Introduction to

Apache Streams™

ApacheCon Europe, September 2015

sblackmon@apache.org
Agenda

- Problem: Proliferation
- Activity Streams
- Apache Streams
- Compatibility
- Schemas
- Resources
Problem: Proliferation!
It’s challenging to get a composite picture of a person or organization because data resides in many systems that are not easily integrated.
Standards

S We have no universally adopted standard for structuring social profiles, or for transmitting activities across data silos.

S This is true across web sites, as well across enterprise applications.
Most silos make minimal if any effort to promote interoperability by publishing machine-readable schemas for their APIs, or supporting standardized data formats.
Many data silos recommend usage of one of their SDKs to use their data services, however:

These SDKs impose their preferred libraries (such as HTTP clients and json libraries) on us without actually making development easier.
We have an unprecedented range of choices for how and where we store data.

Developers often have a handful they prefer to use, and aren’t eager to learn the protocols and assumptions of a new DB.

Many applications require a polyglot architecture to scale.
Frameworks

S Frameworks can be very helpful when building scalable systems, but they all enforce conventions and have constraints.

S Frameworks lead to lock-in, unless your team is extra-ordinarily vigilant.
Running code in the cloud may be cheaper, but runtime-specific variation impacts the way we:

- Package
- Deploy
- Configure
- Monitor

Runtimes lead to lock-in, unless your team is extra-ordinarily vigilant.
A public specification for describing digital activities and identities in JSON format

1.0 – 2011

2.0 – WIP
Activity Streams Objectives

S Language agnostic
S Cross-application interoperability
S Support for multiple schemas
S Stream Federation
S Stream Filtering
Activity Streams Basics

Normalized form for entities and events

<actor> did <verb> with <object> (to <target>) at <published>

objectTypes

Person, Organization, Image, Video, etc...

Verbs

Post, Share, Like, etc...
Implementation Challenges

- **Adoption**
  - Industry support has been tepid at best

- **Ambiguity**
  - The spec itself is open to interpretation

- **Extensions**
  - The spec rightly allows for arbitrary extensions

- **Flexibility**
  - As a result, activities from any two providers are just barely interoperable

- **Validation**
  - Data correctness or coherence is not covered by spec
Apache Streams

- A lightweight (yet scalable) framework for Activity Streams
- An SDK for building data-centric JVM applications
- A set of patterns for building reliable, adaptable, data processing pipelines
Philosophies

- Be Database agnostic
- Be Runtime agnostic
- Enforce task and document serializability
- Documents as the core unit of processing
- Support any type of documents and arbitrary metadata
- Encourage explicit specification of documents via json schema and xml schema
- Assist with conversion to and from activitystrea.ms
Interfaces

**S Provider**
- Task running within Activity Streams deployment that sources documents for the stream, likely in their original data format.

**S Processor**
- Task running within Activity Streams deployment that transforms documents, perhaps with a synchronous call to an external system.

**S Persist Reader**
- Task running within an Activity Streams deployment that sources documents from a file system or database.

**S Persist Writer**
- Task running within an Activity Streams deployment that saves documents to a file system or database.
Compatibility Dimensions

S Providers
S Persistence
S Pipelines
S Runtimes
S Schemas
Compatibility: Providers

Datasift, Facebook, GMail, Gnip, Google Plus, Instagram, Moreover, RSS, Sysomos, Twitter, YouTube
Compatibility: Persistance

- Buffer (file system)
- Cassandra
- Elasticsearch
- Graph (neo4j)
- HBase
- HDFS
- MongoDB
- Kafka
- Kinesis
- S3
Compatibility: Runtime Frameworks

S Docker
S Dropwizard
S Pig
S Spark
S Storm
Compatibility: Runtime Roadmap

- Crunch
- Flink
- Logstash
- NiFi
- Samza
- Spark Streaming
- Twill
Compatibility: Schemas

Schemata are:
The presence and absence of fields and structure
Different from class and from format

Strategies for Schema Management
Many-to-Many
Many-to-Mine
Many-to-One

Schema-related Challenges
Schema Management: Many-To-Many

For every provider and type, map and convert to compatible types from all other providers

This is the default modality for data and it sucks
Schema Management: Many-To-Mine

Specify internal types, then for every provider and type: assess, align and convert to preferred internal representation.

This is better, but it fails as soon as we want to interoperate with other departments or organizations who are all using their own internal schemas.

Expect to change your internal spec relatively often in early stages, meaning you probably have to either upgrade your data or guarantee backward compatibility in-application.
For every provider and type, a community dedicated to the inter-operability of that dataset sorts out a reasonable mapping to a relatively static public specification.

Where the existing public specs are inadequate, the community can find a way to establish compatibility via convention.

Open-source communities and standards bodies can collaborate for benefit of all.
Schema Challenges: Sharing

Business-as-usual:
S Schemas are often implicit, shared via unstructured web documentation and language specific sdks

Streams:
S Streams source code contains json and xml schemas for many supported providers
S Anyone can import or extend these schemas (via HTTP!)
Schema Challenges: Date-Times

Business-as-usual:
Here’s a string, have fun!

Streams:
Every library on the classpath declares its preferred format(s)
Framework resolves any known format and uses Joda to convert to RFC3339
Schema Challenges: Versioning

S Business-as-usual:
S Schemas change as product and API features evolve, and everyone just muddles through.

S Streams:
S Schemas get published with every release and every snapshot for benefit of those responsible for dependent libraries
S Changes get described in release notes
S Updates to unit and integration tests
Schema Challenges: IDE Support

S Business-as-usual:
S Import our SDK or GTFO

S Streams:
S All streams types have a Serializable POJO representation
S Importable with maven to specific version
S Convertible to ancestor, sibling, and child types with a cast
S Convertible to other types with a one-liner
**Schema Challenges:**

**Imports**

**Business-as-usual:**

Every service is an island

**Streams:**

‘Extends’ capability of json schema allows for emergence of a web of related types

Describe your objects as a delta to base schemas or a mashup of several

Undeclared fields propagate by default
**Schema Challenges: Conversion**

**Business-as-usual:**
- Either get too much type safety or none, take your pick
- If you’re lucky, framework helps with serialization and compression

**Streams:**
- Includes multiple type conversion options, available as processors for your streams or singleton utility classes to embed in your code
  - jackson conversion
  - hocon conversion
  - via java/scala
Resources

Website
http://streams.incubator.apache.org/

Source Code
https://github.com/apache/incubator-streams

Documentation
http://streams.incubator.apache.org/site/0.2-incubating/streams-project/index.html

Examples
https://github.com/apache/incubator-streams-examples

Examples Documentation
http://streams.incubator.apache.org/site/0.2-incubating-SNAPSHOT/streams-examples/index.html