Avro and Beam Schemas

Without Smashing Your Keyboard

Devon Peticolas
Oden Technologies

devon@peticol.as
@dpet@mastodon.social

BEAM SUMMIT
NYC 2023

These slides are available at

github.com/x/slides/tree/master/beam-summit-2023

Agenda

- Intro and Background
- What is Avro (and what is an Avro Schema)
- What is are Beam Schemas
- Making Them Work Together
- The Cost Benefits on GCP
- Combined Batch and Stream IO with Avro and Beam Schemas
- Q&A

Introduction

Who Am I

Devon Peticolas

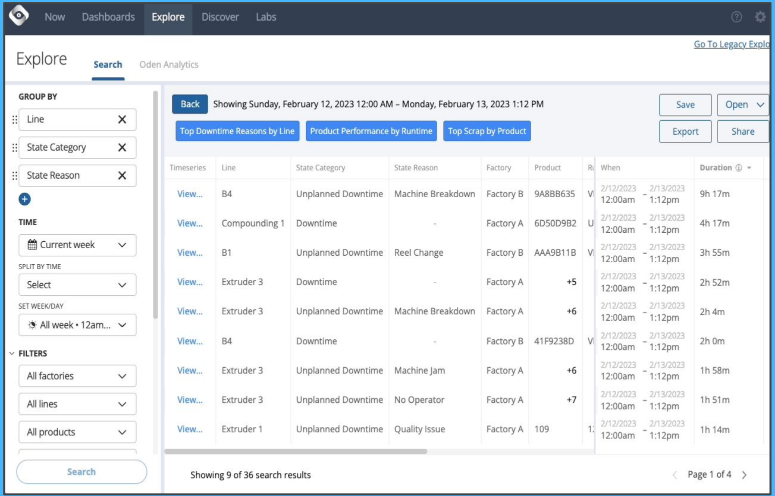
- Principal Engineer at Oden
- Beam user for 5 years
- Lead the “Efficiency” team at Oden
- This is my **third** Beam Summit talk

Likes	Dislikes
Burritos	Hot Dogs
Python	Python Beam SDK
withAllowed TimestampSkew	Every line of Java I’ve ever written

Who is Oden Technologies?

Oden Technologies

- Think “New Relic but for manufacturing”
- Real-time and historical analytics for manufacturing
- We have customers in plastics, chemical, packaging
- We have lots of time-series data



Explore

GROUP BY: Line; State Category; State Reason

TIME: Current week

SPLIT BY TIME: Select

SET WEEK/DAY: All week • 12am...

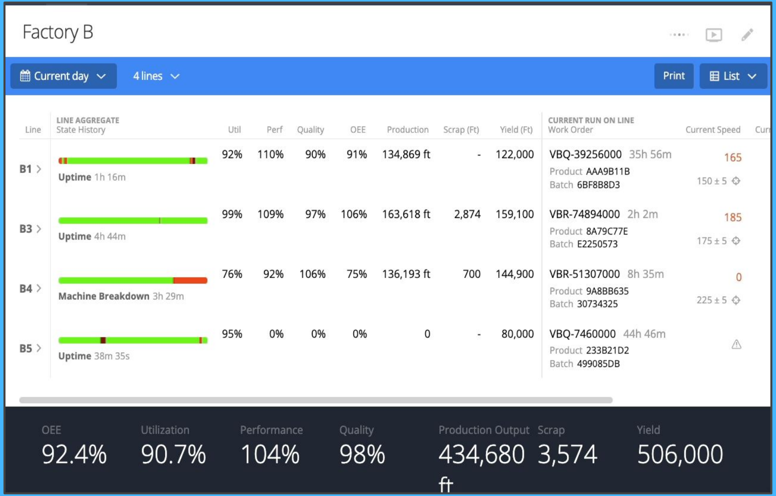
FILTERS: All factories; All lines; All products

Showing Sunday, February 12, 2023 12:00 AM – Monday, February 13, 2023 1:12 PM

Timeseries	Line	State Category	State Reason	Factory	Product	Ru	When	Duration
View...	B4	Unplanned Downtime	Machine Breakdown	Factory B	9A8BB635	V	2/12/2023 12:00am – 2/13/2023 1:12pm	9h 17m
View...	Compounding 1	Downtime	-	Factory A	6D50D9B2	U	2/12/2023 12:00am – 2/13/2023 1:12pm	4h 17m
View...	B1	Unplanned Downtime	Reel Change	Factory B	AAA9B11B	V	2/12/2023 12:00am – 2/13/2023 1:12pm	3h 55m
View...	Extruder 3	Downtime	-	Factory A	+5		2/12/2023 12:00am – 2/13/2023 1:12pm	2h 52m
View...	Extruder 3	Unplanned Downtime	Machine Breakdown	Factory A	+6		2/12/2023 12:00am – 2/13/2023 1:12pm	2h 4m
View...	B4	Downtime	-	Factory B	41F9238D	V	2/12/2023 12:00am – 2/13/2023 1:12pm	2h 0m
View...	Extruder 3	Unplanned Downtime	Machine Jam	Factory A	+6		2/12/2023 12:00am – 2/13/2023 1:12pm	1h 58m
View...	Extruder 3	Unplanned Downtime	No Operator	Factory A	+7		2/12/2023 12:00am – 2/13/2023 1:12pm	1h 51m
View...	Extruder 1	Unplanned Downtime	Quality Issue	Factory A	109	1	2/12/2023 12:00am – 2/13/2023 1:12pm	1h 14m

Showing 9 of 36 search results

Page 1 of 4



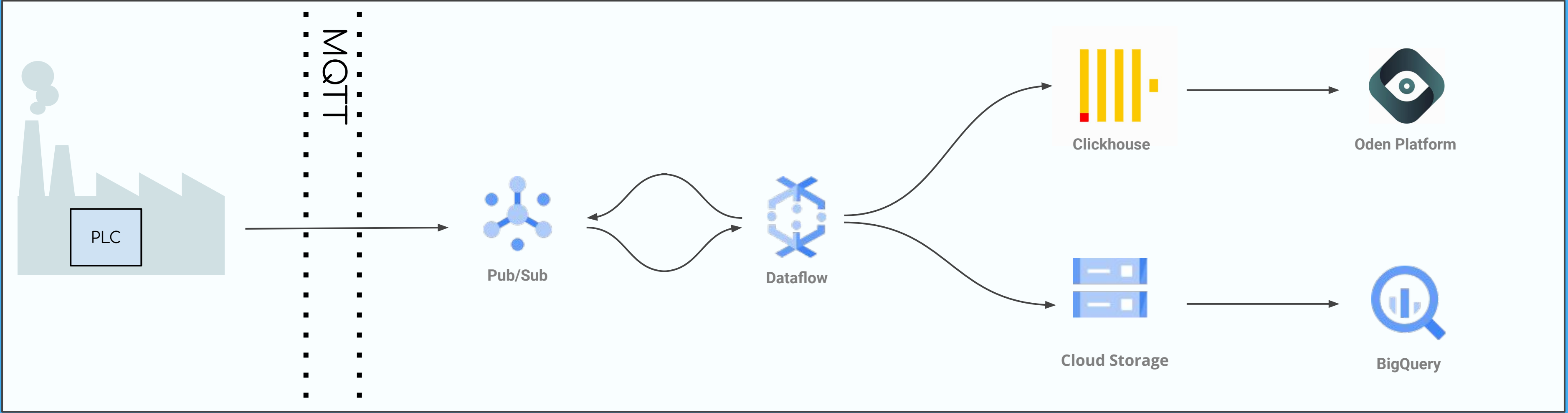
Factory B

Line	LINE AGGREGATE State History	Util	Perf	Quality	OEE	Production	Scrap (Ft)	Yield (Ft)	CURRENT RUN ON LINE Work Order	Current Speed
B1	Uptime 1h 16m	92%	110%	90%	91%	134,869 ft	-	122,000	VBQ-39256000 35h 56m Product AAA9B11B Batch 6BF8B8D3	165 150 ± 5
B3	Uptime 4h 44m	99%	109%	97%	106%	163,618 ft	2,874	159,100	VBR-74894000 2h 2m Product 8A79C77E Batch E2250573	185 175 ± 5
B4	Machine Breakdown 3h 29m	76%	92%	106%	75%	136,193 ft	700	144,900	VBR-51307000 8h 35m Product 9A8BB635 Batch 30734325	0 225 ± 5
B5	Uptime 38m 35s	95%	0%	0%	0%	0	-	80,000	VBQ-7460000 44h 46m Product 233B21D2 Batch 499085DB	⚠

OEE	Utilization	Performance	Quality	Production Output	Scrap	Yield
92.4%	90.7%	104%	98%	434,680 ft	3,574	506,000

How Does Oden Use Beam

- Streaming ingest of “raw” manufacturing data and transformation into metrics
- Streaming transformation and streaming joins of metrics
- Streaming transformation of metrics into contextual intervals
- Streaming delivery of metrics into Clickhouse (TSDB) and GCS (backup)
- Batch versions of all of the above for outage recoveries



How Does Oden Use Beam

- Streaming ingest of “raw” manufacturing data and transformation into metrics
- Streaming transformation and streaming joins of metrics
- Streaming transformation of metrics into contextual intervals
- Streaming delivery of metrics into Clickhouse (TSDB) and GCS (backup)
- **Batch versions of all of the above for outage recoveries**

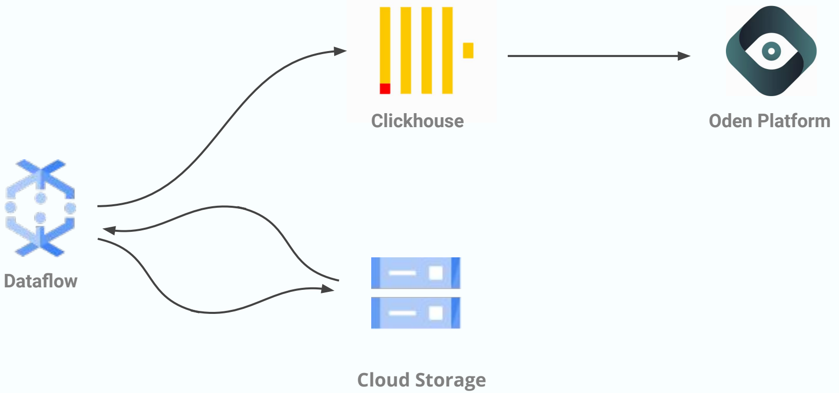
LEVERAGING BEAM'S BATCH-MODE FOR ROBUST RECOVERIES AND LATE-DATA PROCESSING OF STREAMING PIPELINES

Devon Peticolas
Principal Engineer
@ Oden Technologies

BEAM SUMMIT

August 4-6, 2021
100% online and free
2021.beamsummit.org

youtu.be/g1p8PR44l90



Example Oden Beam Job

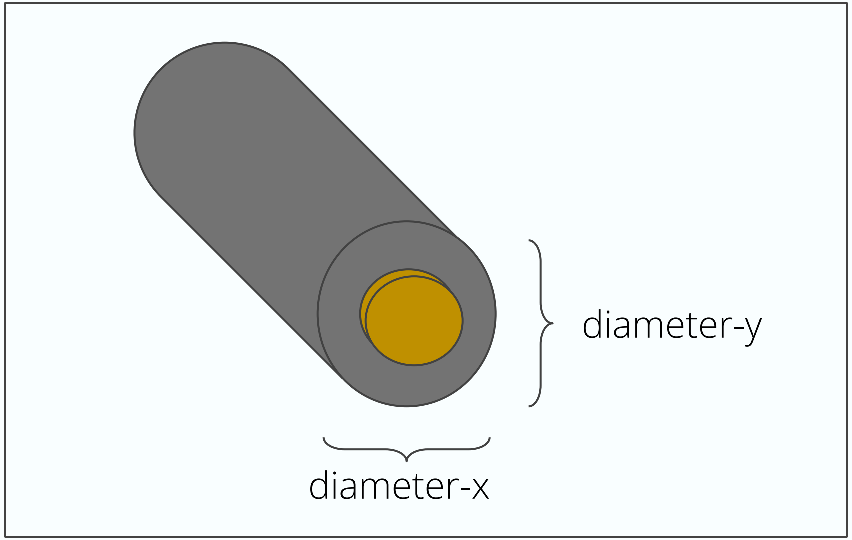
User Has

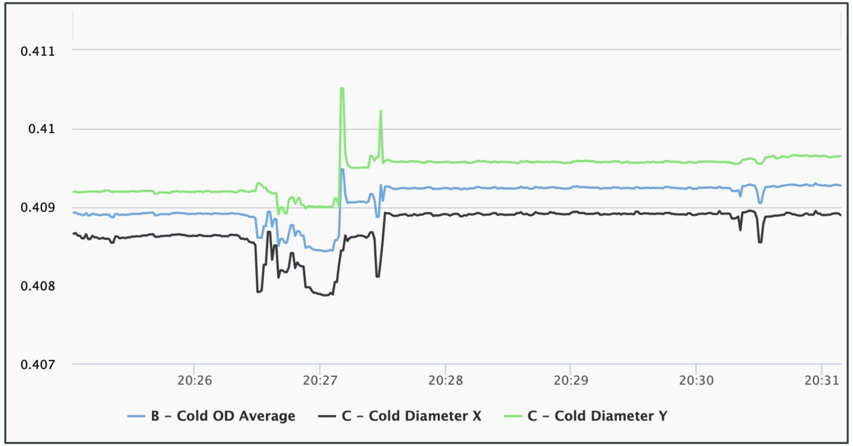
`diameter-x` and `diameter-y`

User Wants

```
avg-diameter = (diameter-x + diameter-y) / 2
```

- Metrics need to be computed in real-time
- Components can be read from different devices with different clocks
- Formulas are stored in postgres





Example Oden Beam Job 2

User Has

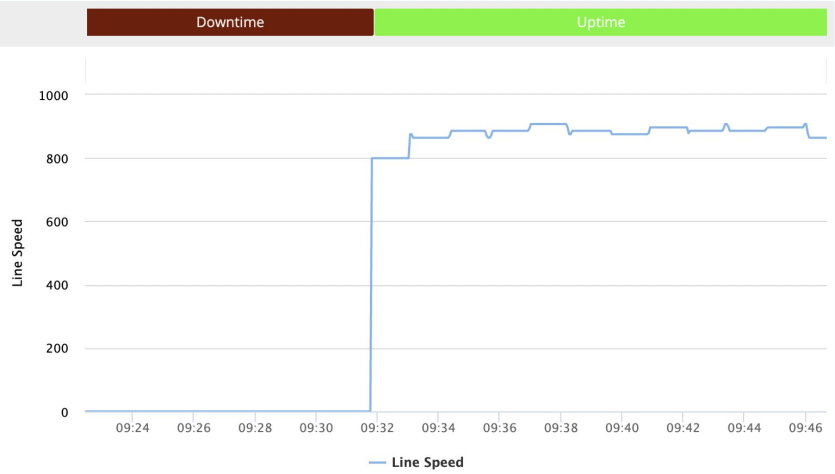
`line-speed`

User Wants

`Uptime and Downtime`

- Sometimes, a metric crossing a threshold defines an “interval change”
- These intervals need to be created in real-time

youtu.be/MywQDgnJCw0



Avro and Beam Schemas

Apache AVRO

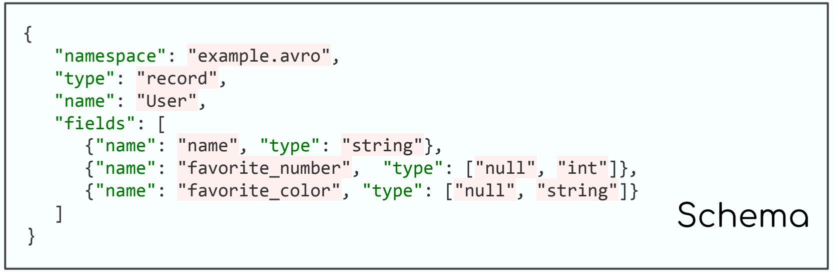
- Language-independent RPC and Serialization format
- An Avro Schema is a JSON blob that defines a record type
- Records are encoded and decoded using the Avro Schema
- Avro files contain the Avro Schema as part of the header (self-describing)

In Java

- Schemas are used to generate classes

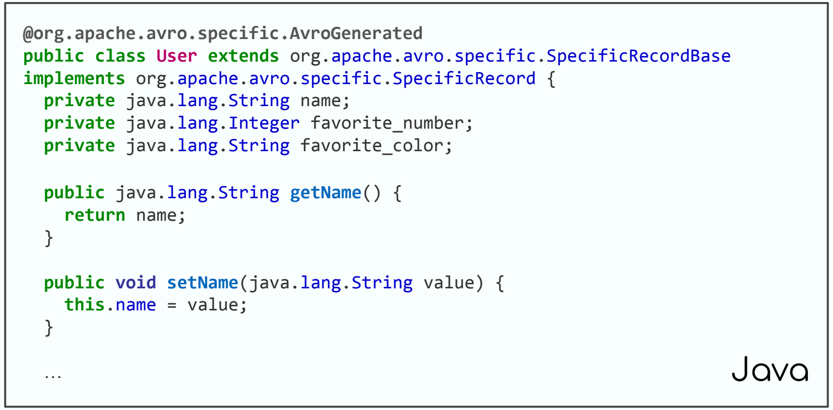
In Python

- Schemas are used dynamically



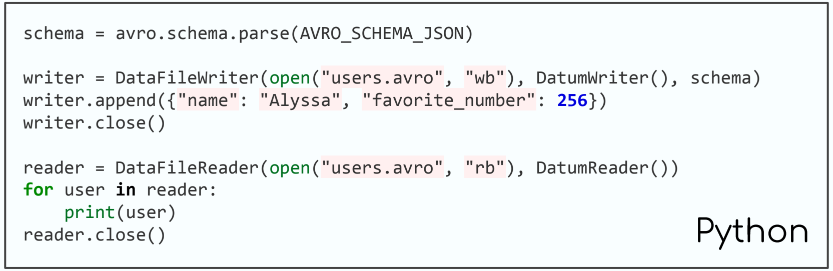
```json
{
   "namespace": "example.avro",
   "type": "record",
   "name": "User",
   "fields": [
      {"name": "name", "type": "string"},
      {"name": "favorite_number",  "type": ["null", "int"]},
      {"name": "favorite_color", "type": ["null", "string"]}
   ]
}
```

Schema



```java
@org.apache.avro.specific.AvroGenerated
public class User extends org.apache.avro.specific.SpecificRecordBase
implements org.apache.avro.specific.SpecificRecord {
  private java.lang.String name;
  private java.lang.Integer favorite_number;
  private java.lang.String favorite_color;

  public java.lang.String getName() {
    return name;
  }

  public void setName(java.lang.String value) {
    this.name = value;
  }

  …
```

Java



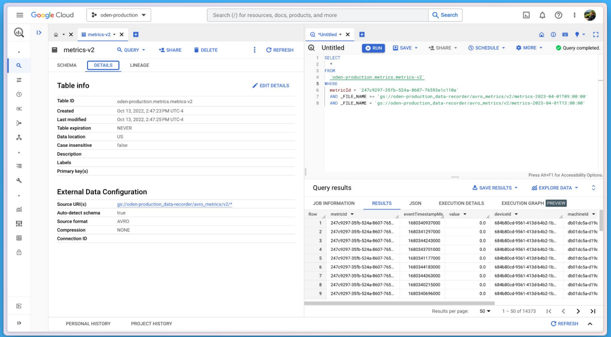
```python
schema = avro.schema.parse(AVRO_SCHEMA_JSON)

writer = DataFileWriter(open("users.avro", "wb"), DatumWriter(), schema)
writer.append({"name": "Alyssa", "favorite_number": 256})
writer.close()

reader = DataFileReader(open("users.avro", "rb"), DatumReader())
for user in reader:
    print(user)
reader.close()
```

Python

Apache Avro w/ GCP

Google Cloud Platform Specific

- Avro files in GCS can be queried via a BigQuery external table
- PubSub Topics can be assigned an Avro Schema and enforce payloads match
- "BigQuery Subscriptions" let you easily populate a "real" BigQuery table from an avro-encoded PubSub topic



metrics-v2

Table info

Table ID	oden-production.metrics.metrics-v2
Created	Oct 13, 2022, 2:47:23 PM UTC-4
Last modified	Oct 13, 2022, 2:47:25 PM UTC-4
Table expiration	NEVER
Data location	US
Case insensitive	false
Description	
Labels	
Primary key(s)	

External Data Configuration

Source URI(s)	gs://oden-production_data-recorder/avro_metrics/v2/*
Auto-detect schema	true
Source format	AVRO
Compression	NONE
Connection ID	

Untitled

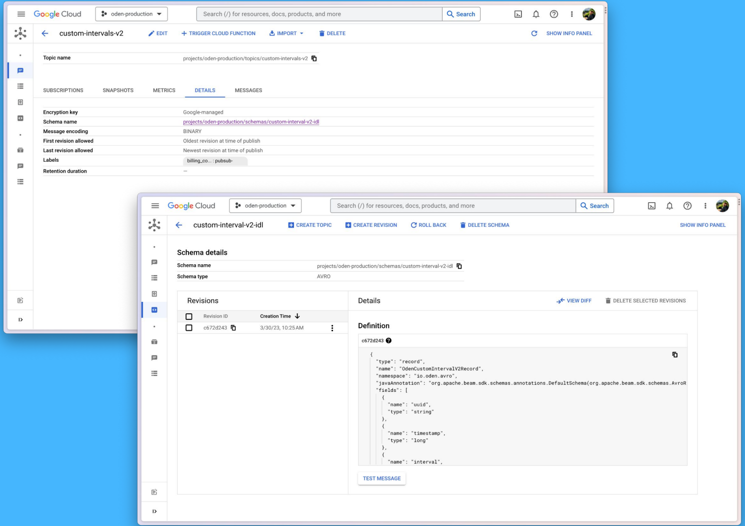
```
SELECT
  *
FROM
  `oden-production.metrics.metrics-v2`
WHERE
  metricId = '247c9297-35fb-524a-8607-76593e1c110a'
  AND _FILE_NAME >= 'gs://oden-production_data-recorder/avro_metrics/v2/metrics-2023-04-01T09:00:00'
  AND _FILE_NAME < 'gs://oden-production_data-recorder/avro_metrics/v2/metrics-2023-04-01T13:00:00'
```

Query results

Row	metricId	eventTimestampMs	value	deviceId	machineId
1	247c9297-35fb-524a-8607-765...	1680340937000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
2	247c9297-35fb-524a-8607-765...	1680341297000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
3	247c9297-35fb-524a-8607-765...	1680344243000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
4	247c9297-35fb-524a-8607-765...	1680343701000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
5	247c9297-35fb-524a-8607-765...	1680341177000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
6	247c9297-35fb-524a-8607-765...	1680344183000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
7	247c9297-35fb-524a-8607-765...	1680344363000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
8	247c9297-35fb-524a-8607-765...	1680340215000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a
9	247c9297-35fb-524a-8607-765...	1680340696000	0.0	684b80cd-9561-413d-b4b2-1b...	db01dc5a-d19a

Results per page: 50 1 – 50 of 14373

BigQuery Avro External Table



custom-intervals-v2

Topic name	projects/oden-production/topics/custom-intervals-v2

Encryption key	Google-managed
Schema name	projects/oden-production/schemas/custom-interval-v2-idl
Message encoding	BINARY
First revision allowed	Oldest revision at time of publish
Last revision allowed	Newest revision at time of publish
Labels	billing_co... : pubsub-
Retention duration	—

custom-interval-v2-idl

Schema details

Schema name	projects/oden-production/schemas/custom-interval-v2-idl
Schema type	AVRO

Revisions

☐	Revision ID	Creation Time ↓
☐	c672d243	3/30/23, 10:25 AM

Details

Definition

c672d243

```
{
  "type": "record",
  "name": "OdenCustomIntervalV2Record",
  "namespace": "io.oden.avro",
  "javaAnnotation": "org.apache.beam.sdk.schemas.annotations.DefaultSchema(org.apache.beam.sdk.schemas.AvroR
  "fields": [
    {
      "name": "uuid",
      "type": "string"
    },
    {
      "name": "timestamp",
      "type": "long"
    },
    {
      "name": "interval",
```

PubSub Topic w/ Avro Schema

Beam Schemas

- Language-independent type-system for records in Beam jobs
- Many classes can share a Beam Schema
- Special PTransforms for PCollections with schemas
- **Convert** PTransform lets you map between classes with the same Beam Schema

Java Beam (explicit)

```
Schema userSchema =
    Schema.of(
        Schema.Field.of("name", Schema.FieldType.STRING),
        Schema.Field.of("favorite_number", Schema.FieldType.INT64))
        Schema.Field.of("favorite_color", Schema.FieldType.STRING));
```

Java Beam (implicit)

```
@DefaultSchema(JavaBeanSchema.class)
public class User implements Serializable {
         public String name;
         public long favorite_number;
         public String favorite_color;

         public User() {}
}
```

Python Beam (implicit)

```
class User(typing.NamedTuple):
    name: str
    favorite_number: int
    favorite_color: str
```

Go Beam (implicit)

```
type User struct {
         Name string `beam:"name""`
         FavoriteColor string `beam:"favorite_color"`
         FavoriteNumber int64 `beam:"favorite_number"`
}
```

Making Them Work Together

Making Them Play Nice

Reading in Java

- We generate classes from Avro Schemas
- Reading Avros, both from PubSubIO and AvroIO (files), works out of the box!

```
{
  "type": "record",
  "name": "OdenMetricV2Record",
  "namespace": "io.oden.avro",
  "fields": [...]
}
```

Schema

```
@org.apache.avro.specific.AvroGenerated
public class OdenMetricV2Record extends
  org.apache.avro.specific.SpecificRecordBase implements
  org.apache.avro.specific.SpecificRecord {
  ...
```

Generated

```
pipeline
    .apply(
        PubsubIO.readAvrosWithBeamSchema(OdenMetricV2Record.class)
            .fromSubscription(options.getSourcePubsubSubscription()))
```

Reading PubSub

```
Pipeline
  .apply(
    AvroIO
      .read(OdenMetricV2Record.class)
      .from(filePattern)
      .withBeamSchemas(true))
```

Reading AvroIO

Making Them Play Nice

The Sandwich

- Our pipelines are "sandwiches"
  - bread is Avro Generated
  - meat is an internal POJO
- POJO shares a schema but has a public API for our Ptransforms so the Avro schemas are free to change without us needing to update our pipelines
- Using **Convert**, we can convert to our internal POJO

```
{
  "type": "record",
  "name": "OdenMetricV2Record",
  "namespace": "io.oden.avro",
  "fields": [...]
}
```

Schema

```
@org.apache.avro.specific.AvroGenerated
public class OdenMetricV2Record extends
  org.apache.avro.specific.SpecificRecordBase implements
  org.apache.avro.specific.SpecificRecord {
  ...
```

Generated

```
@DefaultSchema(JavaFieldSchema.class)
public class Metric implements Serializable {
  …
```

POJO

```
pipeline
    .apply(
        PubsubIO.readAvrosWithBeamSchema(OdenMetricV2Record.class)
            .fromSubscription(options.getSourcePubsubSubscription()))
    .apply(Convert.to(Metric.class))
```

Reading PubSub

```
Pipeline
  .apply(
    AvroIO
      .read(OdenMetricV2Record.class)
      .from(filePattern)
      .withBeamSchemas(true))
    .apply(Convert.to(Metric.class))
```

Reading AvroIO

Making Them Play Nice

Writing in Java

- Just before writing, we **Convert** back from the internal POJO to the Avro class
- In order to do this, the class itself needs a schema (not just the PCollection)
- Avro Schema JSON includes “javaAnnotation” field that sets the DefaultSchema to the AvroRecordSchema
- For PCollections of the Generated Classes, PubSubIO and AvroIO work out of the box.

```
{
  "type": "record",
  "name": "OdenMetricV2Record",
  "namespace": "io.oden.avro",
  "javaAnnotation":
"org.apache.beam.sdk.schemas.annotations.DefaultSchema(org.apache.beam.sdk.e
xtensions.avro.schemas.AvroRecordSchema.class)",
  "fields": [...]
}
```

Schema

```
@org.apache.beam.sdk.schemas.annotations.DefaultSchema(
  org.apache.beam.sdk.extensions.avro.schemas.AvroRecordSchema.class)
@org.apache.avro.specific.AvroGenerated
public class OdenMetricV2Record extends
  org.apache.avro.specific.SpecificRecordBase implements
  org.apache.avro.specific.SpecificRecord {
  ...
```

Generated

```
...
  .apply(Convert.to(OdenMetricV2Record.class))
  .apply(
    PubsubIO.writeAvros(OdenMetricV2Record.class)
      .to(options.getSinkPubsubTopic()))
```

Writing PubSubIO

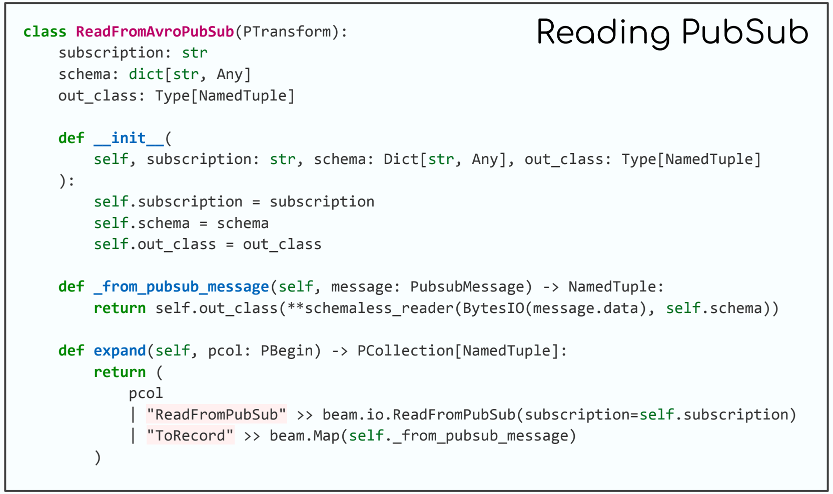
```
...
  .apply(Convert.to(OdenMetricV2Record.class))
  .apply(
    AvroIO.read(OdenMetricV2Record.class)
      .from(options.getSinkFilePattern())
      .withBeamSchemas(true))
```

Writing AvroIO

Making Them Play Nice

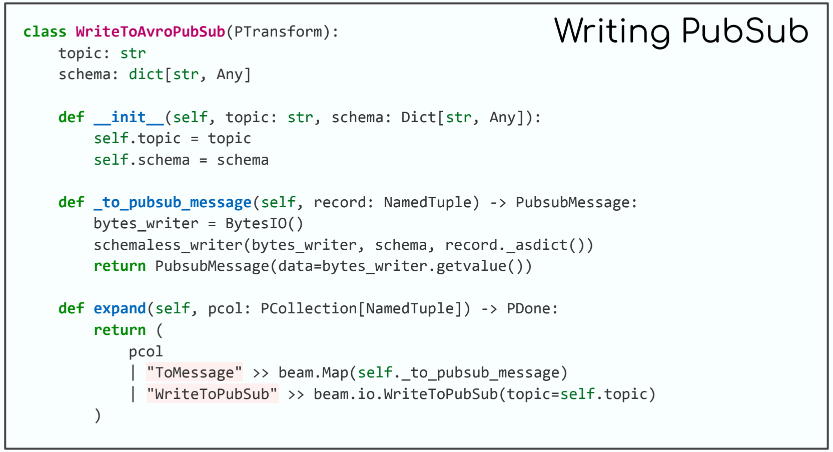
In Python

- Python is significantly less built out
- Out-of-the-box beam.io.avroio reads and writes files *with schemas*
- Anything schemasless requires a custom PTransform
- Converting immediately to the NamedTuple record is the best way I've found to ensure we get schema consistency
- Oden has now moved 100% off of the Python SDK due to performance issues



Reading PubSub

```python
class ReadFromAvroPubSub(PTransform):
    subscription: str
    schema: dict[str, Any]
    out_class: Type[NamedTuple]

    def __init__(
        self, subscription: str, schema: Dict[str, Any], out_class: Type[NamedTuple]
    ):
        self.subscription = subscription
        self.schema = schema
        self.out_class = out_class

    def _from_pubsub_message(self, message: PubsubMessage) -> NamedTuple:
        return self.out_class(**schemaless_reader(BytesIO(message.data), self.schema))

    def expand(self, pcol: PBegin) -> PCollection[NamedTuple]:
        return (
            pcol
            | "ReadFromPubSub" >> beam.io.ReadFromPubSub(subscription=self.subscription)
            | "ToRecord" >> beam.Map(self._from_pubsub_message)
        )
```



Writing PubSub

```python
class WriteToAvroPubSub(PTransform):
    topic: str
    schema: dict[str, Any]

    def __init__(self, topic: str, schema: Dict[str, Any]):
        self.topic = topic
        self.schema = schema

    def _to_pubsub_message(self, record: NamedTuple) -> PubsubMessage:
        bytes_writer = BytesIO()
        schemaless_writer(bytes_writer, schema, record._asdict())
        return PubsubMessage(data=bytes_writer.getvalue())

    def expand(self, pcol: PCollection[NamedTuple]) -> PDone:
        return (
            pcol
            | "ToMessage" >> beam.Map(self._to_pubsub_message)
            | "WriteToPubSub" >> beam.io.WriteToPubSub(topic=self.topic)
        )
```

GCP Cost Impact

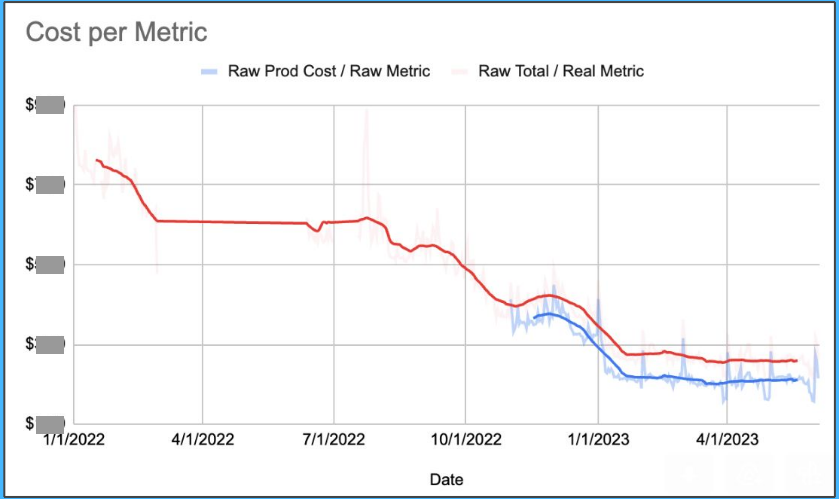
Getting the Most out of Pub/Sub

Oden's PubSub usage is high-element-count, low-element-size, high-write, and compressible.

In 2022, Oden moved our PubSub payloads from JSON to Avro.

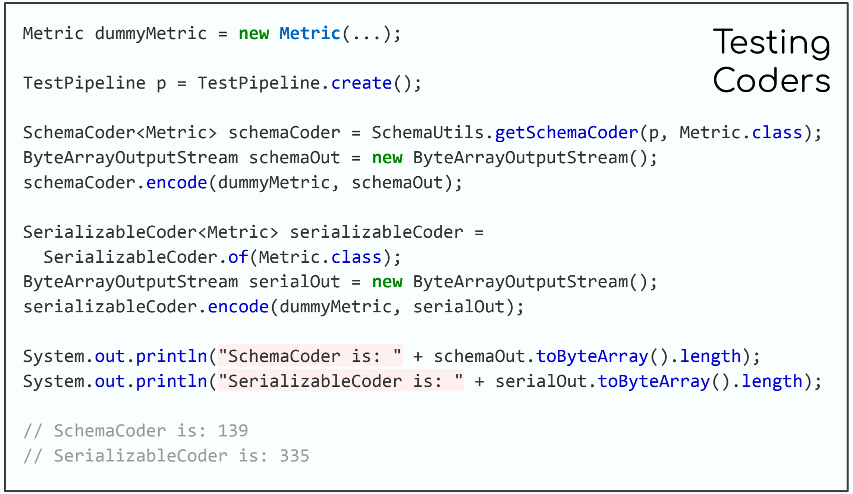
- message_size reduced 499B to 165B
- PubSub SKU "Message Delivery Basic" reduced by 76% (1:1 w/ size)
- Additional value was saved in Dataflow "Streaming data processed" and vCPUs
- 1KB minimum in pricing documentation does not matter when using PubSubIO which batches for you

*June 2023, w/ new BigQuery "Physical Storage Pricing" we see a **95% decrease** by using **BigQuery Subscriptions ➡ BigQuery** instead of **Dataflow ➡ GCS ➡ BigQuery External Table***



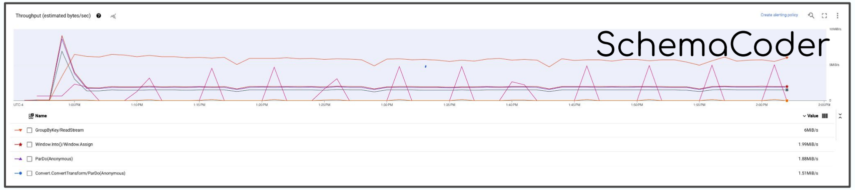
The SchemaCoder Impact

- A large cost driver for our Dataflow Jobs is “Streaming data processed”
- In our experience this is driven by:
  - Bytes Throughput between steps
  - Bytes used in State and Windows
- Using the SchemaCoder instead of the default SerializableCoder reduced messages from 335B to 139B (69%)
- We observe an almost 1:1 decrease in the “Streaming data processed” SKU
- You can (and should) write code to test how effective Coders are



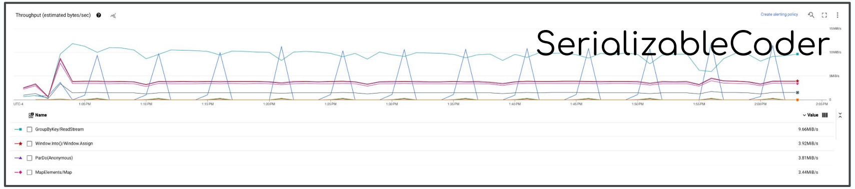
Testing Coders

```java
Metric dummyMetric = new Metric(...);

TestPipeline p = TestPipeline.create();

SchemaCoder<Metric> schemaCoder = SchemaUtils.getSchemaCoder(p, Metric.class);
ByteArrayOutputStream schemaOut = new ByteArrayOutputStream();
schemaCoder.encode(dummyMetric, schemaOut);

SerializableCoder<Metric> serializableCoder =
  SerializableCoder.of(Metric.class);
ByteArrayOutputStream serialOut = new ByteArrayOutputStream();
serializableCoder.encode(dummyMetric, serialOut);

System.out.println("SchemaCoder is: " + schemaOut.toByteArray().length);
System.out.println("SerializableCoder is: " + serialOut.toByteArray().length);

// SchemaCoder is: 139
// SerializableCoder is: 335
```



SchemaCoder

Name	Value
□ GroupByKey/ReadStream	6MiB/s
□ Window.Into()/Window.Assign	1.99MiB/s
□ ParDo(Anonymous)	1.88MiB/s
□ Convert.ConvertTransform/ParDo(Anonymous)	1.51MiB/s



SerializableCoder

Name	Value
□ GroupByKey/ReadStream	9.66MiB/s
□ Window.Into()/Window.Assign	3.92MiB/s
□ ParDo(Anonymous)	3.81MiB/s
□ MapElements/Map	3.44MiB/s

Flexible IO with Schemas

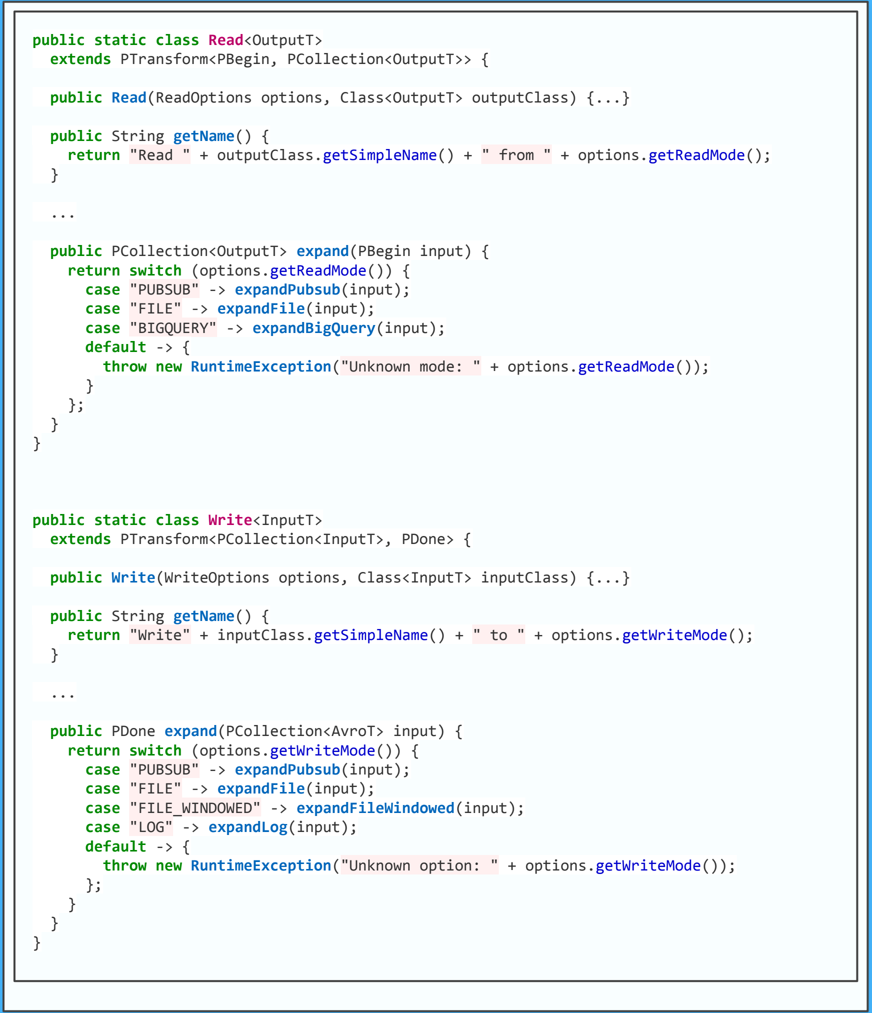
Oden’s Late Data Pipeline

Problem

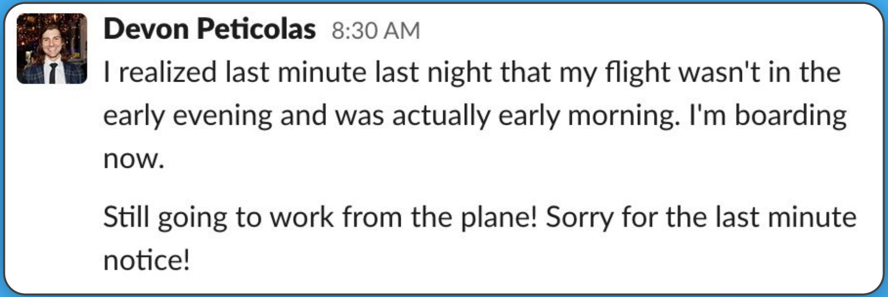
- Factories are bad at sending data.
  - Network outages
  - Wildly incorrect clocks
  - Bad local ISPs
- We sometimes get large bursts of very late data that overwhelms our streaming jobs

Solution

- All late data is sent to GCS
- All **Streaming** jobs are also **Batch** jobs
- We use one unified ReadIO and WriteIO that changes based on job arguments.
- Every night, Airflow orchestrates our entire streaming pipeline in “Batch <ode” on the late data in GCS.



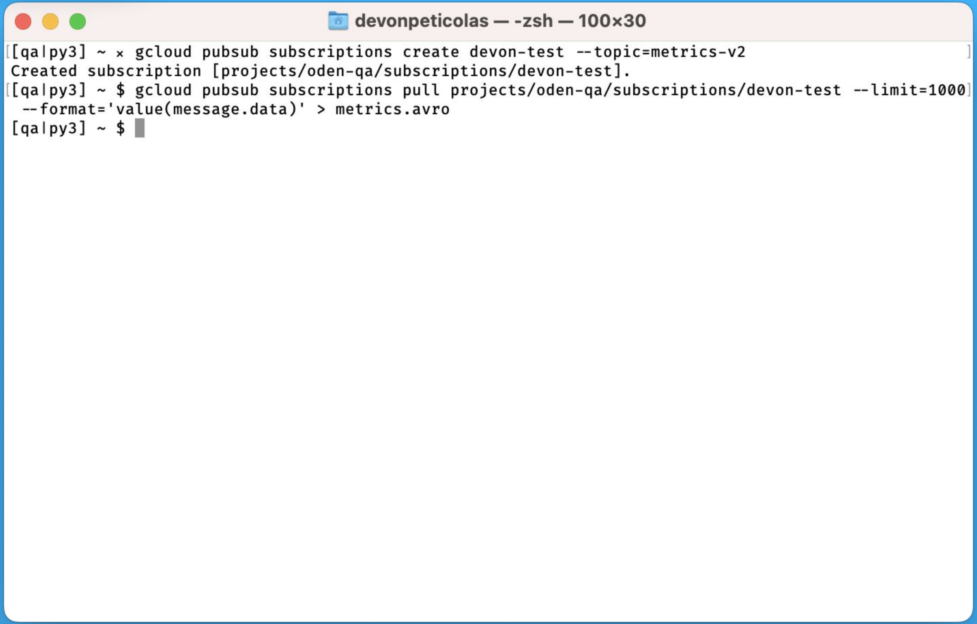
```
public static class Read<OutputT>
  extends PTransform<PBegin, PCollection<OutputT>> {

  public Read(ReadOptions options, Class<OutputT> outputClass) {...}

  public String getName() {
    return "Read " + outputClass.getSimpleName() + " from " + options.getReadMode();
  }

  ...

  public PCollection<OutputT> expand(PBegin input) {
    return switch (options.getReadMode()) {
      case "PUBSUB" -> expandPubsub(input);
      case "FILE" -> expandFile(input);
      case "BIGQUERY" -> expandBigQuery(input);
      default -> {
        throw new RuntimeException("Unknown mode: " + options.getReadMode());
      }
    };
  }
}


public static class Write<InputT>
  extends PTransform<PCollection<InputT>, PDone> {

  public Write(WriteOptions options, Class<InputT> inputClass) {...}

  public String getName() {
    return "Write" + inputClass.getSimpleName() + " to " + options.getWriteMode();
  }

  ...

  public PDone expand(PCollection<AvroT> input) {
    return switch (options.getWriteMode()) {
      case "PUBSUB" -> expandPubsub(input);
      case "FILE" -> expandFile(input);
      case "FILE_WINDOWED" -> expandFileWindowed(input);
      case "LOG" -> expandLog(input);
      default -> {
        throw new RuntimeException("Unknown option: " + options.getWriteMode());
      };
    }
  }
}
```



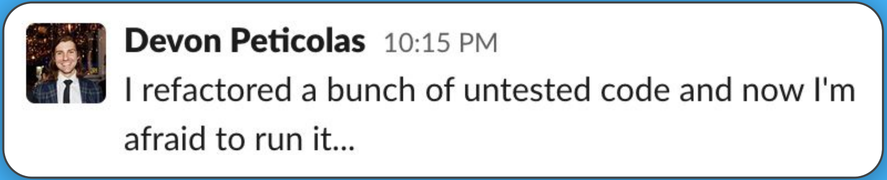
Devon Peticolas 8:30 AM

I realized last minute last night that my flight wasn't in the early evening and was actually early morning. I'm boarding now.

Still going to work from the plane! Sorry for the last minute notice!

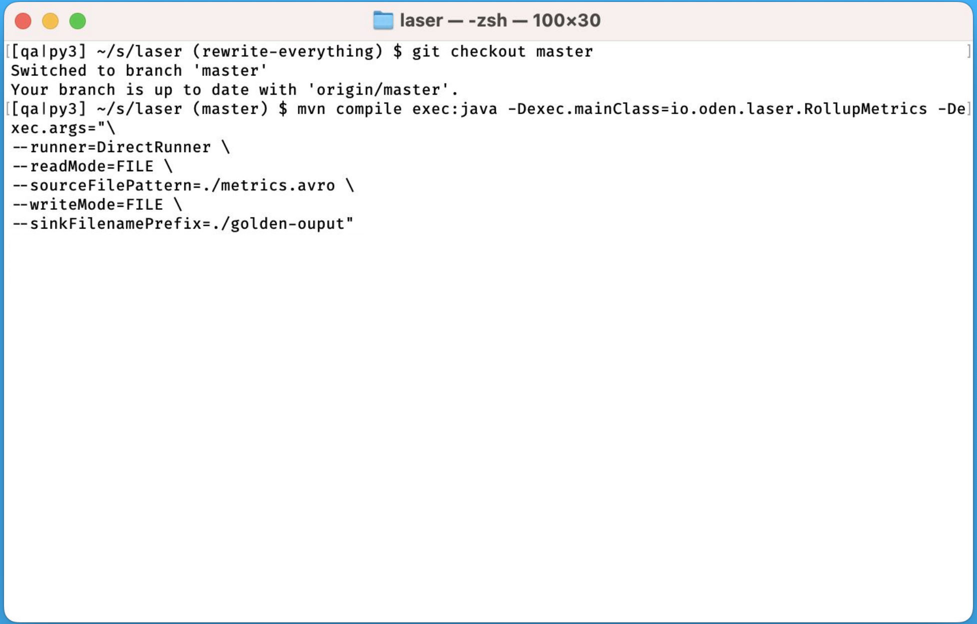


```
[qa|py3] ~ × gcloud pubsub subscriptions create devon-test --topic=metrics-v2
Created subscription [projects/oden-qa/subscriptions/devon-test].
[qa|py3] ~ $ gcloud pubsub subscriptions pull projects/oden-qa/subscriptions/devon-test --limit=1000
 --format='value(message.data)' > metrics.avro
[qa|py3] ~ $ 
```

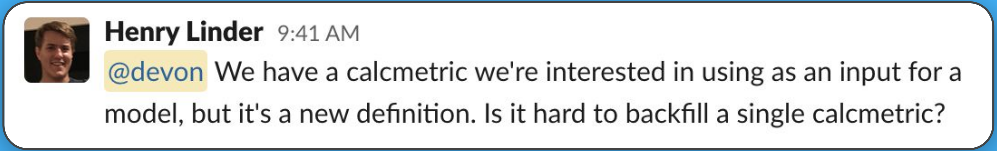


Devon Peticolas 10:15 PM

I refactored a bunch of untested code and now I'm afraid to run it...

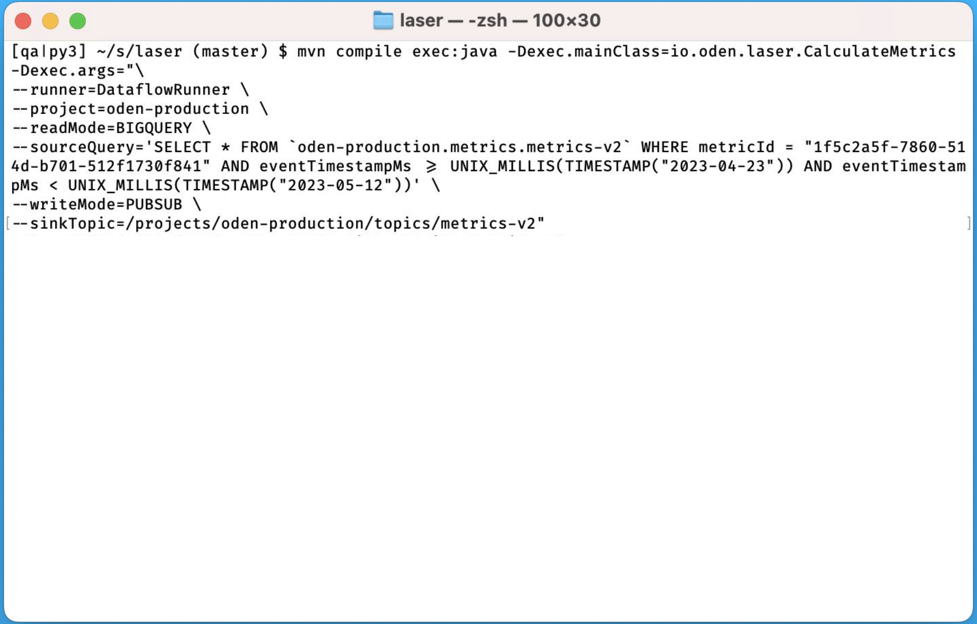


```
[qa|py3] ~/s/laser (rewrite-everything) $ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
[qa|py3] ~/s/laser (master) $ mvn compile exec:java -Dexec.mainClass=io.oden.laser.RollupMetrics -De
xec.args="\
--runner=DirectRunner \
--readMode=FILE \
--sourceFilePattern=./metrics.avro \
--writeMode=FILE \
--sinkFilenamePrefix=./golden-ouput"
```



Henry Linder 9:41 AM

@devon We have a calcmetric we're interested in using as an input for a model, but it's a new definition. Is it hard to backfill a single calcmetric?



```
[qa|py3] ~/s/laser (master) $ mvn compile exec:java -Dexec.mainClass=io.oden.laser.CalculateMetrics
-Dexec.args="\
--runner=DataflowRunner \
--project=oden-production \
--readMode=BIGQUERY \
--sourceQuery='SELECT * FROM `oden-production.metrics.metrics-v2` WHERE metricId = "1f5c2a5f-7860-51
4d-b701-512f1730f841" AND eventTimestampMs ≥ UNIX_MILLIS(TIMESTAMP("2023-04-23")) AND eventTimestam
pMs < UNIX_MILLIS(TIMESTAMP("2023-05-12"))' \
--writeMode=PUBSUB \
--sinkTopic=/projects/oden-production/topics/metrics-v2"
```

Avro + Schemas = Generic IO

By Avro-Generated Classes Having Schemas

- A unified ReadAvroIO can read non-Avro sources like BigQuery
- We can easily convert to an internal representation
- A unified WriteAvroIO can convert internal representation back to Avro

ReadAvro

```
public class ReadAvro<AvroT extends SpecificRecordBase>
   extends PTransform<PBegin, PCollection<AvroT>> {
 private final ReadAvroOptions options;
 private final Class<AvroT> avroClass;

 public ReadAvro(ReadAvroOptions options, Class<AvroT> avroClass) {...}

 public PCollection<AvroT> expand(PBegin input) {
   return switch (options.getReadMode()) {
     case "FILE" -> input.apply(
         AvroIO
             .read(avroClass)
             .from(options.getFilePattern())
             .withBeamSchemas(true));
     case "PUBSUB" -> input.apply(
         PubsubIO
             .readAvrosWithBeamSchema(avroClass)
             .fromSubscription(options.getSubscription()));
     case "BIGQUERY" -> input
         .apply(
             BigQueryIO
                 .readTableRowsWithSchema()
                 .fromQuery(options.getQuery()))
         .apply(Convert.to(avroClass));
     default -> {
       throw new RuntimeException("Unknown mode: " + options.getReadMode());
     }
   };
  }
  ...
```

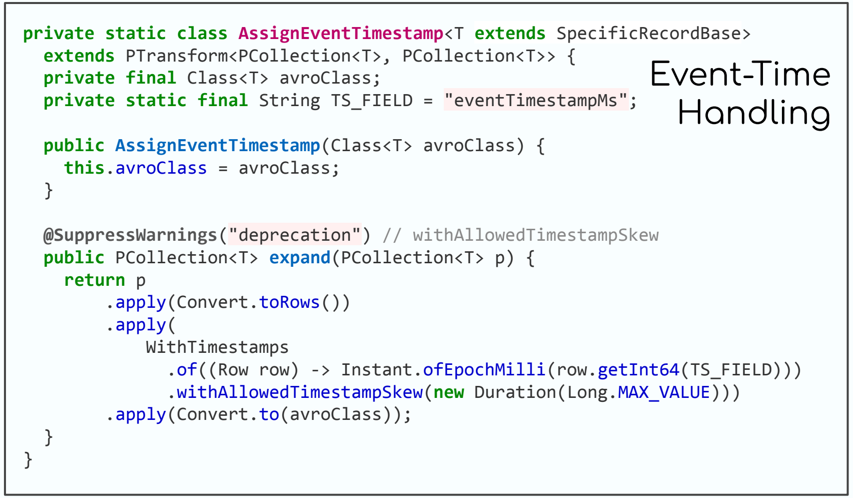
Job

```
public class MyJob {
  public interface MyJobOptions extends ReadAvroOptions {...}

  public static void main(String[] args) {
    MyJobOptions options = ...
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Metric> metrics = pipeline
      .apply(new ReadAvro(options, OdenMetricV2Record.class))
      .apply(Convert.to(Metric.class));
    ...
```

Avro + Schemas = Generic IO

Continued

- By using a generic AvroReadIO on avro classes that have schemas, we can convert them into Rows and do things like assign event time dynamically based on a field.
- This unifies batch and streaming event time assignment.
- In the past, all of our data types had to implement an interface, now they just need to share a field.



Event-Time Handling

```java
private static class AssignEventTimestamp<T extends SpecificRecordBase>
  extends PTransform<PCollection<T>, PCollection<T>> {
  private final Class<T> avroClass;
  private static final String TS_FIELD = "eventTimestampMs";

  public AssignEventTimestamp(Class<T> avroClass) {
    this.avroClass = avroClass;
  }

  @SuppressWarnings("deprecation") // withAllowedTimestampSkew
  public PCollection<T> expand(PCollection<T> p) {
    return p
        .apply(Convert.toRows())
        .apply(
            WithTimestamps
                .of((Row row) -> Instant.ofEpochMilli(row.getInt64(TS_FIELD)))
                .withAllowedTimestampSkew(new Duration(Long.MAX_VALUE)))
        .apply(Convert.to(avroClass));
  }
}
```

Avro and Beam Schemas - In Summary

- Avro is 👍
  - It's great at serializing/deserializing
  - It makes things small
  - It's easy to use in Java and Python
  - It's powerful when combined w/ Dataflow, PubSub, GCS, and BigQuery
- Beam Schemas are 👍
  - They give you access to some nice PTransforms (but mainly Convert)
- When using them together
  - Make a sandwich 🥪 with Convert
  - Add necessary decorators via the Avro Schema JSON
  - Avoid Python if possible
- Together they make Streaming jobs cheaper 💸
  - Avro is a cheap PubSub payload encoding
  - BigQuery subscriptions are (potentially) cheaper than a Dataflow job
  - SchemaCoder saves bytes over SerializableCoder
- Together they make IO flexible 🤸
  - AvroIO makes it easy to Read and Write from multiple sources
  - Converting things to rows removes the need for interfaces
  - They help Oden keep every streaming job trivially batch compatible

Devon Peticolas

QUESTIONS?

Find these slides at
github.com/x/slides/tree/master/beam-summit-2023

Find me at
devon@peticol.as
@doet@mastodon.social

BEAM SUMMIT
NYC 2023