Practical Pipelines

A houseplant alerting system with ksqlDB

dfine@confluent.io @TheDanicaFine linkedin.com/in/danica-fine/

dfine@confluent.io

@TheDanicaFine

linkedin.com/in/danica-fine/

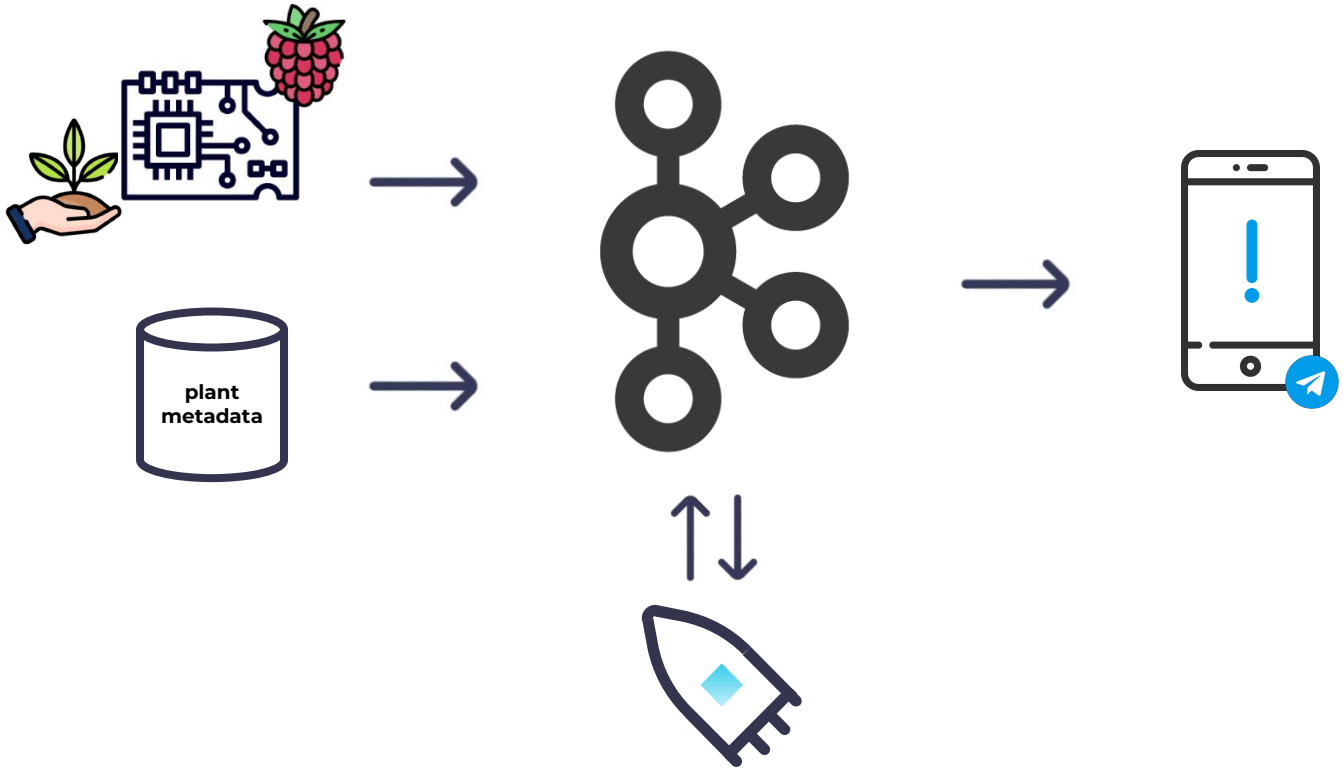
Exhibit A: Crouton the Croton

Is there a better way?

~~Is there a better way?~~

Is there a *more interesting* way?

A Practical Pipeline



Kafka? What's *that*?

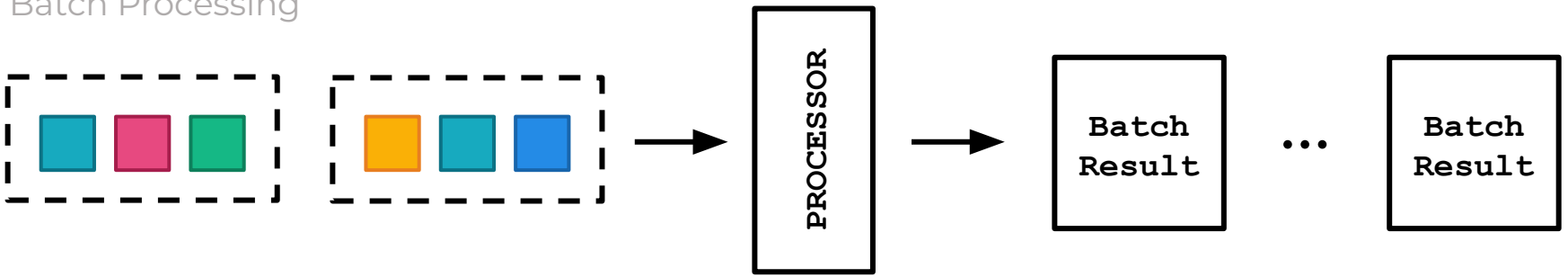
Kafka? What's *that*?

A distributed event streaming platform.

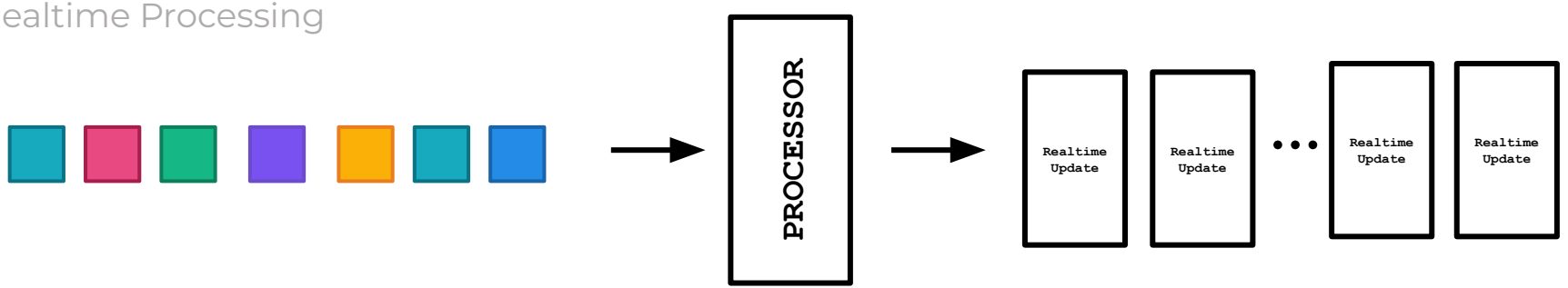
A distributed event ***streaming*** platform.

Paradigm Shift

Batch Processing



Realtime Processing



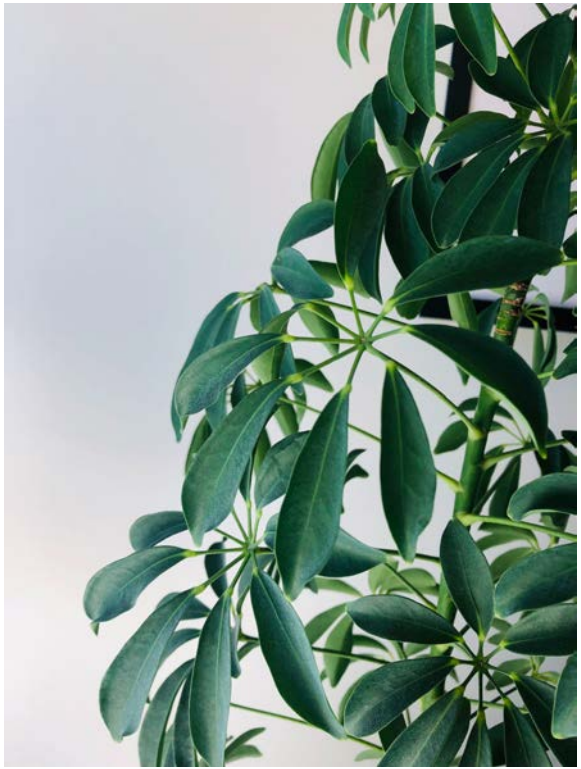
A distributed ***event*** streaming platform.

Thinking in Events

- Natural way to reason about things
- Indicate that something *has happened*
  - When
  - What/Who
- Immutable pieces of information

Thinking in Events

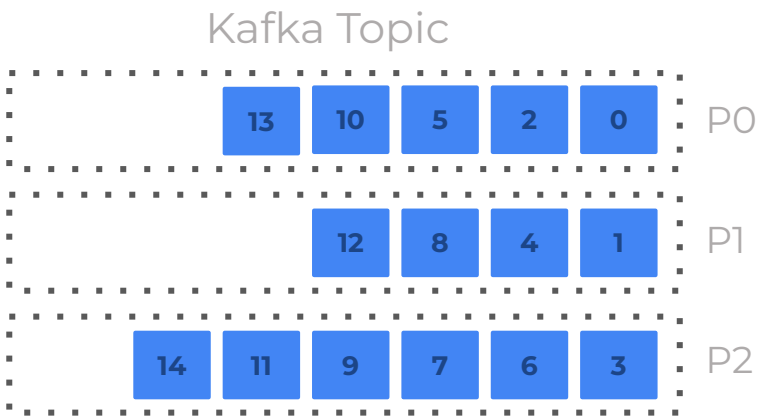
- Natural way to reason about things
- Indicate that something *has happened*
  - When
  - What/Who
- Immutable pieces of information



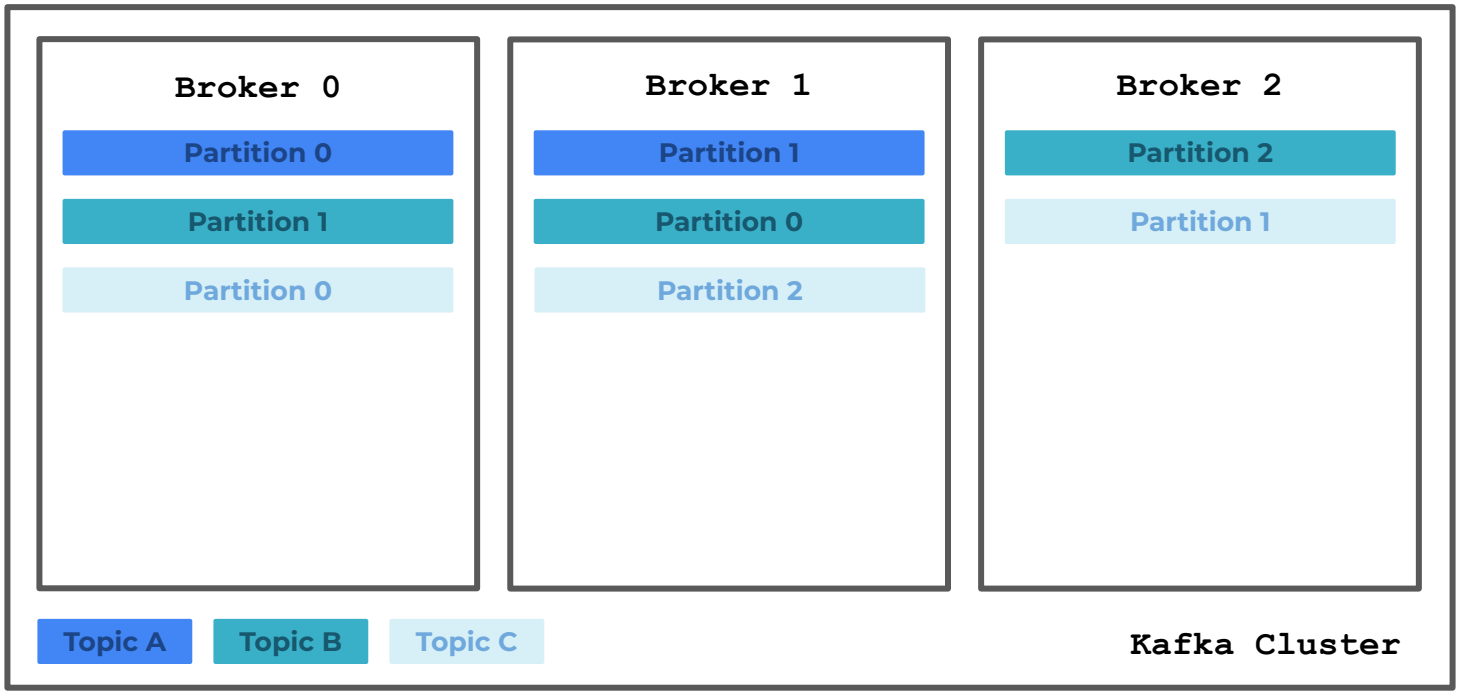
A ***distributed*** event streaming platform.

Kafka Storage

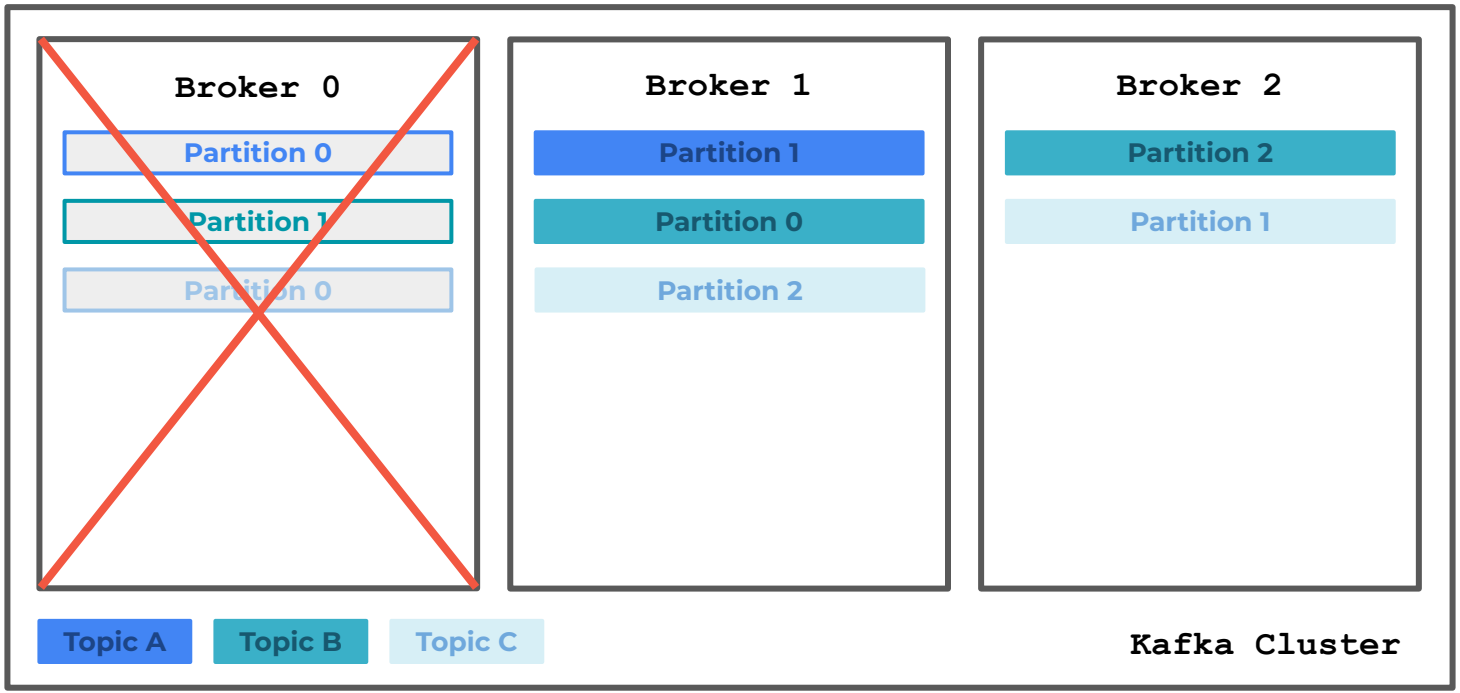
- Topics
  - Basic storage unit
  - Read/Write:
    - Producer and consumer clients
    - Completely decoupled
- Partitions
  - Immutable, append-only logs
  - Data is replicated at this level



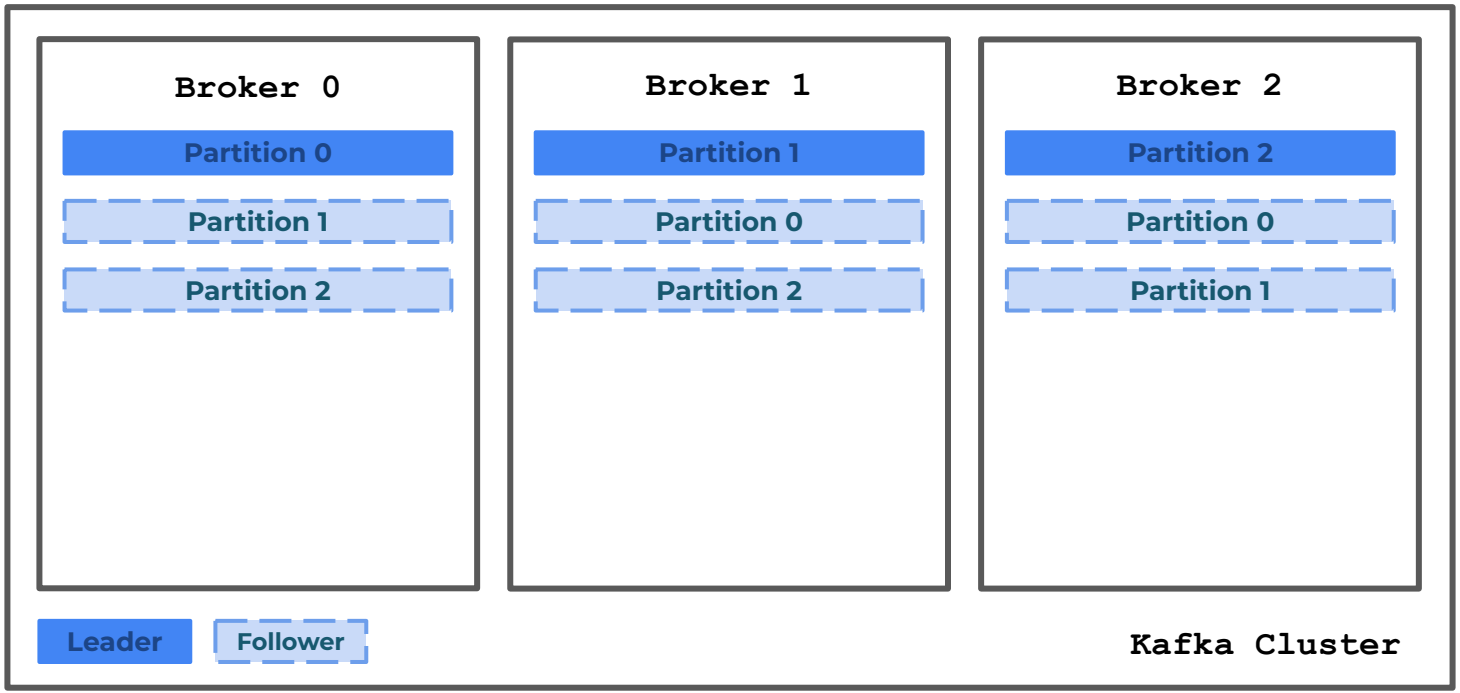
Kafka Cluster



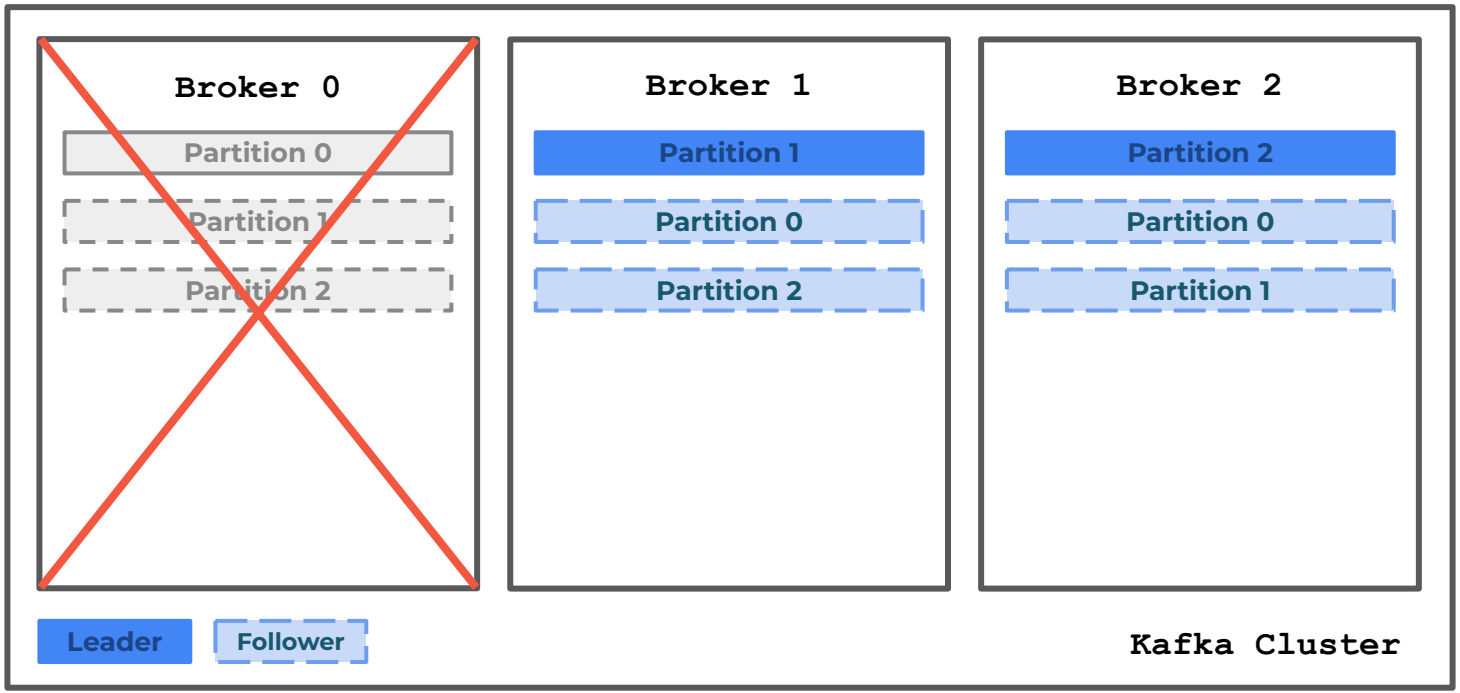
Kafka Cluster



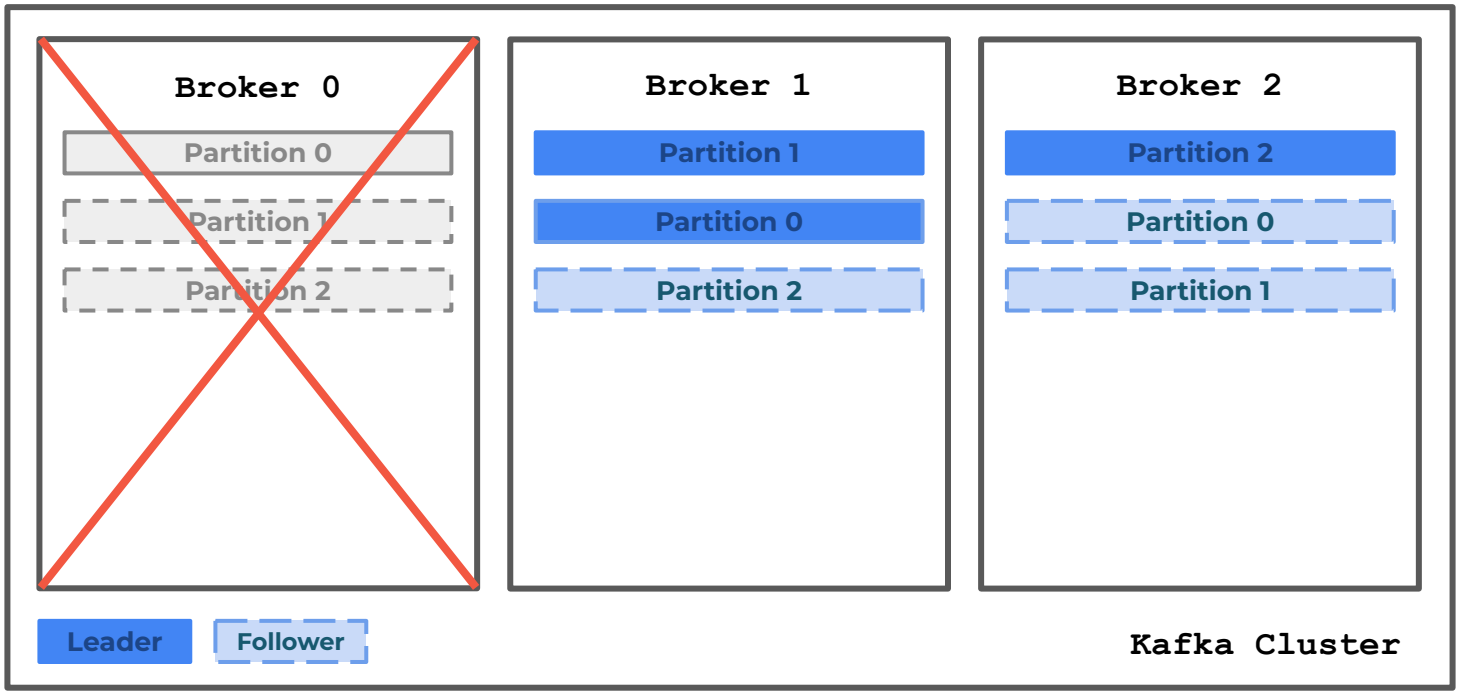
Kafka Replication



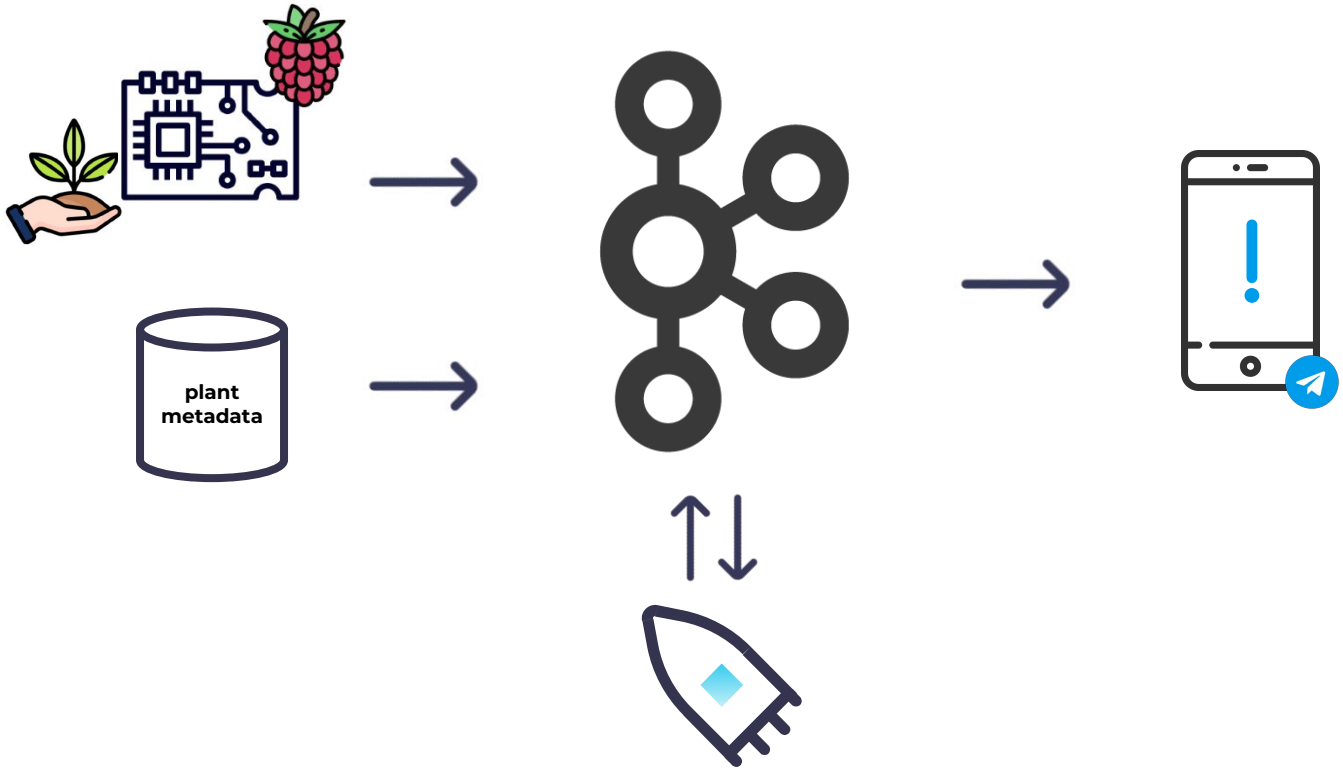
Kafka Replication



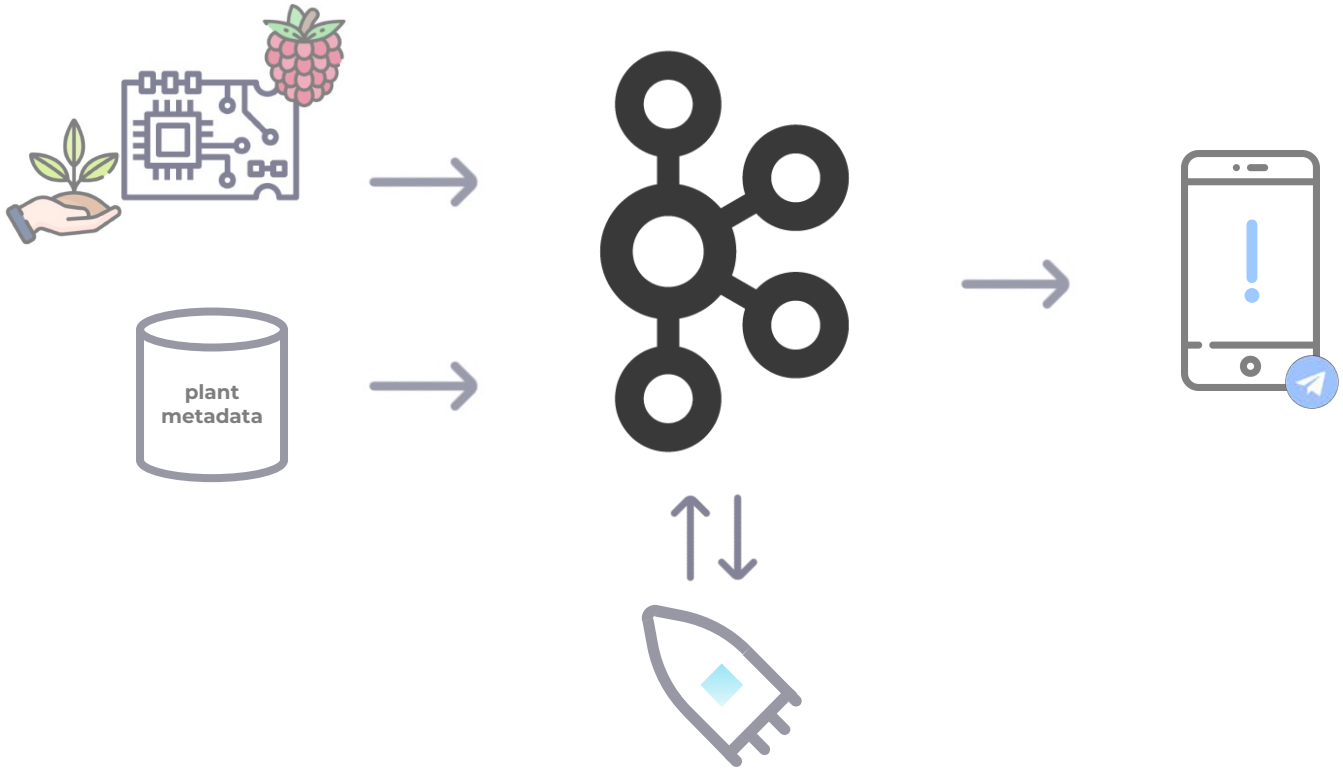
Kafka Replication



A Practical Pipeline



A Practical Pipeline: Cluster Creation

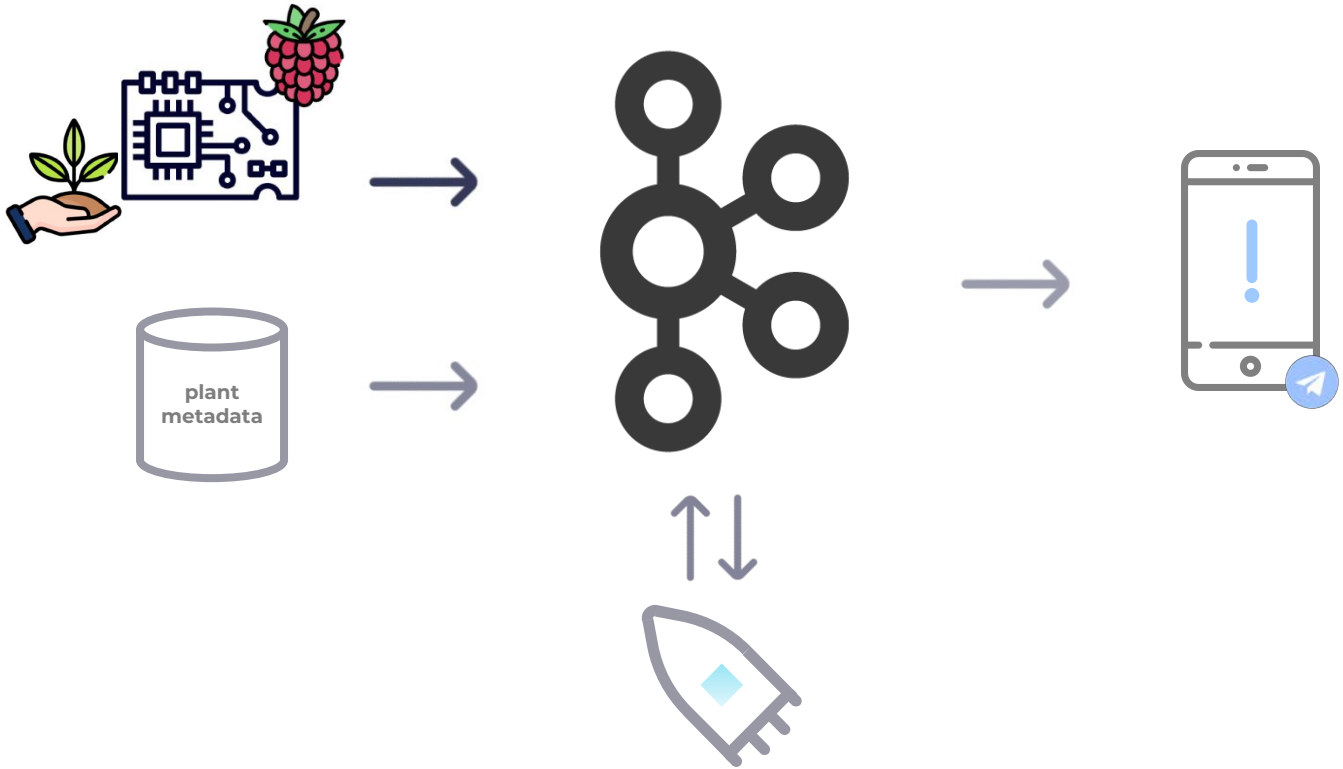


Kafka, but make it simple.

- Fully-managed, cloud-based Kafka
- Auxiliary tools:
  - Kafka Connect
  - ksqlDB
  - Schema management

CONFLUENT

A Practical Pipeline: The Raspberry Pi



The Hardware

- Raspberry Pi 4 Model B 8GB
- Half-Sized Breadboard
- Adafruit STEMMA Soil I2C Capacitive Moisture Sensors
- A lot of cables

Know Your Data

```
{
  "doc": "Houseplant reading taken from meters.",
  "fields": [
    {
      "doc": "Unique plant identification number.",
      "name": "plant_id",
      "type": "int"
    },
    {
      "doc": "Soil moisture as a percentage.",
      "name": "moisture",
      "type": "float"
    },
    {
      "doc": "Temperature in degrees C of the soil of this plant.",
      "name": "temperature",
      "type": "float"
    }
  ],
  "name": "reading",
  "type": "record"
}
```

Producing Data to Kafka

Producer API or Kafka Connect

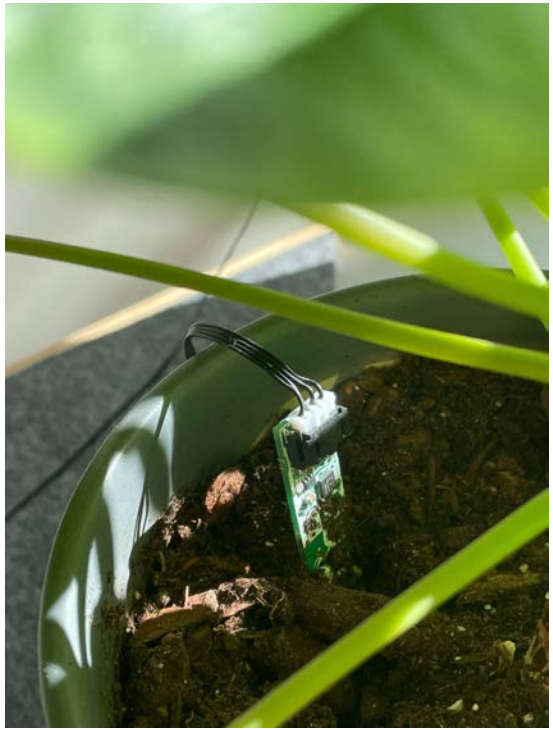
dfine@confluent.io @TheDanicaFine linkedin.com/in/danica-fine/

Writing Data to Kafka

- Kafka Producer API
  - All of your favorite languages
  - Great when you own the application producing the data
- Kafka Connect
  - Low- to no-code option
  - Hundreds of data sources (and sinks)
  - Many fully-managed through Confluent Cloud
  - Great for integrating with data *at rest*

Capturing Houseplant Readings

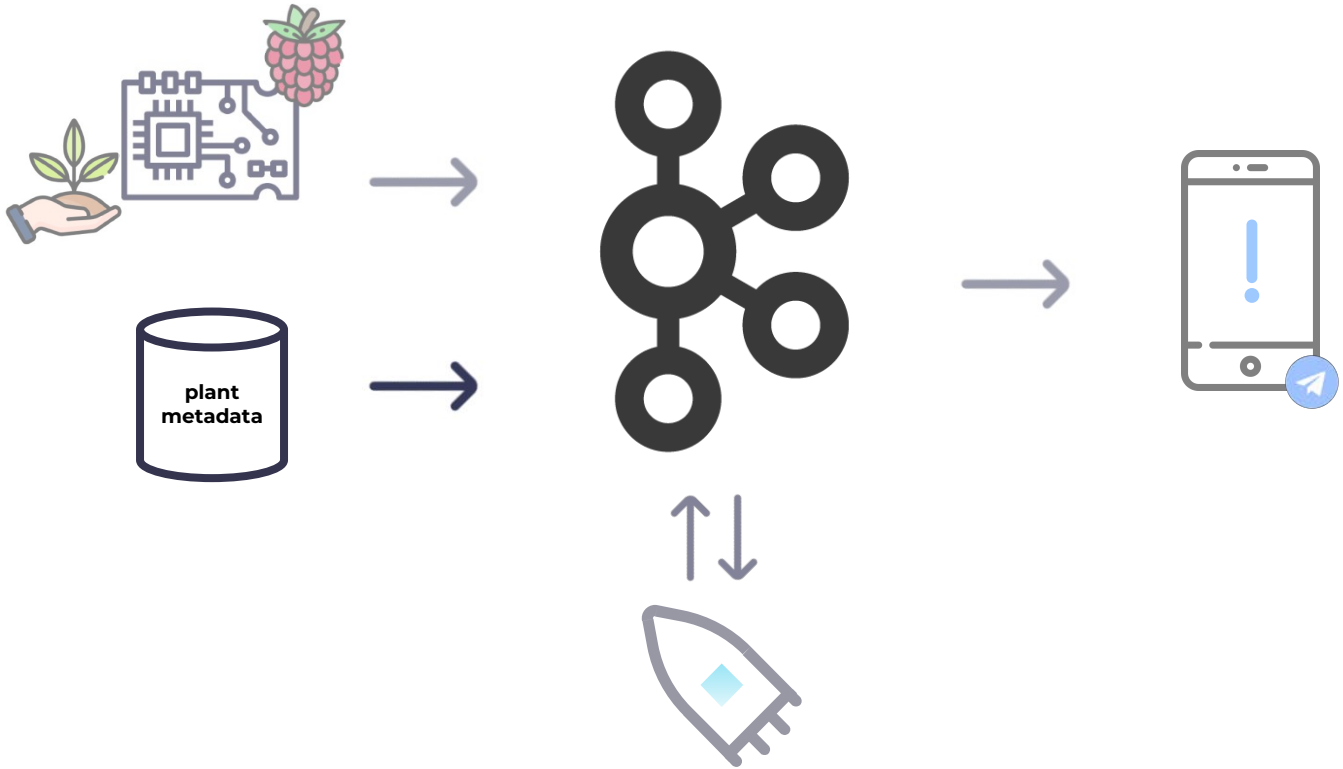
- `confluent_kafka` Python library
- The Process:
  - Predefined I^2C addresses `0x36` … `0x39`
  - Capture sensor readings every 30 seconds
  - Serializing Kafka Producer writes readings data into Kafka



Capturing Houseplant Readings

```
for address,plant_id in plant_addresses.items():
      # connect to sensor
      ss = Seesaw(i2c_bus, addr=int(address, 16))

      # read moisture, compute percentage
      touch = ss.moisture_read()
      touch_percent = (touch - sensor_values) / (sensor_values - sensor_values) * 100

      # read temperature
      temp = ss.get_temp()

      # send data to Kafka
      reading = Reading(int(plant_id), round(touch_percent, 3), round(temp, 3))
      producer.produce(readings_topic, key=str(plant_id), value=reading)
```

A Practical Pipeline: Capturing Metadata



Know Your Plants

```
{
  "doc": "Houseplant metadata.",
  "fields": [
    {
      "doc": "Unique plant identification number.",
      "name": "plant_id",
      "type": "int"
    },
    {
      "doc": "Scientific name of the plant.",
      "name": "scientific_name",
      "type": "string"
    },
    {
      "doc": "The common name of the plant.",
      "name": "common_name",
      "type": "string"
    },
    {
      "doc": "The given name of the plant.",
      "name": "given_name",
      "type": "string"
    },
    {
      "doc": "Lowest temperature of the plant.",
      "name": "temperature_low",
      "type": "float"
    },
    {
      "doc": "Highest temperature of the plant.",
      "name": "temperature_high",
      "type": "float"
    },
    {
      "doc": "Lowest moisture of the plant.",
      "name": "moisture_low",
      "type": "float"
    },
    {
      "doc": "Highest moisture of the plant.",
      "name": "moisture_high",
      "type": "float"
    }
  ],
  "name": "houseplant",
  "type": "record"
}
```

Producing Data to Kafka

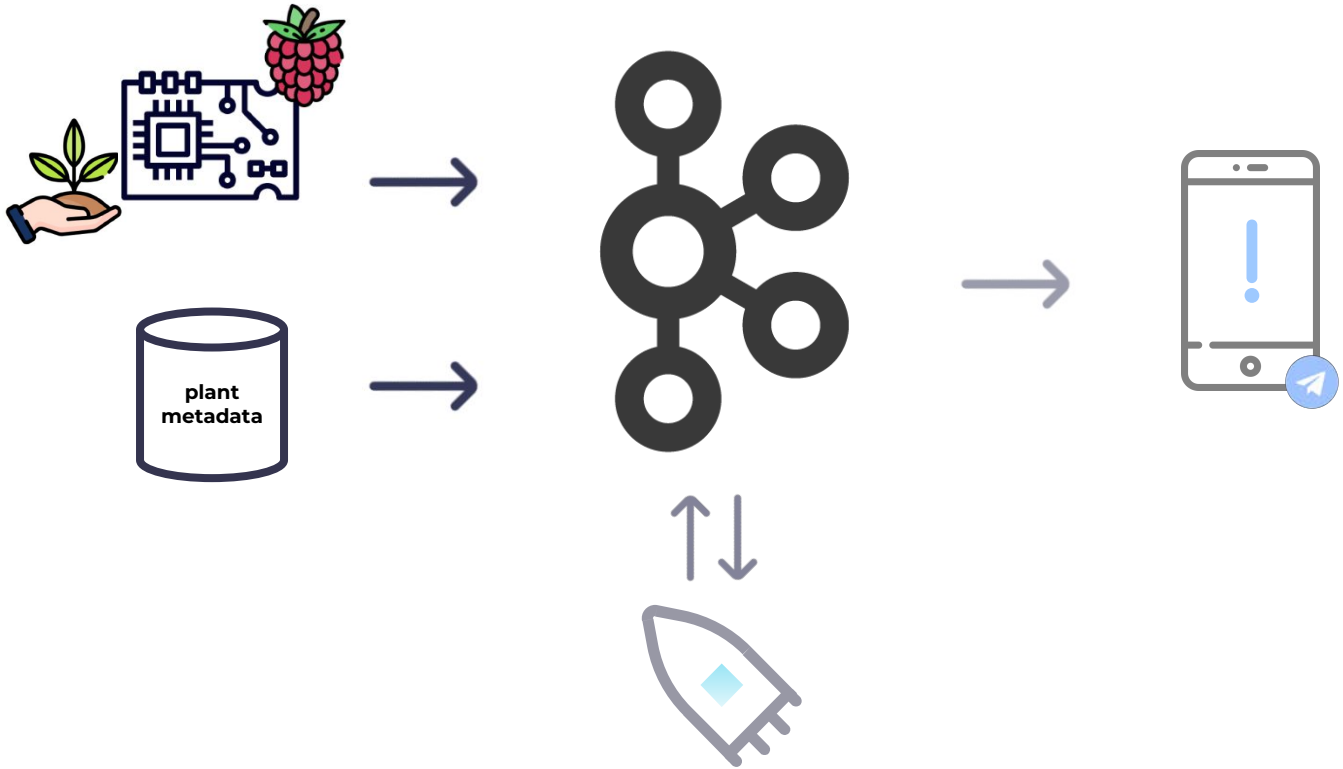
Producer API or Kafka Connect

dfine@confluent.io @TheDanicaFine linkedin.com/in/danica-fine/

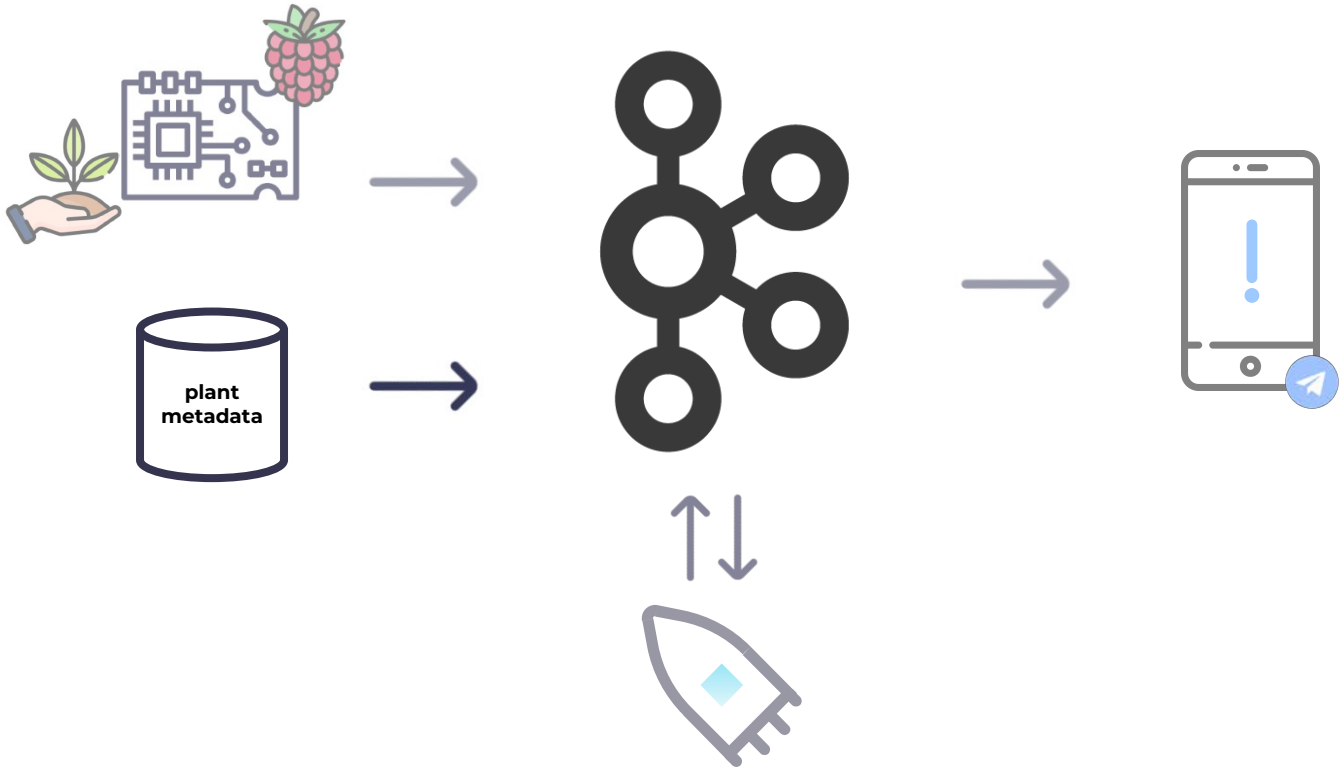
Capturing Houseplant Metadata

- `confluent_kafka` Python library
- The Process
  - Manually enter metadata entries
  - Serialize according to AVRO schema
  - Kafka Producer to write into Kafka

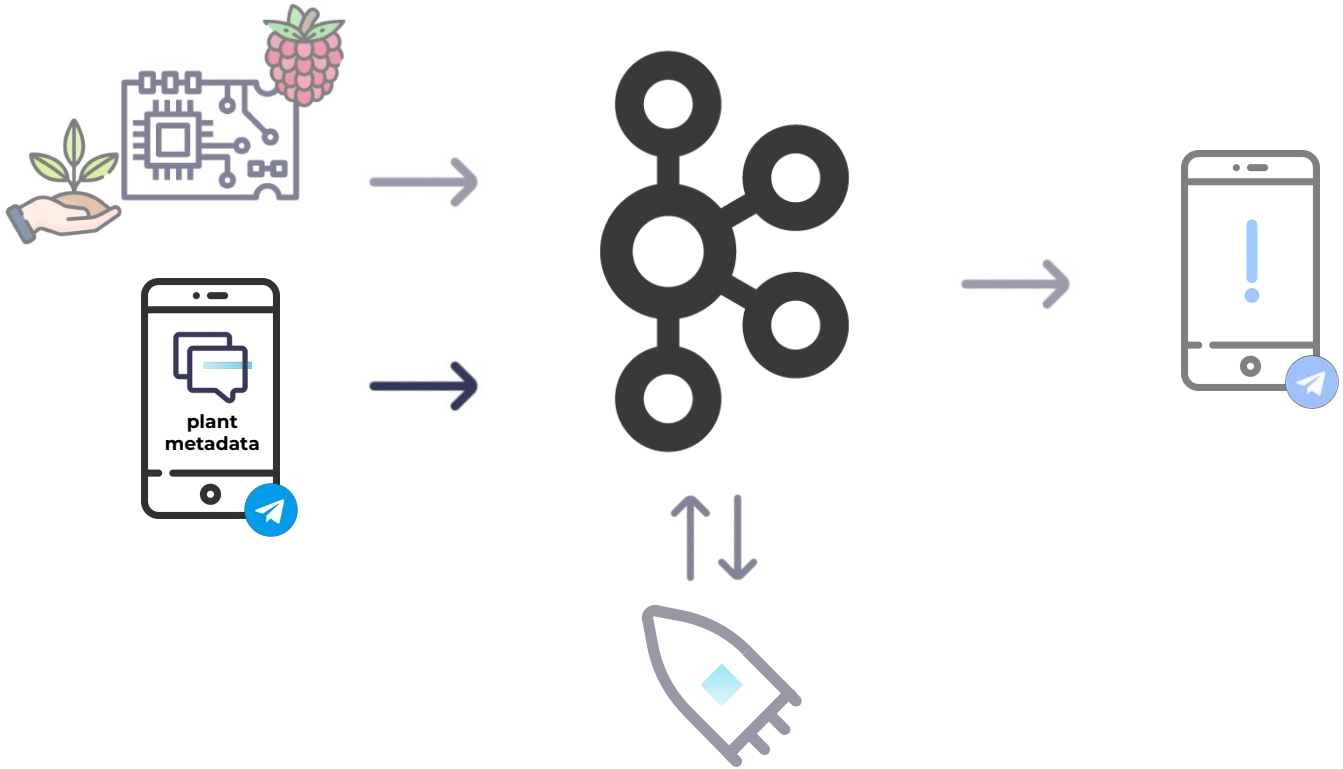
A Practical Pipeline... but more event-driven



A Practical Pipeline… but more event-driven

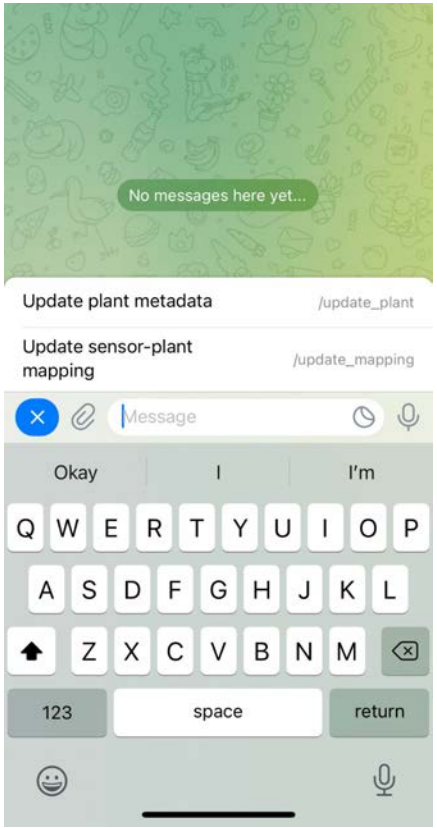


A Practical Pipeline... but more event-driven



Telegram as a Producer

- `python-telegram-bot` library
  - Wrapper for Telegram API
  - Define conversation handlers
- Produce data to Kafka
  - Update plant metadata
  - Update plant-sensor mappings



Conversation Handler

```
update_plant_handler = ConversationHandler(
        entry_points = [CommandHandler('update_plant', update_plant_command)],
        states = {
            PLANT_STATE.ID: [
                MessageHandler(filters.TEXT & ~filters.COMMAND, id_command),
                CommandHandler('cancel', cancel_command)
            ],
            ...
            PLANT_STATE.MOISTURE_HIGH: [
                MessageHandler(filters.TEXT & (~filters.COMMAND | filters.Regex("^\/skip$")), moisture_high_command),
                CommandHandler('cancel', cancel_command)
            ],
            PLANT_STATE.COMMIT_PLANT: [
                CommandHandler('y', commit_plant_command),
                CommandHandler('n', cancel_command)
            ]
        },
        fallbacks = [CommandHandler('cancel', cancel_command)]
    )
```

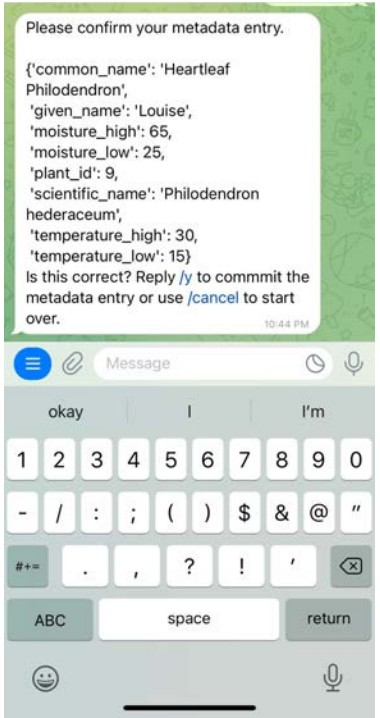
Command Handler

```python
async def moisture_low_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    # store low moisture data
    moisture_low = update.message.text
    if moisture_low != '/skip':
        # update state
        context.user_data['plant']['moisture_low'] = int(moisture_low)
    else:
        context.user_data['plant']['moisture_low'] = 25

    # prompt for high moisture data
    await update.message.reply_text(
            "Please enter plant's high moisture threshold or /skip to use the default."
         )

    return PLANT_STATE.MOISTURE_HIGH
```

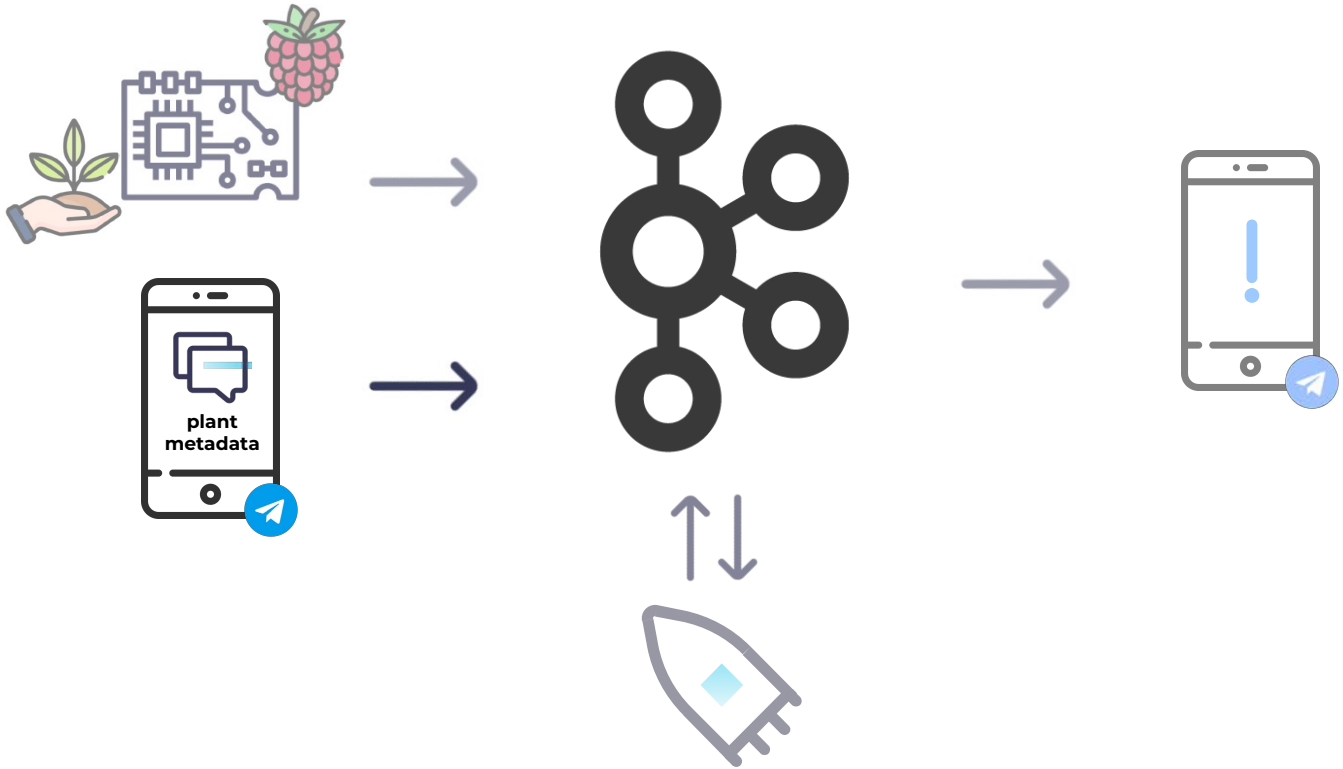
Producer Code

```
def send_metadata(metadata):
    # send houseplant metadata message
    try:
        # set up Kafka producer for houseplant metadata
        producer = clients.producer(clients.houseplant_serializer())

        # prep key and value for message
        k = str(metadata.get('plant_id'))
        value = Houseplant.dict_to_houseplant(metadata)

        logger.info("Publishing houseplant metadata message for key %s", k)
        producer.produce(config['topics']['houseplants'], key=k, value=value)
    except Exception as e:
        logger.error("Got exception %s", e)
        raise e
    finally:
        producer.poll()
        producer.flush()
```

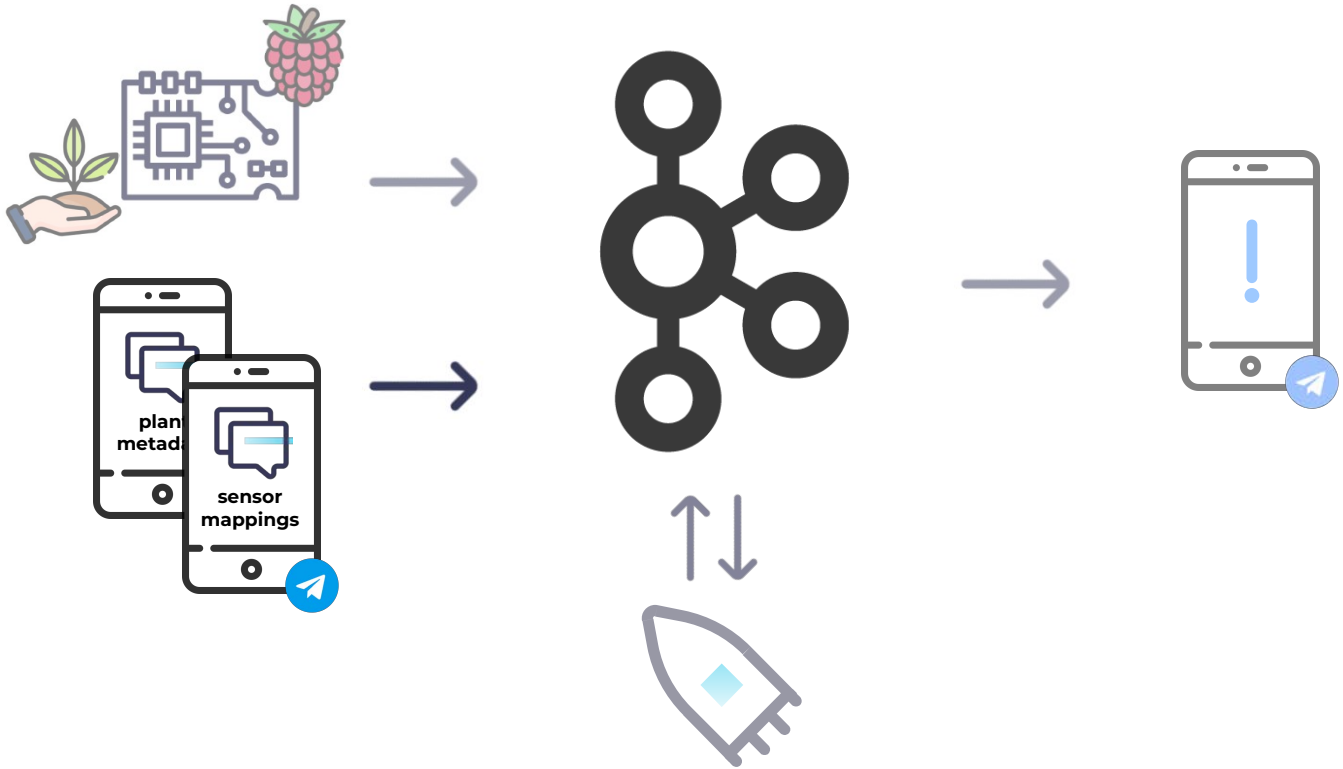
Updating Plant Data



A Practical Pipeline... but more event-driven



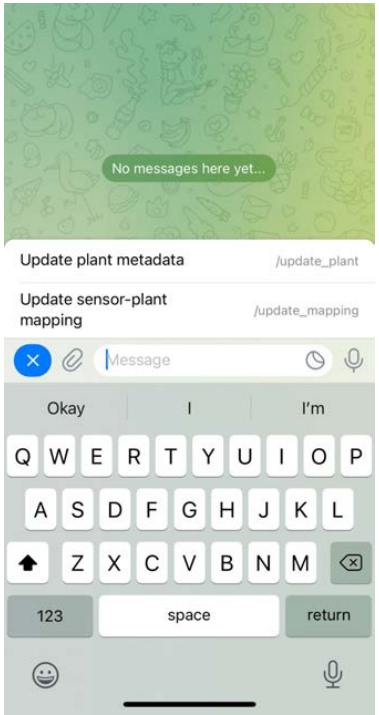
A Practical Pipeline… but more event-driven



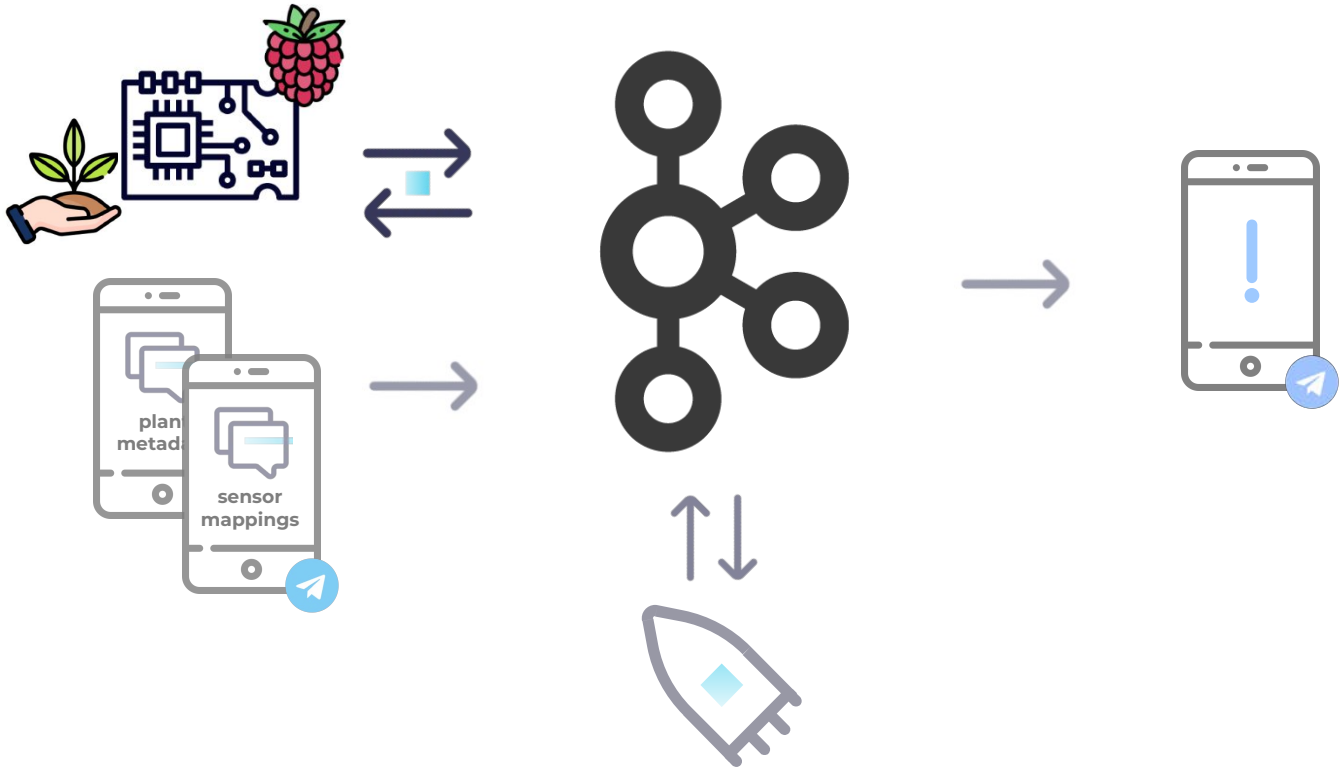
New Schema Unlocked!

```
{
  "doc": "Sensor-houseplant mapping.",
  "fields": [
    {
      "doc": "Hardcoded ID of the physical soil sensor.",
      "name": "sensor_id",
      "type": "string"
    },
    {
      "doc": "Plant identification number.",
      "name": "plant_id",
      "type": "int"
    }
  ],
  "name": "mapping",
  "namespace": "com.houseplants",
  "type": "record"
}
```

Updating Sensor Mappings



A Practical Pipeline… but more event-driven



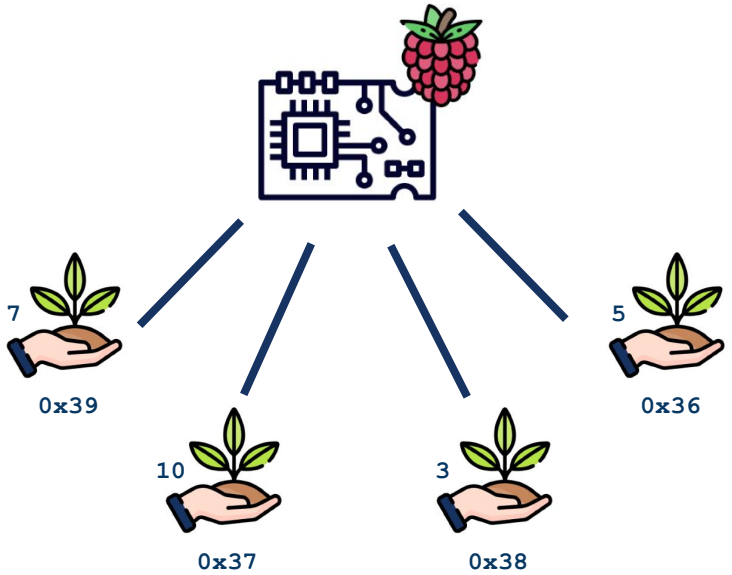
Consuming Data from Kafka

Consumer API or Kafka Connect

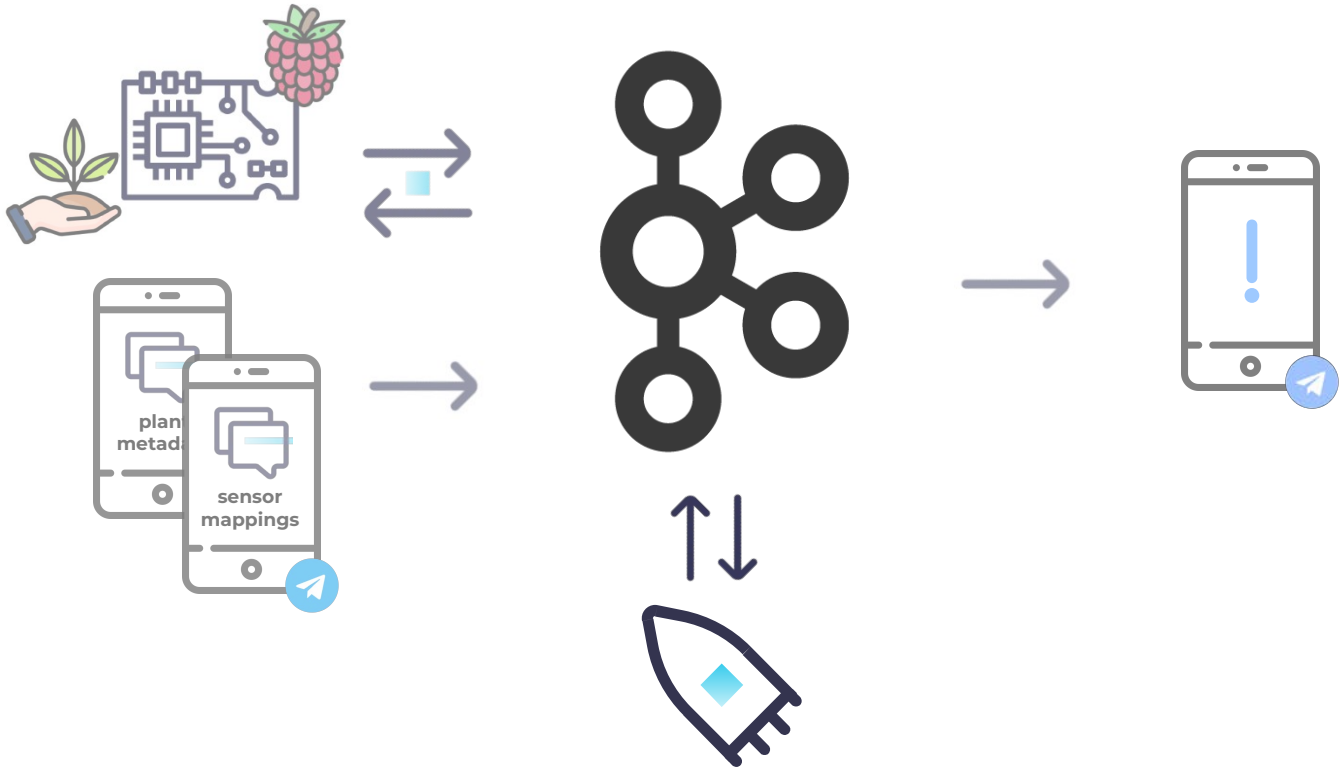
dfine@confluent.io @TheDanicaFine linkedin.com/in/danica-fine/

Consuming Mapping Updates

- Compact `mapping` topic
- Loop
  - Consume new mappings
  - Use mappings in producer
- Non-committing consumer
  - `'auto.offset.reset': 'earliest'`
  - `'enable.auto.commit': 'false'`



A Practical Pipeline: Stream Processing



Processing with Kafka

- Consumer/Producer API
  - Lowest level
  - Consume – Process – Produce
  - Manually define state/fault-tolerance
- Kafka Streams
  - Java library for stream processing
  - Built-in state handling and failover
- ksqlDB
  - SQL syntax
  - Kafka Streams under the hood
  - Cloud based offering

```
subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

```
KStream, KTable,
filter(), map(), flatMap(),
join(), aggregate(), …
```

```
CREATE STREAM, CREATE TABLE,
SELECT, JOIN, GROUP BY, SUM, …
```

Processing with ksqlDB

Creating a Table

```
CREATE TABLE houseplant_metadata (
  id STRING PRIMARY KEY
) WITH (
  kafka_topic='houseplant-metadata',
  value_format='AVRO'
);
```

Creating a Table

```
CREATE TABLE houseplant_metadata (
  id STRING PRIMARY KEY
) WITH (
  kafka_topic='houseplant-metadata',
  value_format='AVRO'
);
```

Creating a Table

```
CREATE TABLE houseplant_metadata (
  id STRING PRIMARY KEY
) WITH (
  kafka_topic='houseplant-metadata',
  value_format='AVRO'
);
```

plant_analysis

Editor | Flow | Streams | Tables | Persistent queries | Performance | Settings | CLI instructions

```
1  CREATE TABLE houseplant_metadata (
2    id STRING PRIMARY KEY
3  ) WITH (
4    kafka_topic='houseplant-metadata',
5    value_format='AVRO'
6  );
```

Add query properties

auto.offset.reset = Latest

+Add another field

Stop | Run query

Creating a Stream

```
CREATE STREAM houseplant_readings (
  id STRING KEY
) WITH (
  kafka_topic='houseplant-readings',
  value_format='AVRO'
);
```

Stream Enrichment

```
CREATE STREAM houseplant_readings_enriched WITH (
  kafka_topic='houseplant-readings-enriched',
  value_format='AVRO'
) AS
SELECT
  houseplant_readings.id                AS plant_id,
  houseplant_readings.ROWTIME           AS ts,
  houseplant_readings.moisture          AS moisture,
  houseplant_readings.temperature       AS temperature,
  houseplant_metadata.scientific_name   AS scientific_name,
  houseplant_metadata.common_name       AS common_name,
  houseplant_metadata.given_name        AS given_name,
  houseplant_metadata.temperature_low   AS temperature_low,
  houseplant_metadata.temperature_high  AS temperature_high,
  houseplant_metadata.moisture_low      AS moisture_low,
  houseplant_metadata.moisture_high     AS moisture_high
FROM houseplant_readings
INNER JOIN houseplant_metadata
ON houseplant_readings.id = houseplant_metadata.id
EMIT CHANGES;
```

Stream Enrichment

```
CREATE STREAM houseplant_readings_enriched WITH (
  kafka_topic='houseplant-readings-enriched',
  value_format='AVRO'
) AS
SELECT
  houseplant_readings.id                AS plant_id,
  houseplant_readings.ROWTIME           AS ts,
  houseplant_readings.moisture          AS moisture,
  houseplant_readings.temperature       AS temperature,
  houseplant_metadata.scientific_name   AS scientific_name,
  houseplant_metadata.common_name       AS common_name,
  houseplant_metadata.given_name        AS given_name,
  houseplant_metadata.temperature_low   AS temperature_low,
  houseplant_metadata.temperature_high  AS temperature_high,
  houseplant_metadata.moisture_low      AS moisture_low,
  houseplant_metadata.moisture_high     AS moisture_high
FROM houseplant_readings
INNER JOIN houseplant_metadata
ON houseplant_readings.id = houseplant_metadata.id
EMIT CHANGES;
```

Stream Enrichment

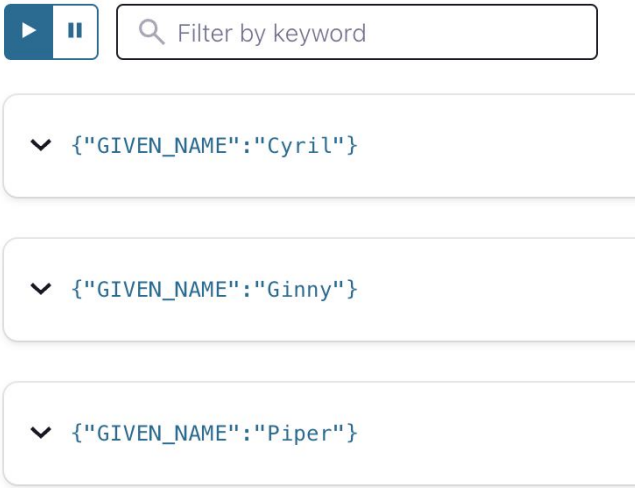
```
CREATE STREAM houseplant_readings_enriched WITH (
  kafka_topic='houseplant-readings-enriched',
  value_format='AVRO'
) AS
SELECT
  houseplant_readings.id                AS plant_id,
  houseplant_readings.ROWTIME           AS ts,
  houseplant_readings.moisture          AS moisture,
  houseplant_readings.temperature       AS temperature,
  houseplant_metadata.scientific_name   AS scientific_name,
  houseplant_metadata.common_name       AS common_name,
  houseplant_metadata.given_name        AS given_name,
  houseplant_metadata.temperature_low   AS temperature_low,
  houseplant_metadata.temperature_high  AS temperature_high,
  houseplant_metadata.moisture_low      AS moisture_low,
  houseplant_metadata.moisture_high     AS moisture_high
FROM houseplant_readings
INNER JOIN houseplant_metadata
ON houseplant_readings.id = houseplant_metadata.id
EMIT CHANGES;
```

Stream Enrichment

```
CREATE STREAM houseplant_readings_enriched WITH (
  kafka_topic='houseplant-readings-enriched',
  value_format='AVRO'
) AS
SELECT
  houseplant_readings.id                AS plant_id,
  houseplant_readings.ROWTIME           AS ts,
  houseplant_readings.moisture          AS moisture,
  houseplant_readings.temperature       AS temperature,
  houseplant_metadata.scientific_name   AS scientific_name,
  houseplant_metadata.common_name       AS common_name,
  houseplant_metadata.given_name        AS given_name,
  houseplant_metadata.temperature_low   AS temperature_low,
  houseplant_metadata.temperature_high  AS temperature_high,
  houseplant_metadata.moisture_low      AS moisture_low,
  houseplant_metadata.moisture_high     AS moisture_high
FROM houseplant_readings
INNER JOIN houseplant_metadata
ON houseplant_readings.id = houseplant_metadata.id
EMIT CHANGES;
```

Pull Query

```
SELECT
  given_name
FROM houseplants
WHERE low_moisture > 20;
```



Acting on the data

Windowing and Computing Alerts

```
CREATE TABLE houseplant_low_readings WITH (
  kafka_topic='houseplant-low-readings',
  format='AVRO'
) AS
SELECT
  plant_id,
  scientific_name,
  common_name,
  given_name,
  moisture_low,
  CONCAT(given_name, ' the ', common_name, ' (', scientific_name, ') is looking pretty dry...') AS message,
  COUNT(*) AS low_reading_count
FROM houseplant_readings_enriched
WINDOW TUMBLING (SIZE 6 HOURS, RETENTION 7 DAYS, GRACE PERIOD 10 MINUTES)
WHERE moisture < moisture_low
GROUP BY plant_id, scientific_name, common_name, given_name, moisture_low
HAVING COUNT(*) > 120
EMIT FINAL;
```

Windowing and Computing Alerts

```
CREATE TABLE houseplant_low_readings WITH (
  kafka_topic='houseplant-low-readings',
  format='AVRO'
) AS
SELECT
  plant_id,
  scientific_name,
  common_name,
  given_name,
  moisture_low,
  CONCAT(given_name, ' the ', common_name, ' (', scientific_name, ') is looking pretty dry...') AS message,
  COUNT(*) AS low_reading_count
FROM houseplant_readings_enriched
WINDOW TUMBLING (SIZE 6 HOURS, RETENTION 7 DAYS, GRACE PERIOD 10 MINUTES)
WHERE moisture < moisture_low
GROUP BY plant_id, scientific_name, common_name, given_name, moisture_low
HAVING COUNT(*) > 120
EMIT FINAL;
```

Windowing and Computing Alerts

```
CREATE TABLE houseplant_low_readings WITH (
  kafka_topic='houseplant-low-readings',
  format='AVRO'
) AS
SELECT
  plant_id,
  scientific_name,
  common_name,
  given_name,
  moisture_low,
  CONCAT(given_name, ' the ', common_name, ' (', scientific_name, ') is looking pretty dry...') AS message,
  COUNT(*) AS low_reading_count
FROM houseplant_readings_enriched
WINDOW TUMBLING (SIZE 6 HOURS, RETENTION 7 DAYS, GRACE PERIOD 10 MINUTES)
WHERE moisture < moisture_low
GROUP BY plant_id, scientific_name, common_name, given_name, moisture_low
HAVING COUNT(*) > 120
EMIT FINAL;
```

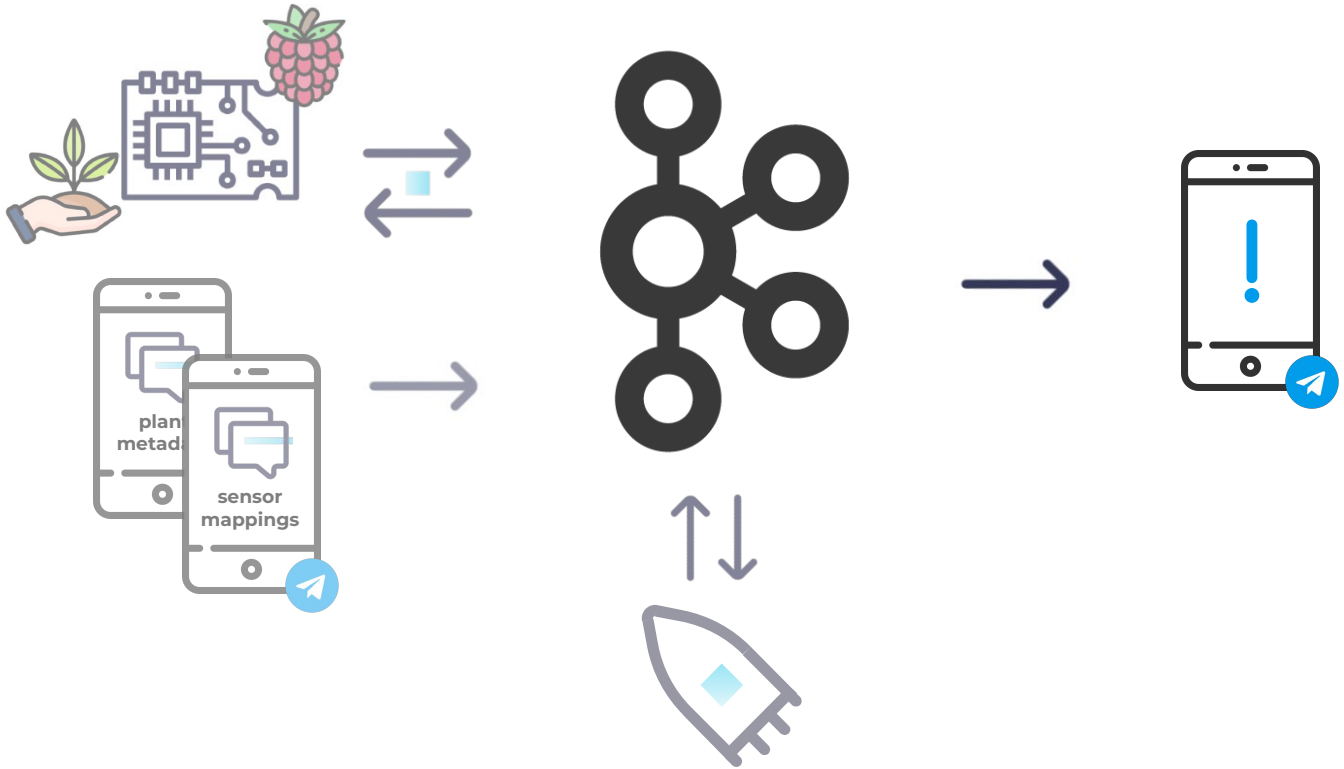
Windowing and Computing Alerts

```
CREATE TABLE houseplant_low_readings WITH (
  kafka_topic='houseplant-low-readings',
  format='AVRO'
) AS
SELECT
  plant_id,
  scientific_name,
  common_name,
  given_name,
  moisture_low,
  CONCAT(given_name, ' the ', common_name, ' (', scientific_name, ') is looking pretty dry...') AS message,
  COUNT(*) AS low_reading_count
FROM houseplant_readings_enriched
WINDOW TUMBLING (SIZE 6 HOURS, RETENTION 7 DAYS, GRACE PERIOD 10 MINUTES)
WHERE moisture < moisture_low
GROUP BY plant_id, scientific_name, common_name, given_name, moisture_low
HAVING COUNT(*) > 120
EMIT FINAL;
```

Windowing and Computing Alerts

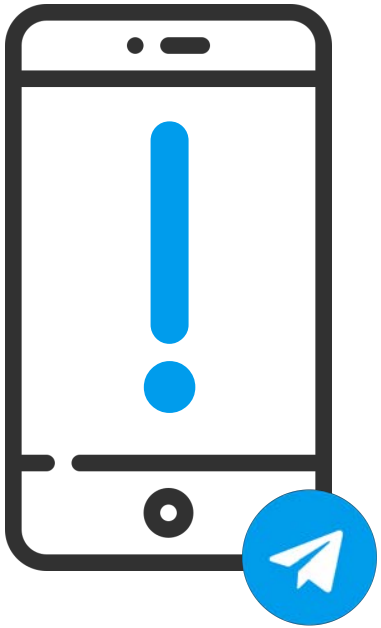
```
CREATE TABLE houseplant_low_readings WITH (
  kafka_topic='houseplant-low-readings',
  format='AVRO'
) AS
SELECT
  plant_id,
  scientific_name,
  common_name,
  given_name,
  moisture_low,
  CONCAT(given_name, ' the ', common_name, ' (', scientific_name, ') is looking pretty dry...') AS message,
  COUNT(*) AS low_reading_count
FROM houseplant_readings_enriched
WINDOW TUMBLING (SIZE 6 HOURS, RETENTION 7 DAYS, GRACE PERIOD 10 MINUTES)
WHERE moisture < moisture_low
GROUP BY plant_id, scientific_name, common_name, given_name, moisture_low
HAVING COUNT(*) > 120
EMIT FINAL;
```

A Practical Pipeline: Alerting



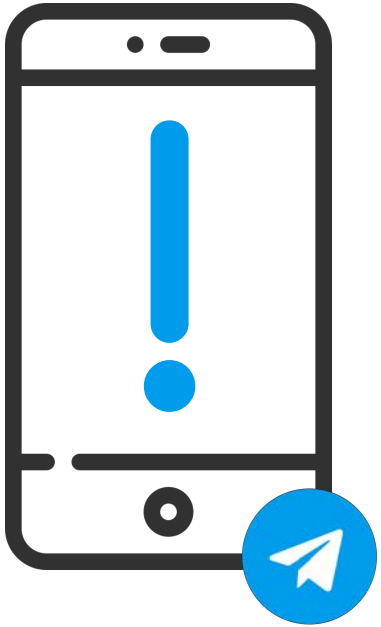
Alerting with Telegram

- Simple messaging bot
- Kafka Connect HTTP Sink Connector
  - Fully-managed in Confluent Cloud
  - Configuration-based

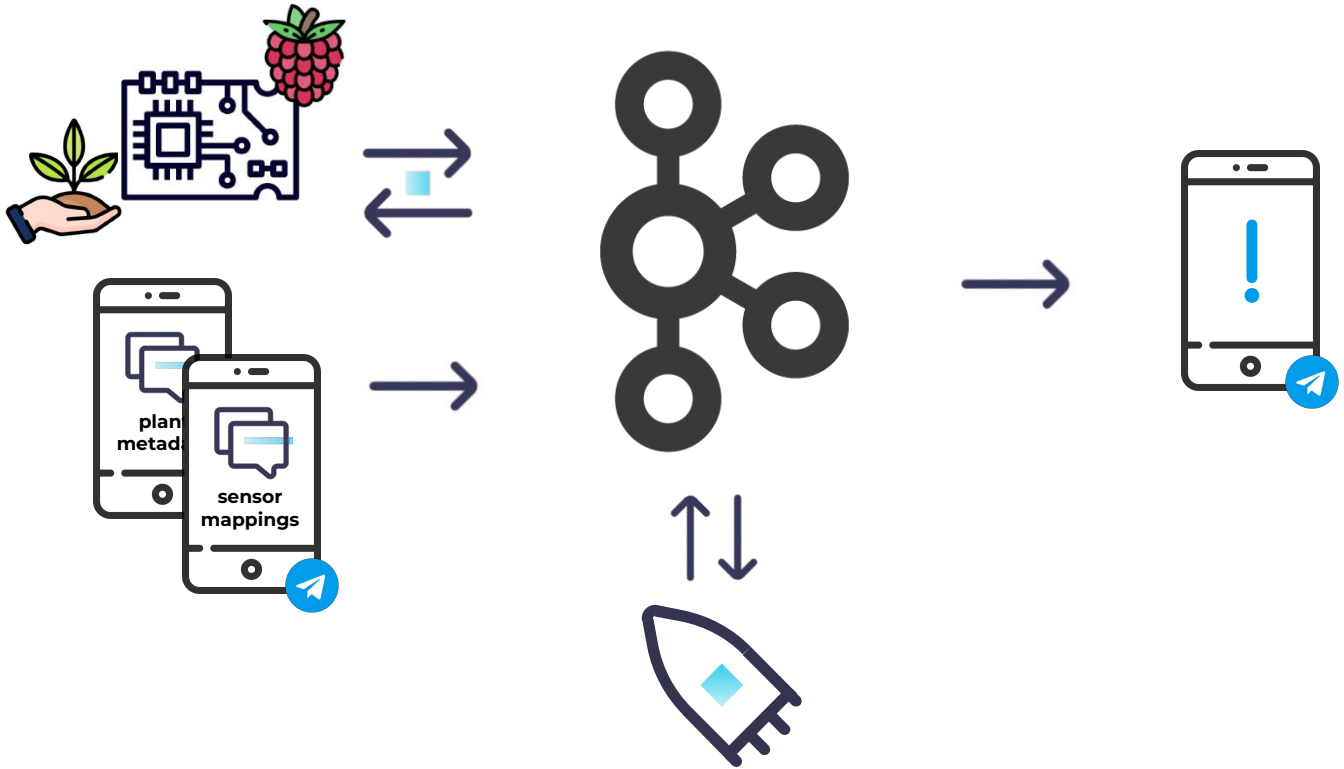


Alerting with Telegram

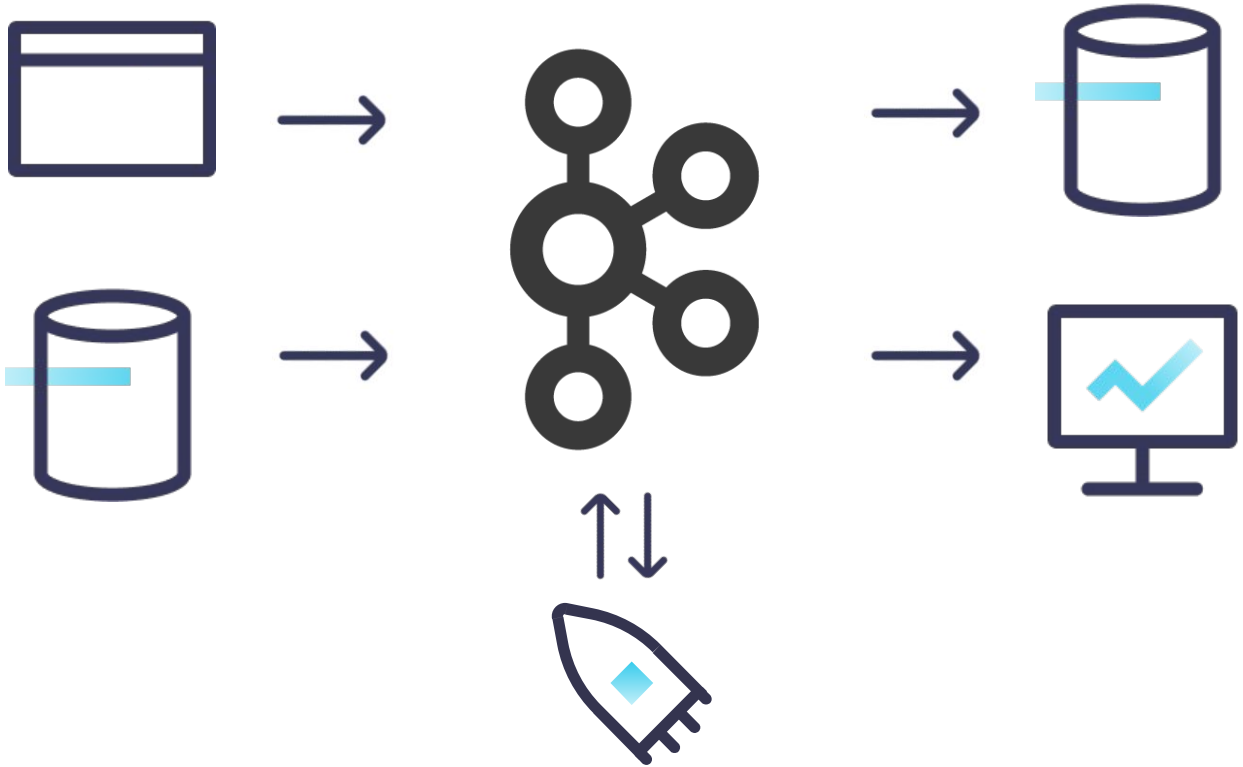
- Simple messaging bot
- Kafka Connect HTTP Sink Connector
  - Fully-managed in the Cloud
  - Configuration-based



A Practical Pipeline



Streaming Data Pipeline



Plant the seeds—get started!

LinkTree Resources

Questions?