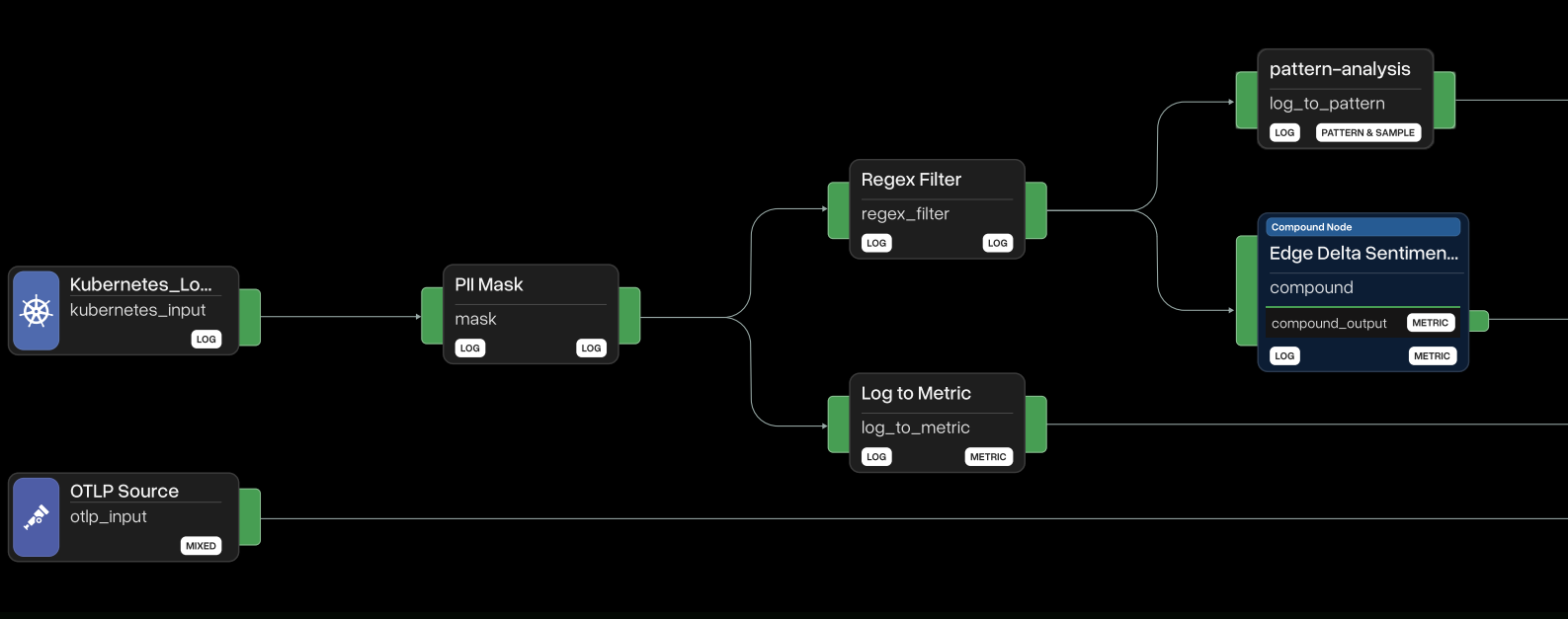
WHITE PAPER

EDGE DELTA

Next Generation Telemetry Pipelines: Low Cost, High Scalability

How Edge Delta's Architecture Delivers Superior Efficiency



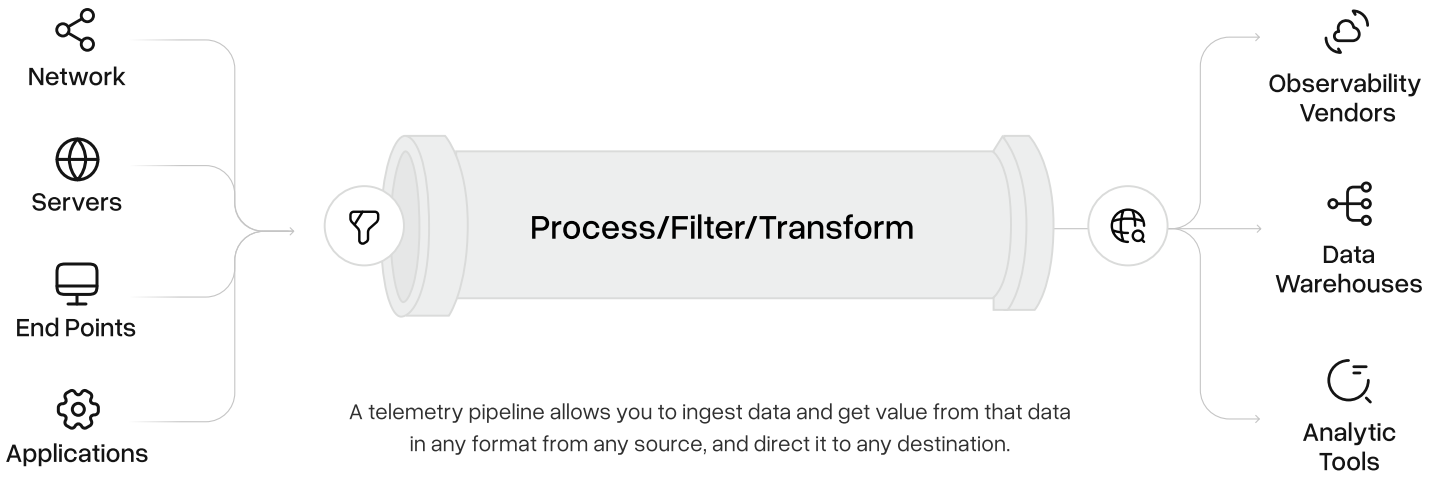
Abstract

As the volume of telemetry data continues to grow exponentially, conventional telemetry pipelines are struggling to keep pace, both in terms of performance and cost-effectiveness. In this white paper we explore four distinct models, with a focus on manageability, cost-efficiency, and scalability. Our analysis reveals that intelligent, end-to-end pipelines, like those pioneered by Edge Delta, far outperform traditional architectures in these areas. Through an in-depth cost comparison experiment, we demonstrate how these next-generation telemetry pipelines not only reduce Mean Time to Detection (MTTD) and Resolution (MTTR), but also dramatically lower total cost of ownership (TCO). By shifting left and beginning the processing of logs, metrics, traces, and events at the source, the Edge Delta architecture enables organizations to handle large volumes of telemetry data without sacrificing performance or incurring exorbitant costs. For enterprises seeking to optimize their data strategies and significantly reduce observability and security expenses, the Edge Delta solution presents a compelling and foundational approach.

Background

Observability frameworks aim to provide users with an intuitive and effective interface to interact with their system's telemetry data, which is vital for maintaining system health and performance. Telemetry pipelines are a key component of any observability solution, as they are the mechanisms by which telemetry data is appropriately collected, processed, and routed from source to destination. Pipeline architectures are typically highly configurable, empowering users to modify their telemetry pipelines to best align with their specific data needs and goals. To achieve comprehensive insight into a system, choosing the optimal observability solution — and therefore the optimal telemetry pipeline — is an absolute must.

Before moving on, we should clarify what exactly is meant by "telemetry pipeline." At a high level, telemetry pipelines are data pipelines that exclusively work with telemetry data — i.e., raw logs, metrics, traces, and events produced by systems and applications. There are a few key components of telemetry pipelines, each of which plays an important role in the collection → processing → routing process.



A telemetry pipeline allows you to ingest data and get value from that data in any format from any source, and direct it to any destination.

Sources are the origins from which the system's telemetry data is created. They can include, but are not limited to: networks, applications, and low-level infrastructure. These sources emit data describing their current state in the form of logs, metrics, traces, and events. Sources serve as pipeline inputs by generating telemetry data streams that feed into the processors.

Processors are the middle-men that ingest telemetry data from sources and manipulate them as needed. Processors can modify the incoming data in a variety of ways, including filtering, shaping, enriching, masking, and more. Once finished, the processors forward the newly created data to the appropriate destination.

Destinations are the locations into which the processed data is loaded. Data can be sent to analytics platforms (e.g., Edge Delta, Datadog, Splunk), archival destinations (e.g., Amazon S3, Azure Blob Storage, Google Cloud Storage), or some combination of both, while monitor alert messages can be sent to downstream alerting tools like Slack.

Telemetry pipelines are an abstraction, referring to the combination of sources, processors, and destinations that ingest, modify, and route telemetry data generated by the system. Well-built telemetry pipelines efficiently extract data from sources, intelligently process that data, and reliably ship it off to the corresponding destinations.

Problem Definition

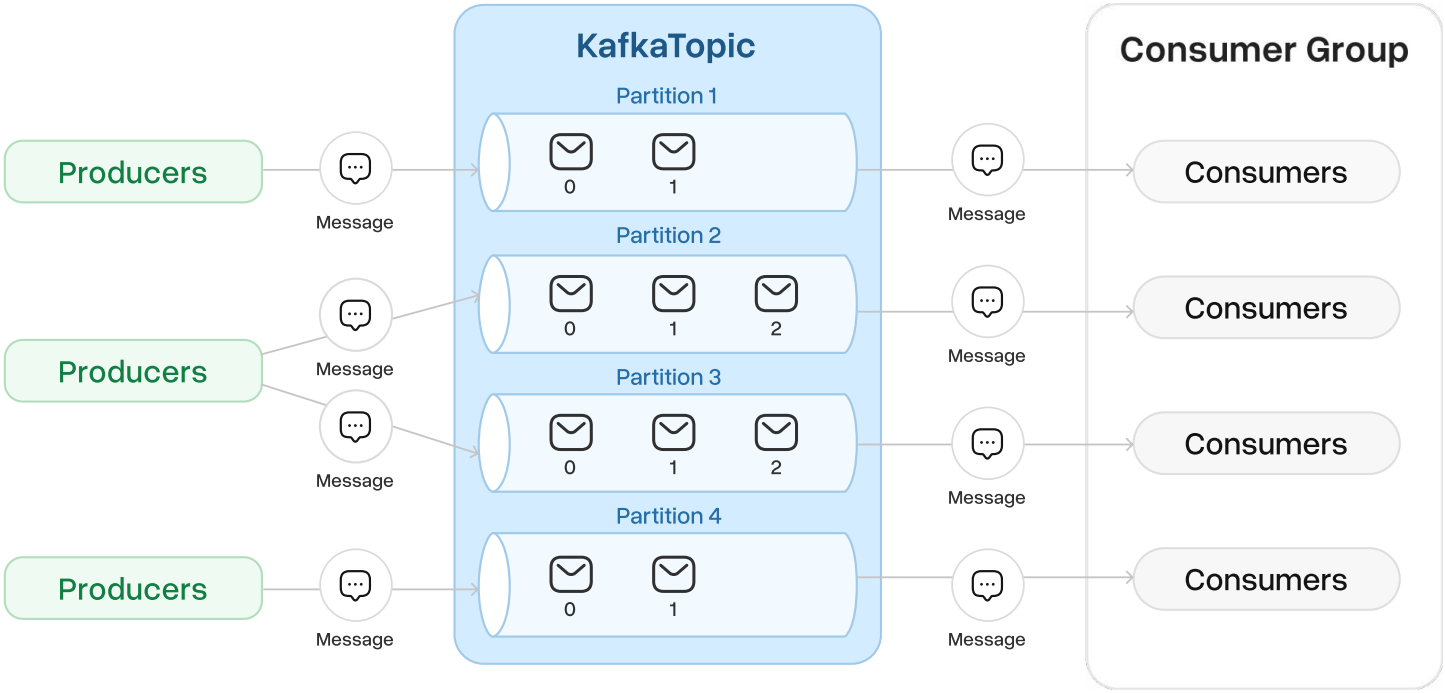
As architectural approaches shift towards cloud-native and distributed environments, and as the quantities of telemetry data continue to grow exponentially, the challenges associated with maintaining an effective and cost-efficient telemetry pipeline will only increase.

Distributed and cloud-native systems can have an incredibly high number of moving parts to ensure the system performs optimally in any scenario. This can increase latency and, if not managed well, lead to higher MTTD and MTTR. A well-designed pipeline helps keep these values low.

Additionally, it's expensive to use and maintain these environments, especially as your data volumes expand. Popular cloud hosting environments like AWS or GCP can easily cost thousands of dollars per month when working with data on the scale of multiple terabytes per day. Organizations are now, more than ever, in need of a fundamentally new approach to controlling and managing their telemetry data in order to minimize costs while maintaining high performance.

Analyzing the Functionality of Potential Solutions

There are a number of different pipeline architectures that organizations can adopt to try to maintain high levels of functionality when dealing with massive amounts of telemetry data. One of the most popular pipeline tools used is Kafka, which follows the "centralize-then-analyze" approach of collecting data from multiple sources and centralizing it before processing. It is an event streaming platform that can collect, process, and store streaming data in real-time. Its distributed architecture makes it a popular choice for processing telemetry data, particularly at scale.



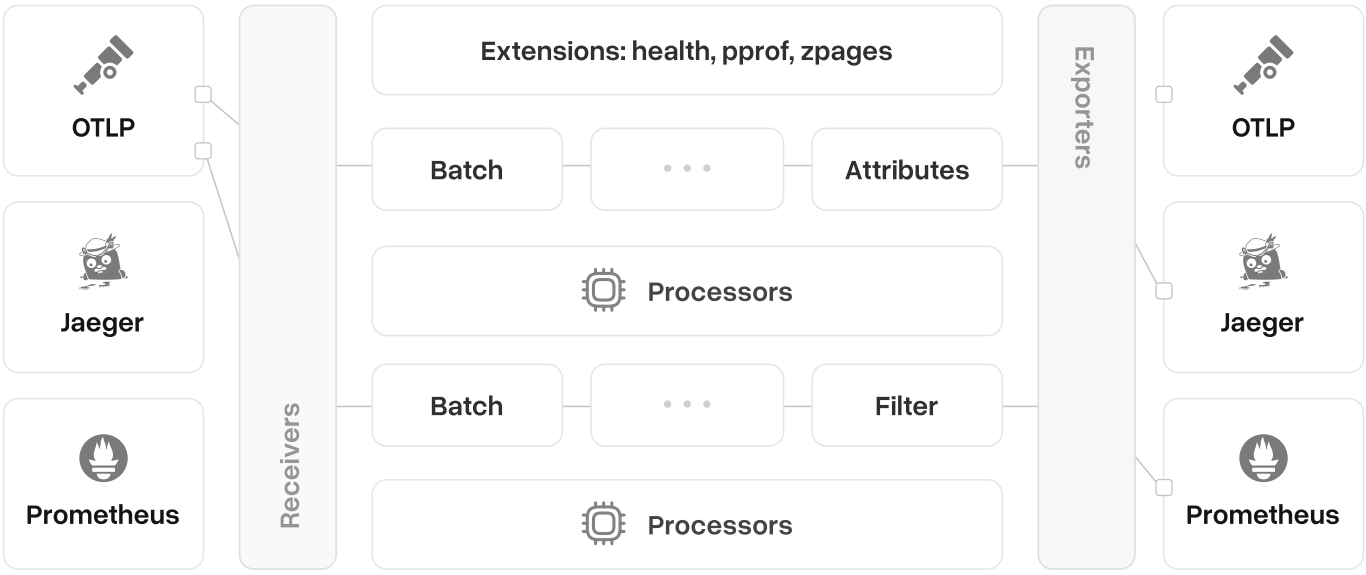
> *Kafka manages topics, which are user-defined categories used to organize incoming data. Each piece of data can go to one or multiple topics, where it can then be processed and routed to consumers appropriately.*

Kafka is technologically complex, and as such Kafka solutions often require constant maintenance and monitoring to ensure proper performance. While Kafka has the flexibility to manage variable workloads, it can fall short of hitting its functional goals. If, for instance, data is not evenly distributed across partitions as it flows into the system, certain workloads can experience high levels of latency. Partition rebalancing can fix this issue, but it might overload brokers and, in the worst case, risk the loss of associated data.

A Dynamic Architecture: Balancing Edge and Cloud

End-to-end pipelines are a new approach to telemetry pipeline architectures that offer flexibility and balance when compared to traditional centralized cloud architectures. By shifting the processing step to within the user environment, much closer to the data sources, the transformation and routing operations become far less resource intensive.

One such tool is the OpenTelemetry Collector, a vendor-agnostic tool that can receive, process, and export telemetry data. It functions as an all-in-one pipeline solution and requires minimal effort to instrument it into a codebase.

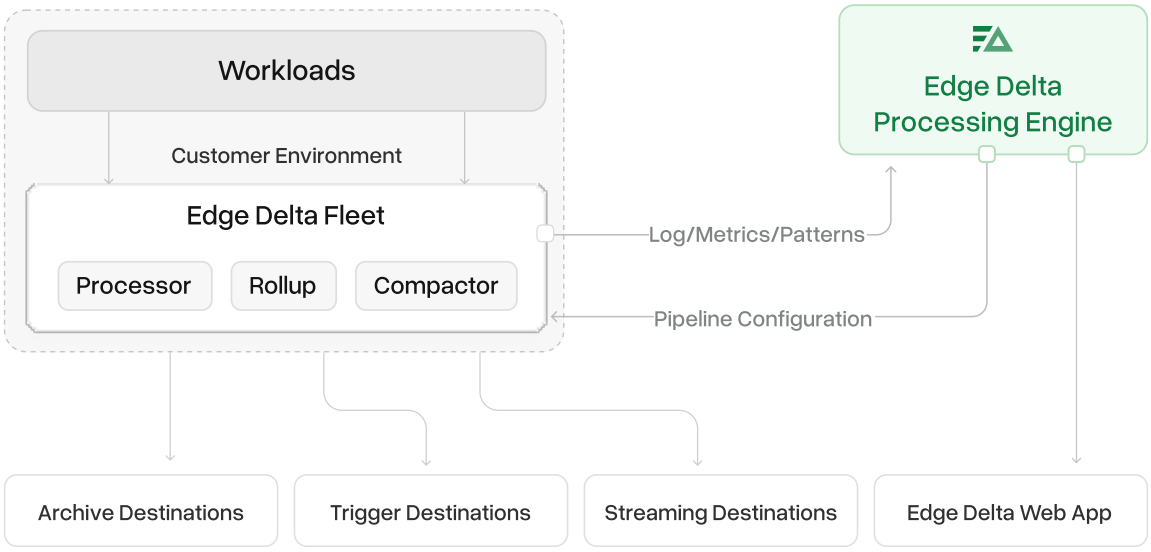


> *The OpenTelemetry Collector is a combination of receivers, processors, and exporters that collect data from specified sources, batch process it, and send it to configured downstream destinations.*

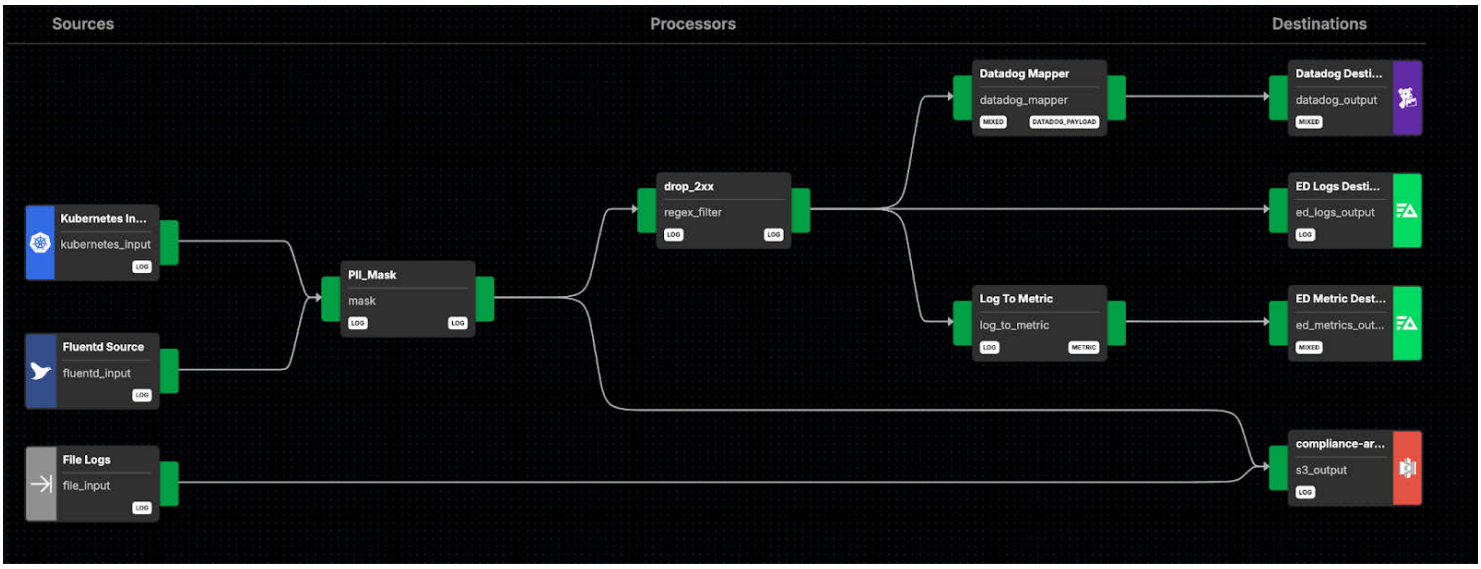
OpenTelemetry provides a standard baseline for observability functionality. It can aggregate telemetry data from its sources through its receivers, batch process it through its processors, and export it to backend observability platforms through its exporters.

Edge Delta's telemetry pipelines are another example of a modern, end-to-end pipeline solution, allowing users to shift left and begin processing logs, metrics, traces, and events at the source, and shipping them off to any number of downstream destinations.

Here's how it works:



The Edge Delta fleet lives directly within the customer environment and ingests data as it is produced from system workloads. After processing the logs in a variety of different ways, they are shipped to their respective long-term destinations, as per user specifications. The pipeline might look something like this:



> *In this example, data is collected from Kubernetes, Fluentd, and local files that logs are being written to. Data is then masked to remove any PII, and all logs that capture successful response codes (200–299) are dropped. Finally, data is formatted appropriately (all logs being sent to Datadog are mapped to the Datadog format before being pushed, and logs are converted to metrics before being pushed to the Edge Delta Metrics backend), and sent to the respective downstream destination.*

One of the major benefits of this approach is processing efficiency. Traditional pipeline approaches require each piece of telemetry data to be sent downstream for processing, regardless of importance. This is incredibly inefficient, increasing the latency gap from data generation to visibility in the backend. Users can vary the methods by which the data is processed, but it isn't a solution built for the increasing volumes of telemetry data that organizations now generate year over year.

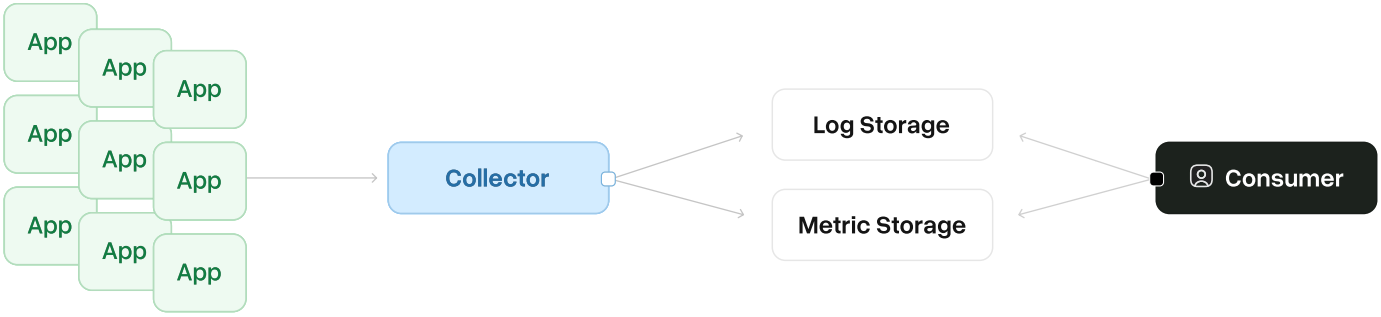
Edge Delta circumvents this problem intelligently. Instead of processing the data in its entirety, an agent fleet sitting directly within the environment can decide, in real time, which telemetry data is most important, either processing directly and/or sending to be processed only the highest priority data.

From an Edge Delta perspective, our agent fleet contains in particular a rollup and compactor component, the former aggregating metric data by optimizing data frequency and cardinality, and the latter compressing and encoding metric and log data into more efficient formats. This preprocessing notably reduces downstream storage needs, and enables accelerated data retrievals in observability tools. Additionally, the fleet is highly scalable; adding new agents to the fleet is a simple process, and the overhead for managing them is low.

While modern telemetry pipeline solutions are functionally optimal, how do they compare to other telemetry pipeline models from a cost perspective? The remainder of this white paper will be focused on experimentally determining the TCO of multiple traditional and next-generation architectures to determine which is optimal from a cost perspective.

Experimental Design

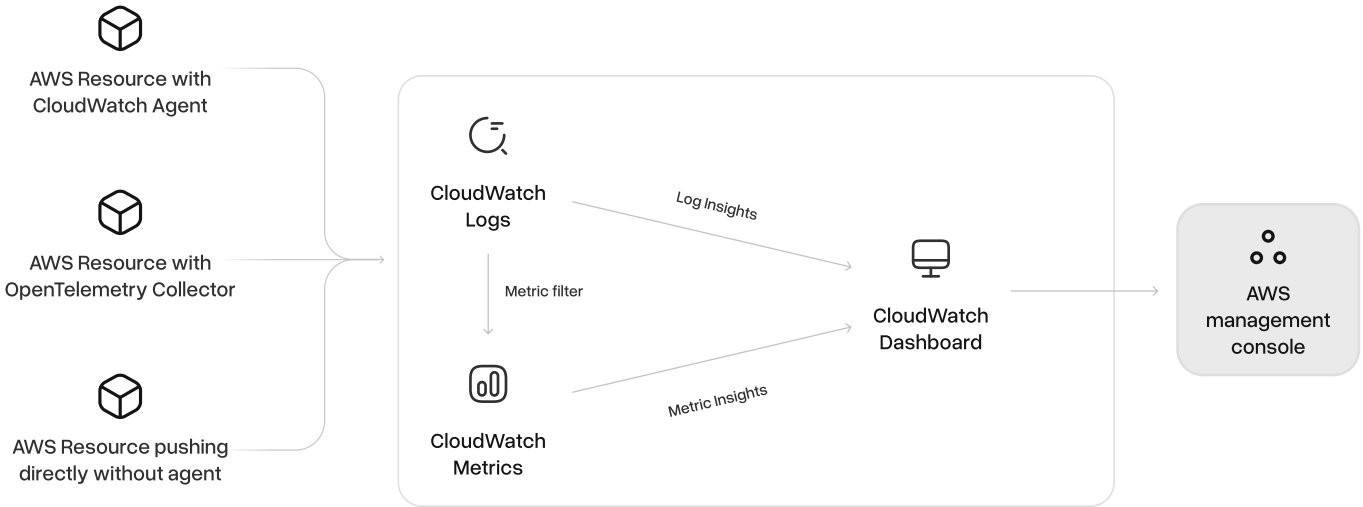
For the scope of this experiment, we chose four distinct pipelines that leverage popular data collection and aggregation tools. For simplicity's sake, we experimented with performance on only the most common pipeline functionality: transforming logs into metrics via log aggregation. Logs were converted to metrics via regular expressions.



Before diving deeper into the experiment, it is important to discuss the environments and infrastructures we used. First, each pipeline was hosted in AWS — on EC2 instances, specifically — and Amazon Cloudwatch and Amazon S3 were used for querying and storing logs, respectively. Additionally, for Pipeline #2, Kafka was hosted using Confluent, a Kafka-based cloud platform. Here is a short breakdown of these tools:

Amazon CloudWatch

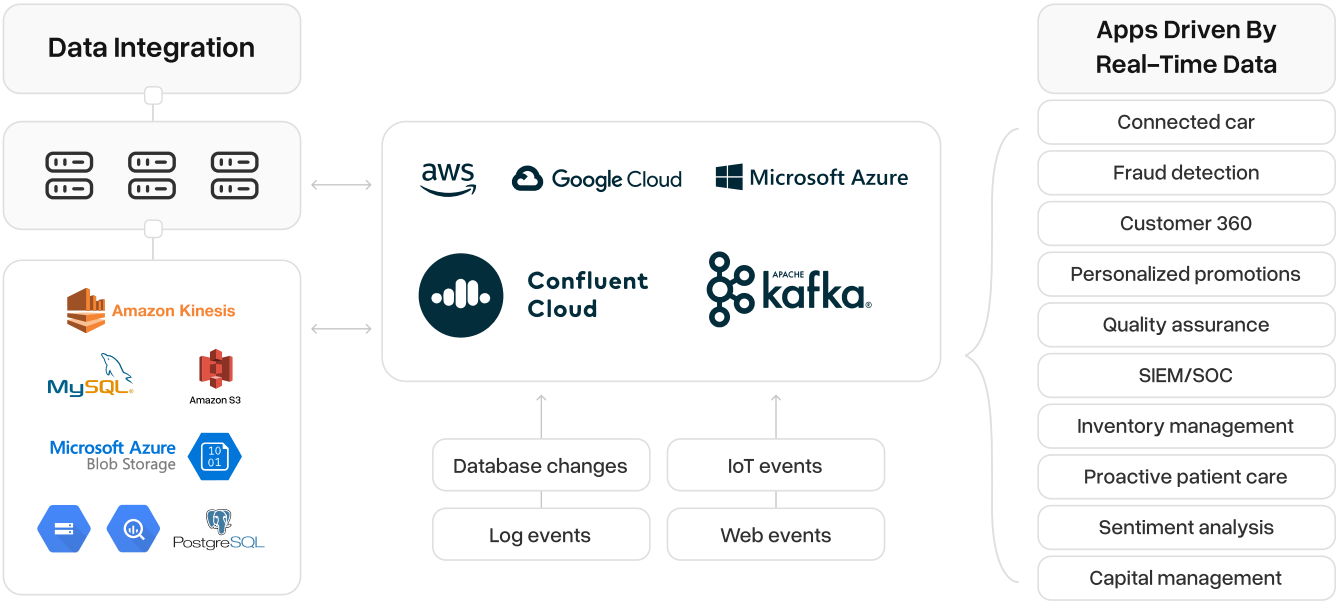
Amazon CloudWatch is Amazon's native monitoring tool, enabling users to track and derive insights from their various AWS services.



CloudWatch Logs track all ingested log data, and CloudWatch Metrics track metric data about the performance of your systems. CloudWatch also has a Logs Insights API, which enables users to query logs into metrics however they see fit.

Confluent

Confluent is a cloud-native data streaming platform that utilizes Apache Kafka for stream-processing.



It effectively serves as a cloud wrapper around Kafka, maintaining Kafka's benefits and adding cloud-native benefits as well.

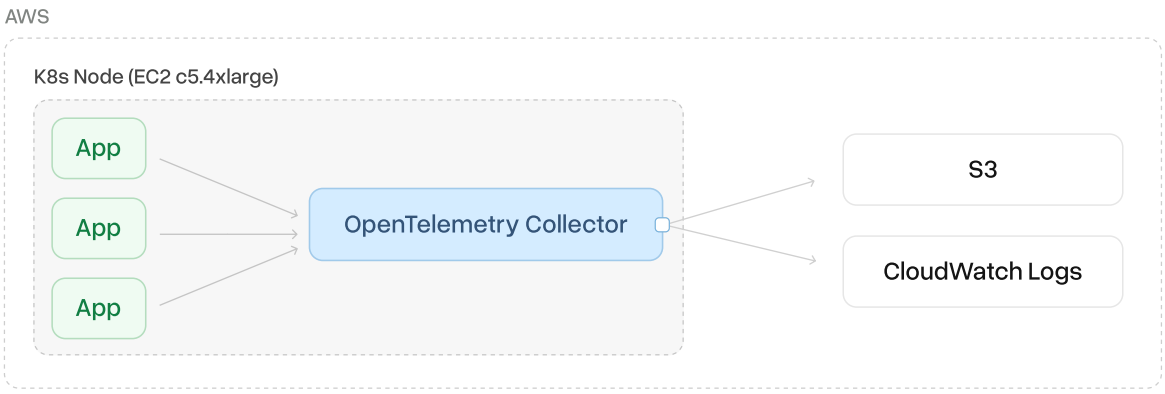
Testing Methodology

To test these pipelines, we conducted a 30-minute experiment within a newly provisioned Kubernetes cluster; each distinct pipeline and test setting configuration was provisioned its own cluster to work on. Each pipeline was hosted in a mimicked Kubernetes cluster environment by spinning up a K3s cluster on top of a single Amazon EC2 c5.4xlarge machine. We generated synthetic logs in the Apache access log format using https://github.com/mingrammer/flog. We ran two distinct test cases to determine which pipeline is the most cost-effective: 1 terabyte-per-day log throughput, with 5 metrics to query; and 1 terabyte-per-day log throughput, with 20 metrics to query (we will explain how metrics are queried shortly).

Here are the pipelines included in the experiment:

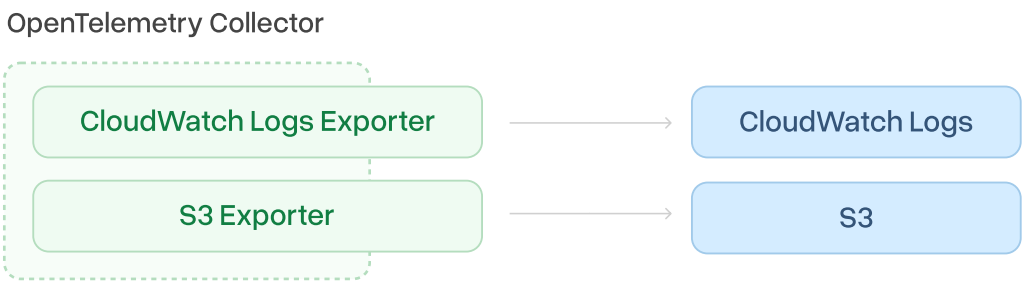
Pipeline #1: OpenTelemetry Collector

In this pipeline architecture, logs are collected from the source using an OpenTelemetry Collector and pushed directly into Amazon CloudWatch Logs.

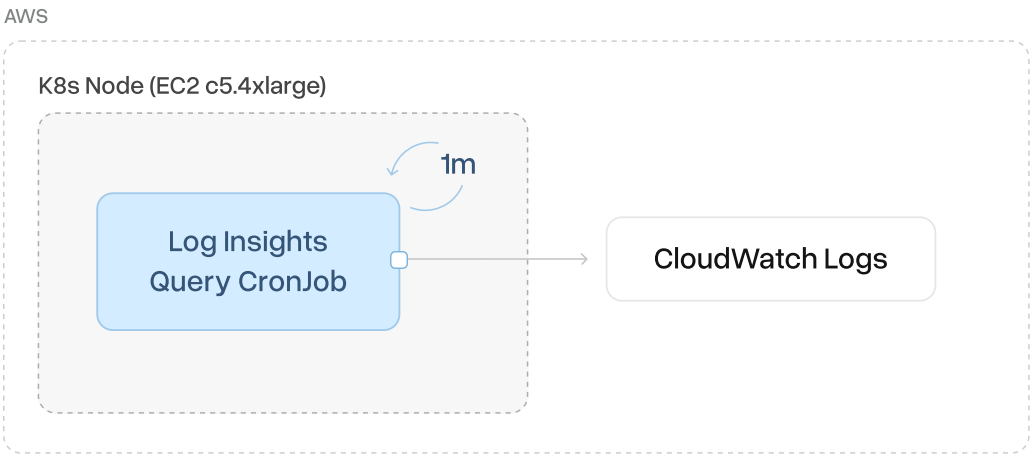


The OpenTelemetry collector collects the synthetic logs and exports them to the specified destination in batches.

In our experiment, we specified two exporters with the collector: CloudWatch Logs Exporter and S3 Exporter, which send data to CloudWatch Logs and S3 respectively.

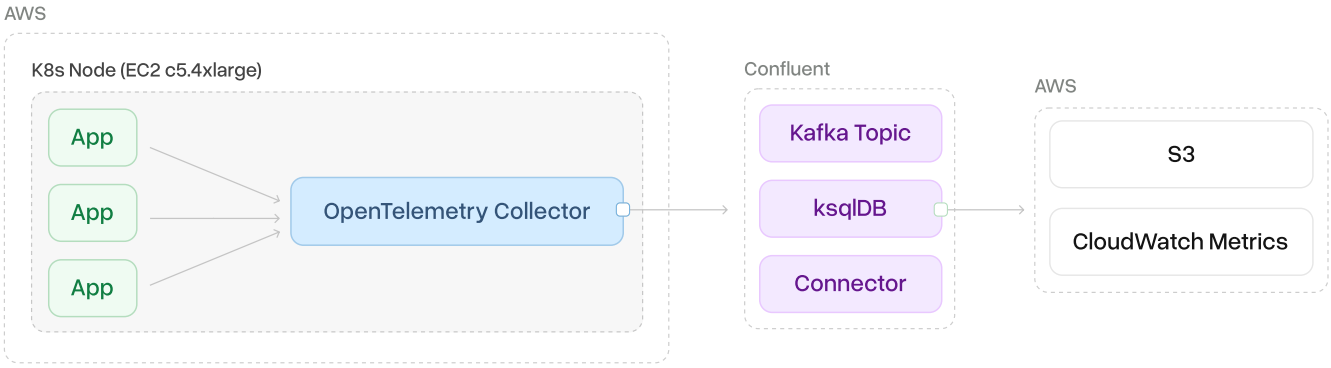


Once in CloudWatch Logs, a CronJob runs every minute to query the data. The CronJob is a containerized Go application which uses the CloudWatch Logs Insights API to query log groups into metric values. This functionality mimics the functionality of fetching metric data to render in observability dashboard interfaces. The metrics are calculated from logs aggregated from the previous 10 minutes.



Pipeline #2: Adding Kafka Middleware

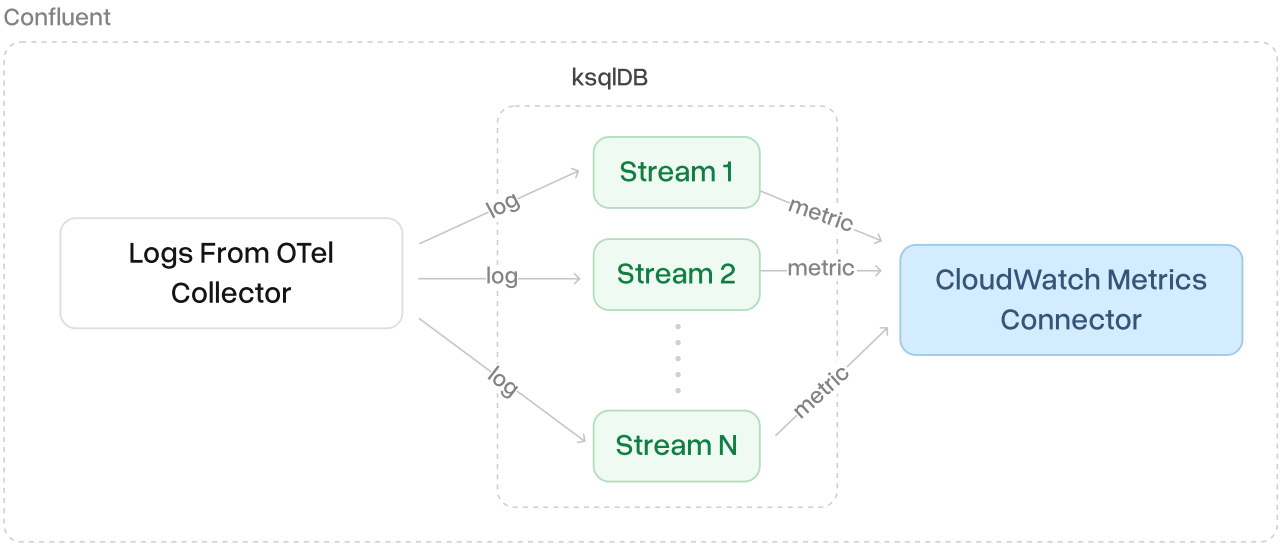
This setup is structurally similar to the previous pipeline, with the addition of Kafka middleware to handle the generated data streams.



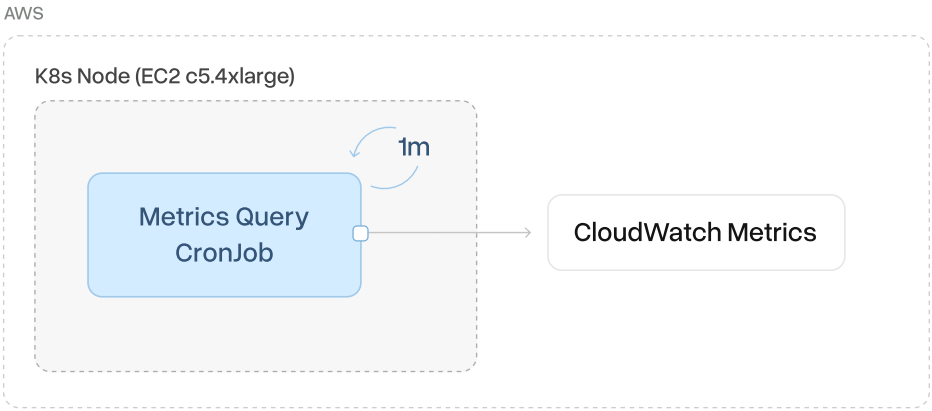
In this architecture, logs are collected via the OpenTelemetry Collector and pushed outside the customer environment to a dedicated Kafka topic. The logs are then consumed from the Kafka topic and converted to metrics using ksqlDB, and then pushed to CloudWatch Metrics. We used Confluent as a third-party cloud-hosted Kafka provider, which is one of the most inexpensive options to host a Kafka solution. The key difference between this and the previous architecture is that the Kafka connector aggregates logs internally and pushes the created metrics directly to CloudWatch Metrics, instead of creating metrics on demand during the log query, as the previous pipeline architecture does.

To convert log data into metrics, we used Kafka and ksql as follows:

- Split the logs coming in batches to the appropriate Kafka topic
- Process and clean the incoming log streams
- Convert log messages to specific metric data points in the CloudWatch MetricDatum format
- Push custom metrics to CloudWatch Metrics and raw data to S3 via Connectors



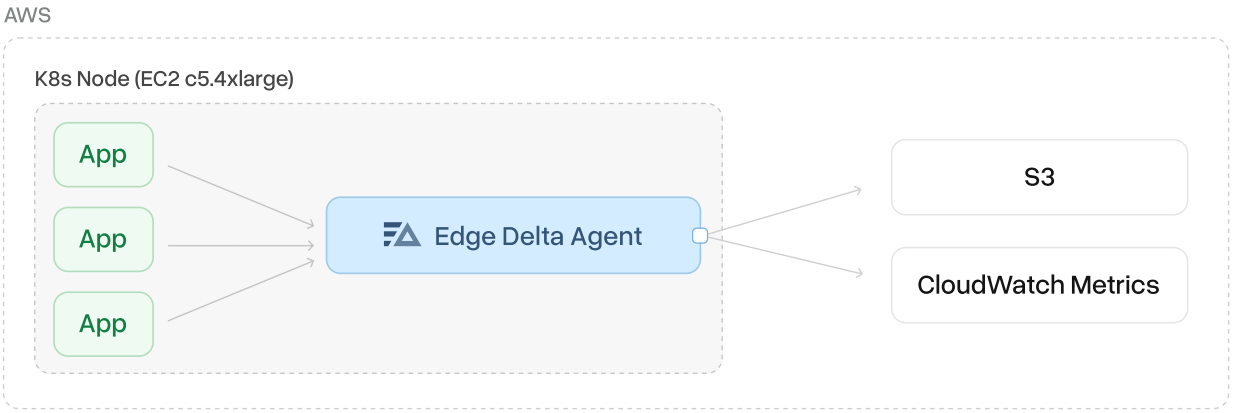
To query metrics, we used the same CronJob and modified it to query CloudWatch Metrics via the GetMetricsStatistics API to retrieve aggregated metrics from the last 10 minutes.



Pipeline #3: Edge Analytics

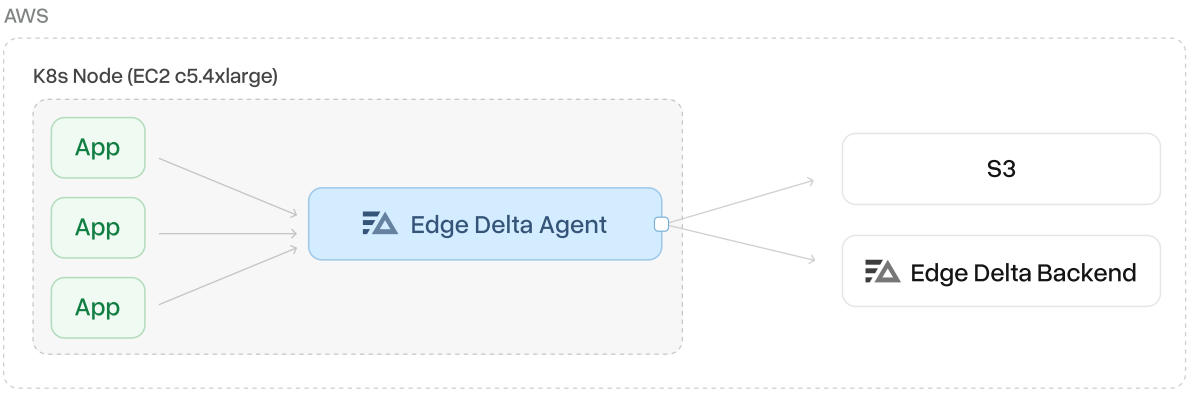
In this setup, an Edge Delta fleet resides directly in the customer environment, right where the logs are generated. The agents in the fleet tail logs, apply a set of regexes, and convert them into metrics. These metrics are then directly shipped into S3 and CloudWatch Metrics.

When querying the logs on the 10 minute window scale every minute, we used the same approach as in the previous pipeline, where we used the GetMetricStatistics API to query them directly from CloudWatch Metrics. This saves on both compute and cost compared to generating the metrics on demand, and removes the need to host a Kafka solution as middleware in your pipeline.



Pipeline #4: End-to-End Pipeline Architecture

This setup moves one step further from the previous pipeline by shifting away from CloudWatch Metrics entirely. Instead, the agent fleet derives metrics from the log streams generated at the source and forwards them directly to the Edge Delta backend. Since the metrics are converted at the source and remain in the Edge Delta environment, there is no need to run any query job from external sources.



For each test, we created the data ingestion pipeline, which was configured to collect the raw synthetic data from within its Kubernetes environment. Each pipeline architecture was designed to stream all logs directly into S3. Pipelines #1–3 were also designed to:

- Ingest analyzed data into CloudWatch
- Run a CronJob to query the specified number of metrics (5–20) every minute, on a 10 minute window aggregation

Each case ran in the AWS us-west-2 region, for consistency.

For each pipeline–test combination we calculated the TCO over 30 minutes of runtime, and extrapolated to find the TCO over a 24-hour period. Cost calculations were calculated using two distinct methods:

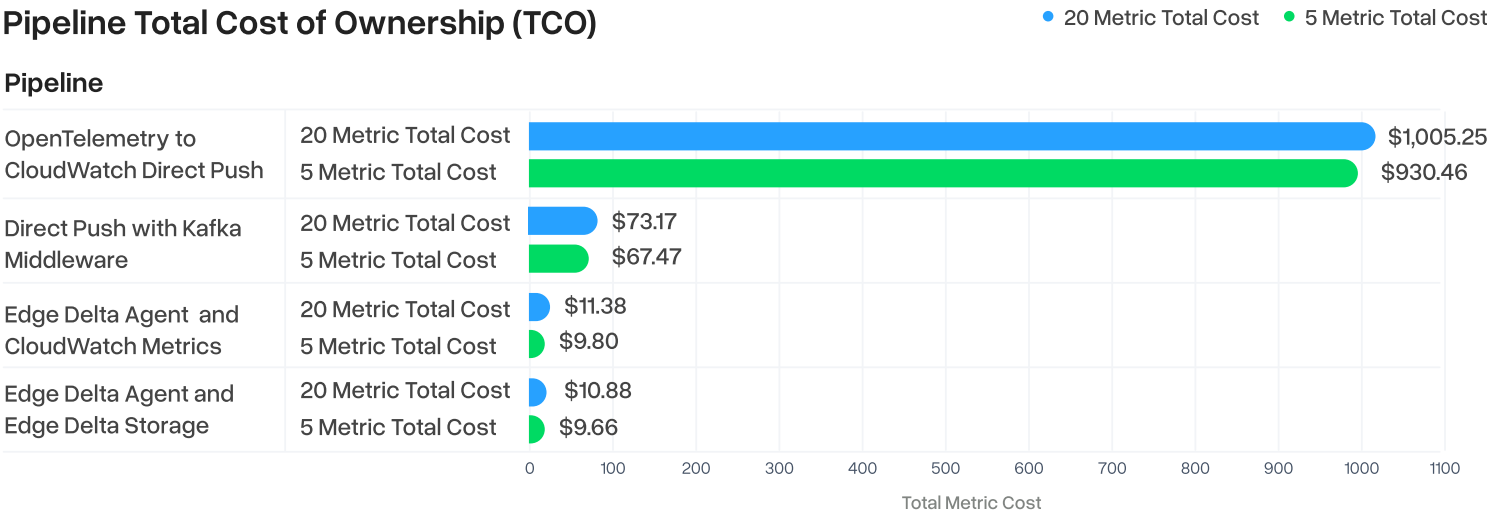
- Collecting pricing rates (per GB of data queried, for example) from AWS and Confluent pricing information guidelines, and calculating the total cost based on the amount of data generated, processed, stored, and queried.
- Recording the daily total cost per service (for AWS only) via the AWS Cost Explorer, and estimating the per-component usage based on resource allocation. In other words, assigning costs to components running in AWS (OpenTelemetry, Edge Delta fleet) based on the proportion of CPU and memory usage each one utilized.

Here is the cost breakdown for the different tools:

- For CloudWatch, using the official pricing information:
  - Standard data ingestion: $0.50 per GB
  - Data storage: $0.03 per GB
  - Log Insights queries: $0.005 per GB of data scanned
  - Log Insights query cost can be calculated by volume of data scanned, which is available post-query.
- For EC2 Compute, using the pricing information:
  - $0.68 on-demand hourly rate, with 16 vCPUs and 32 GiB of memory
- For OpenTelemetry Collector and Edge Delta Agent Fleet using estimation:
  - As the OpenTelemetry Collector and Edge Delta agent fleet are hosted in an EC2 instance, we calculated their respective costs based on the proportion of total vCPU and memory allocated for them to run
- For Confluent Kafka, we used the official pricing information:
  - We reviewed the total cost using the Confluence cost analysis tool post-experiment

Results

Pipeline Total Cost of Ownership (TCO)



Pipeline	Daily Cost (20 Metrics)
Pipeline #1: OpenTelemetry Collector → CloudWatch Logs	$ 1005.25
Pipeline #2: OpenTelemetry Collector with Kafka Middleware	$ 73.17
Pipeline #3: Edge Delta Agent → CloudWatch Metrics	$ 11.38
Pipeline #4: Edge Delta Agent → Edge Delta Backend	$ 10.88

Details

Case	Compute (OpenTelemetry Collector/Edge Delta Agent)	CloudWatch Storage/ Ingestion	Kafka (Confluent)	Platform Pricing (Edge Delta)	S3 Storage	Query (CloudWatch Logs/ Metrics)	Daily Total
Direct Push (OpenTelemetry Collector → CloudWatch Logs)	$2.37	$902.37	–	–	$0.79	$24.93 ($99.72 – 20 metrics)	$930.46 ($1005.25 – 20 metrics)
Direct Push with Kafka Middleware and CloudWatch Metrics	$3.06	$0.05 ($0.2 – 20 metrics)	$63.49 ($68.82 – 20 metrics)	–	$0.79	$0.075 ($0.3 – 20 metrics)	$67.47 ($73.17 – 20 metrics)
Edge Delta Agent and CloudWatch Metrics	$2.04 ($3.26 – 20 metrics)	$0.05 ($0.2 – 20 metrics)	–	$6.83	$0.79	$0.075 ($0.3 – 20 metrics)	$9.80 ($11.38 – 20 metrics)
Edge Delta Agent and Edge Delta Backend	$2.04 ($3.26 – 20 metrics)	–	–	$6.83	$0.79	–	$9.66 ($10.88 – 20 metrics)

Discussion

The experimental results reveal significant discrepancies in total cost of ownership (TCO) across the evaluated telemetry pipeline architectures, particularly under high-volume data ingestion scenarios. For processing approximately 1 TB of data daily, the OpenTelemetry Collector pipeline (Pipeline #1) incurred a daily cost of approximately $1,000 when handling 20 metric queries, while Pipeline #2, utilizing Kafka middleware, reduced this to around $75. In contrast, the two Edge Delta architectures (Pipelines #3 and #4) achieved exceptional cost efficiency, with a daily TCO of approximately $11.

Additionally, the rate of cost increase with scaling metric queries further underscores the efficiencies of Edge Delta's architecture. For Pipeline #1, increasing the metric query count from 5 to 20 led to an incremental cost of $70, whereas Pipeline #2 saw a $7 increase for the same scaling. In comparison, the Edge Delta pipelines incurred only a marginal $1 increase in TCO, demonstrating robust scalability in the face of higher query volumes.

A closer examination of cost distribution reveals that data ingestion and processing operations constitute the bulk of expenses in traditional pipelines. Specifically, for Pipeline #1, 90% of the total cost is attributed to log ingestion and storage in CloudWatch Logs, while in Pipeline #2, Kafka's processing and streaming functionalities account for approximately 93% of overall expenses. Conversely, Edge Delta's pipelines minimize these processing costs by executing data transformations at the source. Here, the primary cost driver — comprising roughly 67% of TCO — is the Edge Delta platform's standard pricing, estimated at $7 per day for the 1 TB data throughput.

Projected annually, these cost differences become even more striking. For enterprises handling 1 TB of daily data, annual observability expenses can escalate to approximately $360,000 with Pipeline #1 and $30,000 with Pipeline #2. By comparison, Edge Delta's next generation architecture maintains a projected yearly cost of around $4,000 under identical conditions, even with considerable growth in log volume and metric queries. At 1 PB of daily data — a level of data volume that's common in many large enterprises — $360 million or $30 million respectively to spend on Observability or Security is entirely infeasible. This demonstrates not only the need for next generation architectures but also Edge Delta's substantial advantage in scalability and cost-efficiency, positioning it as an ideal solution for enterprises with high telemetry data demands.

Conclusion

As the volume and complexity of telemetry data grows exponentially, traditional telemetry pipelines are becoming increasingly inefficient, both in terms of performance and cost. Edge Delta's novel architecture offers a breakthrough solution, transforming how logs, metrics, traces, and events are able to be processed and scaled. By shifting left and moving the initialization of data processing closer to the source, Edge Delta reduces latency, accelerates MTTD and MTTR, and enables organizations to handle growing data volumes far more efficiently than legacy architectures.

Our analysis demonstrates that Edge Delta's architecture significantly outperforms traditional pipeline models in both functionality and cost-effectiveness. The ability to start the processing of telemetry locally minimizes the need for over-engineered, costly downstream infrastructure and reduces TCO dramatically.

As data scales, these cost savings become even more pronounced, making Edge Delta's solution not just efficient, but purpose built for enterprises that deal with large volumes of streaming data.

For organizations looking to optimize both cost and performance while managing increasing telemetry demands, Edge Delta offers a robust, scalable, and efficient option worthy of serious consideration. Traditional pipelines are simply not built to handle the massive growth in telemetry data efficiently or cost-effectively. Edge Delta's approach eliminates these bottlenecks by providing a scalable, highly-performant solution that delivers significant savings without sacrificing functionality. As telemetry demands continue to rise, this new approach stands out as the next-generation solution that organizations need to evaluate to remain competitive and cost-efficient in the data-driven future.

Get Up and Running in Minutes

Visit EdgeDelta.com