

Lyft and the California Consumer Privacy Act

Bootstrapping user data deletions and export.

Alejo Grigera Sutro & Shankar Garikapati

PEPR 2022



Lyft In A Nutshell

18 Million

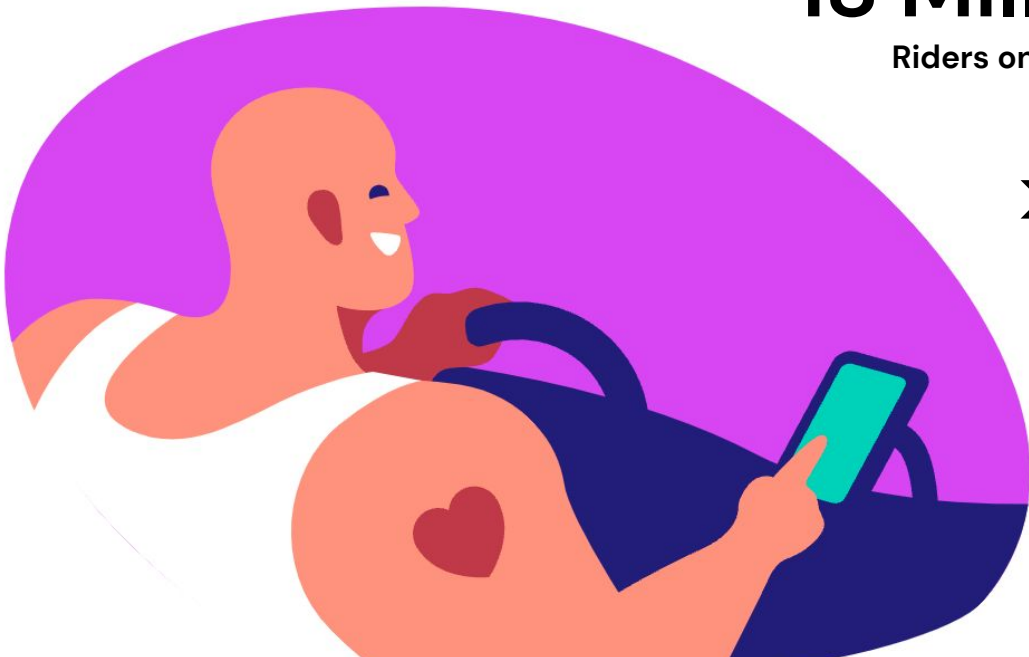
Riders on our platform.

>1,000



Microservices powering the platform.

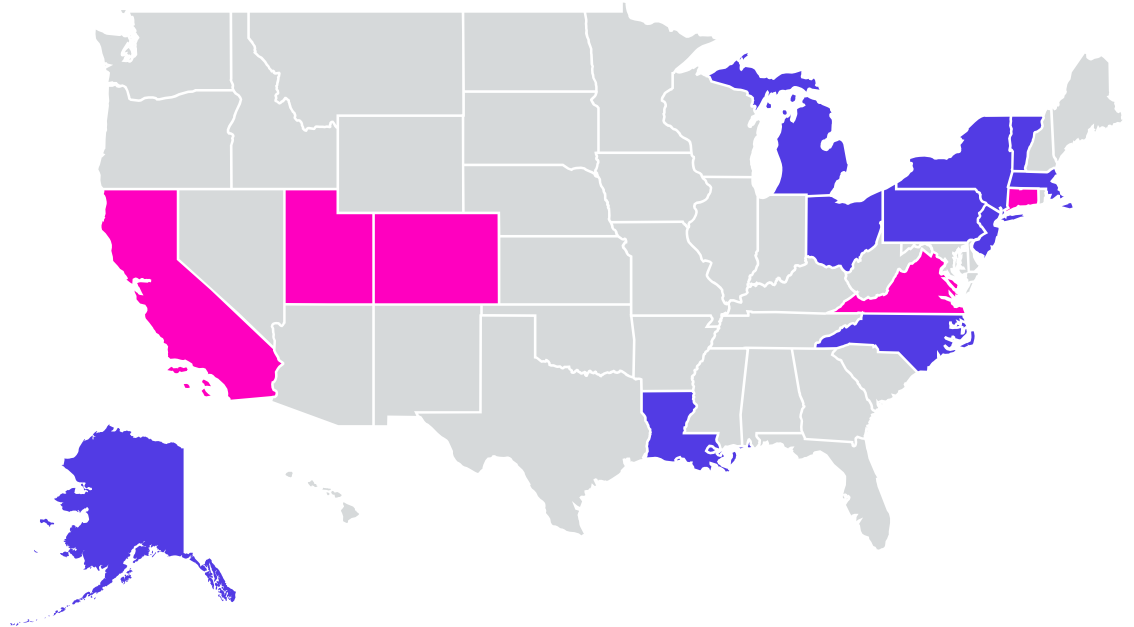
10 Years

Lyft has been in business.



CCPA Wasn't A Task. It Was A Beginning

-  Signed Privacy Legislation
-  Privacy Legislation In Committee



Export & Deletion Strategic Goals

One-size-fits-all solution.

Our products are as diverse as our users. That can mean special infrastructure or needs, so whatever we build has to work everywhere.

Secure and usable.

Changing user data should always be done with care! Our solution has to be trustworthy, reliable, and- most importantly- useful to our users.

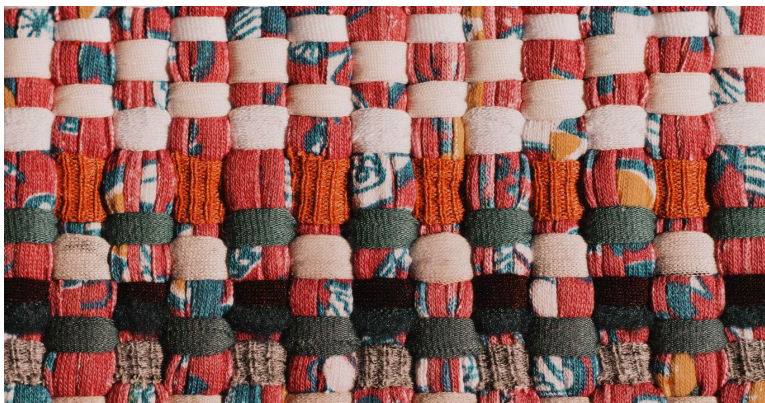
Balances competing needs.

Some business functions need data to be consistent in order to work properly. We can't have people deleting their account halfway through a ride.



Solutions & Implementation

Our Platform & Architecture



Non-monolithic infrastructure

Lyft teams design, build, and manage microservices on AWS infrastructure. Services interact with each other using RPC and do not share any stateful resources.



High freedom, high responsibility

Teams follow a decentralized governance structure. They're responsible for all technical aspects of the software they build.

Strategic Goal: A One-Size-Fits-All Solution

We wanted a common API with one standard SLA that applies across a diverse set of services.

Centralization wasn't an option

Any centralized solution would fail, since they can't accommodate local conditions. Some services and datastores have huge differences in their latencies and operating costs.

Decentralization wasn't an option either

Not very effective from a business perspective. The cost to benefit ratio of a decentralized system was poor. Setting up and managing any coordination mechanism would be expensive.



Shared Responsibility model

1

Federated Architecture

- Cater to local conditions, while pursuing high level privacy goals uniformly
- Keep data lifecycle ownership with product teams

2

A Simple Finite State Machine

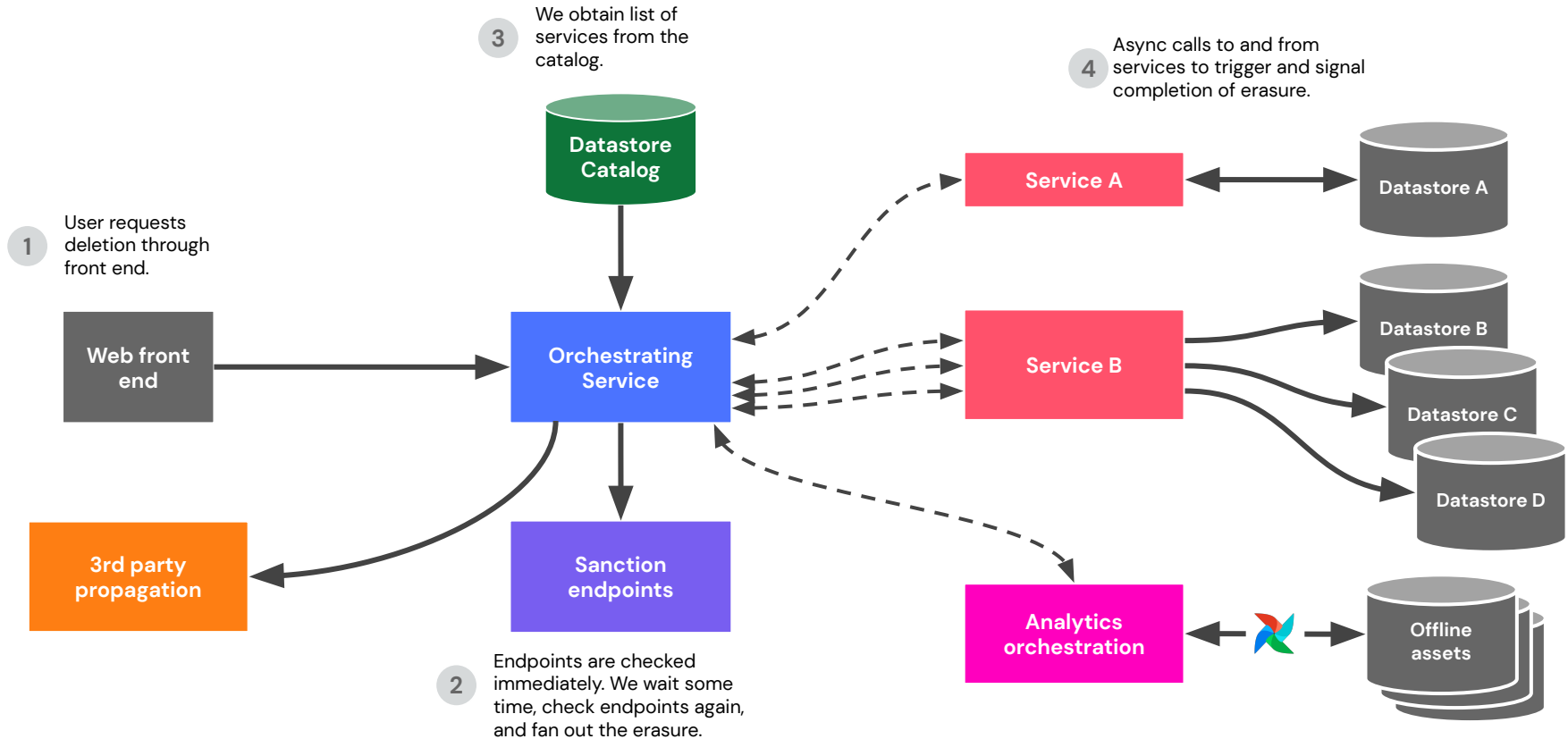
- Tasks as a building block to get things done
- Use intuitive, meaningful states like pending, events sent, completed, and failed

3

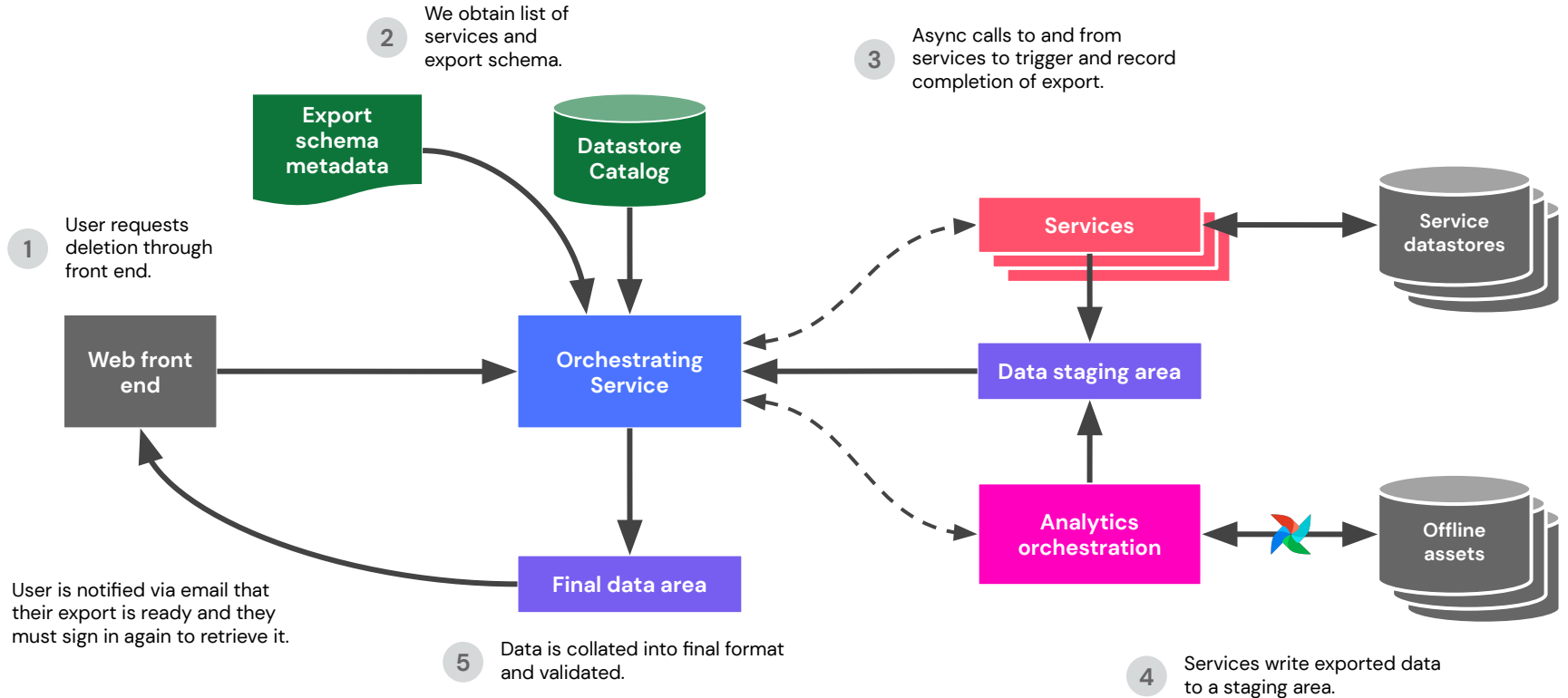
Event-Driven Philosophy

- Allow services to manage their own local state and interact with the orchestrator
- Allow messages can be arbitrarily delayed and reordered
- Use a robust set of checks to ensure safety and liveness

Erasure Requests



Export Requests



Strategic Goal: Secure, Reliable, And Usable

Designed for safety

We wanted systems to help protect developers from making mistakes. To build a secure and reliable system, we would need automation that catches problems early and raises warnings when it's still easy to make changes.

Principle of least astonishment

Deletion and export mechanisms should be designed so that product engineers can easily understand how the whole system works, and what part they are responsible for. This helps them to debug and fix any problems.



Bake Privacy into SDLC

1

Maximize Developer Experience

- Privacy as Code, using privacy annotations in Terraform, IDL files
- Overcome human errors/omissions with code automation and testing
- Use abstraction to provide a uniform experience

2

DRY Principle

- Common libraries that cater to a wide range of microservices
- Observability tools and runbooks to diagnose failures

Strategic Goal: Balancing Competing Needs

Some deletions may have hurdles

There may be valid and important use cases where data deletion could cause serious problems. If the service is still in progress or if we have pending payments to drivers or from riders, we don't want to delete data prematurely.

Sanction Endpoints delegate the decision

Our solution is to allow services with better context and situational awareness to assess the decision. When situations complicate deletion, we try to inform the user about limitations and what options they might have available.

Data Vaulting offers a compromise

Competing requirements- especially legal ones- might force us to store data for a long time. Vaulting lets us extract data from various datastores, store a copy in a high-security and purpose-limited vault, and then delete the originals.



Need for analytics orchestration



Exactly-once vs at-least-once guarantee

Deleting user data once may not be sufficient. Data might be in transit from other internal data-storage systems.



Design for failures

Since ETL processes can be brittle, designing for an optimal scenario is not sufficient, especially for destructive operations that involve rewriting offline datastores with layers of abstraction.

Engineering lessons learned

1

Avoid distributed coordination

2

Systems & users are unpredictable

3

Good observability for End-to-End Signals

**Then What
Happened?**

We built & launched it

On time, too!

- **One-size-fits-all solution** by using a federated process that “shifts left” privacy concerns.
- **Secure and usable** by developing event-driven models that make system diversity a strength.
- **Balanced competing needs** with fault-tolerant, non-blocking, idempotent actions and vaulting.



And Our Planning Paid Off



Scale

Through mindful system development, we can cater to diverse services and technical configurations.

Observability

Logging infrastructure and careful tracking of requests let us troubleshoot problem areas.

Resilience

No single points of failure that threatens business systems.

Thank you

Lyft is hiring in US and Mexico! If you're passionate about privacy and security by design and building the infrastructure that powers it all, come join our team.

<https://www.lyft.com/careers>

AlejoGrigeraSutro@lyft.com

SGarikapati@lyft.com

