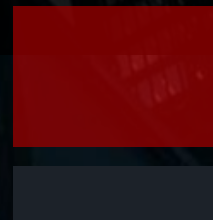
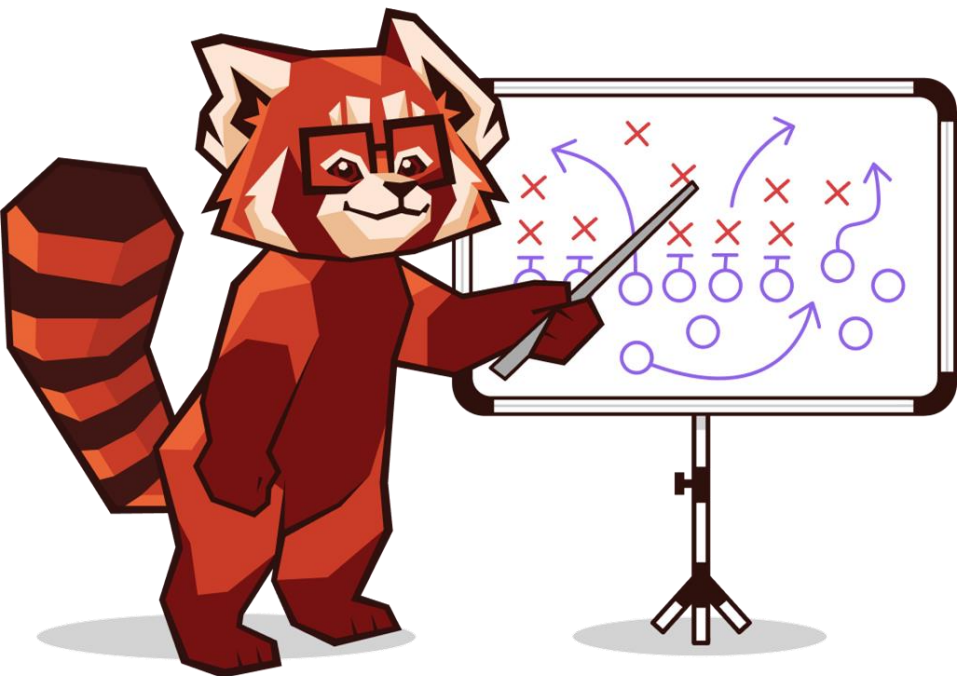




Real-Time AI & Agentic Systems with Redpanda

Harnessing Agentic AI & Redpanda to Automate, Optimize, and Scale AI-Driven Workflows





Data Pipelines, Connectors & Enrichment(Mixed Roles)

Introduction to Redpanda Connect



- Architecture and philosophy
- YAML-based declarative pipeline configs
- Prebuilt connectors and integration targets

What is Redpanda Connect?

Redpanda Connect is a data streaming service for building scalable, high-performance data pipelines that drive real-time analytics and actionable business insights.

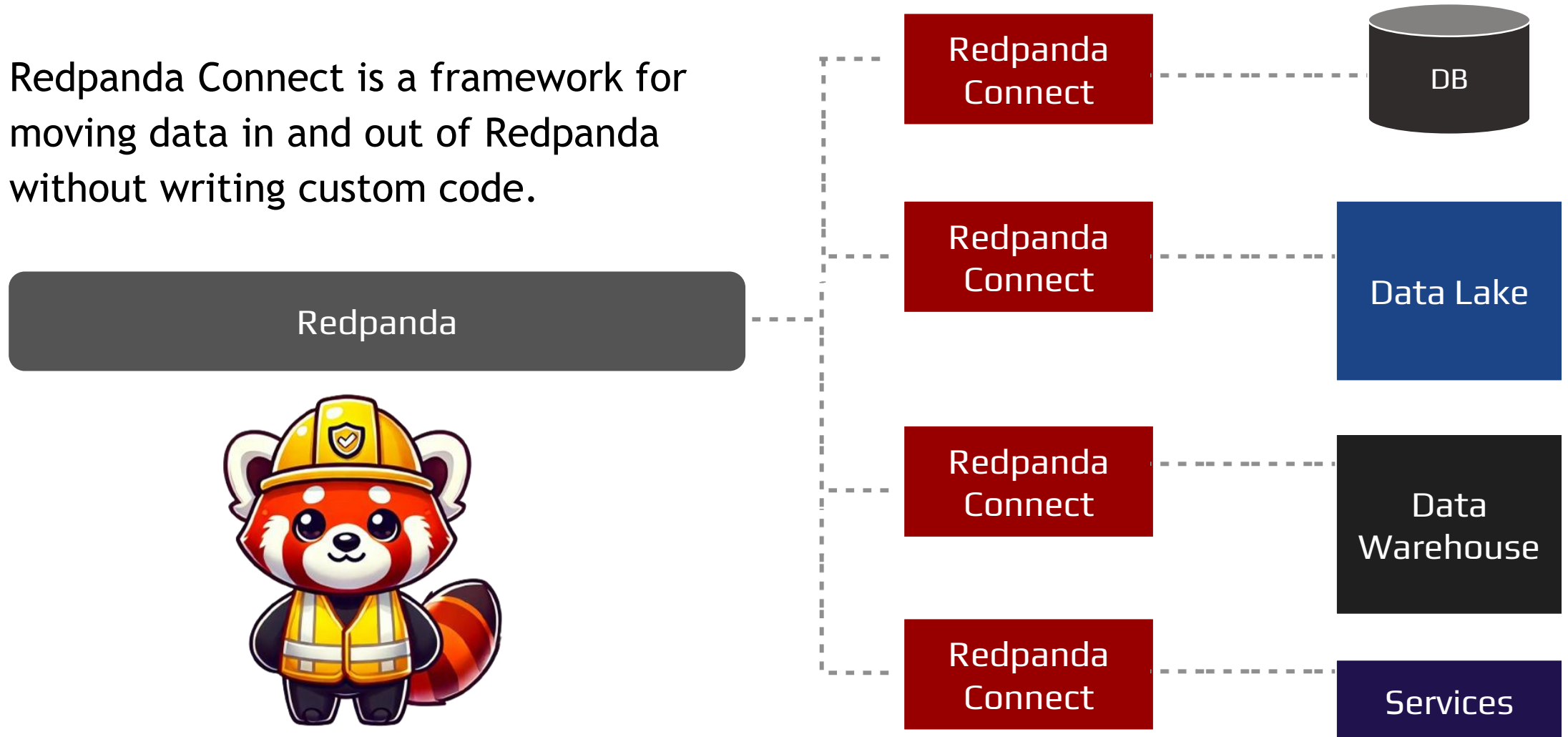
Integrate data across systems with hundreds of prebuilt connectors, change data capture (CDC) capabilities, and YAML-configurable pipelines.

Redpanda Connect = plug-and-play pipelines for databases, streams, warehouses, and cloud storage without writing custom code.

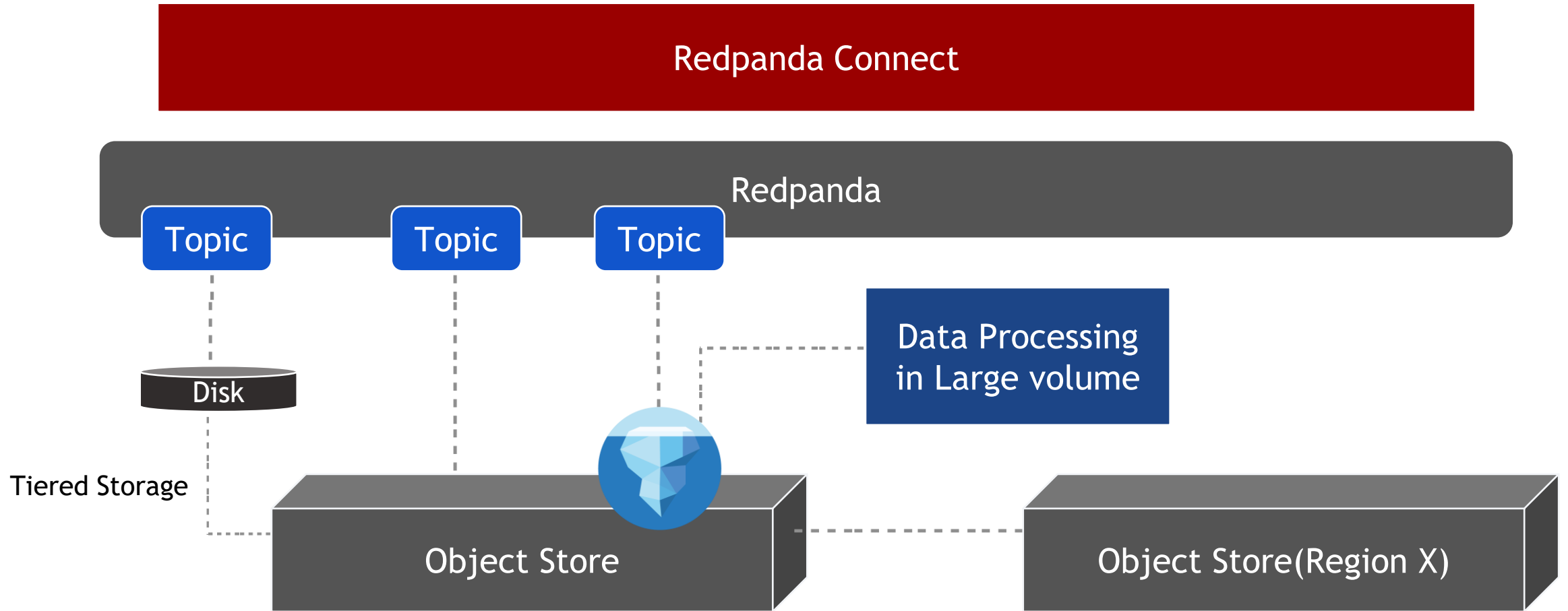


Architecture and philosophy

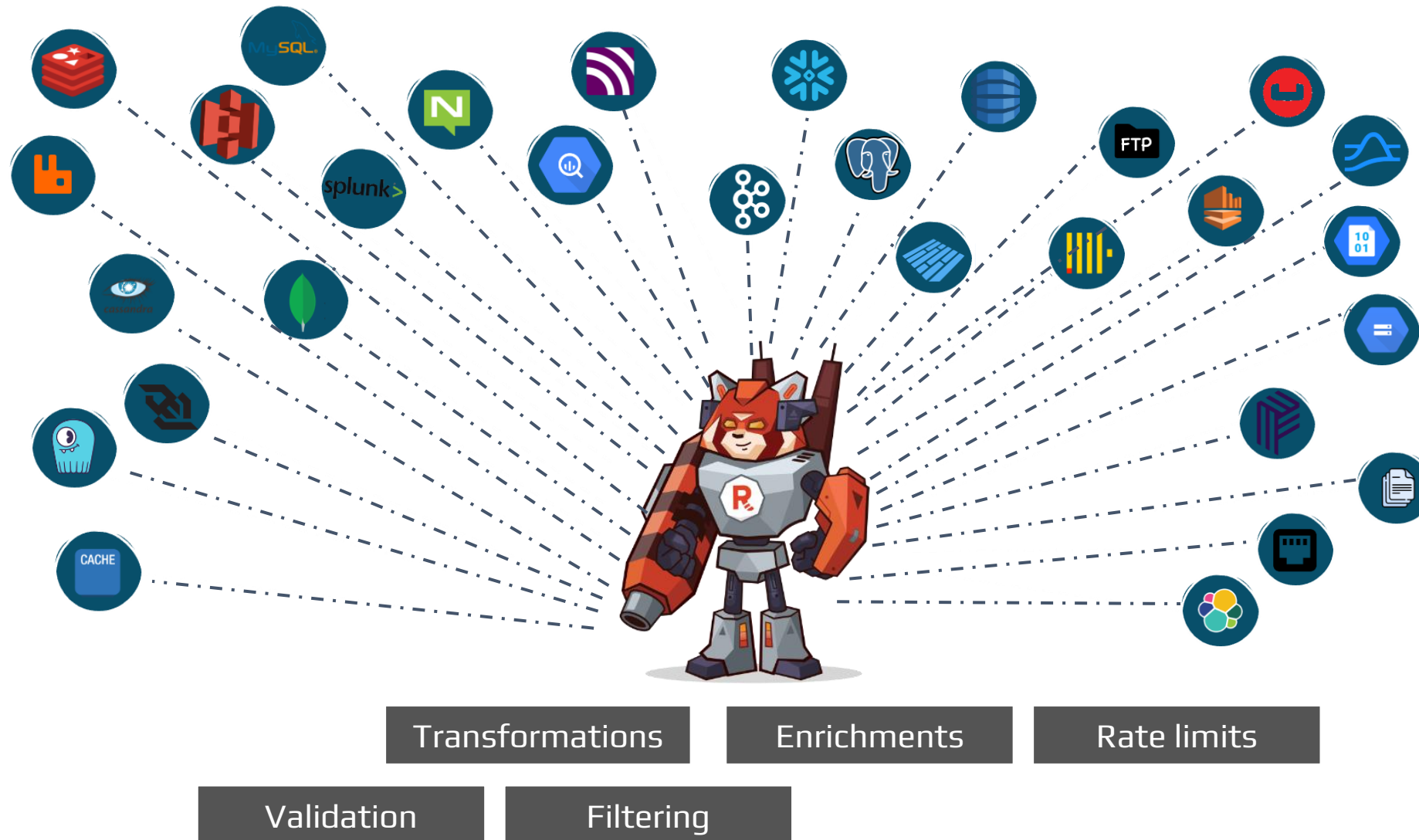
Redpanda Connect is a framework for moving data in and out of Redpanda without writing custom code.



Redpanda One



Redpanda Connect - Streaming ETL



Architecture and philosophy

Single Binary,
no more runtime dependency

Simple YAML,
no more Java classes names

Payload agnostic,
Support structured or
binary data

Redpanda Connect is
boring



Stateless and fast,
no more complex cluster setup

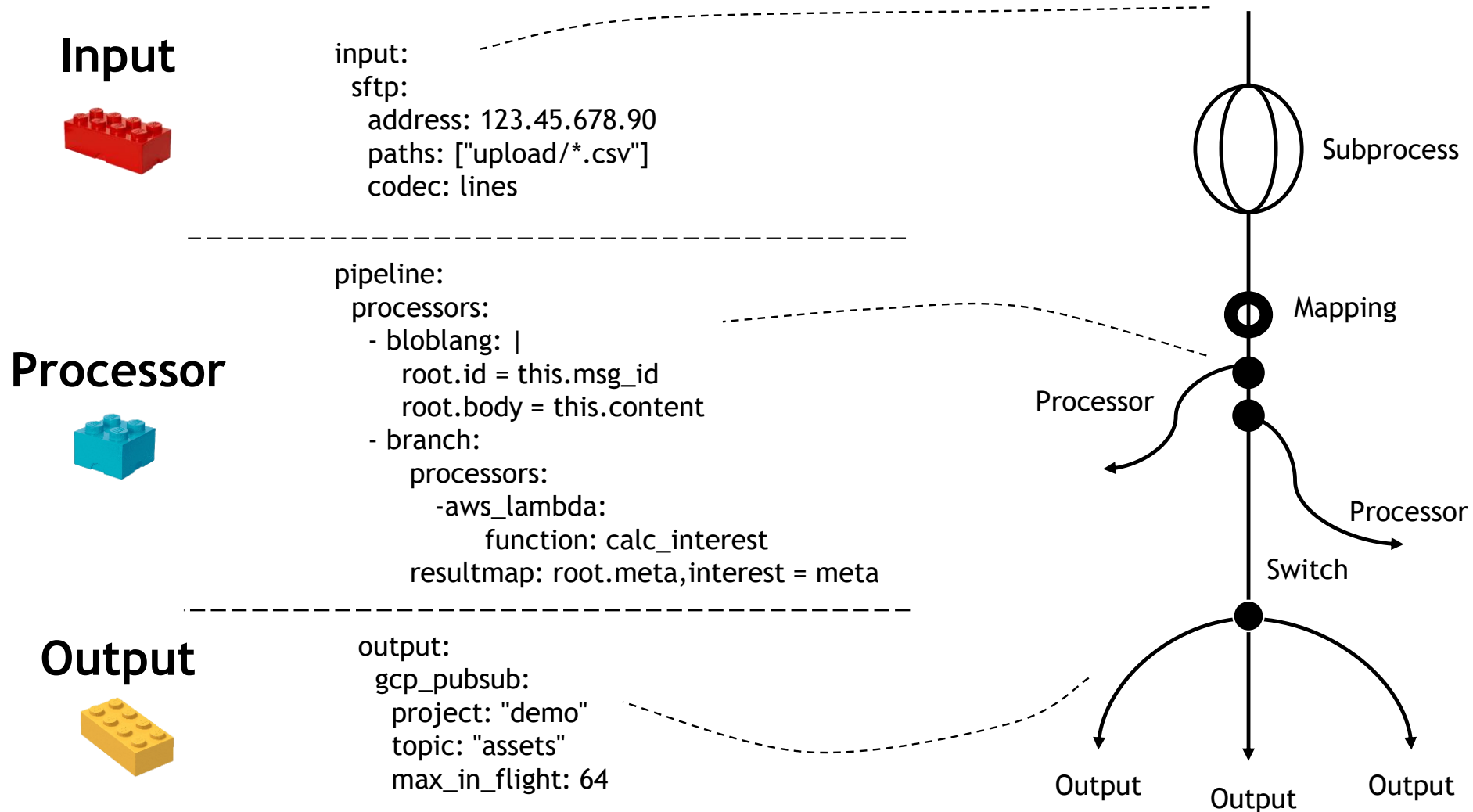
**Out of Box
Observability,**
Tracing, Metrics, and
Logs

Architecture and philosophy

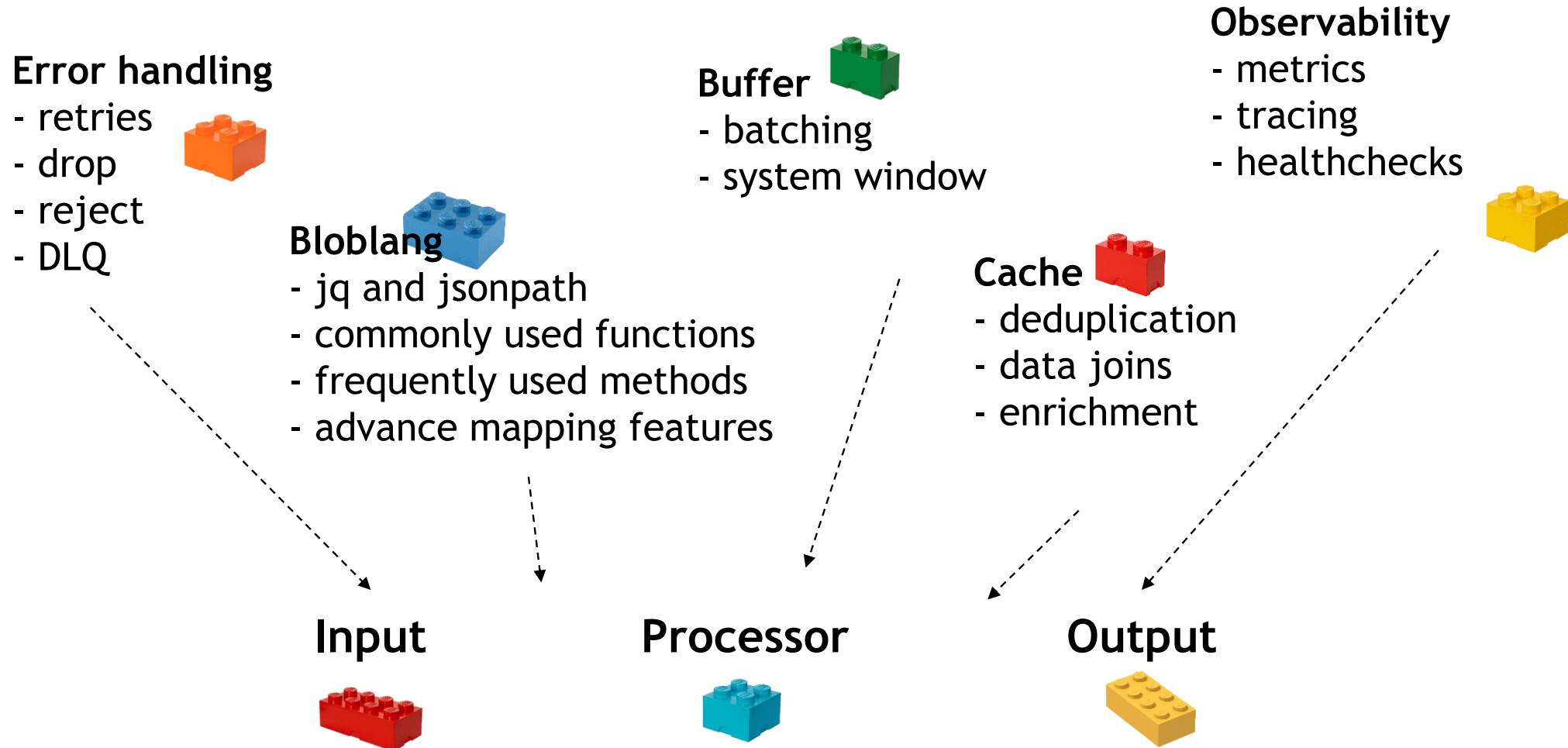


rpk connect **run** config.yaml

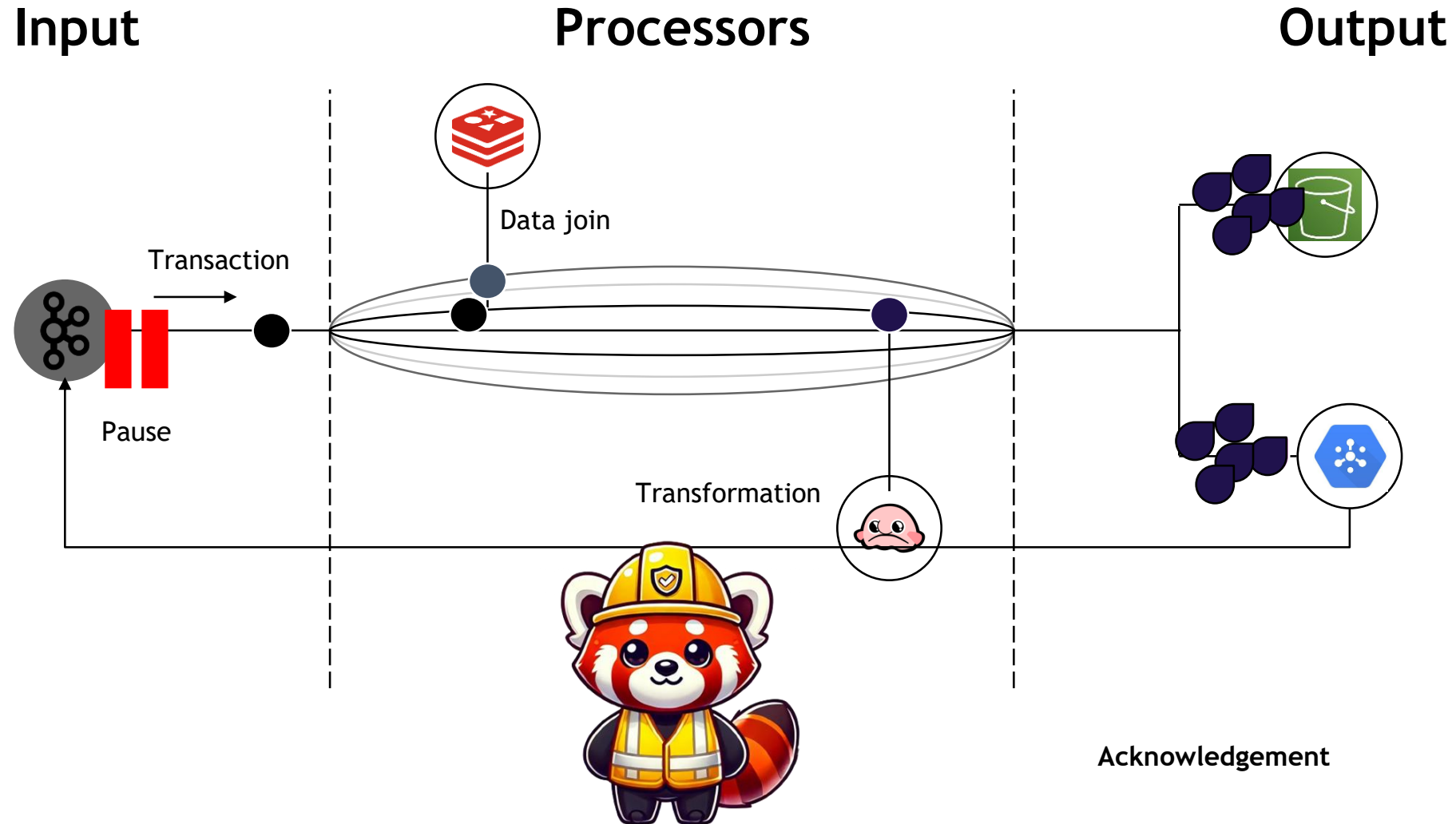
Configuring Redpanda Connect



Intuitive Declarative Configuration



Crash Resilient, Delivery Guarantee - It is Safe



Just run it!

create

lint

rpk connect run config.yaml

streams

test

YAML-based declarative pipeline configs

Redpanda Connect pipelines are configured in a YAML file that consists of a number of root sections

```
input:
  kafka:
    addresses: [ TODO ]
    topics: [ foo, bar ]
    consumer_group: foogroup

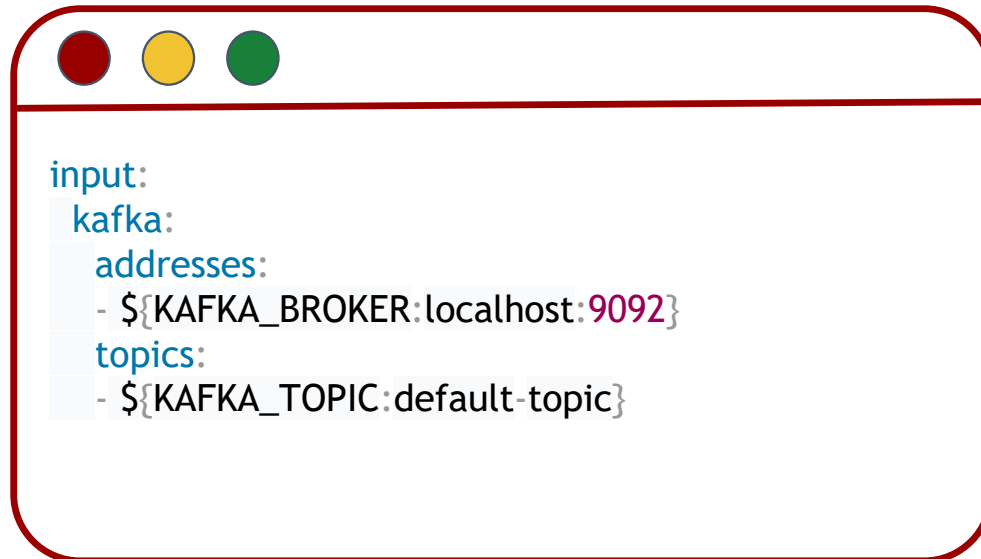
pipeline:
  processors:
    - mapping: |
        root.message = this
        root.meta.link_count = this.links.length()

output:
  aws_s3:
    bucket: TODO
    path: '${! meta("kafka_topic")} / ${! json("message.id")} .json'
```

This is powerful but can potentially lead to large and cumbersome configuration files. This document outlines tooling provided by Redpanda Connect to help with writing and managing these more complex configuration files.

Customizing your configuration


Sometimes it's useful to write a configuration where certain fields can be defined during deployment. For this purpose Redpanda Connect supports environment variable interpolation, allowing you to set fields in your config with environment variables



```
input:
  kafka:
    addresses:
      - ${KAFKA_BROKER:localhost:9092}
    topics:
      - ${KAFKA_TOPIC:default-topic}
```

Reusing configuration snippets

Sometimes it's necessary to use a rather large component multiple times. Instead of copy/pasting the configuration or using YAML anchors you can define your component as a resource.



```
pipeline:
  processors:
    - resource: get_foo
    - catch:
        mapping: |
          root = this
          root.content = this.content.strip_html()
    - resource: get_foo

  processor_resources:
    - label: get_foo
      http:
        url: http://example.com/foo
        verb: POST
      headers:
        Something: "set-to-this"
        SomethingElse: "set-to-something-else"
```

Prebuilt Connectors

Redpanda Connect comes with a wide library of ready-to-use connectors so you don't have to build everything from scratch.

These connectors cover both:

- **sources** (where data comes from)
- **sinks** (where data goes).



Prebuilt Connectors

Common Prebuilt Source Connectors

- **Databases** - PostgreSQL, MySQL, MongoDB, Oracle, SQL Server
- **Message Brokers** – Apache Kafka, RabbitMQ, NATS, MQTT
- **Cloud Storage** – AWS S3, GCP Storage, Azure Blob, MinIO
- **APIs & Streams** – HTTP, WebSockets, CDC (Change Data Capture), FileStream

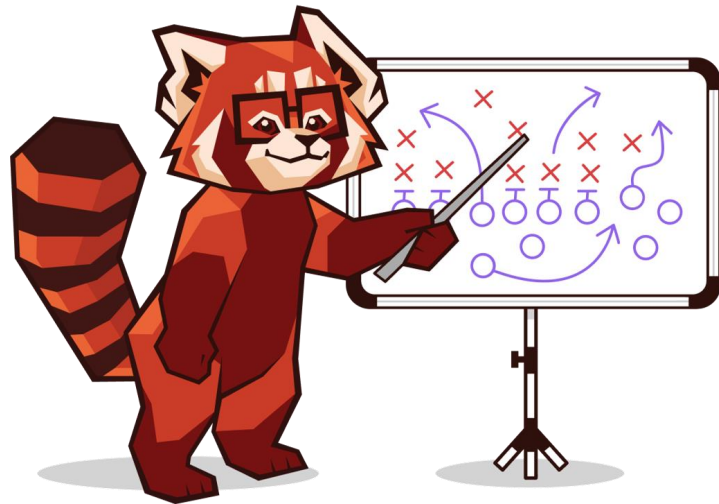
Prebuilt Connectors

Common Prebuilt Sink Connectors

- **Databases** - PostgreSQL, MySQL, ClickHouse, Snowflake, MongoDB
- **Cloud Storage** – AWS S3, GCS, Azure Blob, MinIO
- **Message Queues & Streams** – Kafka, Pulsar, RabbitMQ, NATS
- **Warehouses & Analytics** – BigQuery, Redshift, Snowflake, ElasticSearch, OpenSearch

Integration Targets

Redpanda Connect
integrates seamlessly with:



- **Data Lakes & Warehouses** → S3, BigQuery, Snowflake, Redshift
- **Databases (OLTP + OLAP)** → Postgres, MySQL, MongoDB, ClickHouse
- **Streaming Platforms** → Kafka, Pulsar, RabbitMQ, NATS, MQTT
- **Analytics & Search** → ElasticSearch, OpenSearch, Druid, Prometheus
- **Enterprise Tools** → REST APIs, gRPC endpoints, Local file systems



Data Pipelines, Connectors & Enrichment(Mixed Roles)

Bloblang for Stream Transformation



- Bloblang syntax and manipulation patterns
- Mapping, filtering, enrichment, conditional logic

What is Bloblang?

A lightweight mapping language to reshape, enrich, filter, and route messages in streaming pipelines (used in Redpanda Connect/Benthos). Think: JSON-in → rules → JSON-out, at very high throughput.

When to use it

- Clean/normalize events before sinking to DB/warehouse
- Enrich with constants/metadata/lookup results
- Mask/obfuscate PII
- Branch, drop, or reroute messages based on content

Bloblang Basics: Syntax and Structure

- root represents the new document's root.
- this refers to the input document or context.
- Assignment : root.field = this.value.
- Create objects dynamically with dot-separated paths.

```
{ "flight": "DE712",  
  "customer": {  
    "name": "Jane Doe",  
    "email": "jane@example.com",  
    "contact": {  
      "phone": "123-3456-78",  
      "text": "123-456-78" }  
  }  
}
```

root.customer.contact.name
= this.customer_name

```
pipeline:  
  processors:  
    - bloblang: |  
  
pipeline:  
  processors:  
    - mapping: |  
  
pipeline:  
  processors:  
    - branch:  
      request_map: |  
        processors: ...  
      result_map: |
```

```
{ "flight": "DE712",  
  "customer_name": "Jane Doe",  
  "customer_email": "jane@example.com",  
  "customer_phone": "123-3456-78",  
  "customer_text": "123-3456-78"  
}
```

Filtering and Sampling

Filtering events in Redpanda Connect is both easy and flexible, this cookbook demonstrates a few different types of filtering you can do. All of these examples make use of the mapping processor but shouldn't require any prior knowledge.

The basic filter

Dropping events with Bloblang is done by mapping the function `deleted()` to the `root` of the mapped document. To remove all events indiscriminately you can simply do:

```
pipeline:  
  processors:  
    - mapping: root = deleted()
```

Conditional Event Deletion in Stream Processing

We can instead only delete an event under certain conditions with a **match** or **if** expression:

```
pipeline:
  processors:
    - mapping: |
        root = if @topic.or("") == "foo" ||
          this.doc.type == "bar" ||

        this.doc.urls.contains("https://www.benthos.dev/").catch(false) {
          deleted()
        }
```

- The metadata field `topic` is equal to `foo`
- The event field `doc.type` (a string) is equal to `bar`
- The event field `doc.urls` (an array) contains the string <https://www.benthos.dev/>

Enrichment

Redpanda Connect enables real-time data enrichment by adding context, transforming formats, and integrating external sources like caches or AI systems.

How Enrichment Works in Redpanda Connect

- **Workflow Processors:** Perform multiple enrichments in parallel using dependency-aware processors.
- **Declarative Configuration:** Define pipelines and enrichment logic via simple YAML.
- **Integration:** Connect to databases, APIs, files, sinks, or AI models for transformations.
- **Reusable Code:** Apply JavaScript, Wasm, AWS Lambda, or Linux commands per message.
- **Benthos Integration:** Enables advanced mapping, filtering, and enrichment for complex analytics and AI use cases.

Typical Enrichment Use Cases

1

Data Normalization: Standardize messages for analysis and system compatibility.

2

User Activity Analysis: Enrich events for personalization, analytics, and fraud detection.

3

Joining Streams: Combine streams and add context from databases or caches.

4

Real-time AI & Analytics: Enhance data with AI model or API insights.

5

Enrichment Benefits: Enables scalable, advanced pipelines for real-time analytics across industries.

Conditional logic

Redpanda Connect uses processors, Bloblang mapping, and branching to enable dynamic, conditional pipeline processing.

How Conditional Logic Works in Redpanda Connect

- **Workflow Branches:** Execute pipeline stages based on specific criteria.
- **Bloblang Mapping:** Use `if` statements and `match` expressions for conditional mapping, including nested structures.
- **Filtering & Dropping:** Use `deleted()` to skip messages that don't meet conditions.
- **Conditional Routing:** Route outputs dynamically based on message content.
- **Practical Use:** Enables real-time filtering and selective enrichment, e.g., processing only relevant user activity or adding missing context.

Conditional Features

Feature	Example Usage	How to Specify
Branching Workflows	Execute stage if field matches condition	Define branch condition
Bloblang if or match Mapping	Map field only if expression is true	Inline YAML or mapping
Conditional Dropping/Filtering	Drop messages not meeting conditions	Set target to deleted()
Conditional Output Multiplexing	Route based on field value	Interpolate output config



Data Pipelines, Connectors & Enrichment(Mixed Roles)

Building Resilient Pipelines



- Retry and catch processors, error handling
- Dead-Letter Queues, fallback outputs, poison pill isolation
- Throughput tuning and scaling
- Expanded Connector Deep Dive
- Detailed modules on Snowflake, MongoDB, PostgreSQL connectors
- Practical examples

Retry and catch processors

Redpanda Connect offers robust error handling and recovery through its retry and catch processors, making stream processing pipelines more resilient and reliable.

Retry Processor: Retries child processors until success or limits are reached.

- **max_retries:** Max attempts (0 = unlimited).
- **backoff.initial_interval:** Wait between retries (default 500ms).
- **backoff.max_interval:** Max backoff interval (default 3s).
- **backoff.max_elapsed_time:** Total retry duration (0 = unlimited).

Retry and catch processors

Catch Processor

- **Functionality:** The catch processor handles failed messages by allowing additional processors to act on the error, enabling recovery, logging, or rerouting.

Usage Example:



```
pipeline:
  processors:
    - resource: foo # Processor that might fail
    - catch:
      - resource: bar # Recovery or mitigation logic
```

- After passing through the catch block, messages have their failure flags removed and are treated as regular messages. If this is not desired, switch logic can simulate catch with more control.

Error handling

- **Error Flagging:** Failed messages get metadata flags for recovery or routing.
- **Catch Processor:** Intercepts failures, runs recovery, clears flags.
- **Switch Processor:** Routes errored messages by failure details.
- **Logging:** Use `error()` to log or augment payloads with error info.
- **Dropping:** Map errored messages to `deleted()` for safe ack.
- **Rejecting:** Send failures to `reject_errored` outputs (nacks).
- **Dead-Letter Queues:** Redirect failed messages for storage/inspection.
- **Conditional Outputs:** Route based on specific error content.

Admin Tasks: GUI Topic Setup & Editing

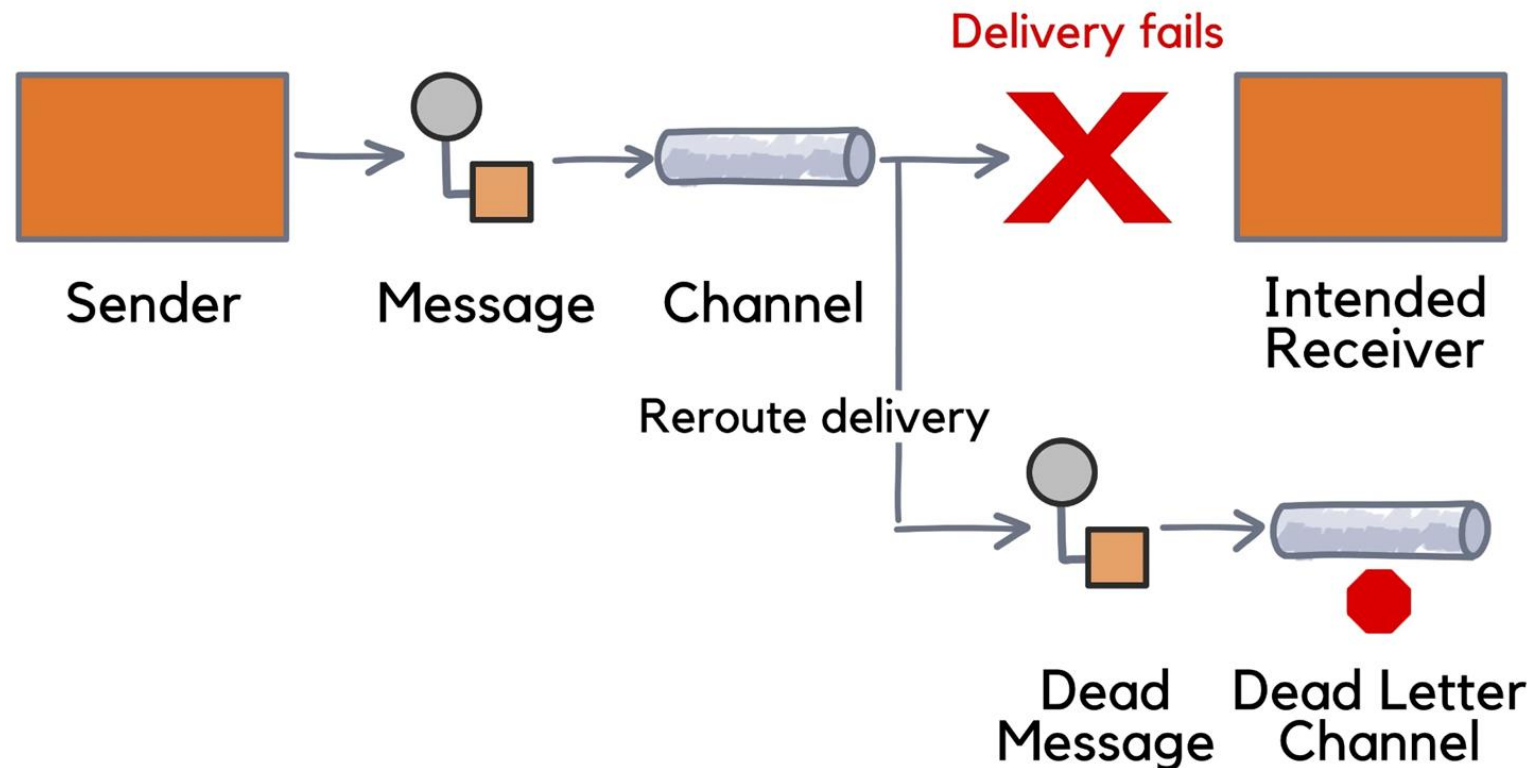
Redpanda Console provides an intuitive web-based interface that makes it easy for administrators to manage topics without relying on CLI commands. Through the GUI, you can:

- **Create Topics** - Set up new topics with partitions, replication, and advanced configurations.
- **Edit Topics** - Update topic settings, scale partitions, or adjust replication as needed.
- **Delete Topics** - Remove unused topics to keep the cluster clean and efficient.
- **Provides Visibility** - View real-time topic configurations at a glance.
- **Reduces Errors** - Minimize risks compared to manual CLI commands.
- **Simplifies Management** - Manage topics easily through a no-code GUI.

Dead-letter queue

Dead-letter queue (DLQ) holds the messages which couldn't be processed by the consumer or routed to their destination.

The DLQ follows the Dead Letter Channel design pattern described in the famous Enterprise Integration Patterns book.



Dead-Letter Queues (DLQ)

1

Capture messages that fail after retries.

2

Any Redpanda topic can serve as a DLQ.

3

Prevents pipeline blocking or message loss.

4

Isolates problematic messages for analysis or reprocessing.

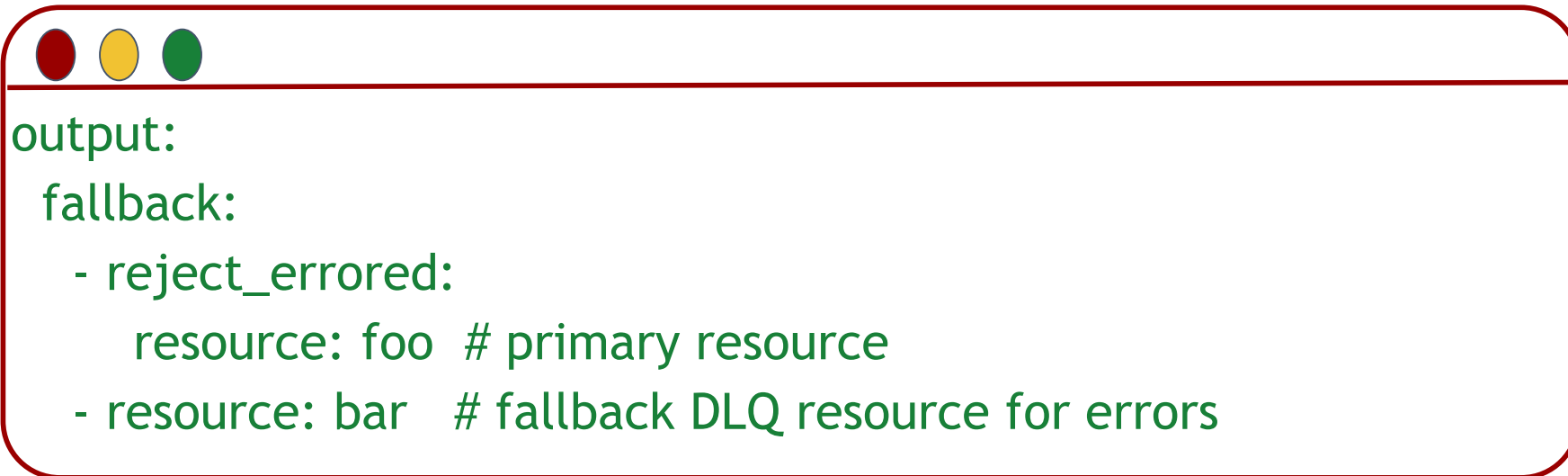
5

Messages can be fixed, alerted on, or dropped after review.

Fallback Outputs

Route failed messages to secondary sinks or DLQs when the primary output fails, isolating bad records and keeping the main pipeline uninterrupted.

Example fallback configuration:



```
output:  
  fallback:  
    - reject_errorred:  
      resource: foo # primary resource  
    - resource: bar # fallback DLQ resource for errors
```

Poison Pill Isolation

Core Metrics

- **Identify:** Detect poison pills using error flags when messages repeatedly fail.
- **Retry:** Apply configurable retry logic before isolation.
- **Isolate:** Route persistent failures to DLQs or fallback outputs.
- **Protect:** Prevent pipeline blockage and shield downstream systems.
- **Manage:** Analyze, reprocess, or drop problematic messages as needed.



Dead-Letter Queues, fallback outputs, poison pill isolation

Concept	Purpose	Redpanda Connect Feature
Dead-Letter Queue	Isolate failed messages for reprocessing or analysis	Use any topic as DLQ with manual routing
Fallback Output	Route errored messages away from main output	fallback output with failover message sinks
Poison Pill Isolation	Prevent problematic messages from blocking	Error flags + routing to DLQ/fallback

Throughput tuning and scaling

Throughput Throttling:

- Redpanda supports throughput throttling independently on ingress (data coming in) and egress (data going out).
- Throttling can be configured at both broker and client levels to prevent unbounded network and disk usage by clients.
- Broker-wide limits restrict total traffic on the broker, while client limits prevent some clients from starving others.
- When throughput limits are breached, Redpanda applies back pressure to clients by adding throttle delays in responses or read operations.
- You can configure throughput quotas for individual clients or groups of clients using properties like `target_quota_byte_rate`, `target_fetch_quota_byte_rate`, and group quota properties.

Batch Tuning for Throughput

- Batching groups messages into larger units sent as a single request, improving efficiency by reducing the overhead of multiple small requests.
- Larger batches reduce CPU utilization and increase compression efficiency but introduce slight additional latency since messages are buffered before sending.
- Proper tuning of batch size and linger time can improve throughput and reduce broker CPU saturation and request backlog.
- There are trade-offs between latency and throughput based on batch tuning settings.

Scaling

Architecture:

Resilient, high-performance YAML pipelines with connectors and processors.

Connectors:

220+ adapters with CDC for real-time, at-least-once DB sync.

Customization:

Transform messages via WASM, JavaScript, Linux, or AWS Lambda.

Deployment:

Efficient YAML pipelines, cloud/on-prem/hybrid, with observability and Kubernetes support.

Research & AI:

Streams data to warehouses, notebooks, ML endpoints, and vector DBs for real-time analytics.

By combining throughput throttling to manage resource usage, tuning batch sizes to maximize efficiency, and scaling clusters vertically and horizontally, Redpanda achieves a high throughput performance and scalability for streaming workloads.

Expanded Connector Deep Dive

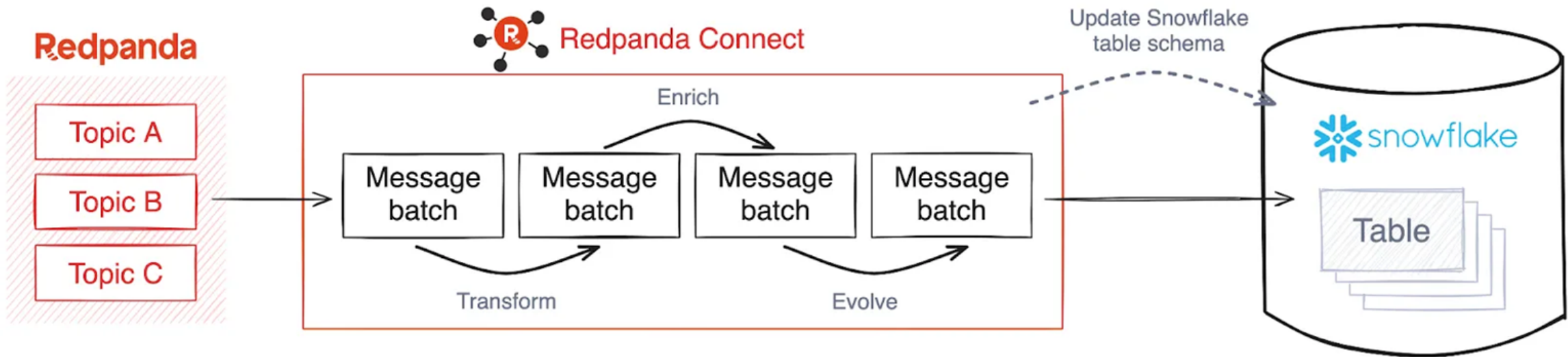
Redpanda Connect deep dive focuses on declarative, high-performance connectors for seamless streaming data integration across diverse systems.

Redpanda Connect - Technical Dive

- **Architecture:** High-performance, resilient pipelines with YAML-defined connectors and processors for mutation, enrichment, or filtering.
- **Connectors:** 220+ adaptable connectors; CDC for real-time DB sync with at-least-once delivery.
- **Customization:** Per-message transformations via WASM, JavaScript, Linux commands, or AWS Lambda.
- **Deployment & Operations:** Resource-efficient YAML pipelines, cloud/on-prem/hybrid, with observability and Kubernetes scalability.
- **Research & AI:** Integrates streaming data with warehouses, notebooks, ML endpoints, and vector DBs for real-time analytics.

Snowflake

Snowflake provides fast, flexible analytics with seamless Apache Kafka® integration.



Snowflake

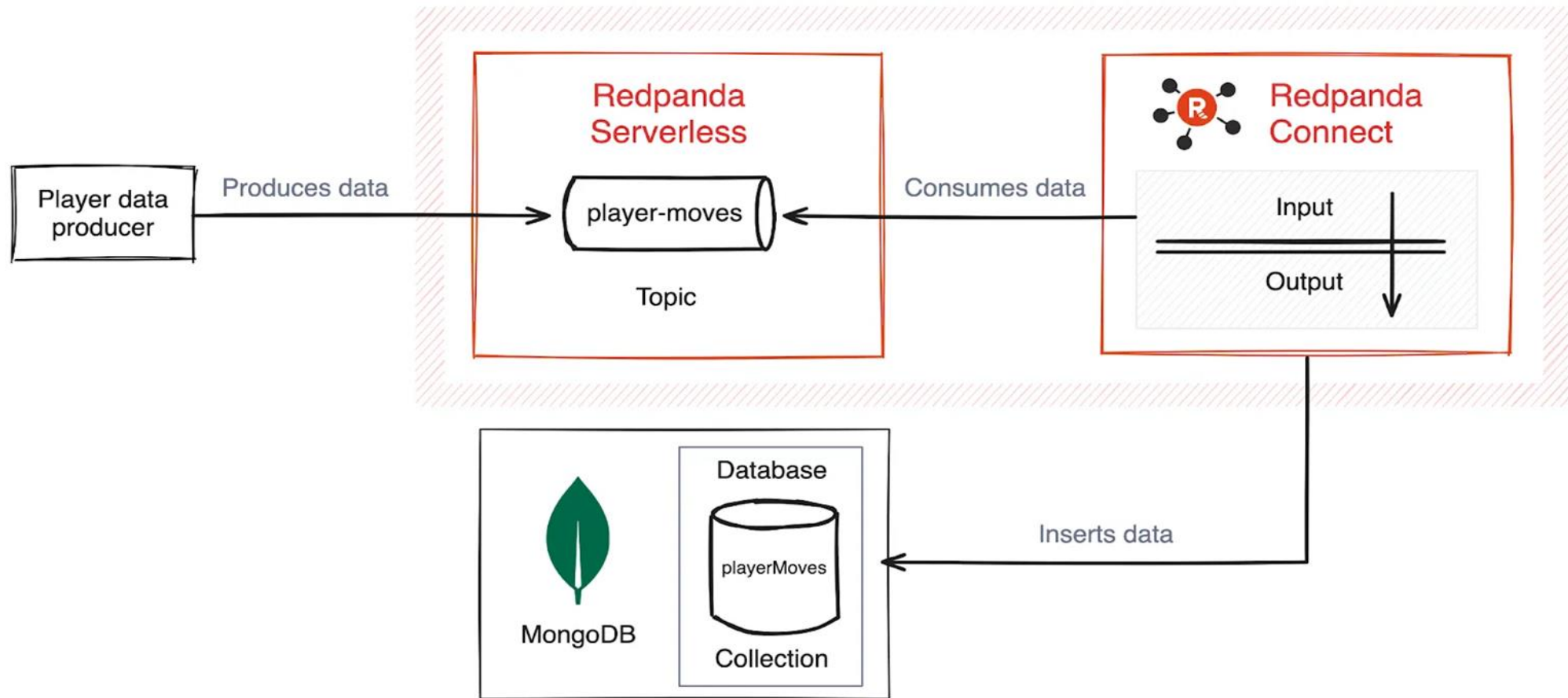
Redpanda Connect provides a powerful and simple way to stream data directly into Snowflake using Snowpipe Streaming. This integration is designed to unify streaming and batch processing with near real-time data ingestion, offering low latency and high throughput compared to traditional Kafka Connect.

Key Features

- **Fast ingestion:** Up to 2X faster writes compared to Kafka Connect.
- **No-code configuration:** Easily set up with just a few lines of YAML configuration.
- **Schema evolution:** Automatic table creation and schema updates when new columns appear.
- **Secure connection:** Supports encrypted private keys and SASL authentication.
- **Flexible:** Supports splitting streams into multiple tables and custom transformations on data in-flight.

MongoDB

Seamless real-time data streaming for use cases like AdTech, SaaS, AI, and game analytics.



MongoDB

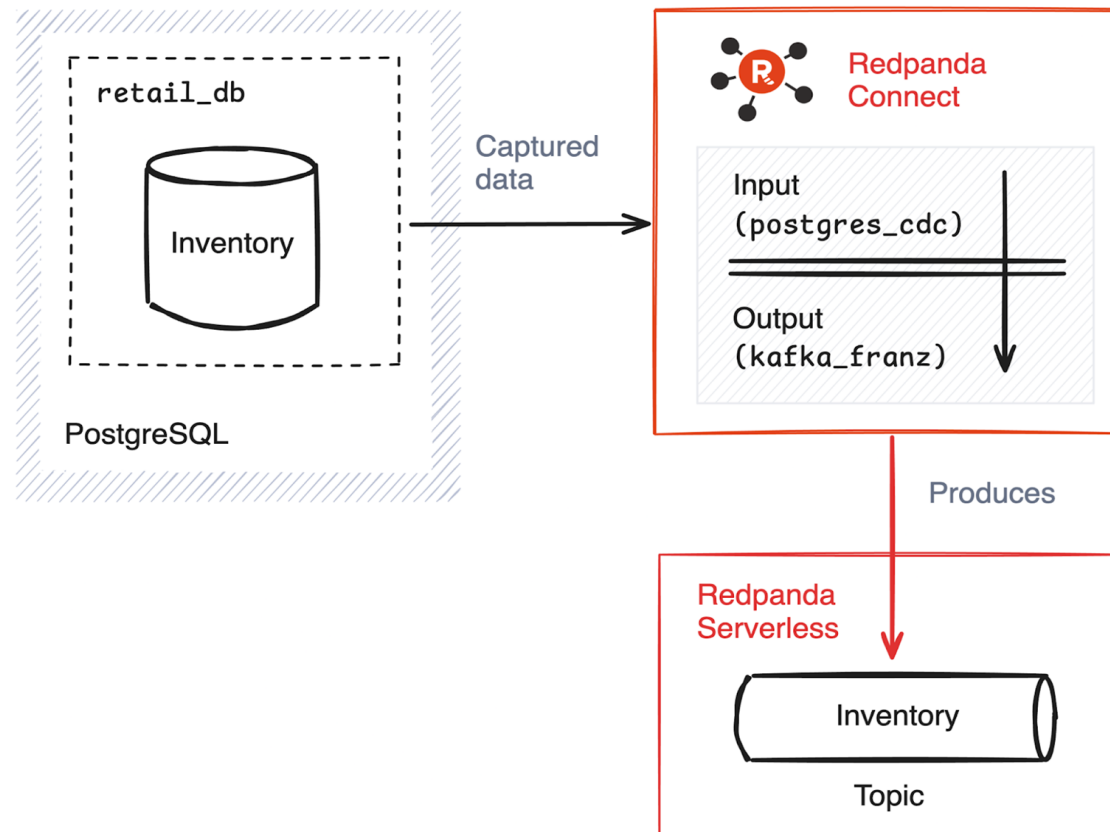
Redpanda offers fully managed MongoDB connectors for both source and sink integrations that simplify real-time data streaming between MongoDB and Redpanda Cloud without needing custom Kafka Connect setups.

Key Points of MongoDB Integration with Redpanda

- **Source Connector:** Streams real-time changes from MongoDB oplog into Redpanda topics.
- **Sink Connector:** Writes events from Redpanda topics into MongoDB collections.
- **Management:** Fully supported in Redpanda Cloud, configurable via Console UI or CLI.
- **CDC Fidelity:** Uses oplog for accurate Change Data Capture.
- **Customization:** Control tasks, filtering, mappings, and write operations.

PostgreSQL connectors

captures real-time changes using logical replication and streams them into Redpanda topics for event-driven data processing.



PostgreSQL connectors

Redpanda offers fully managed PostgreSQL connectors for both source and sink integrations that simplify real-time data streaming between PostgreSQL and Redpanda Cloud without needing custom Kafka Connect setups.

Key Points of PostgreSQL Integration with Redpanda

- **Source Connector:** Streams real-time changes from PostgreSQL logical replication into Redpanda topics.
- **Sink Connector:** Writes events from Redpanda topics into PostgreSQL tables.
- **Management:** Fully supported in Redpanda Cloud, configurable via Console UI or CLI.
- **CDC Fidelity:** Uses PostgreSQL logical replication slots for accurate Change Data Capture.
- **Customization:** Control tasks, filtering, mappings, and write operations.

Practical examples

- **Scalable Streaming:** Low-latency, high-throughput event streaming using Kafka APIs with topics and partitions.
- **Data Flows:** Research, sensor, and operational data streamed for real-time analytics or stored in data lakes.
- **SQL Integration:** With Upsolver SQLake, stream JSON into lake tables and update materialized views via SQL.
- **Simplified Ops:** Eliminates complex scheduling/orchestration while scaling with data volumes.
- **Data Processing:** Redpanda Connect supports filtering, mapping, enrichment, and external lookups.
- **Ecosystem Integration:** Works with databases, queues, and cloud storage systems.



Data Pipelines, Connectors & Enrichment (Mixed Roles)

Redpanda One & Zero-Copy Search to Snowflake



- Managed connectors as queryable DBs
- Performance and schema considerations

Managed connectors as queryable DBs

Fully-managed, pre-built connectors in Redpanda Cloud that simplify integration between Redpanda streams and external databases or systems without manual Kafka Connect setup.



- Connect to MongoDB, MySQL, PostgreSQL, SQL Server, SQLite, Snowflake, BigQuery, and more.
- Stream data in real time with automatic schema management.
- Import database changes with source connectors and export streams with sink connectors.
- Configure pipelines via Redpanda Console UI or YAML.

Performance and schema considerations

Performance

- Redpanda achieves lower latency and higher throughput than Kafka.
- Up to 70× faster tail latencies with fewer nodes for 1GB/sec workloads.
- Thread-per-core C++ design enables high throughput on cost-effective hardware.
- Cloud connectors offer auto-scaling, fault tolerance, and backpressure handling.
- Tune connectors and streams for optimal performance.

Performance and schema considerations

Schema Management

- Managed connectors auto-discover and evolve schemas, reducing manual effort.
- Schema registries ensure consistent schemas and validate events.
- Connectors support schema transformations and filtering for clean data.
- Stream schemas should optimize indexing, partitioning, and query performance.
- Supports Avro, Protobuf, and JSON for flexible serialization.

Performance and schema considerations

Best Practices

- Monitor connector metrics and tune configurations to avoid bottlenecks.
- Design stream schemas for downstream queries to minimize migrations.
- Keep Redpanda and connectors updated for performance and schema improvements.
- Use Zero-Copy Search with proper schemas for fast, duplicate-free streaming queries.

A low-angle, upward-looking photograph of several modern skyscrapers with glass facades. The image is overlaid with a semi-transparent dark grey rectangle in the center, which contains the text "THANK YOU". Additionally, there are three solid red rectangular blocks: one in the upper center, one in the lower left, and one in the lower right. A small black horizontal bar is positioned directly below the red block in the lower left.

THANK YOU