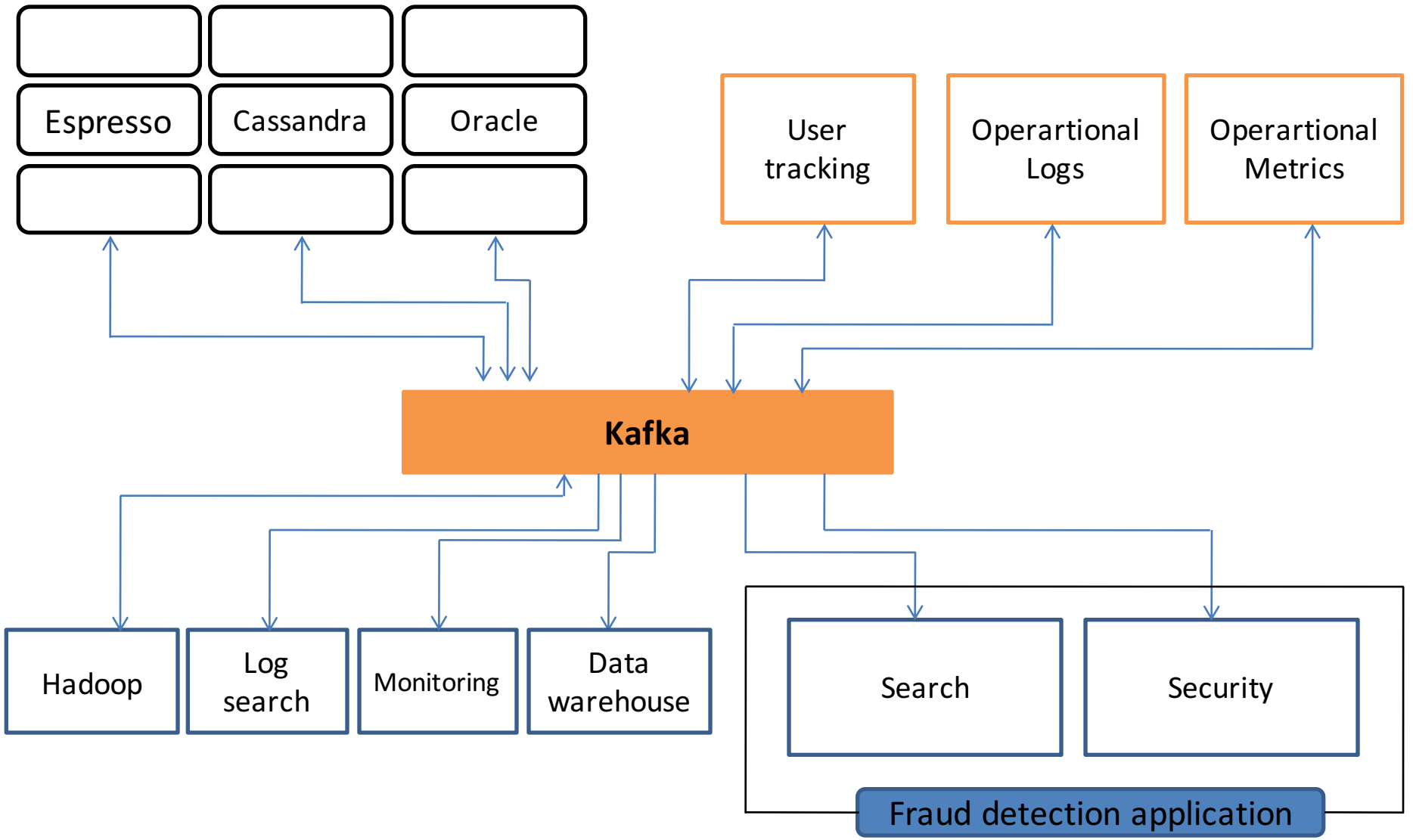
Kafka Connect

Central Hub with Kafka

Tos



Kafka Connect

Tos

Simplified, scalable data import/export for Kafka

What is Kafka Connect?

Tos

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?

Tos

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

Many common requirements

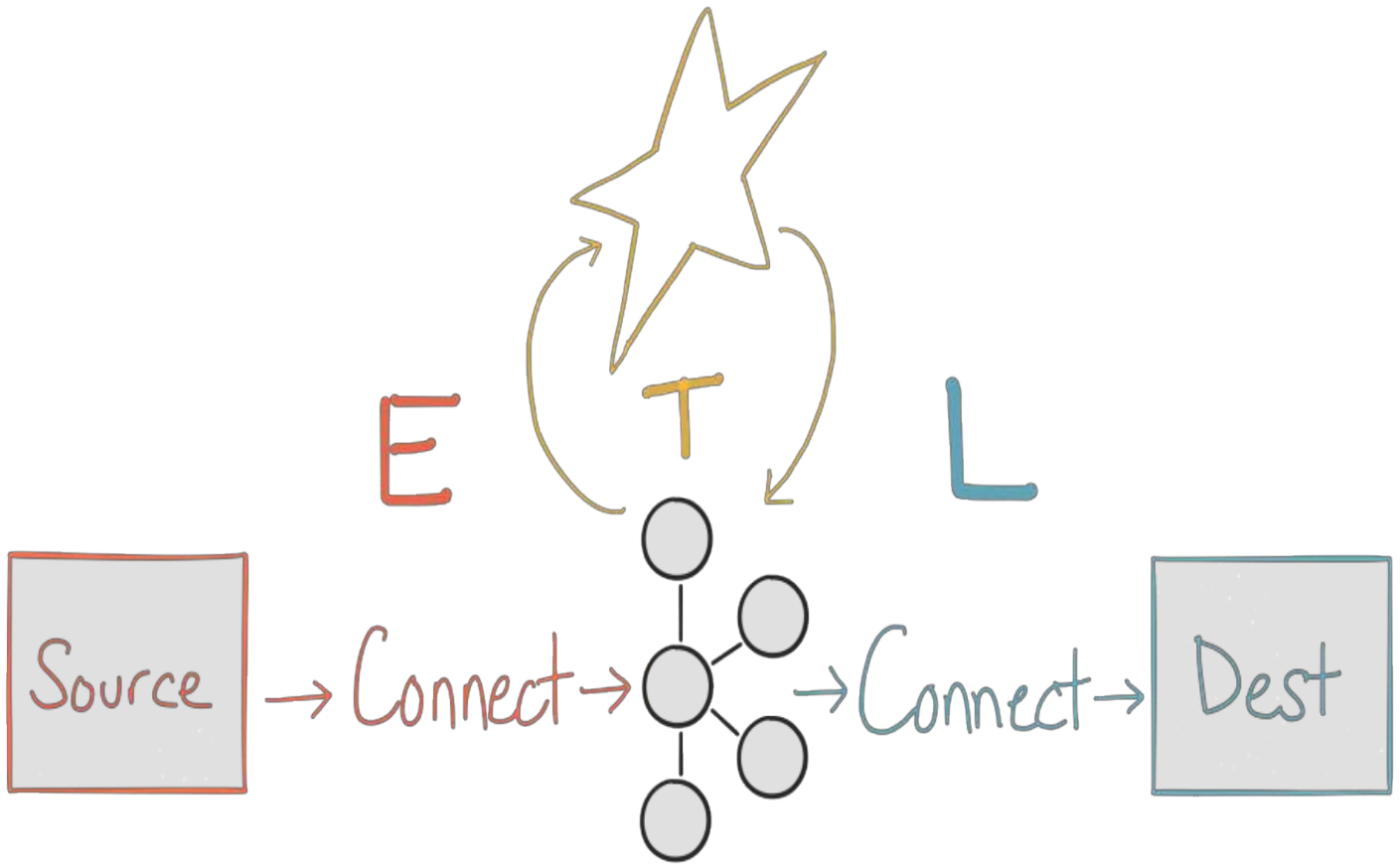
- Data conversion (serialization)
- Parallelism / scaling
- Load balancing
- Fault tolerance / fail-over
- General management

A few data-specific requirements

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

Kafka Connect : Separation of Concerns

Tos



High-level Model

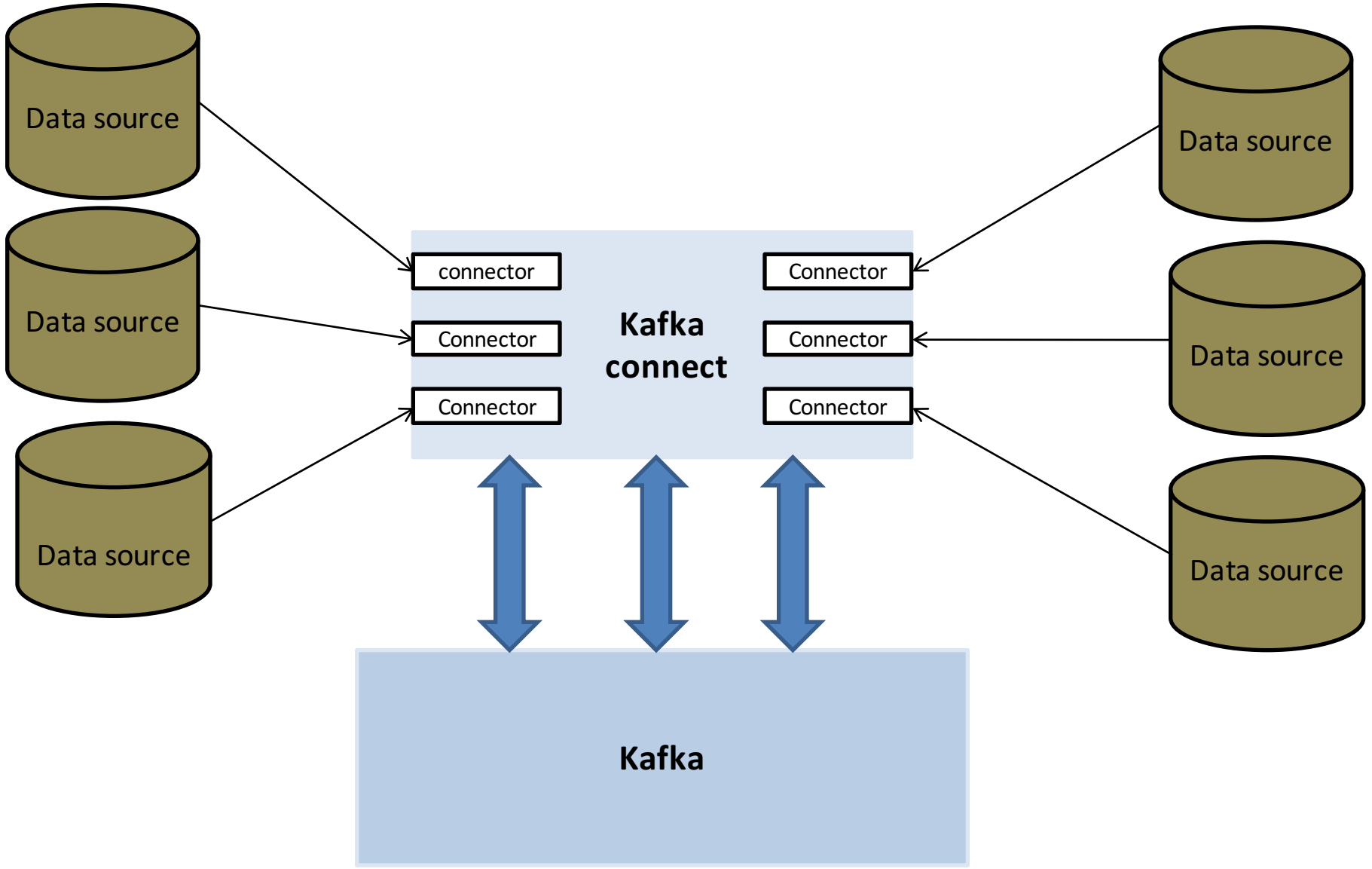
Tos

Source connectors:

Sink connectors:

Kafka Connect : Source & Sink Connectors

Tos



How is Connect different than a producer or consumer?

Tos

- 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

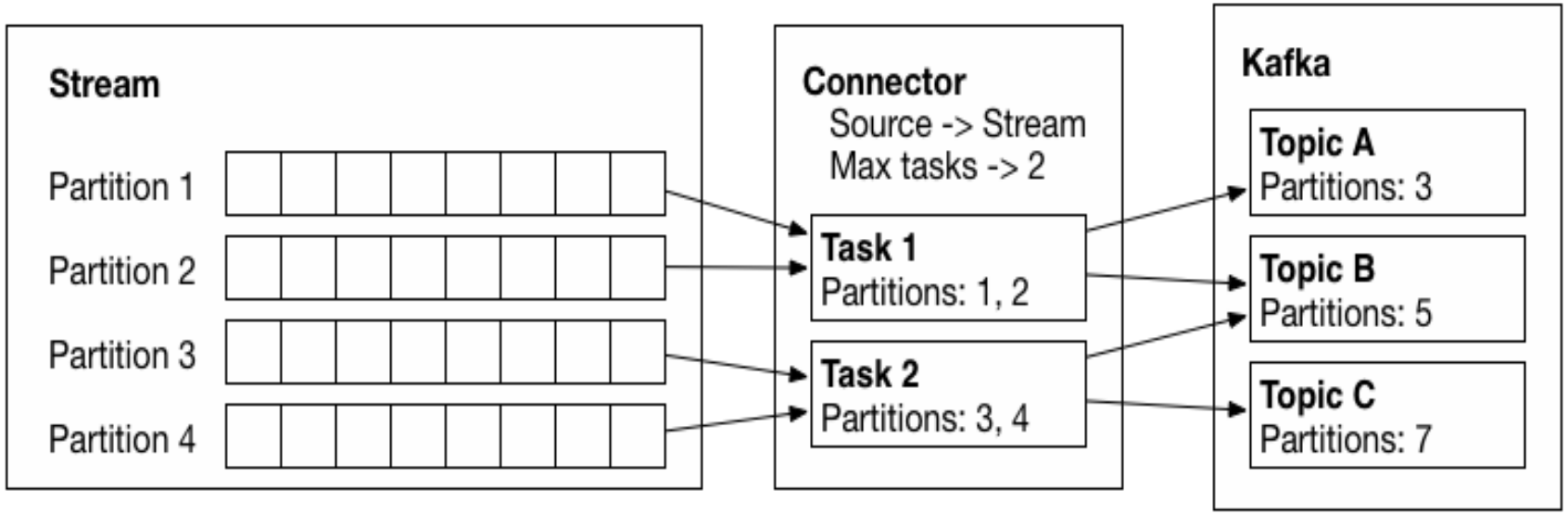
Tos

- 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)

Tos

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

Tos

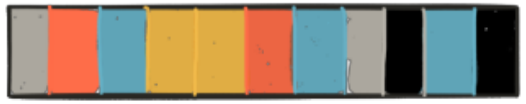
Connect workers can run in two modes: standalone or distributed

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).

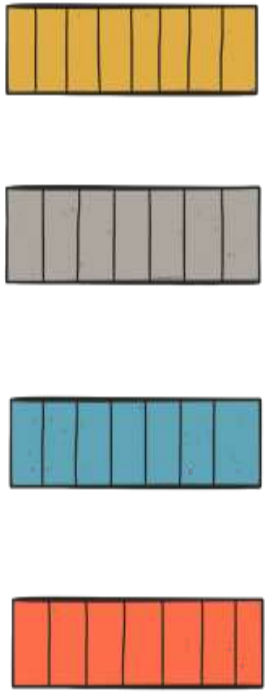
Distributed mode runs in multiple processes on the same machine or spread across multiple machines. Distributed mode is fault tolerant and scalable (thanks to Kafka Consumer Group support).

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

Tos

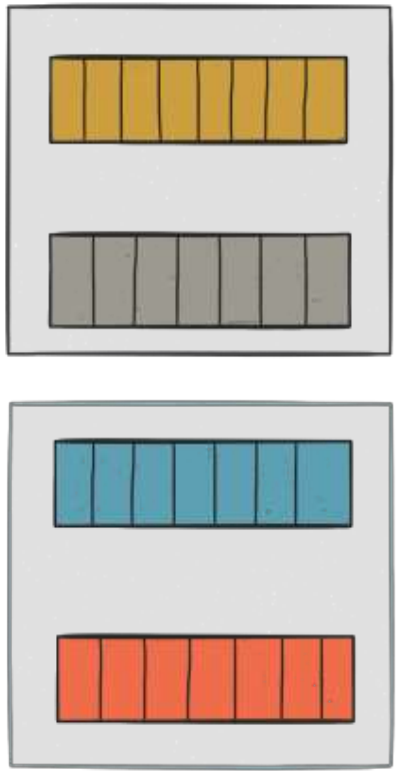


Stream of records
(ex: Database)



Stream partitions
(ex: Database Tables)

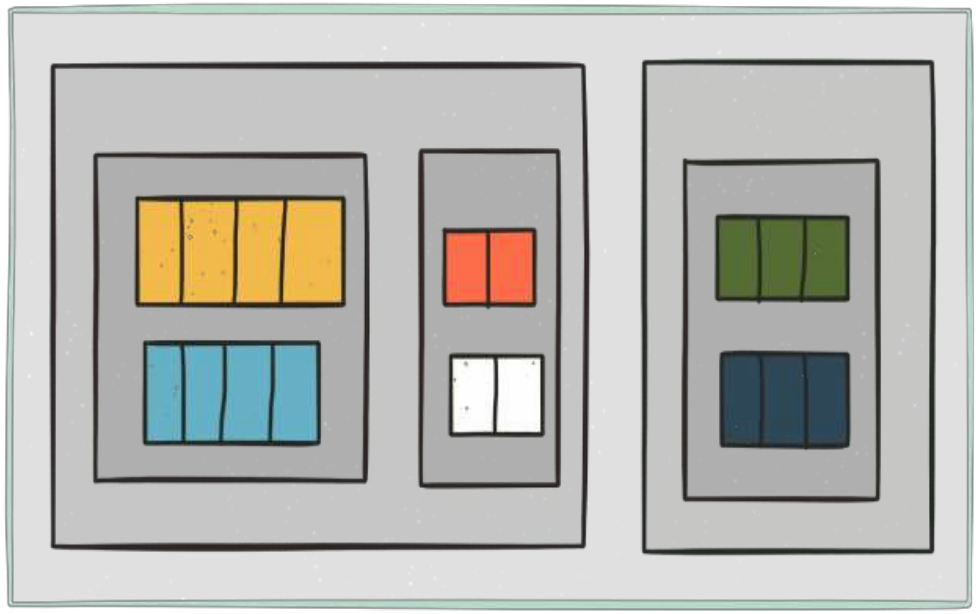
Task 1



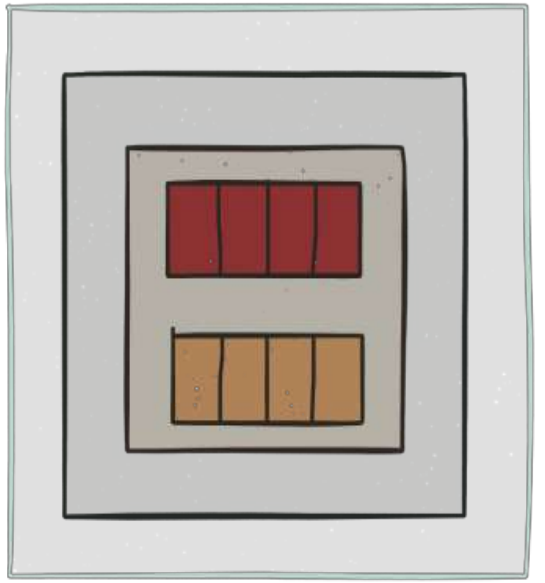
Task 2

Kafka Connect Architecture : Execution Model

Tos



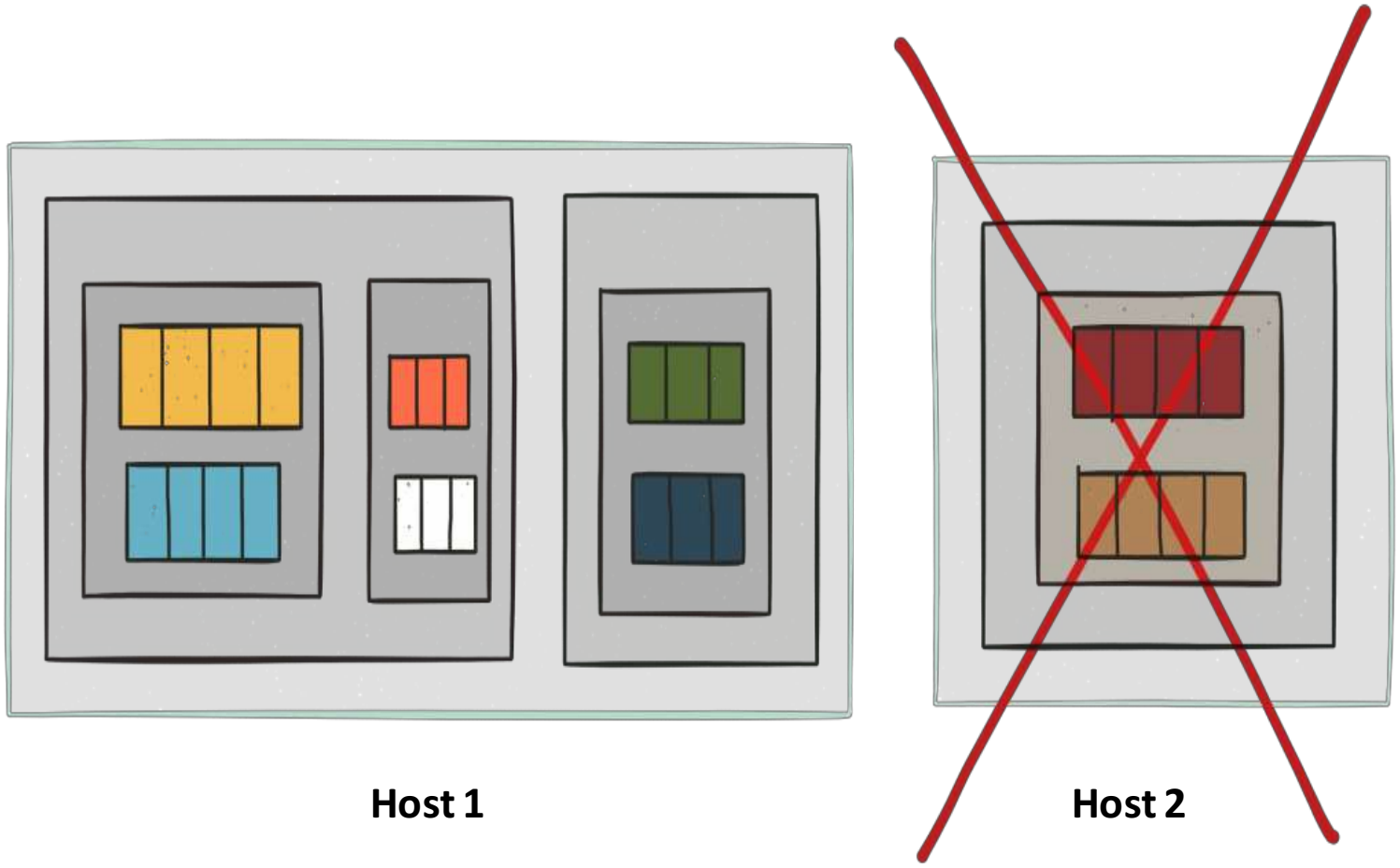
Host 1



Host 2

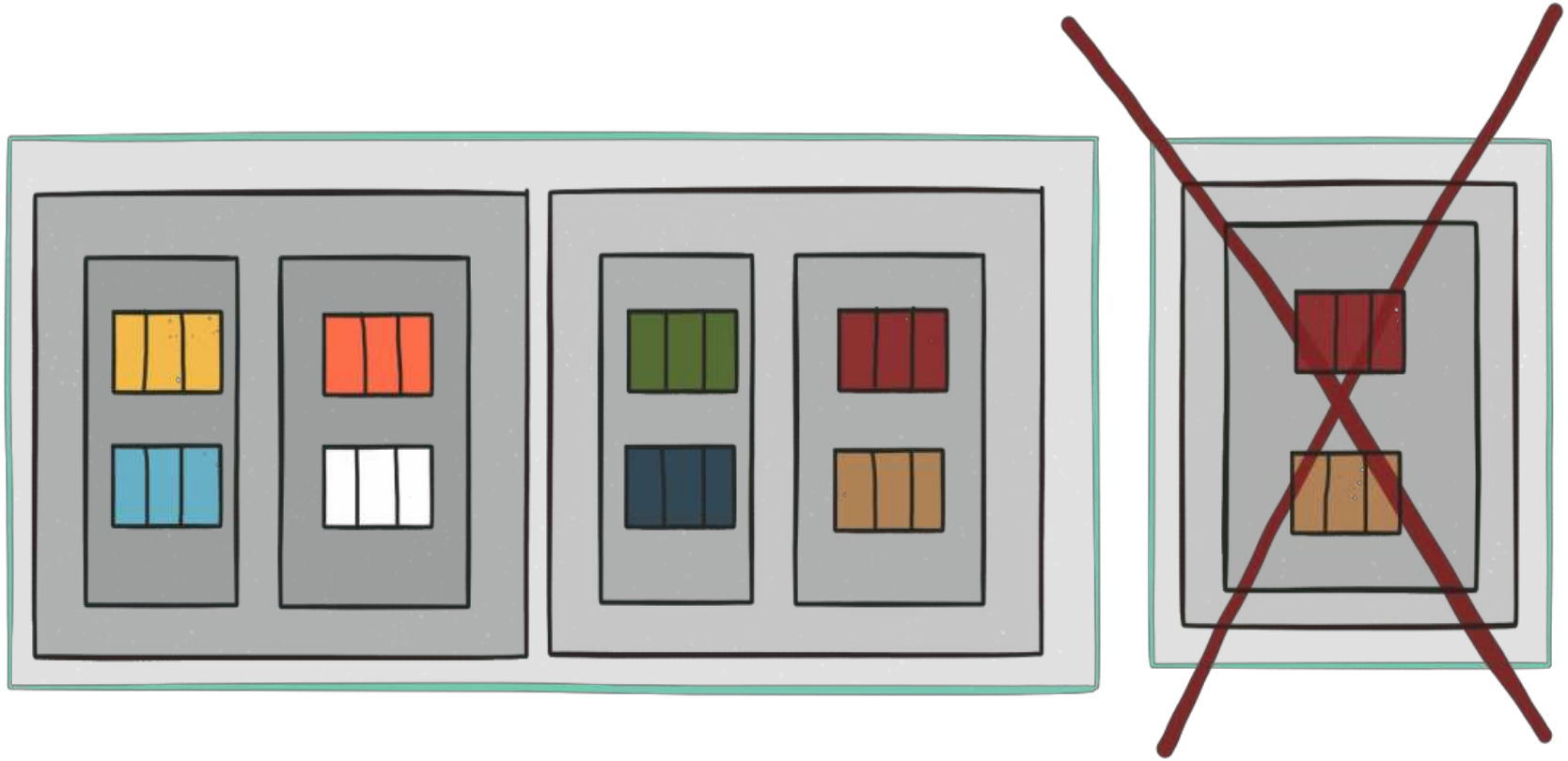
Kafka Connect Architecture:Fault Tolerance and Elasticity

Tos



Kafka Connect Architecture:Fault Tolerance and Elasticity

Tos

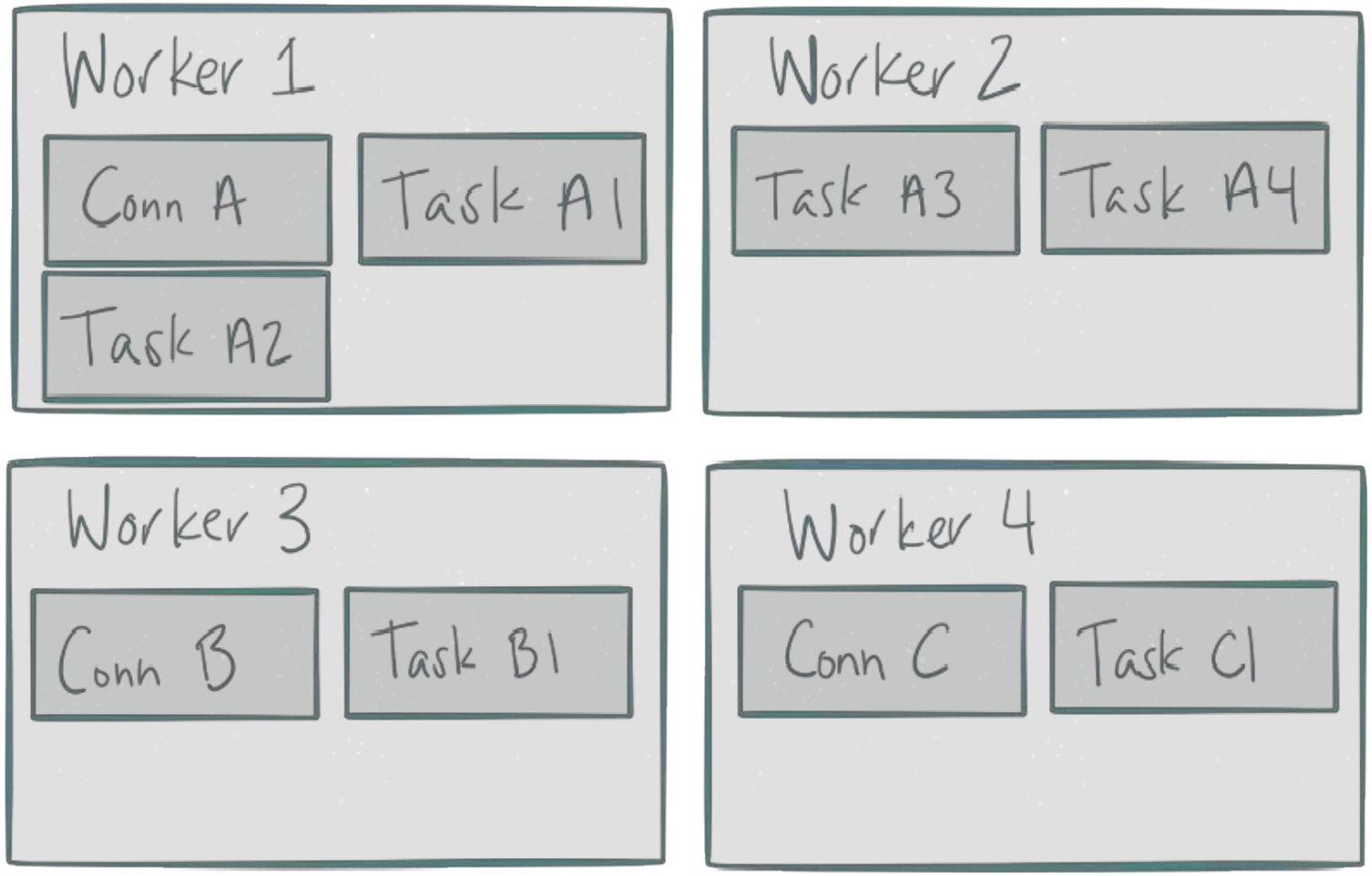


Host 1

Host 2

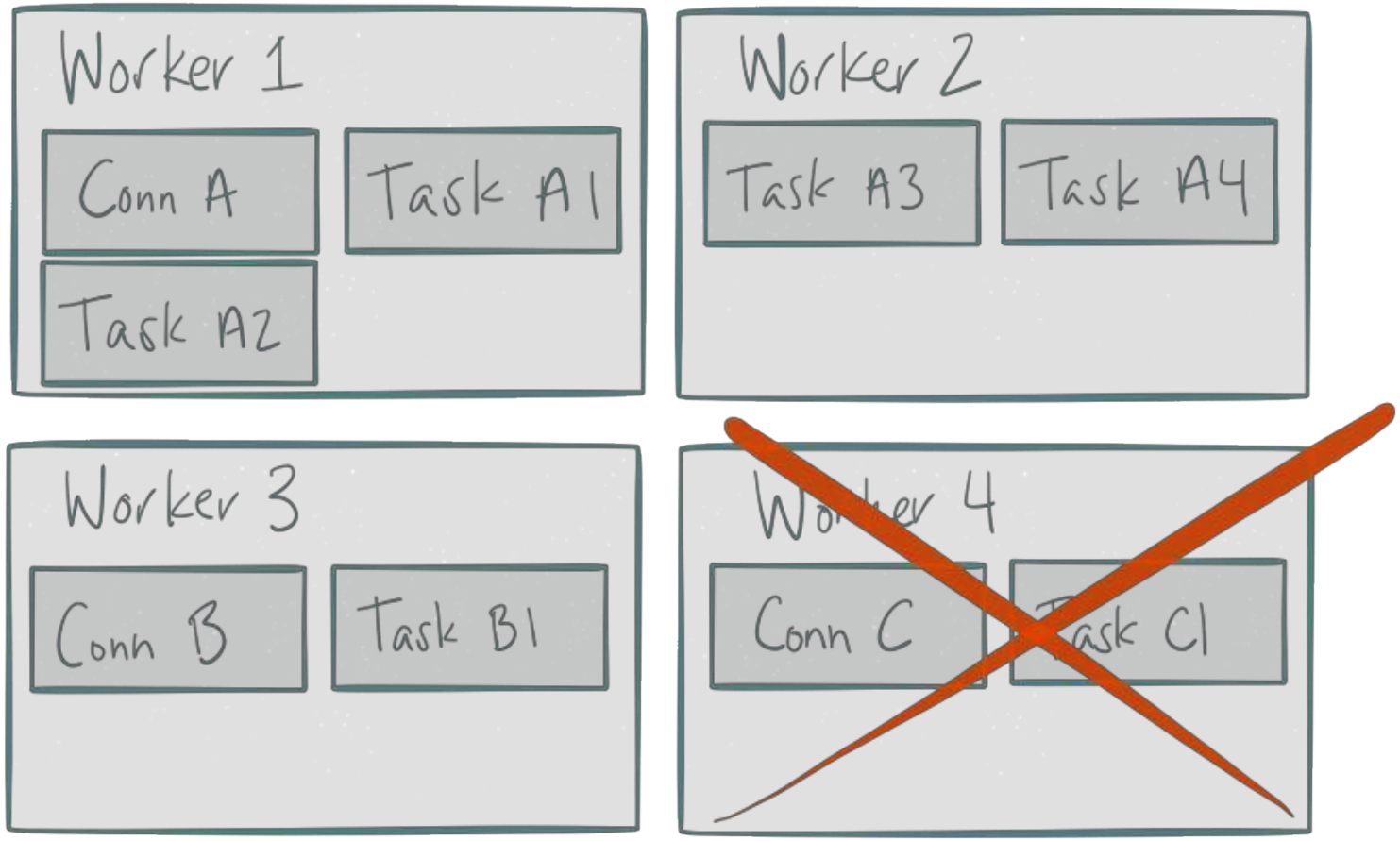
Execution Model - Distributed

Tos



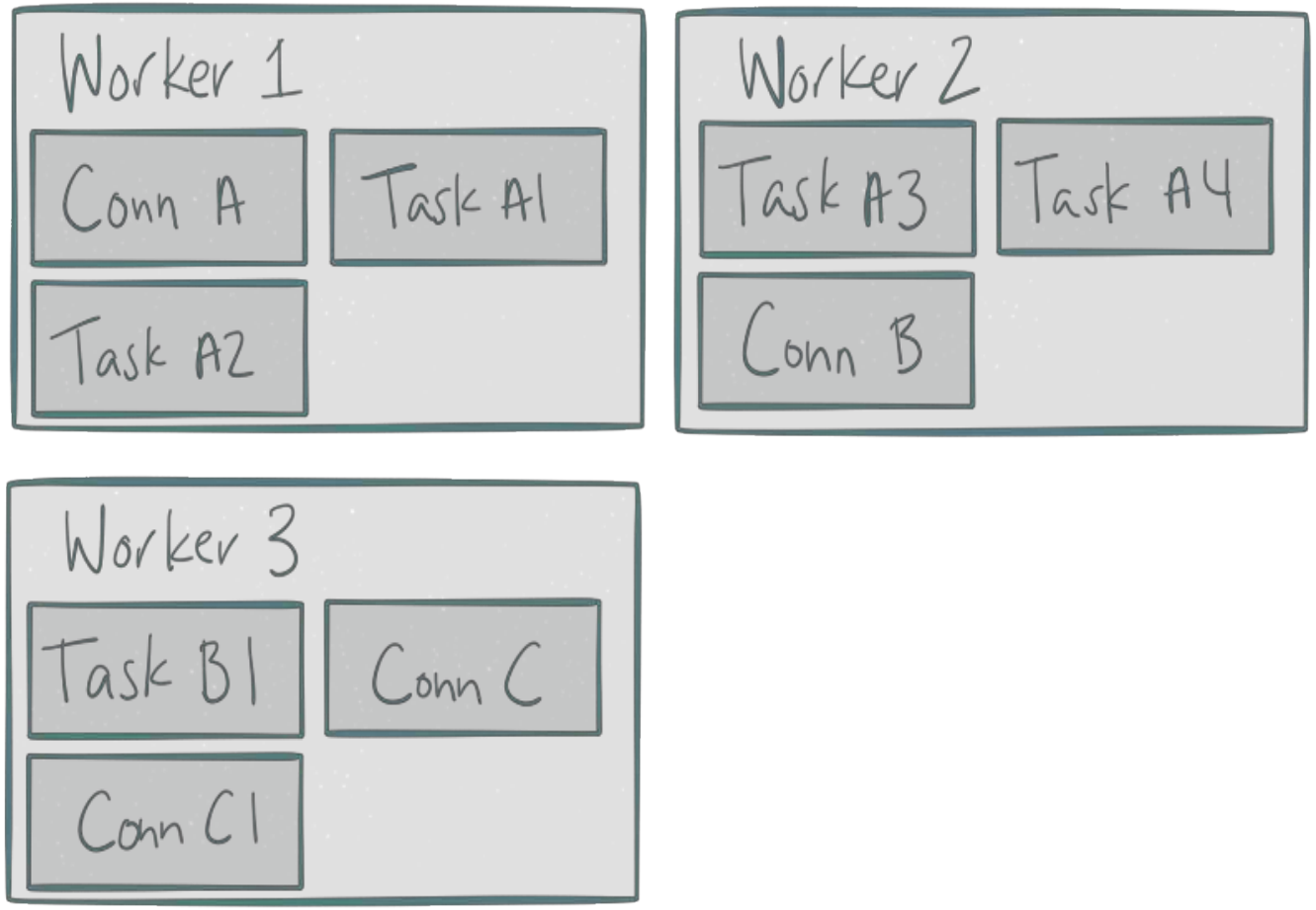
Execution Model - Distributed

Tos



Execution Model - Distributed

Tos



Converters

Tos

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

Tos

- 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)

Tos

The following discrete configuration dimensions exist for Connect:

- ❖Distributed mode Vs Standalone Mode
- ❖Mode Connectors
- ❖ Workers
- ❖ Overriding Producer and Consumer settings

Connect configuration (2 of 2)

Tos

- 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.dass	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

Tos

- 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

Tos

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

Tos

- **Standalone mode**

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

- **Distributed mode**

Parameter	Description
group.id	Aunique 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

Tos

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

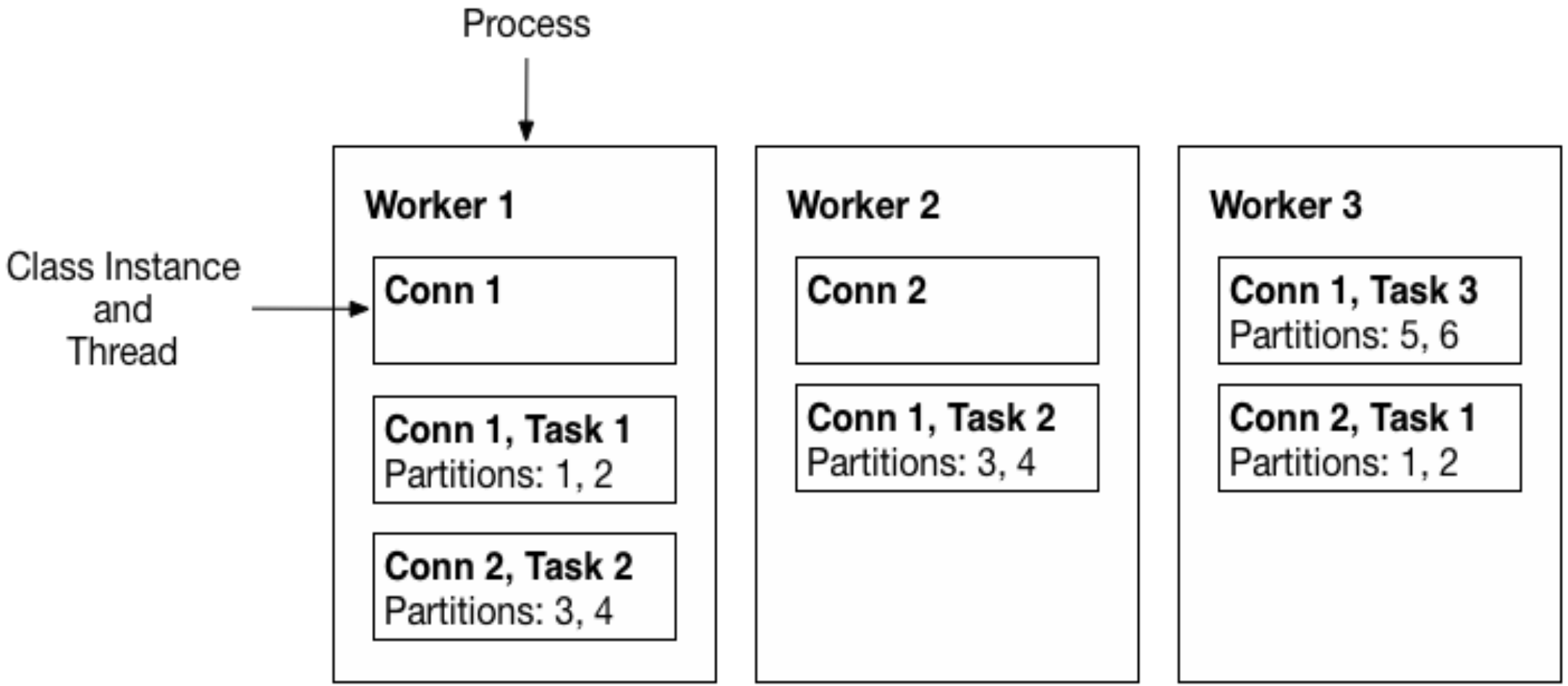
  - *Connect-standalone worker.properties connector1.properties [connector2.properties connector3.properties ...]*

- Each connector instance will be run in its own thread.

- Configuration will be covered later.

Distributed Mode

Tos



Running in distributed mode

Tos

- 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

Tos

- 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

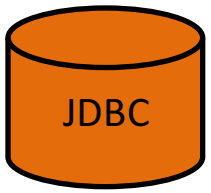
Tos

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/<tasks-id>/	Retrieve details for specific tasks
DELETE	/connectors/(string:name)	Delete a configured connector from the worker pool

Kafka Connectors

Tos

- Confluent-supported connectors (included in Confluent Platform)



- Community-written connectors (just a sampling)

Local File Source and Sink Connector

Tos

- 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

Tos

- 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

Tos

- 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

Tos

- 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

Tos

- 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

Tos

- ✓ 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

Lab : Kafka Connector