



# Navigating Secure and Cost-Efficient Flink Batch on Kubernetes with Airflow

Yahoo! | Purshotam Shah & Prakash Nandha Mukunthan



# 3.0

# Presentation

## Agenda

01	02	03
<b>The Challenge: Batch Processing</b>	<b>Flink Overview</b>	<b>Security Recommendations</b>
04	05	06
<b>Problem Use Cases</b>	<b>Airflow Integration</b>	<b>High Level Architecture</b>
07	08	09
<b>End to End Flow</b>	<b>Security Challenges</b>	<b>Cost Optimization</b>



# The Challenge: Batch Processing for the Ad Team at Yahoo

## Reduced Learning Curve

- Reuse existing Flink expertise to seamlessly extend into batch processing.

## Lower Operational Overhead

- One platform for streaming and batch simplifies operations and reduces effort.

## Cost Efficiency

- Scale resources on demand to run batch jobs more affordably.

## Enterprise-Grade Security

- Built-in compliance with Yahoo's data protection and security standards.

# Why Flink?



## Unified Processing Engine

- One runtime for both **streaming** and **batch** (DataStream, Batch, Table/SQL).
- Simplifies ETL + real-time pipelines in a single framework.



## Autoscaling & auto-tuning

**Autoscaling:** Flink on EKS automatically scales TaskManagers up or down based on workload needs.

**Autotuning:** It adjusts memory and task settings on the fly to keep jobs efficient and stable.



## EKS Operator Ecosystem

The Flink Kubernetes Operator can manage end-to-end lifecycle of Flink applications, including submission, upgrades, rollbacks, and savepoints.



## Enhanced Resource Isolation

Better isolation for batch jobs with per-application Flink clusters.



## Cloud-Native S3

**Integration** Batch pipelines on EKS work seamlessly with S3 without relying on Hadoop.

# The Solution Apache Flink & The Flink Kubernetes Operator

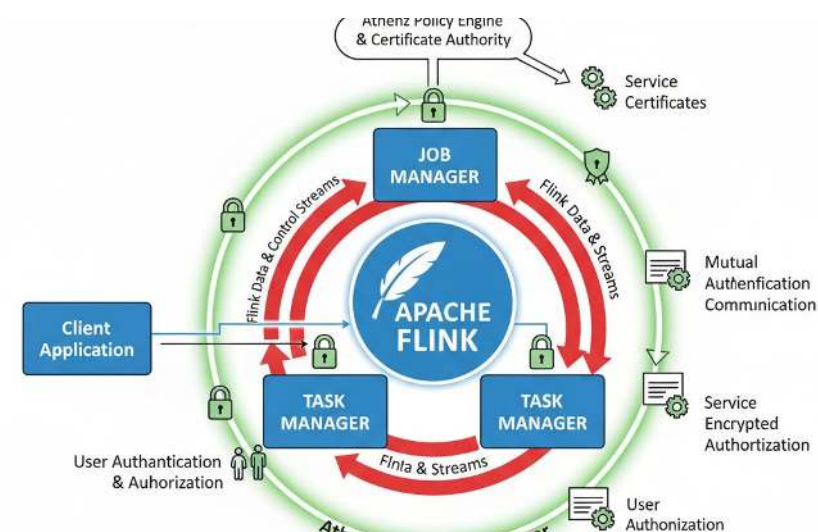
## Apache Flink: Stream & Batch Processing

Flink is an open-source engine for stream and batch processing, built for fast, reliable, and fault-tolerant real-time analytics.

## The Flink Kubernetes Operator

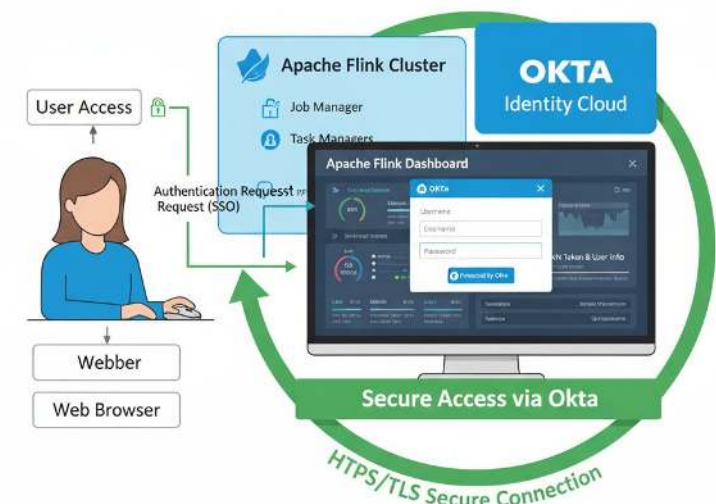
The operator streamlines running Flink on Kubernetes, managing job lifecycles, handling autoscaling and autotuning, performing rolling updates, and ensuring recovery.

# Security Recommendations



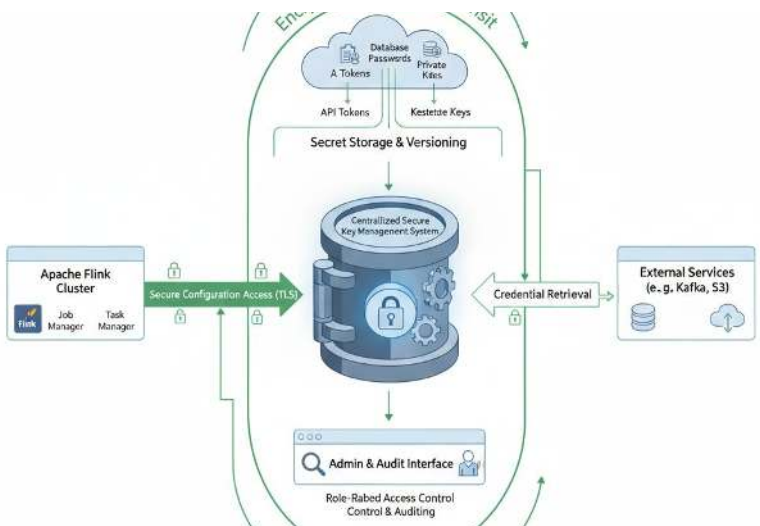
## mTLS for Cluster Communication

Secure all Flink cluster communication with Athenz mTLS, using certificates to protect data and verify users and services.



## Secure Flink UI with Okta Authentication

Secure Flink UI with Okta, allowing access only to authorized users.



## Centralized Secret Management

Keep Flink secrets, including keystore and truststore passwords, in Yahoo's Managed Key Service for better security.

# mTLS - External Communication (Outside the Cluster)

When an external client or service connects to Flink (e.g., submitting a job, accessing the REST API, or metrics endpoint), it must present a client certificate.



## 1 Certificate Common Name (CN) Pattern

The certificate's Common Name (CN) must match a pattern:

```
# Regex applied to client certificate's CNcheck =  
<Athenz-domain>:([0-9a-z_-]+).*
```

Example: `finance:([0-9a-z_-]+).*`

## 2 Security Assurances

- Only users with valid roles in the Athenz domain can connect.



# mTLS - Internal Communication (Inside the Cluster)

Communication between Flink components (JobManager ↔ TaskManager, TaskManager ↔ TaskManager) also requires certificates.

## Certificate Common Name (CN) Exact Match

Here, the CN must match exactly:

```
<Athenz-domain>.app-1
```

Example: `finance.app-1`

## Security Assurances

- A rogue TaskManager or compromised pod cannot impersonate another component.





# Okta Integration for Flink UI Authentication: Flow

Add Okta Sidecar



Okta Token Validation



Invalid Token: Access Denied



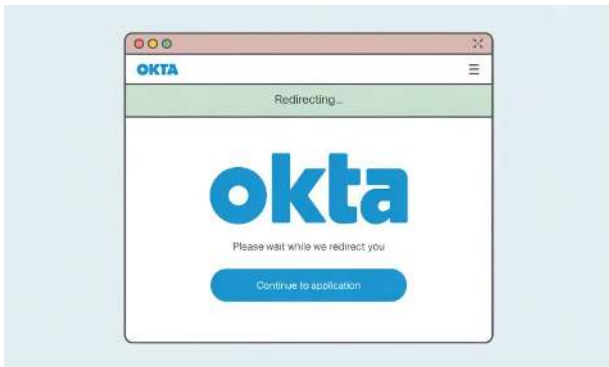
Endpoint Exposure



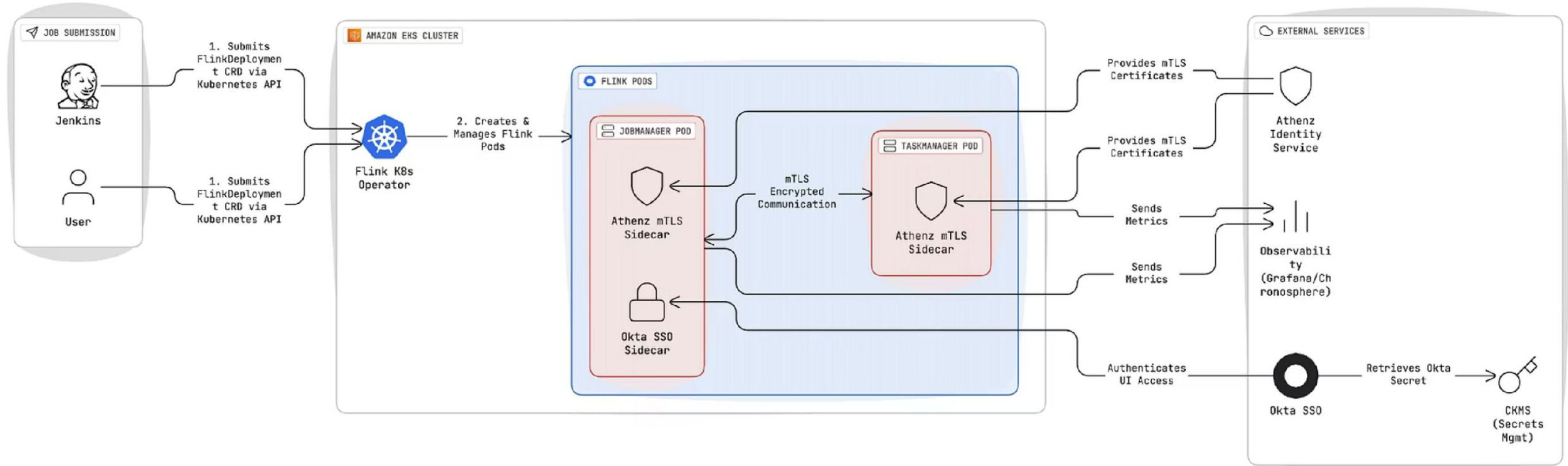
Valid Token: Access Granted



No Token: Redirect to Okta



# Flink Deployment - Architecture



# The Problem: Scaling Data Ingestion

At Yahoo, we faced the critical challenge of processing massive amounts of conversion data from a multitude of partners. This involved ingesting and analyzing **millions of records every single hour**, demanding a robust and highly scalable batch processing solution.



# The Legacy Pains: Our Old Stack

Our existing Oozie + EMR + Pig stack was struggling to keep up with our evolving security and performance needs.

## Old & Unreliable Workflows

Oozie workflows were XML-based, making them difficult to write, maintain, and debug.

## Inefficient Resource Usage:

Persistent EMR clusters led to significant costs from idle compute resources.

## Security and Compliance Gaps:

Retrofitting modern security standards like mTLS and granular IAM was complex and unreliable.

# Airflow to the Rescue



## Apache Airflow

Enables workflow orchestration with DAG scheduling, dependency management, and monitoring for data pipelines



## FlinkKubernetesOperator

Custom Airflow operator for declarative Flink job management via Kubernetes integration.

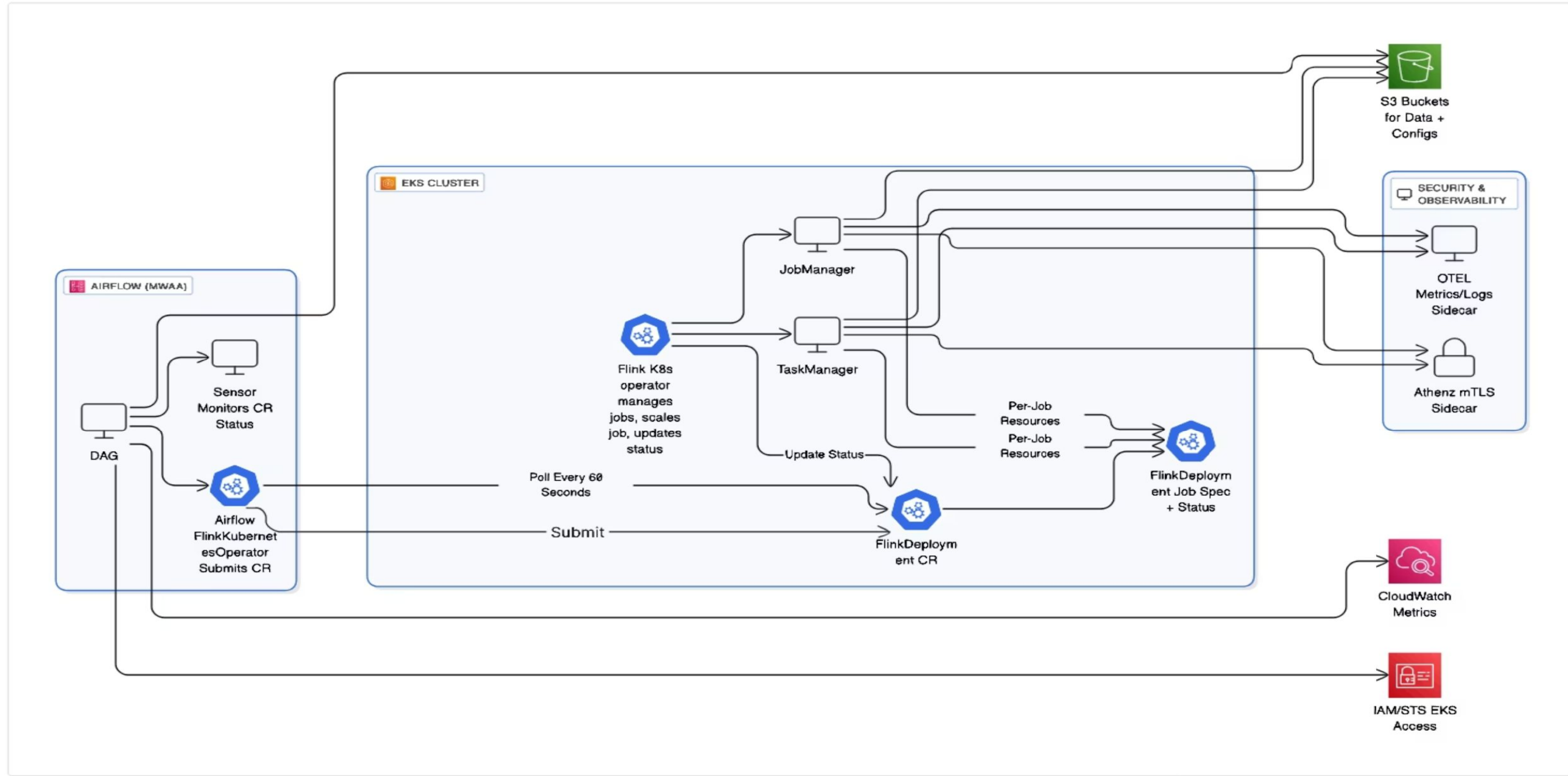


## Automated Lifecycle

Automates Flink cluster setup, job execution, monitoring, and cleanup—minimizing manual effort.

**Result:** From manual & error-prone → automated, scalable, reliable pipelines.

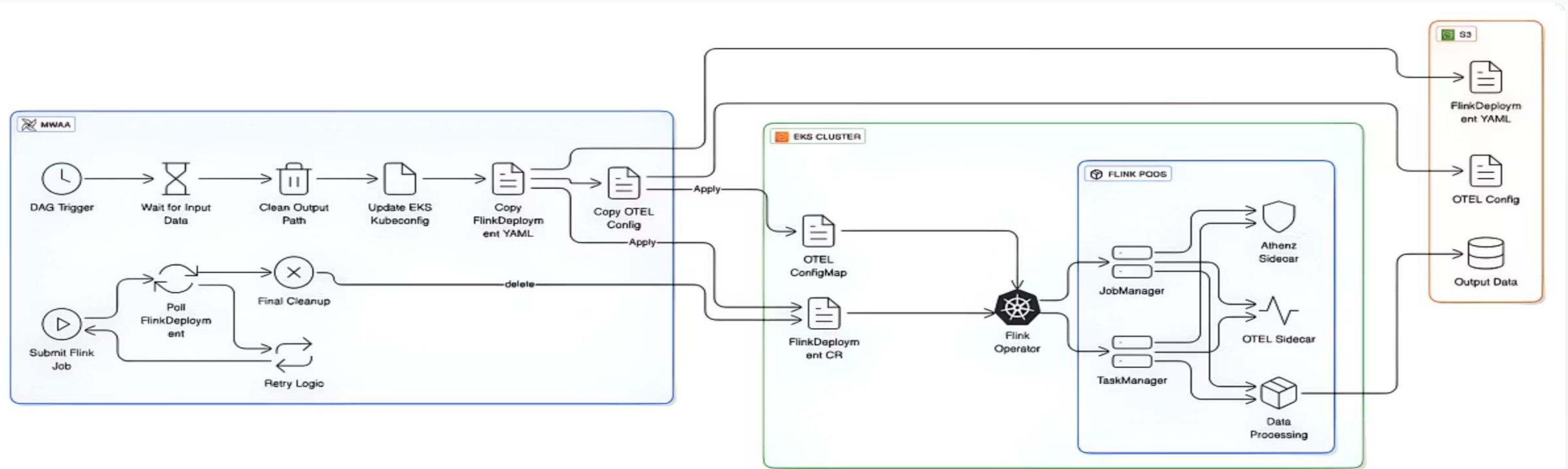
# High Level Architecture



- Isolated compute per job
- Strict access control
- No hardcoded credentials
- Observability



# End Flow to End Flow



# Security Model Challenges

## 1 EKS Athenz Integration

No IAM support, requires custom cert management.

## 2 MWAA Athenz Gap

No native Athenz integration for cert fetch/refresh.

## 3 IAM Bridge Solution

MWAA↔EKS via IAM, Flink internal via Athenz.

## 4 Monitoring Workaround

MWAA monitors via Flink CRDs using EKS IAM.

Hybrid model ensures Yahoo-compliance + AWS integration.



# Cost Controls: How We Achieve Efficiency



## Immediate Cleanup

Cleanup Flink clusters and Kubernetes operators immediately upon job completion - no lingering resources consuming costs.



## Per-Job Isolation

Each job gets its own dedicated cluster, ensuring optimal resource allocation and preventing resource contention between different workloads.



## Automated Lifecycle

Full automation eliminates manual intervention, reducing operational overhead and ensuring consistent resource management practices.

# Production Health Monitoring - Production Engineering DAG

## Why Monitor the Monitors?

- Data pipelines are only as reliable as their orchestration
- Failed DAGs = missed SLAs and data gaps
- Proactive monitoring prevents reactive firefighting

## PE (Production Engineering) DAG: Our Watchdog

- Runs every 5 minutes - continuous health checks
- Queries all application DAGs for failure states
- Publishes metrics to CloudWatch - FailedDagCount per DAG
- Triggers Chronosphere alerts - immediate notification to on-call teams



# Takeaways

**Airflow + EKS + Flink Operator = strong pattern for secure batch.**

**Simplified idle flink cluster removal**

**Security must be baked into orchestration, not added later.**

**Simple deployment**

**Per-job clusters reduce cost and improve isolation.**

**Observability and cleanup are not optional.**

# Reference

**Flink Kubernetes Operator:** <https://nightlies.apache.org/flink/flink-kubernetes-operator-docs-main/>

**MWAA Best Practices:** <https://docs.aws.amazon.com/mwaa/latest/userguide/best-practices.html>

## Questions?

**Connect with us:**

[purushah@yahooinc.com](mailto:purushah@yahooinc.com)

[prakashnm@yahooinc.com](mailto:prakashnm@yahooinc.com)