codecentric

Test Driving Data Streaming

Thorsten Brunzendorf

Thorsten Brunzendorf

Senior IT consultant at codecentric Nürnberg

Various roles: developer, architect, product owner, …

Main focus on JVM ecosystem, cloud native, testing, CI/CD and feedback mechanisms in general

Involved in software crafters communities, e.g. Softwerkskammer Nürnberg and co-organizing SoCraTes Day Franken

Twitter @thbrunzendorf

- TDD
- BDD
- Domain Driven Design
- Product Management
- Mending Legacy Code
- Mutation Testing
- Property Based Testing
- Cloud Native

codecentric

Conference for Software Craft and Testing
SoCraTes Franken

Motivation

Why Test Driving?

- **Question on Software Crafters Slack #tdd channel**
- **Let's try that!**
- **Disclaimer: This is an experiment**

"Why is there hardly any test driven approach in introductions to new tech libraries and frameworks"
(paraphrased)

https://softwarecrafters.slack.com

Subject

Why Data Streaming with Apache Beam?

- **Personal experience in professional project**
- **Dedicated support for testing**
- **Interesting concepts worth learning**
- **Considered “relevant”**

Apache Beam

APR 2019

TRIAL

Apache Beam is an open-source unified programming model for defining and executing both batch and streaming data parallel processing pipelines. The Beam model is based on the Dataflow model which allows us to express logic in an elegant way so that we can easily switch between batch, windowed batch or streaming. The big data-processing ecosystem has been evolving quite a lot which can make it difficult to choose the right data-processing engine. One of the key reasons to choose Beam is that it allows us to switch between different runners — a few months ago Apache Samza was added to the other runners it already supports, which include Apache Spark, Apache Flink and Google Cloud Dataflow. Different runners have different capabilities and providing a portable API is a difficult task. Beam tries to strike a delicate balance by actively pulling innovations from these runners into the Beam model and also working with the community to influence the roadmap of these runners. Beam has SDKs in multiple languages including Java, Python and Golang. We've also had success using Scio which provides a Scala wrapper around Beam.

https://www.thoughtworks.com/de/radar/languages-and-frameworks/apache-beam

What is Data Streaming?

"a type of data processing engine that is designed with **infinite data sets** in mind“ (Tyler Akidau)

Apache Beam = Batch + Str**eam**ing

All about Apache Beam

Unified

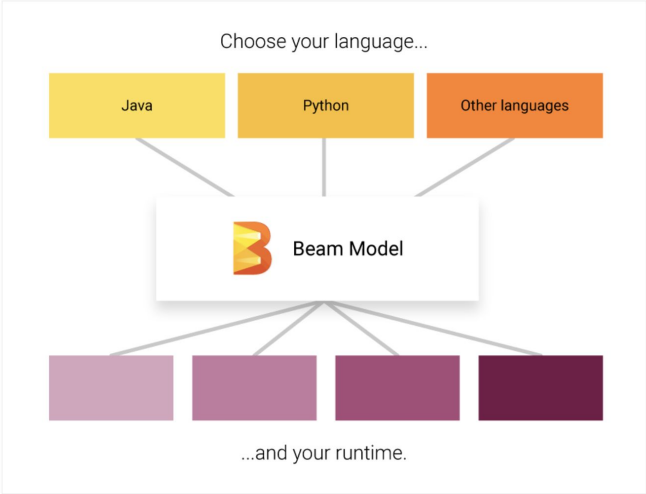
Use a single programming model for both batch and streaming use cases.

Portable

Execute pipelines on multiple execution environments.

Extensible

Write and share new SDKs, IO connectors, and transformation libraries.



Toadstool?

Example 1

Greeting Reviewer

Requirements

- The reviewer eads a list of greetings from an input file, reviews each greeting and writes the list of approved greetings to an output file
- The reviewer should be strict:
  Only accepts greetings starting with hello
- The reviewer should be polite:
  Does not like screaming and therefore switches everything to lowercase

Java 8 Streams Implementation

ReviewerStreamTest.java

```java
package de.thbrunzendorf.beam.greetings;

import ...

public class ReviewerStreamTest {
    @Test
    public void acceptFromListWithOneEntryMatchingExactly() {
        List<String> input = Arrays.asList("hello world!");
        List<String> output = new ReviewerStream().accept(input);
        assertThat(output, Matchers.containsInAnyOrder( ...items: "hello world!"));
    }

    @Test
    public void acceptFromListWithOneEntryPolitely() {
        List<String> input = Arrays.asList("HELLO World!");
        List<String> output = new ReviewerStream().accept(input);
        assertThat(output, Matchers.containsInAnyOrder( ...items: "hello world!"));
    }

    @Test
    public void acceptFromListWithOneEntryNotMatching() {
        List<String> input = Arrays.asList("Nice to meet you!");
        List<String> output = new ReviewerStream().accept(input);
        assertThat(output, Matchers.empty());
    }

    @Test
    public void acceptFromListWithManyEntry() {
        List<String> input = Arrays.asList("hello world!", "Nice to meet you!", "Hello BEAM!", "How are you?
        List<String> output = new ReviewerStream().accept(input);
        assertThat(output, Matchers.containsInAnyOrder( ...items: "hello world!", "hello beam!"));
    }

    @Test
    public void acceptFromEmptyList() {
        List<String> input = Arrays.asList();
        List<String> output = new ReviewerStream().accept(input);
        assertThat(output, Matchers.empty());
    }
}
```

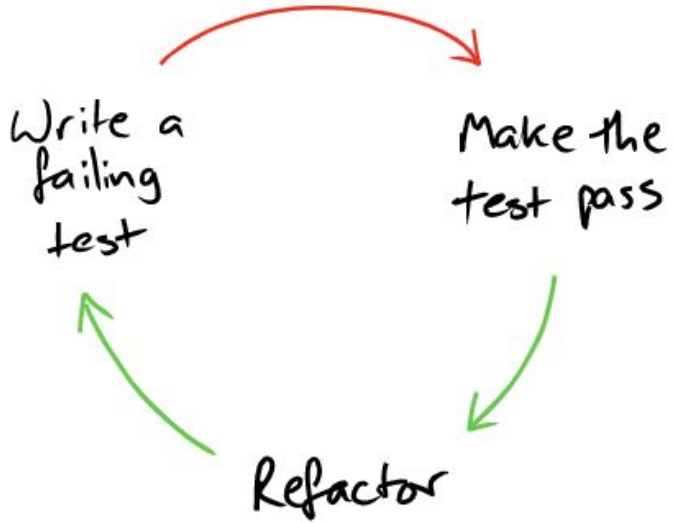
ReviewerStream.java

```java
package de.thbrunzendorf.beam.greetings;

import ...

public class ReviewerStream {
    public List<String> accept(List<String> input) {
        return input.stream()
                .map(String::toLowerCase)
                .filter(s -> s.startsWith("hello"))
                .collect(Collectors.toList());
    }
}
```

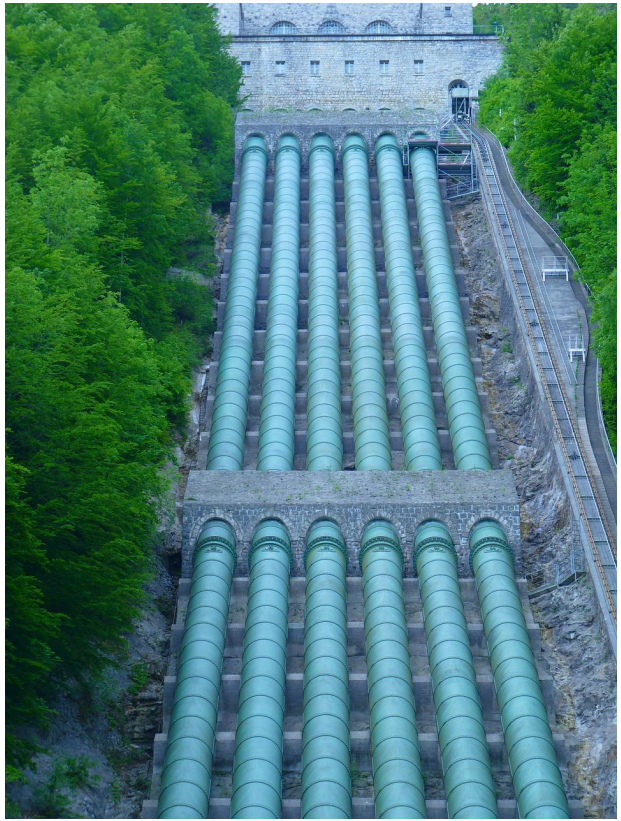
Test Driving

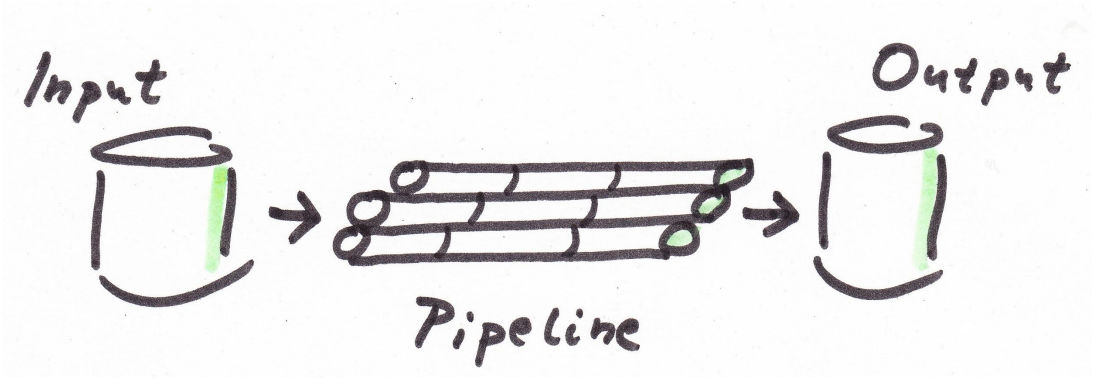
Feedback mechanisms for our approach to learning

- **Primarily we will**
  - **Explore IDE suggestions**
  - **Lean on the compiler**
  - **Lean on runtime error messages**
  - **Lean on failed assertion messages**
- **But we will also**
  - **Consult documentation**
  - **Do just enough design upfront**

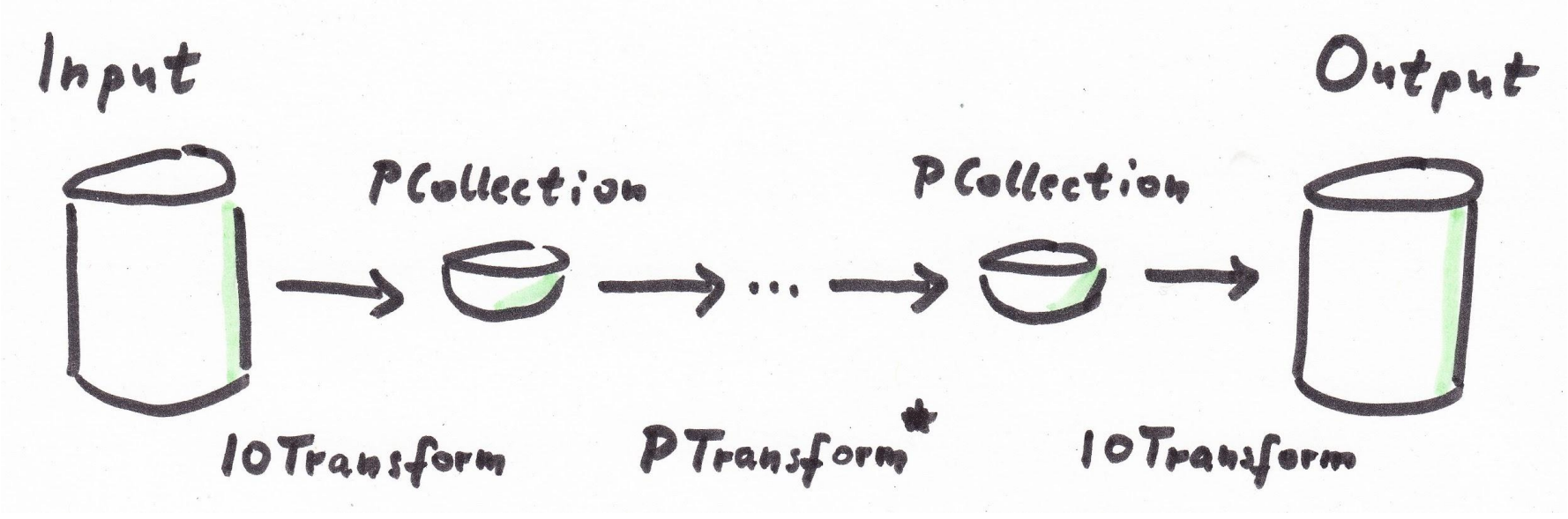


Apache Beam Concepts: Pipeline





Apache Beam Concepts: PCollection and PTransform



A Pipeline is a directed acyclic graph
with PCollections as nodes connected by PTransforms as edges

Testing Apache Beam Pipelines

Recipe

1. Create a TestPipeline.
2. Create some static, known test input data.
3. Use the Create transform to create a PCollection of your input data.
4. Apply your … transform to the input PCollection and save the resulting output PCollection.
5. Use PAssert … to verify that the output PCollection contains the elements that you expect.

https://beam.apache.org/documentation/pipelines/test-your-pipeline/

Example 1 (again)

Greeting Reviewer

Requirements

- The reviewer eads a list of greetings from an input file, reviews each greeting and writes the list of approved greetings to an output file
- The reviewer should be strict:
  Only accepts greetings starting with hello
- The reviewer should be polite:
  Does not like screaming and therefore switches everything to lowercase

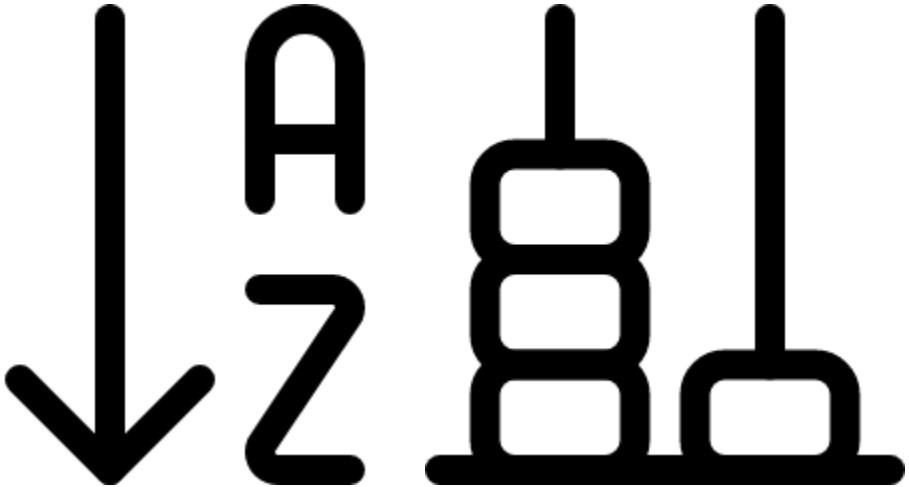
Let’s start coding!

https://github.com/thbrunzendorf/test-driving-data-streaming

Example 2

Character Counting

More elaborate bounded stream



Requirements

- We want to read lines of text and count occurrences of each character
- We are only interested in letters, not numbers or other characters
- We won't distinguish between upper and lower case

Character Counting Example Shows

- Using PipelineOptions
- Implementing custom filters and mappings with ParDo
- Aggregating data and using the KV type
- Writing custom Coders
- Creating composite PTransforms

We will take a short look at the code

Challenges with Streaming Data

- With unbounded input we can not wait until “the end“ to publish outputs because there is no end

- Applying stateless transformations (e.g. simple mappings and filters) continuously for every input item is easy - Applying stateful aggregations (e.g. count, sum, avg) not so much

- Idea: Apply such transformations per “window” (based on timestamps of items)

Window

- Definition: Bounded subdivision of a (potentially unbounded) PCollection according to the timestamps of its individual elements

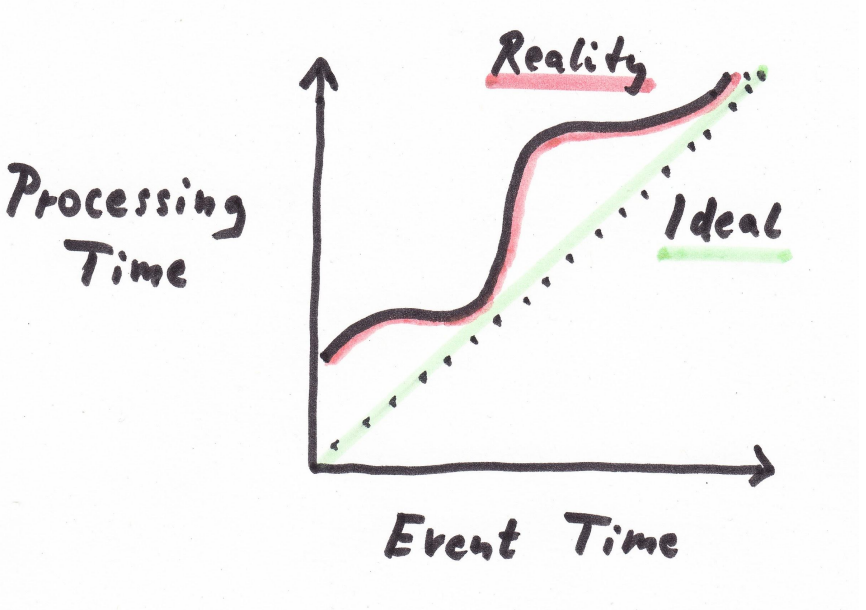
- Supported Windowing Functions
  - Fixed Time Windows (or Tumbling Window)
  - Sliding Time Windows (or Hopping Window)
  - Per-Session Windows
  - Single Global Window

- See https://blog.codecentric.de/en/2018/10/window-functions-in-stream-analytics/

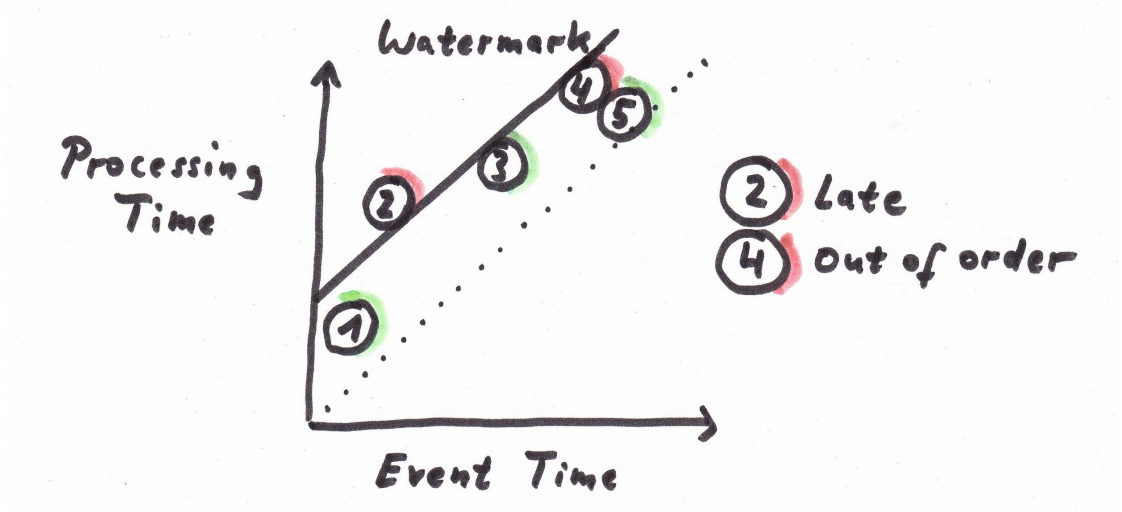
- Which timestamp do we want to use?

Event vs Processing Time

- **Event time: time at which events actually occurred**
- **Processing time: time at which events are observed in the system**
- **New problems we have to consider**
  - **out-of-order events**
  - **late events**



Watermarks, Late and Out-of-Order Events, Triggers



Watermarks are a way the system measures progress and completeness and often are estimates: can be too slow or two fast.
Triggers answer the question when to start calculations and can allow emitting early results as well as processing of late data.

Testing Streaming Data

- How to test **non-deterministic** streaming data
  with behavior dependent on different timing?

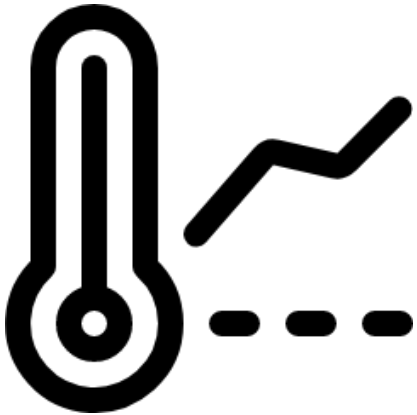
- **TestStream** is a PTransform
  that performs a series of events
  - adding additional elements to a pipeline
  - advancing the watermark of the TestStream
  - advancing the pipeline processing time clock

- TestStream permits tests
  which observe the effects of triggers on the output a pipeline produces

Example 3

Weather statistics

Unbounded stream



Requirements

- We will get current weather measurement data continuously
- Each data record consists of the location of a weather station, a timestamp and measured data, where we will start with temperature only
- We are interested in statistics for each location, starting with the total number of measurements and the average temperature

Switching Execution Engines

- Here: Using the FlinkRunner instead of the DirectRunner
- About Flink
  - “Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams”
- To use Flink in embedded mode
  - Add dependency containing FlinkRunner
  - Specify runner as FlinkRunner in PipelineOptions
- To use Flink in cluster mode additionally
  - Build a fat jar
  - Specify flinkMaster URL in PipelineOptions (extend FlinkPipelineOptions)
- Demo

Spotify Scio

- Scala API for Apache Beam
  - Developed by Spotify
  - https://spotify.github.io/scio/
- Demo
  - Just a quick glance at the code
- Comparison
  - Idiomatic Scala especially when transforming SCollections
  - End-to-End tests for entire pipeline using JobTest

Test support as first class citizen, transformation logic well testable

Tests still bound to JUnit Vintage

API not always intuitive

It helps to know some of the concepts ;)

codecentric

codecentric AG
Frankenstr. 152
90461 Nürnberg

Thorsten Brunzendorf
Senior IT Consultant
thorsten.brunzendorf@codecentric.de
www.codecentric.de

@thbrunzendorf

List of References

- Tyler Akidau: Streaming 101: The world beyond batch - A high-level tour of modern data-processing concepts - https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
- Tyler Akidau: Streaming 102: The world beyond batch - The what, where, when, and how of unbounded data processing - https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102
- Thomas Groh: Testing Unbounded Pipelines in Apache Beam - https://beam.apache.org/blog/2016/10/20/test-stream.html
- Henry Suryawirawan: Apache Beam Katas - https://beam.apache.org/blog/2019/05/30/beam-kata-release.html
- Matthias Niehoff: Verteilte Stream Processing Frameworks für Fast Data & Big Data – Ein Meer an Möglichkeiten - https://blog.codecentric.de/2017/03/verteilte-stream-processing-frameworks-fuer-fast-data-big-data-ein-meer-moeglichkeiten/
- Matthias Niehoff: Event-Zeit-Verarbeitung in Apache Spark und Apache Flink - https://blog.codecentric.de/2017/04/event-zeit-verarbeitung-apache-spark-und-apache-flink/
- Frank Rosner: Window Functions in Stream Analytics - https://blog.codecentric.de/en/2018/10/window-functions-in-stream-analytics/
- Chengzhi Zhao: Reading Apache Beam Programming Guide (Series of Posts) - https://medium.com/@chengzhizhao/reading-apache-beam-programming-guide-1-overview-3adde0898b02 and following posts
- Daniel Foley: Let's Build a Streaming Data Pipeline - Apache Beam and DataFlow for real-time data pipelines - https://towardsdatascience.com/lets-build-a-streaming-data-pipeline-e873d671fc57
- Neil Stevenson: Running Apache Beam on Hazelcast Jet - https://hazelcast.com/blog/running-apache-beam-on-hazelcast-jet
- Software Crafters Slack - https://softwarecrafters.slack.com
- ThoughtWorks Radar Apache Beam - https://www.thoughtworks.com/de/radar/languages-and-frameworks/apache-beam
- Apache Beam Website - https://beam.apache.org/
- Spotify Scio Website - https://spotify.github.io/scio/
- Apache Flink Website - https://flink.apache.org/