Turning the wheel - Streaming 4.4 billion events with Apache Kafka

Streaming customer, policy and vehicle information via Apache Kafka and MongoDB – and what we learnt on the way …

Simon Aubury

April 2019

Agenda

Tonight

- Context - what problem are we trying to solve
- Architecture of our data flow
- Kafka & Kafka Connect
- Challenges .. and solutions

iag

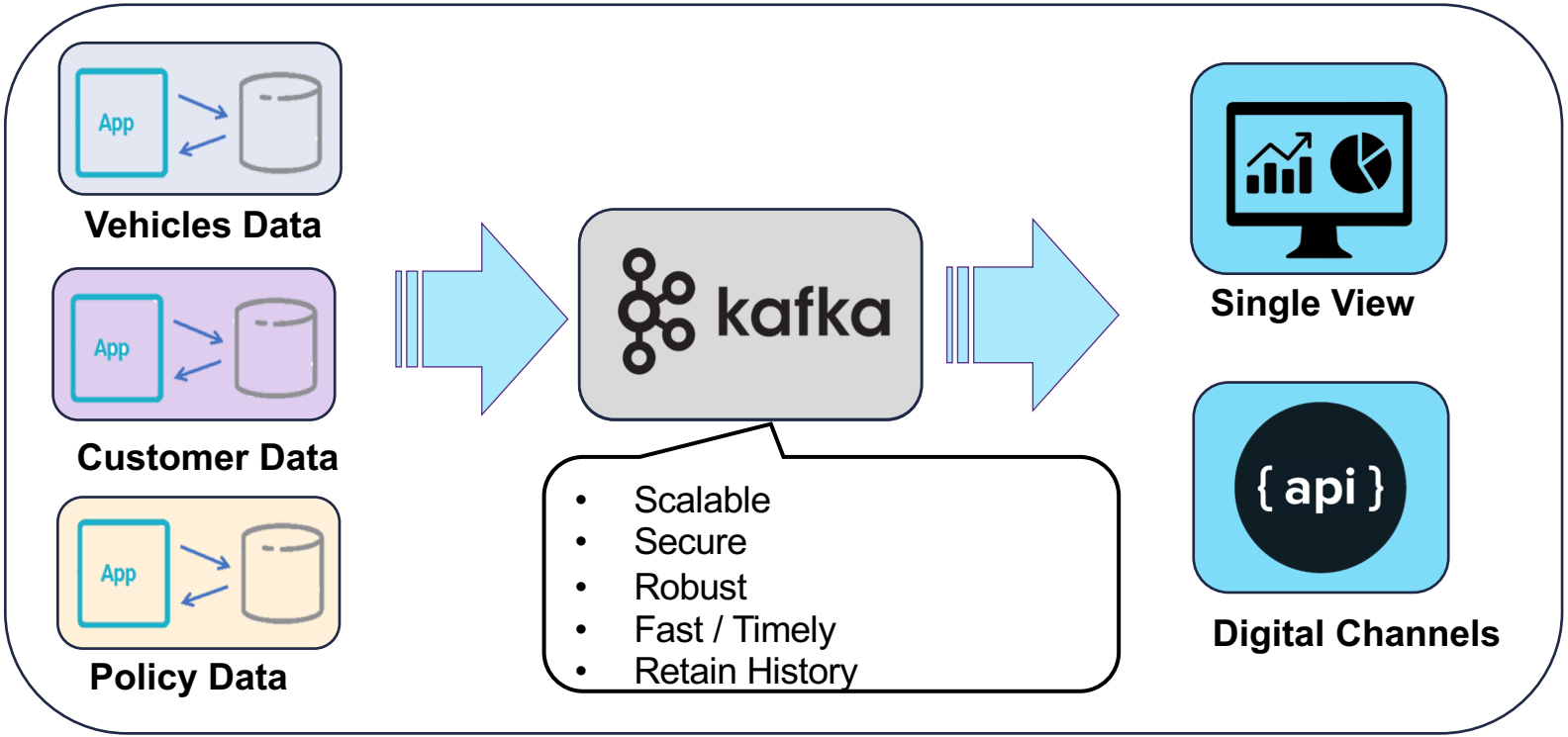
We’ve a lot of data

Context #1

mvyear	mvmake	mvmodel	mvbody
1886	RUDGE	PENNY FARTHING	MBIKE
1896	FORD	QUADRICYCLE	CONVT
1896	FORD	QUADRICYCLE	CONVT
1896	FORD	QUADRICYCLE	CONVT
1896	FORD	QUADRICYCLE	CONVT
1896	FORD	QUADRICYCLE	CONVT
1896	FORD	QUADRICYCLE	CONVT
1896	FORD	QUADRICYCLE	CONVT
1896	FORD	QUADRICYCLE	CONVT
1900	MERCEDES		CONVT

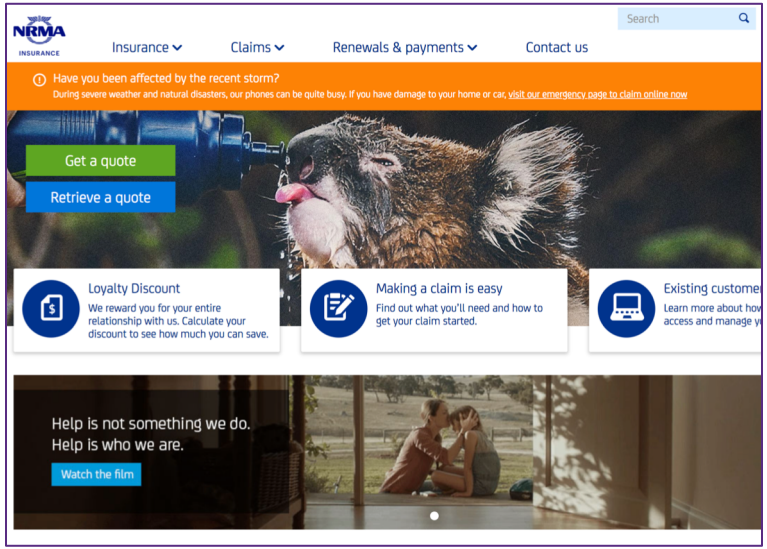
We've a lot of systems

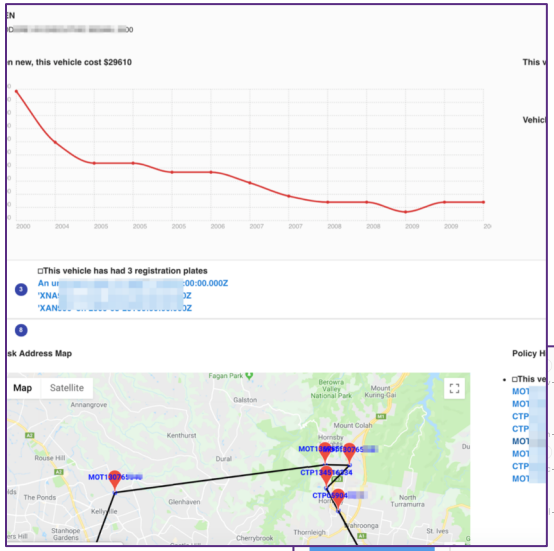
Context #2



We want to tie it together

Context #3



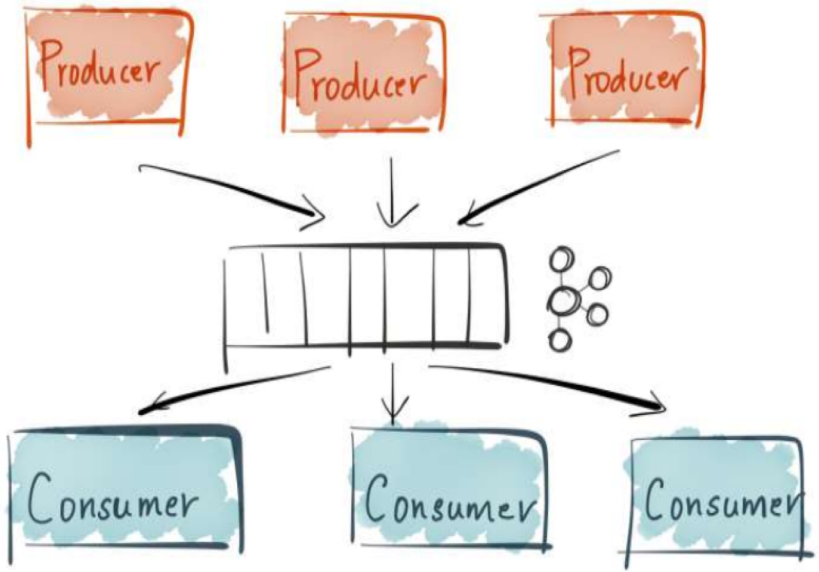


What is Kafka?

A very quick into

Apache Kafka : *Unified, high-throughput, low-latency platform for handling real-time data feeds*

- Originally developed by LinkedIn, open sourced in early 2011
- “The global commit log thingy”
- Kafka maintains feeds of messages in topics
- Appends ; ordered, immutable sequence

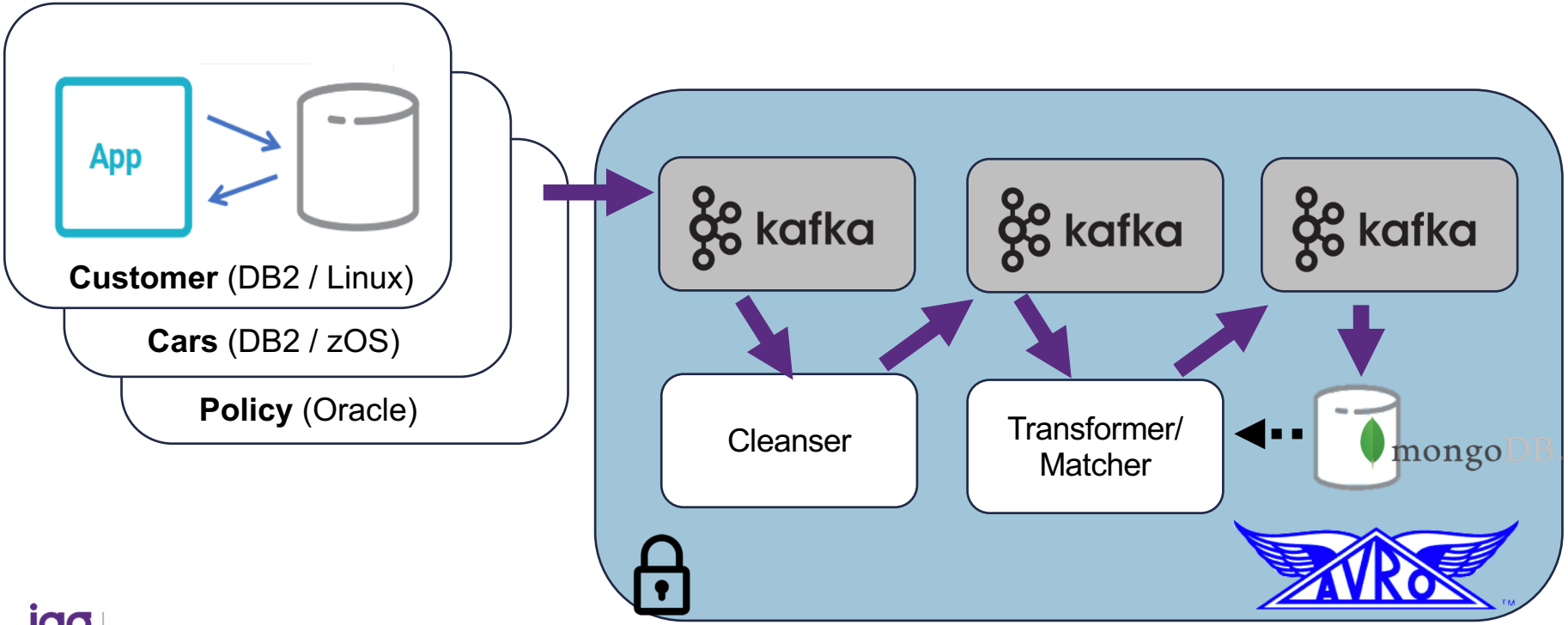


iag

Credit: https://www.confluent.io/blog/stream-data-platform-1/

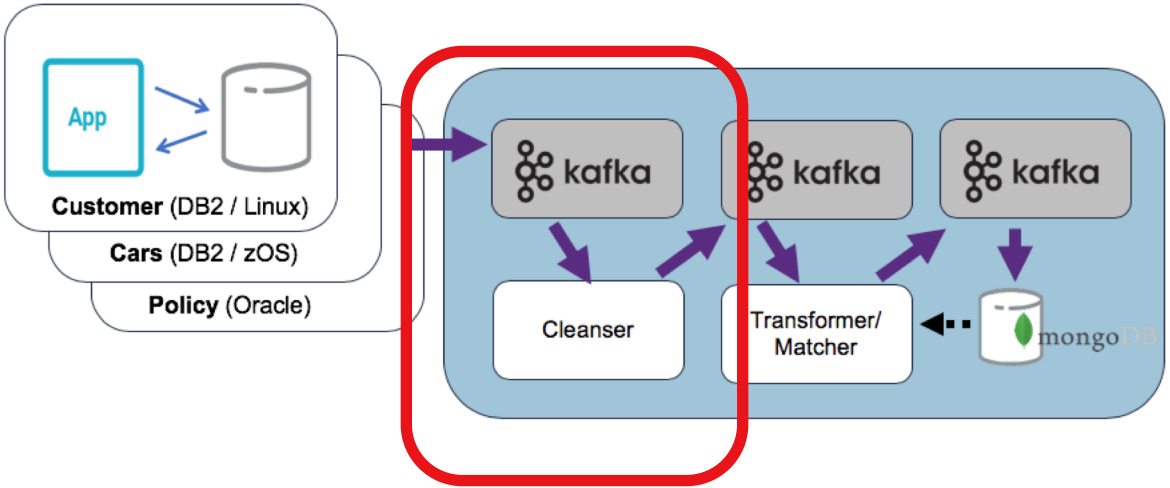
Architecture of our data flow

Lots of boxes



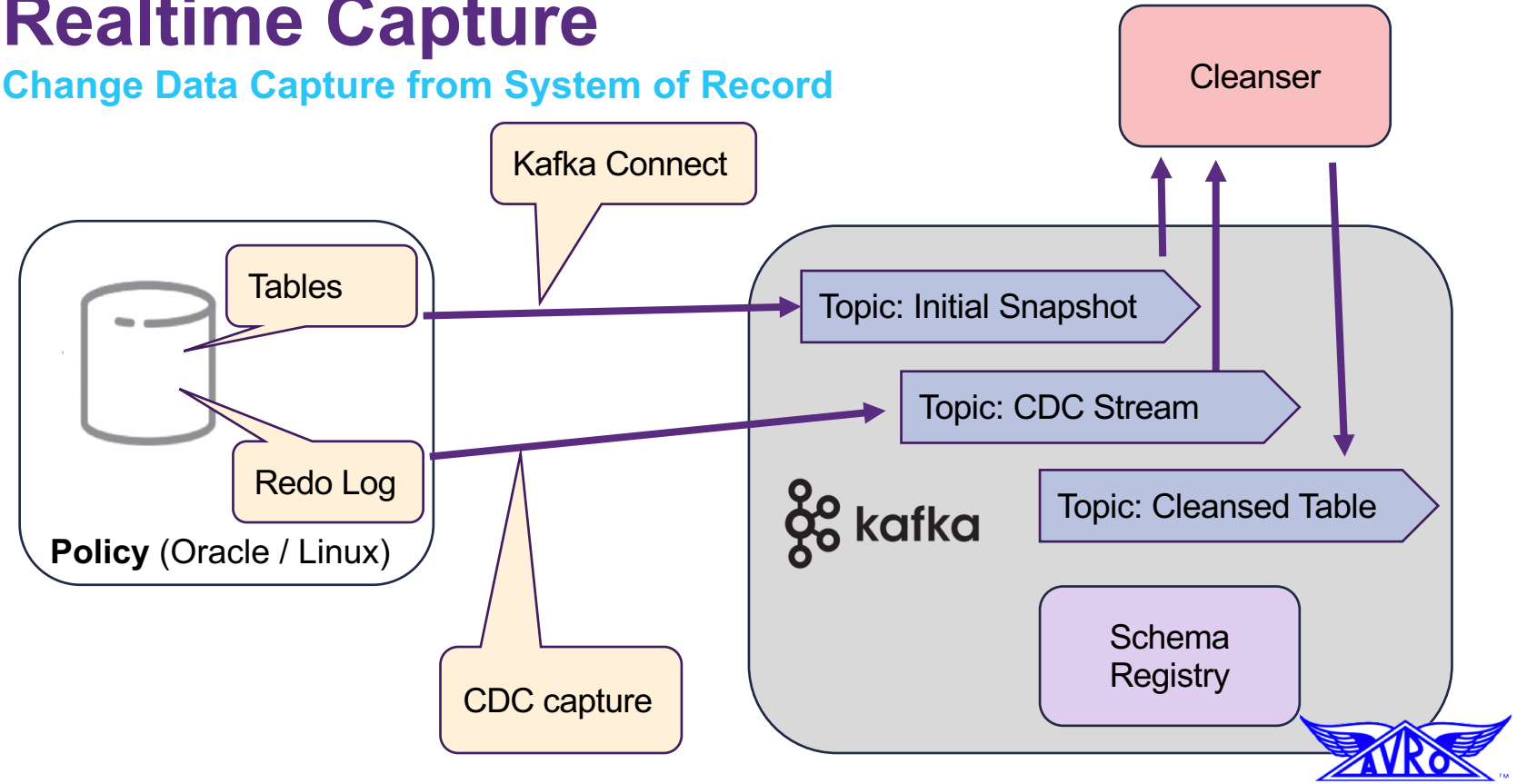
Part 1 - Extract

Source System Low Touch Data Acquisition



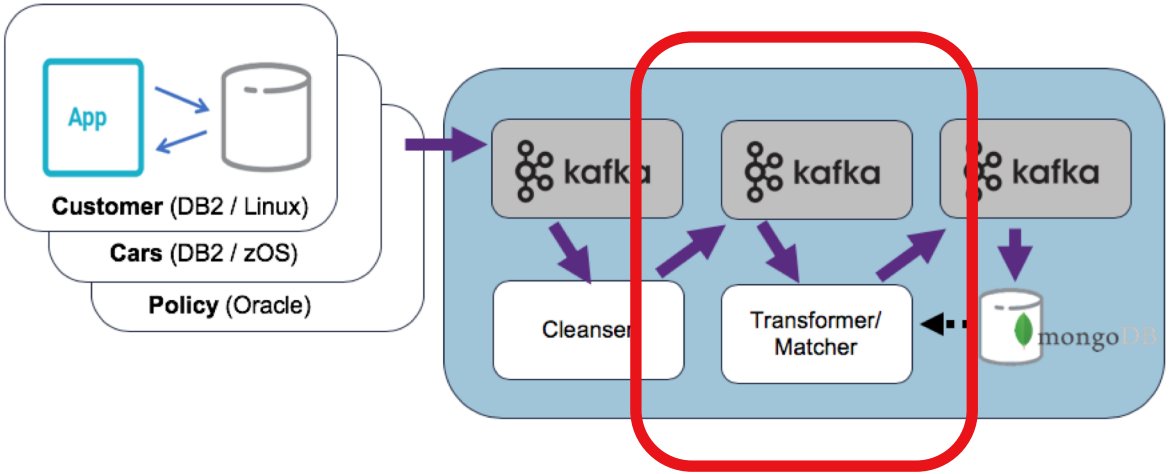
Realtime Capture

Change Data Capture from System of Record

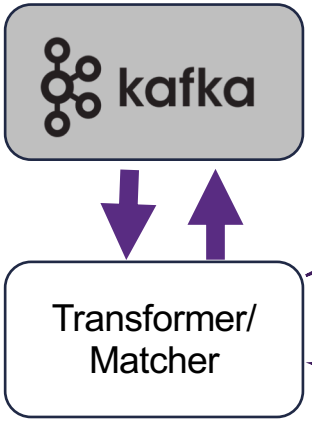


Part 2 – Transformation & Matching

Finding stuff



Transform & Match



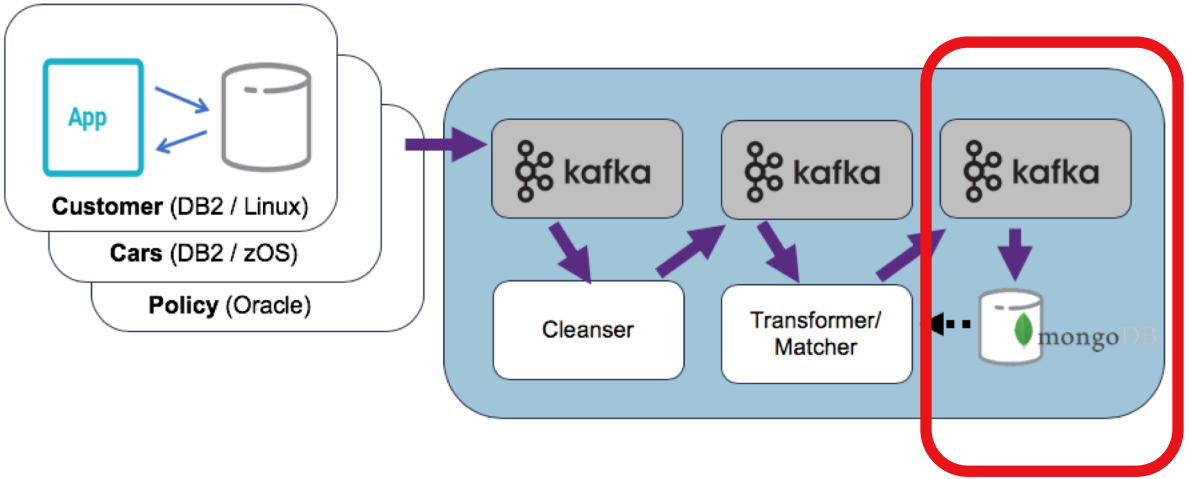

```
CREATE STREAM insurance_event_with_repairer AS \
SELECT *, geo_distance(iel.pc_lat, iel.pc_long, rct.lat, rct.long,
'km') AS dist_to_repairer_km
FROM insurance_event_with_location iel \
INNER JOIN repair_center_tab rct ON iel.pc_state = rct.repair_state;
```

```
final KTable<CcuserCcPolicyKey, ClaimDetails> claimDetailsTable = ccPiSorStreams.claim() KSt
    .flatMapValues(statefulFilter(validClaim)) KStream<CcuserCcClaimKey, CcuserCcClaimEnvelope>
    .mapValues((k, v) -> v.getAfter()) KStream<CcuserCcClaimKey, CcuserCcClaim>
    .leftJoin(
        ccPiGlobalTables.brandExt(),
        (k, v) -> new BrandTypeCodeKey(v.getCLAIMNUMBER().substring(0, 3).toUpperCase()),
        (claim, brandExtEnv) -> {...}) KStream<CcuserCcClaimKey, ClaimDetails>
    .groupBy((k, v) -> new CcuserCcPolicyKey(v.getPolicyId()), serdes.groupedWith()) KGroupe
    .reduce((prev, latest) -> latest, serdes.materializedAs( topicName: "claimDetails")) KTabl
    .join(
        policyTable,
        (claimDetails, policy) ->
            ClaimDetails.newBuilder(claimDetails)
                .setPolicyDetails(
                    PolicyDetails.newBuilder()
                        .setBrand(policy.getDIDISTRIBUTOR())
                        .setPolicyNumber(policy.getPOLICYNUMBER())
                        .build())
                .build(),
        serdes.materializedAs( topicName: "claimDetailsWithPolicy"));
```

iag

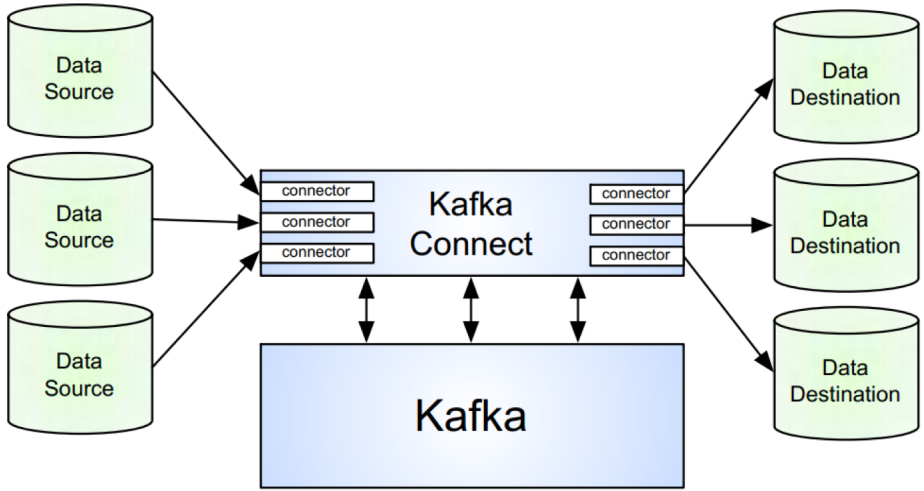
Part 3 – Serving Layer

Sinking Results



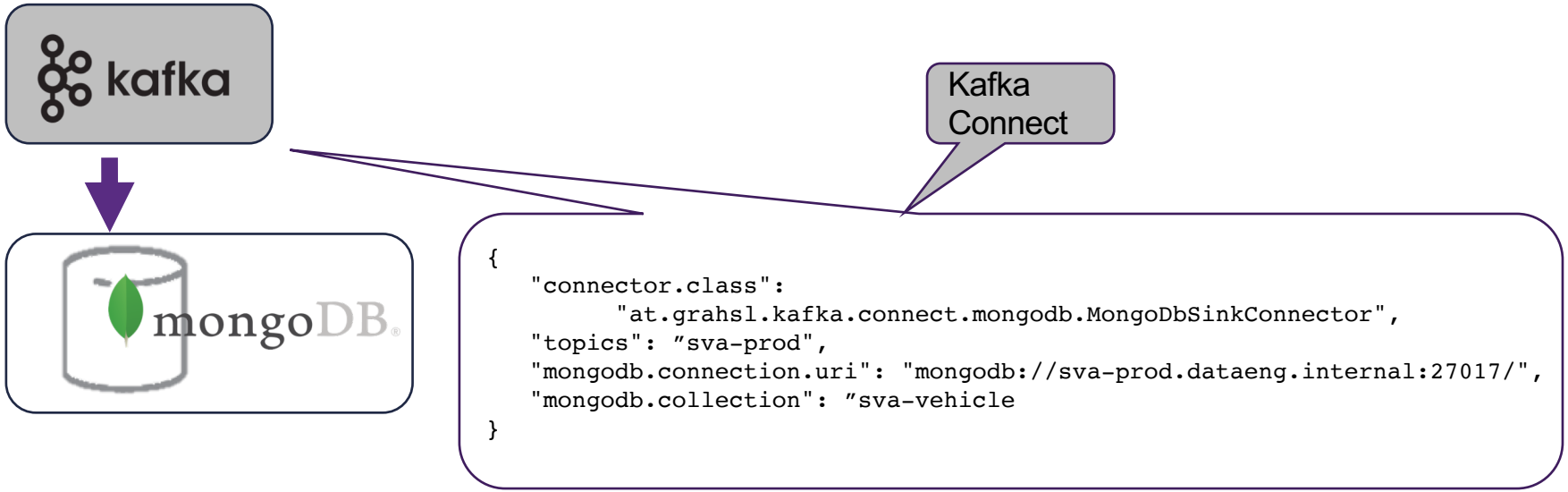
Kafka Connect

- Distributed, scalable, fault-tolerant service designed to reliably stream data between Kafka and other data systems
- **Source Connectors** import data from another system (e.g. a relational database into Kafka)
- **Sink Connectors** export data (e.g. the contents of a Kafka topic to an HDFS file).



Kafka Connect Sink

Writing to MongoDB Serving Layer

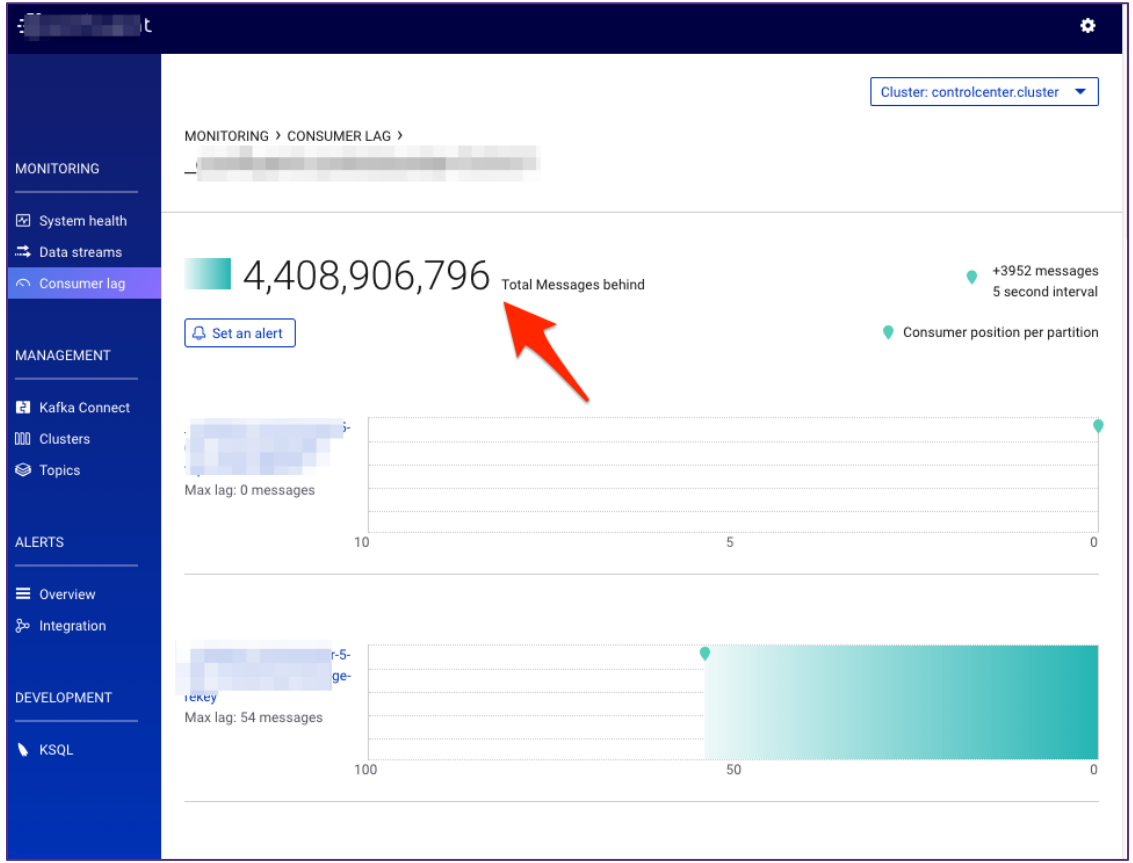


What did we discover?

Slow to fast … to *really* fast!

Challenge

Lots of data



Hot code #1

A bit of caching

Before: very slow transform
30 records / sec / table

```
-       doApply = transform[R](new AvroData(new AvroDataConfig(props)), schemaFetcher, registerSchema)
+       doApply = transform[R](new AvroData(new AvroDataConfig(props)), cachingSchemaFetcher, registerSchema)
```

After: add cache for schema lookup
200 records / sec / table

```
val cachingSchemaFetcher: ConnectRecord[R] => Throwable \/ Schema =
  record => schemaCache.get(record.topic()) match {
    case Some(schema) => schema.right[Throwable]
    case None =>
      val result = for {
        initTopicName <- \/.fromEither(TopicName.fetch(record.topic))
          .leftMap(_ => new IllegalArgumentException(s"Invalid init topic format: ${record.topic}"))
        schema <- schemaFetcher.apply(initTopicName)
      } yield schema
      result.foreach(schema => schemaCache = schemaCache + (record.topic() -> schema))
      result
  }
```

iag

Hot code #2

A bit more caching

Before: still slow transform
200 records / sec / table

```
doApply = transform[R](new AvroData(new AvroDataConfig(props)), cachingSchemaFetcher, registerSchema)
doApply = transform[R](new AvroData(new AvroDataConfig(props)), cachingSchemaFetcher, registerSchema, cachingDocParser)
```

After: add cache for field metadata
5,500 records / sec / table

```
val cachingDocParser: String => String \/ PwxAvroFieldDocComment =
  docField => docCommentCache.get(docField) match {
    case Some(docComment) => docComment.right[String]
    case None =>
      val result = PwxAvroFieldDocComment.parse(docField)
      result.foreach(docComment => docCommentCache = docCommentCache + (docField -> docComment))
      result
  }
```

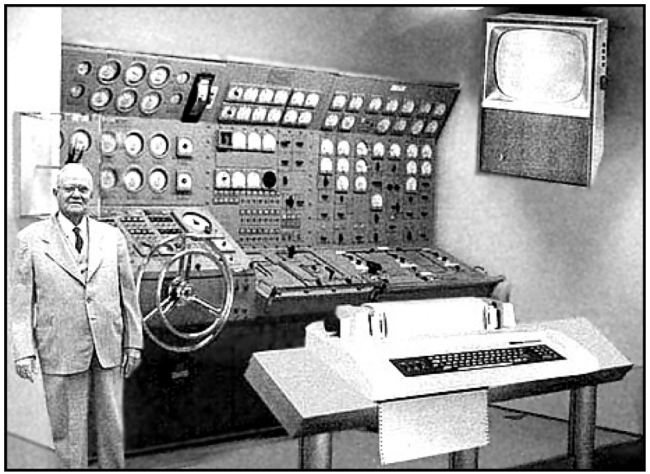
iag

Horizontal scaling?

Theory

To scale out, you simply start another instance of your stream processing application, e.g. on another machine. The instances of your application will become aware of each other and automatically begin to share the processing work.

https://www.confluent.io/blog/elastic-scaling-in-kafka-streams/

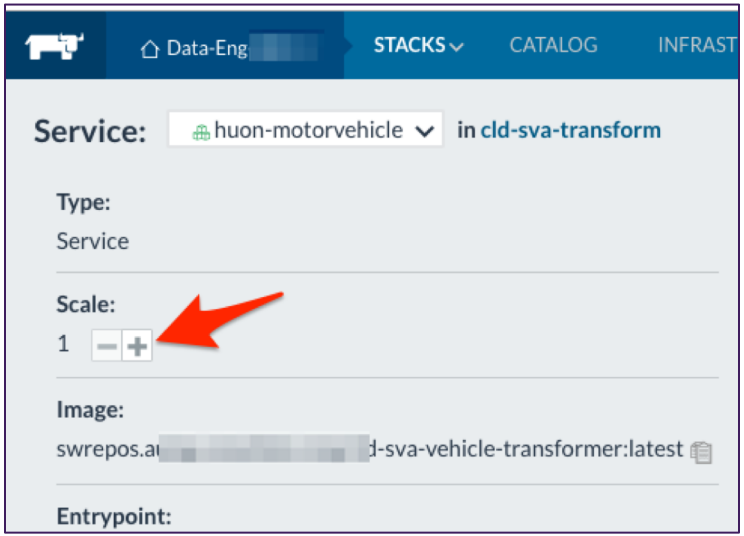


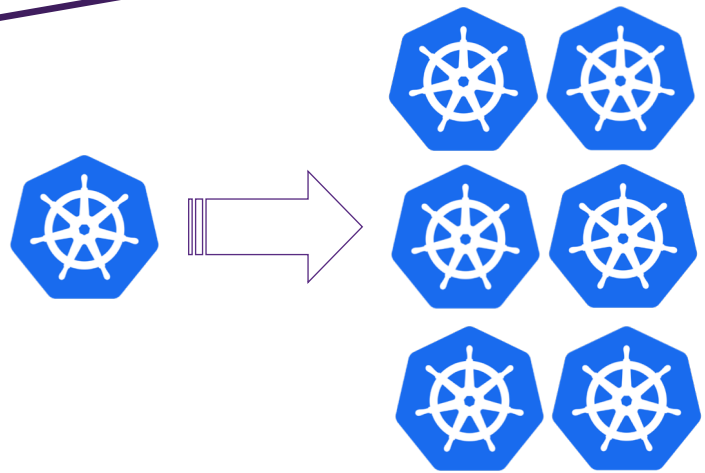
iag

Horizontal scaling … scales horizontally!

Testing

[redacted] 9:55 AM
scaling out huon now
will expand to 6 nodes then restart them all clearing out state store
that will spread the work more evenly, otherwise it will leave a lot of the stateful work on the original two nodes (since it favours nodes that have already run tasks previously so it doesn't rebuild state stores)
that should hopefully move things along 🙂





Horizontal scaling … meet efficient code

Reality

30 records / sec

Approx 100,000 / hr

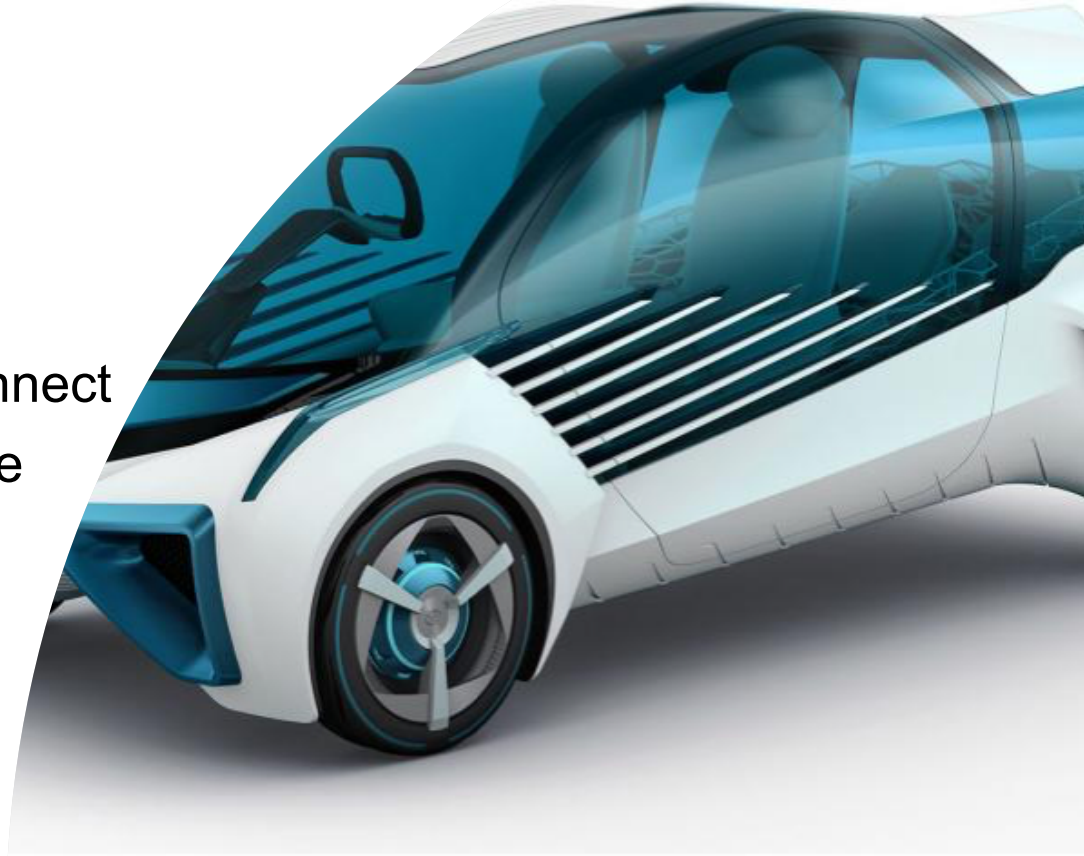
33,500 records / sec

Approx 130 mill / hr

Summary

What did we cover again?

- Architecture of our data flow
  - Extract - Low touch CDC
  - Transformation & Match
  - Serving ; Kafka & Kafka Connect
- Solutions for high performance



iag

Questions?

Australia

NRMA INSURANCE · SGIO · SGIC · CGU · wfi · swann insurance

New Zealand

STATE · ami · N1 · nac INSURANCE · LANTERN insurance · swann insurance

Asia

N1 · AmAssurance · AKURNIA INSURANS · ASURANSI PAROLAMAS · SBIGeneral INSURANCE · AAA ASSURANCE