The Future of Data Engineering

Chris Riccomini / WePay / @criccomini / QCon SF / 2019-11-12
This talk

• Context
• Stages
• Architecture
Context
Me

- WePay, LinkedIn, PayPal
- Data infrastructure, data engineering, service infrastructure, data science
- Kafka, Airflow, BigQuery, Samza, Hadoop, Azkaban, Teradata
Me

- **WePay**, LinkedIn, PayPal
- **Data infrastructure, data engineering**, service infrastructure, data science
- **Airflow, BigQuery, Kafka**, Samza, Hadoop, Azkaban, Teradata
Me

• WePay, LinkedIn, PayPal
• Data infrastructure, data engineering, service infrastructure, data science
• Airflow, BigQuery, Kafka, Samza, Hadoop, Azkaban, Teradata
Me

• WePay, LinkedIn, **PayPal**
• Data infrastructure, data engineering, service infrastructure, **data science**
• Airflow, BigQuery, Kafka, Samza, Hadoop, Azkaban, **Teradata**
Data engineering?
A data engineer’s job is to help an organization move and process data.
“...data engineers build tools, infrastructure, frameworks, and services.”

-- Maxime Beauchemin, The Rise of the Data Engineer
Why?
How we built analytics at Ada with Airflow and Redshift

Patryk Skorko
May 28 - 11 min read

At the beginning of 2018, when I was packing my bags to move from Moscow to Toronto, my dear colleagues, whom I didn’t know yet, were receiving an increasing amount of requests for insights into how Ada’s virtual assistant was doing. How many conversations did the virtual assistant start and received a response from the user? How many conversations happened on a mobile device? What are the most popular responses virtual assistant gives to French-speaking customers?

Both the Development and Customer Success teams rushed into answering those questions. Ada was barely a dozen people by that time, and all of them did their best to give clients their insights. But we had a huge feature...
Building WePay's data warehouse using BigQuery and Airflow

By Chris Riccomini on July 2014

Over the coming weeks, we'll be writing a series of posts describing how we've built and run WePay's data warehouse. This series will cover our usage of Google Cloud Platform, BigQuery, and Apache Airflow (incubating), as well as how we handle security, data quality checks, and our plans for the future.

The beginning

We run MySQL as our primary OLTP database at WePay. Most of the data resides in a single monolithic database cluster that runs inside Google compute engine. When we started, we, developers, analysts, product managers, and others were all running their analytic queries on a replica of this MySQL cluster. This replica was used to do things like train machine learning models, generate business and financial reports, build business intelligence dashboards, debug production issues, and so on.

This approach had the advantages of being convenient and having up-to-date data (since it was just a replica of the database). The disadvantage of this setup is that MySQL is not very good at handling large analytic-style queries. This is not surprising. It's meant to be used as a low latency OLTP database, not a data warehouse. As we (and our data) grew, we bumped up against some problems:

- Multi-tenancy issues, where one user would severely degrade the entire cluster.
- Performance problems, timeouts, and queries that never finish.
- Inability to easily run custom logic on the data (UDFs).

It was obvious that we needed to move to a real data warehouse. This post describes that journey, the lessons learned, and the technology that we used.

Picking the stack

WePay made the transition to Google Cloud Platform last year. One of the services that Google provides is BigQuery, a distributed data warehouse built on top of a bunch of great Google technology including Spanner, Borg, Colossus, and Jupiter. We did an evaluation of BigQuery, and decided to use it as our data warehousing solution.

After picking BigQuery, we needed to get data into it. A naive approach would be to run a Python script via CRON that wakes up periodically, selections rows from MySQL, and inserts them into BigQuery. We did this initially as a quick short-term solution. This approach has a number of problems, though:

- What happens if things fail? Should you retry?
- How do you know if the script succeeded?
- What if things need to happen in sequence, or depend on each other?
- What if the script runs slowly, and a second iteration of the script starts before the first finishes?

Most of these problems (and many others) are solved by workflow schedulers, so we opted to use one to run our ETL scripts. The four that we looked at were Oozie, Azkaban, Luigi, and Airflow.
This post is almost verbatim the same path we went through in 2015. Next two iterations of this are to migrate to a real-time pipeline, and then build a completely self-serve (automated) pipeline.
Chris Riccomini
@crccomini

There are three stages of data pipeline maturity.

1. Land (batch, few systems integrated)
2. Expand (realtime, many systems integrated)
3. On demand (self-serve, automated tooling, data catalog, MDM, DataOps, data mesh)

The current frontier is (2), trending toward (3).

12:16 PM - 18 Jul 2019
12 Retweets 43 Likes
The Future of Data Engineering

I have been thinking lately about where we’ve come in data engineering over the past few years, and about what the future holds for work in this area. Most of this thought has been framed in the context of what some of our teams are doing at WePay, but I believe the framework below applies more broadly, and is worth sharing.

Data engineering’s job is to help an organization move and process data. This generally requires two different systems, broadly speaking: a data pipeline, and a data warehouse. The data pipeline is responsible for moving the data, and the data
Six stages of data pipeline maturity

• Stage 0: None
• Stage 1: Batch
• Stage 2: Realtime
• Stage 3: Integration
• Stage 4: Automation
• Stage 5: Decentralization
Six stages of data pipeline maturity

- **Stage 0: None**
- **Stage 1: Batch**
- **Stage 2: Realtime**
- **Stage 3: Integration**
- **Stage 4: Automation**
- **Stage 5: Decentralization**
You might be ready for a data warehouse if...

• You have no data warehouse
• You have a monolithic architecture
• You need a data warehouse up and running *yesterday*
• Data engineering isn’t your full time job
Stage 0: None

Diagram:

- Monolith
- DB

Connection: Monolith to DB
Stage 0: None
WePay circa 2014
Problems

• Queries began timing out
• Users were impacting each other
• MySQL was missing complex analytical SQL functions
• Report generation was breaking
Six stages of data pipeline maturity

- Stage 0: None
- **Stage 1: Batch**
- Stage 2: Realtime
- Stage 3: Integration
- Stage 4: Automation
- Stage 5: Decentralization
You might be ready for batch if...

• You have a monolithic architecture
• Data engineering is your part-time job
• Queries are timing out
• Exceeding DB capacity
• Need complex analytical SQL functions
• Need reports, charts, and business intelligence
Stage 1: Batch

- Monolith
- DB
- Scheduler
- DWH
WePay circa 2016
Problems

• Large number of Airflow jobs for loading all tables
• Missing and inaccurate create_time and modify_time
• DBA operations impacting pipeline
• Hard deletes weren’t propagating
• MySQL replication latency was causing data quality issues
• Periodic loads cause occasional MySQL timeouts
Six stages of data pipeline maturity

• Stage 0: None
• Stage 1: Batch
• **Stage 2: Realtime**
• Stage 3: Integration
• Stage 4: Automation
• Stage 5: Decentralization
You might be ready for realtime if...

• Loads are taking too long
• Pipeline is no longer stable
• Many complicated workflows
• Data latency is becoming an issue
• Data engineering is your fulltime job
• You already have Apache Kafka in your organization
Stage 2: Realtime

Diagram showing a flow from Monolith to DB to Streaming Platform to DWH.
Streaming databases in realtime with MySQL, Debezium, and Kafka

By Chris Riccomini on Feb 11, 2017

Change data capture has been around for a while, but some recent developments in technology have given it new life. Notably, using Kafka as a backbone to stream your database data in realtime has become increasingly common.

If you’re wondering why you might want to stream database changes into Kafka, I highly suggest reading The Hardest Part About Microservices: Your Data. At WePay, we wanted to integrate our microservices and downstream databases with each other, so every system could get access to the data that it needed. We used Kafka as our data integration layer, so we needed a way to get our database data into it.

Last year, WePay’s engineering team published an excellent series of posts on their data pipeline. These included a discussion on how they stream MySQL data into Kafka. Their architecture involves a series of homogenous pieces of software to accomplish the task, notably schematizer and MySQL Streamer. The write-up triggered a thoughtful post on Debezium’s blog about a proposed equivalent architecture using Kafka connect, Debezium, and Confluent’s schema registry. This proposed architecture is what we’ve been implementing at WePay, and this post describes how we leverage Debezium and Kafka to connect and stream our MySQL databases into Kafka.

Architecture

The flow of data starts with each microservice’s MySQL database. These databases run in Google Cloud as CloudSQL. MySQL instances with GTIDs enabled. We’ve set up a downstream MySQL cluster specifically for Debezium. Each CloudSQL instance replicates its data into the Debezium cluster, which consists of two MySQL machines: a primary (active) server and secondary (passive) server. This single Debezium cluster is an operational trick to make it easier for us to operate Debezium. Rather than having Debezium connect to dozens of microservice databases directly, we can connect to just a single database. This also isolates Debezium from impacting the production OLTP workload that the master CloudSQL instances are handling.

We run one Debezium connector (in distributed mode on the Kafka connect framework) for each microservice database. Again, the goal is isolation. Theoretically, we could run a single Debezium connector that produces messages for all databases (since all microservice databases are in the Debezium cluster). This approach would actually be more resource efficient since each Debezium connector has to read MySQL’s entire log file. Anyway, we opted not to do this because we wanted to be able to bring Debezium connectors up and down, and configure them differently for each microservice DB.

The Debezium connectors feed the MySQL messages into Kafka (and add their schemas to the Confluent schema registry), where downstream systems can consume them. We use our Kafka connect BigQuery connector to load the MySQL data into BigQuery using BigQuery’s streaming API. This gives us a data warehouse in BigQuery that is usually less than 30 seconds behind the data that’s in production. Other microservices, stream processors, and data infrastructure consume the feeds as well.
WePay circa 2017
WePay circa 2017

Diagram showing the architecture:
- Service
- MySQL
- Debezium
- Kafka
- KCBQ
- BQ
Change data capture?
...an approach to data integration that is based on the identification, capture and delivery of the changes made to enterprise data sources.

https://en.wikipedia.org/wiki/Change_data_capture
Debezium sources

- MongoDB
- MySQL
- PostgreSQL
- SQL Server
- Oracle (Incubating)
- Cassandra (Incubating)
<table>
<thead>
<tr>
<th>Time</th>
<th>Session Title</th>
<th>Host/Chair(s)</th>
<th>Date/Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00AM</td>
<td>It Really is a Series of Tubes</td>
<td>Molly Wright Steenzen - AI Ethics &amp; Design Author, Designer, Professor, Research Leader</td>
<td>9:00AM</td>
<td></td>
</tr>
<tr>
<td>10:00AM</td>
<td>Microservices Patterns &amp; Practices</td>
<td>Host: Collin Brock</td>
<td>10:00AM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modern Data Architectures</td>
<td>Host: Owen Shipora</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modern CS in the Real World</td>
<td>Host: Weser Schuster</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Practices of DevOps &amp; Lean Thinking</td>
<td>Host: John Willis</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pushing the Web Forward: JavaScript, Frameworks, Transpilers, and WebAssembly</td>
<td>Host: Dylan Schiemann Host: Phil Haack</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Host: Alex Qin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:30AM</td>
<td>Managing Failure Modes in Microservice Architectures</td>
<td>Adrian Cockcroft - Amazon Web Services</td>
<td>10:30AM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Future of Data Engineering</td>
<td>Chris Riccomini - WayPay</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Anti-Entropy Using CRDTs on HA Datastores @ Netflix</td>
<td>Suresh Pudi - Netflix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DevOps &amp; Lean Thinking Panel</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Malcode: Types, Games and Machine Code</td>
<td>Mikhail Menkel - Microsoft</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pacific LMNO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>How Do We Heal?</td>
<td>Alex Qin - the code cooperative</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Real-time Log Analytics at Scale with Kafka</td>
<td>Kresen Tourn - Humo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Session Continues</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:00AM</td>
<td>User &amp; Device Identity for Microservices @ Netflix</td>
<td>Satyajit Thakur - Netflix</td>
<td>11:00AM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data Mesh Paradigm Shift in Data Platform Architecture</td>
<td>Zhamak Dehghan - ThoughtWorks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Talk You’ve Been Awaiting For</td>
<td>Steve Kalnisk - Rust Core Team</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JavaScript Open Space</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Holistic EdTech &amp; Diversity</td>
<td>Antonie Patern - Unlock Academy</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Next Era in Engineering Velocity</td>
<td>Bryan Holmamp - Code Climate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Session Continues</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:00PM</td>
<td>Stateful Programming Models in Serverless Functions</td>
<td>Chris Gillis - Microsoft</td>
<td>12:00PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Taming Large State: Lessons From Building Stream Processing</td>
<td>Sonali Sharma - Netflix &amp; Shryna Ana - Netflix</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Swift 4 TF</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DevOps &amp; Lean Thinking Open Space</td>
<td>Pacific LMNO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Goodbye View Source: Hello JavaScript in the Age of Compilers</td>
<td>Ania Hussain - google</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Pacific BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:00PM</td>
<td>Controlled Chaos: Taming Organic, Fedorated Growth of Microservices</td>
<td>Tobias Kunze - Glassostain</td>
<td>2:00PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modern Data Architectures Open Space</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MI/AI Panel</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The System of Profound Knowledge</td>
<td>Pacific LMNO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Build Your Own Web/Assembly</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Pacific BC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:45PM</td>
<td>Beyond Microservices: Streams, State and Scalability</td>
<td>Colin McCabe - Confluent</td>
<td>3:45PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kafka Needs No Keeper</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sortby: Why and How We Built a Typechecker for Ruby</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Incident Management in the Age of DevOps &amp; SRE</td>
<td>Pacific LMNO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Make Your Electron App Feel at Home</td>
<td>Pacific LMNO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Pacific LMNO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:00PM</td>
<td>Microservices Open Space</td>
<td>Pacific DEJK</td>
<td>5:00PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Probabilistic programming for software engineers</td>
<td>Pacific DEJK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Javascript</td>
<td>Pacific LMNO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Impact Starts With You</td>
<td>Pacific BC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EXHIBITOR RECEPTION IN EXHIBITOR AREAS ON MARKET STREET LEVEL AND PACIFIC LEVEL**
Kafka Connect BigQuery

- Open source connector that WePay wrote
- Stream data from Apache Kafka to Google BigQuery
- Supports GCS loads
- Supports realtime streaming inserts
- Automatic table schema updates
Problems

• Pipeline for Datastore was still on Airflow
• No pipeline at all for Cassandra or Bigtable
• BigQuery needed logging data
• Elastic search needed data
• Graph DB needed data
Six stages of data pipeline maturity

- Stage 0: None
- Stage 1: Batch
- Stage 2: Realtime
- **Stage 3: Integration**
- Stage 4: Automation
- Stage 5: Decentralization
You might be ready for integration if...

- You have microservices
- You have a diverse database ecosystem
- You have many specialized derived data systems
- You have a team of data engineers
- You have a mature SRE organization
Stage 3: Integration
WePay circa 2019

Service → MySQL → Debezium → Kafka → KCBQ → BQ

PHP Monolith → MySQL → Debezium → Kafka

Service → Cassandra → Debezium

Service → Waltz → KCW

Service → Graph DB
Waltz: A Distributed Write-Ahead Log

By Yasuhiro Matsuoka on Sep 9, 2014

We are happy to announce the open source release of Waltz. Waltz is a distributed write-ahead log. It was initially designed to be the ledger of money transactions on the WePay system and was generalized for broader use cases of distributed systems that require serializable consistency. Waltz is similar to existing log systems like Kafka in that it accepts/originates/propagates transaction data produced/consumed by many services. However, unlike other systems, Waltz provides a machinery that facilitates a serializable consistency in distributed applications. It detects conflicting transactions before they are committed to the log. Waltz is regarded as the single source of truth rather than the database, and it makes a highly reliable log-centric system architecture.

Background

Databases

The WePay system has been constantly growing to handle more traffic and more functionalities. We split a large service into smaller services to keep the system manageable when it makes sense. Each service typically has its own database. For better isolation, it is not shared with other services.

It is not trivial to keep all databases consistent when there are faults such as network failures, process failures, and machine failures. Services interact with each other over the network. Interactions often result in database updates on both sides. Faults may cause inconsistencies between the databases. Most such inconsistencies are fixed by daemon threads that perform check-and-repair operations periodically. But not every repair can be automated. Sometimes manual operations are required.

On top of this, databases are replicated for fault tolerance. We use MySQL async replication. When the primary region goes down, a region failure will happen, and the backup region will take over the processing so that we can continue processing payments. Multi-region replication has its own issues. A database update in the master database will not appear in a slave database instantly. There is always a latency, and replication lags are the norm. There is no guarantee that new master databases have all up-to-date data nor that they are in sync with each other.

Stream Oriented Processing

We employ asynchronous processing in many places. We want to defer updates that don’t require immediate consistency. This makes the main transactions lighter and improves the response and throughput. We do so by using a stream-oriented processing with Kafka. A service updates its own database and writes messages to Kafka at the same time. The service or a different service asynchronously performs another database update when it consumes Kafka messages. This works well, but the drawback is that a service has to write to two separate storage systems, a database and Kafka. We still need check-and-repair.

Basic Idea

Waltz is what we describe as a write-ahead log. This recorded log is neither the output of a change-data-capture from a database nor a secondary output from an application. It is the primary information of the system.
Metcalf's law
Kafka is your escape hatch

I’ve become much more comfortable with the idea of vendor lock-in. Or rather, I don’t feel as locked in as I used to. The odd thing is, I’m using more proprietary systems than I ever have before (thanks to the cloud). Apache Kafka is what’s making me comfortable. Specifically, Kafka connect.

The best place to follow me is on my mailing list. Get new posts and recommended reading every Friday.
Problems

• Add new channel to replica MySQL DB
• Create and configure Kafka topics
• Add new Debezium connector to Kafka connect
• Create destination dataset in BigQuery
• Add new KCBQ connector to Kafka connect
• Create BigQuery views
• Configure data quality checks for new tables
• Grant access to BigQuery dataset
• Deploy stream processors or workflows
Six stages of data pipeline maturity

• Stage 0: None
• Stage 1: Batch
• Stage 2: Realtime
• Stage 3: Integration
• Stage 4: Automation
• Stage 5: Decentralization
You might be ready for automation if...

- Your SREs can’t keep up
- You’re spending a lot of time on manual toil
- You don’t have time for the fun stuff
Realtime Data Integration

Stage 4: Automation

Automated Data Management
- Data Catalog
- RBAC/IAM/ACL
- DLP
- ...

Automated Operations
- Orchestration
- Monitoring
- Configuration
- ...

DB
Service
Streaming Platform
NoSQL
DWH
New SQL
Graph DB
Search
Automated Operations
“If a human operator needs to touch your system during **normal operations**, you have a bug.”

-- Carla Geisser, Google SRE
Normal operations?

- Add new channel to replica MySQL DB
- Create and configure Kafka topics
- Add new Debezium connector to Kafka connect
- Create destination dataset in BigQuery
- Add new KCBQ connector to Kafka connect
- Create BigQuery views
- Configure data quality checks for new tables
- Granting access
- Deploying stream processors or workflows
Automated operations

- Terraform
- Ansible
- Helm
- Salt
- CloudFormation
- Chef
- Puppet
- Spinnaker
provider "kafka" {
    bootstrap_servers = ["localhost:9092"]
}

resource "kafka_topic" "logs" {
    name               = "systemd_logs"
    replication_factor = 2
    partitions         = 100

    config = {
        "segment.ms"  = "20000"
        "cleanup.policy" = "compact"
    }
}
provider "kafka-connect" {
    url = "http://localhost:8083"
}

resource "kafka-connect_connector" "sqlite-sink" {
    name = "test-sink"

    config = {
        "name" = "test-sink"
        "connector.class" = "io.confluent.connect.jdbc.JdbcSinkConnector"
        "tasks.max" = "1"
        "topics" = "orders"
        "connection.url" = "jdbc:sqlite:test.db"
        "auto.create" = "true"
    }
}
But we were doing this... why so much toil?

- We had Terraform and Ansible
- We were on the cloud
- We had BigQuery scripts and tooling
Spending time on data management

• Who gets access to this data?
• How long can this data be persisted?
• Is this data allowed in this system?
• Which geographies must data be persisted in?
• Should columns be masked?
Regulation is coming
Regulation is coming here

GDPR, CCPA, PCI, HIPAA, SOX, SHIELD, ...
Automated Data Management
Set up a data catalog

• Location
• Schema
• Ownership
• Lineage
• Encryption
• Versioning
explorers
Sep 13, 2004 - Mar 21, 2019
Data for famous world explorers

Columns

explorer_id int
primary key for explorers

first_name string
Explorer's given name

last_name string
Explorer's family name

birthday date
Explorer's date of birth

place_of_origin string
Country of birth

regions_explored int
Count of regions explored. Regions defined in explored_regions

distance_travelled int
Total kilometers travelled based only on entries in trips table

nationality string
Country of citizenship, may be different from place_of_origin
## explorers

Data for famous world explorers

### Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>explorer_id</td>
<td>int</td>
<td>primary key for explorers</td>
</tr>
<tr>
<td>first_name</td>
<td>string</td>
<td>Explorer's given name</td>
</tr>
<tr>
<td>last_name</td>
<td>string</td>
<td>Explorer's family name</td>
</tr>
<tr>
<td>birthday</td>
<td>date</td>
<td>Explorer's date of birth</td>
</tr>
<tr>
<td>place_of_origin</td>
<td>string</td>
<td>Country of birth</td>
</tr>
<tr>
<td>regions_explored</td>
<td>int</td>
<td>Count of regions explored. Regions defined in explored_regions</td>
</tr>
<tr>
<td>distance_travelled</td>
<td>int</td>
<td>Total kilometers travelled based only on entries in trips table</td>
</tr>
<tr>
<td>nationality</td>
<td>string</td>
<td>Country of citizenship, may be different from place_of_origin</td>
</tr>
</tbody>
</table>
explorers

Sep 13, 2004 – Mar 21, 2019
Data for famous world explorers

Columns

explorer_id int
  primary key for explorers

first_name string
  Explorer's given name

last_name string
  Explorer's family name

birthday date
  Explorer's date of birth

place_of_origin string
  Country of birth

regions_explored int
  Count of regions explored. Regions defined in explored_regions

distance_travelled int
  Total kilometers travelled based only on entries in trips table

nationality string
  Country of citizenship, may be different from place_of_origin

Amundsen was last indexed on March 21st 2019 at 6:21:41 am
Explorers
Sep 13, 2004 – Mar 21, 2019
Data for famous world explorers

Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>explorer_id</td>
<td>int</td>
<td>Primary key for explorers</td>
</tr>
<tr>
<td>first_name</td>
<td>string</td>
<td>Explorer's given name</td>
</tr>
<tr>
<td>last_name</td>
<td>string</td>
<td>Explorer's family name</td>
</tr>
<tr>
<td>birthday</td>
<td>date</td>
<td>Explorer's date of birth</td>
</tr>
<tr>
<td>place_of_origin</td>
<td>string</td>
<td>Country of birth</td>
</tr>
<tr>
<td>regions_explored</td>
<td>int</td>
<td>Count of regions explored. Regions defined in explored_regions</td>
</tr>
<tr>
<td>distance_travelled</td>
<td>int</td>
<td>Total kilometers travelled based only on entries in trips table</td>
</tr>
<tr>
<td>nationality</td>
<td>string</td>
<td>Country of citizenship, may be different from place_of_origin</td>
</tr>
</tbody>
</table>

Owned by:
- email@lyft.com
- user@lyft.com

Frequent users:
- User 1
- User 2
- User 3
- User 4

Generated by:
- explorers.of_explorers

Source code:
- explorers.of_explorers

Table lineage (beta):
- explorers.of_explorers

Table profile (beta):
- Preview data

Tags:
- explorers
explorers
Sep 13, 2004 - Mar 21, 2019
Data for famous world explorers

Columns

explorer_id int
primary key for explorers

first_name string
Explorer's given name

last_name string
Explorer's family name

birthday date
Explorer's date of birth

place_of_origin string
Country of birth

regions_explored int
Count of regions explored. Regions defined in explored_regions

distance_travelled int
Total kilometers travelled based only on entries in trips table

nationality string
Country of citizenship, may be different from place_of_origin
Stage 4: Automation

Automated Data Management
- Data Catalog
- RBAC/IAM/ACL
- DLP
- ...

Automated Operations
- Orchestration
- Monitoring
- Configuration
- ...

Realtime Data Integration

Stage 4: Automation

DB Service
Streaming Platform
DWH
NoSQL Service
New SQL
Graph DB
Search

Automated Operations
Orchestration
Monitoring
Configuration
Configure your access

- RBAC
- IAM
- ACL
Configure your policies

• Role based access controls
• Identity access management
• Access control lists
provider "kafka" {
  bootstrap_servers = ["localhost:9092"]
  ca_cert = file("../secrets/snakeoil-ca-1.crt")
  client_cert = file("../secrets/kafkacat-ca1-signed.pem")
  client_key = file("../secrets/kafkacat-raw-private-key.pem")
  skip_tls_verify = true
}

resource "kafka_acl" "test" {
  resource_name = "syslog"
  resource_type = "Topic"
  acl_principal = "User:Alice"
  acl_host = "*"
  acl_operation = "Write"
  acl_permission_type = "Deny"
}
Automate management

• New user access
• New data access
• Service account access
• Temporary access
• Unused access
Detect violations

• Auditing
• Data loss prevention
Cloud Data Loss Prevention

Automatically discover and redact sensitive data everywhere.

Classify and redact sensitive data

Cloud DLP helps you better understand and manage sensitive data. It provides fast, scalable classification and redaction for sensitive data elements like credit card numbers, names, social security numbers, USS and selected international identifier numbers, phone numbers, and GCP credentials. Cloud DLP classifies this data using more than 90 predefined detectors to identify patterns, formats, and checksums, and even understands contextual clues. You can optionally redact data as well, using techniques like masking, secure hashing, tokenization, bucketing, and format-preserving encryption. Try Cloud DLP in this demo application.

Discover and classify sensitive data

One of the first steps to properly managing your sensitive data is knowing where it exists. Cloud DLP gives you the power to scan, discover, classify, and report on data from virtually anywhere. Cloud DLP has native support for scanning and classifying sensitive data in Cloud Storage, BigQuery, and Cloud Databases and a streaming content API to enable support for additional data sources, custom workloads and applications.

Automatically mask your data to safely unlock more of the cloud

Today your data is your most critical asset. Cloud DLP provides tools to classify, mask, analyze, and interpret sensitive elements in your environment in ways like...
Detecting sensitive data

{
    "item":{
        "value":"My phone number is (415) 555-0890"
    },
    "inspectConfig":{
        "includeQuote":true,
        "minLikelihood":"POSSIBLE",
        "infoTypes":{
            "name":"PHONE_NUMBER"
        }
    }
}

{
    "result":{
        "findings":[
            {
                "quote":"(415) 555-0890",
                "infoType":{
                    "name":"PHONE_NUMBER"
                },
                "likelihood":"VERY_LIKELY",
                "location":{
                    "byteRange":{
                        "start":"19",
                        "end":"33"
                    }
                }
            }
        ]
    }
}
Progress

• Users can find the data that they need
• Automated data management and operations
Problems

• Data engineering still manages configuration and deployment
Six stages of data pipeline maturity

• Stage 0: None
• Stage 1: Batch
• Stage 2: Realtime
• Stage 3: Integration
• Stage 4: Automation
• **Stage 5: Decentralization**
You might be ready for decentralization if...

- You have a fully automated realtime data pipeline
- People still come to you to get data loaded
If we have an automated data pipeline and data warehouse, do we need a **single team** to manage this?
Stage 5: Decentralization

- Service
  - NoSQL
  - DB
  - New SQL

- Streaming Platform

- Search
- DWH
- Graph DB

Automated Data Management
- Data Catalog
- RBAC/IAM/ACL
- DLP
- ...

Automated Operations
- Orchestration
- Monitoring
- Configuration
- ...

Realtime Data Integration
From monolith to microservices microwarehouses
How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh

Many enterprises are investing in their next-generation data lake, with the hope of democratizing data at scale to provide business insights and ultimately make automated intelligent decisions. Data platforms based on data lake architectures have common failure modes that lead to unfilled promises at scale. To address these failure modes we need to shift from the centralized paradigm of a lake, or its predecessor data warehouse. We need to shift to a paradigm that draws from modern distributed architectures: considering domains as the first-class concern, applying platform thinking to create self-service data infrastructure, and treating data as a product.

28 May 2019

CONTENTS

The current enterprise data platform architecture
Architectural failure modes
Centralized and monolithic
Coupled pipeline decomposition
Stale and hyper-specialized ownership
The next enterprise data platform architecture
Data and distributed domain-driven architecture convergence
Domain oriented data decomposition and ownership
Source oriented domains data
Consumer oriented and shared domain data
Distributed pipelines as domain internal implementation
Data and product thinking convergence
Domain data as a product
Closurizable
Addressable
Trustworthy and truthful
Self-describing semantics and syntax
Inter-operable and governed by global standards
Secure and governed by a global access control
Domain data across functional teams
Data and self-service platform design convergence
The paradigm shift towards a data mesh

Becoming a data-driven organization remains one of the top strategic goals of many companies I work with. My clients are well aware of the benefits of becoming intelligently empowered: providing the best customer experience based on data and hyper-personalization; reducing operational costs and time through data-driven optimizations; and giving employees superpowers with trend analysis and business intelligence. They have been investing heavily in building enablers such as data and intelligence platforms. Despite increasing effort and investment in building such
<table>
<thead>
<tr>
<th>Time</th>
<th>Session Title</th>
<th>Presenter(s)</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00AM</td>
<td>WOMEN &amp; ALLIES IN TECH COMMUNITY BREAKFAST CO-SPONSORED BY NETFLIX - REGISTRATION REQUIRED</td>
<td>Molly Wright Steen - AI, Ethics &amp; Design: Author, Designer, Professor, Research Leader</td>
<td>Host: Nithi Bhardi</td>
</tr>
<tr>
<td>10:15AM</td>
<td>Microservices Patterns &amp; Practices</td>
<td>Host: Colin Breck</td>
<td>Ballroom A</td>
</tr>
<tr>
<td></td>
<td>Modern Data Architectures</td>
<td>Host: Owen Shapira</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>Modern CS in the Real World</td>
<td>Host: Werner Schuster</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>Practices of DevOps &amp; Lean Thinking</td>
<td>Host: John Willis</td>
<td>Bayview A</td>
</tr>
<tr>
<td></td>
<td>Pushing the Web Forward: JavaScript, Frameworks, Transpilers, and WebAssembly</td>
<td>Host: Dylan Schiemann, Host: Phil Haack</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Host: Alex Qin</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td>10:30AM</td>
<td>Managing Failure Modes in Microservices Architectures</td>
<td>Chris Riosco - Microsoft</td>
<td>Ballroom A</td>
</tr>
<tr>
<td></td>
<td>Future of Data Engineering</td>
<td>Adriaan Cokx - Amazon Web Services</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>Anti-Entropy Using CRDTs on HA Databases</td>
<td>Chris Riosco - Microsoft</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>DevOps &amp; Lean Thinking Panel</td>
<td>Host: John Willis</td>
<td>Bayview A</td>
</tr>
<tr>
<td></td>
<td>Malware: Types, Games and Machine Code</td>
<td>Host: Phil Haack</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>How Do We Heal? Alex Qin - The Code Cooperative</td>
<td>Host: Alex Qin</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Host: Alex Qin</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>SPONSORED SOLUTIONS TRACK III</td>
<td>Host: Nithi Bhardi</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>SPONSORED SOLUTIONS TRACK IV</td>
<td>Host: Nithi Bhardi</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td>11:15AM</td>
<td>BREAK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:30AM</td>
<td>User &amp; Device Identity for Microservices &amp; Netflix Scale</td>
<td>Satyajit Thadnawal - Netflix</td>
<td>Ballroom A</td>
</tr>
<tr>
<td></td>
<td>Data Mesh Paradigm Shift in Data Platform Architecture</td>
<td>Zbigniew Dehghan - ThoughtWorks</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>The Talk You’ve Been Awaiting For</td>
<td>Steve Klabnick - Rust Care Team</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Mapping the Evolution of Software Technical Systems</td>
<td>Cat Swett - Ticketmaster</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>JavaScript Open Space</td>
<td>Facilitated Peer Sharing</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Holistic: EdTech &amp; Diversity</td>
<td>Antoine Paterns - Unlock Academy</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>The Next Era in Engineering</td>
<td>Bryan Holinkamp - Code Climate</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td>11:50AM</td>
<td>BREAK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:00PM</td>
<td>Statetable Programming Models in Serverless Functions</td>
<td>Chris Gillam - Microsoft</td>
<td>Ballroom A</td>
</tr>
<tr>
<td></td>
<td>Taming Large State: Lessons From Building Stream Processing</td>
<td>Sonali Sharma - Netflix &amp; Shunya Aziz - Netflix</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>Swift 4 TF</td>
<td>Paige Bailey - Google</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>DevOps &amp; Lean Thinking Open Space</td>
<td>Facilitated Peer Sharing</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Goodbye View Source: Hello JavaScript In The Age of Compilers</td>
<td>Alban Hussein - npm</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Host: Alex Qin</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>The Case for Human-Centric Observability</td>
<td>Austin Parker - LightStep</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>SPONSORED SOLUTIONS TRACK III</td>
<td>Host: Nithi Bhardi</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>SPONSORED SOLUTIONS TRACK IV</td>
<td>Host: Nithi Bhardi</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td>1:40PM</td>
<td>CONTROLLED CHAOS: Taming Organic, Federated Growth of Microservices</td>
<td>Tobias Kuske - Glasscows</td>
<td>Ballroom A</td>
</tr>
<tr>
<td></td>
<td>Modern Data Architectures Open Space</td>
<td>Facilitated Peer Sharing</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>The System of Profound Knowledge</td>
<td>Ben Rockwood - Packet</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Build Your Own Web/Assembly</td>
<td>Cihan Eberhardt - Scott Logic</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Host: Alex Qin</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>My Code Is Broken But So Is This Debugging Experience</td>
<td>Matt Robboll - Sentry</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td>2:30PM</td>
<td>BREAK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:45PM</td>
<td>Beyond Microservices: Streams, State and Scalability</td>
<td>Owen Shapira - Confluent</td>
<td>Ballroom A</td>
</tr>
<tr>
<td></td>
<td>Kafka Needs No Keeper</td>
<td>Colin McCabe - Confluent</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>Sorbet: Why and How We Built a Typechecker for Ruby</td>
<td>Dmitry Porozhnoi - Steppe</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Incident Management in the Age of DevOps &amp; SRE</td>
<td>Giusse Emidio - Hundec</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Make Your Electron App Feel at Home Everywhere</td>
<td>Kilam Valdett - Polypody</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Socially Conscious Software</td>
<td>Host: Alex Qin</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Aurl</td>
<td>Gil Time - Aurl</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td>4:40PM</td>
<td>BREAK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:00PM</td>
<td>Microservices Open Space</td>
<td>Facilitated Peer Sharing</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>Practical Change Data Streaming Use Cases With Apache Kafka &amp; Debezium</td>
<td>Gaurav Morling - Red Hat</td>
<td>Ballroom BC</td>
</tr>
<tr>
<td></td>
<td>Probabilistic Programming for Software Engineers</td>
<td>Michael Umlay - Facebook</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Balancing Priorities: Revenue Generation vs. Revenue Protection</td>
<td>Dominique DeCarvalho - Tinkoff</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>JavaScript</td>
<td>Pacific DEKJ</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td></td>
<td>Impact Starts With You</td>
<td>Julia Nguyen - Multichip</td>
<td>Pacific DEKJ</td>
</tr>
<tr>
<td>6:15PM</td>
<td>BREAK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EXHIBITOR RECEPTION IN EXHIBITOR AREAS ON MARKET STREET LEVEL AND PACIFIC LEVEL
Partial decentralization

- Raw tools are exposed to other engineering teams
- Requires Git, YAML, JSON, pull requests, terraform commands, etc.
Full decentralization

• Polished tools are exposed to everyone
• Security and compliance manage access and policy
• Data engineering manages data tooling and infrastructure
• Everyone manages data pipelines and data warehouses
Realtime Data Integration

Modern Data Pipeline

Automated Data Management
- Data Catalog
- RBAC/IAM/ACL
- DLP
- ...

Automated Operations
- Orchestration
- Monitoring
- Configuration
- ...

- Streaming Platform
  - Service
  - NoSQL
  - DB
  - New SQL
  - Search
  - DWH
  - Graph DB

- Automated Operations
Thanks!
(..and we’re hiring)