VOLT ACTIVE DATA

VoltSP

Make Decisions in the Stream, Not After It

Kafka and its peers move data at enormous scale. But moving data is not deciding on it. The moment a decision needs trusted context, such as a balance, a session, a device status or an inventory level, the event leaves the stream for a round trip to another system. Each round trip adds latency, another component to operate and another place to fail.

When decisions lag the data, the consequences are immediate:

- Fraud Gets Through
- Offers Miss the Moment
- Enforcement Lags
- Cloud Costs Climb
- Stacks Sprawl

The longer the path from event to decision, the less the decision is worth. But bolting a cache, a rules engine and a hand-written client application onto the stream only moves the problem. You shouldn't have to choose between deciding fast and deciding right.

VoltSP eliminates that trade-off. It runs stream processing pipelines that read from Kafka, Pulsar, MQTT and other sources, call Volt for the live state a decision needs, and push the result downstream in milliseconds. One pipeline, from event to decision to action.

How it works

A VoltSP pipeline is a source, one or more processors and a sink, defined using a visual workflow editor or using YAML configuration or Java code. Sources read from Kafka, Pulsar, MQTT, HTTP, S3 and files. Processors filter, transform, aggregate over time windows, run machine-learning models, or call Volt stored procedures to read and update state. Sinks write to Kafka, Pulsar, HTTP, S3, Elasticsearch, Iceberg tables or back into Volt.

Pipelines run as self-contained pods on Kubernetes, with a Helm chart provided, or on bare metal. There is no external scheduler or resource manager. Each pipeline scales independently of the message broker and of the Volt cluster, so processing capacity is sized to the workload rather than to the infrastructure around it.

Stateless work happens in the pipeline. Stateful decisions happen in VoltDB, where a single partitioned procedure reads the context, applies the logic and records the outcome in one transaction. VoltSP commits each batch of events only when every downstream operation has succeeded, and circuit breakers hold the pipeline back when a downstream system is struggling rather than overwhelming it.

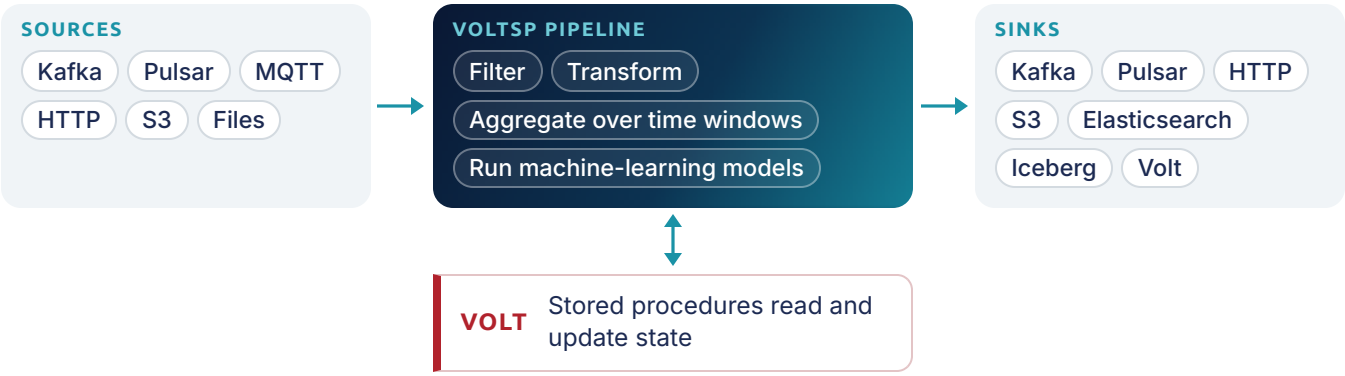
What this means in practice

Decisions with context. Every event can be judged against current state rather than a stale copy: the balance as it is now, the session as it stands, the profile as it was last updated. For charging, policy and fraud systems, that is the difference between a decision and a guess.

Less data moved, less data stored. Filter, deduplicate and aggregate in the stream so that only what matters reaches downstream systems. Deciding early cuts cloud ingestion, storage and processing cost, and it is the one benefit VoltSP delivers even before Volt is involved.

One pipeline, not a stack. VoltSP replaces the client applications, caches and glue code that normally sit between the broker and the data platform. Fewer network hops, fewer failure points, one thing to operate and scale.

Built for production from day one. Batch commit with retry, local and global circuit breakers, and Kubernetes-native deployment mean the pipeline degrades predictably when something downstream fails and recovers on its own when it returns.



One pipeline, from event to decision to action.

From data pipeline to decision pipeline

	Typical Stream Processor	VoltSP with VoltDB
Stateful decisions	External lookup per event, via a cache or a database round trip	Transactional call into Volt from inside the pipeline
Consistency	Eventual; dual writes and stale reads	Decision and state change in one ACID transaction in Volt
Time-based aggregation	Application code	Tumbling, sliding and session windows built in
Deployment	Separate processing cluster to size and operate	Self-contained Kubernetes pods, scaled per pipeline
Machine learning	Separate inference service	ONNX models run in the pipeline, hot-reloaded from MLflow
Failure handling	Custom retry and back-pressure logic	Batch commit, retry and circuit breakers built in

Architectural properties

- Pipelines defined in a visual editor or in YAML configuration files or in Java DSL (for maximum control); processors in Java, JavaScript or Python
- Sources: Kafka and Kafka-compatible brokers, Pulsar, MQTT, HTTP, S3 and SQS, files, network sockets, Elastic Beats. Sinks: Kafka, Pulsar, MQTT, HTTP, S3, Elasticsearch, Iceberg, syslog, files, and Volt procedures or bulk insert
- Window aggregation with tumbling, sliding and session windows; Parquet encoding for data-lake sinks
- Batch processing with every asynchronous operation tracked to completion; automatic retry; local and global circuit breakers with configurable failure rate, window and delay
- Kubernetes-native: Helm chart, independent scaling per pipeline, no external scheduler; bare metal supported
- Machine-learning inference in the pipeline: ONNX and ONNX GenAI processors on CPU or GPU, with model hot reload from an MLflow registry. These capabilities are the foundation of Active(AI)

Key outcomes

Decide in the stream	Act on each event while it is still relevant, with trusted context
Cut downstream cost	Filter and aggregate before data reaches cloud ingestion and storage
Simplify the stack	Remove the client applications, caches and glue code between broker and data platform
Scale what you need	Pipelines scale independently of the broker and of Volt
Bring ML to the data path	Run and update models in the pipeline, with no separate inference tier

AVAILABILITY

VoltSP 1.8 is available now and is licensed as the Active(SP) feature of a Volt license.

Book a technical call voltactivedata.com/contact

Volt Active Data is the real-time decisioning layer for mission-critical systems.