Why you shouldn't care about Apache Iceberg

Ryan Blue

Data Council Austin, January 2022

What is Iceberg?

Tabular

Iceberg is an open standard for tables with SQL behavior

Tabular

Iceberg's insight

Tabular

Iceberg should be invisible

- Avoid unpleasant surprises
  - Principle of least surprise
- Don’t steal attention
  - Reduce context switching

Simplest form: Reliable updates

- Stop manual cleanup
  - All changes are successful
  - OR nothing changed at all
- Enable targeted updates
  - Rewrite only what's needed

(You know, the boring stuff)

Tabular

Today’s use case: PoochFitness

Tabular

Congratulations!

- You're the newest employee at PoochFitness
  - PoochFitness sells the *premium fitness tracker* for man's best friend
- Events are already flowing
  - 12 months of data available
- Everyone is eager for insights!

```
CREATE TABLE pooch_logs (
    event_serial bigint,
    event_ts timestamptz,
    device_id string,
    steps int,
    possible_shake boolean)
```

Tabular

Problem #1: Bad data

Tabular

A PII bug!

- You find email addresses in the `device_id` column
  - Field order changed in PoochFitness Rev1.3
  - No one uses device_id
  - Event parsing is fixed

- Option #1: Babysitting
  - Rewrite a year of events

- Option #2: Drop & Add

```
ALTER TABLE pooch_logs
  DROP COLUMN device_id
ALTER TABLE pooch_logs
  ADD COLUMN device_id string
```

Tabular

Drop and Add

```
ALTER TABLE pooch_logs DROP COLUMN device_id;
ALTER TABLE pooch_logs ADD COLUMN device_id string;

SELECT count(🧟) FROM pooch_logs WHERE device_id LIKE '%@%';
=> 4198274192872
```

Spark does better!

```
UnsupportedOperationException: Unrecognized column change class
org.apache.spark.sql.connector.catalog.TableChange$DeleteColumn.
You may be running an out of date ^H^H^H version.
```

Tabular

Schema evolution

- Instantaneous – no rewrites
- Safe – no undead columns 🧟
- Saves days of headache

```
ALTER TABLE db.tab
RENAME COLUMN
  id TO customer_id
```

Tabular

Problem #2: Slow queries

Tabular

Queries are slow

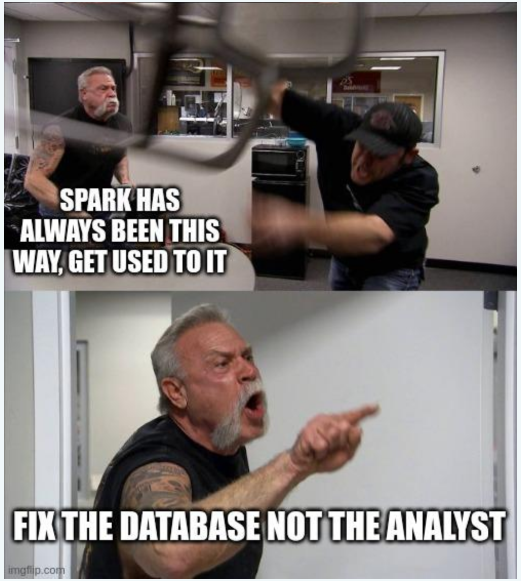
- Analysts need your help
  - The table used to be fine
  - Queries were slower over time
  - Everyone eventually gave up

- Cause #1: No partition filters
  - Analysts just need to be trained to filter data twice!

- Cause #2: No partitioning
  - Let's hope not...

Hidden partitioning

- No silent correctness bugs
- No conversion mistakes
- Query without being an expert or DBA



Tabular

Problem #3: No partitions

What if there was no partitioning?

- No partitioning
  - No one knew this was a thing
  - Everyone's too busy working on PoochFitness Rev1.5 to worry about this!
- Migrate to a new table?
  - Rewrite all the queries
  - Rewrite all the data

Tabular

Layout evolution

- Lazy – only rewrite if needed
- Partitioning mistakes are okay
- Changes with your data
- Saves a month of headache

```
ALTER TABLE pooch_logs
ADD PARTITION FIELD
  days(event_ts) as ts_date
```

Iceberg should be invisible

- Avoid unpleasant surprises
  - No zombie columns
  - Performance should not be mysterious
- Don’t steal attention
  - No rewriting to drop a column
  - Don’t make people filter twice
  - Fix problems without migration

Tabular

Sounds like I should care about Iceberg?

Tabular

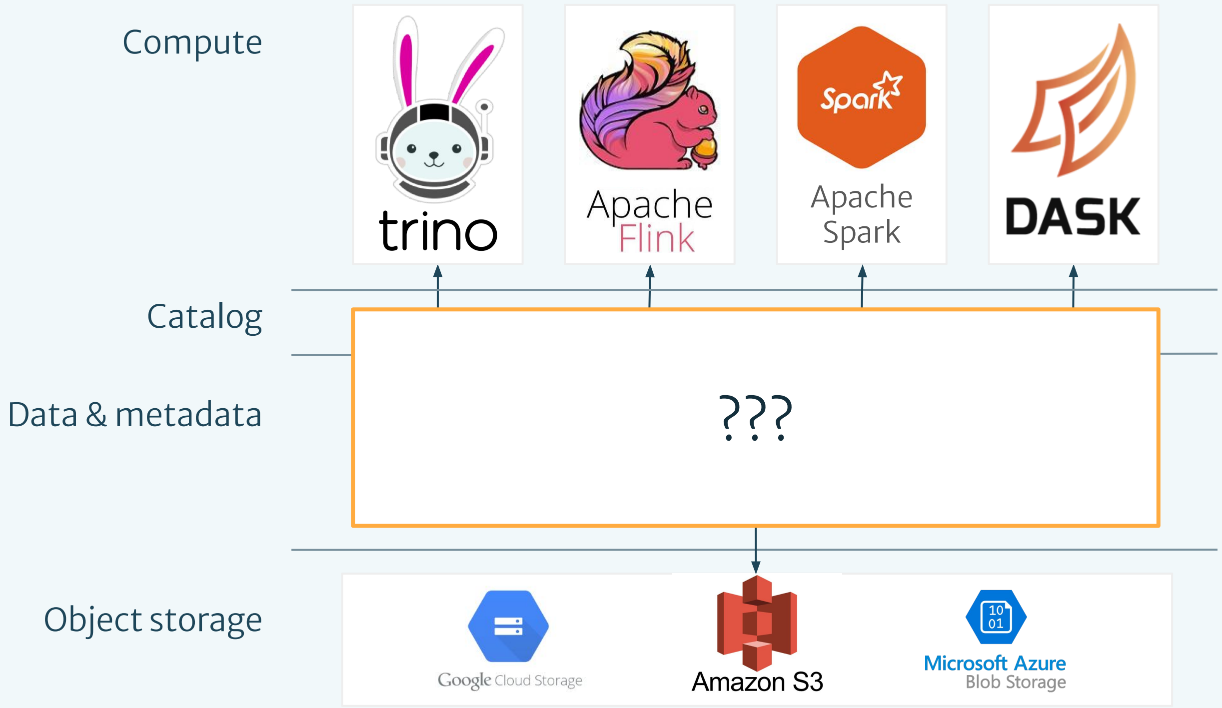
Care about capabilities, not formats

The data landscape is changing

- Central table storage
  - Independent from compute
  - Think about data gravity
- Portable compute
  - Multi-engine is the new normal
  - Portable logic and code
- Access control
  - Consistent authorization policy
  - Enforced across engines
- Stop losing structure
  - Most “unstructured” data didn’t start that way
  - Share, don’t copy

Tabular

This multi-engine platform is the next challenge



Tabular

Iceberg is the foundation

- Open standard for huge tables
- SQL abstraction and behavior
- Data warehouse fundamentals
- Data services
  - Don’t make humans babysit
- Declarative data engineering
  - Vastly different engines require better ways to work

Thank you!

Tabular

Iceberg is much more …

- Expressive SQL commands
  - MERGE INTO
  - UPDATE … WHERE
  - Lazy and eager rewrites (copy-on-write, merge-on-read)
- Declarative data engineering

```
ALTER TABLE ...
WRITE ORDERED BY event_ts
```

- Time travel
- Indexed data and metadata
- Branching and tagging for CI/CD

Tabular

Apache Flink Adoption at Shopify

Yaroslav Tkachenko

👋 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.

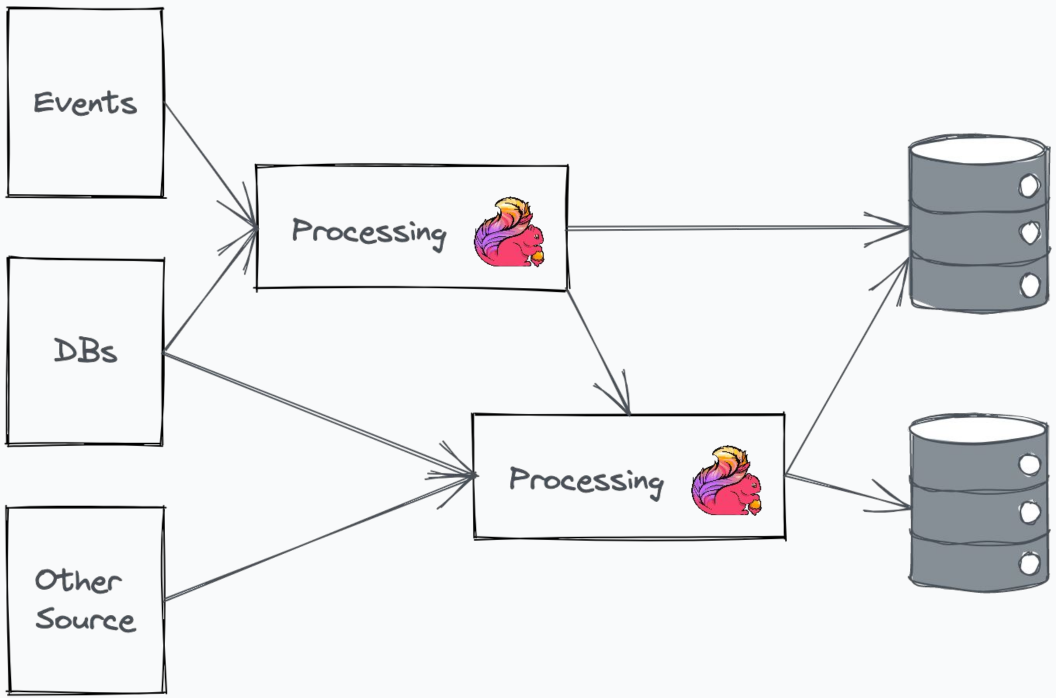
1.7 Million+
NUMBER OF MERCHANTS

~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

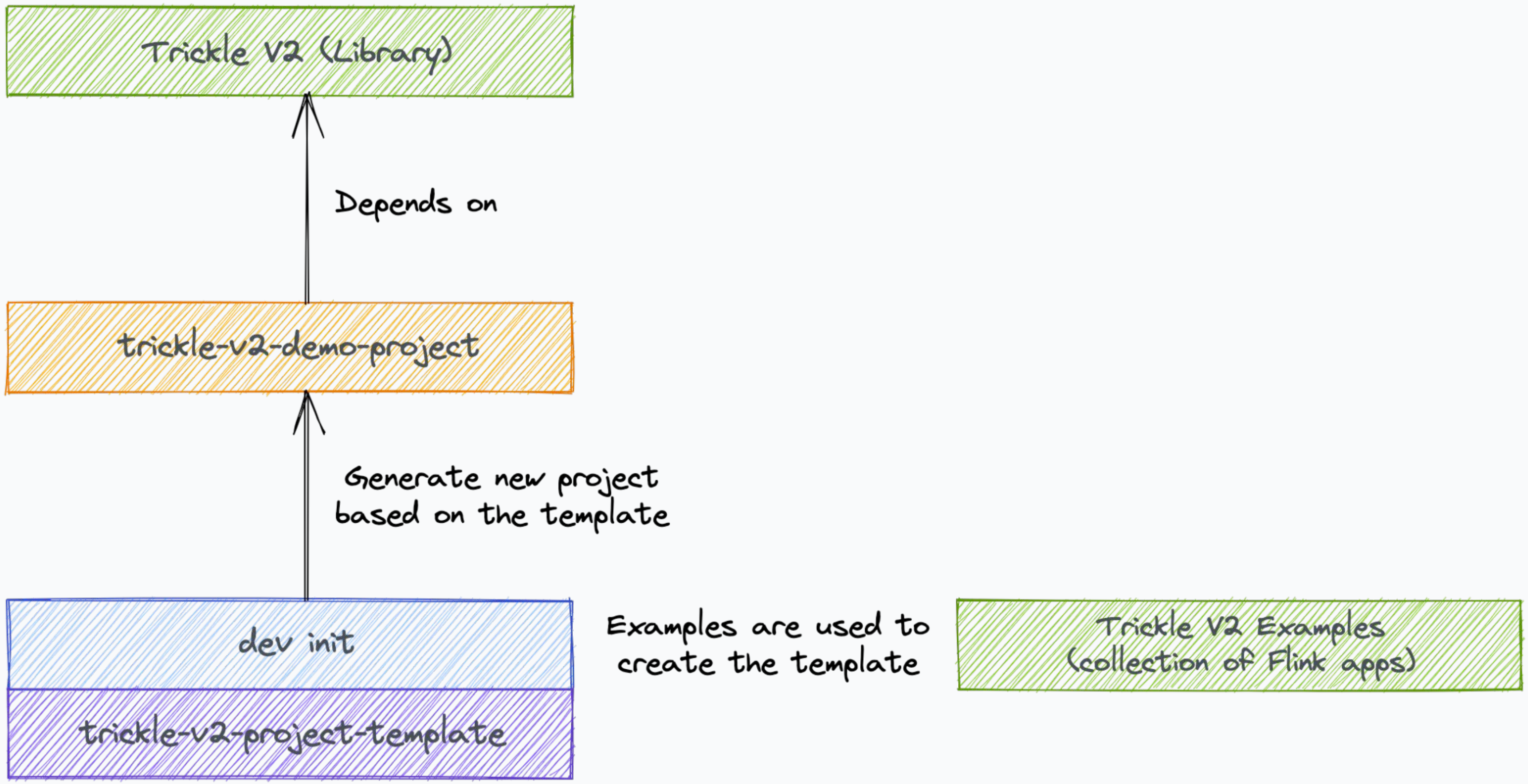
A single **shared** “serverless” runtime powering many use-cases.

Examples: Google BigQuery, Amazon Redshift Serverless.

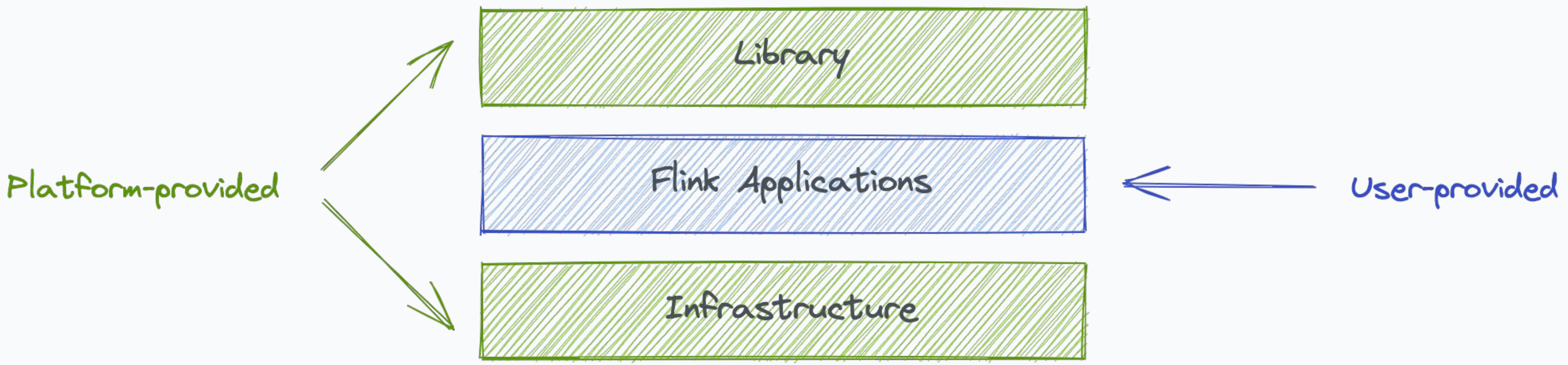
Our strategy: start with an ***ecosystem*** *of tools, evolve to a “****serverless****” platform*

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

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

val checkoutTrackSource: DataStream[CheckoutTrack] =
CheckoutTrackMonorailSource()

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

Typical Flink application

```scala
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

```scala
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

```scala
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

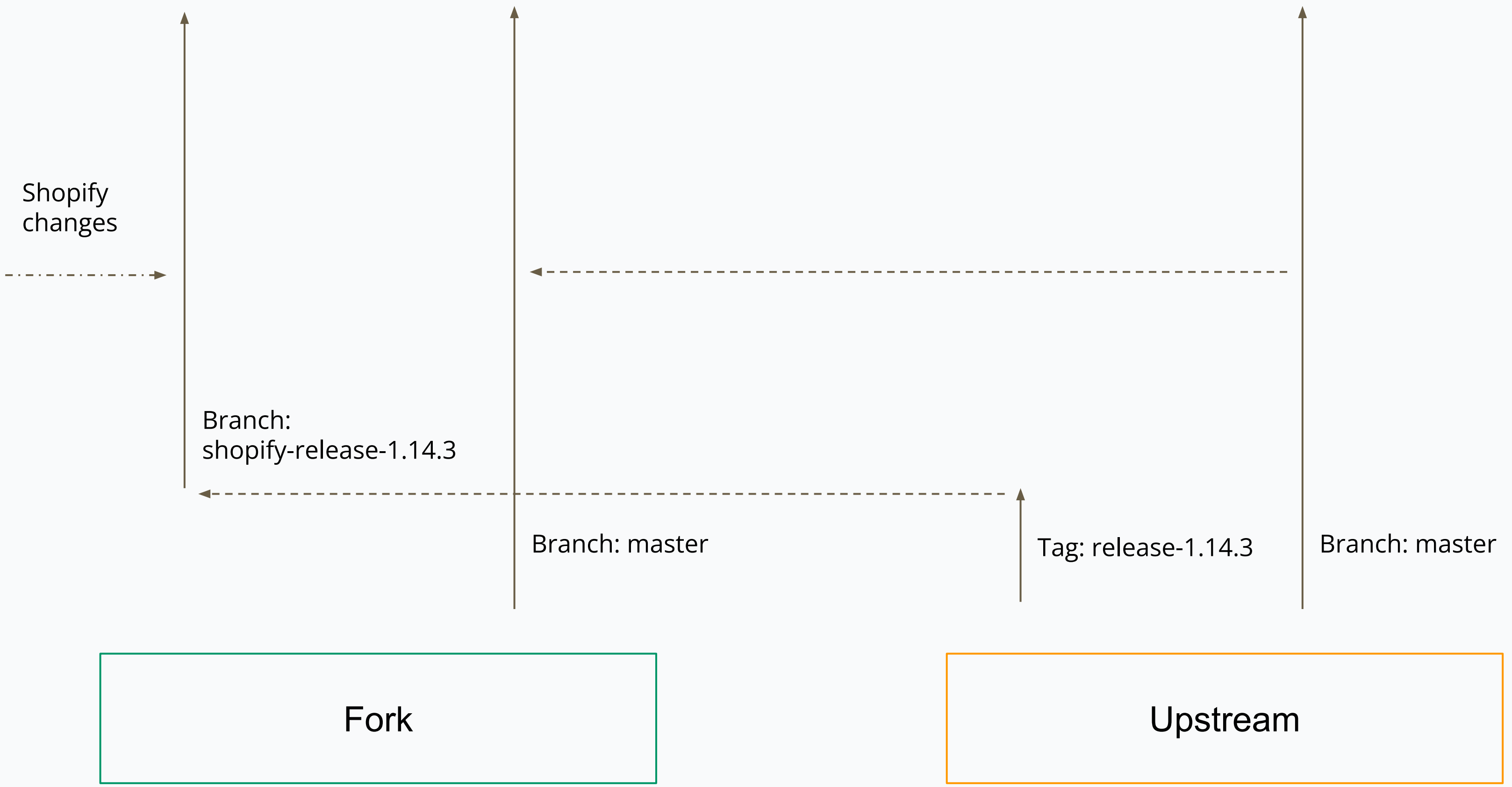
```
val lineItemsSource: DataStream[LineItemRecord] =
  CDCSource[LineItemRecord](
    name = "line-items",
    topic = "core-line_items-v2",
    Configuration.clusterConfig(CDCKafka),
    kafkaSSLConfig,
    offsetReset = Some(pipelineConfig.sourceOffsetReset)
  )
```

Trickle V2 documentation

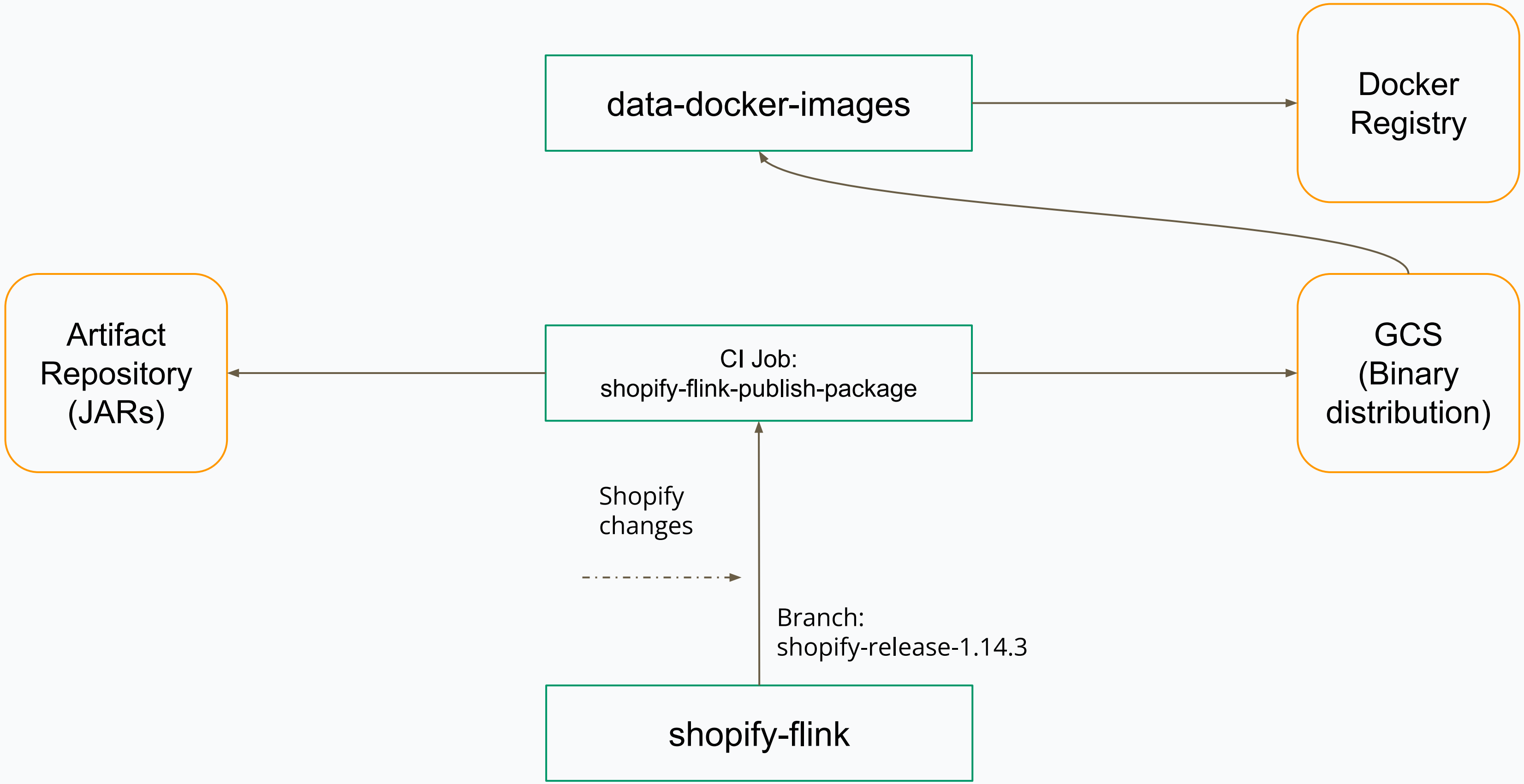
Lessons Learned

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!

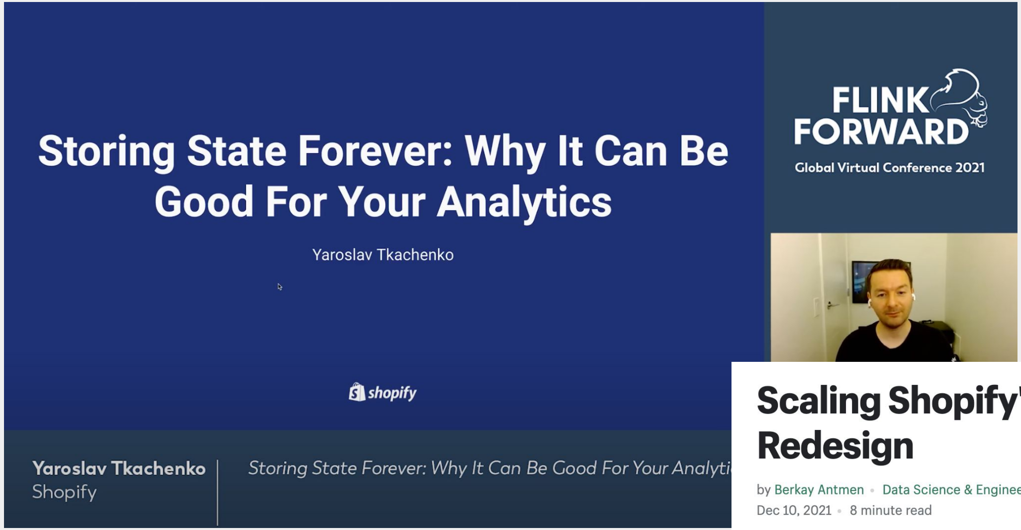
Scaling Adoption

- 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
Yaroslav Tkachenko
Shopify
Storing State Forever: Why It Can Be Good For Your Analyti


Scaling Shopify's BFCM Live Map: An Apache Flink Redesign
by Berkay Antmen • Data Science & Engineering
Dec 10, 2021 • 8 minute read
BFCM 2021

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

How 200+ Leaders Made Business Data Work Harder

Jesika Haria, LogicLoop

What you'll get out of this talk

How to get more operational use out of your data

You = Dani, the Data Engineer

5-10 years in industry

Build reports and pipelines for business users

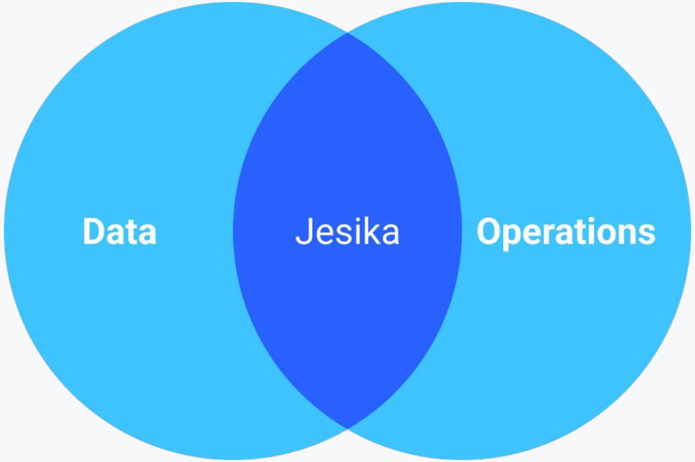
Want to do more high-leverage work

We'll cover

- 200+ leaders' operations data needs
- A system for business alerting & automation
- How to get the most out of it
- References & success stories

Make data work harder than people!

Know Thy Speaker



Jesika Haria

CEO, LogicLoop

@jesikaharia

LogicLoop

Founder and CEO
Operations automation for high-growth companies to move faster without engineers

INSTABASE

Founding Team #5
Built 1st remote eng team, customer success for top 10 banks, founded Product org

TRIFACTA

Sr Software Engineer
Built 1st cloud product used by 100,000+ analysts as Google Cloud Dataprep

facebook

Graph Search Engineer
Ranking algorithms for Groups

Massachusetts Institute of Technology

EECS | Advanced Researcher
1 of 3 from all over India selected

200+ leaders share their operations data needs

Operational data is an under-utilized lever in business growth

Analytical

Operational

200+ leaders use operational data across verticals

Compliance & Fraud

Brex chime coinbase

"...flag risky transactions"

Trust & Safety

airbnb Uber Quora

"...remove spam and abuse"

Business Operations

ula loop LendFlow

"...remediate fulfillment issues"

Logistics

DOORDASH Ready

"...adjust demand and supply live"

Healthcare

oscar Cerebral

"...allocate COVID vaccine tiers to patients"

Customer & Growth

Dover affinity

"...automate account summary emails"

"We ingest operational data…"

Expectation

Data Warehouses

Reality

Databases

Data Warehouses

Google Sheets

SaaS Apps

Product Analytics

Data APIs

“...to trigger actions”

Google Sheets

Slack

Emails

Customer Comms

Tickets

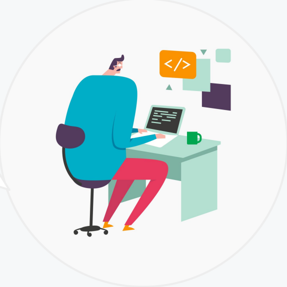
Internal Services

Databases

Fast-growing companies are bottlenecked

Can you update this fraud policy urgently?

Errr.. next sprint?



Ollie, the Ops

- Frustrated by slowness
- Cannot experiment
- No visibility or governance

Dani, the Data Eng

- Overwhelmed fighting fires
- High-leverage work suffers

Growth = new business apps & workflows
Creates demand for new data pipelines

Only ~20% requests get fulfilled by data engineers

Self-serve is the key

$$\text{Work} = \frac{\text{Data} \times \text{Growth Pressure}}{\text{Engineers} \times \text{Self-Serve}}$$

TAKE AWAY

🔑 Operations data maturity checklist

- ☐ Clean operational data exists
- ☐ Business knows where to find it
- ☐ Business can self-serve insights

“read”

- ☐ Business can proactively identify issues
- ☐ Business can debug and rectify exceptions

“write”

- ☐ Business can automate handling exceptions
- ☐ Business can improve operational processes over time

“leverage”

TAKE AWAY

Introducing a system for business alerting and automation

Motivating use cases

Trigger a Slack alert for large transactions

```
SELECT * from transactions
where amount > $100,000
```

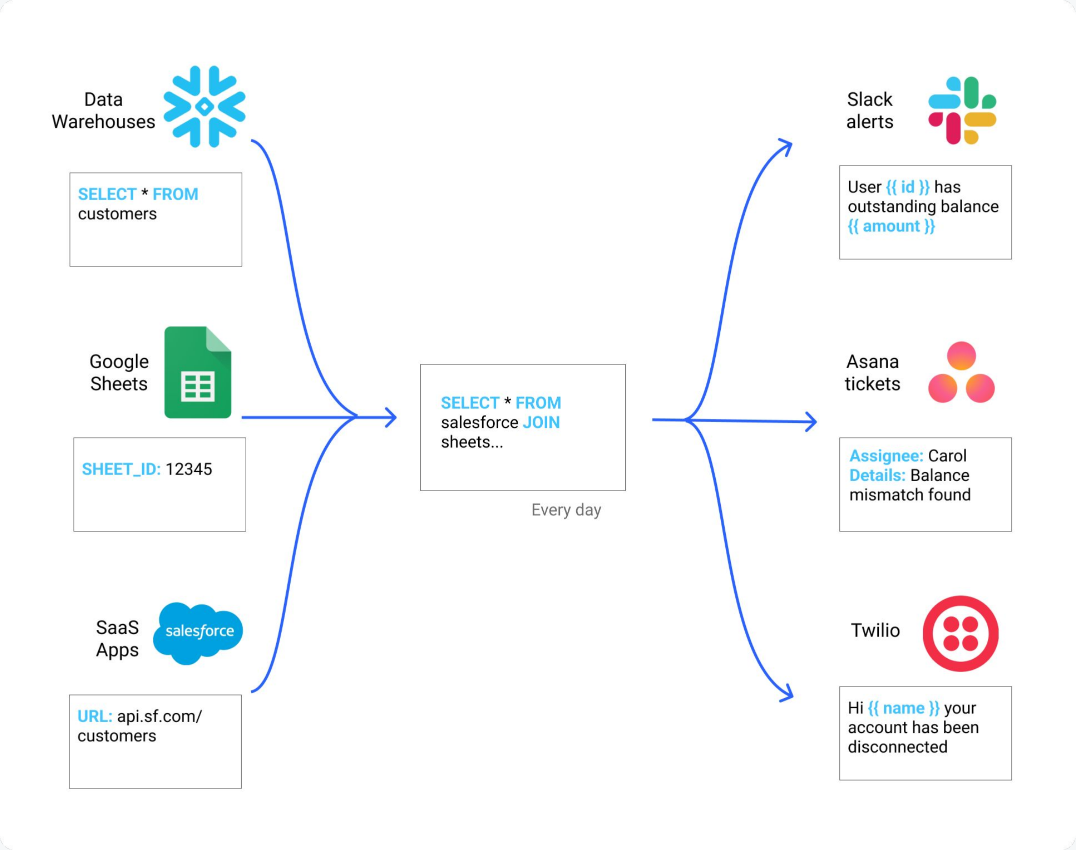
Send users a text reminder to pay

```
SELECT * from users where
payment_due < now() + ‘24
hours’
```

Automate weekly account summary emails

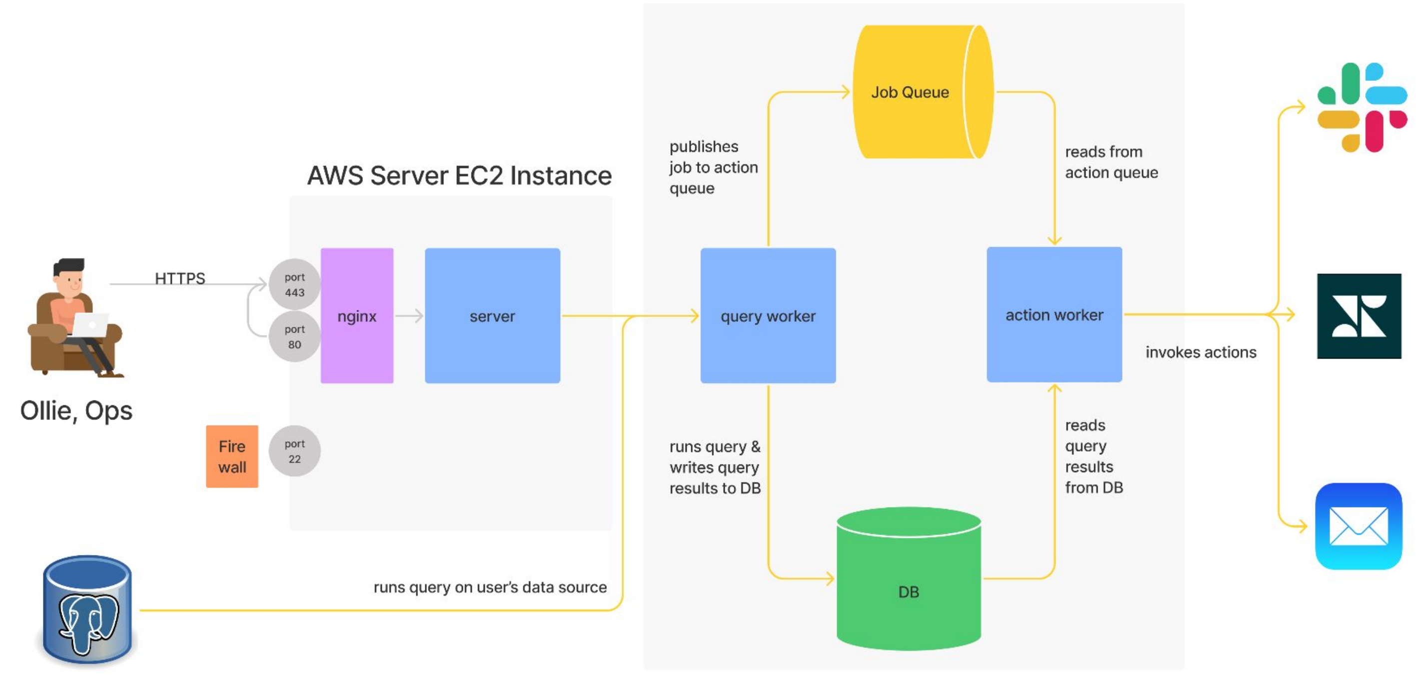
```
SELECT stats FROM accounts
```

An ideal business user experience

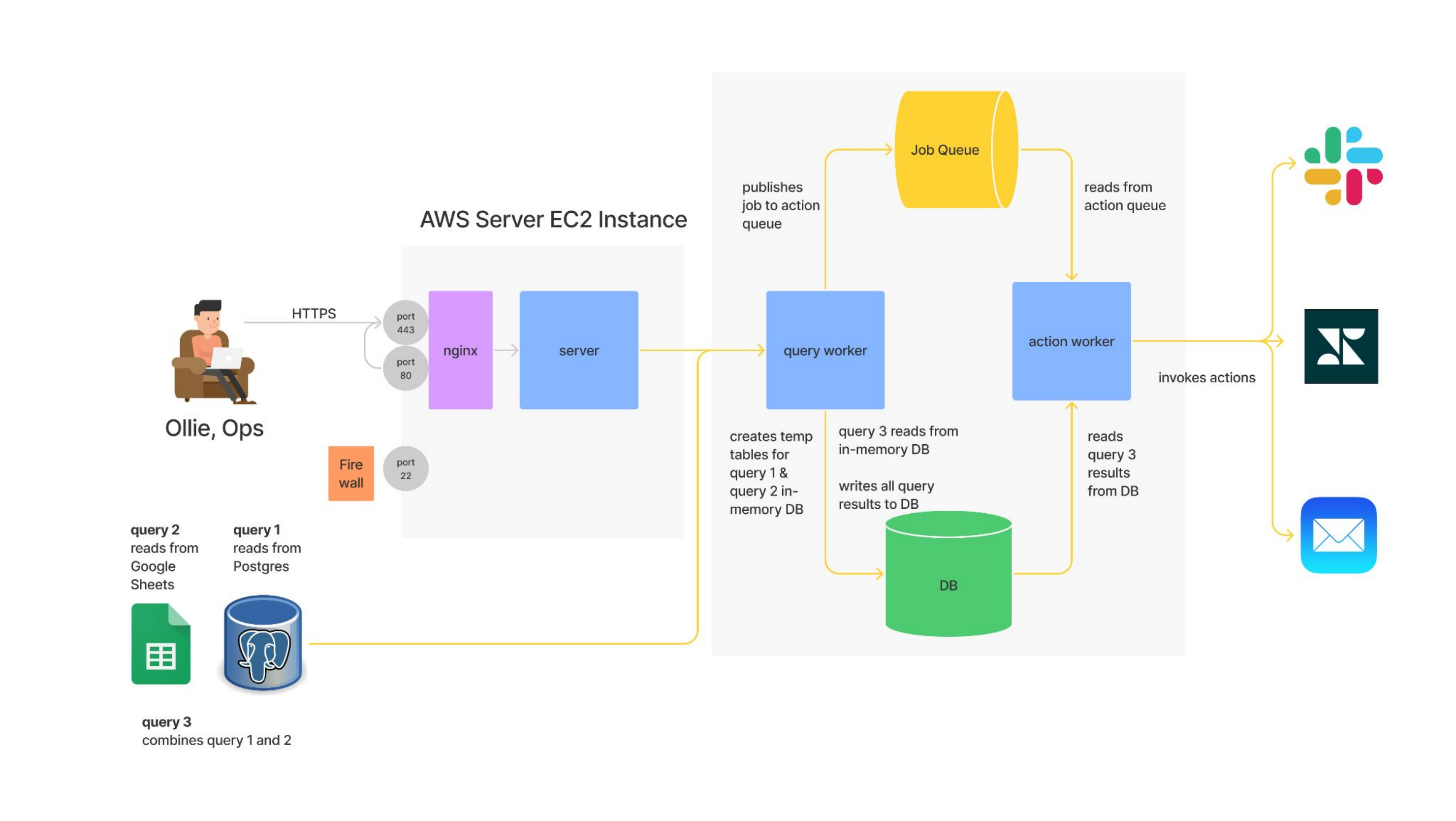


TAKE AWAY

What a solution could look like



What a solution could look like



🔑 Common pattern amongst best internal tools

A system for *business users* to detect and act quickly

Business users need to combine real-time with warehouse data

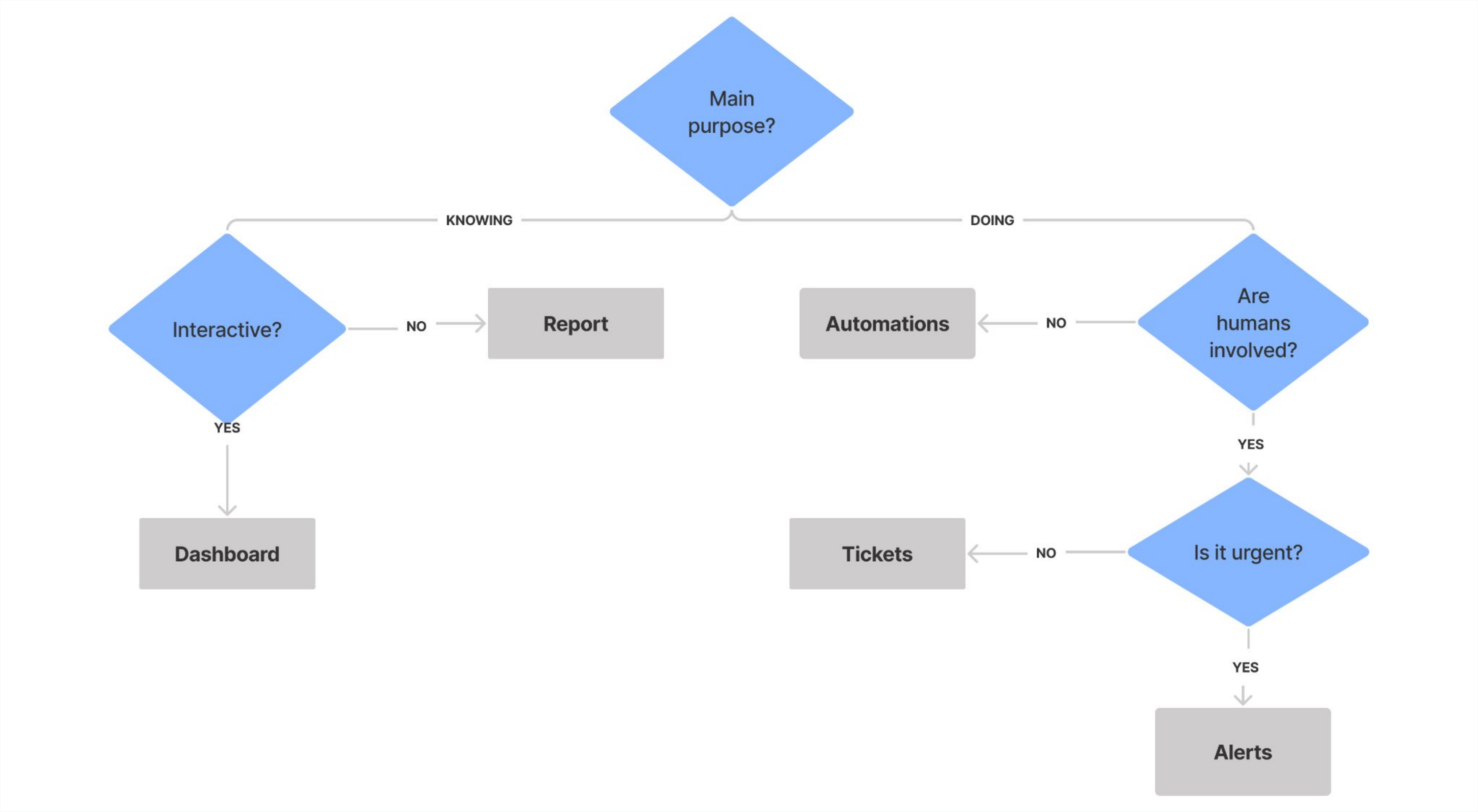
Business users need to set and change thresholds

Business users need to route data to Slack alerts, emails, customer communications etc.

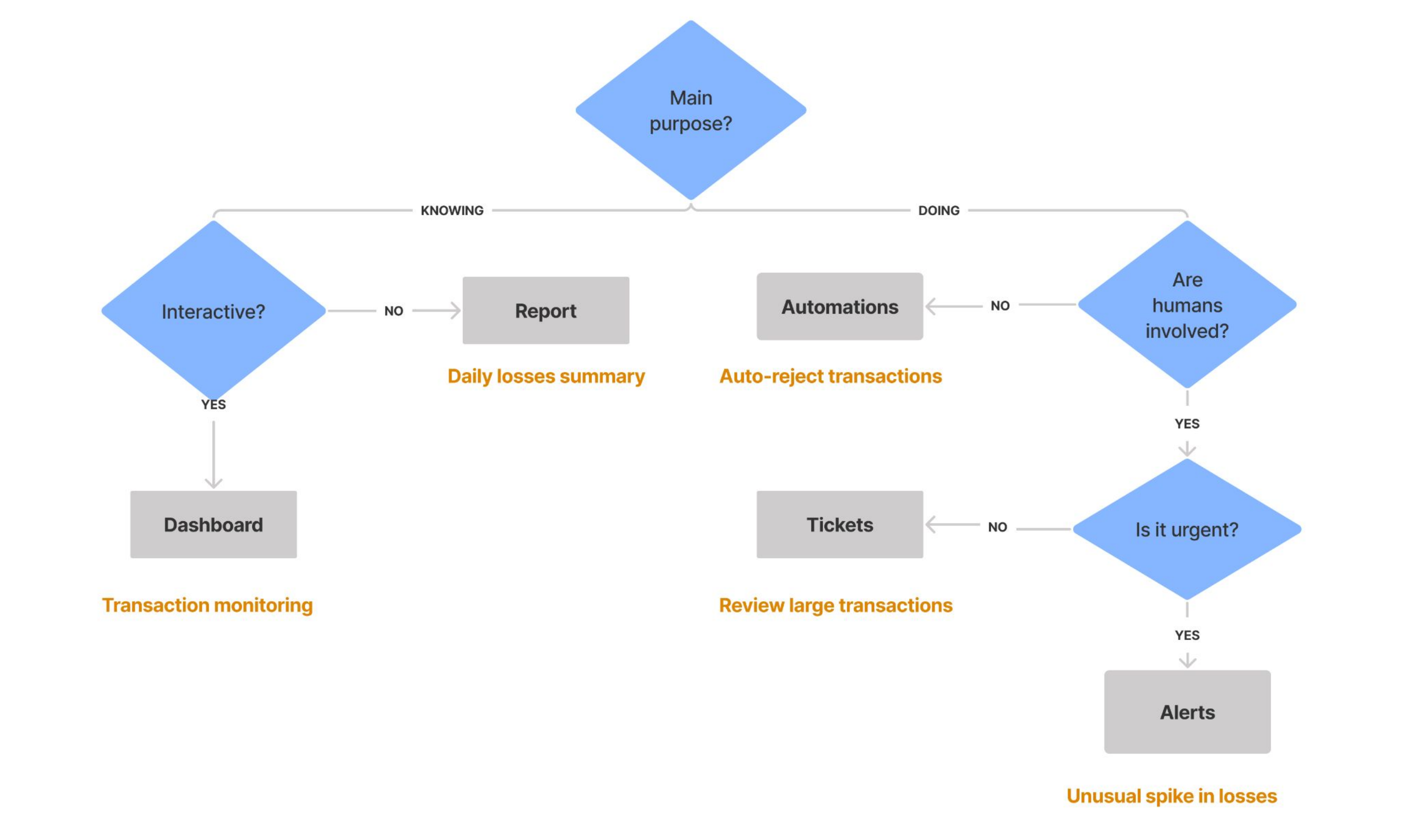
TAKE AWAY

How to get the most out of a business alerting and automation system

#1 Route operations data correctly



Example: fraud data operations



#2 Deep dive: alerting best practices

Alerts should be real, urgent and actionable

Creation

- ❏ Don't re-alert for the same issue
- ❏ Calibrate as early and often as possible
- ❏ Aim for <5 / week

Tip: Over-monitoring is harder to solve than under-monitoring

Content

- ❏ Which system created the alert
- ❏ Description
- ❏ Severity of deviance
- ❏ Link to resolve / debug
- ❏ Owner
- ❏ SLA for resolution

Tip: Use emojis to help skim!

Management

- ❏ Audit and action logs
- ❏ Debugging dashboards

Tip: Snooze or set reminder schedules

TAKE AWAY

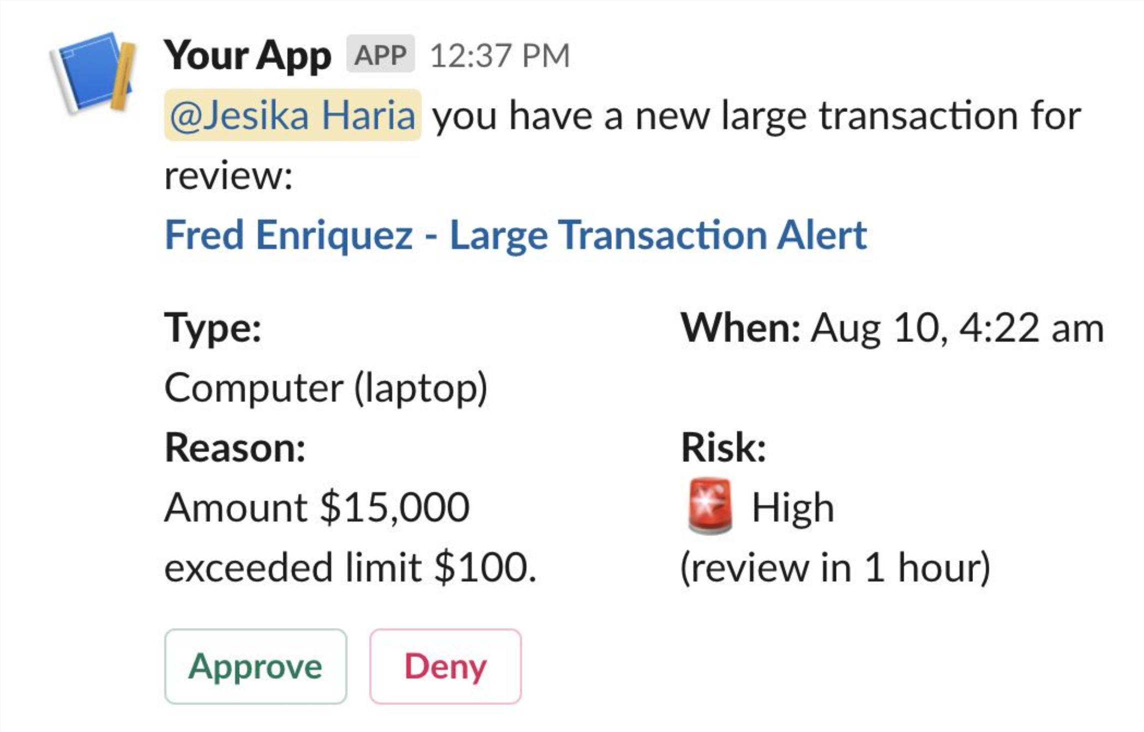
Example: a fraud alert

System of origin

Clear owner

Deviance

Call to action



Skimmable

Link to details

Relevant info

Urgency level

#3 Iterate, iterate, iterate

Improve signal

- ❏ Weed out alerts >x% false positive rate
- ❏ Consolidate alerts that have >x% overlap
- ❏ Distinguish between data and system failure

Monitoring as code

- ❏ Version control changes
- ❏ Backtest
- ❏ Permissioning
- ❏ Approval process

Management

- ❏ Ensure commensurate staffing
- ❏ Groom backlogs every month
- ❏ Track time to resolve and automate biggest time sinks
- ❏ Consolidate decisioning systems

Example: Improving fraud alerting system

Step 1

Add a Slack notification & adjust thresholds

Step 2

Create review tickets

Step 3

Auto reject

🔑 How to get the most out of a business alerting and automation system

Route data correctly
Decision Tree + Example

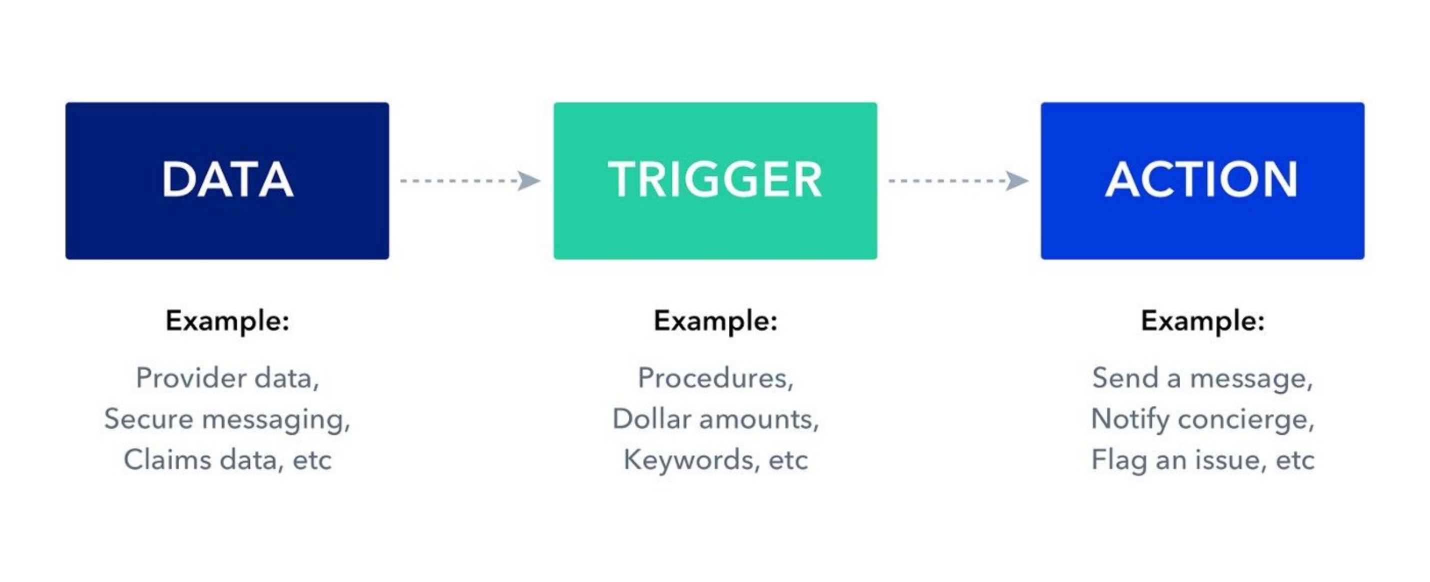
Deep dive: alerting best practices
Checklist + Example

Iterate, iterate, iterate
Checklist + Ladder

References & Success Stories

Case study: Oscar Health

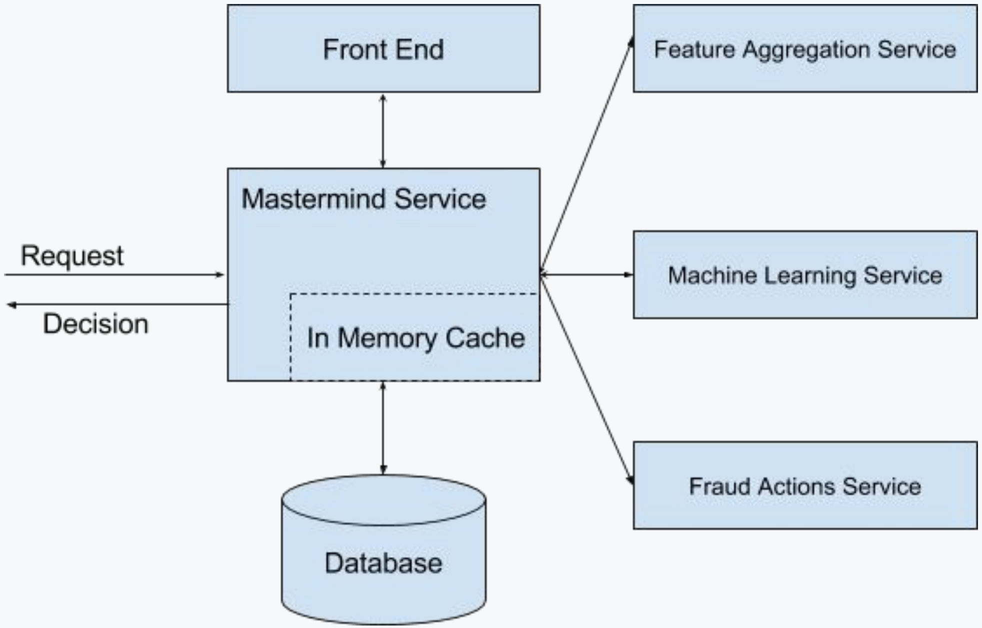
Built Automat, a self-service configuration platform



Reference https://medium.com/oscar-tech/an-equal-partnership-between-tech-and-business-69b8c16a499b, 03/20/2022

Case study: Uber

Built Mastermind, a real-time fraud rules engine



Reference https://eng.uber.com/mastermind/ , 03/20/2022

And results you won't find blogs about

$95B fintech

1M saved

per quarter by risk analyst writing fraud rules

Series B fintech

40% revenue

generated by retargeting high-value customers

Recruiting Co

32x ROI

on automating account summary emails

Series C fintech

2 eng months

saved with deduping

Summary

🔑 What we talked about

- ❏ **200+ leaders' operations data needs**
  - ❏ Self-serve maturity checklist
- ❏ **A system for business alerting & automation**
  - ❏ Reference architecture for Data → Triggers → Action
- ❏ **How to get the most out of it**
  - ❏ Examples and best practices on how to route data, alert and iterate
- ❏ **References & success stories**
  - ❏ Architectures and case studies

LogicLoop

I think about this a lot because we're building it – let's talk!

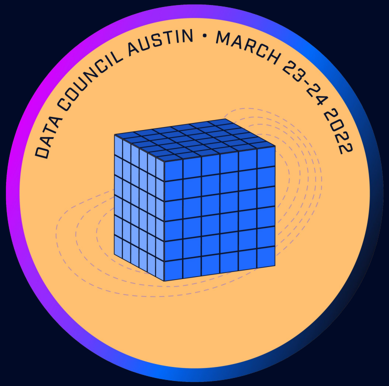
@jesikaharia
jesika@logicloop.com

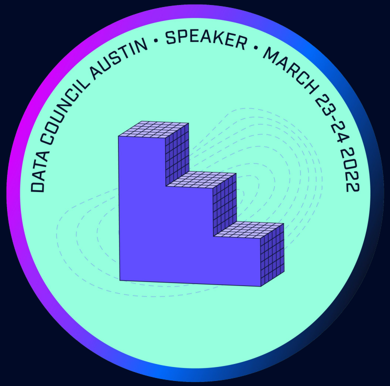
TAKE AWAY

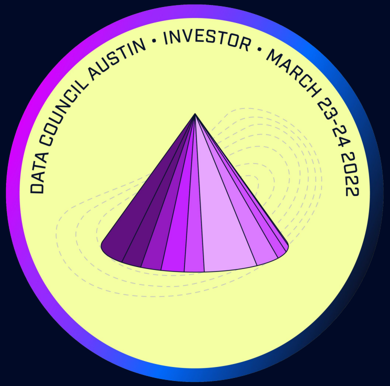
Data Council Austin

MARCH 23 - 24, 2022

NFT Drop


DATA COUNCIL AUSTIN • MARCH 23-24 2022


DATA COUNCIL AUSTIN • SPEAKER • MARCH 23-24 2022


DATA COUNCIL AUSTIN • INVESTOR • MARCH 23-24 2022

Closing Keynote

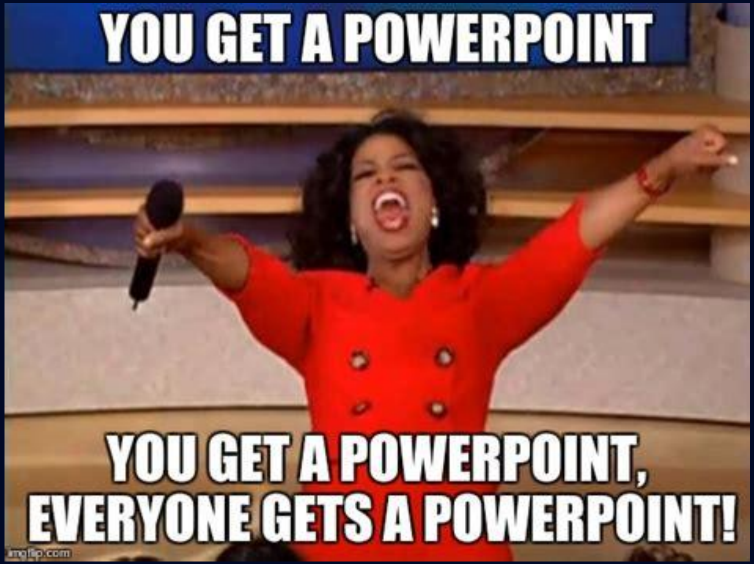
DevOps for ML and Other Half-Truths

Diego Oppenheimer, EVP, DataRobot

Data Council Austin

MARCH 23 - 24, 2022

When Speaker Slides?



Thank you