for resilience.

5 Phase 5: Operationalization

CI/CD pipelines, monitoring integration, and SLA management. Documentation and knowledge transfer.

Common Pitfalls and How to Avoid Them

→ Resource Misallocation

Over-provisioning executors leads to inefficient cluster usage; under-provisioning causes job failures. Use dynamic allocation with reasonable bounds.

→ Inappropriate Storage Formats

Using CSV/JSON for large datasets instead of Parquet/ORC. Implement columnar storage with compression for 3-5x performance gains.

→ Excessive Shuffling

Data skew and unnecessary repartitioning create bottlenecks. Profile with Spark UI to identify and refactor problematic transformations.

→ Orphaned Resources

Failed jobs leaving dangling pods and persistent volumes. Implement proper cleanup hooks and resource quotas as safeguards.

Implementation Roadmap: Getting Started

Start Small

Begin with a single non-critical pipeline as a proof-of-concept. Document baseline metrics before migration for comparison.

Build Infrastructure

Set up Kubernetes cluster with appropriate node pools. Deploy Airflow using the Helm chart with customized values. Configure networking and security.

Migrate Gradually

Convert jobs incrementally, running in parallel with legacy systems. Implement comprehensive logging and monitoring from day one.

Optimize Continuously

Establish regular performance reviews. Create a tuning playbook based on production observations. Share knowledge across teams.

Key Takeaways: The Path to ETL Excellence

Architectural Discipline

Container-native design principles deliver portability and scalability benefits that far outweigh migration effort.

Performance Optimization

Systematic tuning of PySpark parameters based on workload characteristics can yield 30-50% efficiency improvements.

Resilient Orchestration

Invest in robust DAG design patterns with proper error handling to achieve high reliability even with variable data quality.

Operational Excellence

End-to-end observability and SLA monitoring are not optional—they're foundational for maintaining data pipeline trust.

Thank You