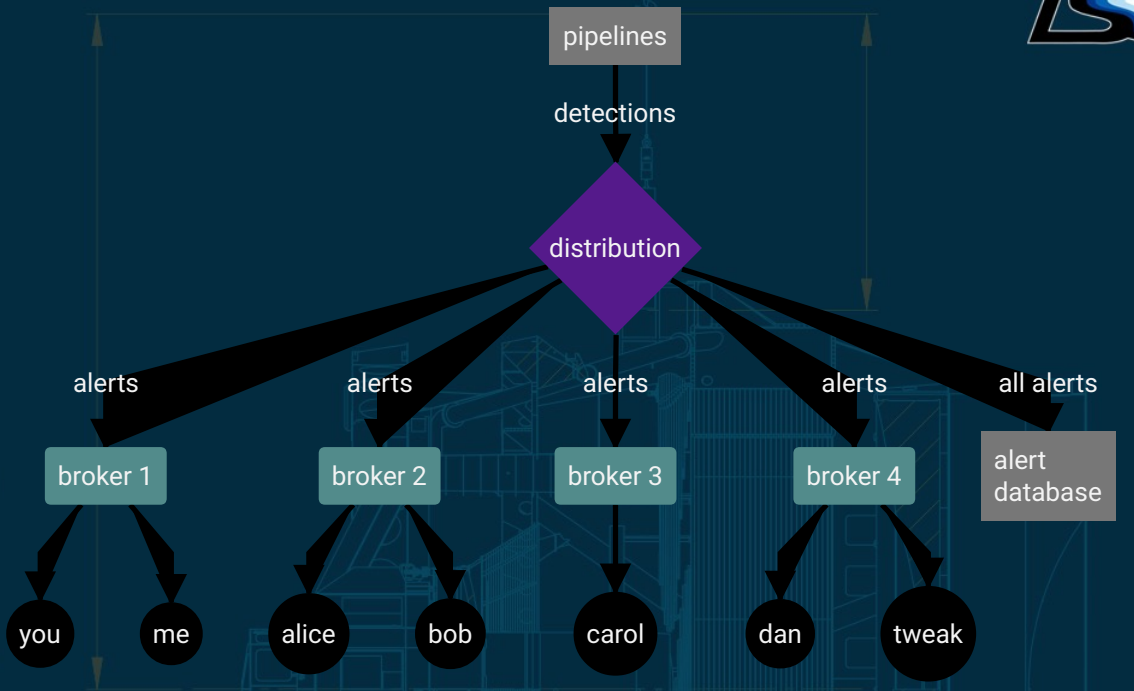
Alert Filtering System Prototyping for ZTF and LSST

Maria Patterson

IVOA northern interop, Shanghai 2017 May

High Level

Goal

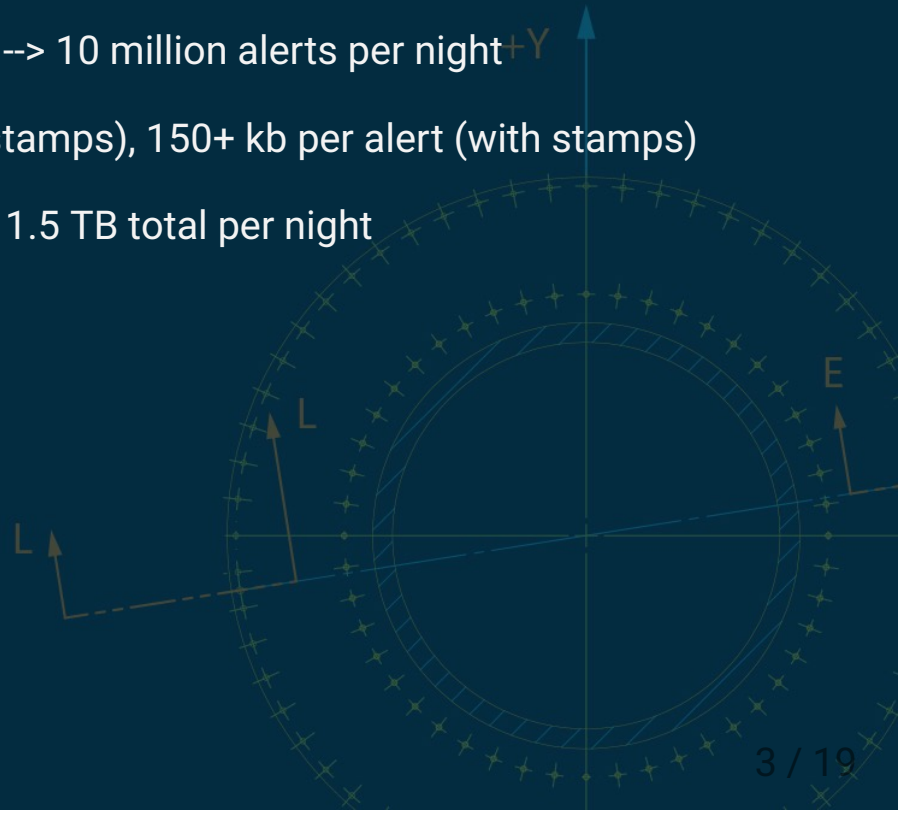


Needs

1. Support scalability with addition of brokers
2. Sink records to some alert archive accurately
3. Not lose any data
4. Package and send postage stamp cutouts
5. Allow simple filtering using Python or SQL-like interface (mini-broker)

For LSST, alert stream is large

- 10,000 alerts per visit
  - OSS-REQ-0193 "Minimum number of alerts required to be accommodated from a single standard visit"
- 39 second visit intervals
- 1,000 visits per night --> 10 million alerts per night
- 50+ kb per alert (no stamps), 150+ kb per alert (with stamps)
- potentially 600 GB to 1.5 TB total per night



For ZTF, alert stream is also sizable

- 1,000 alerts per visit
- 45 second visit intervals
- 1,000 visits per night --> 1 million alerts per night
- 20+ kb per alert (includes 3 cutouts)
- --> 20-40 GB total per night
- http://voeventdb.4pisky.org/apiv1/count : 1,603,506

Transport system: Apache Kafka

- Scalability
- Replication
- Allows stream "rewind"

Data formatting: Apache Avro

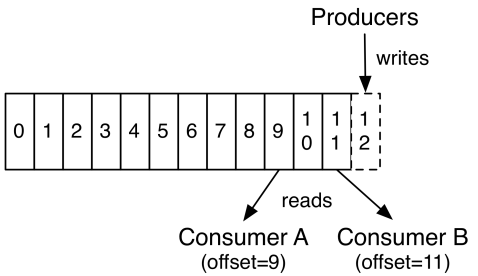
- Fast parsing with structured messages (typing)
- Strictly enforced schemas, but schema evolution
- Allows postage stamp cutout files

Filtering component: Apache Spark

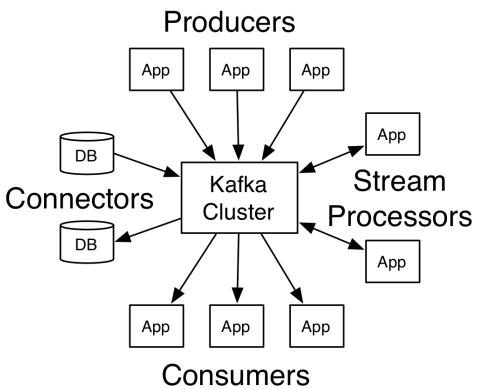
- Direct connection to transport system
- Stream interface similar to batch
- Allows for Python or simple SQL-like queries

Transport prototype: Apache Kafka

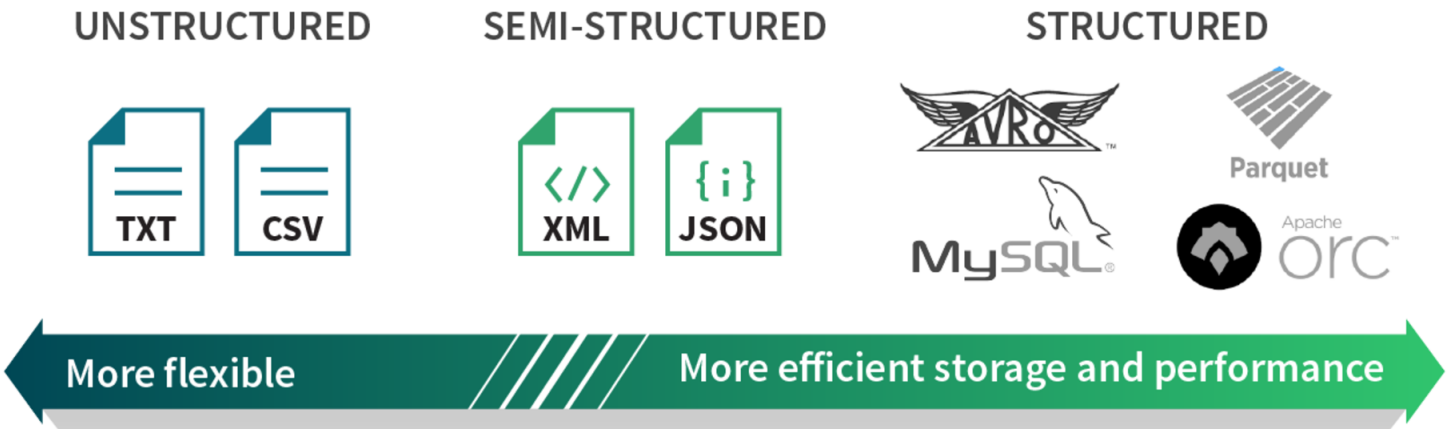
- Distributed log system/ messaging queue
- Reinvented as strongly ordered, pub/sub streaming platform



- Highly scalable, in production at LinkedIn, Netflix, Microsoft
- Great clients + connectors, including Python - good usability



Data formatting: Apache Avro



Alert event prototype: Apache Avro

- Schemas defined with JSON
- Flexible format- schema evolution
- Dynamic typing- strict adherence
- Also used in production, science, recommended by Kafka

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

```
{"name": "Ben",
 "favorite_color": "red",
 "favorite_number": random.randint(0,10)}
```

Simulated alerts in Avro format

https://github.com/ZwickyTransientFacility/ztf-avro-alert

Includes cutout jpgs (or any file format)

Schema

```
alert_schema = '''
{
    "namespace": "ztf",
    "type": "record",
    "name": "alert",
    "doc": "sample avro alert schema v1.0",
    "fields": [
        {"name": "alertId", "type": "long", "doc": "add descriptions like this"},
        {"name": "candid", "type": "long"},
        {"name": "candidate", "type": "ztf.alert.candidate"},
        {"name": "prv_candidates", "type": [{
                "type": "array",
                "items": "ztf.alert.prv_candidate"}, "null" ], "default": null},
        {"name": "cutoutScience", "type": ["ztf.alert.cutout", "null"], "default":
        {"name": "cutoutTemplate", "type": ["ztf.alert.cutout", "null"], "default"
        {"name": "cutoutDifference", "type": ["ztf.alert.cutout", "null"], "defaul
            ]
}
'''
```

Simulated alerts in Avro format

https://github.com/lsst-dm/sample-avro-alert

Includes cutout jpgs (or any file format)

```
  stamp_schema = '''

   {"namespace": "ztf.alert",
    "type": "record",
    "name": "cutout",
    "fields": [
      {"name": "filename",
       "type": "string"},
      {"name": "stampdata",
       "type": "bytes"}
     ]
   }

'''
```

```
cutout_file = 'stamp-54720.fits'
with open(cutout_file, mode='rb') as f:
   stampdata = f.read()
```

Mock alert producer stream with Kafka

https://github.com/lsst-dm/alert_stream

```
streamProducer = AlertProducer(topic, schema, **kafkaConf)

def send_batch():

    for i in range(number_of_alerts):

        streamProducer.send(dict_data, avroEncode=True/False)

    streamProducer.flush()
```

- Configures connection to Kafka broker and sets stream topic
- Takes dict (with stamps)
- Can turn Avro encoding on or off
- Uses single repeated "alert" in testing
- Sends configurable batch size continuously every 39/45 s

Template alert consumers

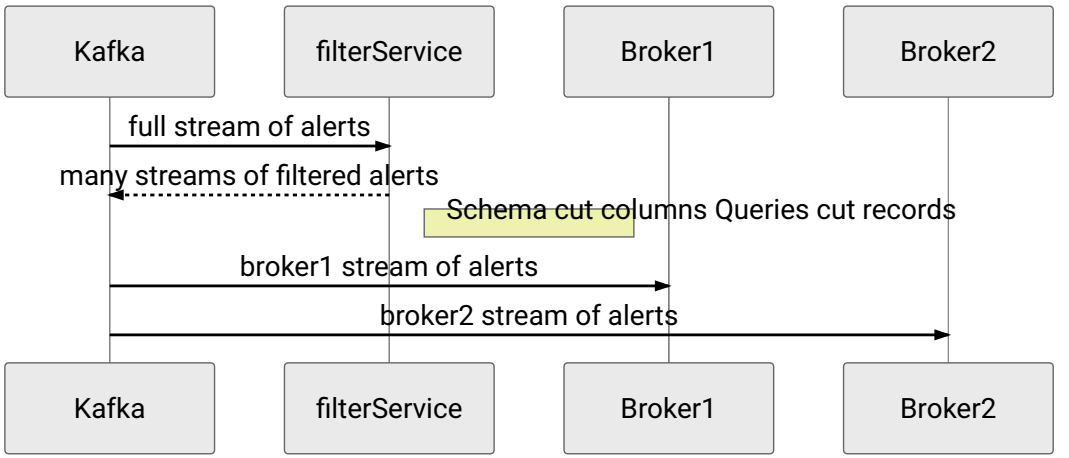
https://github.com/lsst-dm/alert_stream

```
streamReader = AlertConsumer(topic, schema, **kafkaConf)

while True:

  msg = streamReader.poll(avroDecode=True/False)

      if msg is None:
          continue
      else:
          print(msg)
```

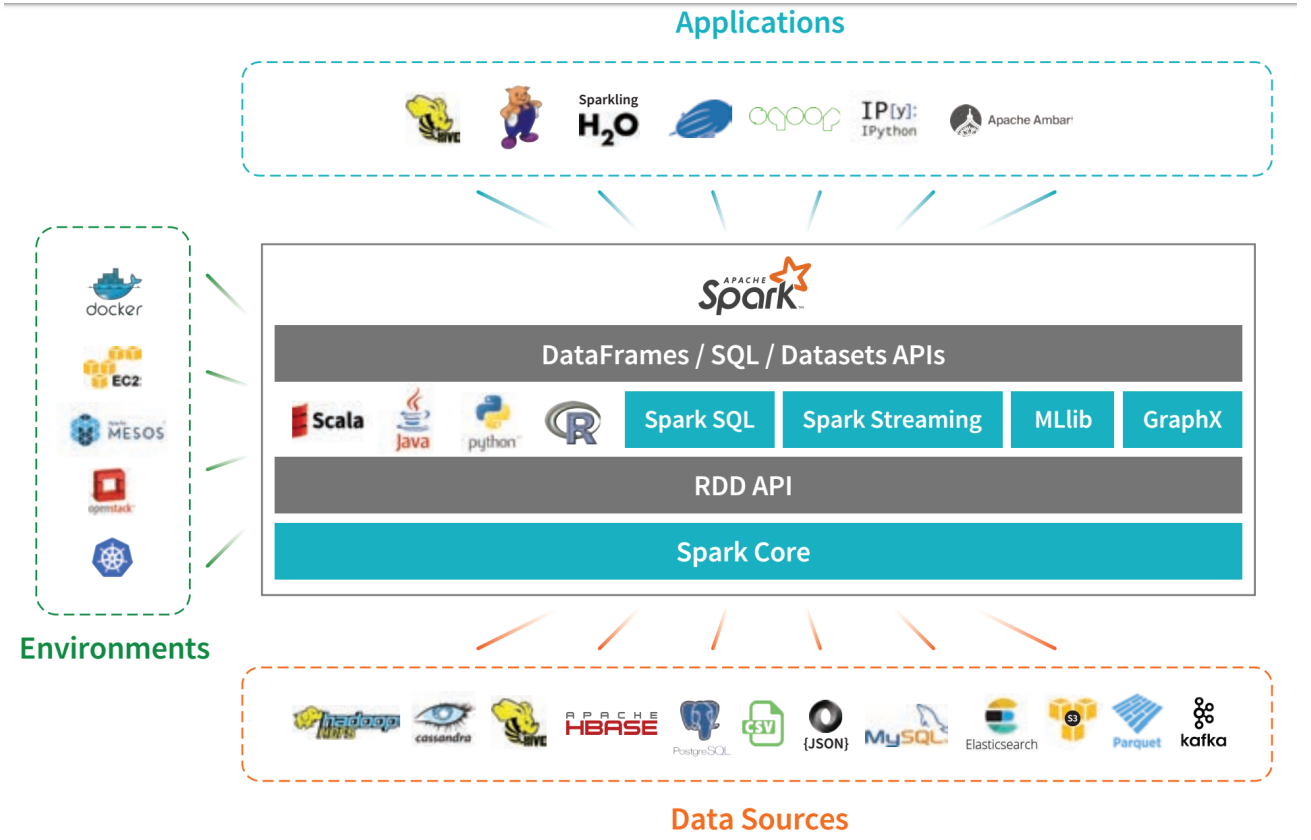
- Configures connection to Kafka broker and subscribes to topic
- Script for printing alert content
  - Avro decoding on or off
  - Stamp collection on or off
- Script for monitoring the stream (drops content)

Possible data flow

Design



Spark, possible filtering technology



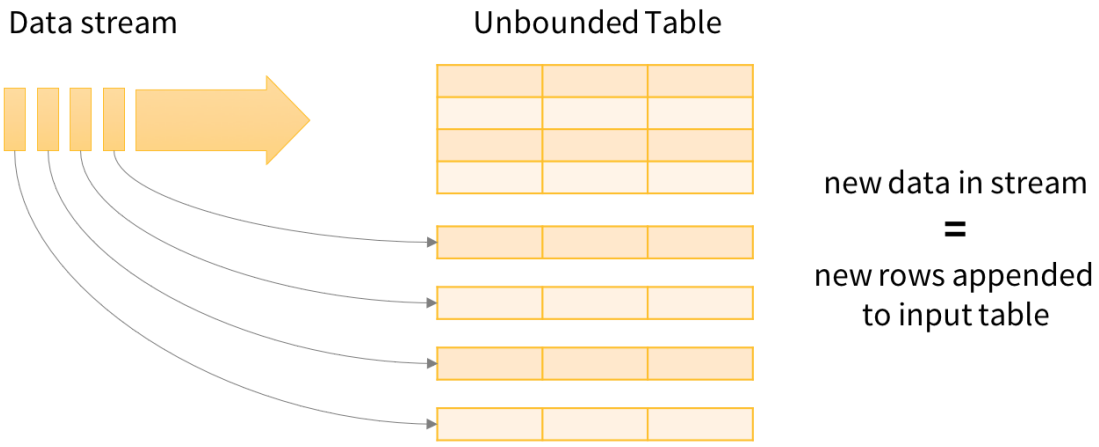
Interacting with batch and stream the same way is a huge win

Property	Structured Streaming	Spark Streaming	Apache Storm	Apache Flink	Kafka Streams	Google Dataflow
Streaming API	incrementalize batch queries	integrates with batch	separate from batch	separate from batch	separate from batch	integrates with batch
Prefix Integrity Guarantee	✔	✔	✘	✘	✘	✘
Internal Processing	exactly once	exactly once	at least once	exactly once	at least once	exactly once
Transactional Sources/Sinks	✔	some	some	some	✘	✘
Interactive Queries	✔	✔	✘	✘	✘	✘
Joins with Static Data	✔	✔	✘	✘	✘	✘

Filtering

Design

Tech



Data stream as an unbounded Input Table

Filtering the stream https://github.com/lsst-dm/filtering-blueprints

Set up connection

```
from pyspark.streaming.kafka import KafkaUtils

kafkaStream = KafkaUtils.createDirectStream(
                ['my-stream'],
                {'bootstrap.servers': 'kafka-server:9092',
                'auto.offset.reset': 'smallest',
                'group.id': 'spark-group' })

alerts = kafkaStream.map(lambda x: x[1])
```

Example map filter definition

```
def map_alertId(alert):
    return alert['alertId']
```

Apply the map filter

```
alertIds = alerts.map(map_alertId)  # apply map
alertIds.pprint()  # print to the screen
sparkStreamingContext.start()
```

Filtering the stream https://github.com/lsst-dm/filtering-blueprints

Set up connection

```
from pyspark.streaming.kafka import KafkaUtils

kafkaStream = KafkaUtils.createDirectStream(
                ['my-stream'],
                {'bootstrap.servers': 'kafka-server:9092',
                'auto.offset.reset': 'smallest',
                'group.id': 'spark-group' })

alerts = kafkaStream.map(lambda x: x[1])
```

Example filter filter definition

```
def filter_Ra(alert):
  return alert['diaSource']['ra'] > 350
```

Apply the filter filter

```
alertsRA = alerts.filter(filter_Ra)  # apply filter
alertsRA.pprint()  # print to the screen
sparkStreamingContext.start()
```

ZTF and LSST system prototype tech and features

- Transport: Apache Kafka
  - Scalable, replicated, allows for stream "rewind"
- Serialization: Apache Avro
  - Fast parsing with structured messages (typing), strictly enforced schemas, but allows for schema evolution
- Filtering: Apache Spark
  - Direct connection to Kafka, batch-like stream interface (Python or SQL-like)

Additional use cases / bonuses

- For small data streams but very unique events, rewindability is big bonus
- These tech choices put no hard requirements on programming language
- Wide-support (outside of astro) enhances flexibility, usability, sustainability

Resources / Opportunities

- Deploy mock stream testbed Dockerized with Kafka
  - https://github.com/lsst-dm/alert_stream
- Sample Avro data reference for ZTF and LSST (VOEvent evolution?)
  - https://github.com/lsst-dm/sample-avro-alert
  - https://github.com/ZwickyTransientFacility/ztf-avro-alert
- See Jupyter notebook output for how-to filtering in Spark
  - https://github.com/lsst-dm/filtering-blueprints