Building Stream Processing as a Service (SPaaS)

Steven Wu

@stevenzwu

NETFLIX
Why stream processing?
Unbounded user activity stream

Alice

Bob

Jane
Unbounded data - batch

Feb 25

Alice

Bob

Jane

Feb 26
Unbounded data - batch

Feb 25

Alice

Bob

Jane

Feb 26
Unbounded data - batch

February 25

Alice

Bob

Jane

February 26

[Diagram showing data representations for Alice, Bob, and Jane on both February 25 and February 26]
Unbounded data - batch

Feb 25

Alice

Bob

Jane

Feb 26

Feb 25

Feb 26
Unbounded data - stream

Alice

Bob

Jane
Agenda

- Introduction
- Apache Flink primer
- SPaaS Overview
- Keystone Router
- Custom Stream Processing Applications
- Backfill and Rewind
Agenda

- Introduction
- Apache Flink primer
- SPaaS Overview
- Keystone Router
- Custom Stream Processing Applications
- Backfill and Rewind
Real Time Data Infrastructure

Keystone Data Pipeline

Topics  Routing  Transformations (filter, projection, UDF)  Schema / Hygiene

Producers → Messaging As A Service (Kafka) → Consumers

Stream Processing As A Service (Flink)
Stream Processing

Keystone Data Pipeline

- Topics
- Routing
- Transformations (filter, projection, UDF)
- Schema / Hygiene

Producers

Messaging As A Service (Kafka)

Stream Processing As A Service (Flink)

Consumers
highly available ingest pipelines - the backbone of a real-time data infrastructure
Agenda

- Introduction
- **Apache Flink primer**
- SPaaS Overview
- Keystone Router
- Custom Stream Processing Applications
- Backfill and Rewind
Exact-once semantics for stateful computation

Source: http://flink.apache.org/
Flexible windowing

Source: http://flink.apache.org/
Event time semantics

Source: http://flink.apache.org/
State backends and checkpointing

Available
- Memory
- File system
- RocksDB (support incremental checkpoint)

Source: http://flink.apache.org/
Checkpoint is lightweight

Source: http://flink.apache.org/
Levels of abstraction

- Stream SQL (high-level language)
- Table API (dynamic tables) (declarative DSL)
- DataStream API (streams, windows) (stream processing & analytics)
- Process Function (events, state, time) (low-level (stateful stream processing))
Agenda

- Introduction
- Apache Flink primer
- **SPaaS Overview**
- Keystone Router
- Custom Stream Processing Applications
- Backfill and Rewind
Offerings by complexity

- Simple **drag and drop**: filter, projection, data hygiene
  - *Available now via Keystone router*
- Medium: **SQL, UDF** (User Defined Function)
  - *Coming 2018*
- Advanced: **custom** stream processing applications
  - *Available now*
Ease of use v.s. capability

Ease of use

difficult
easy

Capability

Limited

Full feature

Color legend

Available now

Coming 2018

Keystone Router
Ease of use v.s. capability

- **Keystone Router**
  - Limited

- **Custom SPaaS App**
  - Full feature
  - Available now
  - Coming 2018

Color legend:
- Green: Available now
- Red: Coming 2018
Ease of use v.s. capability

- Keystone Router
- Custom SPaaS App
- SQL

Color legend:
- Green: Available now
- Red: Coming 2018
Ease of use v.s. capability

- **Easy**
  - Limited
  - Keystone + UDF
  - SQL + UDF

- **Difficult**
  - Custom SPaaS App

**Color legend**
- Green: Available now
- Red: Coming 2018
SPaaS running on Titus
(Netflix’s in-house container runtime)
Job isolation: single job

Flink standalone cluster

Titus Job #1

Job Manager

Titus Job #2

Task Manager

Task Manager

Task Manager
Agenda

- Introduction
- Apache Flink primer
- SPaaS Overview
- **Keystone Router**
- Custom Stream Processing Applications
- Backfill and Rewind
Events are published to fronting Kafka directly or via proxy.
Events land up in fronting Kafka cluster

- Event Producer
- HTTP / gRPC
- KS Gateway
- Fronting Kafka
- Keystone Management
- Flink Router
- Stream Consumers
- S3
- elasticsearch
- Consumer Kafka
- HIVE
Events are polled by router, filter and projection applied
Router sends events to destination

Event Producer

KS Gateway

Fronting Kafka

Keystone Management

Flink Router

Consumer Kafka

S3

elasticsearch

Stream Consumers
Keystone pipeline system boundary

Event Producer

HTTP / gRPC

KS Gateway

Fronting Kafka

Keystone Management

Flink Router

Consumer Kafka

Stream Consumers

Kafka Event Producer

Kafka Stream Consumers

HTTP / gRPC

Flink Router

KS Gateway

Fronting Kafka

Keystone Management

S3

HIVE

elasticsearch
highly available ingest pipelines
Keystone scale

- >1,000,000,000 unique events ingested per day
- >99.9999% of delivery rate
Demo: provision a data stream (mini pipeline)
Stream `sa ny 2018`

Owner: foo@netflix.com

Description: demo

PROD

US-EAST-3

Enable Region...

TEST

US-EAST-3

Stream is queued for automatic provisioning in this region; an email notification will be sent to the owner when it has been created.

Producers

Keystone

Hive
(new output)

Keystone

Events sent to Keystone are routed to one or more configured Outputs.
Outputs can be added using the "Stream Actions" menu.
Configure outputs
Drag-and-drop Keystone router

- Stateless and embarrassingly parallel
- ~2,000 jobs in prod
- Self serve and fully managed
- At least once delivery semantics
- Isolation
Agenda

- Introduction
- Apache Flink primer
- SPaaS Overview
- Keystone Router
- Custom Stream Processing Applications
- Backfill and Rewind
Out-Of-The-Box Functionality

- Templates (Java / Scala)
- Build and Deployment tooling
- Connectors
- Dashboards
- Logs
- Alerts
- Titus Integration
- Capacity Management
Demo: SPaaS project bootstrap
Hello, stevenwu.
We are going to generate a SPaaS Streaming Processing Job template.

Using /Users/stevenwu/tmp/sa_ny_2018 to initialize git repository...

? Initialize which directory for spaas-job project? .
? Would you like me to set up a Stash repo? Yes
? Would you like me to set up Jenkins jobs? Yes
? Enter the name of your Stash project (the parent group for the repo), ~stevenwu
? Enter the name of your Stash project (the parent group for the repo), ~stevenwu
   for personal project: ~stevenwu
? Enter the name of the Stash repo: [? for help] (sa_ny_2018)
Skeleton code

createSource("example-kafka-source")
    .addSink(getSink("null-sink")).name("null-sink");
Add business logic

createSource("example-kafka-source")
  .keyBy(<key selector>)
  .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
  .reduce(<window function>);
  .addSink(getSink("hive-sink").name("hive-sink"));
Demo: create a new Flink job
Override source config

Override Kafka cluster VIP

example-kafka-source  Job  nullsink

Kafka Source - example-kafka-source

<table>
<thead>
<tr>
<th>Name</th>
<th>Template Value</th>
<th>Optional Override</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic Name</td>
<td>clevent_ihs</td>
<td>kafka-prod:2181</td>
</tr>
<tr>
<td>Vip</td>
<td>kafka-test:2181</td>
<td></td>
</tr>
</tbody>
</table>

Override Kafka cluster VIP
Override job config

Example Kafka Source

Resources

- spaas.job.name: saNy_2018
- spaas.job.namespace: spaas.saNy_2018
- aNy_2018.flink.checkpoint.interval: 60000
Configure resources

Specify the number of resources required to run this job.

<table>
<thead>
<tr>
<th>Resources</th>
<th>CPU</th>
<th>Network (Mbps)</th>
<th>Memory (MB)</th>
<th>Disk Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers</td>
<td>8</td>
<td>1000</td>
<td>27000</td>
<td>54000</td>
</tr>
</tbody>
</table>

Job Properties
Configure multiple sources or sinks

**Kafka Source - memberevents**

<table>
<thead>
<tr>
<th>Name</th>
<th>Template Value</th>
<th>Optional Override</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic Name</td>
<td>ab_member_events</td>
<td></td>
</tr>
<tr>
<td>Vip</td>
<td>kafka-test:2181</td>
<td>kafka-prod:2181</td>
</tr>
</tbody>
</table>
Deep links
Duplo blocks

- Filter
- Projector
- Data Hygiene
- Connectors
Supported Source and Sink Connectors

Sources
- Kafka
- Hive

Sinks
- Elasticsearch
- Kafka
- Hive
- Keystone
Agenda

- Introduction
- Apache Flink primer
- SPaaS Overview
- Keystone Router
- Custom Stream Processing Applications
- Backfill and Rewind
Things can go wrong
Application bug

Source → Flink Streaming Job → Sink
Sink failure
Dependency service failure

Source → Flink Streaming Job → Sink

Data Enrichment

Micro Service
How to recover

- Backfill *(available now)*
- Rewrite Flink job *(coming soon)*
How to recover

- Backfill
- Rewind Flink job
Live job continues

Kafka -> Live Job

outage period -> Now -> Sink

Time
Hive as backfill source

Kafka

Live Job

Backfill Job

Sink

outage period

Now ->

Time
Choose Hive source

- **hive_source**
- **kafka_source**
- **hive_sink**

**Job**

- **Properties**
- **Resources**
- **Security Groups**

**PROPERTY**

- **source.chosen**: hive_source
Configure Hive source

### Hive Source - hive_source

<table>
<thead>
<tr>
<th>Name</th>
<th>Template Value</th>
<th>Optional Override</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>default</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>clevent</td>
<td></td>
</tr>
<tr>
<td>Where</td>
<td>dateint=20180201 and hour=0</td>
<td></td>
</tr>
</tbody>
</table>
Not a lambda architecture

- Single streaming code base
- Just switch source from Kafka to Hive
Hive backfill probably not for stateful jobs

- Warm-up issue
- Ordering issue
Hive backfill probably not for stateful jobs

- Warm-up issue
- Ordering issue
Stateful stream processor

Image adapted from Stephen Ewen
Warm-up period
No output emit during warm-up
Hive backfill probably not for stateful jobs

- Warm-up issue
- Ordering issue
Kafka: messages ordered within a partition

Anatomy of a Topic

Partition 0
0 1 2 3 4 5 6 7 8 9 1 1 1
0 1 0 1 2

Partition 1
0 1 2 3 4 5 6 7 8 9

Partition 2
0 1 2 3 4 5 6 7 8 9 1 1 1
0 1 0 1 2

Old       New

Source: kafka.apache.org
Hadoop input split

files  f0  f1  f2  f3  f4
Hadoop input split

<table>
<thead>
<tr>
<th>files</th>
<th>f0</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>splits</td>
<td>s0</td>
<td>s1</td>
<td>s2</td>
<td>s3</td>
<td>s4</td>
</tr>
<tr>
<td></td>
<td>s5</td>
<td>s6</td>
<td>s7</td>
<td>s8</td>
<td>s9</td>
</tr>
</tbody>
</table>
Hadoop input split

files
- f0
- f1
- f2
- f3
- f4

splits
- s0
- s1
- s2
- s3
- s4
- s5
- s6
- s7
- s8
- s9

Job Manager

Split calculation

s0 ... s9

Task Manager

Task Manager

Task Manager
Hadoop input split

files
f0  f1  f2  f3  f4

splits
s0  s1  s2  s3  s4  s5  s6  s7  s8  s9

Job Manager

Task Manager
s0  s3  s6  s9

Task Manager
s1  s4  s7

Task Manager
s2  s5  s8

Split Assignment

Split calculation
Where is the order?

files
f0 (hour=0)  f1  f2 (hour=23)  f3 (hour=12)  f4 (hour=3)

splits
s0  s1  s2  s3  s4  s5  s6  s7  s8  s9

Task Manager
s0 (hour=0)  s3 (hour=23)  s6 (hour=12)  s9 (hour=3)

...  ...
Does time/ordering matters?

- Probably not for stateless computation
- Probably important for stateful computation
Window with allowed lateness

DataStream<T> input = ...;

input
   .keyBy(<key selector>)
   .window(<window assigner>)
   .allowedLateness(<time>)
   .<windowed transformation>(<window function>);
How to recover

- Backfill
- Rewind Flink job
Flink checkpoint and fault tolerance

Checkpoint \( x-1 \)  \rightarrow  \text{Application}  \leftarrow  \text{Checkpoint } x  \rightarrow  \text{State}  \rightarrow  \text{Now}
Flink checkpoint and fault tolerance
Flink rewind

Checkpoint y  →  Checkpoint x  →  Checkpoint x+1

outage window

Now
Flink rewind

Checkpoint $y$  
Checkpoint $x$  
Checkpoint $x+1$

outage window
Kafka retention

As far as we can go back

outage window  Kafka retention  Now

Time
## Hive backfill v.s. Flink rewind

<table>
<thead>
<tr>
<th></th>
<th>Hive backfill</th>
<th>Flink rewind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warm-up issue</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Ordering issue</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data retention</td>
<td>Months</td>
<td>Hours or days</td>
</tr>
<tr>
<td>Applicability</td>
<td>Stateless</td>
<td>Stateless and stateful</td>
</tr>
</tbody>
</table>
Pros for Hive backfill source

- Long-term storage (a few months)
- Fast recovery
  - S3 is very scalable
  - Runs in parallel with live job
Today's recommendation

Stateless

Stateful

Hive backfill

Flink rewind
Is this the future?

Stateless

Hive backfill

Stateful

Flink rewind
Or is this the future?

Stateless  ::  Stateful

Flink rewind
Caveats for reprocessing

- Does not overwhelm external services
- Non-retractable sink output
- Non-replayable dependencies
Does not overwhelm external services

- Source
- Flink Streaming Job
- Sink
- Micro Service

10x load
Non-retractable sink output

- Duplicates are ok
- Idempotent sink
- Cleanable sink
  - e.g. drop Hive partition with bad data
Non-replayable dependencies

Source -> Flink Streaming Job -> Sink

Flink Streaming Job -> A/B Service

Time
Non-replayable dependencies

Source ➔ Flink Streaming Job ➔ Sink

A/B Service

Process live msg X

Time
Non-replayable dependencies

Source -> Flink Streaming Job -> Sink

A/B Service

Alice?

Process live msg X
Non-replayable dependencies

Source → Flink Streaming Job → Sink

Flink Streaming Job:
- Alice?
- Cell A

A/B Service:
- Process live msg X
Non-replayable dependencies

Flink Streaming Job

Source -> Flink Streaming Job -> Sink

A/B Service

Allocation change

Process live msg X

Time
Non-replayable dependencies

Source \rightarrow \text{Flink Streaming Job} \rightarrow \text{Sink}

\text{A/B Service}

Process live msg X
Allocation change

outage period

Time
Non-replayable dependencies

- Source
- Flink Streaming Job
- A/B Service
- Sink

Process live msg X
Allocation change
Rewind

outage period

Time
Non-replayable dependencies

Source → Flink Streaming Job → Sink

A/B Service

Process live msg X
Allocation change
Rewind
Reprocess old msg X

outage period

Time
Non-replayable dependencies

Source → Flink Streaming Job → Sink

A/B Service

Alice? → Process live msg X

Cell B → Allocation change

outage period

Rewind → Reprocess old msg X

Time
Convert table to stream

Source → Flink Streaming Job → Sink

Flink Streaming Job → A/B Service

Table lookup
Convert table to stream

A/B data becomes part of app state
Stream Kong
Putting together
SPaaS Layered Cake

<table>
<thead>
<tr>
<th>Dashboards</th>
<th>Metrics &amp; Monitoring</th>
<th>Keystone Routers (with UDF)</th>
<th>SQL (with UDF)</th>
<th>Streaming Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Reusable Components</td>
<td>Stream Processing Platform (Flink Streaming Engine, Config Management)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Titus Container Runtime</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amazon EC2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Management Service & UI | Streaming Job Development
IT'S SO EASY

A CAVE MAN CAN DO IT
Thank you!