

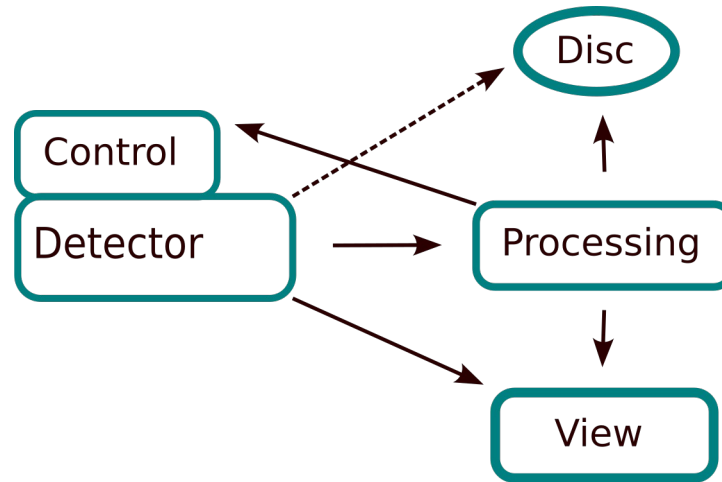
ASAPO: A High-performance streaming framework for real-time data analysis

Mikhail Karnevskiy

DESY IT

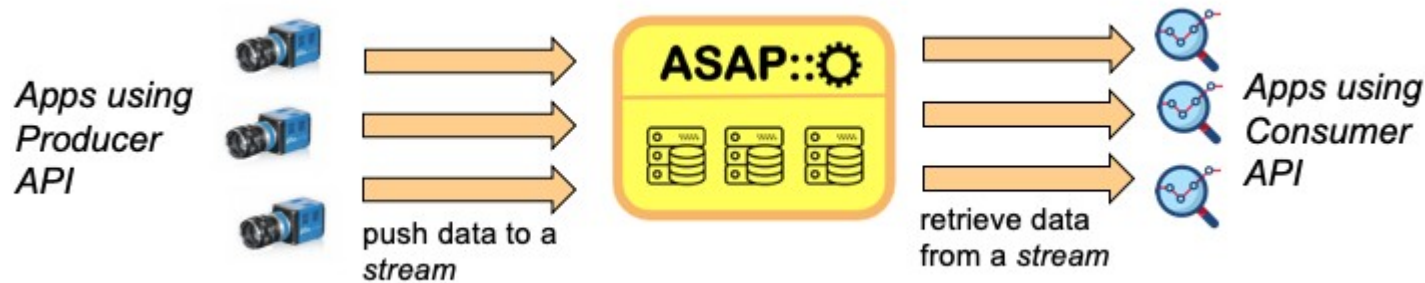
12 August, 2025

Motivation



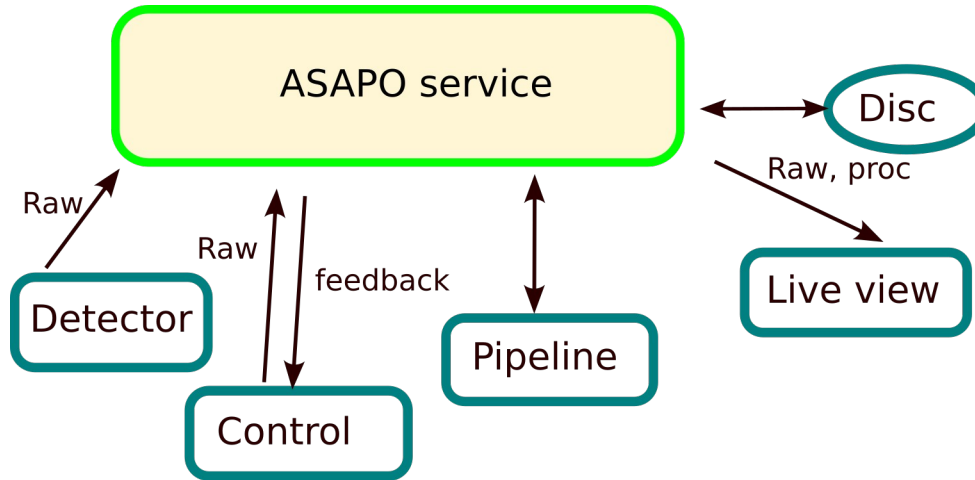
- High resolution and high speed detectors may require data reduction
- Feedback loop for efficient data taking
- Robust is data flow

Asapo introduction



- High-bandwidth communication between state-of-the-art detectors, control system, the storage system, and independent analysis processes across DESY facility
- In-memory data transfer with optional caching on disc
 - Large, scalable in-memory cache
 - Saving data to disc as service
 - Deliver data from disc
- Easy to use C++/Python interfaces:
 - Producer: sends data to Asapo
 - Consumer: get data from Asapo
- Stays for: Online data-processing. Data reduction. Feedback.

Asapo Data flow



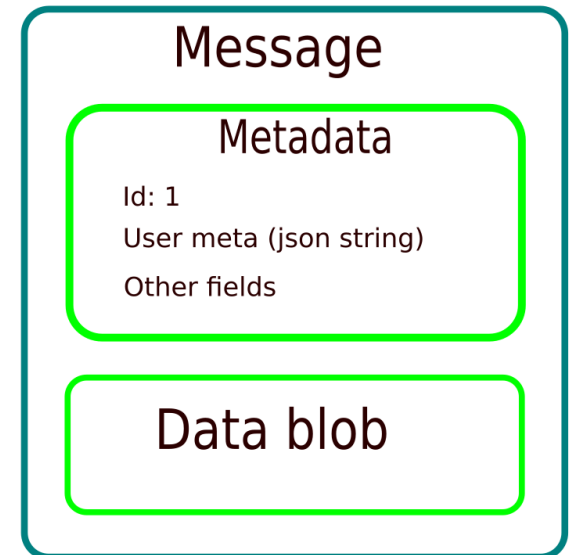
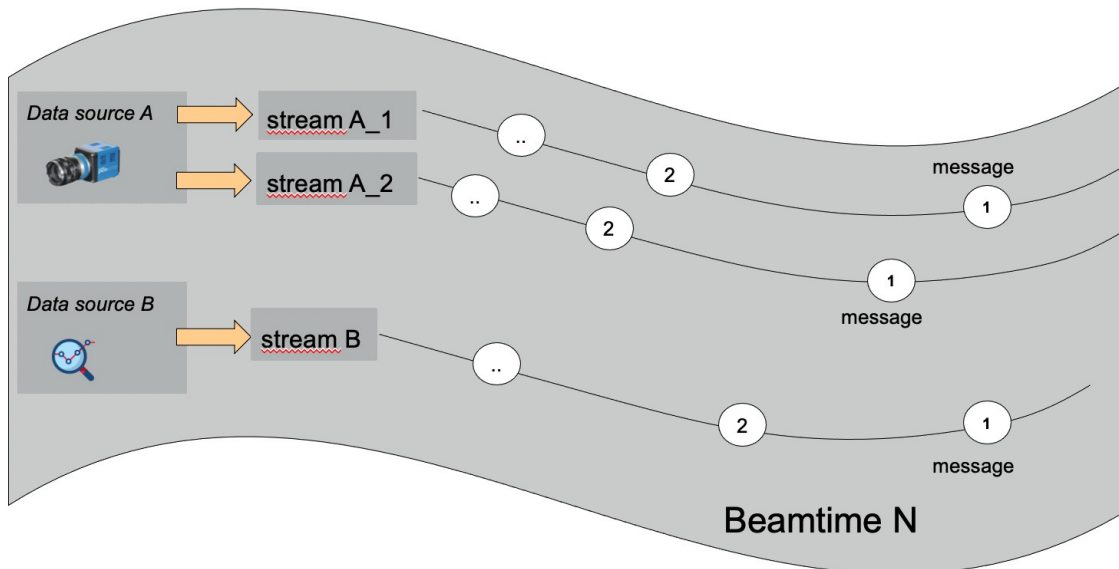
- Communication between different components across DESY facility
- Data-buffet with optional caching on disc
- Data writing goes in parallel with processing
- Multiple components are provided as a service for scientists

Challenges

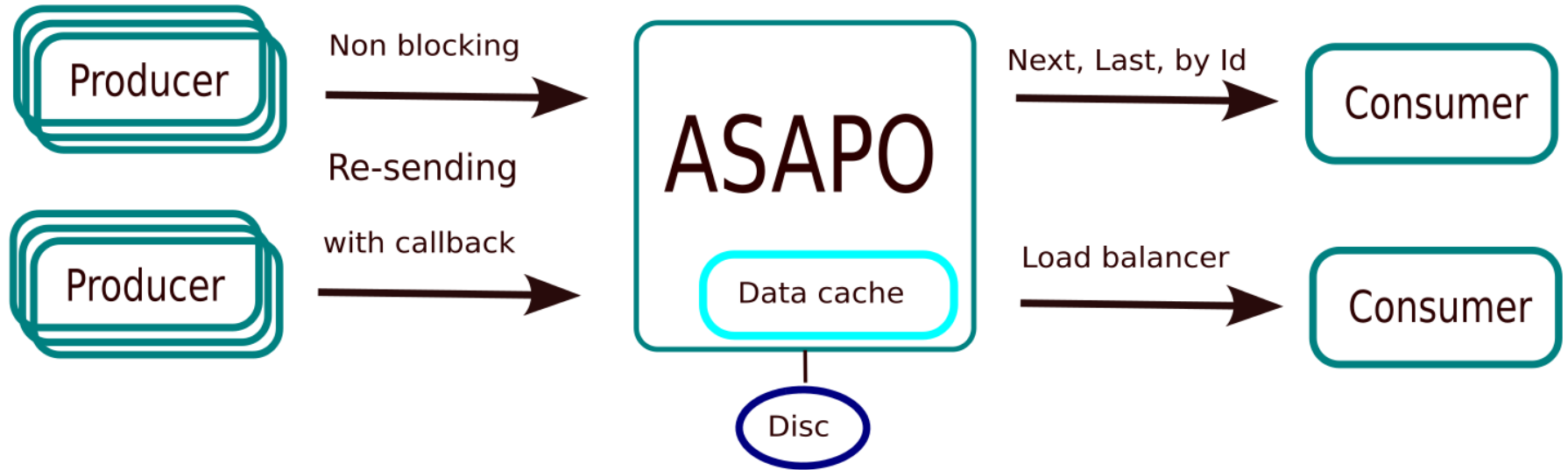
- Data-flow at kHz rate for 9M (highly compressed) images.
- Different detectors streams the data in different formats and being controlled in different ways
- Control data may come from different sources
- Data stream may not know, when it finishes
- Reprocessing may be requested at any time
- Minimum action from beamline scientists: data-processing as service
- Very little time for commissioning

Data in ASAPO

- Messages are indexes from 1 to N and form Stream.
- Streams are uniquely identified by its name, beamtime and data-source name
- Each message contains a binary data blob and a JSON metadata.
- Separate handling of data and metadata.
 - Data is stored in memory-cache and on disk, metadata is stored in database
 - This enables rich API and high throughput
- Synchronization of streams: several data-sources can be combined into a dataset



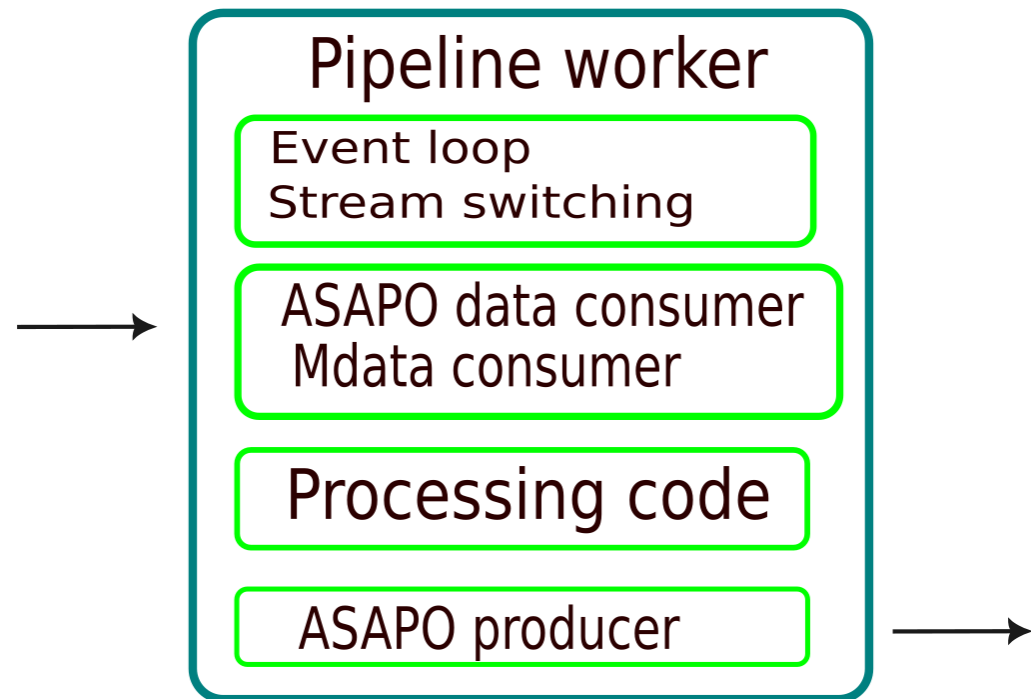
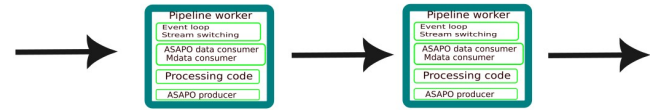
Asapo API



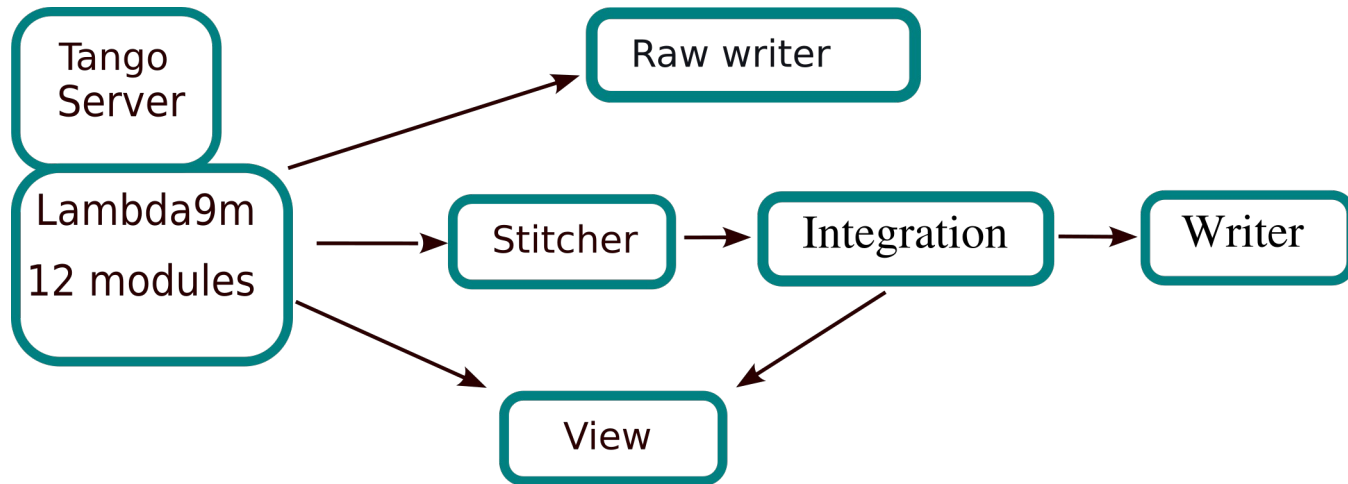
- Producers and consumer configured are fully independent
 - All components, including central service does not know, how many client are running
- Configuration is done using central endpoint, stream and data-source name

ASAPO-based pipeline

- Uses Python or Cpp ASAPO clients
- Data processing with a chain of workers
- Communication via ASAPO service
 - Workers does not know each other, but knows data-source to retrieve.

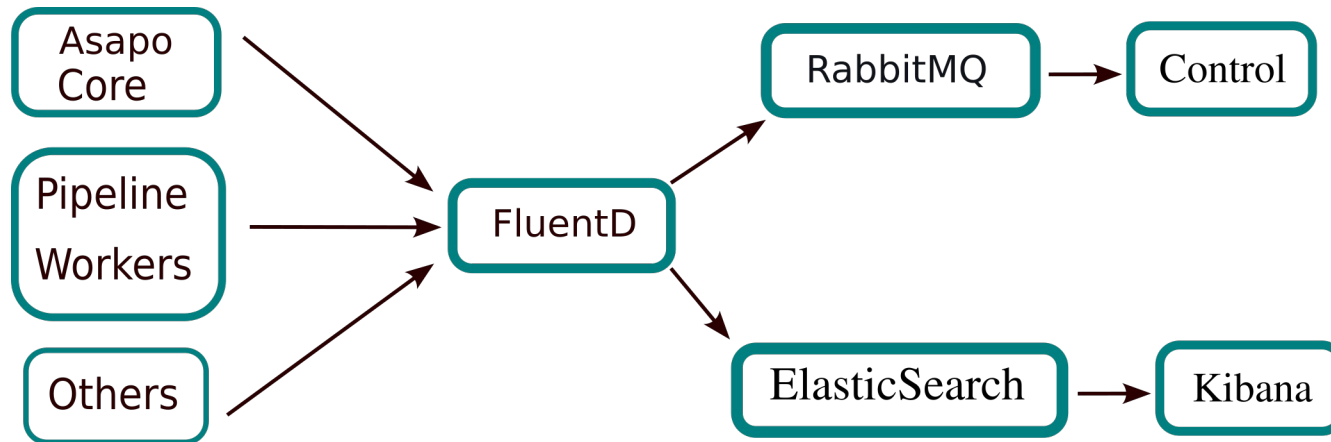


Pipeline example



- Tango servers ingest data from Lambda9M detector to Asapo mem-cache
- Raw data are saved in parallel with processing
- Pipeline stitch image from 12 detector modules and performs radial integration
- View can show live raw and processed data

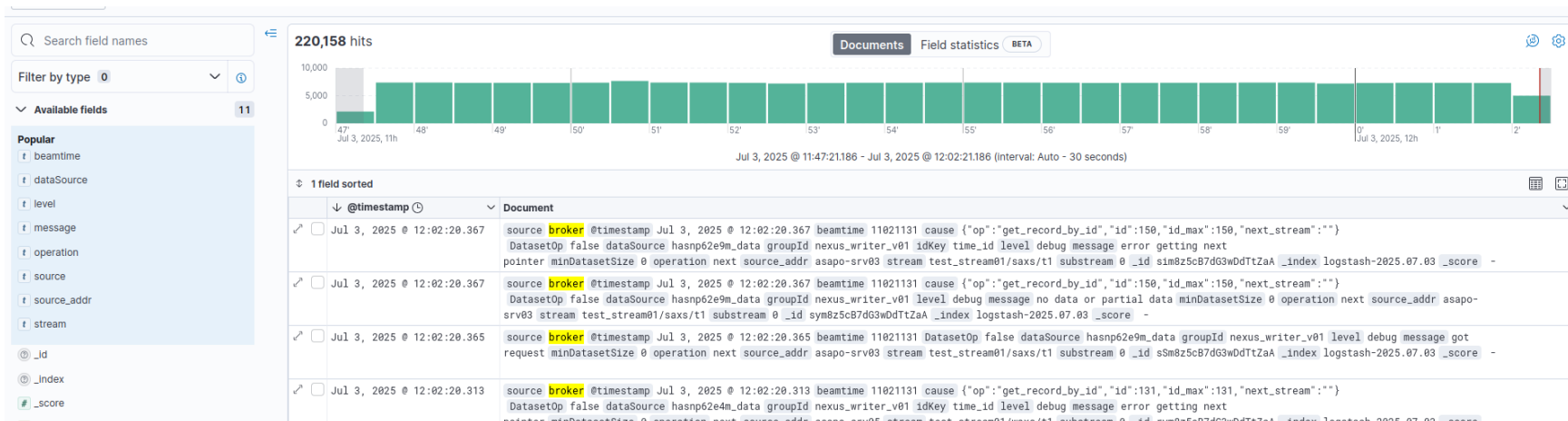
Log flow



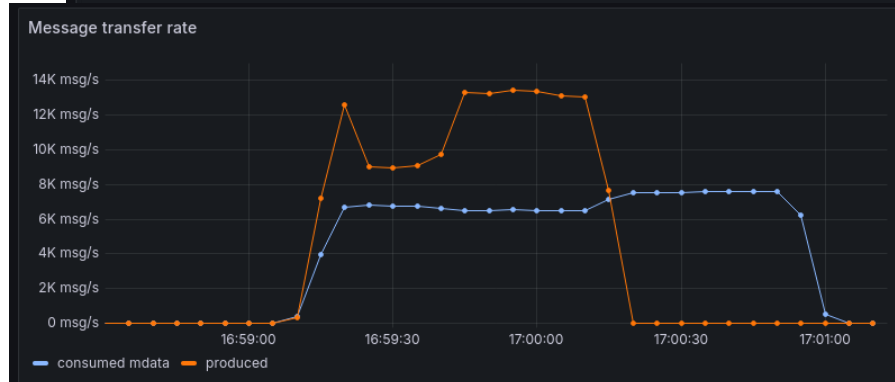
- Provide monitoring and debugging information
- Push feedback to control system
- Established using existing tools, but provided as service

Monitoring

- Monitoring is based on Grafana + InfluxDB
- Logging is based on Kibana and Elasticsearch



2025-02-06 09:41:26.770	data_source_c6e5cc16-663e-4761-8999-b833d5eda8d6_1	5001	unknown
2025-02-06 09:41:26.770	data_source_c6e5cc16-663e-4761-8999-b833d5eda8d6_0	5001	unknown



Try ASAPO



- Git at DESY: <https://gitlab.desy.de/asapo>



- Pipy client packages. 

- Docs: <https://asapo.pages.desy.de/asapo/>

ASAPO standalone service:

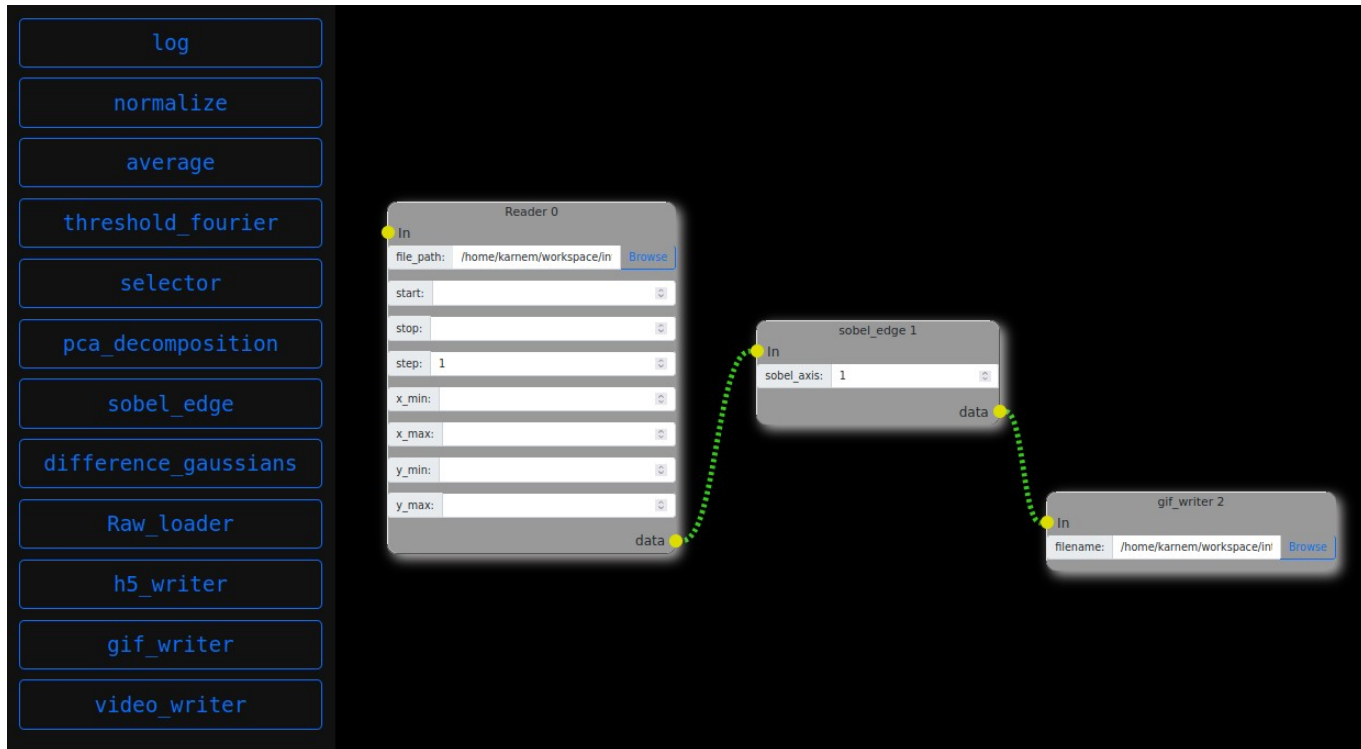
- Single docker with all asapo services
- Monitoring via Grafana  
- Limited functionality (not scalable)
- Fully functional API

Summary

- Asapo is a streaming platform and service provided by DESY-IT
- Service is user at several beamlines at Petra III to establish data-flow and enable online data-processing
- Reliable data-flow at different beamlines is established.
- Services are update few times per years to provide new features. Bugfixes-updates are possible during the user-run.
- Streaming-based data-flow is under commissioning.

Attachments

Single-node shared-mem pipeline



- Pure python implementation.
- Worker may have multiple instances. Each is a python process.
- Communication via multiprocessing queue and shared mem.