Data Pipeline Monitoring

Michiel Kalkman

Mental model of a pipeline

Figure 1: Actually a duct (*Source:wikimedia*)

Map of a real pipeline

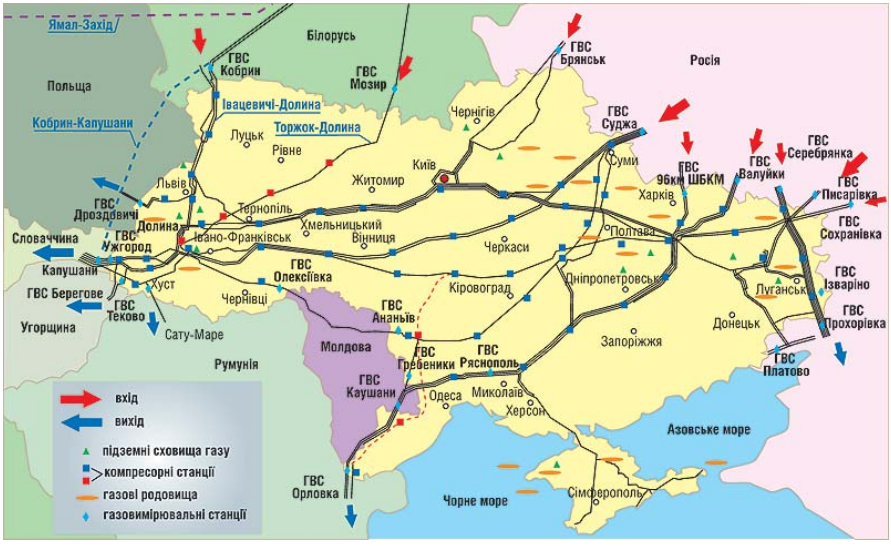

Figure 2: Typical pipeline (*Source:wikimedia*)

Notes

Pipelines,

- are systems
- cross multiple political zones
- cross multiple technical zones
- have multiple inputs (providers, sources)
- have multiple outputs (consumers, sinks)
- carry payloads in multiple stages (refinements)

Break it down

By administrative zones

Defines supportability, frames arguments over responsibility

Observability

Pillars of Observability

	Logs	Metrics	Tracing
Accounting		X	X
Reporting		X	X
Alerting		X	X
Testing	X	X	X
Diagnostics	X	X	X
Verification	X		X
Auditing	X		

What gets measured gets managed

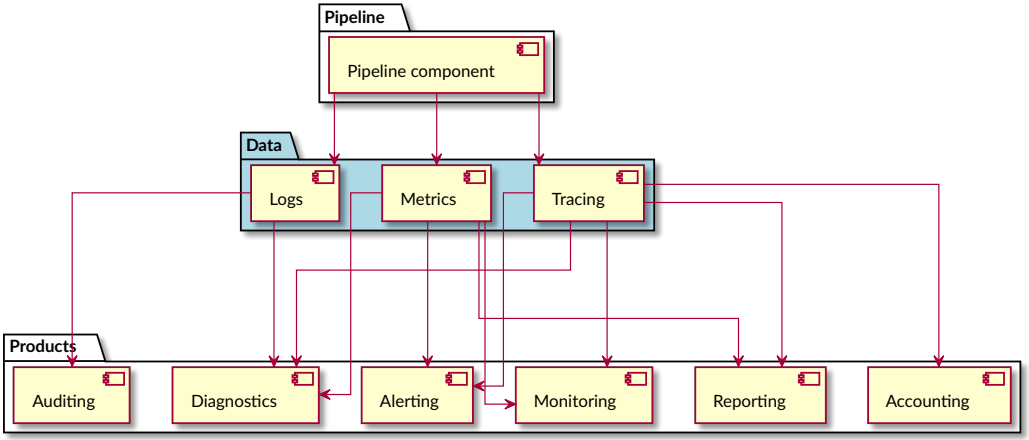

Figure 3: Component observability

What to monitor

- Data flowing across platform boundaries
- Cycles in the pipeline
- Data flow pressure points
- Baseline operation separate from service operation
- Infrastructure separate from service operation
- Quality control gateways for change

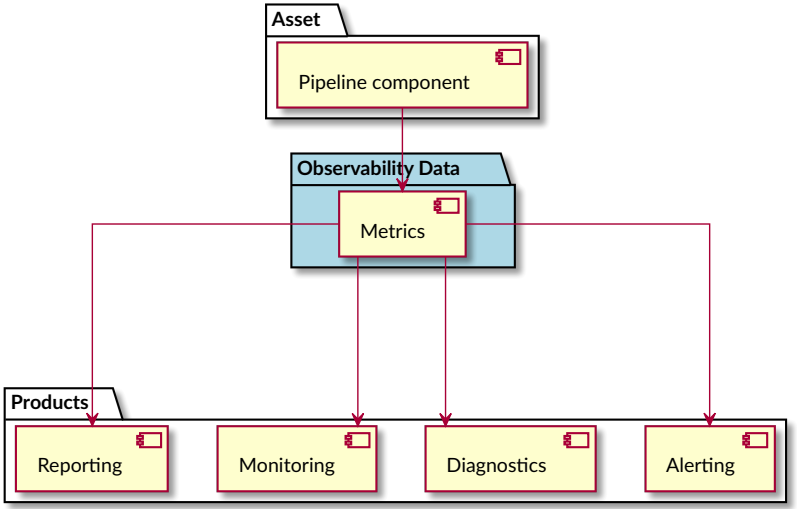

Figure 4: Observability - Metrics

Metrics focus

A wide variety of metrics out there. It's easy to get lost. Define high level metrics that can be compared consistenty across the entire landscape. Focus on two distinct areas. Different sides of the same coin,

Utilization, Saturation, Errors (**USE**)

These are ***resource*** *focused* and provide technical information

- *"Which servers are overloaded?"*

Rate, Errors, Duration (**RED**)

These are ***service*** *focused* and provide business information

- *"Am I meeting my SLA targets?"*

Four Golden Signals (Google SRE)

1. Latency
2. Traffic
3. Errors
4. Saturation

Throughput - components

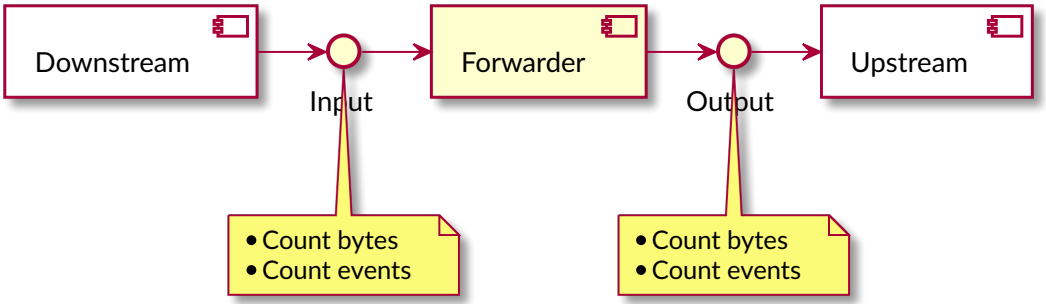

Figure 5: Component metrics

Throughput

Counter	t1	t2
input bytes	100	200
output bytes	150	270
input events	20	30
output events	30	55

- Throughput Rate is $(t2 - t1)$
- Average event size
  - $In = \frac{Rate(BytesIn)}{Rate(EventsIn)}$
  - $Out = \frac{Rate(BytesOut)}{Rate(EventsOut)}$
- Internal buffer pressure
  - $Rate(EventsIn) - Rate(EventsOut)$

Tracing

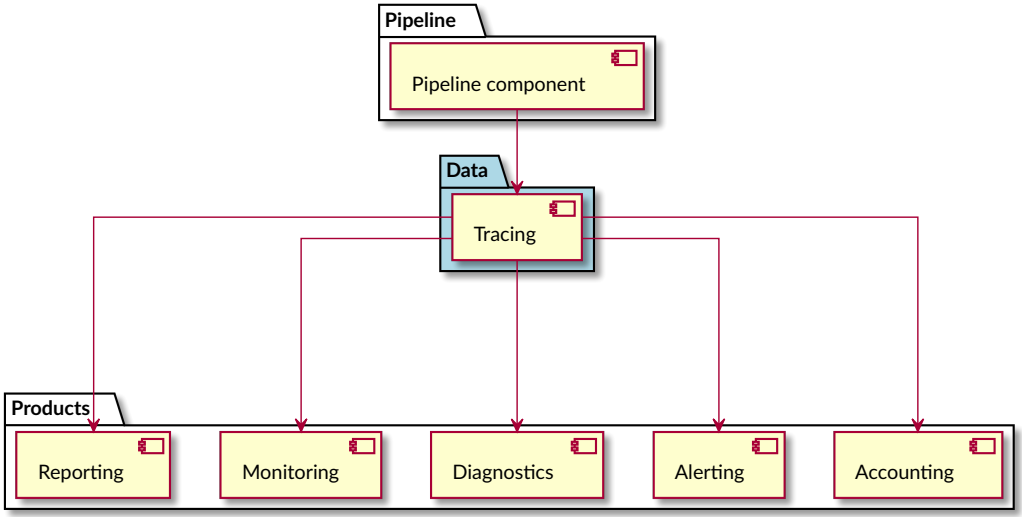

Figure 6: Observability - Tracing

Three Ts

	Inputs	Outputs	Transformation
Transaction	1	1	New data
Transportation	1	1+	Enrichment
Transformation	1+	1+	New data, enrichment

Transportation tracing

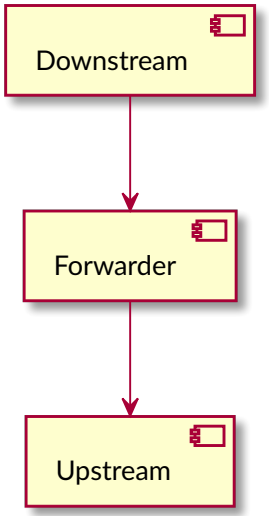

Figure 7: Transportation

Transaction tracing

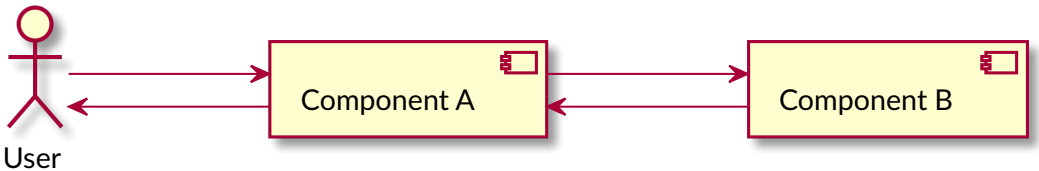

Figure 8: Distributed transaction

Transformation tracing

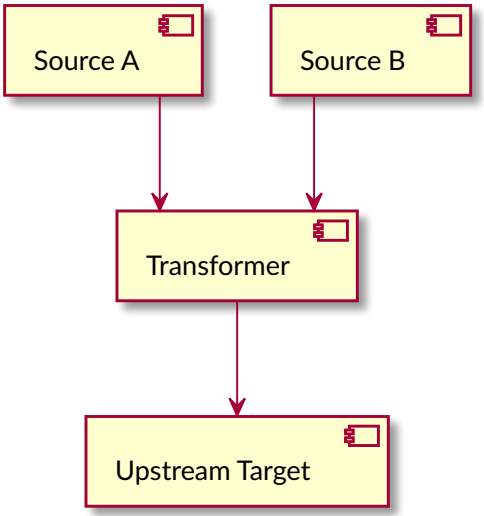

Figure 9: Transformation

Monitoring

Plan for failure

Figure 10: Hopefully not this bad (*Source:wikimedia*)

Key monitoring points

- Integrity
  - Packet/event/record drops
  - Timeouts, queue expiries
  - Data loss scenarios
- Capacity
  - Backpressure signaling
  - Backlog processing
  - Peak hour spikes

Heartbeats

- Add a dummy input channel to each input
- Continuously generate fixed data at fixed rate
- Monitor dummy channel on each boundary
- Alert on dummy channel rate at each boundary

Buffers, Backlogs and Backpressure

MQ pipeline with push - dataflow

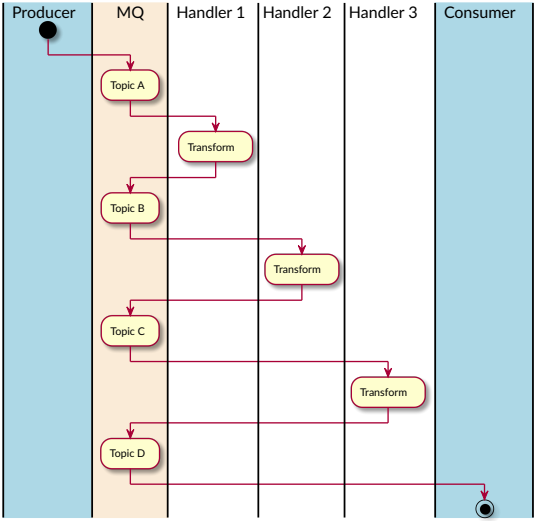

Figure 11: MQ pipeline with push - dataflow

MQ pipeline with push - sequence

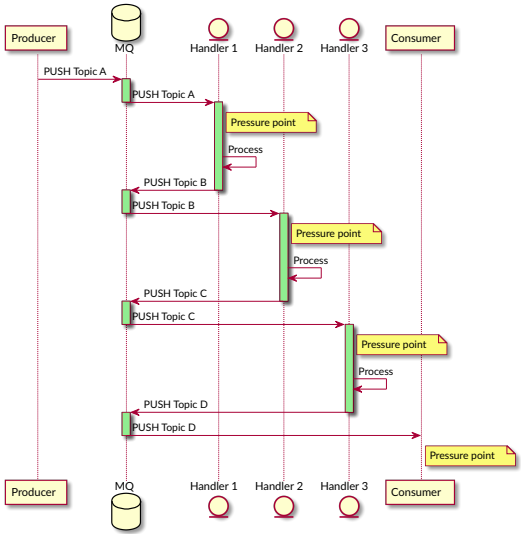

Figure 12: Kafka pipeline with push - sequence

MQ pipeline notes

- This design is **active** here, sends data as it comes in
- Server-push model for moving data
  - *Yes, you can also poll a queue*
- Complex programming model
  - MQ-specific protocol
  - Requires registration of callback
  - Handler process might be unavailable

Model

```
def next(records_in, buffer_size, output_capacity):
  buffer_size = buffer_size + records_in

  if ((buffer_size - output_capacity) >= 0):
      records_out = output_capacity
      buffer_size = buffer_size - output_capacity
  else:
      records_out = buffer_size
      buffer_size = 0

  plot(records_in, buffer_size, records_out)
  return buffer_size
```

Input rate =< output capacity

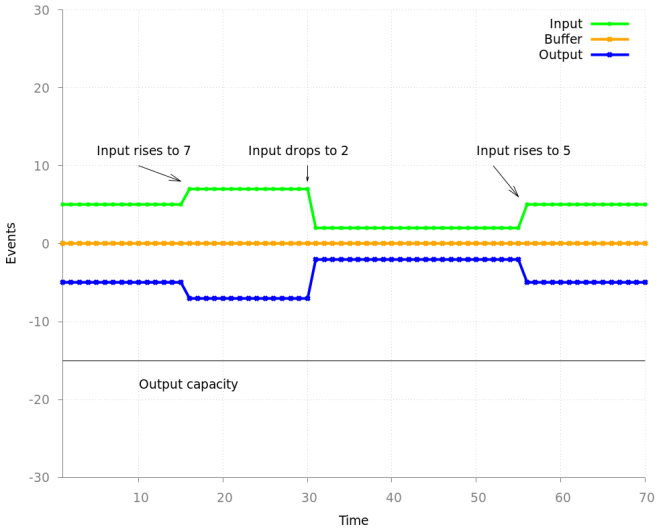

Figure 13: Output capacity = 15 eps

Backlog processing

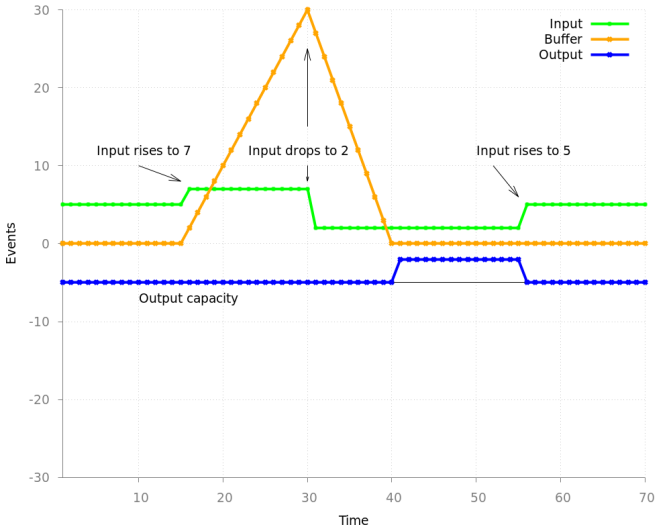

Figure 14: Output capacity = 5 eps

Backlog processing with finite buffer

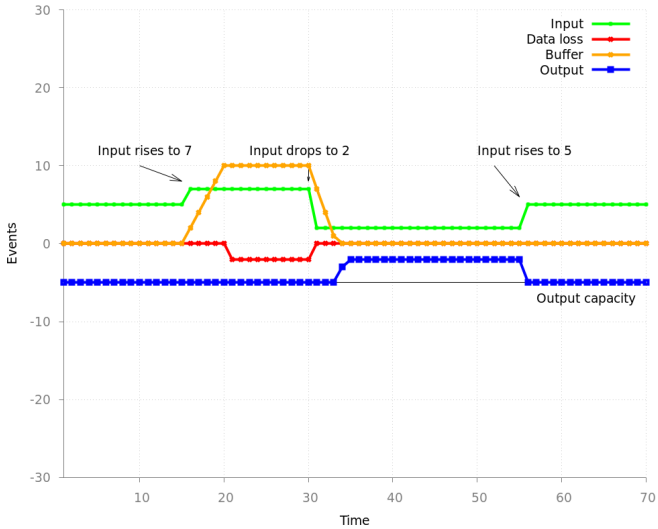

Figure 15: Limit reached with no backpressure means **data loss**

Observing buffer change rate

	t1	t2	t3	t4
$Counter(In)$	5	12	19	26
$Counter(Out)$	5	10	15	20
$Rate_{In}(t)$	N/A	7	7	7
$Rate_{Out}(t)$	N/A	5	5	5
$Rate_{In}(t) - Rate_{Out}(t)$	N/A	**2**	**2**	**2**

$$Rate(n) = Counter(n) - Counter(n-1)\ Buffer(n) = Rate_{In}(n) - Rate_{Out}(n)$$

Buffer change rate

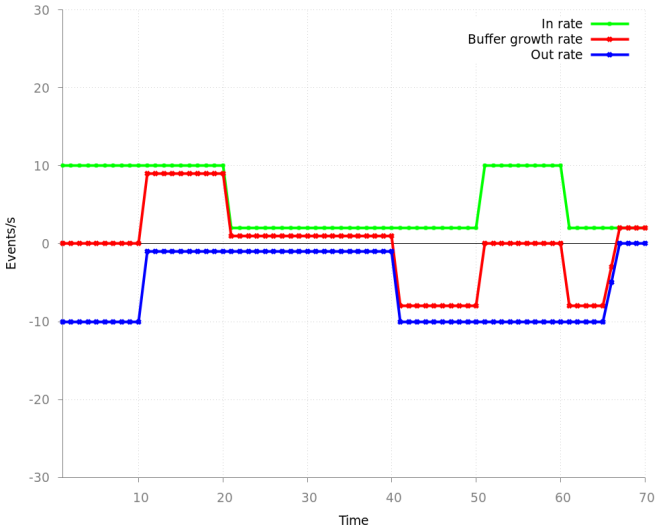

Figure 16: Long term average of the red line should approach zero

Kafka pipeline - Dataflow - by asset

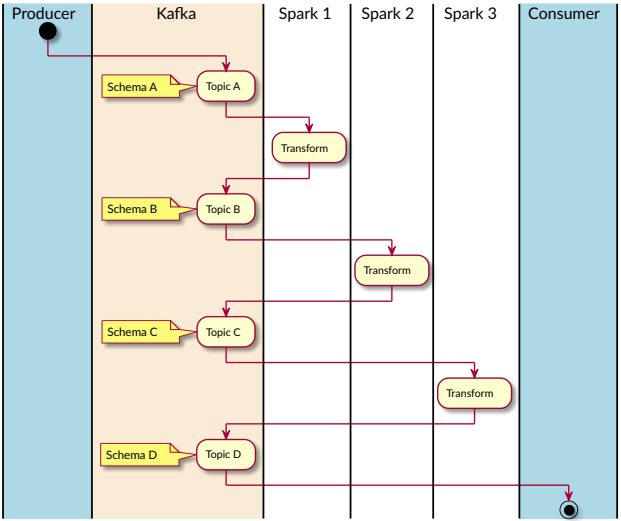

Figure 17: Kafka pipeline Dataflow

Kafka pipeline - connection initiation - by asset

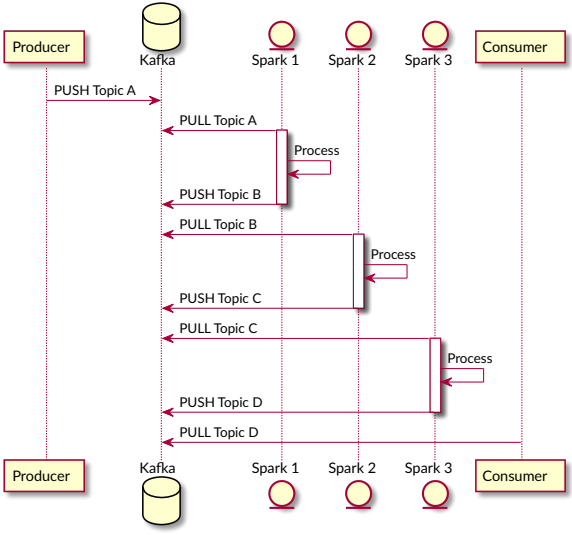

Figure 18: Kafka pipeline sequence

Kafka pipeline - Dataflow - by service

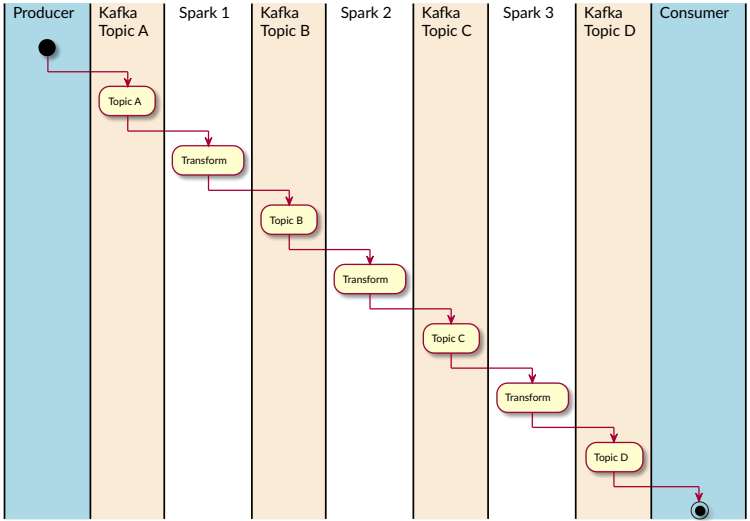

Figure 19: Kafka pipeline Dataflow

Kafka pipeline - connection initiation - by service

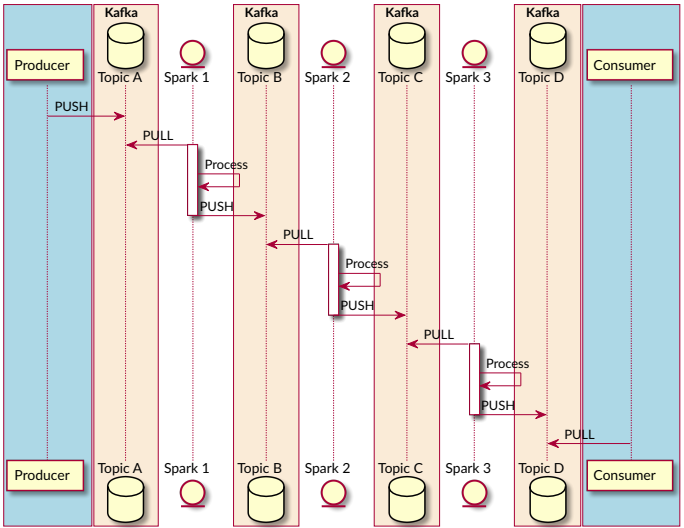

Figure 20: Kafka pipeline sequence

Kafka pipeline notes

- This design is **passive**, does not send data unless asked
- Client-pull model for moving data
- All persistence is done on Kafka
- Very simple programming model
- Well understood wire-protocol (HTTP)