NETCONOMY

# Stateful & Reactive Stream Processing Applications without a Database

## Apache Kafka Streams ❤️ Spring Boot 2.0

# $ whoami

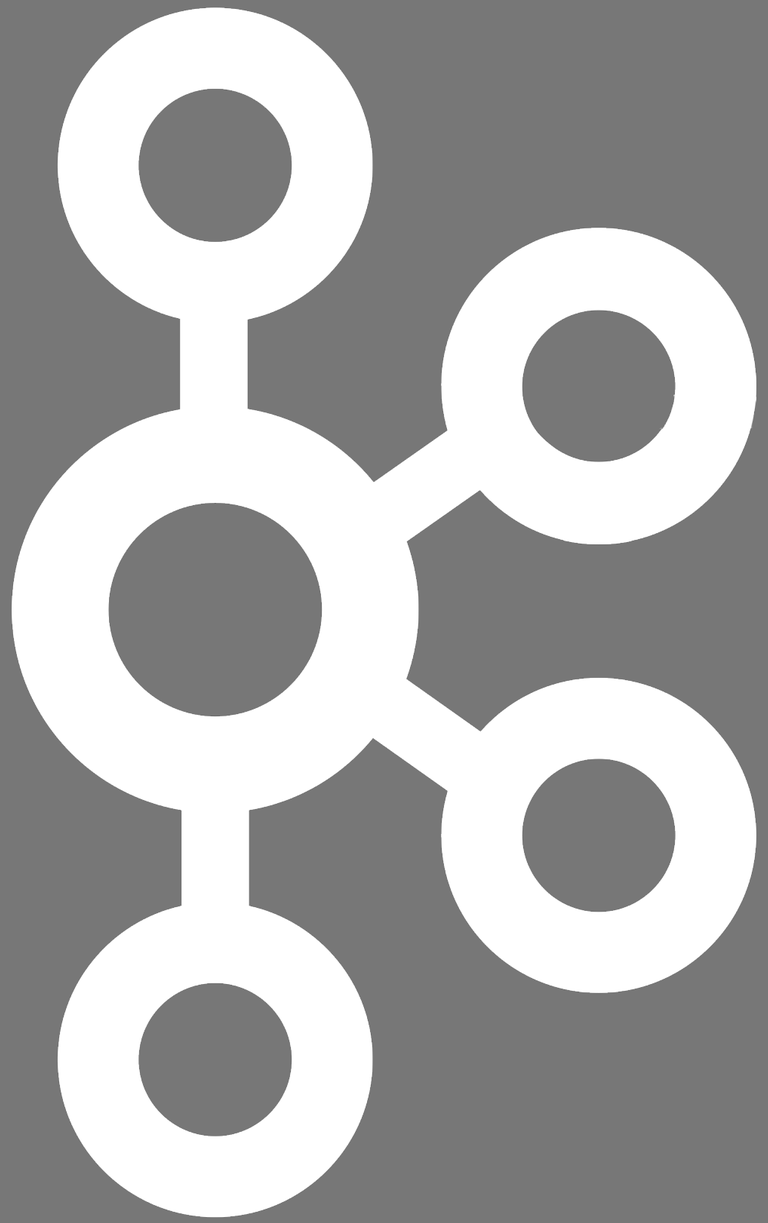
- Hans-Peter Grahsl
- working & living in Graz 🇦🇹
- technical trainer at NETCONOMY
- independent consultant & engineer
- associate lecturer
- 📣 irregular conference speaker 📣

# challenges in today's data architectures

- **rising number of apps** producing + consuming data
- need to integrate **ever more data sources**
- **heterogeneous environments** all over the place
- traditional technologies may **struggle to cope with this**

# challenges may lead to a GIANT MESS

# Apache Kafka



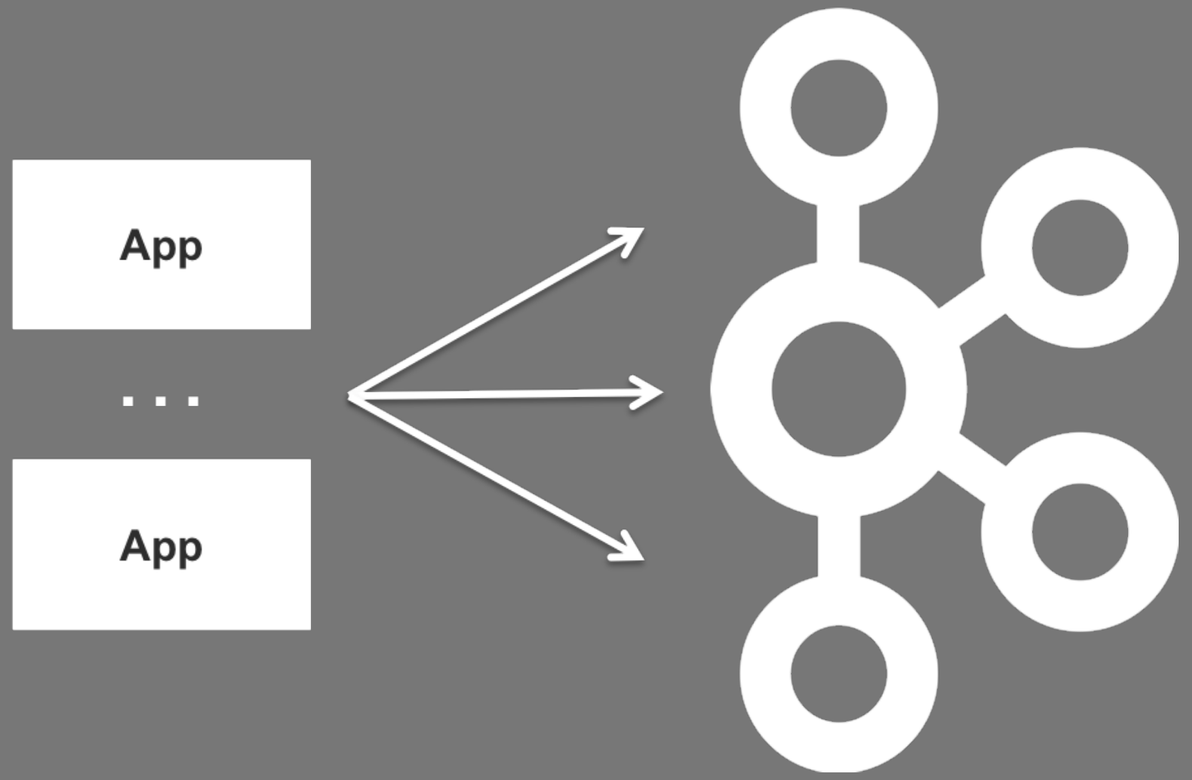
# STREAMING PLATFORM

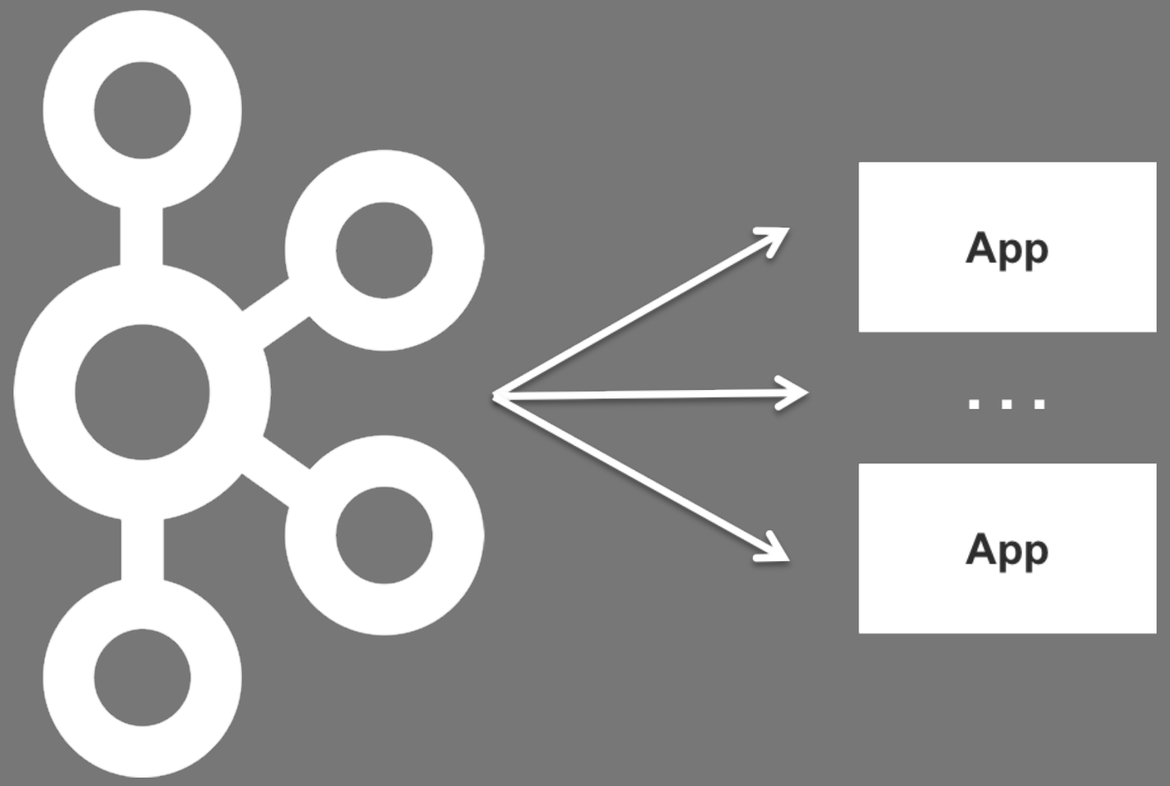
# much more than messaging

- Apache Kafka is offering 3 key capabilities
  - **publish / subscribe to streams** of records
  - (permanently) **store streams** of records
  - **process streams** of records in near **real-time**

# fault-tolerance & horizontal scalability

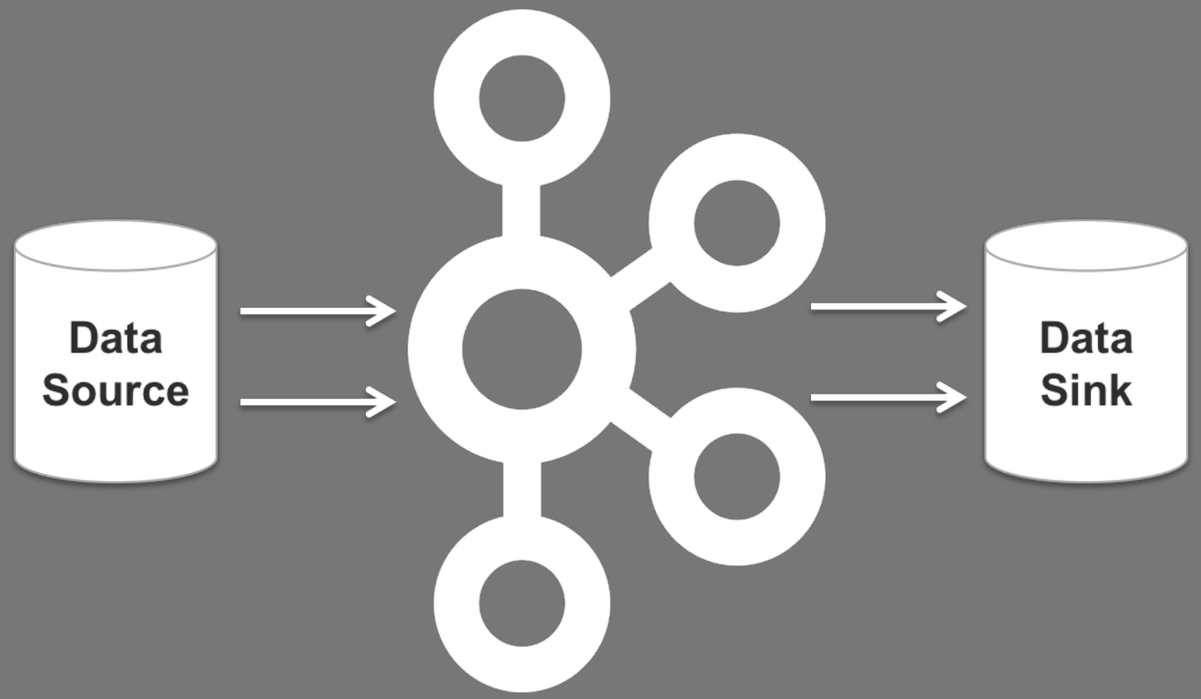
# Producer API

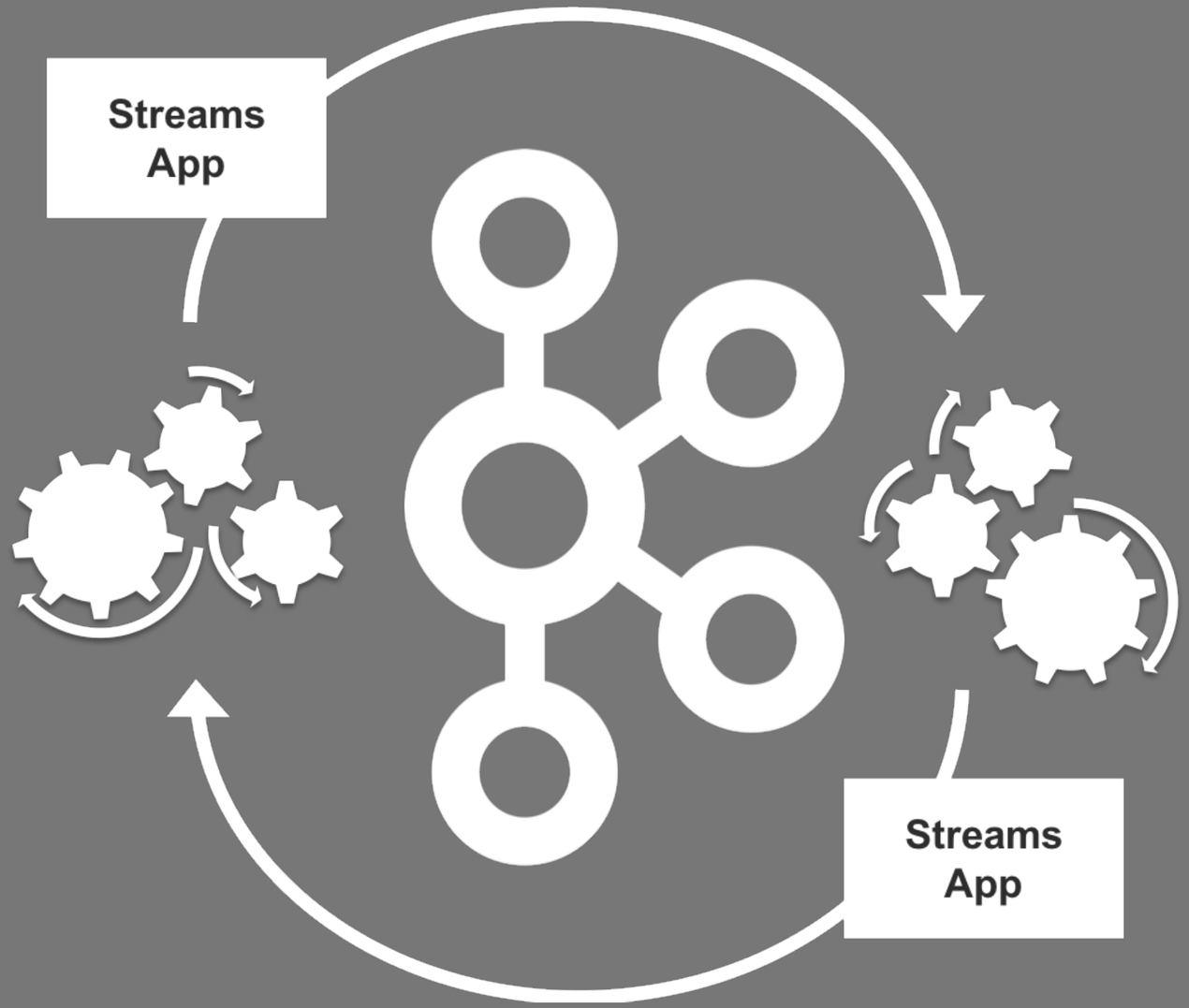





# Consumer API

# Connect API






# Streams
# API

# Kafka Streams API

- stream processing with a **library only approach**
- **lightweight** applications
- **build however & deploy wherever** you like
- **NO(!) additional clusters or frameworks** e.g.


- Processor API & **Streams DSL**
- **configurable delivery guarantees**

# writing applications

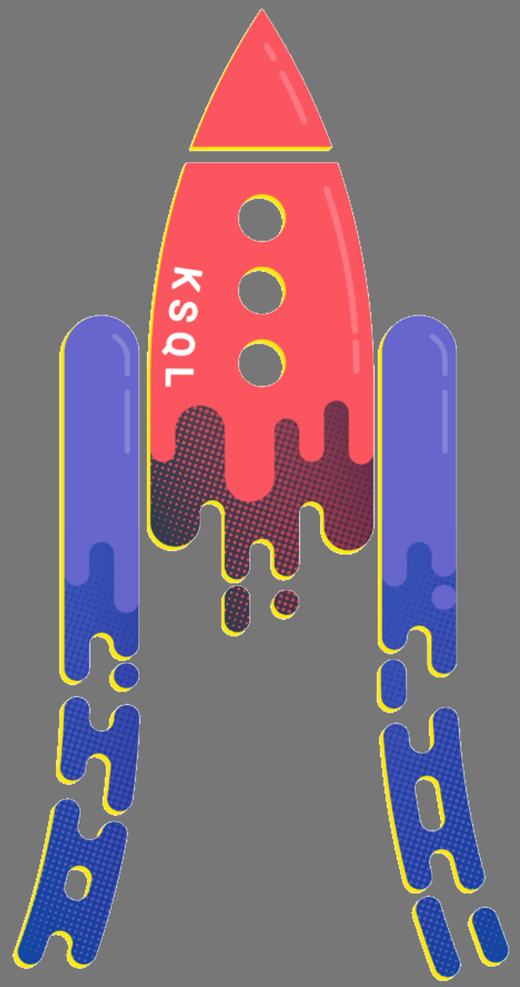
# NOT (!)

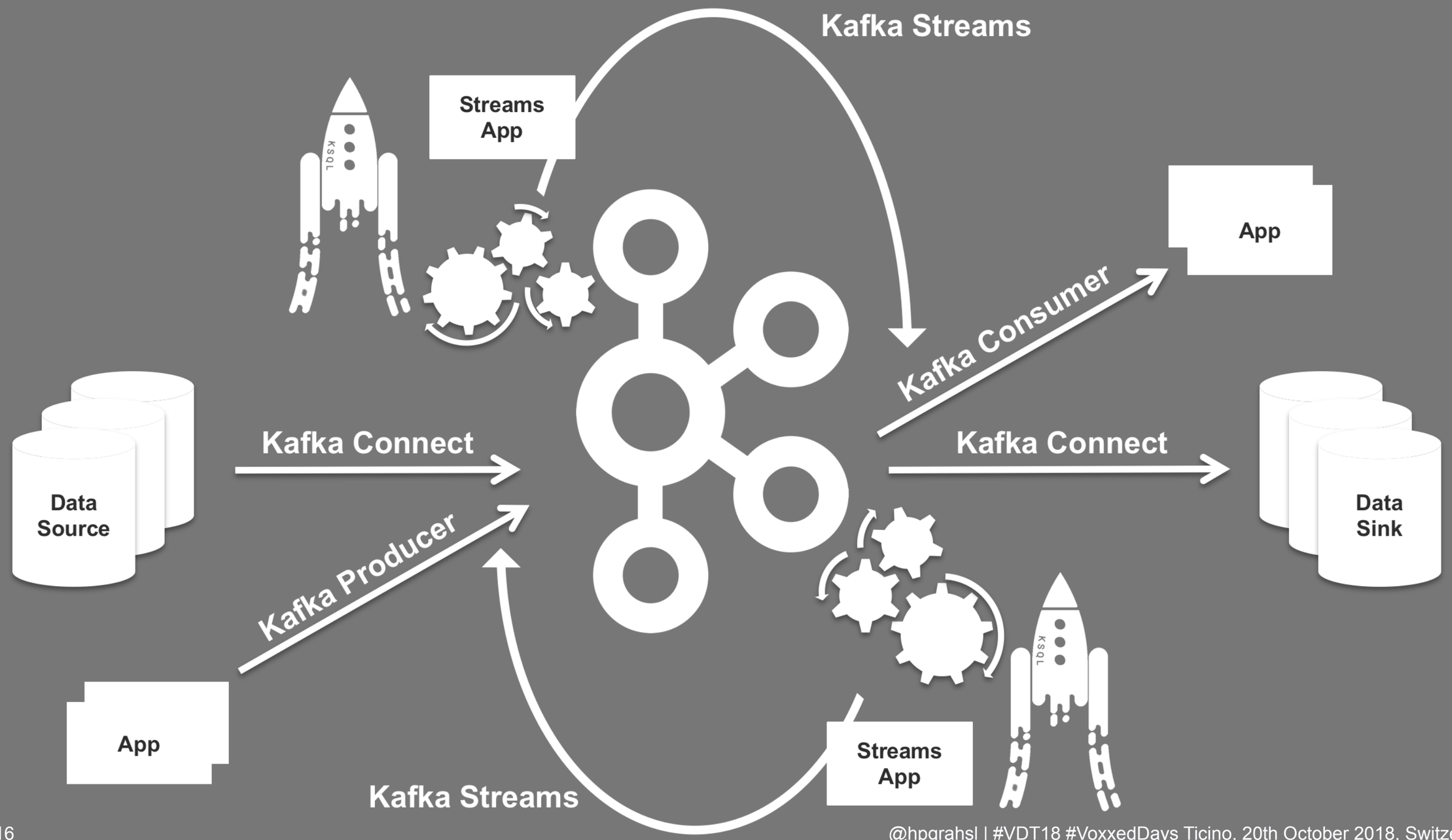
# managing clusters

# Meet KSQL for skyrocketing productivity

# KSQL

- **SQL streaming engine** for Kafka
- concise & expressive
- **SQL-like** language and semantics
- **NO(!) coding** required
- extremely **low entry barrier**
- **joins, aggregations, windowing**
- **UD(A)Fs -** *UDTFs pending...*
- built on top of **KStream API**




Kafka Streams
Streams App
ksql
App
Kafka Consumer
Data Source
Kafka Connect
Kafka Connect
Data Sink
Kafka Producer
App
Kafka Streams
Streams App
ksql

# central nervous system

# example ?

# hmmmm...🤔


EMOJIS
EMOJIS EVERYWHERE
imgflip.com

# example: near real-time Emoji Tracking

| | | | |
|---|---|---|---|
| 🚀 5732 | 🔥 5437 | 😭 5142 | 👉 4699 |
| 👏 2483 | 😀 2145 | ➡️ 2118 | 🙏 2088 |
| 👍 1719 | 🤔 1713 | 😍 1672 | 🎉 1502 |
| 💵 1313 | 💜 1225 | ✨ 1136 | 😩 1134 |
| 📈 1014 | 👀 1005 | $ 981 | 😁 976 |
| 🌟 871 | ▪ 861 | 🎁 850 | 🙌 827 |
| 📣 775 | 😡 775 | 🚨 742 | ⚽ 716 |
| ↪️ 596 | 🏀 577 | 👌 570 | 🔷 565 |
| 🇺🇸 512 | 🍬 495 | 😢 489 | 😅 489 |
| 🏇 457 | 🏆 443 | 🦅 417 | 👍 414 |
| 😈 379 | 💴 368 | 🦄 365 | 🤗 352 |

# HOW TO build this?

# emoji tracking | step 1

# store

ingest subset of public live tweets from Twitter

# emoji tracking | step 2

# process

extract emojis - group & count them - maintain top N

# emoji tracking | step 3

# query

single emoji count - all emoji counts - top N emojis

# emoji tracking | step 4

# notify

consumable near-realtime change streams of updates

# Let's do it!


DATABASES?
WHERE WE'RE GOING WE DON'T NEED DATABASES...
imgflip.com

# example: step 1
# ingest tweets

- using **Kafka Connect**
- e.g. this community connector `https://github.com/jcustenborder/kafka-connect-twitter`
- **configure** the connector (JSON)
- **manage** connector via **REST-like API** `create | pause | resume | delete | status`

```
{ "name": "tweets-twitter-source", "config": {
        "connector.class": "c.g.j.k.c.t.TwitterSourceConnector",
        "twitter.oauth.accessToken": "...",
        "twitter.oauth.consumerSecret": ...",
        "twitter.oauth.consumerKey": "...",
        "twitter.oauth.accessTokenSecret": "...",
        "kafka.status.topic": "tweets",
        "process.deletes": false,
        "key.converter": "org.apache.kafka.connect.json.JsonConverter",
        "key.converter.schemas.enable": false,
        "value.converter": "org.apache.kafka.connect.json.JsonConverter",
        "value.converter.schemas.enable": false,
        "filter.keywords": "..."
} }
```

# NO CODE!

# example: step 2 process tweets

- using **Kafka Streams high-level DSL**
  - **grouping** and **counting** emojis
  - updating **top N emoji counts**
  - map tweets to emoji occurrences
- only a **few lines of Java**

# calculate emoji counts

- It all starts with tweets like this...

🇨🇭 **this is a twitter status** 🏔 **text with** 🧀🧀 **five emojis**

# calculate emoji counts

| | Key | Value |
|---|---|---|
| *raw input* | ID | 🇨🇭 this is a twitter 🇨🇭 status 🏔 text with 🧀🧀 five emojis |
| *extract emoji list* | ID | [🇨🇭,🇨🇭,🏔,🧀,🧀] |

# calculate emoji counts

| Key | Value |
|---|---|
| *flatten the list* | |
| ID | 🇨🇭 |
| ID | 🇨🇭 |
| ID | 🏔️ |
| ID | 🧀 |
| ID | 🧀 |

# calculate emoji counts

| | Key | Value |
|---|---|---|
| *set keys to values* | | |
| | 🇨🇭 | "" |
| | 🇨🇭 | "" |
| | 🏔️ | "" |
| | 🧀 | "" |
| | 🧀 | "" |

**finally group & count by key**

# result: continuously updated KTable with emoji counts

| Key | Value |
|---|---|
| 🇨🇭 | 2 |
| 🏔 | 1 |
| 🧀 | 2 |
| ... | ... |

# 1:1 mapping to KStreams API

```
KTable<String, Long> emojiCounts =
    tweets.map((id,tweet)
                -> KeyValue.pair(id, EmojiUtils...))
          .flatMapValues(emojis -> emojis)
          .map((id,emoji) -> KeyValue.pair(emoji, ""))
          .groupByKey(...).count(...);
```

# example: step 3
# query results

- access to **state stores** with **interactive queries**
- **KStreams** offers all needed **metadata**
- 🤓 **RPC integration** left for **developers** 🤓

**> Reactive WebAPI powered by Spring Boot 2.0 <**

# REST controller

```
@RestController
@RequestMapping("interactive/queries/")
@CrossOrigin(origins = "*")
public class StateStoreController {

    private final StateStoreService service;

    [...]
}
```

# REST controller methods

```
@GetMapping("emojis/{code}")
public Mono<ResponseEntity<EmojiCount>>
                getEmoji(@PathVariable String code) {
    return service.querySingleEmojiCount(code);
}

@GetMapping("emojis")
public Flux<EmojiCount> getEmojis() {
    return service.queryAllEmojiCounts();
}
```

# state store access in service

```
StreamsMetadata metadata =
    kafkaStreams.metadataForKey(
        "your-store-name", emoji, Serializer...
    );
```

# state store access in service

```
if(itsMe.equals(metadata.hostInfo())) {
    ReadOnlyKeyValueStore<String,Long> kvStoreEmojiCounts =
        kafkaStreams.store("your-store-name",
            QueryableStoreTypes.keyValueStore());

    Long count = kvStoreEmojiCounts.get(emoji);
    return Mono.just(
        new ResponseEntity<>(new EmojiCount(...),HttpStatus.OK)
    );
}
```

# state store access in service

```
String location = String.format("http://%s:%d/.../%s",
                    metadata.host(),metadata.port(),emoji);

return Mono.just(
            ResponseEntity.status(HttpStatus.FOUND)
                    .location(URI.create(location)).build()
        );
```


WHAT IF I TOLD YOU
THERE IS NO NEED FOR POLLING?
imgflip.com

# example: step 4
# real-time notifications

- **reactively consume** from changelog topics
- **stream any changes** to clients using SSE

**> Project Reactor's reactor-kafka <**

# notifications via SSE

```
@GetMapping(path = "emojis/updates/notify",
            produces = MediaType.TEXT_EVENT_STREAM_VALUE)
public Flux<EmojiCount> getEmojiCountsStream() {
        return service.consumeEmojiCountsStream();
}
```

# 🔴 LIVE 🔴
# DASHBOARD

| | | | | |
|---|---|---|---|---|
| ✅ 604 | 🤔 579 | 🚨 578 | 🤷‍♀️ 572 | 🙏 529 |
| 👆 480 | ❤️ 452 | 🤑 450 | ‼️ 443 | ⚡ 436 |
| 👍 404 | 💪 402 | 💯 398 | 👉 377 | 😩 351 |
| 💸 294 | ⭕ 284 | 🤦‍♂️ 280 | 😀 269 | ⭐ 259 |
| 👋 232 | 👀 216 | 🛬 208 | 🔅 200 | ⚠️ 192 |
| 👊 188 | 😁 184 | 👌 174 | 😊 171 | 💥 171 |
| 🔶 164 | ✨ 164 | 📋 151 | 🤠 145 | 📈 144 |
| 🎉 139 | 📉 137 | 😉 136 | 🇺🇸 134 | 🗣️ 133 |
| 😒 119 | 😅 115 | 💚 113 | 🔷 112 | 😢 111 |
| 😕 104 | 😌 102 | 😪 101 | 😤 100 | ✔️ 100 |
| 🤗 94 | 😇 93 | 😳 92 | 🔴 90 | $ 89 |
| 🌍 83 | 😐 81 | ⏱️ 80 | ☺️ 80 | ♥ 79 |
| 🖕 75 | !? 75 | 🚽 74 | ⏰ 74 | ▶ 73 |


# mission accomplished

# try it yourself!

## source

https://github.com/hpgrahsl/voxxed-days-ticino-2018

## slides

https://speakerdeck.com/hpgrahsl/stateful-and-reactive-stream-processing-applications-without-a-database-at-voxxedticino-2018

# THANK YOU

# Q&A ?

NETCONOMY