Apache Flink Adoption at Shopify

Yaroslav Tkachenko

shopify

👋 Hi, I’m Yaroslav

Staff Data Engineer @ Shopify (Data Platform: Stream Processing)

Software Architect @ Activision (Data Platform)

Engineering Lead @ Mobify (Platform)

Software Engineer → Director of Engineering @ Bench Accounting (Web Apps, Platform)

sap1ens sap1ens.com

Shopify creates the best commerce tools for anyone, anywhere, to start and grow a business.

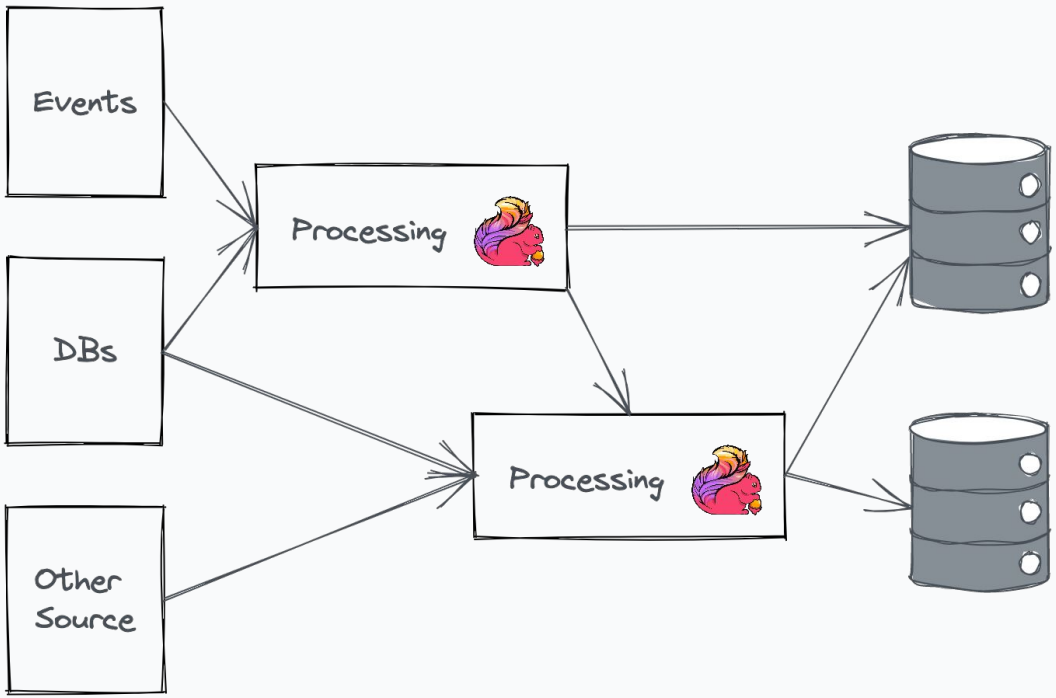
1.7 Million+
NUMBER OF MERCHANTS

~175 Countries
WITH MERCHANTS

~$356 Billion
TOTAL SALES ON SHOPIFY

7,000+
NUMBER OF EMPLOYEES

*What I'm **NOT** going to cover today*



~~"The Pipeline"~~

Instead, we want to make building and operating ***Flink*** *applications* ***as easy*** *as building and operating* ***Rails*** *applications*

We Need Realtime Data Products

- Reporting & Insights
- Product Analytics
- Data Integration
- Data Enrichment
- Sessionization
- …

Everywhere

- Sales & Orders
- Inventory
- Marketing
- Billing
- Customer Behaviour
- Messaging
- Mobile
- 3rd-party APIs
- …

Why Apache Flink?

- We’ve been building streaming applications for many years: Spark Structured Streaming, Beam, in-house tools.
- None of the ways supports large complex stateful transformations.
- None of the ways feels like “just building another app”.

*How do you build a data **platform**?*

Three Levels of Platforms

1. Ecosystem
2. Managed Platform
3. “Serverless” Platform

Three Levels of Platforms

1. Ecosystem

Combination of libraries, tools and **standalone** services.

Examples: Apache Spark/Flink + related tooling.

2. Managed Platform

3. “Serverless” Platform

Three Levels of Platforms

1. Ecosystem

Combination of libraries, tools and **standalone** services.

Examples: Apache Spark/Flink + related tooling.

2. Managed Platform

A single **shared** managed runtime powering many use-cases.

Examples: Google Dataproc, Amazon EMR.

3. “Serverless” Platform

Three Levels of Platforms

1. Ecosystem

Combination of libraries, tools and **standalone** services.

Examples: Apache Spark/Flink + related tooling.

2. Managed Platform

A single **shared** managed runtime powering many use-cases.

Examples: Google Dataproc, Amazon EMR.

3. “Serverless” Platform

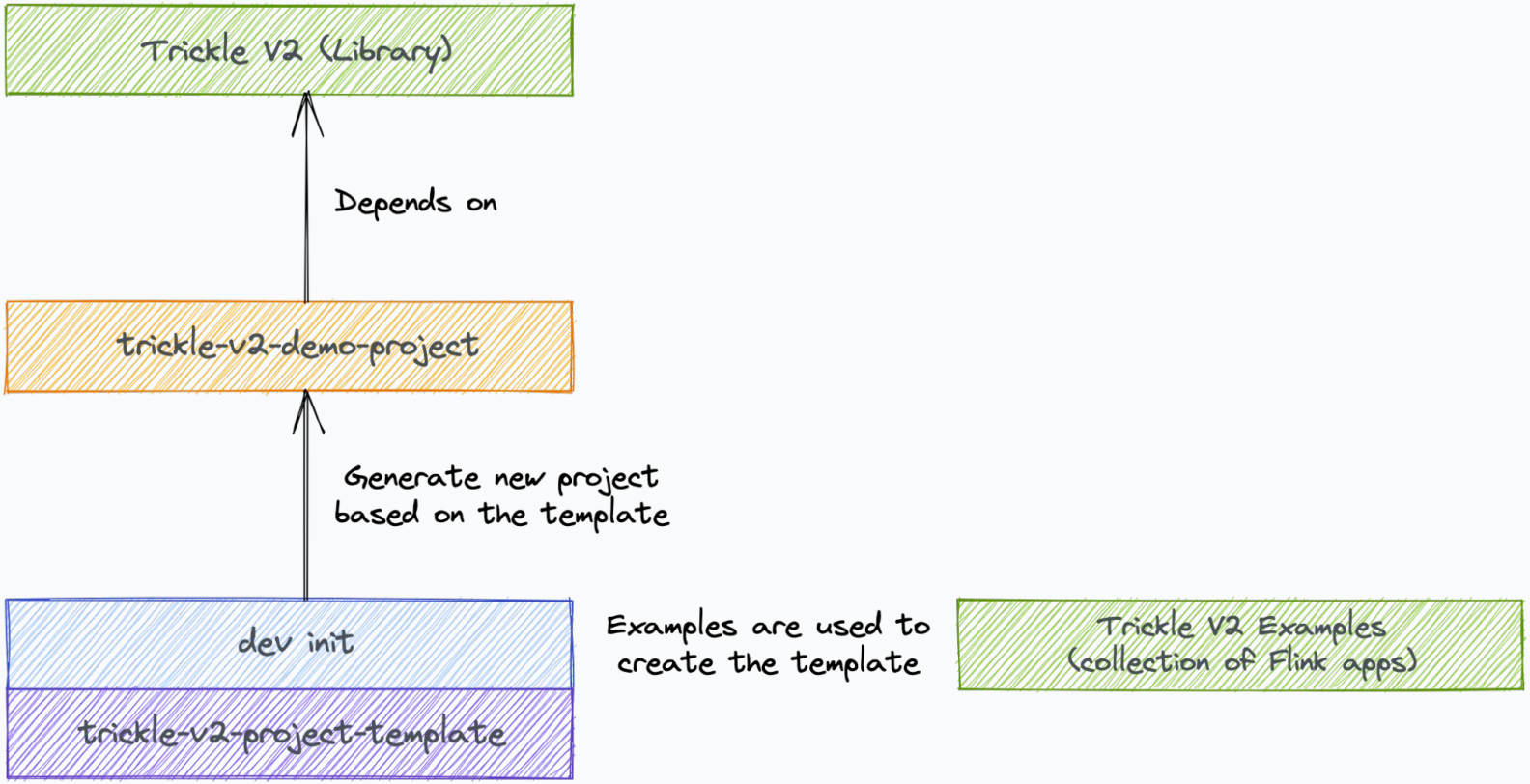
A single **shared** “serverless” runtime powering many use-cases.

Examples: Google BigQuery, Amazon Redshift Serverless.

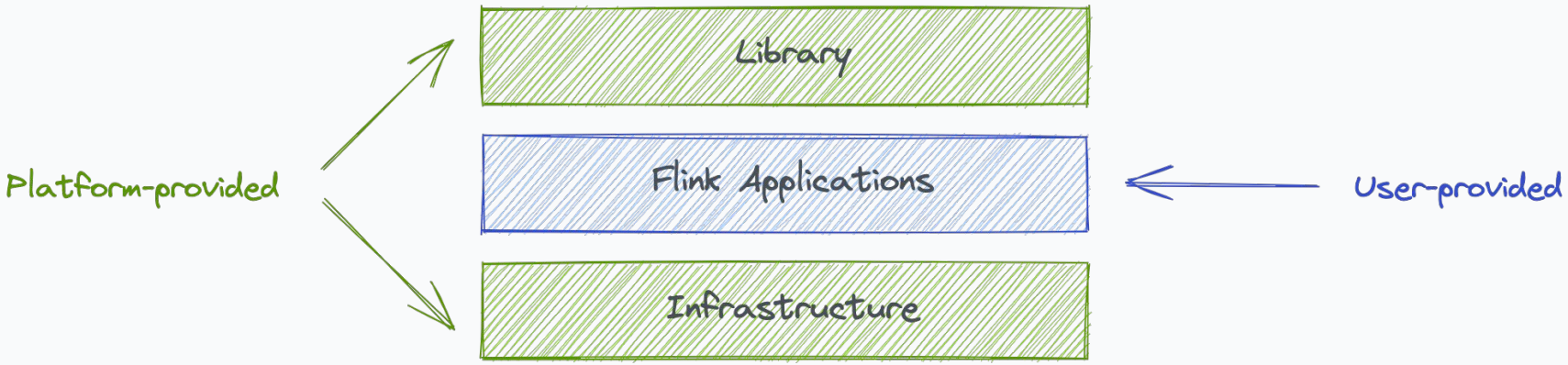
Our strategy: start with an ***ecosystem*** *of tools, evolve to a* *“****serverless****” platform*

Roadmap

- First milestone, foundations:
  - **Library**: common Flink components, helpers and connectors:
    - Kafka (multiple flavours), GCS (many formats), Bigtable.
  - **Observability**: DataDog metrics reporter, structured logging for Splunk.
  - **Examples**: real applications demonstrating common use-cases.
  - **Project generator**: have a working repo in 30 seconds.
  - **Documentation** & customer support.
- Second milestone: launch, learn and iterate.



Ecosystem: Trickle



Ecosystem: Trickle

```
implicit val env = Trickle.createEnv()
```

Typical Flink application

```
implicit val env = Trickle.createEnv()

val checkoutTrackSource: DataStream[CheckoutTrack] =
CheckoutTrackMonorailSource()

val lineItemsSource: DataStream[LineItemRecord] = CDCSource[LineItemRecord](
  topic = "core-line_items-v2", // ...
)
```

Typical Flink application

```
implicit val env = Trickle.createEnv()

val checkoutTrackSource: DataStream[CheckoutTrack] =
CheckoutTrackMonorailSource()

val lineItemsSource: DataStream[LineItemRecord] = CDCSource[LineItemRecord](
  topic = "core-line_items-v2", // ...
)

val sink: SinkFunction[Result] = pipelineConfig.sinkType match {
  case Print => new PrintSinkFunction()
  case Bigtable => BigtableSink[Result](
    table = "vendor_popularity", // ...
  )
}
```

Typical Flink application

```
implicit val env = Trickle.createEnv()

val checkoutTrackSource: DataStream[CheckoutTrack] = CheckoutTrackMonorailSource()

val lineItemsSource: DataStream[LineItemRecord] = CDCSource[LineItemRecord](
  topic = "core-line_items-v2", // ...
)

val sink: SinkFunction[Result] = pipelineConfig.sinkType match {
  case Print => new PrintSinkFunction()
  case Bigtable => BigtableSink[Result](
    table = "vendor_popularity", // ...
  )
}

val checkouts = processCheckoutTrackSource(checkoutTrackSource)
val lineItems = processLineItemsSource(lineItemsSource)
val results = aggregateJoinResults(
  joinCheckoutsAndLineItems(checkouts, lineItems)
)

results.addSink(sink)

env.execute("Demo App")
```

Typical Flink application

```
implicit val env = Trickle.createEnv()

val checkoutTrackSource: DataStream[CheckoutTrack] = CheckoutTrackMonorailSource()

val lineItemsSource: DataStream[LineItemRecord] = CDCSource[LineItemRecord](
  topic = "core-line_items-v2", // ...
)

val sink: SinkFunction[Result] = pipelineConfig.sinkType match {
  case Print => new PrintSinkFunction()
  case Bigtable => BigtableSink[Result](
    table = "vendor_popularity", // ...
  )
}

val checkouts = processCheckoutTrackSource(checkoutTrackSource)
val lineItems = processLineItemsSource(lineItemsSource)
val results = aggregateJoinResults(
  joinCheckoutsAndLineItems(checkouts, lineItems)
)

results.addSink(sink)

env.execute("Demo App")
```

Typical Flink application

CDC Kafka

Provided by `com.shopify.trickle.sources.cdc.CDCSource`. Internally, it creates a Kafka Consumer that uses Confluent Schema Registry in order to fetch CDC Avro message schemas.

You can find more details about CDC in the Production Platform docs. We also have a UI for the CDC Schema Registry.

Here's an example of consuming `core-line_items-v2` messages:

Copy

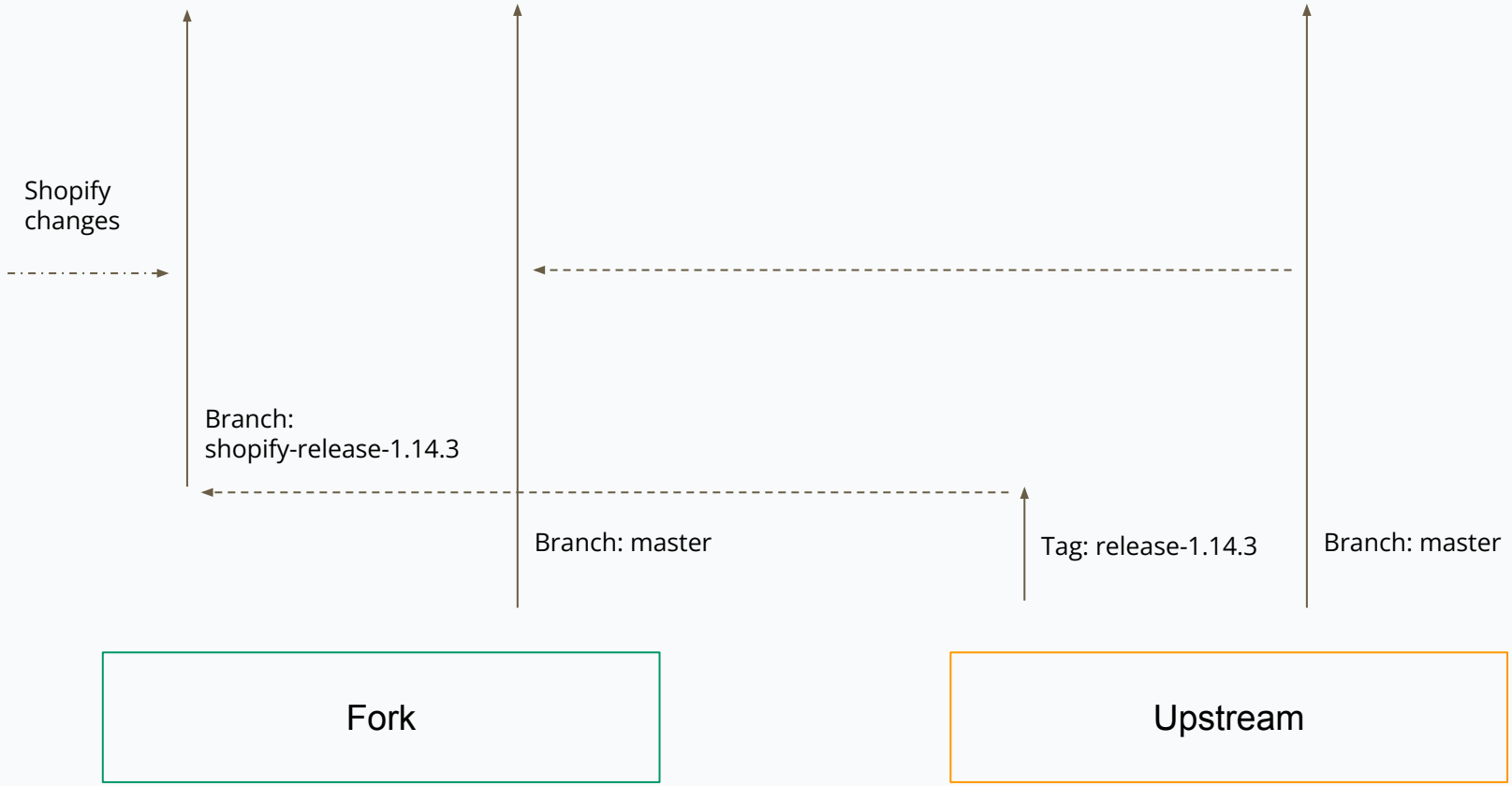
```
val lineItemsSource: DataStream[LineItemRecord] =
  CDCSource[LineItemRecord](
    name = "line-items",
    topic = "core-line_items-v2",
    Configuration.clusterConfig(CDCKafka),
    kafkaSSLConfig,
    offsetReset = Some(pipelineConfig.sourceOffsetReset)
  )
```

Trickle V2 documentation

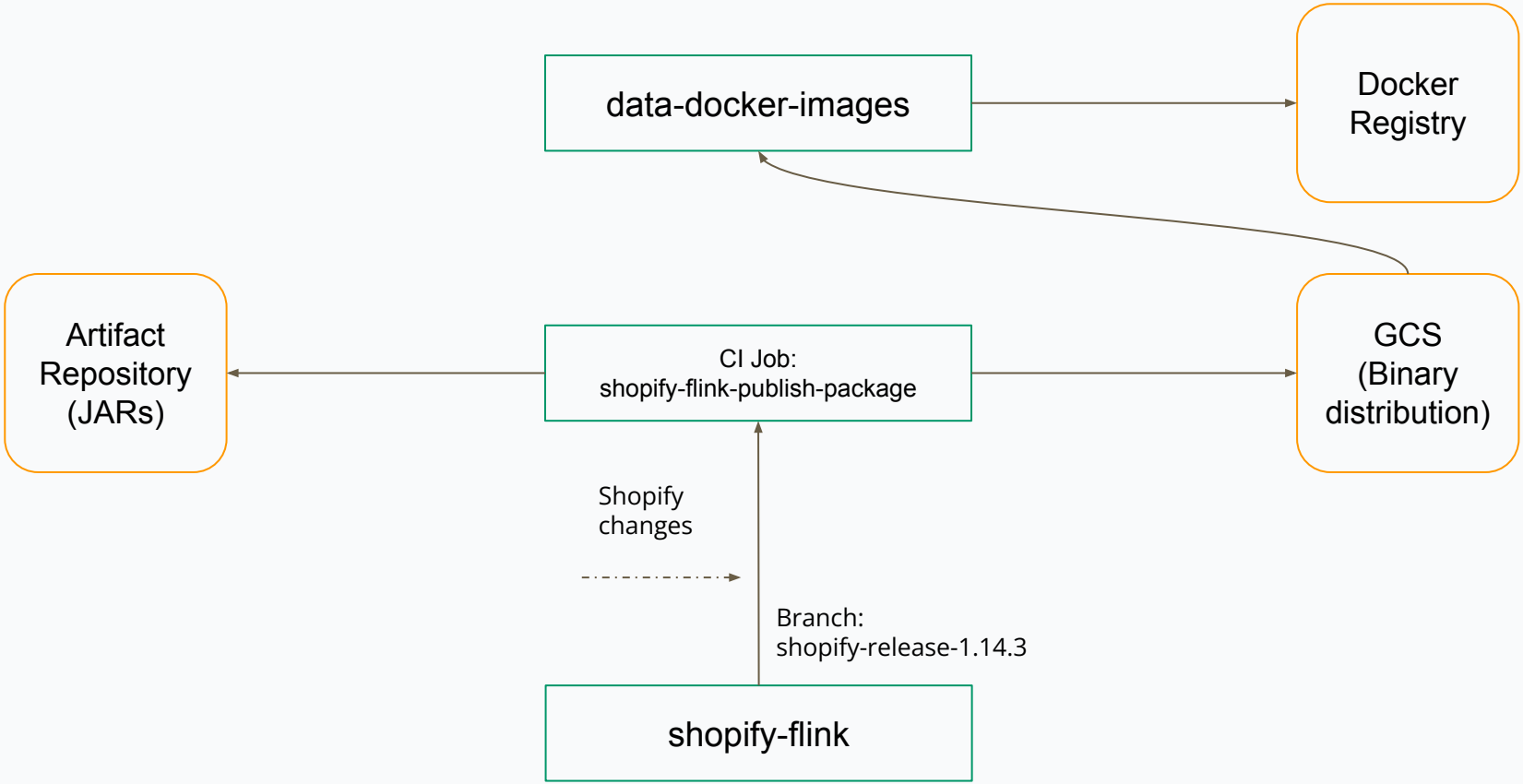
Lessons Learned

Apache Flink Fork

- We had to fork Flink in order to incorporate early features and add bugfixes:
  - E.g. running Flink in GCP might be tricky; Parquet reader < 1.14 has tons of issues.
  - Some things just don’t work properly (there are no good DogStatsD metrics reporters out there).
- Maintaining the fork doesn’t need to be hard!



Our Flink fork branching strategy



Our Flink fork build process

Data Reconciliation

- Consider investing in data reconciliation tooling when migrating workloads.
  - E.g. we have a data integrity service that continuously performs data integrity checks and alerts if necessary.
- Could be as simple as running old and new workloads in parallel and comparing results in some kind of notebook. You may need to make certain design decisions to support it.
- This can *actually* uncover bugs!

Scaling Adoption

- Multiple teams involved: **Streaming Capabilities**, **Customer Success** (DPE).
- The first team manages the core components, the second team helps customers:
  - Triage questions & requests, only escalate what's necessary.
  - Help with onboarding.
  - Act as consultants, be involved in technical designs and discussions.
  - White-glove first key customers.

Building Community

- Engage first adopters to build community!
  - Internal Q&A website.
  - #flink and other Slack channels.
  - Regular Flink User Group meetings.

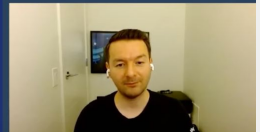
*In **less than 6 months** we had **3** use-cases in production and **10+** prototypes*

Storing State Forever: Why It Can Be Good For Your Analytics

Yaroslav Tkachenko

shopify

FLINK FORWARD
Global Virtual Conference 2021



Scaling Shopify's BFCM Live Map: An Apache Flink Redesign

by Berkay Antmen • Data Science & Engineering
Dec 10, 2021 • 8 minute read

Next Steps

- Production Maturity
  - Better Kubernetes tooling & integrations.
  - Zero-downtime deployments.
  - Autoscaling.
- New Features
  - 1.14 upgrade, Hybrid sources.
  - Python support.
  - Iceberg integration.
  - and more!

Final goal: serverless runtime.

Summary

- Carefully choose the right approach to build a platform.
- Build the foundation and engage customers early.
- Having more control over the key technology (e.g. forking it) may be necessary.
- Create a community, don’t afraid to white-glove first key customers.
- Keep iterating, focus on the biggest gaps.

Questions?

Twitter: @sap1ens

Also, we’re hiring! shopify.com/careers

shopify