Lund&Bendsen
developing developers

Kafka connect

What is Kafka Connect

Lund&Bendsen
developing developers

Kafka Connect is for building scalable and reliable data integration pipelines between Apache Kafka and external data systems. It's part of the Apache Kafka project and is designed to simplify the process of ingesting data into Kafka and exporting data from Kafka to various data sources.

Connectors: Kafka Connect uses connectors to interact with various data sources and sinks. Connectors are available for a wide range of databases, storage systems, and other data platforms. These connectors make it easy to configure and manage the data flow.

Configurability: Kafka Connect is highly configurable. You can define and customize the data pipeline through configuration files without the need for writing code.

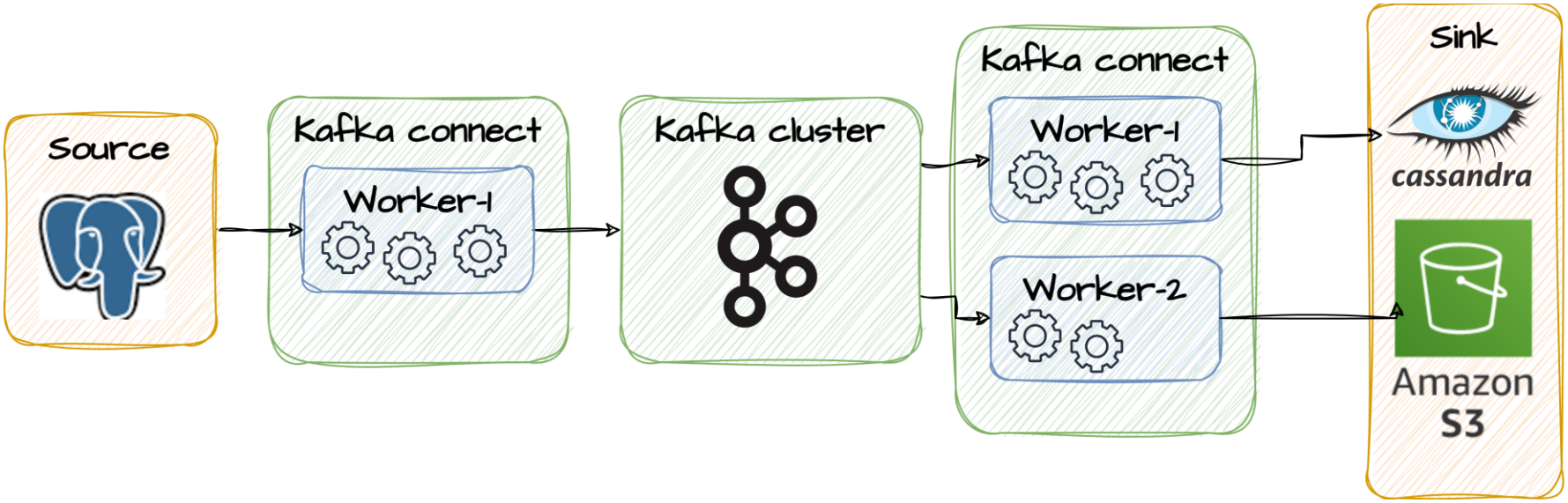
Scalability: It's designed for high throughput and fault tolerance, making it suitable for large-scale data integration tasks. You can scale it horizontally to handle increased data volumes.

Built-in Converters: Kafka Connect provides built-in converters for data serialization and deserialization. It supports various data formats like Avro, JSON, and more.

Distributed and Fault-Tolerant: Kafka Connect is built to run in a distributed and fault-tolerant manner, ensuring data integrity and availability.

Kafka Connect overview

Lund&Bendsen
developing developers



FileStreamSource example

Lund&Bendsen
developing developers

Connect-standalone.properties:

```
bootstrap.servers=kafka1:9092
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
```

Connect-file-source.properties:

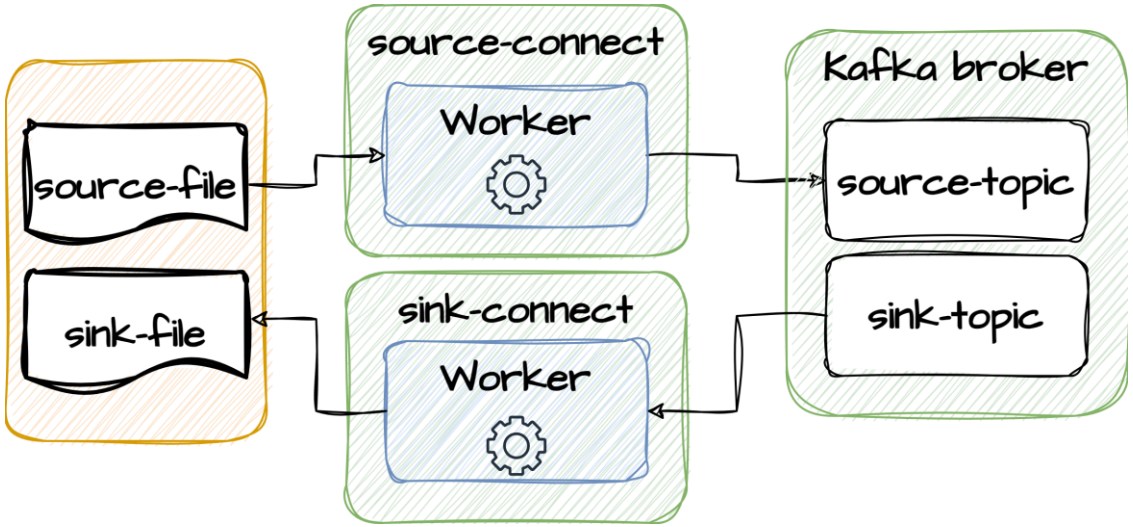
```
name=file-source-connector
connector.class=FileStreamSource
tasks.max=1
file=/tmp/source-file.txt
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
```

Exercise

Lund&Bendsen
developing developers

- Walk through of how the file connectors are configured
  - Connect-standalone.properties
  - Connect-file-sink.properties
  - Connect-file-source.properties

9.1



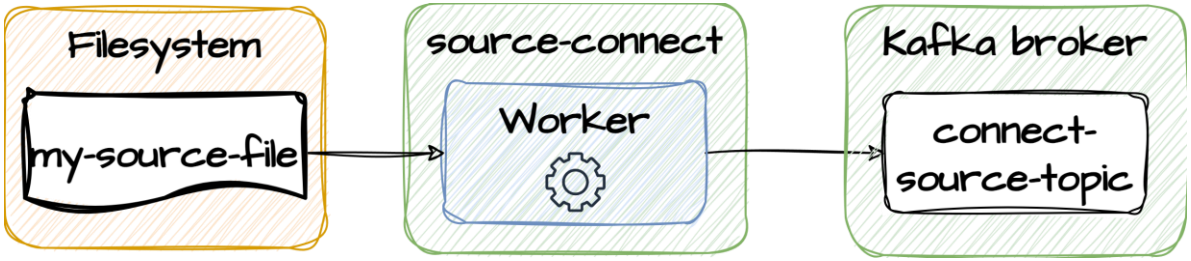
Exercise

Lund&Bendsen
developing developers

Run a Kafka connector that reads from a file and writes to a topic

- Start a consumer that consumes from the topic specified for the connector properties file
- Open the file that Kafka connect if configured to read from
- Type some lines of text and save the file
- See that the lines you just typed are reflected in the topic

9.2



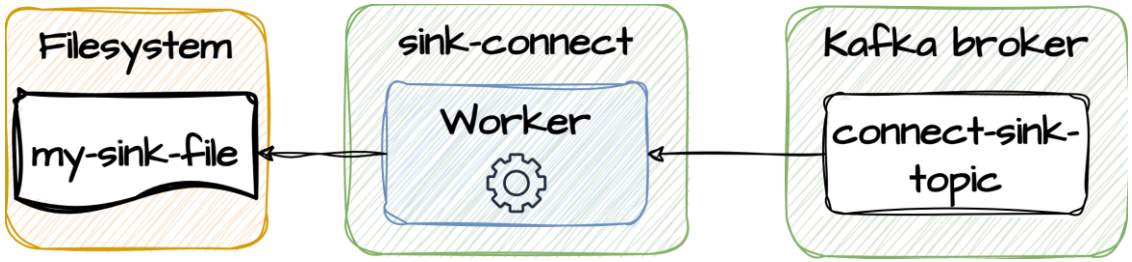
Exercise

Lund&Bendsen
developing developers

Run a Kafka connector that reads from a topic and writes to a file

- Start a producer that produces from the topic specified for the connector properties file
- Write some messages to the topic
- Open the file that Kafka connect if configured to write to as see that the messages are there

9.3



Exercise

Lund&Bendsen
developing developers

Inspect the file connectors using the build in API

- Open a browser and type **http:///localhost:8083/connectors**
- You should see the two connectors that have been configured
- Open a new tab and type **http:///localhost:8083/connectors/file-source-connector**
- You should see how the connector is configured
- Open a new tab and type **http:///localhost:8083/connectors/file-source-connector/status**
- You should see runtime information about the connector

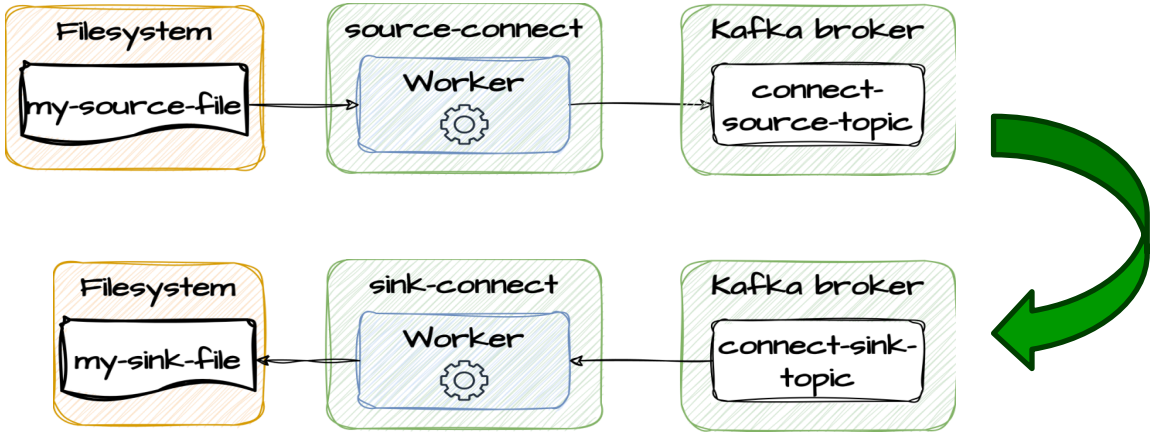
Exercise

Lund&Bendsen
developing developers

Build a kafka Streams application that reads from one topic to the other

- Create a Kafka Streams application that reads from the connect-source-topic, does some String transformation (eg reverse or upperCase) to the message and writes it to the connect-sink-topic
- Start the application
- Open the **my-source-file.txt** and type some lines of text. Save file and see if the transformed lines show up in **my-sink-file.txt**

9.5



PostgresConnector example

Lund&Bendsen
developing developers

Pg-source-config.json:

```
{
  "name": "pg-orders-source",
  "config": {
    "connector.class":   "io.debezium.connector.postgresql.PostgresConnector",
    "database.hostname": "postgres",
    "database.port": "5432",
    "database.user": "postgres",
    "database.password": "postgres",
    "database.dbname": "postgres",
    "database.server.name": "myserver",
    "plugin.name": "pgoutput",
    "table.include.list": "public.customers",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter"
  }
}
```

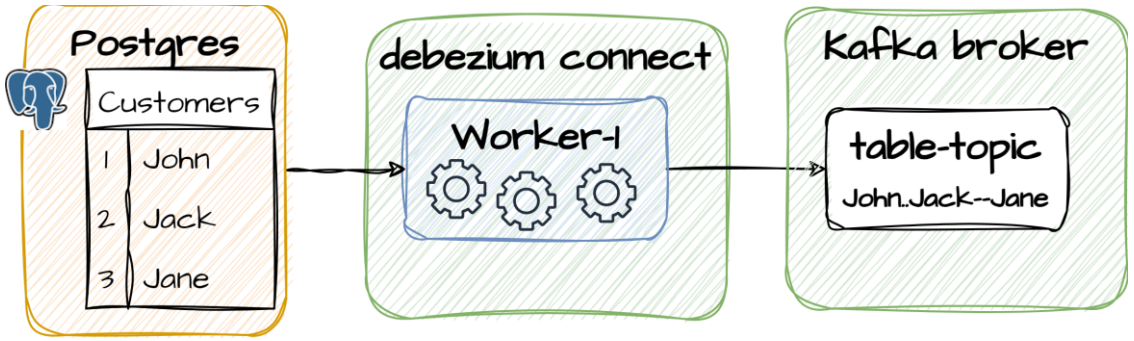
Exercise

Lund&Bendsen
developing developers

Walk through of how the CDC connector is configured

- Docker compose file: postgres and postgres-connect
- pg-source-config.json

9.6



Exercise

Lund&Bendsen
developing developers

9.7

- Configure a Postgres CDC connector
  - Look at the Docker Compose file and find the two services: postgres and postgres-connector. Understand the setup!
  - Connect to Postgres. Open PowerShell and type **docker exec -it postgres bash**
  - Login to Postgres. Type **psql -U postgres**
  - Create db-table, type: **CREATE TABLE customers (id int GENERATED ALWAYS AS IDENTITY PRIMARY KEY, name text);**
  - Alter table, type: **ALTER TABLE customers REPLICA IDENTITY USING INDEX customers_pkey;**
  - Insert data, type: **INSERT INTO customers (name) VALUES ('john'), ('jack'), ('jane');**
  - Configure debezium-connector. Open PowerShell and navigate to <project-folder>/docker/kafka-connect.
  - Open a browser and type **https://localhost:9090** to confirm the connector is running
  - In powershell, type **Invoke-WebRequest -Uri http://localhost:9090/connectors/ -Method POST -ContentType application/json -InFile .\pg-source-config.json**
  - Open akhq.io on http://localhost:8080/ to see that a new topic has been created and populated