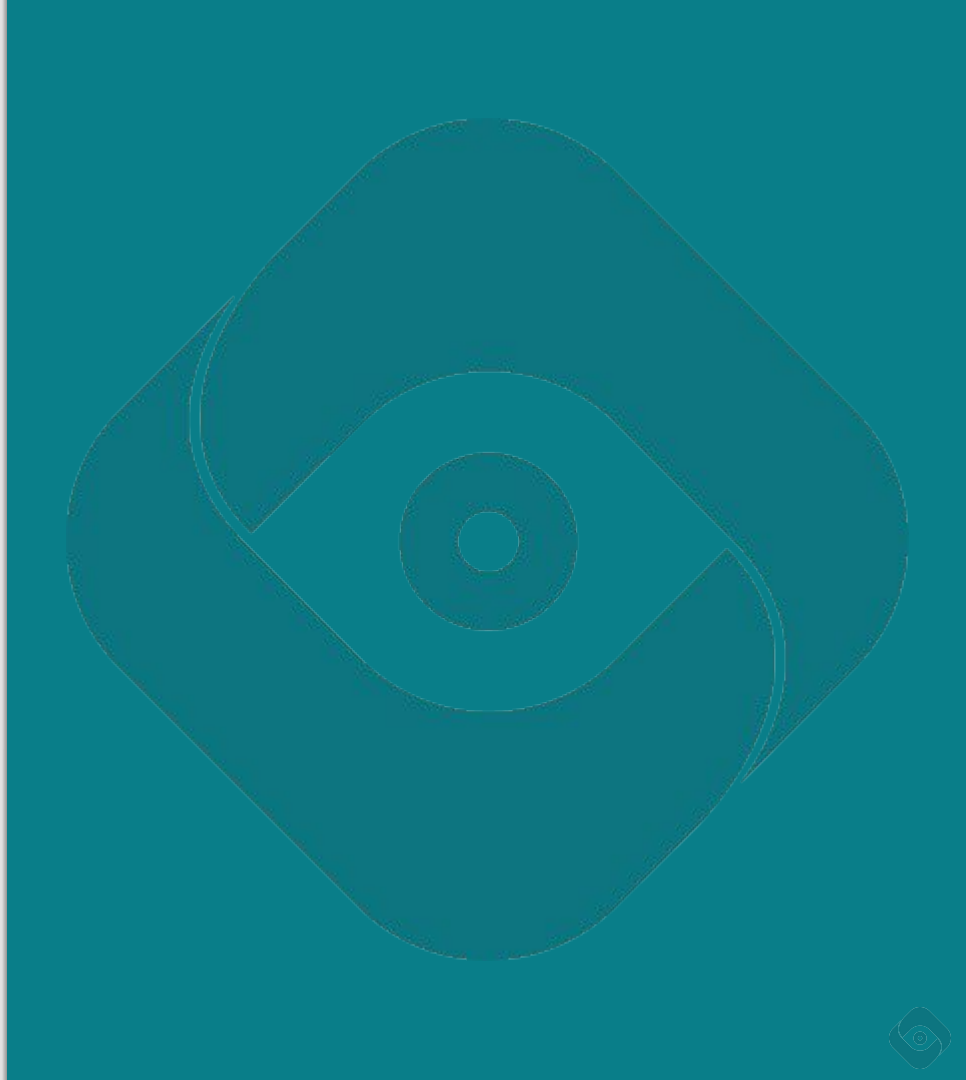
Spanning the Cloud and Factory with Apache Beam

Devon Peticolas - Oden Technologies

Devon Peticolas

Sr. Data Engineer



In This Talk

- Who is Oden (*very fast*)

In This Talk

- Who is Oden (*very fast*)
- What is Apache Beam (*medium fast*)

In This Talk

- Who is Oden (*very fast*)
- What is Apache Beam (*medium fast*)
- What are some *uncool* ways Oden is using Apache Beam

In This Talk

- Who is Oden (*very fast*)
- What is Apache Beam (*medium fast*)
- What are some *uncool* ways Oden is using Apache Beam
- What are some *cool* ways Oden is using Apache Beam

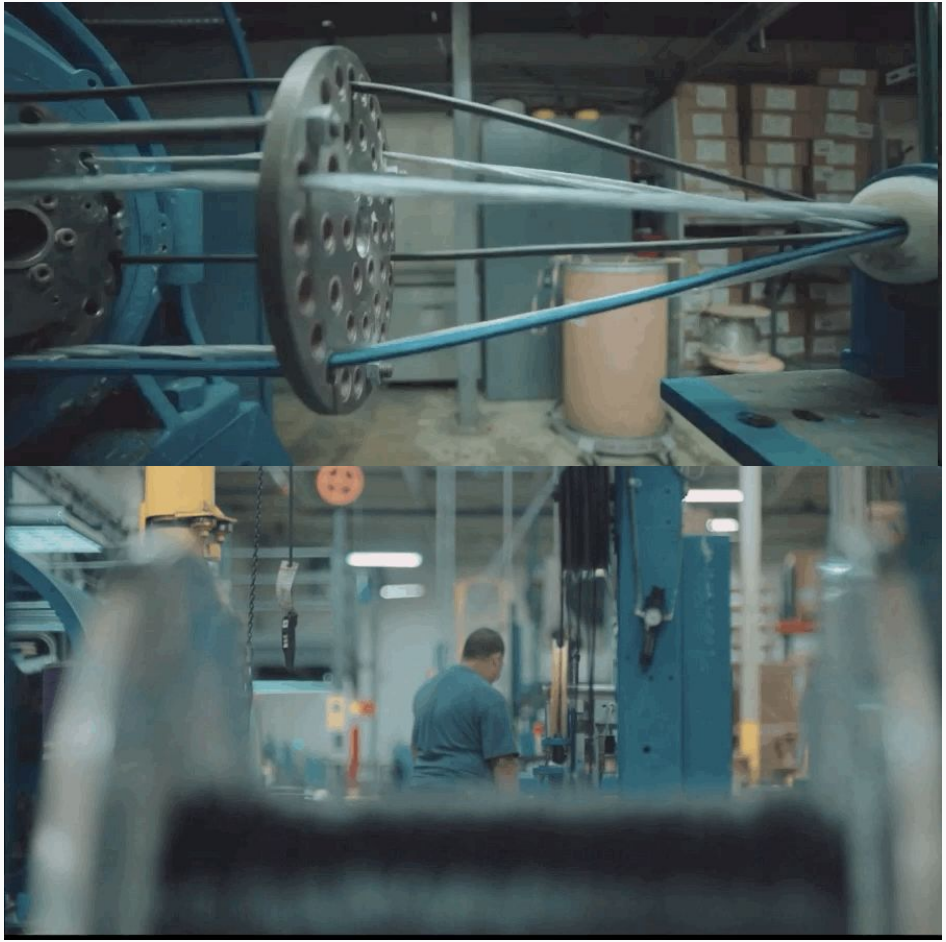
Who Are We

Oden's Customers

Medium to large manufacturers in plastics extrusion, injection molding, and metal stamping.

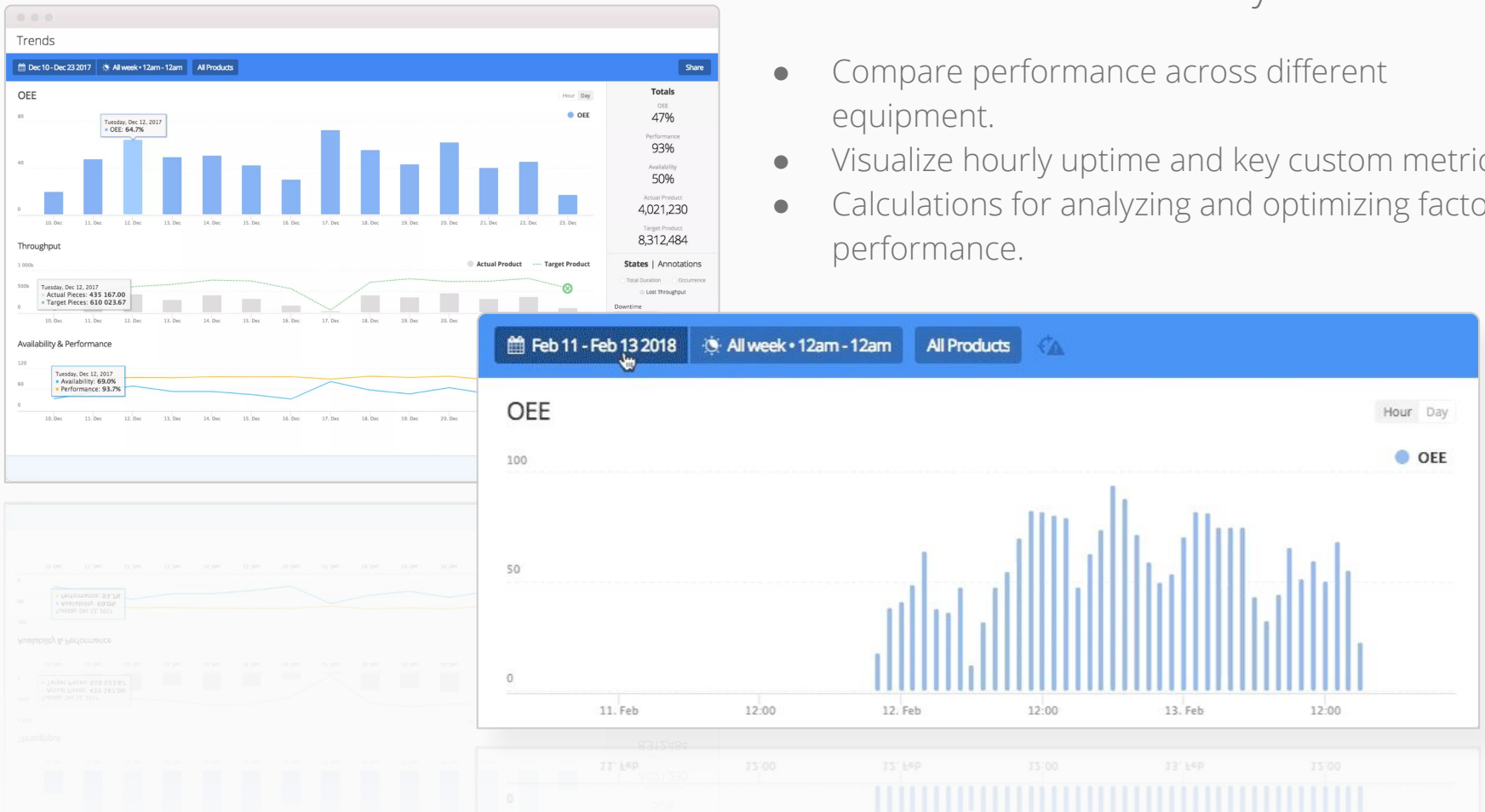
Process and Quality Engineers looking to centralize, analyze, and act on their data.

Plant managers who are looking to optimize logistics, output, and cost.



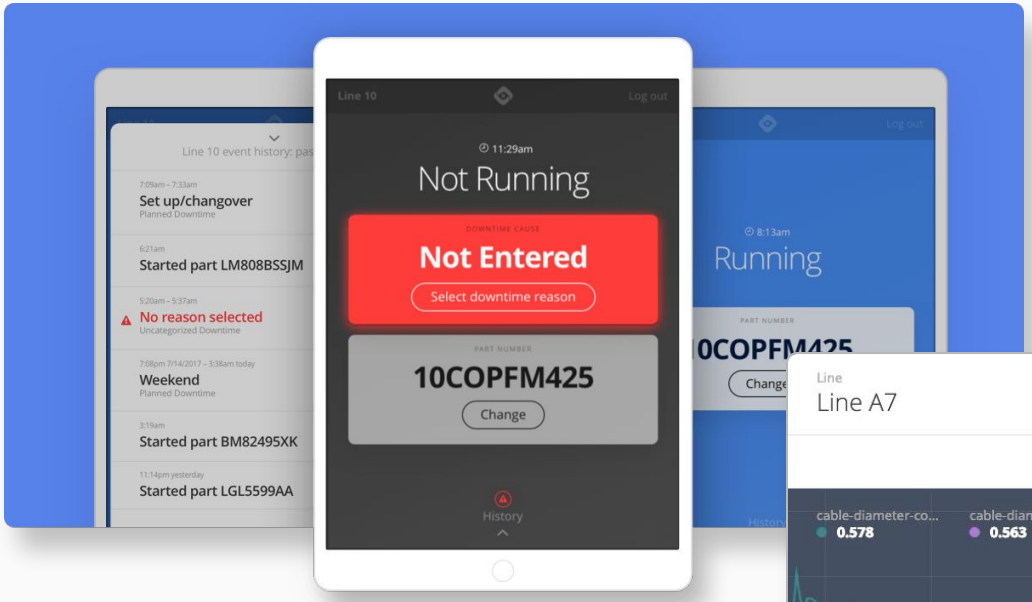
Interactive Time-series Analysis

- Compare performance across different equipment.
- Visualize hourly uptime and key custom metrics.
- Calculations for analyzing and optimizing factory performance.



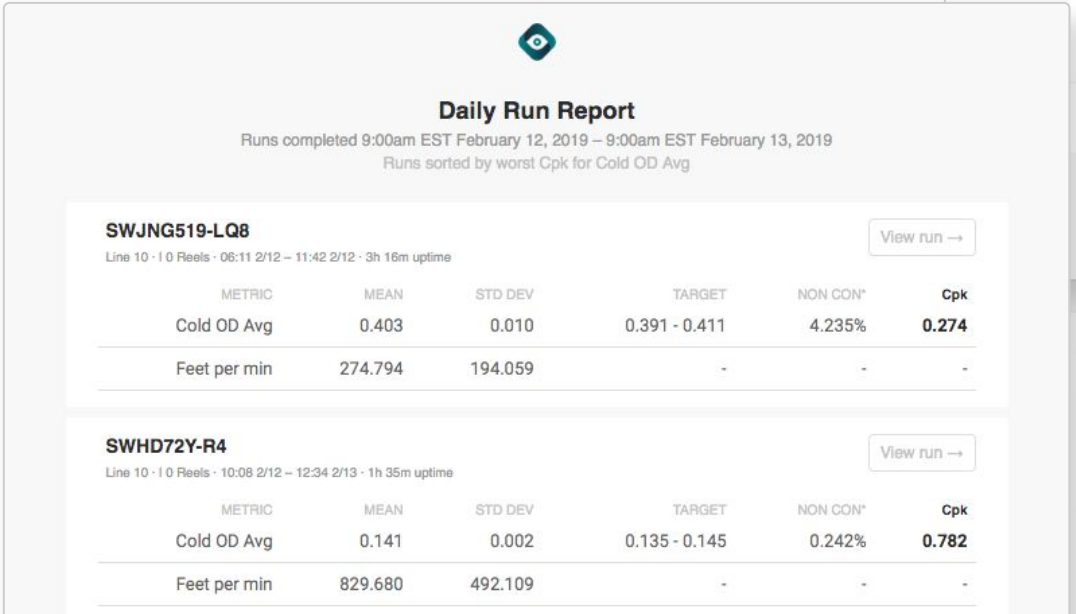
Real Time Manufacturing Data

- Streaming second-by-second metrics
- Interactive app that prompts on production state changes and collects user input.



Reporting and Alerting

- Daily summaries on key process metrics from continuous intervals of production work.
- Real-time email and text alerts on target violations and concerning trends.



Daily Run Report

Runs completed 9:00am EST February 12, 2019 – 9:00am EST February 13, 2019

Runs sorted by worst Cpk for Cold OD Avg

SWJNG519-LQ8

Line 10 · l 0 Reels · 06:11 2/12 – 11:42 2/12 · 3h 16m uptime

View run →

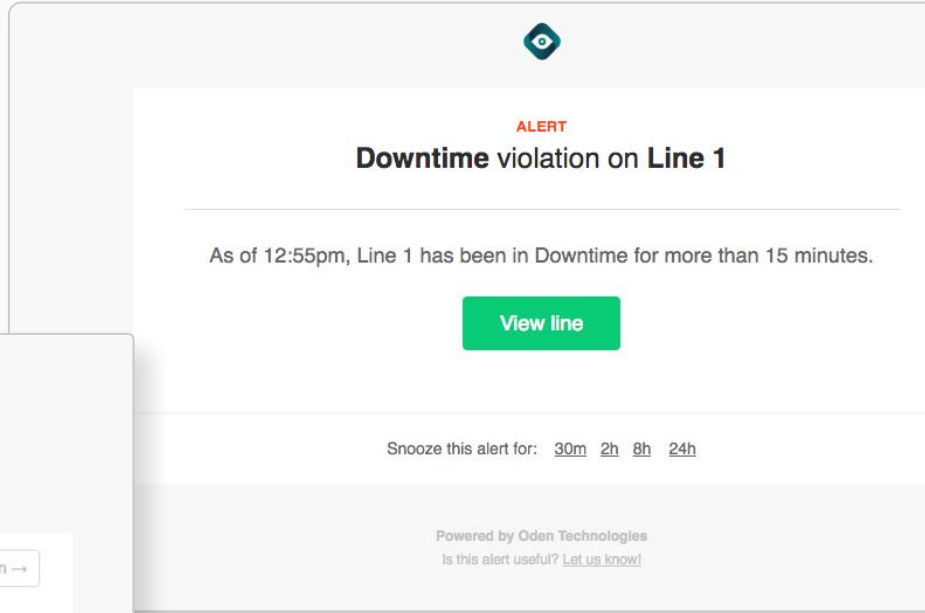
METRIC	MEAN	STD DEV	TARGET	NON CON*	Cpk
Cold OD Avg	0.403	0.010	0.391 - 0.411	4.235%	**0.274**
Feet per min	274.794	194.059	-	-	-

SWHD72Y-R4

Line 10 · l 0 Reels · 10:08 2/12 – 12:34 2/13 · 1h 35m uptime

View run →

METRIC	MEAN	STD DEV	TARGET	NON CON*	Cpk
Cold OD Avg	0.141	0.002	0.135 - 0.145	0.242%	**0.782**
Feet per min	829.680	492.109	-	-	-



ALERT

Downtime violation on **Line 1**

As of 12:55pm, Line 1 has been in Downtime for more than 15 minutes.

View line

Snooze this alert for: 30m 2h 8h 24h

Powered by Oden Technologies

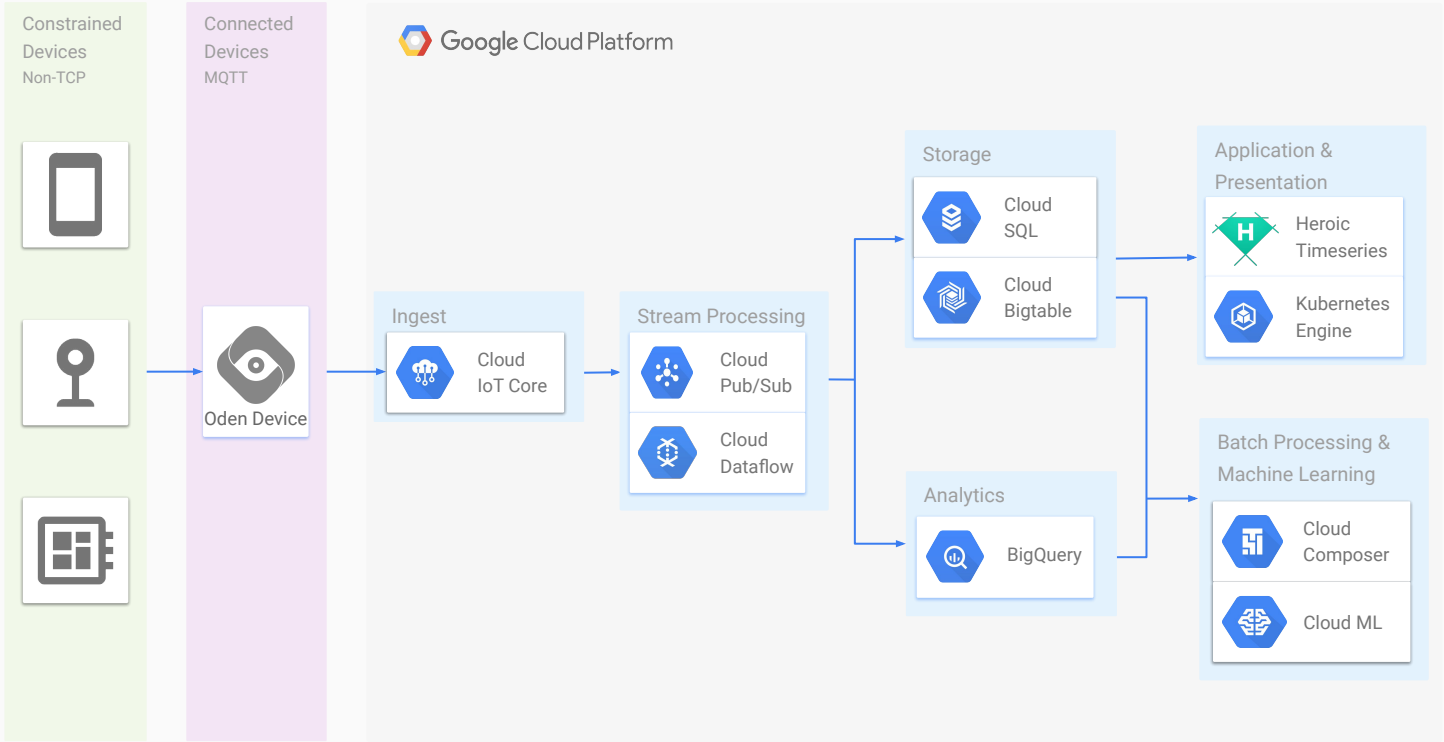
Is this alert useful? Let us know!

Technology - Hardware

Oden Device

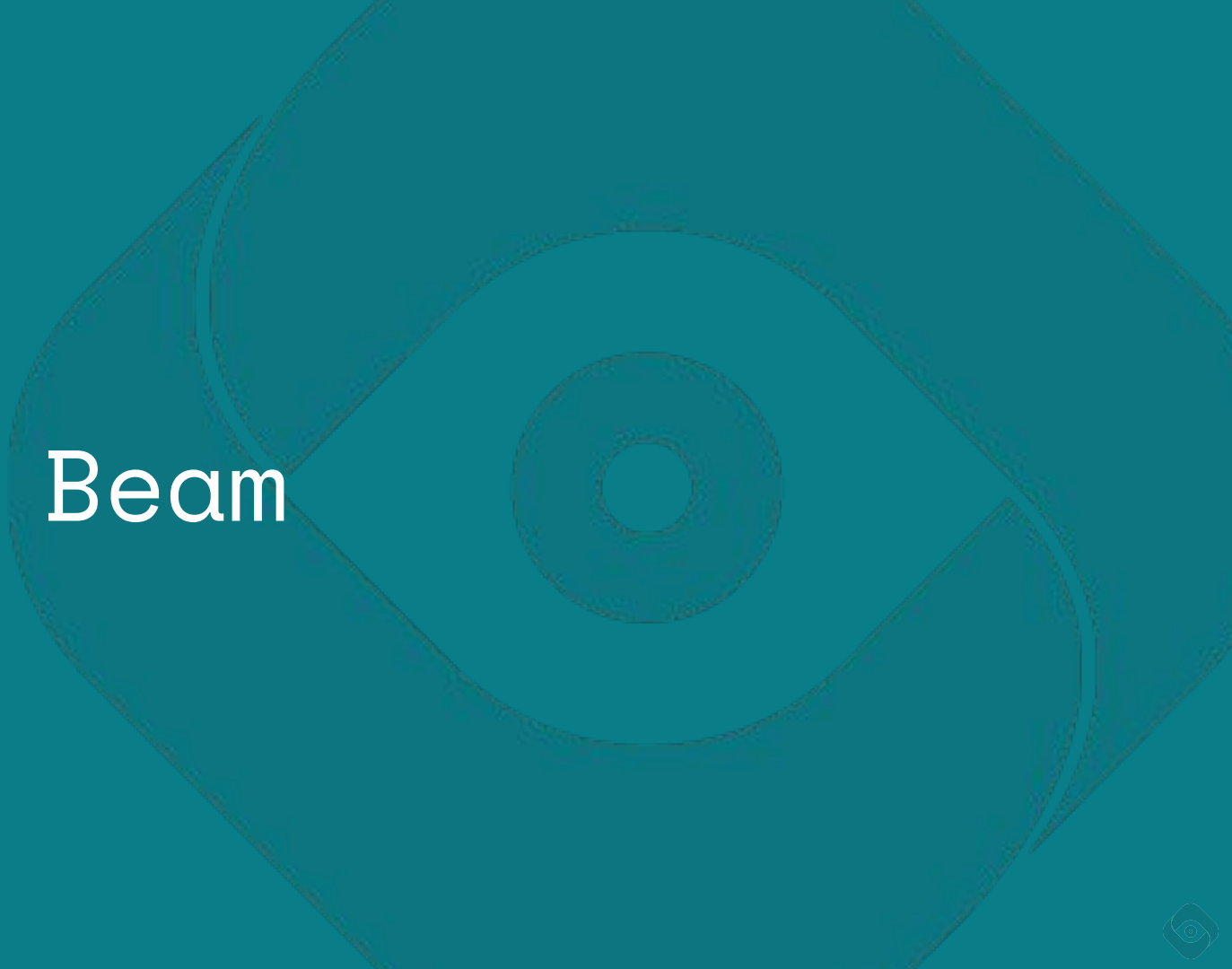
- Embedded Linux device
- Python docker containers that interact with industrial protocols over serial and ethernet
- Connects to cloud via wired, wifi, or cellular networks and Google IoT

Technology - Architecture



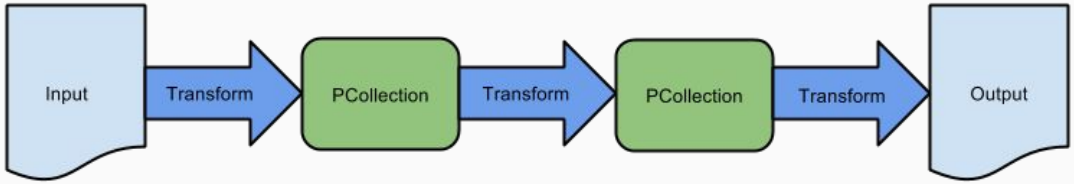
We do a lot with a small team!

Apache Beam



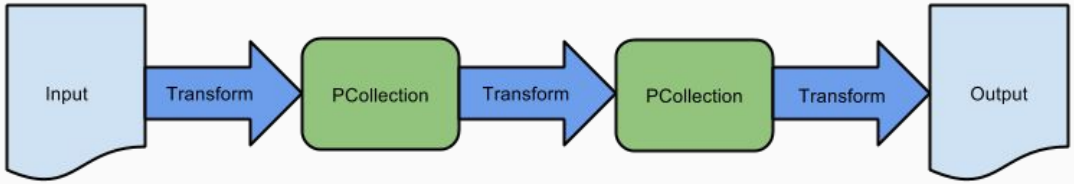
Unified Programming Model

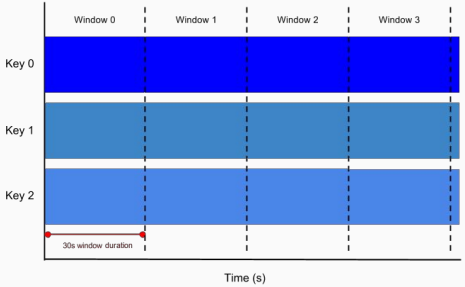
- Unifies inputs, outputs, and intermediate state as **PCollections** (bounded or unbounded) linked by **transforms** built into a **pipeline**.

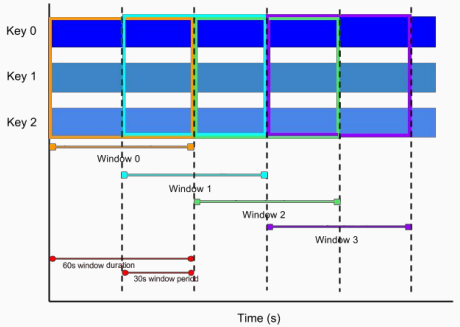


Unified Programming Model

- Unifies inputs, outputs, and intermediate state as **PCollections** (bounded or unbounded) linked by **transforms** built into a **pipeline**.
- Supports streaming joins, group-bys, stepping and sliding windows, and global state.

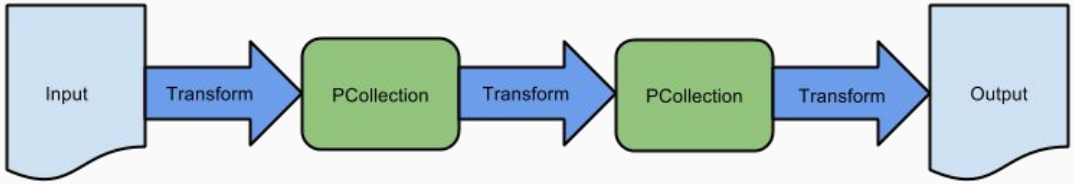


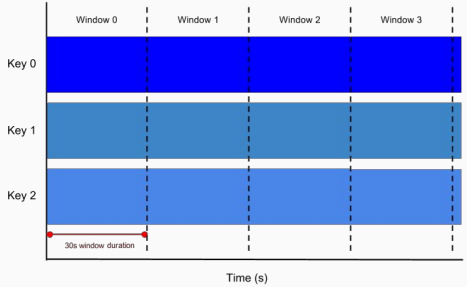


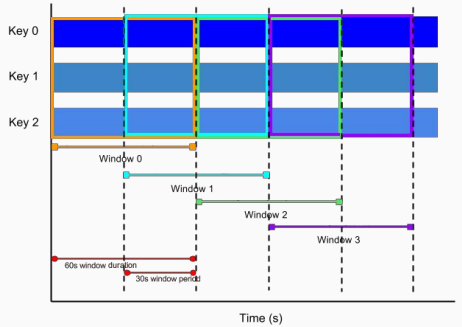


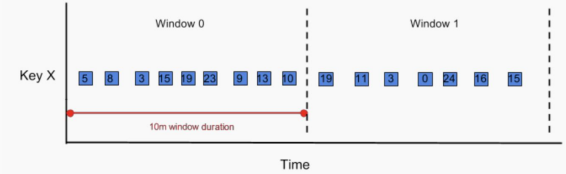
Unified Programming Model

- Unifies inputs, outputs, and intermediate state as **PCollections** (bounded or unbounded) linked by **transforms** built into a **pipeline**.
- Supports streaming joins, group-bys, stepping and sliding windows, and global state.
- Offers fine-grained tooling around handling late data.









8.4.1.1. Accumulating mode

If our trigger is set to accumulating mode, the trigger emits the following values each time it fires. Keep in mind that the trigger fires every time three elements arrive:

```
First trigger firing:  [5, 8, 3]
Second trigger firing: [5, 8, 3, 15, 19, 23]
Third trigger firing:  [5, 8, 3, 15, 19, 23, 9, 13, 10]
```

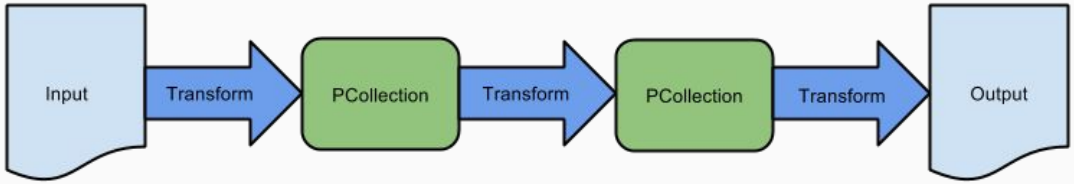
8.4.1.2. Discarding mode

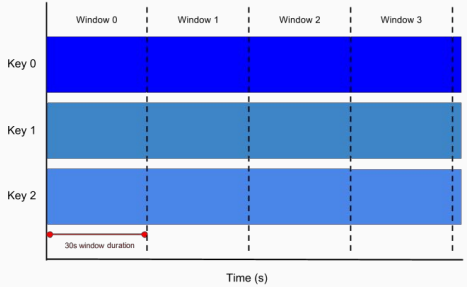
If our trigger is set to discarding mode, the trigger emits the following values on each firing:

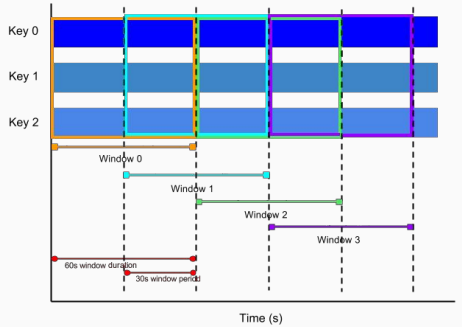
```
First trigger firing:  [5, 8, 3]
Second trigger firing:           [15, 19, 23]
Third trigger firing:                        [9, 13, 10]
```

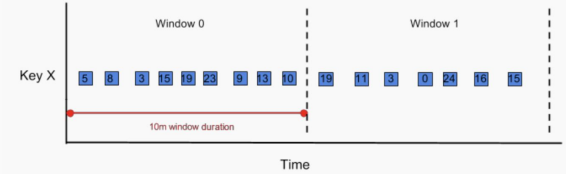
Unified Programming Model

- Unifies inputs, outputs, and intermediate state as **PCollections** (bounded or unbounded) linked by **transforms** built into a **pipeline**.
- Supports streaming joins, group-bys, stepping and sliding windows, and global state.
- Offers fine-grained tooling around handling late data.
- Can be executed over both batch and streaming data.









8.4.1.1. Accumulating mode

If our trigger is set to accumulating mode, the trigger emits the following values each time it fires. Keep in mind that the trigger fires every time three elements arrive:

```
First trigger firing:  [5, 8, 3]
Second trigger firing: [5, 8, 3, 15, 19, 23]
Third trigger firing:  [5, 8, 3, 15, 19, 23, 9, 13, 10]
```

8.4.1.2. Discarding mode

If our trigger is set to discarding mode, the trigger emits the following values on each firing:

```
First trigger firing:  [5, 8, 3]
Second trigger firing:           [15, 19, 23]
Third trigger firing:                        [9, 13, 10]
```

Java and Python SDKs

Java

```
public class MinimalWordCount {
  public static void main(String[] args) {
    PipelineOptions options = PipelineOptionsFactory.create();
    Pipeline p = Pipeline.create(options);
    p.apply(TextIO.read().from("gs://apache-beam-samples/shakespeare/*"))
      .apply(
          FlatMapElements.into(TypeDescriptors.strings())
          .via((String word) -> Arrays.asList(word.split("[^\\p{L}]+"))))
      .apply(Filter.by((String word) -> !word.isEmpty()))
      .apply(Count.perElement())
      .apply(
          MapElements.into(TypeDescriptors.strings())
          .via(
            (KV<String, Long> wordCount) ->
            wordCount.getKey() + ": " + wordCount.getValue()))
      .apply(TextIO.write().to("wordcounts"));
    p.run().waitUntilFinish();
  }
}
```

Python

```
def run(argv=None):
  pipeline_options = PipelineOptions(pipeline_args)
  pipeline_options.view_as(SetupOptions).save_main_session = True
  with beam.Pipeline(options=pipeline_options) as p:
    lines = p | ReadFromText("gs://apache-beam-samples/shakespeare/*")
    counts = (
        lines
        | 'Split' >> (beam.FlatMap(lambda x: re.findall(r'[A-Za-z\']+', x))
                      .with_output_types(unicode))
        | 'PairWithOne' >> beam.Map(lambda x: (x, 1))
        | 'GroupAndSum' >> beam.CombinePerKey(sum))
    def format_result(word_count):
      (word, count) = word_count
      return '%s: %s' % (word, count)
    output = counts | 'Format' >> beam.Map(format_result)
    output | WriteToText(known_args.output)
```

Java and Python SDKs

Multiple Runners!

Multiple Runners (reality)

(click to expand details)

What is being computed?

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark	Apache Apex	Apache Gearpump	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo	Hazelcast Jet
ParDo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GroupByKey	✓	✓	✓	~	✓	✓	✓	✓	✓	✓	✓	✓
Flatten	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Combine	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Composite Transforms	✓	~	~	~	~	~	✓	✓	~	~	✓	~
Side Inputs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	~
Source API	✓	✓	✓	✓	✓	✓	~	✓	✓	✓	✓	✓
Splittable DoFn (SDF)	~	✓	✓	~	~	~	×	×	×	~	×	×
Metrics	~	~	~	~	×	×	~	~	~	~	×	~
Stateful Processing	✓	~	~	×	~	×	~	~	~	~	×	~

(click to expand details)

Where in event time?

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark	Apache Apex	Apache Gearpump	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo	Hazelcast Jet
Global windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Fixed windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sliding windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Session windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Custom windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Custom merging windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Timestamp control	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

(click to expand details)

When in processing time?

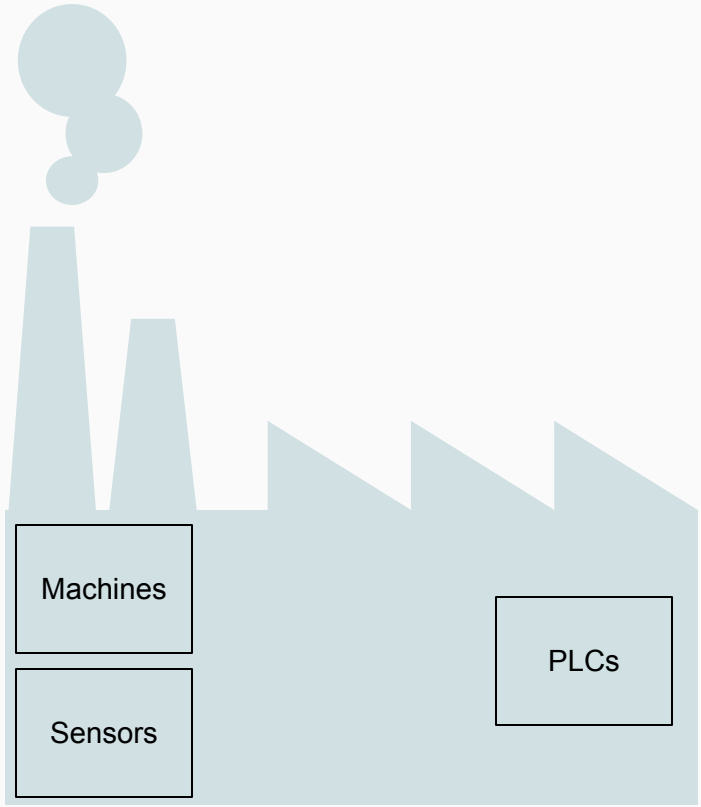
	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark	Apache Apex	Apache Gearpump	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo	Hazelcast Jet
Configurable triggering	✓	✓	✓	×	✓	×	×	✓	✓	✓	✓	✓
Event-time triggers	✓	✓	✓	×	✓	✓	×	✓	✓	✓	✓	✓
Processing-time triggers	✓	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓
Count triggers	✓	✓	✓	×	✓	×	×	✓	✓	✓	✓	✓
[Meta]data driven triggers	× (BEAM-101)	×	×	×	×	×	×	×	×	×	×	×
Composite triggers	✓	✓	✓	×	✓	×	×	✓	✓	✓	✓	✓
Allowed lateness	✓	✓	✓	×	✓	✓	×	✓	✓	✓	✓	✓
Timers	✓	~	~	×	×	×	×	~	~	~	×	~

https://beam.apache.org/documentation/runners/capability-matrix/

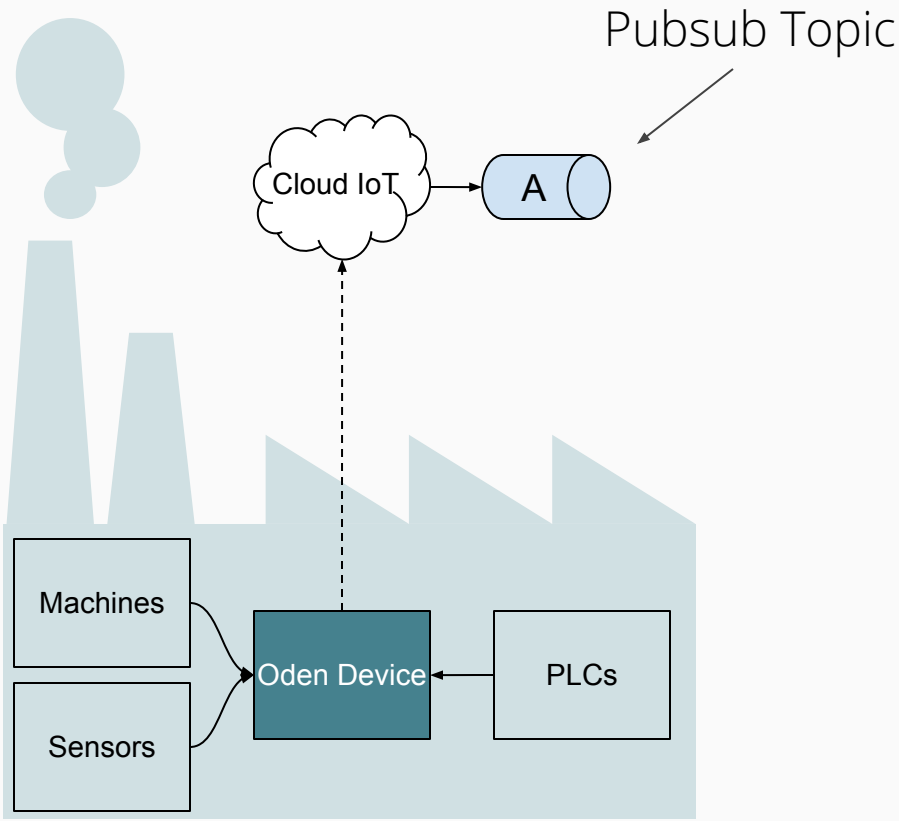
Uncool Ways Oden Uses Apache Beam

Oden's Streaming Data Pipeline

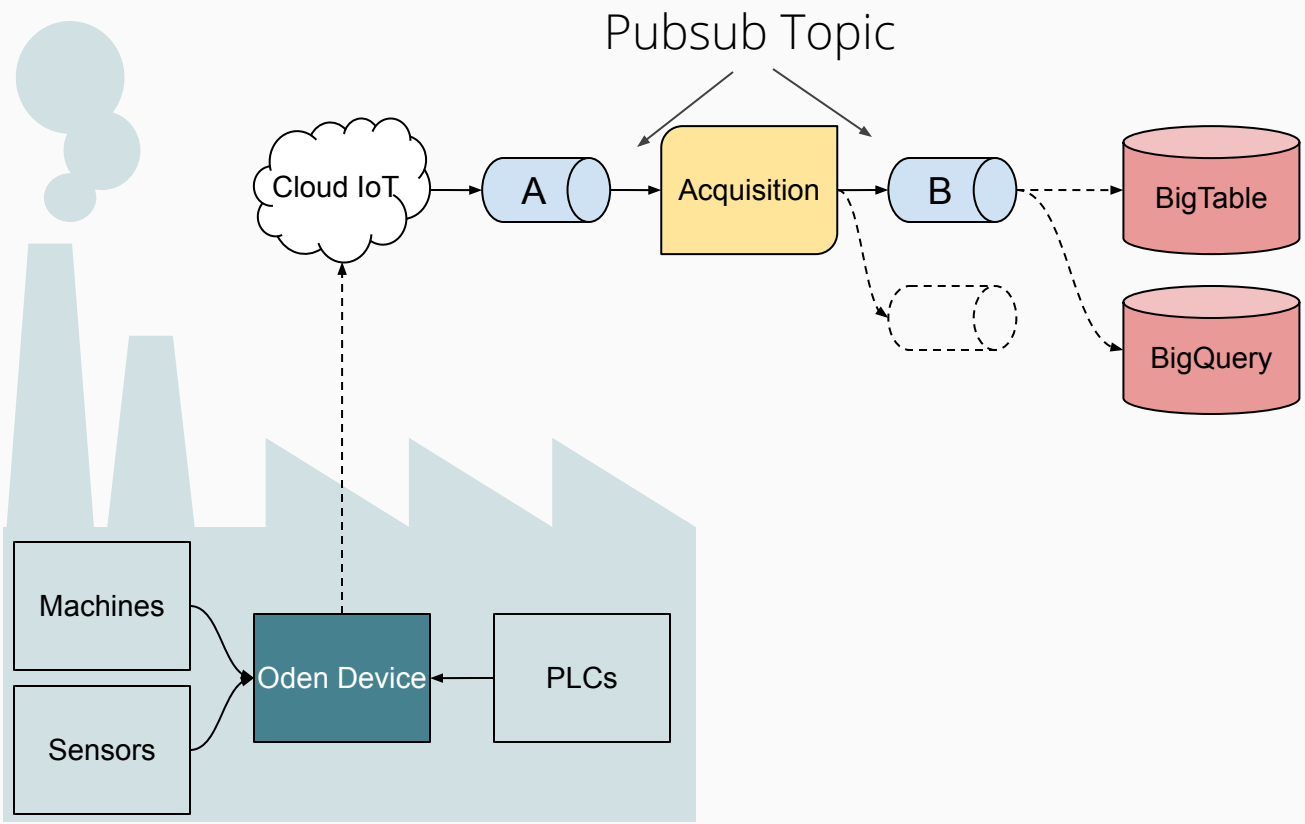
Oden's Streaming Data Pipeline



Oden's Streaming Data Pipeline



Oden's Streaming Data Pipeline



Acquisition

```java
public class Acquisition {
  private static final Logger LOG = LoggerFactory.getLogger(Acquisition.class);

  public interface Options {...}

  public static class EventPartitionFn implements PartitionFn<Event> {
    @Override
    public int partitionFor(Event event, int numPartitions) {
      switch (event.route) {
        case "/metric":
          return 1;
        case "/run":
          return 1;
        default:
          return 0;
      }
    }
  }

  public static void main(String[] args) {
    // Parse the user options passed from the command-line
    Options options = PipelineOptionsFactory.fromArgs(args).as(Options.class);
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Event> events =
        pipeline.apply("ReadEvents", EventIO.readJsonArrays(options, Event.class));

    // Sort the events by their route
    PCollectionList<Event> eventPartitions =
        events.apply("PartitionEvents", Partition.of(3, new EventPartitionFn()));

    // Transform the metric events into metrics and write them to the metrics topic.
    eventPartitions.get(0)
        .apply("MapEventsToMetrics",
            MapElements.into(TypeDescriptor.of(Metric.class)).via((event) -> new Metric(event)))
        .apply("WriteMetrics",
            EventIO.writeJsons(options.getSinkMetricsPubsubTopic(), Metric.class));

    // Transform the run events into runs and write them to the runs topic.
    eventPartitions.get(1)
        .apply("MapEventsToRuns",
            MapElements.into(TypeDescriptor.of(Run.class)).via((event) -> new Run(event)))
        .apply("WriteRuns",
            EventIO.writeJsons(options.getSinkRunsPubsubTopic(), Run.class));

    // Log the ones we don't recognize...
    eventPartitions.get(2)
        .apply("LogUnknowns", MapElements.into(TypeDescriptors.strings()).via((event) -> {
          String s = event.toString();
          Acquisition.LOG.error("Unrecognized event: " + s);
          return s;
        }));
    pipeline.run();
```

Acquisition

1. Read PubSub
2. Decode json as “Event”

```java
public class Acquisition {
  private static final Logger LOG = LoggerFactory.getLogger(Acquisition.class);

  public interface Options {...}

  public static class EventPartitionFn implements PartitionFn<Event> {
    @Override
    public int partitionFor(Event event, int numPartitions) {
      switch (event.route) {
        case "/metric":
          return 1;
        case "/run":
          return 1;
        default:
          return 0;
      }
    }
  }

  public static void main(String[] args) {
    // Parse the user options passed from the command-line
    Options options = PipelineOptionsFactory.fromArgs(args).as(Options.class);
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Event> events =
        pipeline.apply("ReadEvents", EventIO.readJsonArrays(options, Event.class));

    // Sort the events by their route
    PCollectionList<Event> eventPartitions =
        events.apply("PartitionEvents", Partition.of(3, new EventPartitionFn()));

    // Transform the metric events into metrics and write them to the metrics topic.
    eventPartitions.get(0)
        .apply("MapEventsToMetrics",
            MapElements.into(TypeDescriptor.of(Metric.class)).via((event) -> new Metric(event)))
        .apply("WriteMetrics",
            EventIO.writeJsons(options.getSinkMetricsPubsubTopic(), Metric.class));

    // Transform the run events into runs and write them to the runs topic.
    eventPartitions.get(1)
        .apply("MapEventsToRuns",
            MapElements.into(TypeDescriptor.of(Run.class)).via((event) -> new Run(event)))
        .apply("WriteRuns",
            EventIO.writeJsons(options.getSinkRunsPubsubTopic(), Run.class));

    // Log the ones we don't recognize...
    eventPartitions.get(2)
        .apply("LogUnknowns", MapElements.into(TypeDescriptors.strings()).via((event) -> {
          String s = event.toString();
          Acquisition.LOG.error("Unrecognized event: " + s);
          return s;
        }));
    pipeline.run();
```

(1, 2)

Acquisition

1. Read PubSub
2. Decode json as “Event”
3. Sort it into correct PCollection

```java
public class Acquisition {
  private static final Logger LOG = LoggerFactory.getLogger(Acquisition.class);

  public interface Options {...}

  public static class EventPartitionFn implements PartitionFn<Event> {
    @Override
    public int partitionFor(Event event, int numPartitions) {
      switch (event.route) {
        case "/metric":
          return 1;
        case "/run":
          return 1;
        default:
          return 0;
      }
    }
  }

  public static void main(String[] args) {
    // Parse the user options passed from the command-line
    Options options = PipelineOptionsFactory.fromArgs(args).as(Options.class);
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Event> events =
        pipeline.apply("ReadEvents", EventIO.readJsonArrays(options, Event.class));

    // Sort the events by their route
    PCollectionList<Event> eventPartitions =
        events.apply("PartitionEvents", Partition.of(3, new EventPartitionFn()));

    // Transform the metric events into metrics and write them to the metrics topic.
    eventPartitions.get(0)
        .apply("MapEventsToMetrics",
            MapElements.into(TypeDescriptor.of(Metric.class)).via((event) -> new Metric(event)))
        .apply("WriteMetrics",
            EventIO.writeJsons(options.getSinkMetricsPubsubTopic(), Metric.class));

    // Transform the run events into runs and write them to the runs topic.
    eventPartitions.get(1)
        .apply("MapEventsToRuns",
            MapElements.into(TypeDescriptor.of(Run.class)).via((event) -> new Run(event)))
        .apply("WriteRuns",
            EventIO.writeJsons(options.getSinkRunsPubsubTopic(), Run.class));

    // Log the ones we don't recognize...
    eventPartitions.get(2)
        .apply("LogUnknowns", MapElements.into(TypeDescriptors.strings()).via((event) -> {
          String s = event.toString();
          Acquisition.LOG.error("Unrecognized event: " + s);
          return s;
        }));
    pipeline.run();
```

(1, 2)

(3)

Acquisition

1. Read PubSub
2. Decode json as “Event”
3. Sort it into correct PCollection
4. Encode metrics as Json
5. Write metrics PubSub

```java
public class Acquisition {
  private static final Logger LOG = LoggerFactory.getLogger(Acquisition.class);

  public interface Options {...}

  public static class EventPartitionFn implements PartitionFn<Event> {
    @Override
    public int partitionFor(Event event, int numPartitions) {
      switch (event.route) {
        case "/metric":
          return 1;
        case "/run":
          return 1;
        default:
          return 0;
      }
    }
  }

  public static void main(String[] args) {
    // Parse the user options passed from the command-line
    Options options = PipelineOptionsFactory.fromArgs(args).as(Options.class);
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Event> events =
        pipeline.apply("ReadEvents", EventIO.readJsonArrays(options, Event.class));

    // Sort the events by their route
    PCollectionList<Event> eventPartitions =
        events.apply("PartitionEvents", Partition.of(3, new EventPartitionFn()));

    // Transform the metric events into metrics and write them to the metrics topic.
    eventPartitions.get(0)
        .apply("MapEventsToMetrics",
            MapElements.into(TypeDescriptor.of(Metric.class)).via((event) -> new Metric(event)))
        .apply("WriteMetrics",
            EventIO.writeJsons(options.getSinkMetricsPubsubTopic(), Metric.class));

    // Transform the run events into runs and write them to the runs topic.
    eventPartitions.get(1)
        .apply("MapEventsToRuns",
            MapElements.into(TypeDescriptor.of(Run.class)).via((event) -> new Run(event)))
        .apply("WriteRuns",
            EventIO.writeJsons(options.getSinkRunsPubsubTopic(), Run.class));

    // Log the ones we don't recognize...
    eventPartitions.get(2)
        .apply("LogUnknowns", MapElements.into(TypeDescriptors.strings()).via((event) -> {
          String s = event.toString();
          Acquisition.LOG.error("Unrecognized event: " + s);
          return s;
        }));
    pipeline.run();
```

(1, 2)

(3)

(4, 5)

Acquisition

1. Read PubSub
2. Decode json as “Event”
3. Sort it into correct PCollection
4. Encode metrics as Json
5. Write metrics PubSub
6. Encode non-metrics as Json
7. Write non-metrics PubSub

```java
public class Acquisition {
  private static final Logger LOG = LoggerFactory.getLogger(Acquisition.class);

  public interface Options {...}

  public static class EventPartitionFn implements PartitionFn<Event> {
    @Override
    public int partitionFor(Event event, int numPartitions) {
      switch (event.route) {
        case "/metric":
          return 1;
        case "/run":
          return 1;
        default:
          return 0;
      }
    }
  }

  public static void main(String[] args) {
    // Parse the user options passed from the command-line
    Options options = PipelineOptionsFactory.fromArgs(args).as(Options.class);
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Event> events =
        pipeline.apply("ReadEvents", EventIO.readJsonArrays(options, Event.class));

    // Sort the events by their route
    PCollectionList<Event> eventPartitions =
        events.apply("PartitionEvents", Partition.of(3, new EventPartitionFn()));

    // Transform the metric events into metrics and write them to the metrics topic.
    eventPartitions.get(0)
        .apply("MapEventsToMetrics",
            MapElements.into(TypeDescriptor.of(Metric.class)).via((event) -> new Metric(event)))
        .apply("WriteMetrics",
            EventIO.writeJsons(options.getSinkMetricsPubsubTopic(), Metric.class));

    // Transform the run events into runs and write them to the runs topic.
    eventPartitions.get(1)
        .apply("MapEventsToRuns",
            MapElements.into(TypeDescriptor.of(Run.class)).via((event) -> new Run(event)))
        .apply("WriteRuns",
            EventIO.writeJsons(options.getSinkRunsPubsubTopic(), Run.class));

    // Log the ones we don't recognize...
    eventPartitions.get(2)
        .apply("LogUnknowns", MapElements.into(TypeDescriptors.strings()).via((event) -> {
          String s = event.toString();
          Acquisition.LOG.error("Unrecognized event: " + s);
          return s;
        }));
    pipeline.run();
```

(1, 2)

(3)

(4, 5)

(6, 7)

Acquisition

1. Read PubSub
2. Decode json as “Event”
3. Sort it into correct PCollection
4. Encode metrics as Json
5. Write metrics PubSub
6. Encode non-metrics as Json
7. Write non-metrics PubSub
8. Log unknowns

```
public class Acquisition {
  private static final Logger LOG = LoggerFactory.getLogger(Acquisition.class);

  public interface Options {...}

  public static class EventPartitionFn implements PartitionFn<Event> {
    @Override
    public int partitionFor(Event event, int numPartitions) {
      switch (event.route) {
        case "/metric":
          return 1;
        case "/run":
          return 1;
        default:
          return 0;
      }
    }
  }

  public static void main(String[] args) {
    // Parse the user options passed from the command-line
    Options options = PipelineOptionsFactory.fromArgs(args).as(Options.class);
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Event> events =
        pipeline.apply("ReadEvents", EventIO.readJsonArrays(options, Event.class));

    // Sort the events by their route
    PCollectionList<Event> eventPartitions =
        events.apply("PartitionEvents", Partition.of(3, new EventPartitionFn()));

    // Transform the metric events into metrics and write them to the metrics topic.
    eventPartitions.get(0)
        .apply("MapEventsToMetrics",
            MapElements.into(TypeDescriptor.of(Metric.class)).via((event) -> new Metric(event)))
        .apply("WriteMetrics",
            EventIO.writeJsons(options.getSinkMetricsPubsubTopic(), Metric.class));

    // Transform the run events into runs and write them to the runs topic.
    eventPartitions.get(1)
        .apply("MapEventsToRuns",
            MapElements.into(TypeDescriptor.of(Run.class)).via((event) -> new Run(event)))
        .apply("WriteRuns",
            EventIO.writeJsons(options.getSinkRunsPubsubTopic(), Run.class));

    // Log the ones we don't recognize...
    eventPartitions.get(2)
        .apply("LogUnknowns", MapElements.into(TypeDescriptors.strings()).via((event) -> {
          String s = event.toString();
          Acquisition.LOG.error("Unrecognized event: " + s);
          return s;
        }));
    pipeline.run();
```

(1, 2)

(3)

(4, 5)

(6, 7)

(8)

Acquisition

1. Read PubSub
2. Decode json as “Event”
3. Sort it into correct PCollection
4. Encode metrics as Json
5. Write metrics PubSub
6. Encode non-metrics as Json
7. Write non-metrics PubSub
8. Log unknowns

```java
public class Acquisition {
  private static final Logger LOG = LoggerFactory.getLogger(Acquisition.class);

  public interface Options {...}

  public static class EventPartitionFn implements PartitionFn<Event> {
    @Override
    public int partitionFor(Event event, int numPartitions) {
      switch (event.route) {
        case "/metric":
          return 1;
        case "/run":
          return 1;
        default:
          return 0;
      }
    }
  }

  public static void main(String[] args) {
    // Parse the user options passed from the command-line
    Options options = PipelineOptionsFactory.fromArgs(args).as(Options.class);
    Pipeline pipeline = Pipeline.create(options);
    PCollection<Event> events =
        pipeline.apply("ReadEvents", EventIO.readJsonArrays(options, Event.class));

    // Sort the events by their route
    PCollectionList<Event> eventPartitions =
        events.apply("PartitionEvents", Partition.of(3, new EventPartitionFn()));

    // Transform the metric events into metrics and write them to the metrics topic.
    eventPartitions.get(0)
        .apply("MapEventsToMetrics",
            MapElements.into(TypeDescriptor.of(Metric.class)).via((event) -> new Metric(event)))
        .apply("WriteMetrics",
            EventIO.writeJsons(options.getSinkMetricsPubsubTopic(), Metric.class));

    // Transform the run events into runs and write them to the runs topic.
    eventPartitions.get(1)
        .apply("MapEventsToRuns",
            MapElements.into(TypeDescriptor.of(Run.class)).via((event) -> new Run(event)))
        .apply("WriteRuns",
            EventIO.writeJsons(options.getSinkRunsPubsubTopic(), Run.class));

    // Log the ones we don't recognize...
    eventPartitions.get(2)
        .apply("LogUnknowns", MapElements.into(TypeDescriptors.strings()).via((event) -> {
          String s = event.toString();
          Acquisition.LOG.error("Unrecognized event: " + s);
          return s;
        }));
    pipeline.run();
```

(1, 2)

(3)

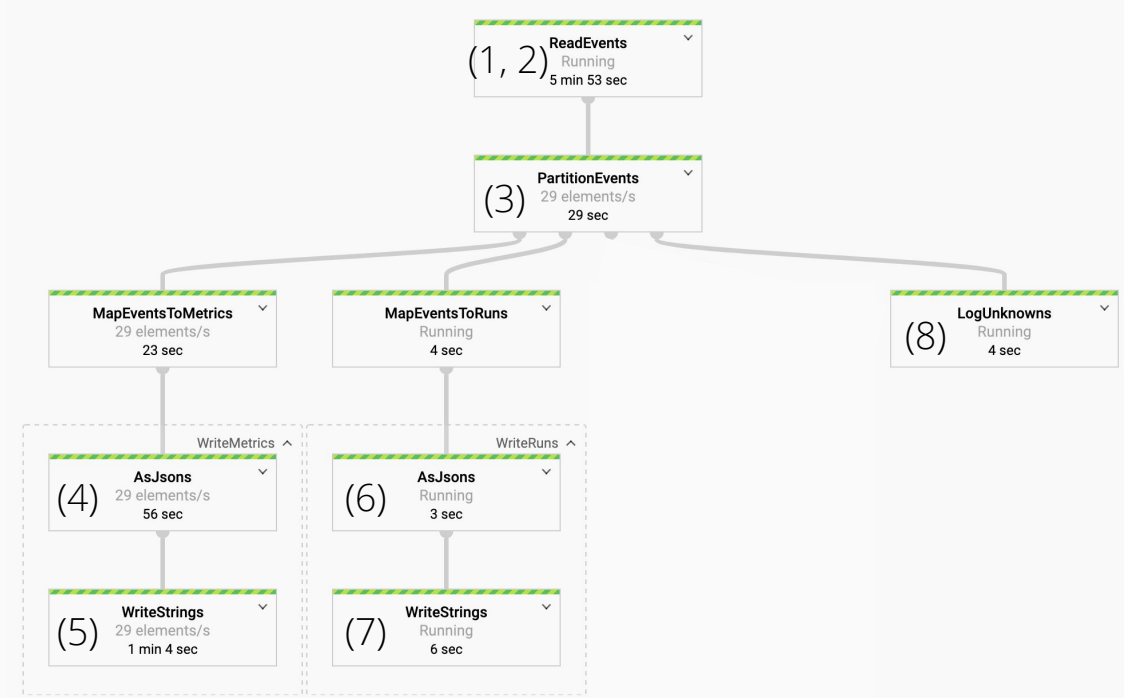
(4, 5)

(6, 7)

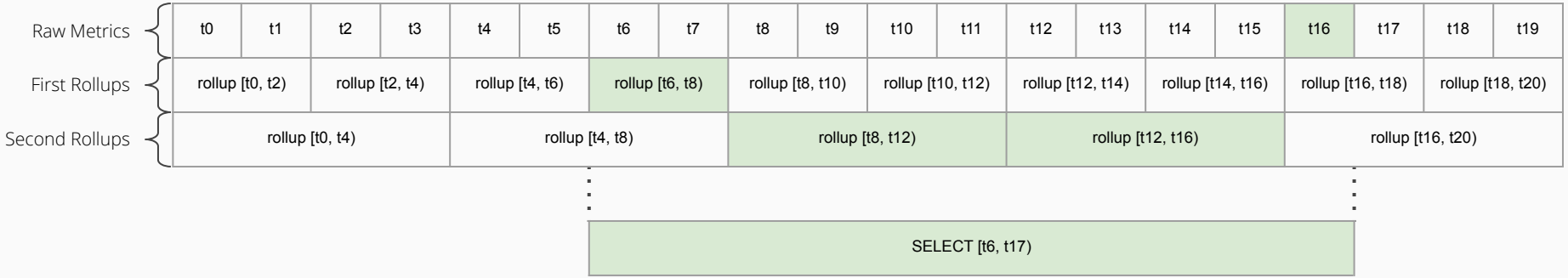
(8)

Acquisition

1. Read PubSub
2. Decode json as “Event”
3. Sort it into correct PCollection
4. Encode metrics as Json
5. Write metrics PubSub
6. Encode non-metrics as Json
7. Write non-metrics PubSub
8. Log unknowns



Rolling Up Metrics to Reduce Scanning



Rollup stepping window of metrics using *associative aggregates.*

- Count
- Sum
- Min, Max
- Sum2 - sum of x squared

$(x * y) * z = x * (y * z)$ *for all x, y, z in S*

$f(A \cup B) = g(f(A), f(B))$

$sum(A \cup B) = sum(A) + sum(B)$

$count(A \cup B) = count(A) + count(B)$

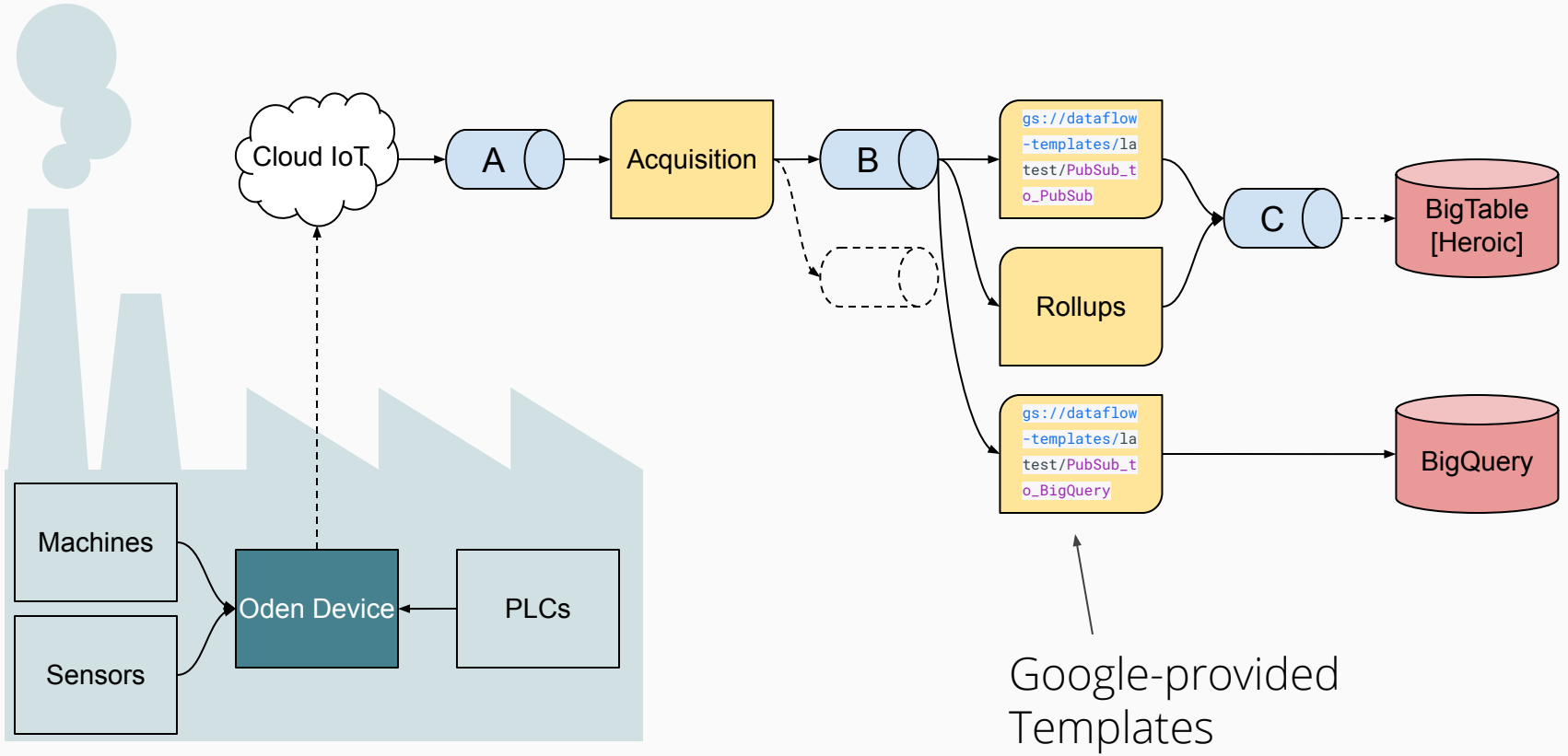
$max(A \cup B) = max(max(A), max(B))$

$sum2(A \cup B) = sum2(A \cup B) + sum2(A \cup B)$

$mean(A \cup B) = sum(A \cup B) / count(A \cup B)$

$stddev(A \cup B) = 1/(count(A \cup B)*(count(A \cup B)-1))*(sum2(A \cup B)-sum(A \cup B)^2)$

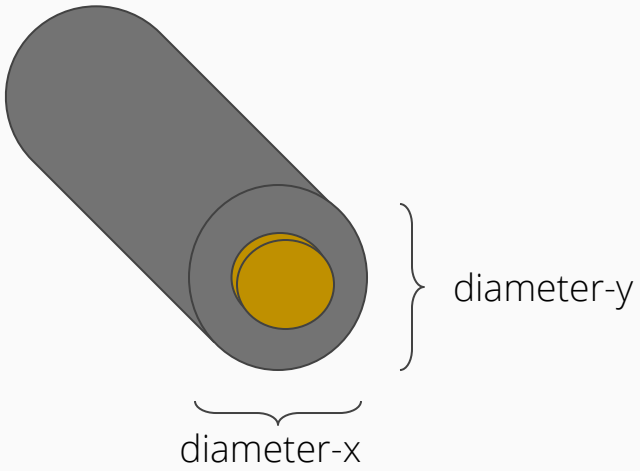
Oden's Streaming Data Pipeline



Rollups

1. Read metrics from PubSub
2. Window into 60s buckets
3. Group by their device
4. Creates aggregates
   a. Min/Max
   b. Sum
   c. Count
   d. Sum2
5. Write aggregate-metrics to PubSub

Calculated Metrics

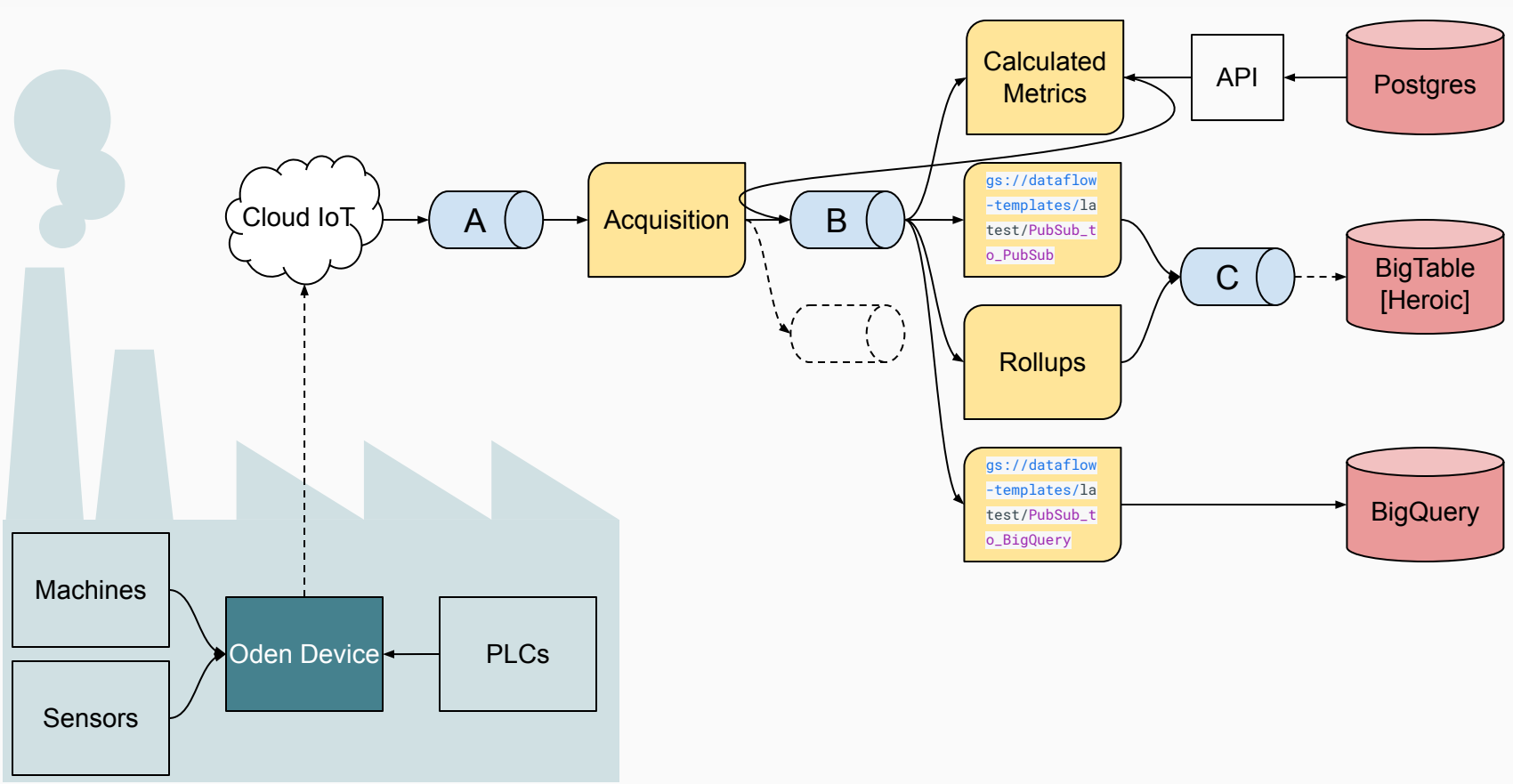


User Has:
diameter-x and **diameter-y**

User Wants:
avg-diameter = (diameter-x + diameter-y) / 2

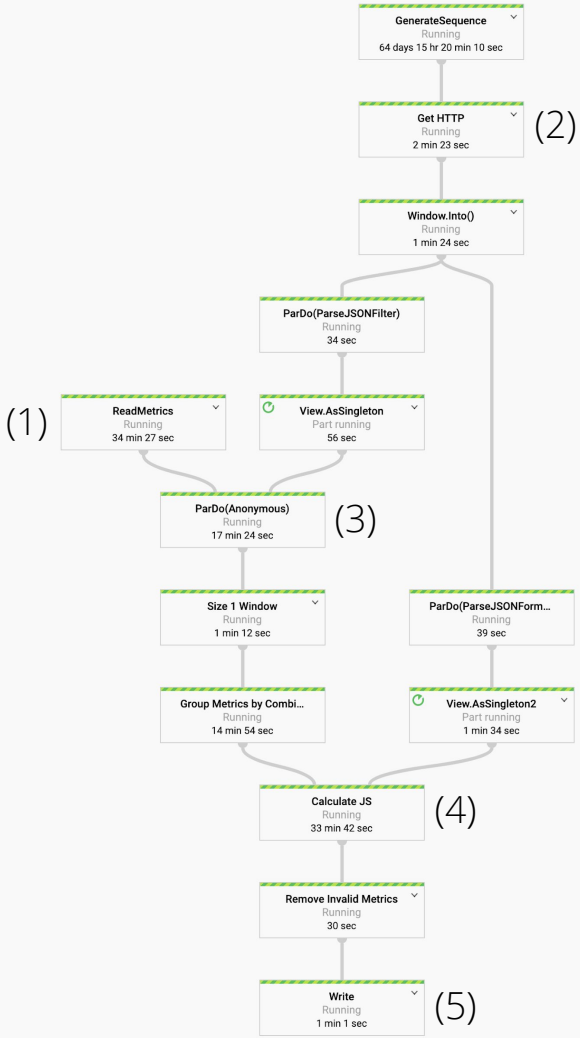
- Metrics need to be computed in real-time.
- Components can be read from different devices with different clocks.
- Formulas are stored in postgres.

Oden's Streaming Data Pipeline



Calculated Metrics

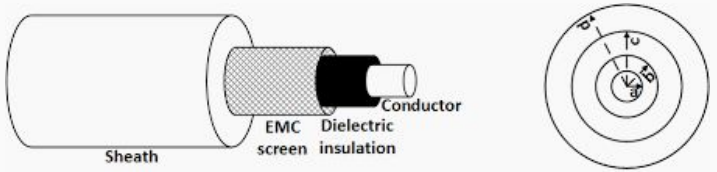
1. Read metrics from PubSub
2. Read configuration (refreshing)
3. Group component metrics by their formulas
4. Use embedded JS interpreter to apply formula
5. Write to PubSub

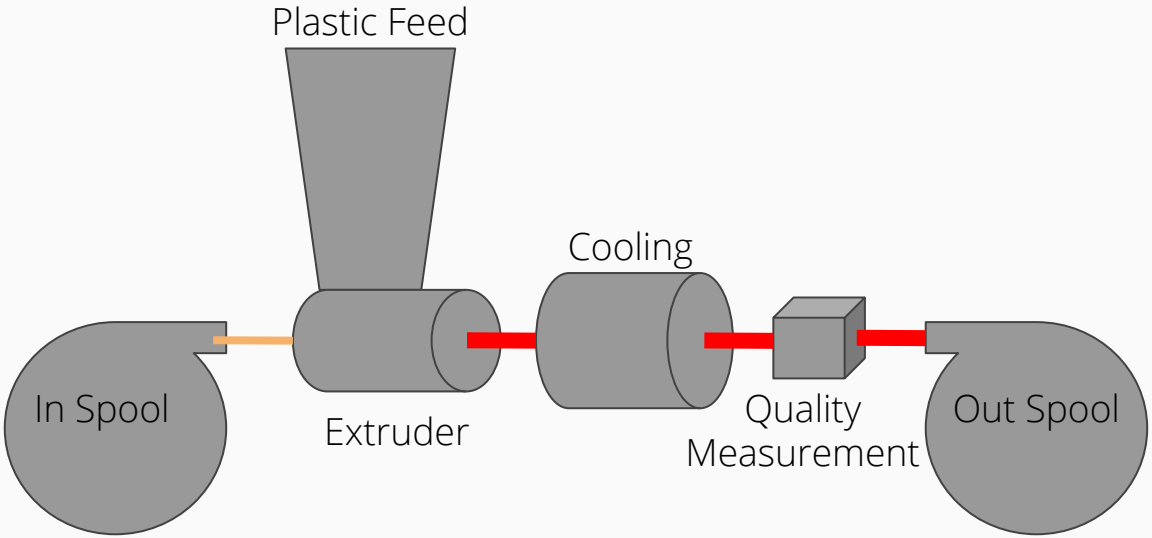


Cool Ways Oden Uses Apache Beam

Cable Manufacturing

- Copper is pulled from an in-spool into an extruder.
- Plastic is melted over the copper to make wire.
- Wire is cooled.
- Wire is pulled into an out-spool.

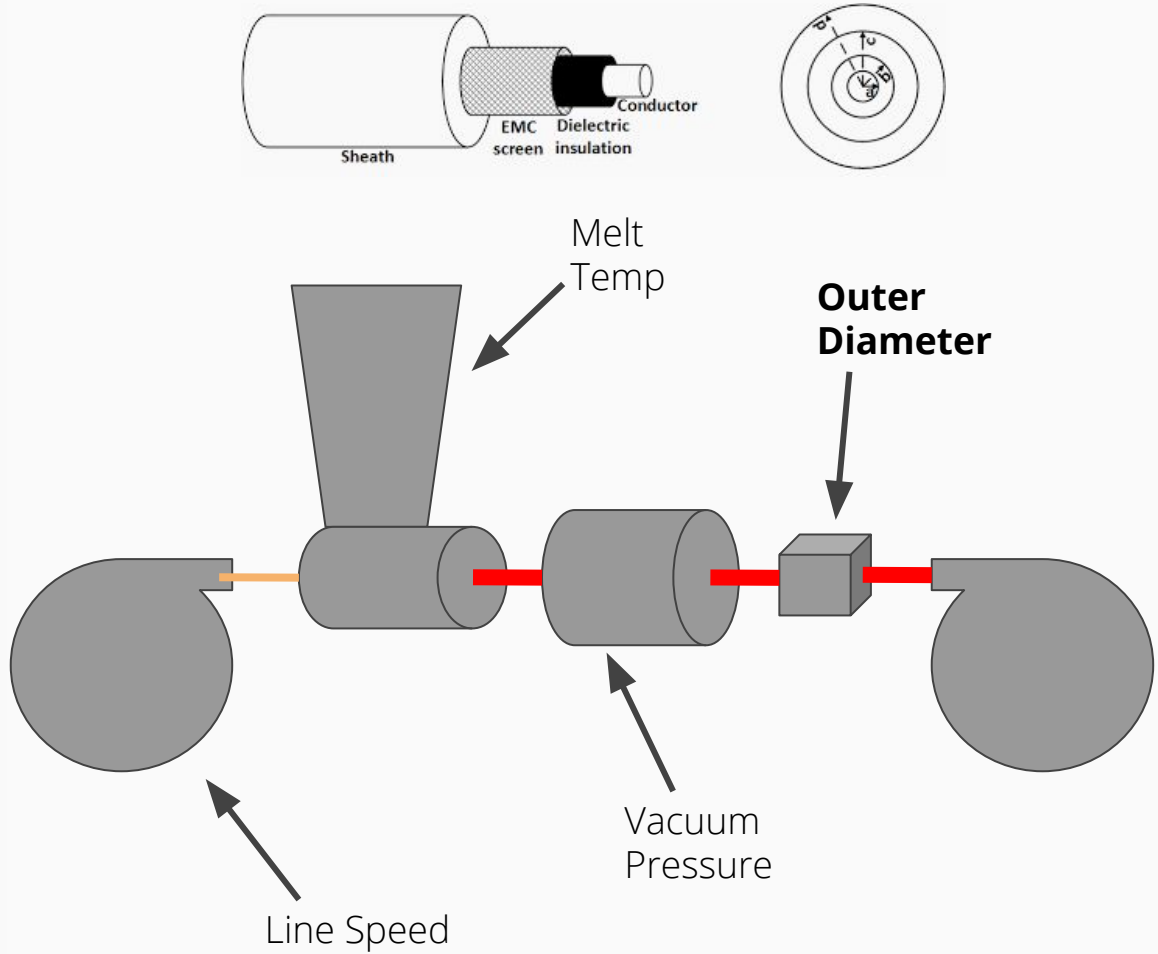




Cable Manufacturing

- Copper is pulled from an in-spool into an extruder.
- Plastic is melted over the copper to make wire.
- Wire is cooled.
- Wire is pulled into an out-spool.

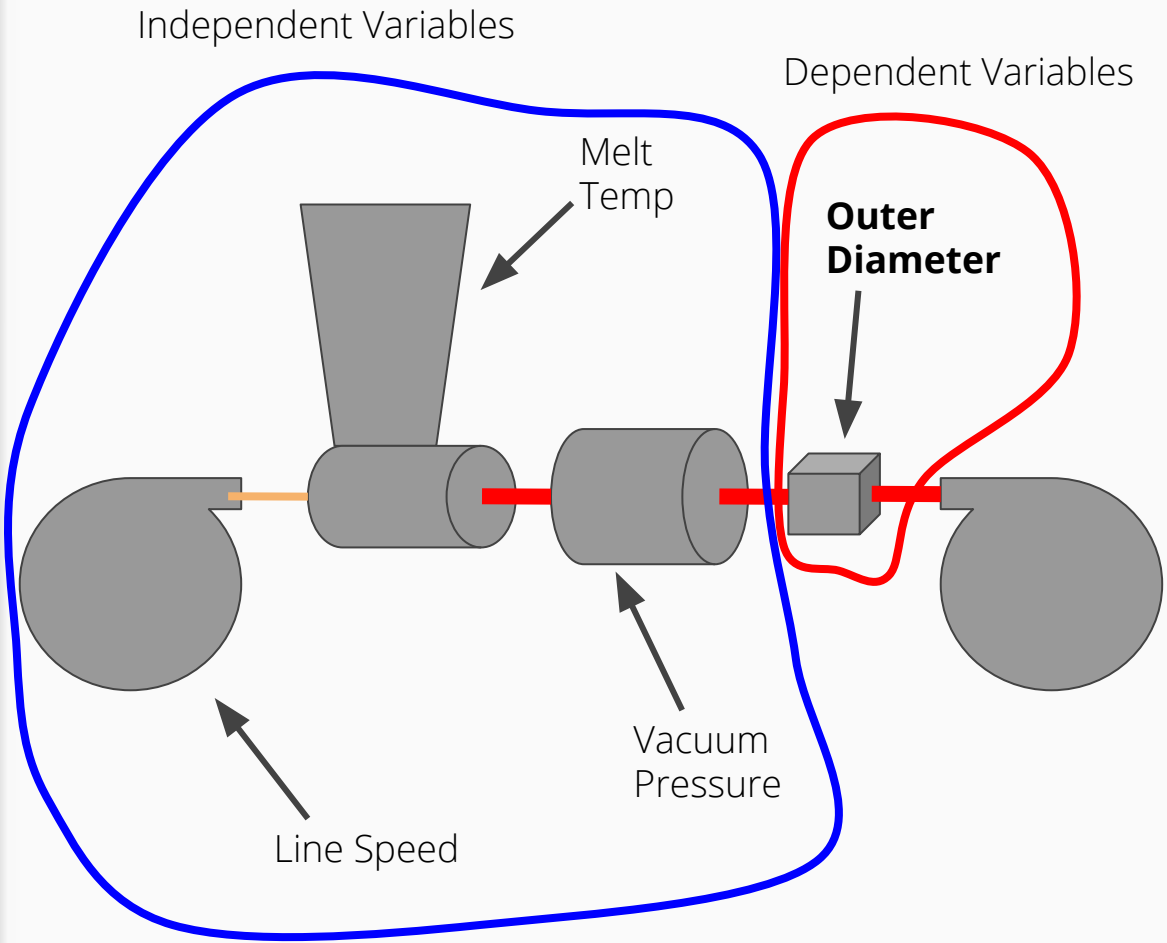
- A laser measures the diameter of the wire to monitor its closeness to spec.



Cable Manufacturing

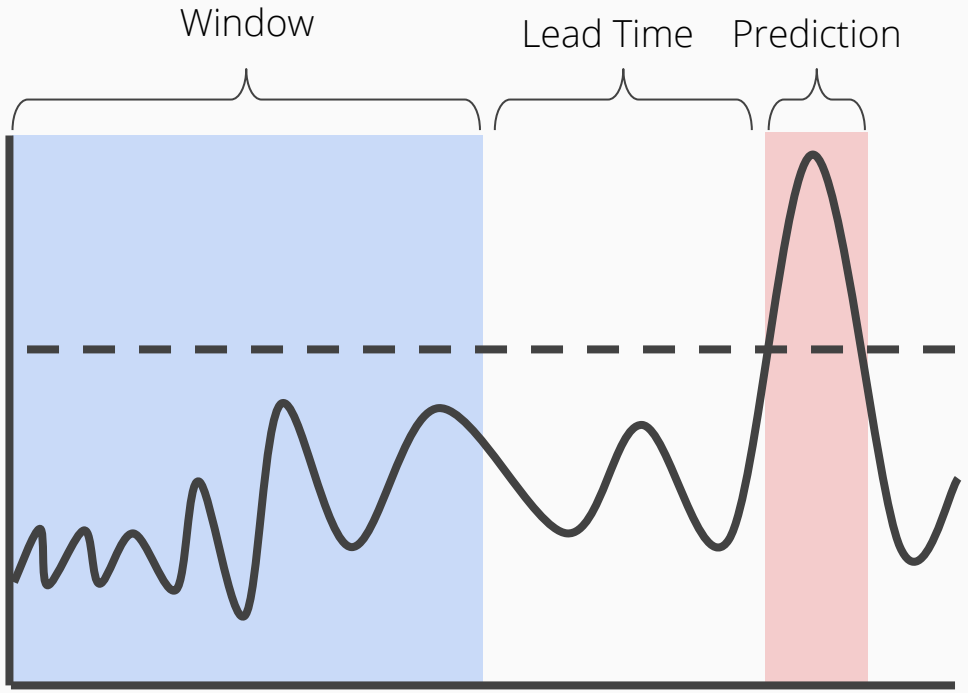
- Copper is pulled from an in-spool into an extruder.
- Plastic is melted over the copper to make wire.
- Wire is cooled.
- Wire is pulled into an out-spool.

- A laser measures the diameter of the wire to monitor its closeness to spec.

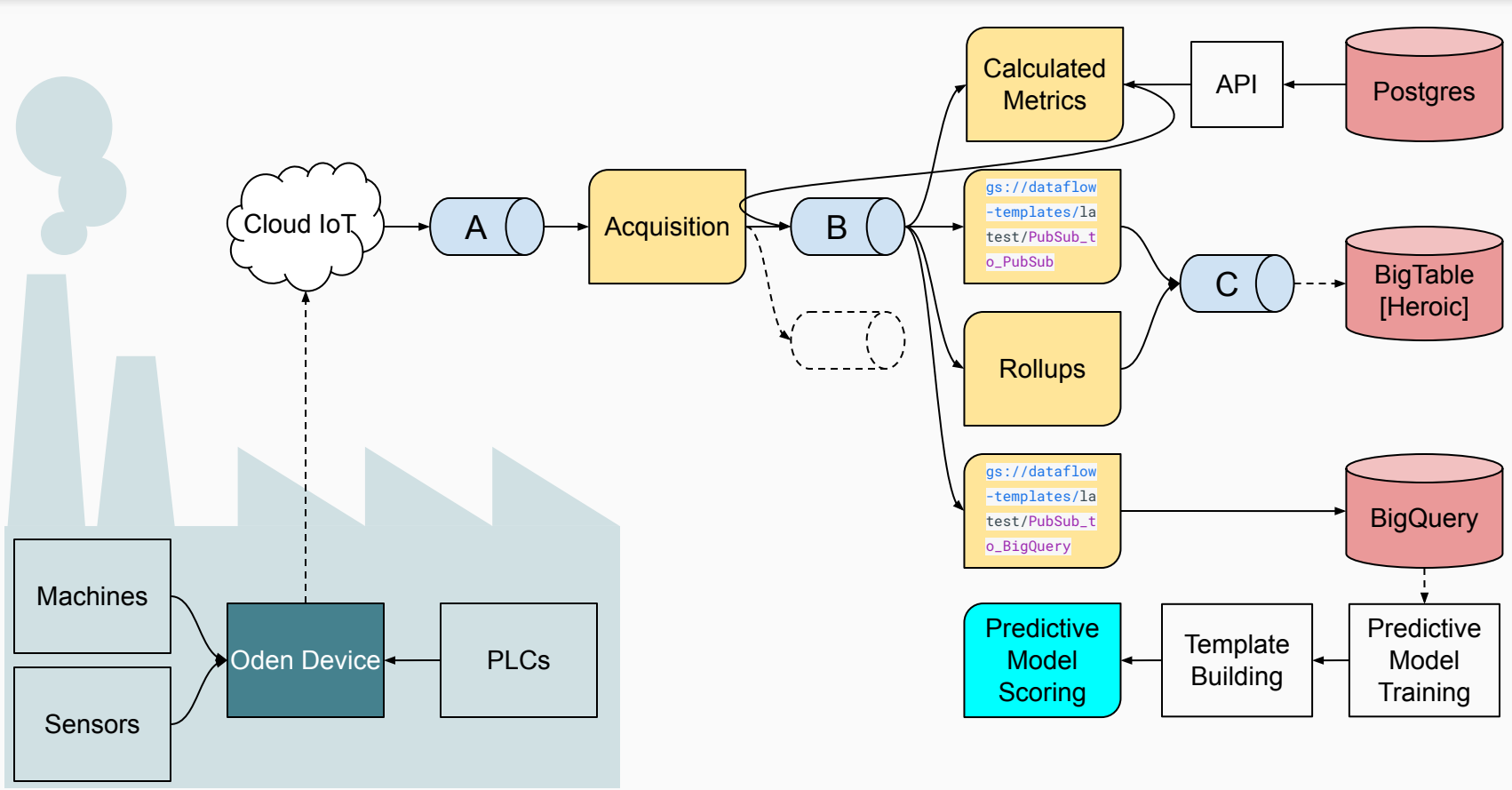


Real Time Stability Prediction

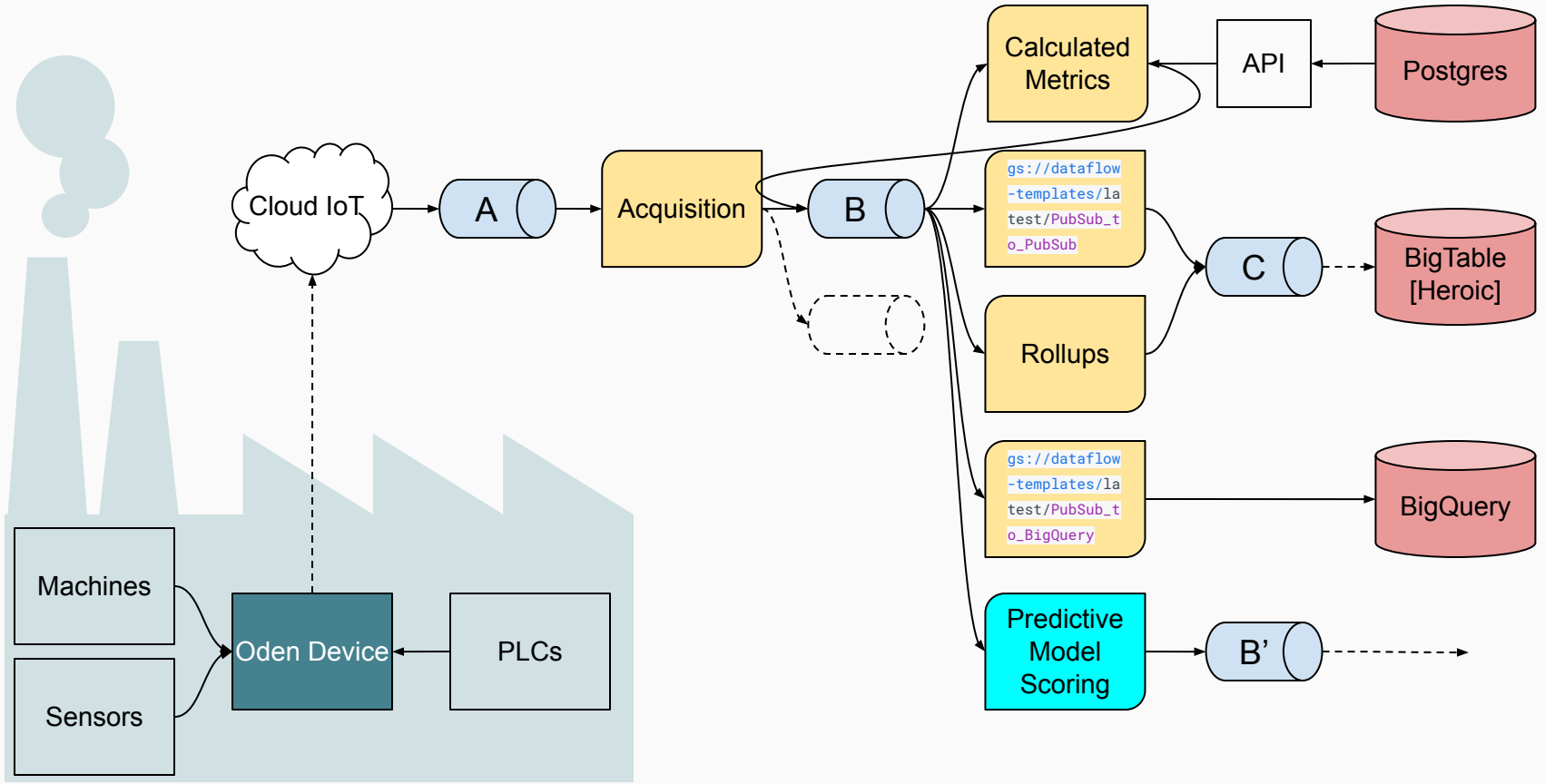
- Given a the set of metrics for a line over some *window*, with some *lead time*, make a *prediction* of a future metric value.



Oden's Streaming Data Pipeline



Oden's Streaming Data Pipeline



Predictive Model Scoring

1. Load skpipeline from GCS
2. Read from pubsub
3. Create prediction (score)
4. Write predictions to pubsub

```
class ModelScoreDoFn(apache_beam.DoFn):
    def process(self, element, skpipeline):
        x = np.array([])
        (id, fv) = element
        if fv is not None:
            x = np.append(x, fv).reshape(1, -1)
            pred = skpipeline.predict(x)
            id_date_list = id.split(".")
            if len(id_date_list) >= 2:
                ts = (pd.to_datetime(id_date_list[1])).timestamp()
                return [json.dumps(...)]

def run(argv=None):
    pipeline_options = PipelineOptions()
    pipeline_options.view_as(SetupOptions).save_main_session = True
    ops = pipeline_options.view_as(UserOptions)

    # Copy the model from Cloud storage
    subprocess.check_call(["gsutil", "cp", ops.modeluri, MODEL_LOCAL_FILE_PATH])
    skpipeline = joblib.load(MODEL_LOCAL_FILE_PATH)

    # Create the Apache Beam Pipeline
    p = apache_beam.Pipeline(options=pipeline_options)

    preds = (p
        # Read from pubsub into a PCollection
        | "read" >> ReadFromPubSub(subscription=ops.inputsubscription)

        # Parse the JSON messages and then score with the model
        | "parse" >> apache_beam.ParDo(FeaturePacktoFeatureVectorDoFn())
        | "score" >> apache_beam.ParDo(ModelScoreDoFn(), skpipeline)

        # Write the result to pubsub
        | "write" >> WriteStringsToPubSub(user_options.outputtopic)
    )

    result = p.run().wait_until_finish()
```

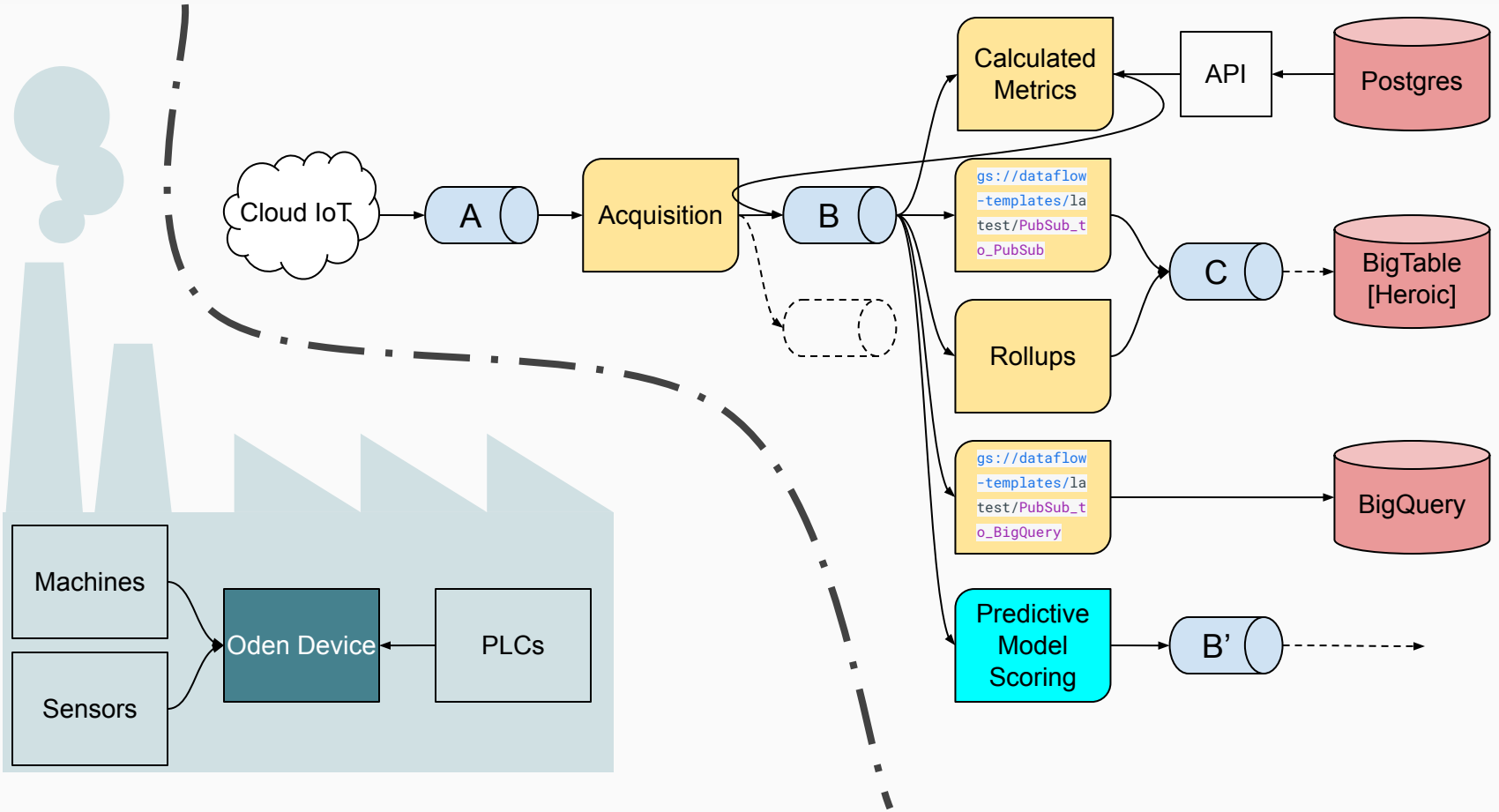
(1)

(2)

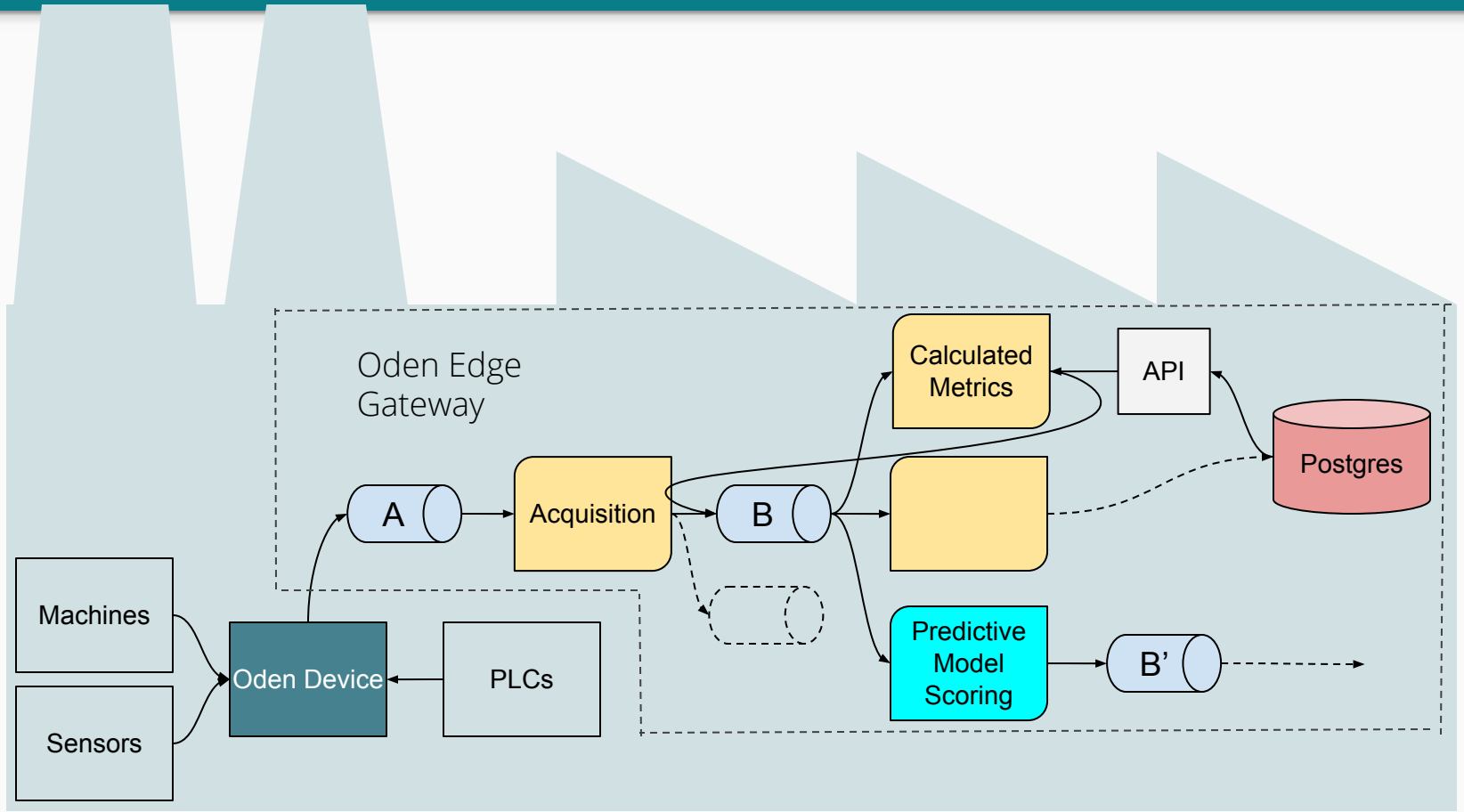
(3)

(4)

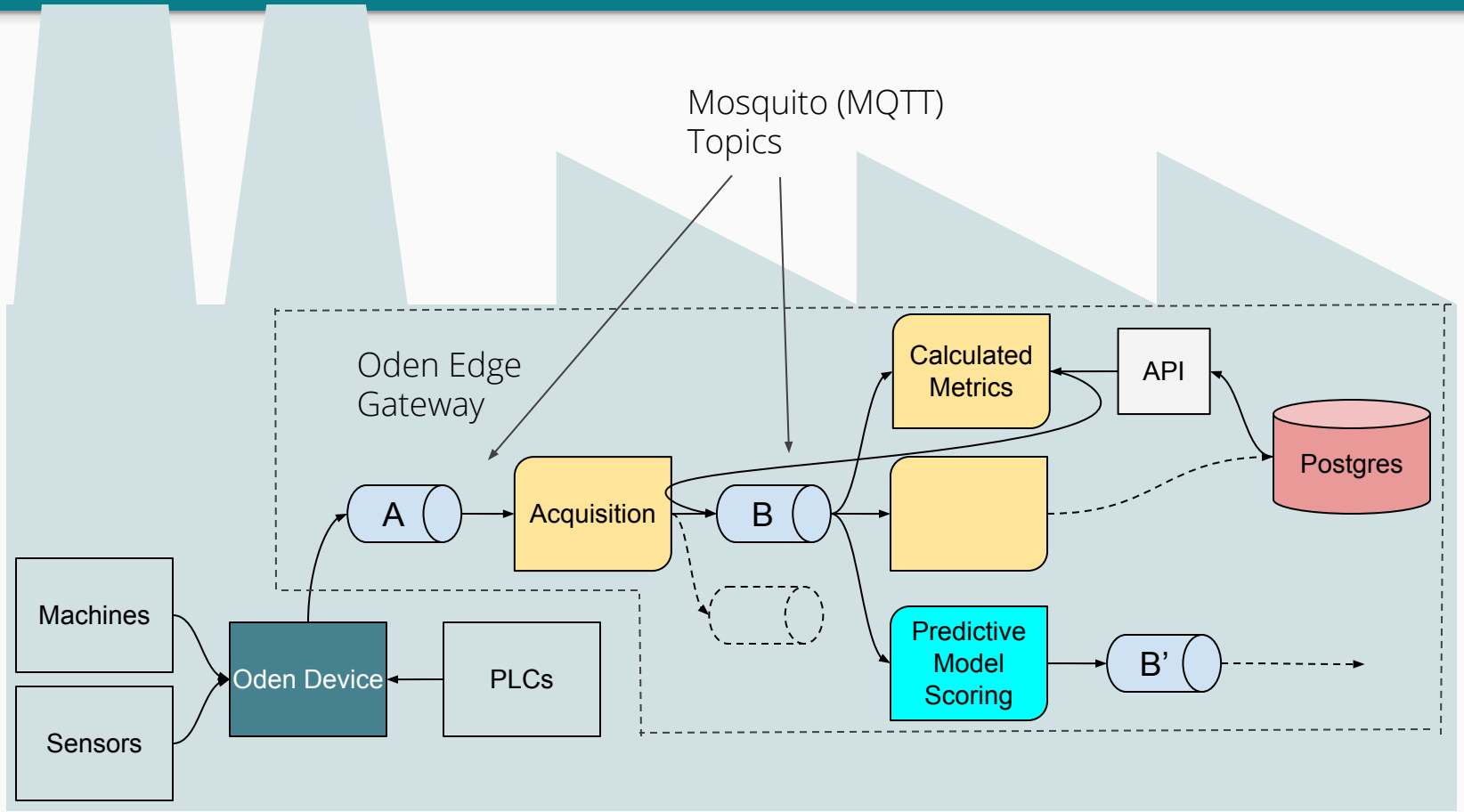
Oden's Streaming Data Pipeline (When Partitioned)



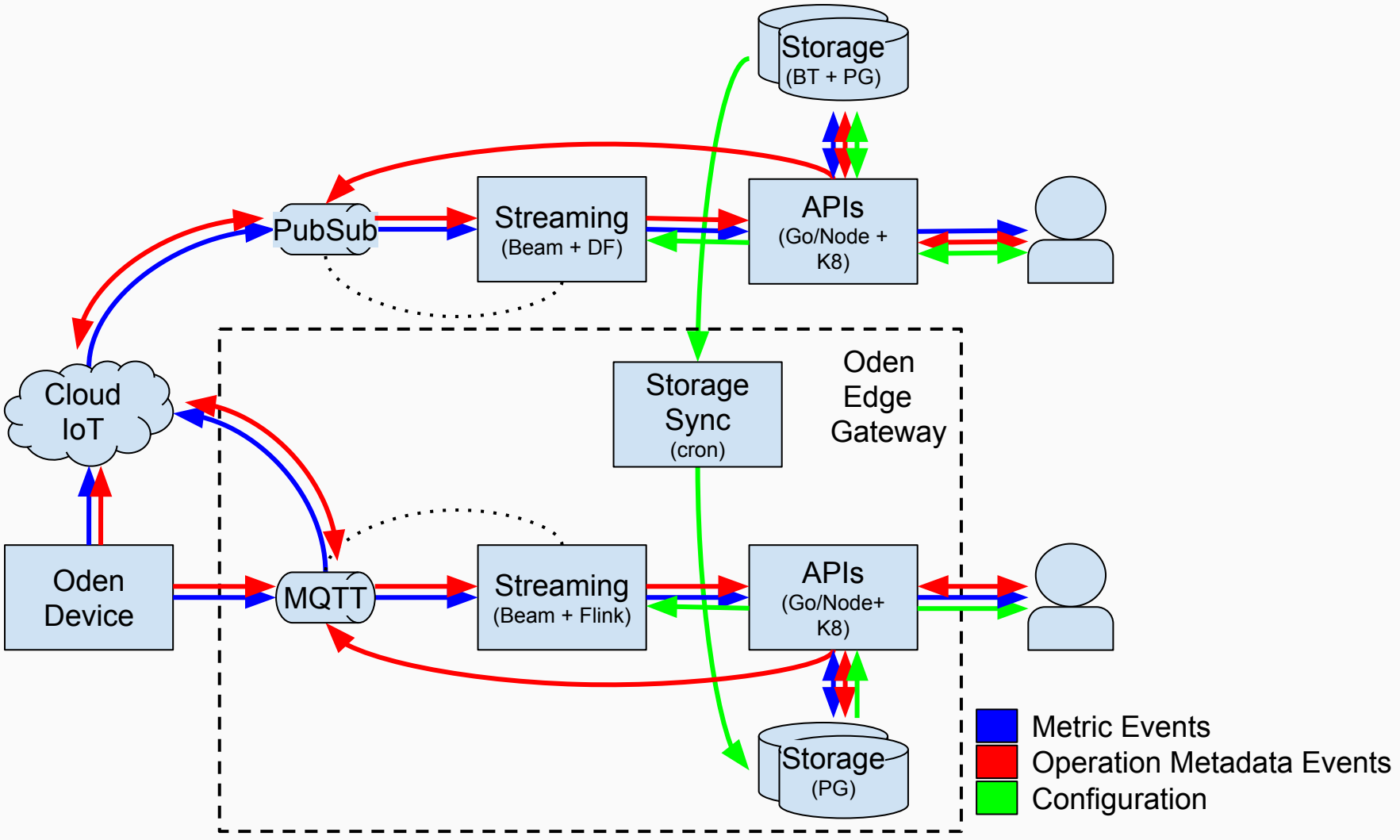
Oden's Streaming Data Pipeline (When Partitioned)



Oden's Streaming Data Pipeline (When Partitioned)



Oden Edge Gateway Server


Storage
(BT + PG)
PubSub
Streaming
(Beam + DF)
APIs
(Go/Node +
K8)
Cloud
IoT
Storage
Sync
(cron)
Oden
Edge
Gateway
Oden
Device
MQTT
Streaming
(Beam + Flink)
APIs
(Go/Node+
K8)
Storage
(PG)
Metric Events
Operation Metadata Events
Configuration

EventIO

Class of static Read and Write PTransform wrappers that can be configured at *template build time* to interface via MQTT or PubSub

```
public class EventIO {
  enum Mode { MQTT, PUBSUB }

  private static Mode getModeFromOption(String mode) {
    if (mode.equals("MQTT")) {
      return Mode.MQTT;
    } else if (mode.equals("PUBSUB")) {
      return Mode.PUBSUB;
    }
  }

  private static class ReadStrings extends PTransform<PBegin, PCollection<String>> {
    private static final long serialVersionUID = 1L;
    private String[] mqttComponents;
    private ValueProvider<String> pubsubSubscription;
    private Mode mode;

    public ReadStrings(
        String mode, String mqttServerAndTopic, ValueProvider<String> pubsubSubscription) {
      this.mode = getModeFromOption(mode);
      this.mqttComponents = mqttServerAndTopic.split(",");
      this.pubsubSubscription = pubsubSubscription;
    }

    @Override
    public PCollection<String> expand(PBegin input) {
      switch (this.mode) {
        case MQTT:
          return input
              .apply(
                  MqttIO.read().withConnectionConfiguration(MqttIO.ConnectionConfiguration.create(
                      this.mqttComponents[0], this.mqttComponents[1], "NewReader")))
              .apply(MapElements.into(TypeDescriptors.strings()).via(...);
        case PUBSUB:
          return input.apply(PubsubIO.readStrings().fromSubscription(this.pubsubSubscription));
      }
    }
  }
  ...
}
```

In summary

- Beam is a great framework for stream processing.
- Building of templates allows Oden to deploy stateless stream processing code, stateful stream processing code, and machine learning models in a unified way.
- Abstracted runners allows us to deploy the same infrastructure on the edge as we do the cloud.

Thank You

We’re hiring! oden.io/jobs