Kafka Connect

Future of Data: Princeton Meetup

Kaufman Ng

kaufman@confluent.io

Agenda

- About Me
- About Confluent
- Apache Kafka 101
- Kafka Connect
  - Logical Architecture
  - Core Components and Execution Model
  - Connector Configuration, Deployment and Management
- Connector Examples
  - Out-of-the-Box
  - Community
- Wrap Up and Questions

About Me

- Solutions Architect, Confluent
- Senior Solutions Architect, Cloudera
- Bloomberg
- Contributor to Kafka, Parquet
- kaufman@confluent.io
- Twitter: @kaufmanng

About Confluent

About Confluent and Apache Kafka

- Founded by the creators of **Apache Kafka**
- Founded **September 2014**
- Technology developed while **at LinkedIn**
- **75%** of active Kafka committers

BENCHMARK Data Collective Index Ventures LinkedIn

Leadership

Jay Kreps
CEO

Todd Barnett
VP WW Sales

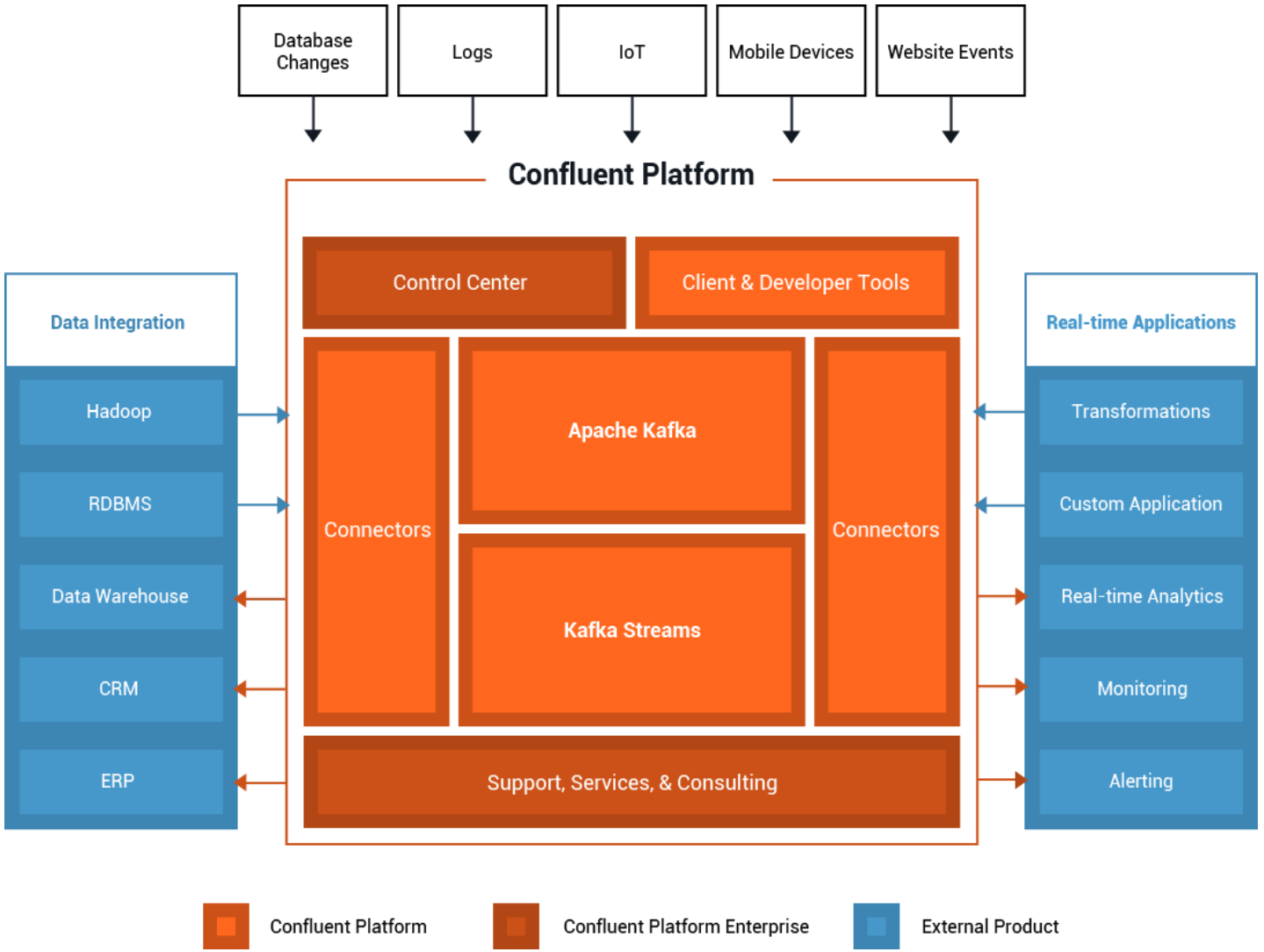
Neha Narkhede
CTO, VP Engineering

Cheryl Dalrymple
CFO

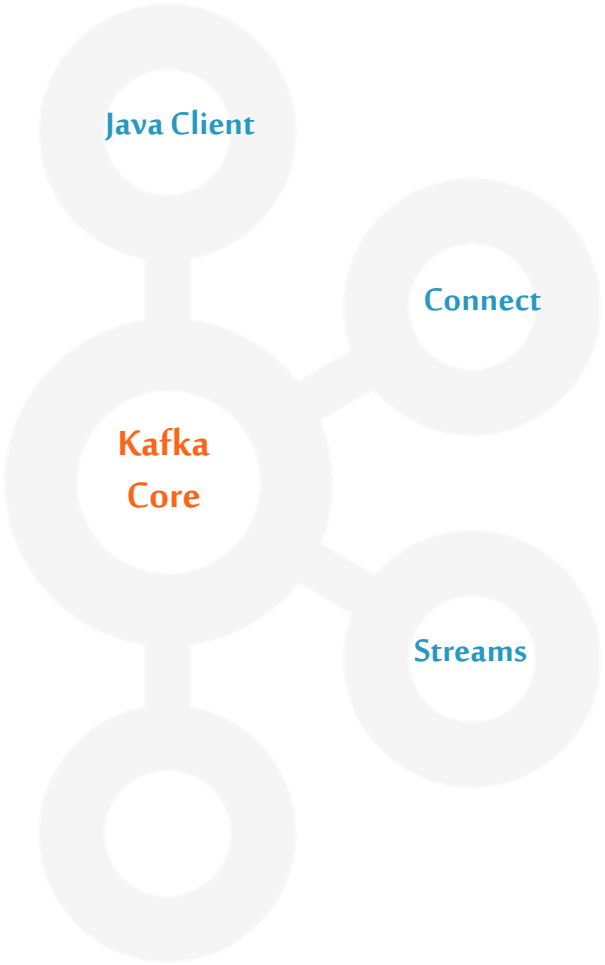
Jabari Norton
VP Business Dev

Luanne Dauber
CMO

What is the Confluent Platform?



Confluent's Offerings



Confluent Platform	Confluent Enterprise
Pre-Built Connectors	Connector Management
Schema Registry	Stream Audit
REST Proxy	Message Delivery
Additional Clients (C/C++, Python)	Stream Monitoring

Confluent Platform: Kafka ++

Feature	Benefit	Apache Kafka	Confluent Platform 3.0	Confluent Enterprise 3.0
Apache Kafka	High throughput, low latency, high availability, secure distributed message system	●	●	●
Kafka Connect	Advanced framework for connecting external sources/destinations into Kafka	●	●	●
Java Client	Provides easy integration into Java applications	●	●	●
Kafka Streams	Simple library that enables streaming application development within the Kafka framework	●	●	●
Additional Clients	Supports non-Java clients; C, C++, Python, etc.		●	●
Rest Proxy	Provides universal access to Kafka from any network connected device via HTTP		●	●
Schema Registry	Central registry for the format of Kafka data – guarantees all data is always consumable		●	●
Pre-Built Connectors	HDFS, JDBC and other connectors fully Certified and fully supported by Confluent		●	●
Confluent Control Center	Includes Connector Management and Stream Monitoring			●
Support	Connection and Monitoring command center provides advanced functionality and control	Community	Community	24x7x365
		Free	Free	Subscription

What's New in Confluent Platform 3.0

Kafka Connect (version 2.0)

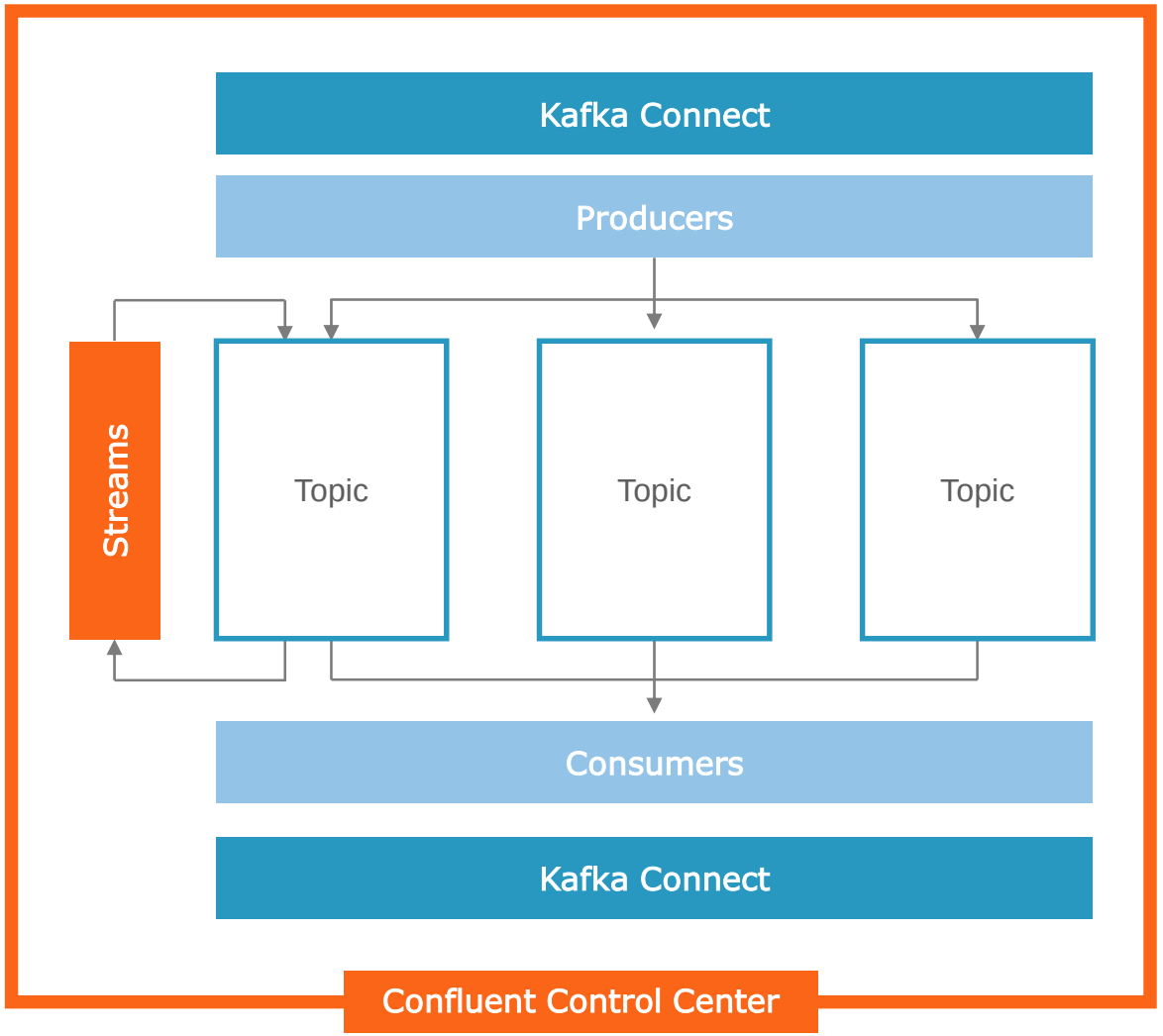
- Expanding community of connectors and architectures
- Improvements to Schema Registry and REST proxy

Kafka Streams

- A library for writing stream data applications using Kafka
- Powerful and lightweight

Confluent's First Commercial Product

- Confluent Control Center
- A comprehensive management system for Kafka clusters
- Stream monitoring capability that monitors data pipelines from end-to-end



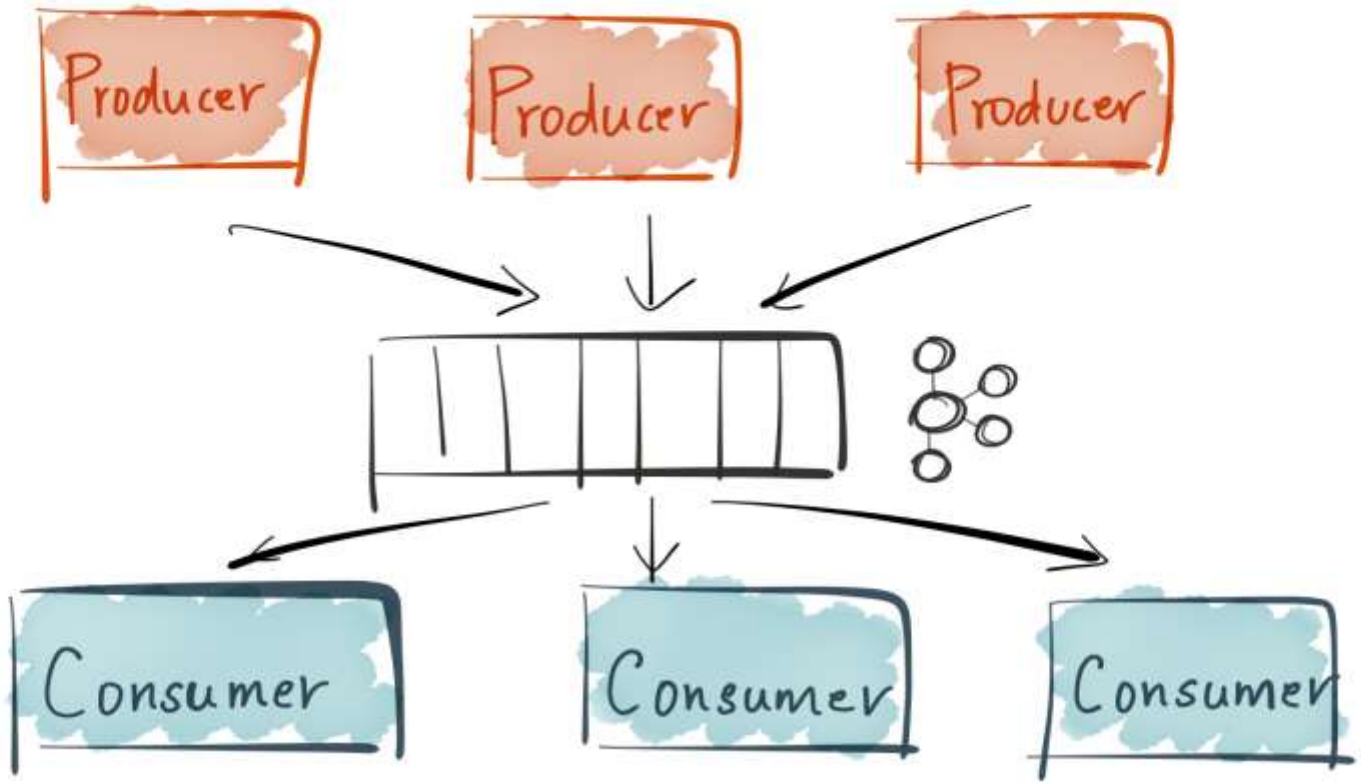
Latest Version: Confluent Platform 3.0.1 (released September 2016)

- Based on latest version of Apache Kafka 0.10.0.1
- Bug fixes + improvements (56 issues)
- Highlights
  - Kafka Streams: new Application Reset Tool
  - Security fixes
  - And many others

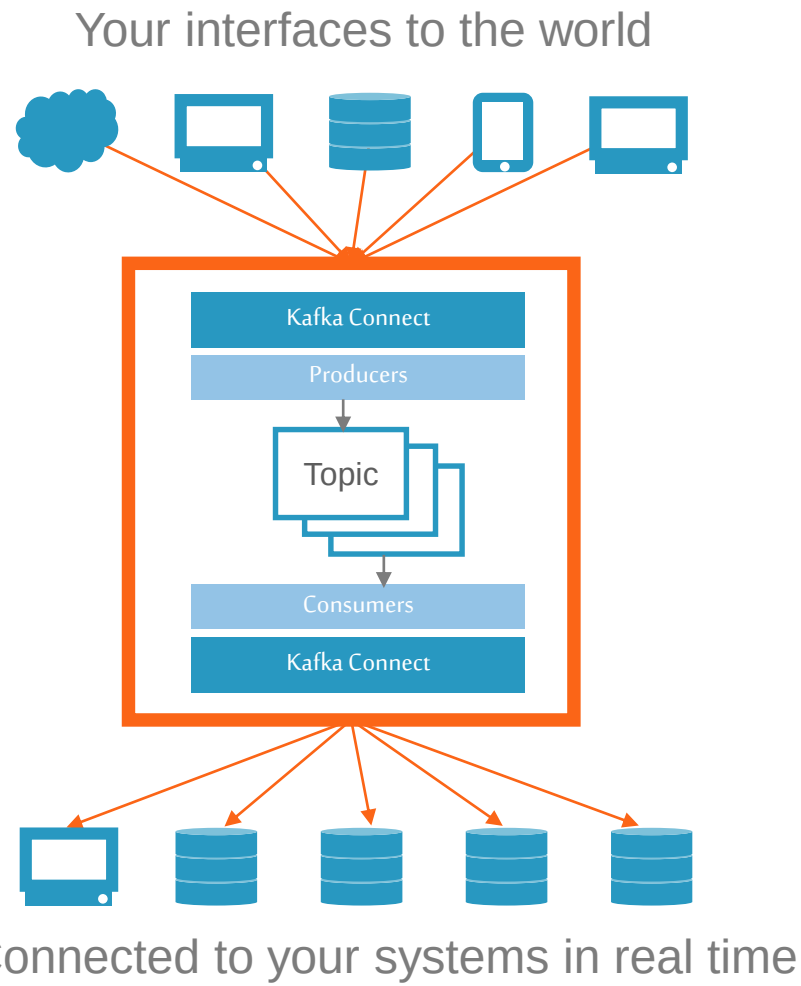
Introduction

To Real Time Streaming with Apache Kafka

Apache Kafka 10,000 Ft View

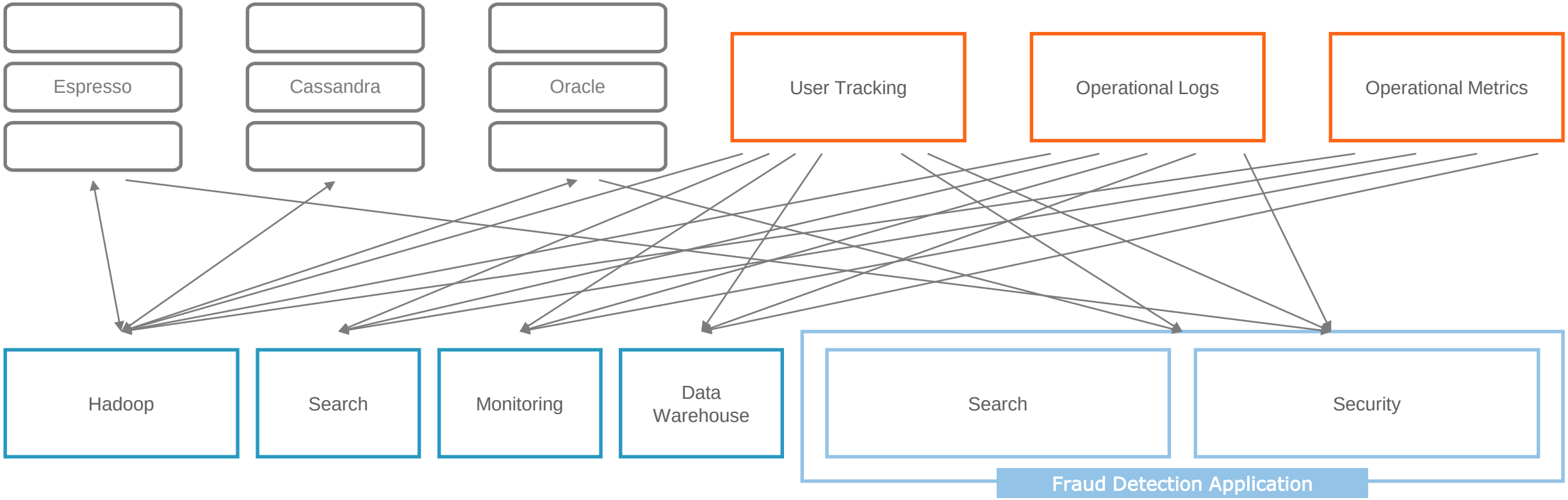


What does Kafka do?

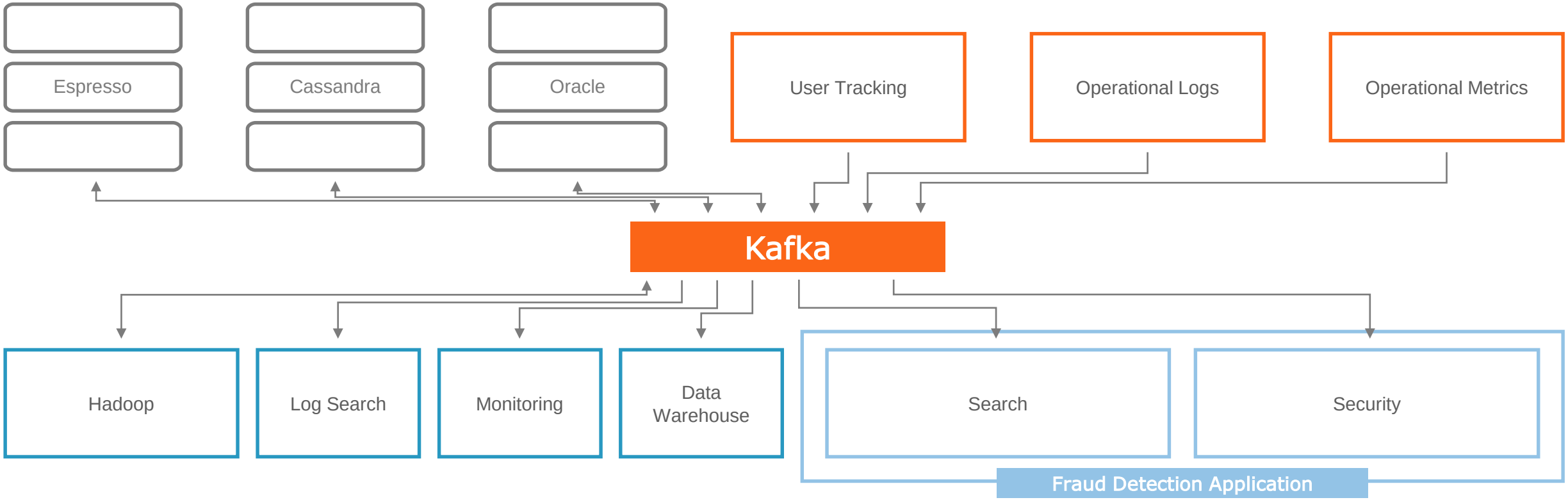


pub-sub

Before: Many Ad Hoc Pipelines



After: Central Hub with Kafka



Kafka Connect

Simplified, scalable data import/export for Kafka

What is Kafka Connect?

Kafka Connect is a distributed, scalable, fault-tolerant service designed to reliably stream data between Kafka and other data systems.

Data is produced from a source and consumed to a sink.

Example use cases:

- Publishing an SQL Tables (or an entire SQL database) into Kafka
- Consuming Kafka topics into HDFS for batch processing
- Consuming Kafka topics into Elasticsearch for secondary indexing
- Integrating legacy systems with modern Kafka framework
- … many others

Why Kafka Connect?

Kafka producers and consumers are simple in theory, yet ungainly in practice

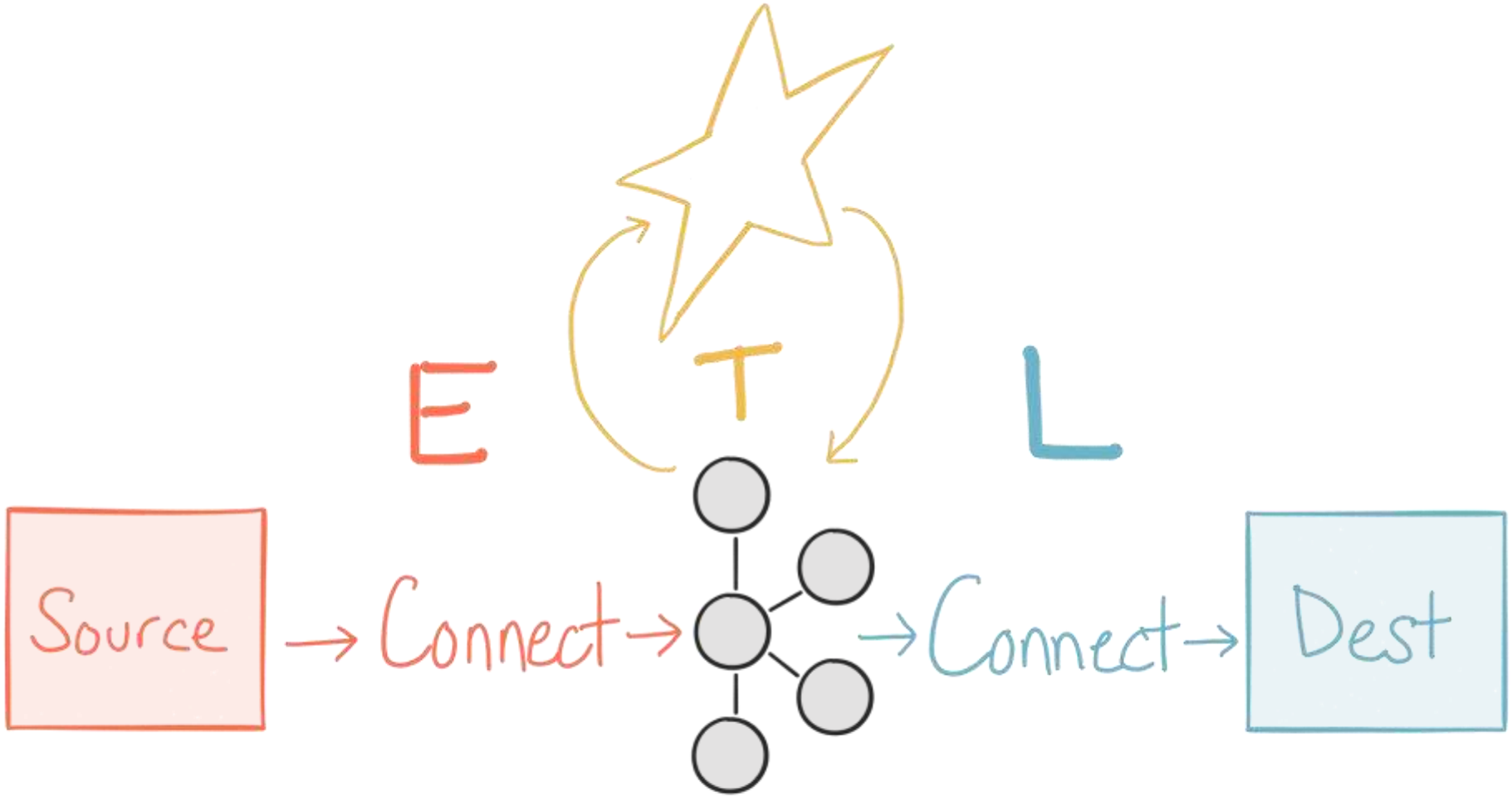
Many common requirements

- Data conversion (serialization)
- Parallelism / scaling
- Load balancing
- Fault tolerance / fail-over
- General management

A few data-specific requirements

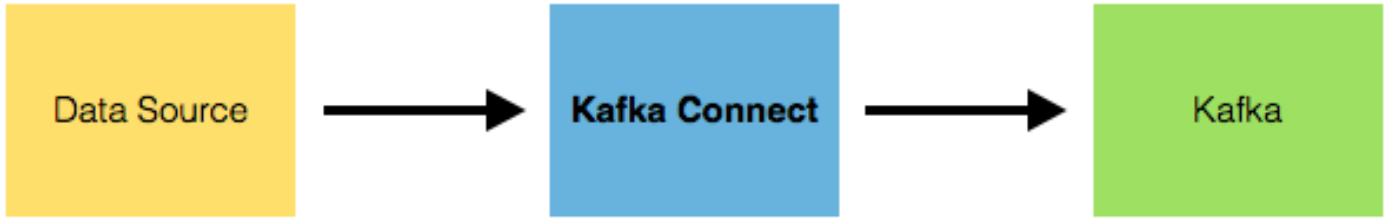
- Move data to/from sink/source
- Support relevant delivery semantics

Kafka Connect : Separation of Concerns

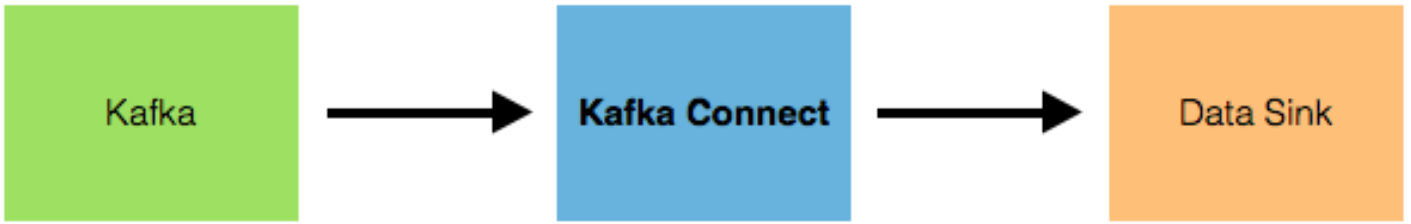


High-level Model

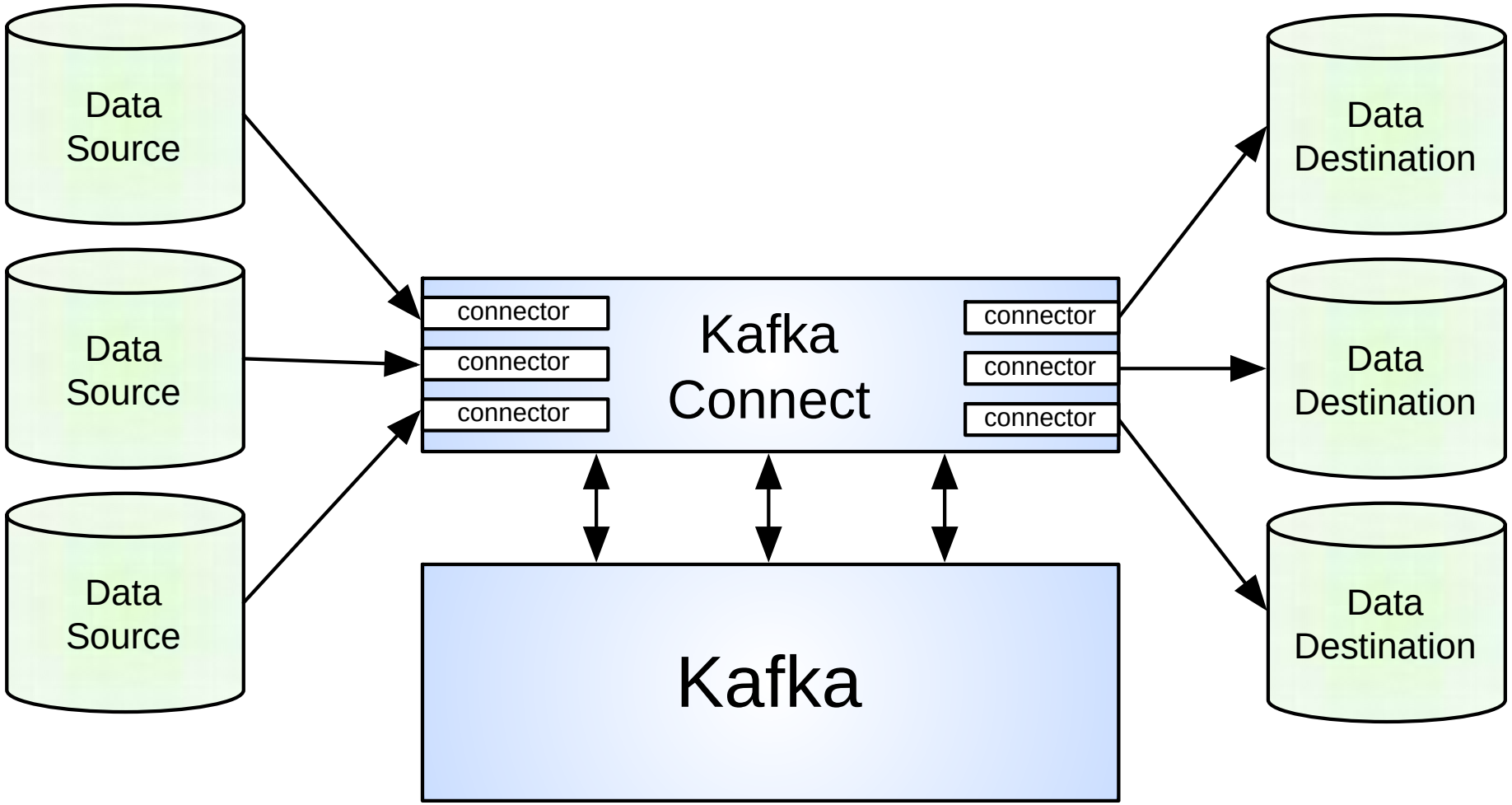
Source connectors:



Sink connectors:



Kafka Connect : Source & Sink Connectors



How is Connect different than a producer or consumer?

- Producers and consumers provide complete flexibility to send any data to Kafka or process it in any way.
  - This flexibility means you do everything yourself.

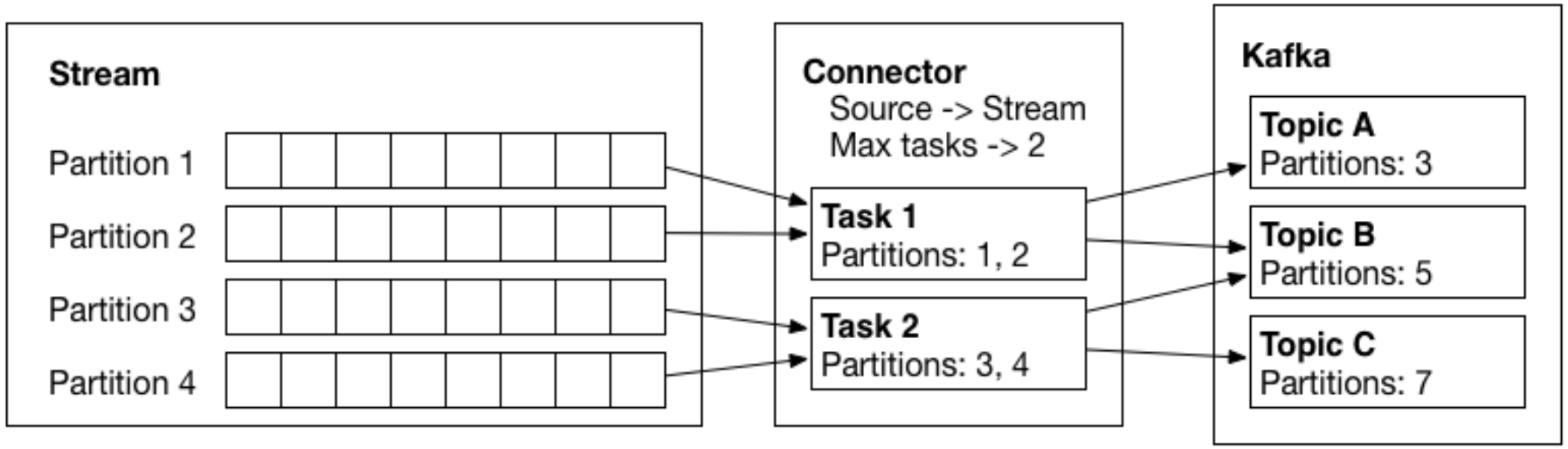
- Kafka Connect's simple framework allows :
  - developers to create connectors that copy data to/from other systems.
  - operators/users to use said connectors just by writing configuration files and submitting them to Connect -- no code necessary
  - community and 3rd-party engineers to build reliable plugins for common data sources and sinks
  - deployments to deliver fault tolerance and automated load balancing out-of-the-box

Let the Framework Do the Hard Work

- Serialization / de-serialization
- Schema Registry integration
- Fault tolerance / failover
- Partitioning / scale-out
- ***… and let the developer to focus on domain specific copying details***

Kafka Connect Architecture (low-level)

Connect has three main components: Connectors, Tasks, and Workers



In practice, a stream partition could be a database table, a log file, etc.

Standalone *vs* Distributed Mode

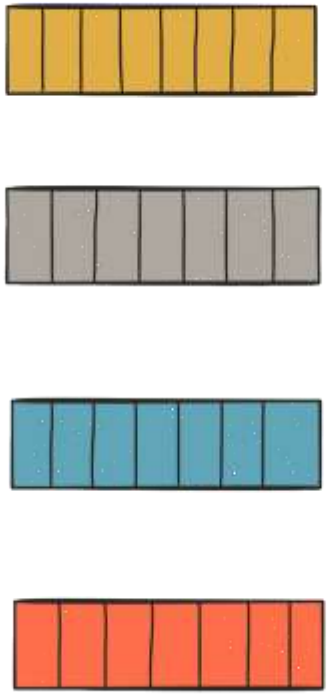
Connect workers can run in two modes: standalone or distributed

Standalone mode runs in a single process -- ideal for testing and development, and consumer/producer use cases that need not be distributed (for example, tailing a log file).

Distributed mode runs in multiple processes on the same machine or spread across multiple machines. Distributed mode is fault tolerant and scalable (thanks to Kafka Consumer Group support).

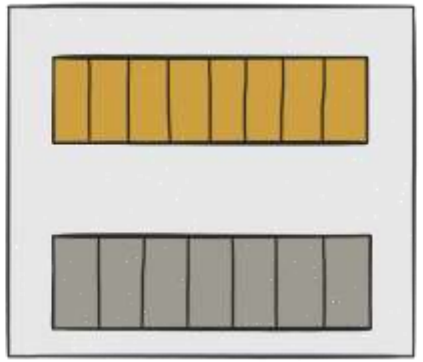
Kafka Connect Architecture (in color) : Streams -> Partitions -> Tasks

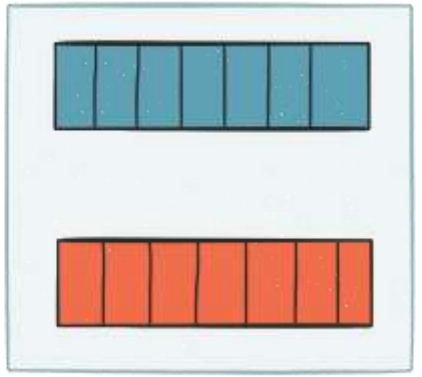
Stream of
records
(ex: Database)



Stream partitions
(ex: Database Tables)

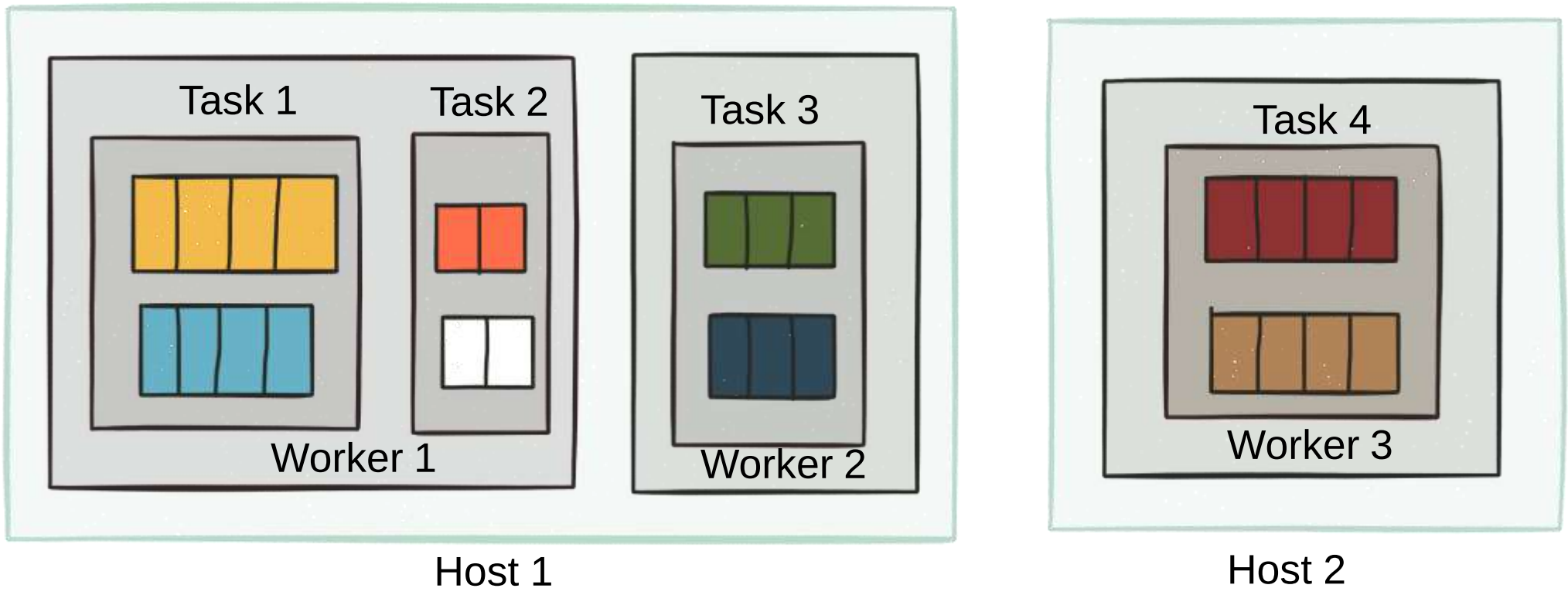
Task 1



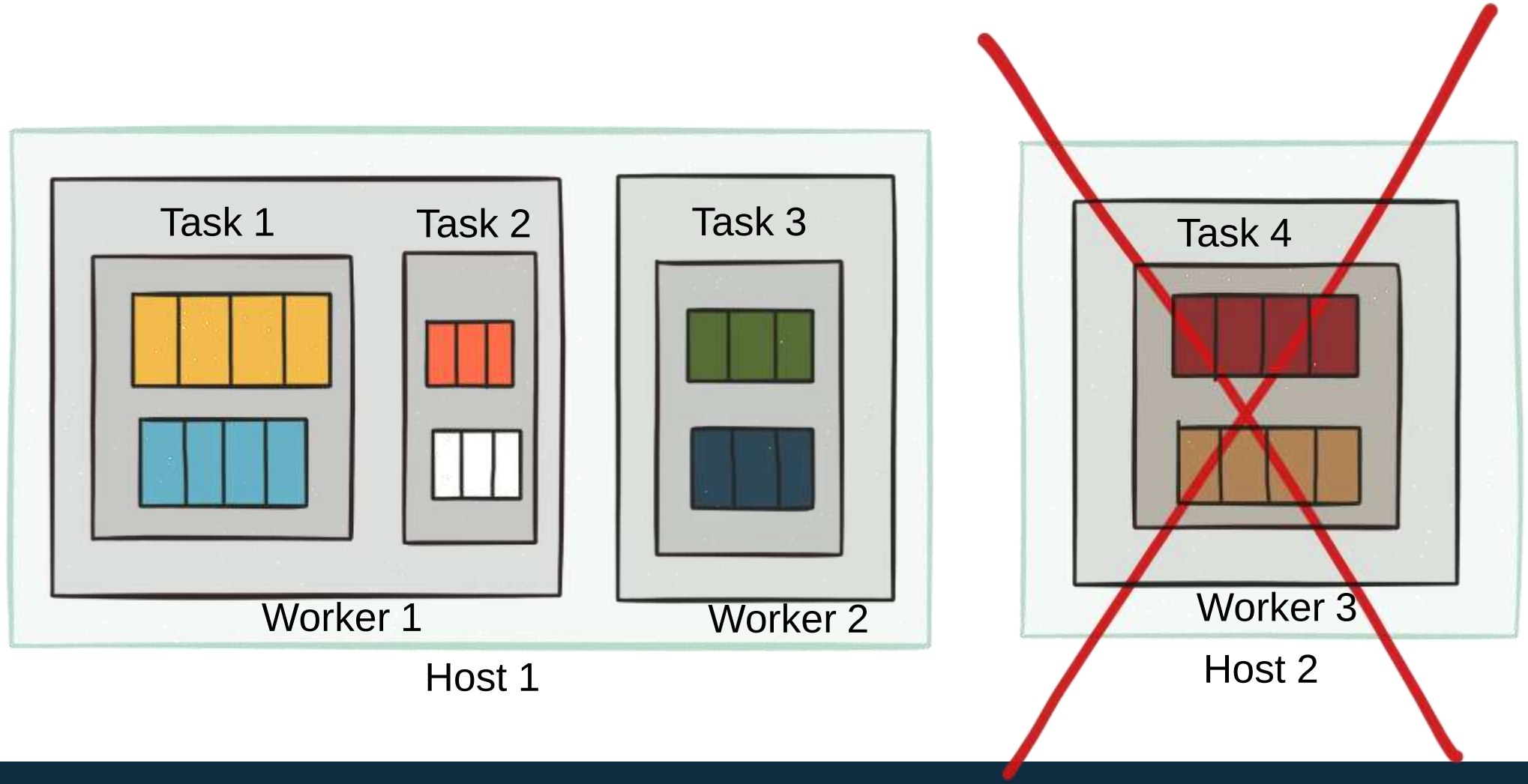


Task 2

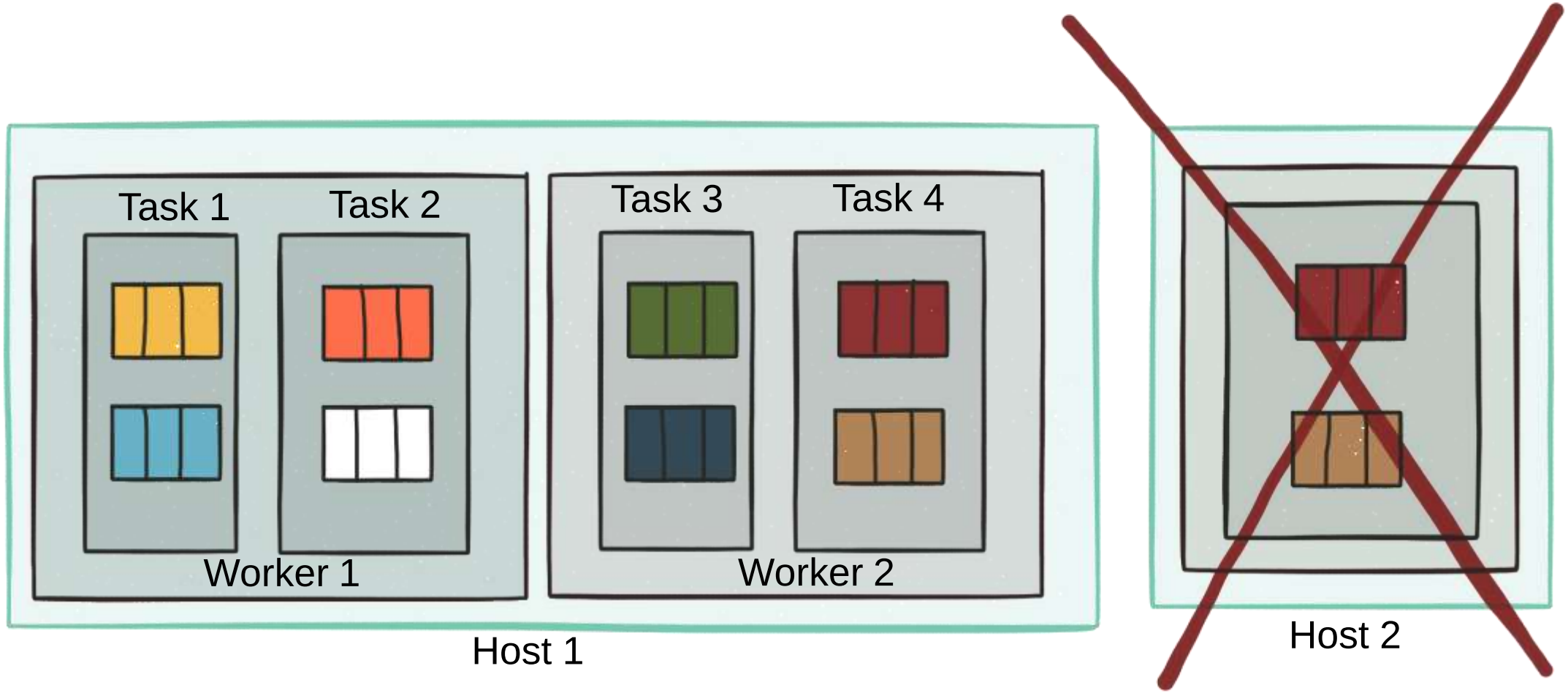
Kafka Connect Architecture : Execution Model



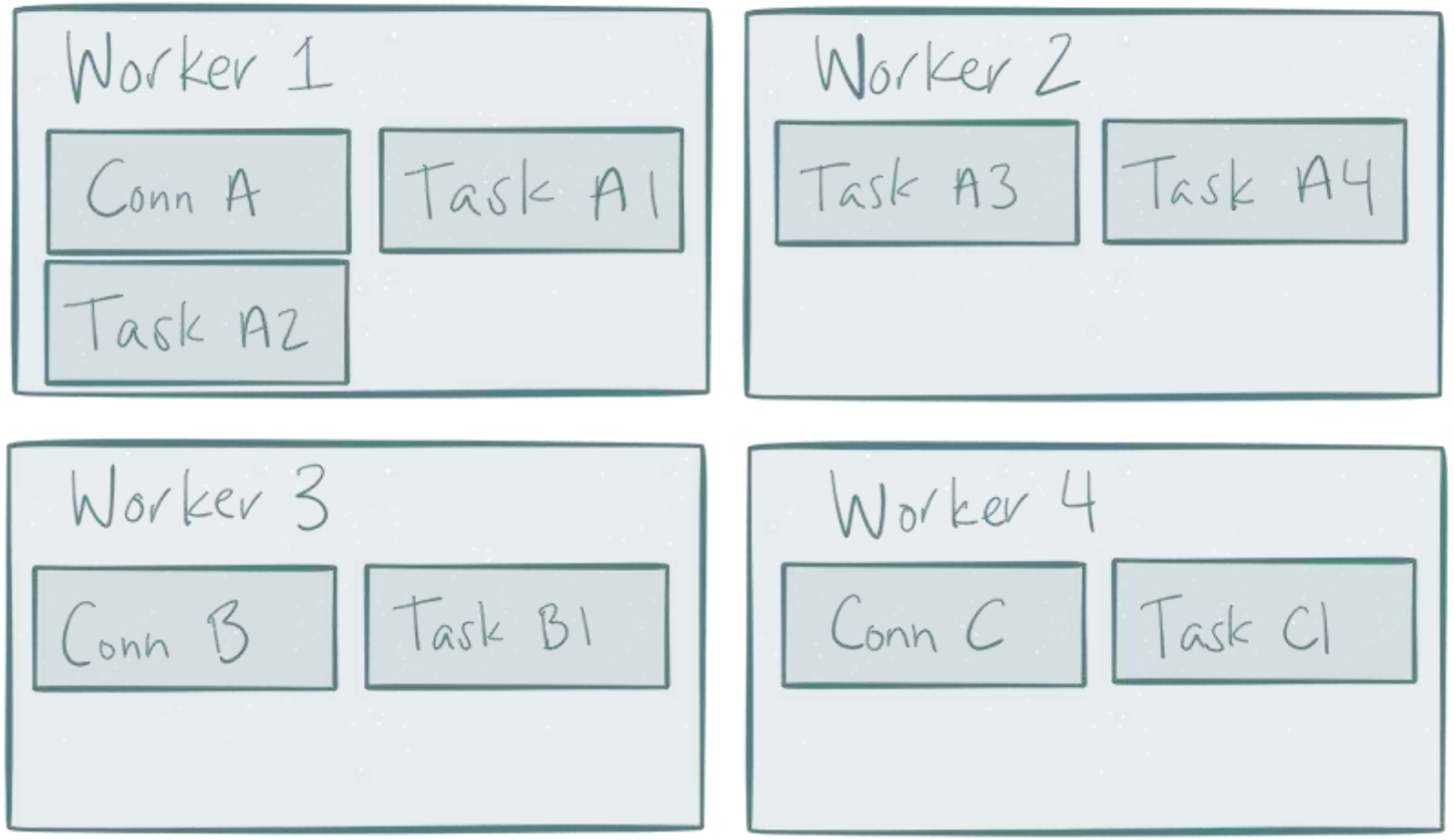
Kafka Connect Architecture : Fault Tolerance and Elasticity



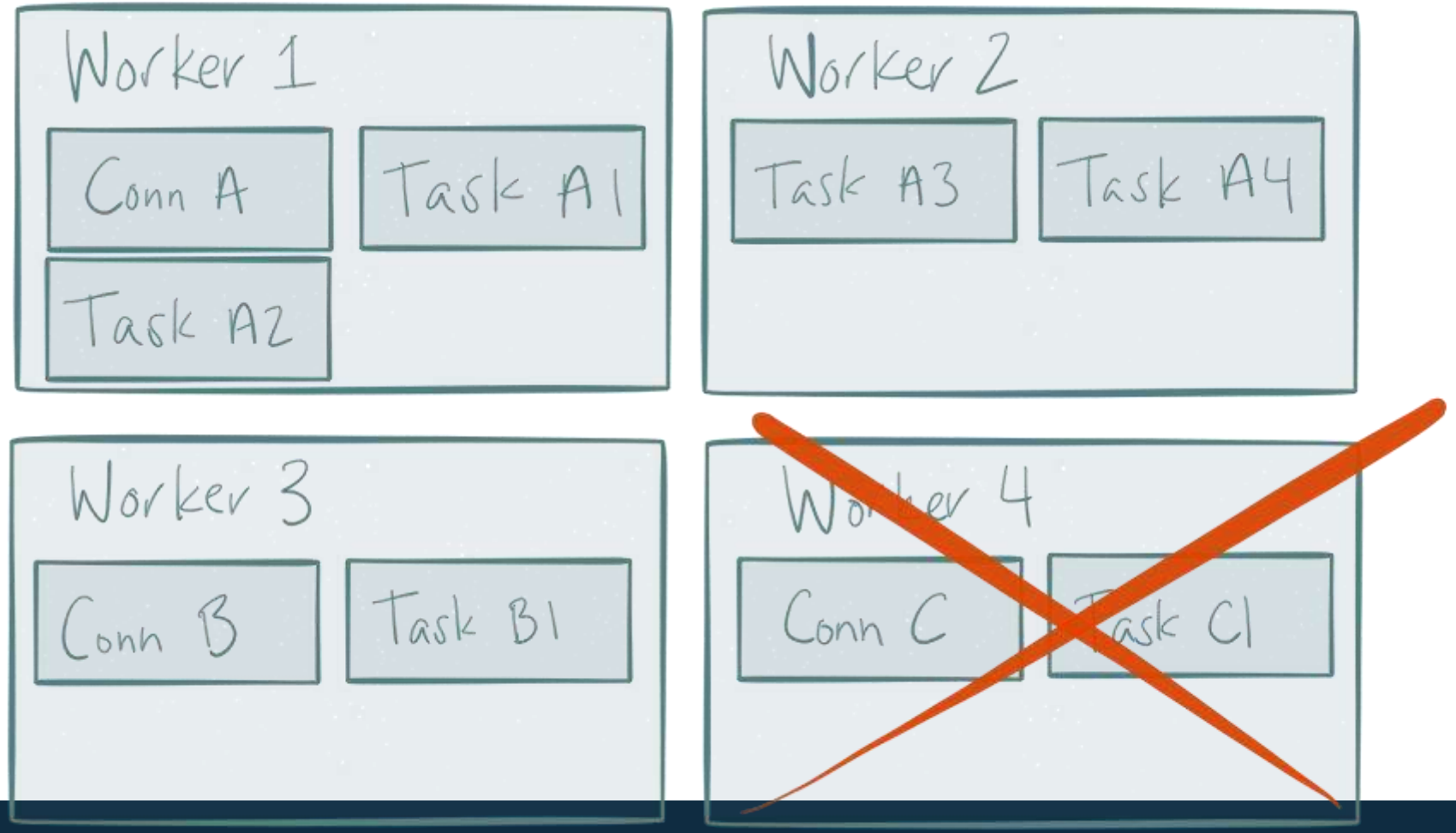
Kafka Connect Architecture : Fault Tolerance and Elasticity



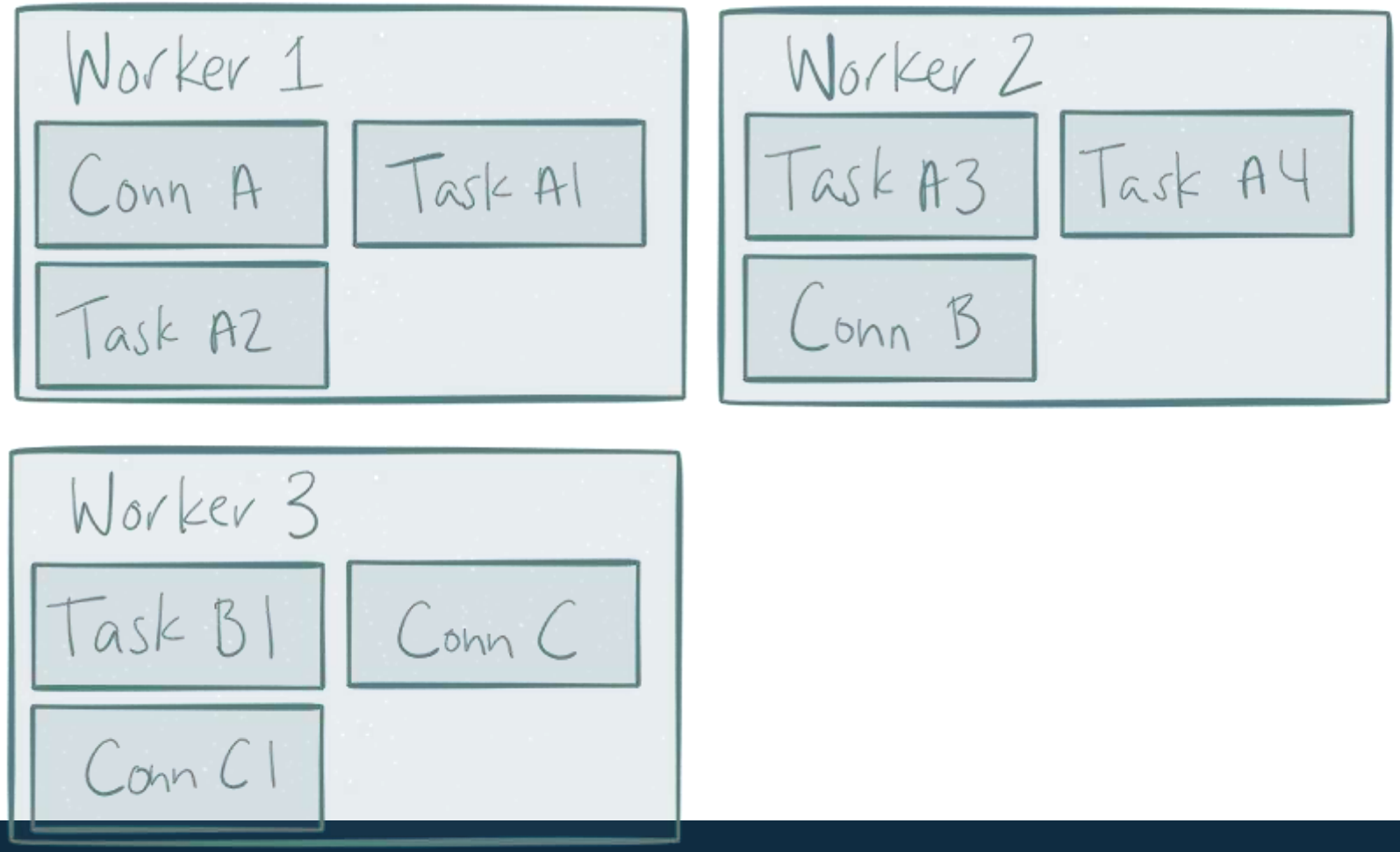
Execution Model - Distributed



Execution Model - Distributed



Execution Model - Distributed



Converters

Pluggable API to convert data between native formats and Kafka

- Source connectors: converters are invoked after the data has been fetched from the source and before it is published to Kafka
- Sink connectors: converters are invoked after the data has been consumed from Kafka and before it is stored to the sink
- Converters are applied to both key and value data

Apache Kafka ships with JSONConverter

Confluent Platform adds AvroConverter

- AvroConverter allows integration with Confluent Schema Registry to track topic schemas and ensure compatible schema evolution.

Source Offset Management

Connect tracks the offset that was last consumed for a source, to restart tasks at the correct “starting point” after a failure. These offsets are different from Kafka offsets -- they’re based on the source system (eg a database, file, etc).

In standalone mode, the source offset is tracked in a local file.

In distributed mode, the source offset is tracked in a Kafka topic.

Connect configuration (1 of 2)

The following discrete configuration dimensions exist for Connect:

- Distributed mode vs standalone mode
- Connectors
- Workers
- Overriding Producer and Consumer settings

Connect configuration (2 of 2)

In **standalone** mode, Connector configuration is stored in a file and specified as a command-line argument.

In **distributed** mode, Connector configuration is set via the REST API, in the JSON payload.

Connect configuration options are as follows:

Parameter	Description
name	Connector's unique name.
connector.class	Connector's Java class.
tasks.max	Maximum tasks to create. The Connector may create fewer if it cannot achieve this level of parallelism.
topics (sink connectors only)	List of input topics (to consume from).

Configuring workers

Worker configuration is specified in a configuration file that is passed as an argument to the script starting Connect.

The following slides will go through important configuration options in three dimensions: standalone mode, distributed mode, and parameters common to both modes.

See docs.confluent.io for a comprehensive list, along with an example configuration file.

Configuring Kafka Connect Workers : Core Settings

These parameters define the worker connection to the Kafka cluster.
See docs.confluent.io for a comprehensive list.

Parameter	Description
bootstrap.severs	A list of host/port pairs to use for establishing the initial connection to the Kafka cluster.
key.converter	Converter class for key Connect data.
value.converter	Converter class for value Connect data.

Configuring Kafka Connect Workers : Task / Storage Control

- Standalone mode

Parameter	Description
offset.storage.file.filename	The file to store Connector offsets in.

- Distributed mode

Parameter	Description
group.id	A unique string that identifies the Connect cluster group this worker belongs to.
config.storage.topic	The topic to store Connector and task configuration data in. This must be the same for all workers with the same group.id.
offset.storage.topic	The topic to store offset data for Connectors in. This must be the same for all workers with the same group.id.
session.timeout.ms	The timeout used to detect failures when using Kafka's group management facilities.
heartbeat.interval.ms	The expected time between heartbeats to the group coordinator when using Kafka's group management facilities. Must be smaller than session.timeout.ms.

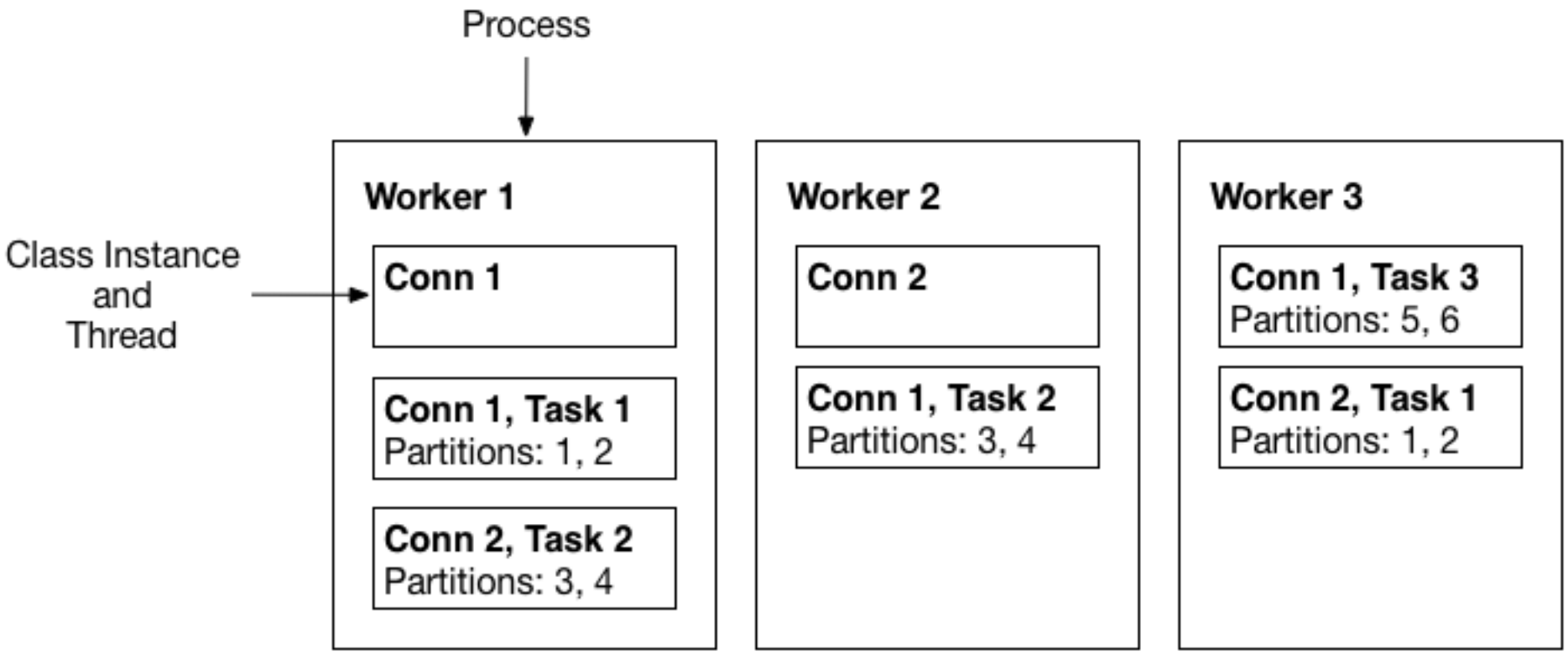
Running in standalone mode

Starting Connect in standalone mode involves starting a process with one or more connect configurations:

connect-standalone worker.properties connector1.properties [connector2.properties connector3.properties ...]

Each connector instance will be run in its own thread. Configuration will be covered later.

Distributed Mode



Running in distributed mode

Starting Connect in distributed mode involves starting connect on each worker node with the following:

connect-distributed worker.properties

Connectors are then added, modified, and deleted via a REST API. Configuration will be covered later.

Managing distributed mode

To install a new connector, the connector needs to be packaged in a JAR in placed in the CLASSPATH of each worker process.

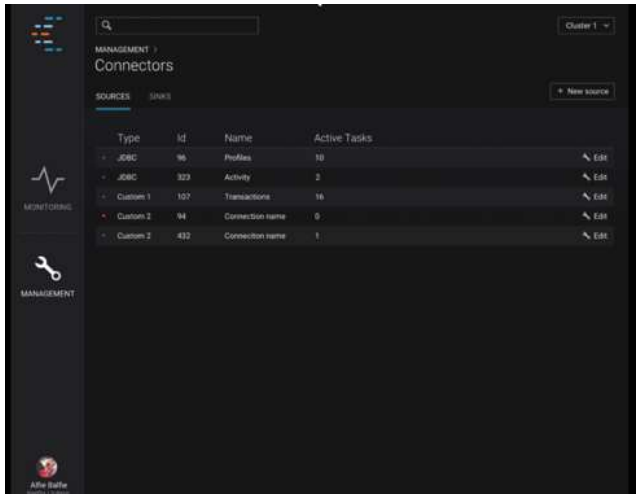
A REST API is used to create, list, modify, or delete connectors in distributed mode. All worker processes listen for REST requests, by default on port 8083. REST requests can be made to any worker node.

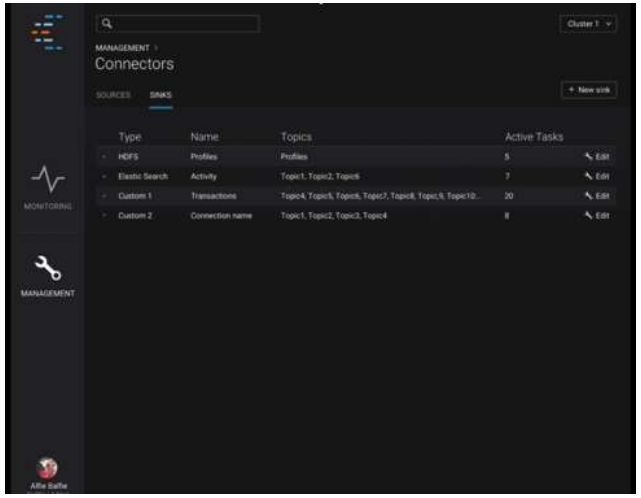
Kafka Connect: Basic REST API

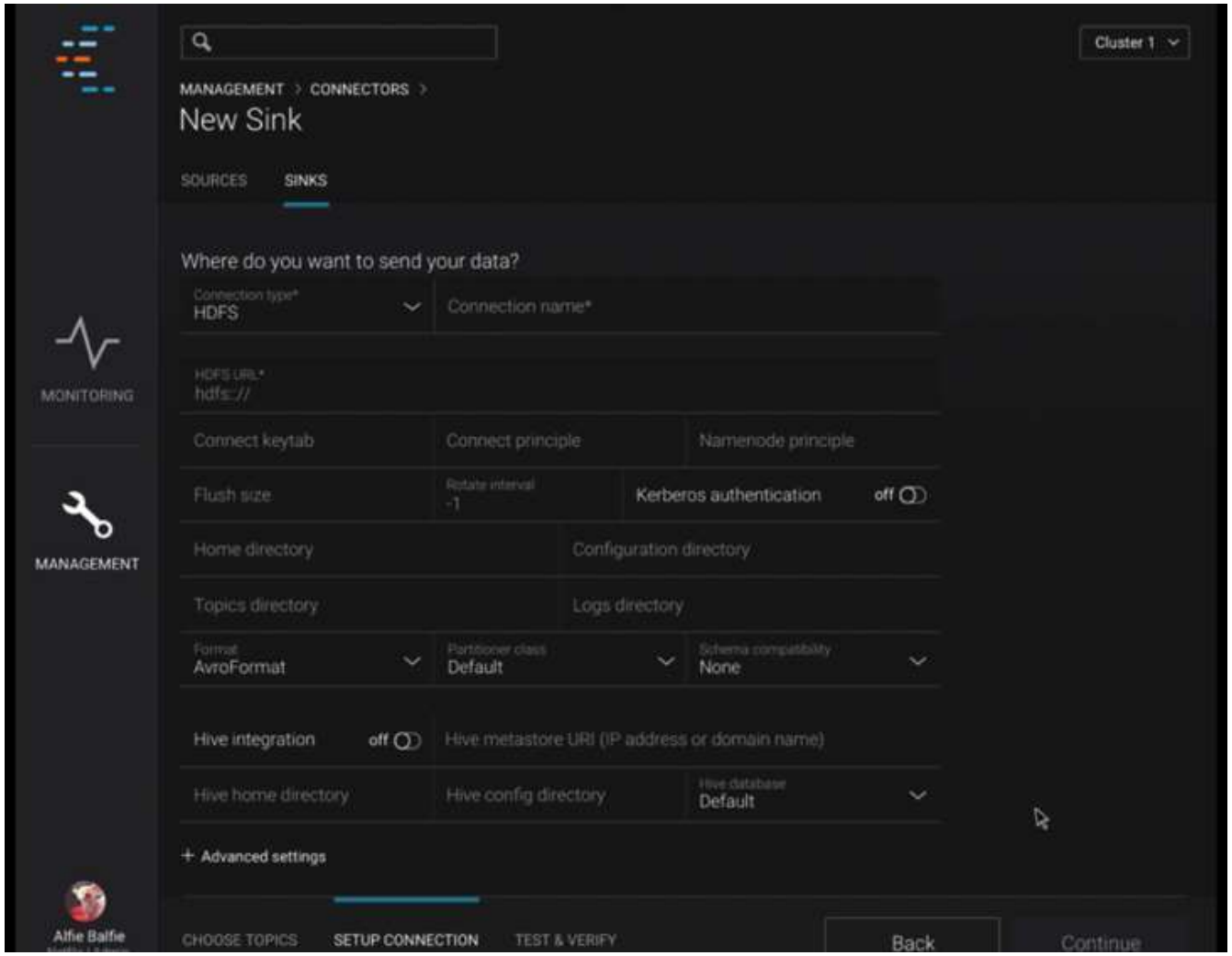
Method	Path	Description
GET	/connectors	Get a list of active connectors.
POST	/connectors	Create a new connector.
PUT	/connectors/(string: name)/config	Create a new connector, or update the configuration of an existing connector.
GET	/connectors/(string: name)/config	Get configuration info for a connector.
GET	/connectors/(string: name)/tasks/<task-id>/	Retrieve details for specific tasks
DELETE	/connectors/(string: name)	Delete a configured connector from the worker pool

Learn more about the REST API at docs.confluent.io

Connector Management

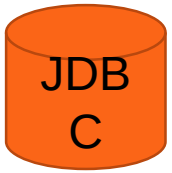






Kafka Connectors

- Confluent-supported connectors (included in Confluent Platform)



- Community-written connectors (just a sampling)

More Connectors Coming

- Cassandra Sink (Data Mountaineers)
- Elastic Search Connector (Confluent)
- JDBC source **and** sink

Connector Hub

http://www.confluent.io/product/connectors

CONNECTOR	CATEGORY	DEVELOPER	SUPPORT
HDFS (Sink)	File System	Confluent	Confluent
JDBC (Source)	RDBMS, DataStore	Confluent	Confluent
Attunity (Sink)	RDBMS, DataStore	Attunity	Attunity
Couchbase (Source)	DataStore	Couchbase	Couchbase
JustOne (Sink)	RDBMS, DataStore	JustOne	JustOne
Striim (Source)	DataStore	Striim	Striim

Local File Source and Sink Connector

- The local file **source** Connector tails a local file
  - Each line is published as a Kafka message to the target topic

- The local file **sink** Connector appends Kafka messages to a local file
  - Each message is written as a line to the target file.

- Both the **source** and **sink** Connectors need to be run in standalone mode.

JDBC Source Connector

The JDBC source Connector periodically polls a relational database for new or recently modified rows, creates an Avro record, and produces the Avro record as a Kafka message.

Records are divided into Kafka topics based on table name.

New and deleted tables are handled automatically by the Connector.

HDFS Sink Connector

The HDFS Sink Connector writes Kafka messages into target files in an HDFS cluster. Kafka topics are partitioned into chunks by the connector, and each chunk is stored as an independent file in HDFS. The nature of the partitioning is configurable (by default, the partitioning defined for the Kafka topic itself is preserved) .

The connector integrates directly with Hive. Configuration options enable the automatic creation/extension of an external Hive partitioned table for each Kafka topic.

The connector supports exactly-once delivery semantics, so there is no risk of duplicate data for Hive queries.

Connector Development : Best Practices

- Define data stream for reasonable parallelism and utility
- Develop flexible message syntax, leveraging Schema Registry
  (users can opt for JSONConverter, so no lock-in)
- Check the community frequently!

Connect Security

- Supports SASL and TLS/SSL
- Uses java keystores/truststores

- ```
  # Source security settings are prefixed with "producer"
  producer.security.protocol=SSL
  producer.ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
  producer.ssl.truststore.password=test1234
  ```

- ```
  # Sink security settings are prefixed with "consumer"
  consumer.security.protocol=SSL
  consumer.ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
  consumer.ssl.truststore.password=test1234
  ```

Kafka Connect : Benefits for Developers

✓ Simplifies data flow in to and out of Kafka

✓ Reduces development costs compared to custom connectors
- Integrated schema management
- Task partitioning and rebalancing
- Offset management
- Fault tolerance
- Delivery semantics, operations, and monitoring

References

Confluent docs: http://docs.confluent.io/current/connect/index.html

Connectors: http://www.confluent.io/product/connectors/

Thank You