

Taking Arrow Flight Supersonic

Apache Arrow 東京ミートアップ2025

David Li (リー・デイビット)

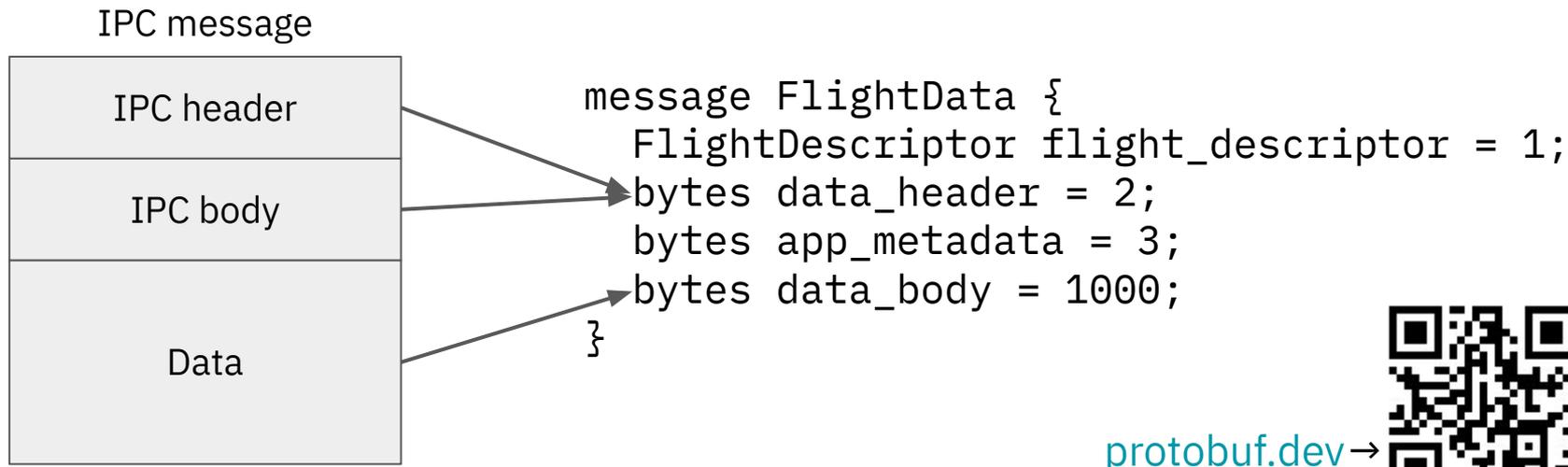
 lidavidm

Voltron Data (ヴォルトロンデータ)

What is Flight?

Flightとは?

(1) An encoding of Arrow IPC data in Protobuf
Arrow IPCデータをProtobufでエンコードする方法

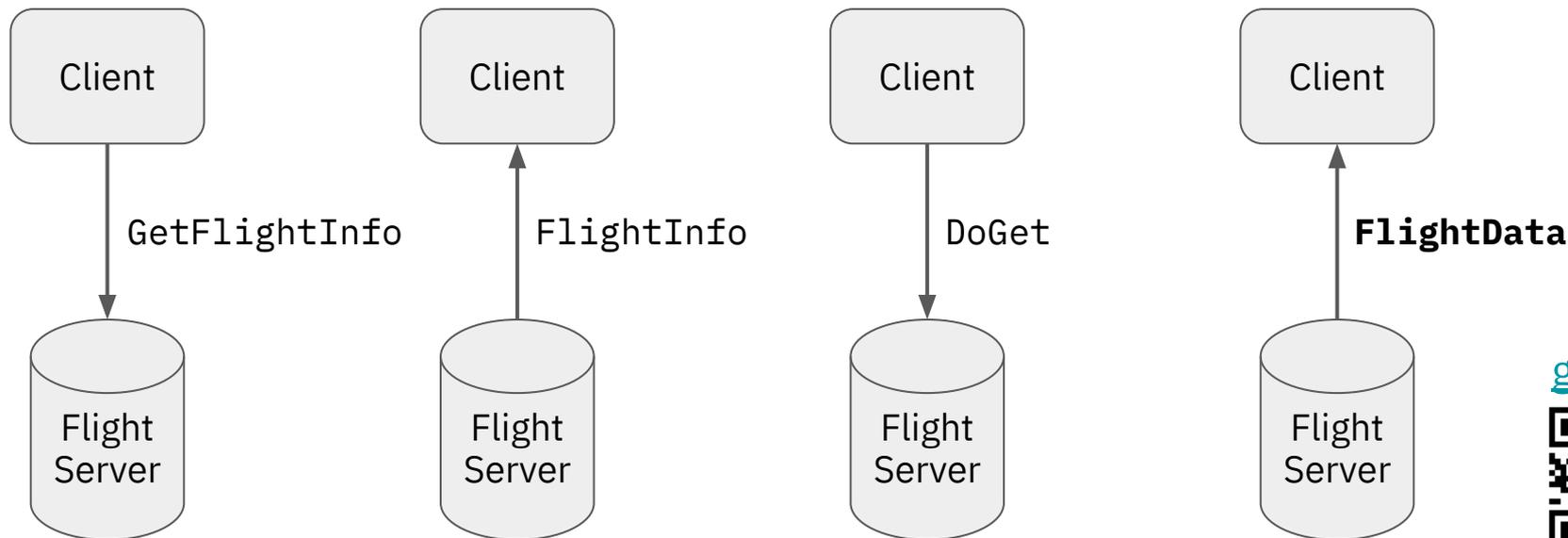


*翻訳は間違いがあるかもしれません。ご注意ください。

What is Flight?

Flightとは?

- (2) A gRPC service definition for building “data services” using (1)
(1)を使ってデータサービスを開発するためのgRPCサービス定義



What is Flight?

Flightとは?

(3) A framework for building services implementing (2)
(2)を実装しているサービスを開発するためのフレームワーク

Used by:

ユーザー:



Airport Extension
for DuckDB



...and more!

Why is Flight great?

Flightのメリットは？

- Easy to get started
簡単に使い始められる
 - Just `import pyarrow.flight`
`import pyarrow.flight`だけで済む
- Optimized for Arrow
Arrow向けに最適化された
 - Flight hooks into gRPC & Protobuf to achieve **zero-copy serialization**
FlightはgRPCとProtobufにフックを入れて、**ゼロコピーシリアライズ**ができる
- Recommended by Arrow developers
Arrowの開発者によって推奨された

But there are some things
that don't work well...

しかし、うまく使えない
ポイントもある…

Flight isn't really gRPC

FlightはgRPCと違う

- Can't define your own RPC endpoints!
自分のRPCエンドポイントを定義できない！
- Can't use gRPC APIs (in C++/Python)
gRPC APIを使えない (C++、Pythonに)
 - No async, telemetry, load balancing APIs
Flightには非同期、テレメトリー、ロードバランシングAPIがない
 - Google adds new features constantly
Googleはしきりに新しい機能を開発している
- Less documentation/smaller community
gRPCよりFlightの方は文書が少ない、コミュニティが小さい

Flight requires tedious boilerplate

Flightを使えば、定型コードが多い

gRPC

1. `rpc CancelQuery (CancelQueryRequest) returns (CancelQueryResponse) {}`
2. Recompile Protobufs
3. Implement `CancelQuery`

**Why should developers
use a framework that
makes their lives
harder?**

Flight

1. Define `CancelQueryRequest` and `CancelQueryResponse`
2. Recompile Protobufs
3. Decide which Flight method to “hijack”
4. Implement custom request router in the server
5. Implement the RPC handler
6. Implement custom Flight client wrapper
7. Add OpenTelemetry instrumentation

Flight breaks standard tooling

Flightを使えば、標準のツールを使えない

Example: DataDog

- Using gRPC: your service is automatically instrumented
gRPCの場合は、サービスは自動計装されている
- Using Flight: you only see Flight's RPC calls, not your own
Flightの場合は、自分のAPIコールが見えなくてFlightのコールだけが見える
 - Must manually annotate your service to get telemetry
サービスを手動で計装しなければならない

Example: grpcurl

- Using gRPC: make requests easily, grpcurl translates JSON to Protobuf via server reflection
gRPCの場合は、grpcurlはJSONをProtobufに交換できてリクエストを簡単に送れる
- Using Flight: must manually construct binary payloads, no type safety
Flightの場合は、バイナリペイロードを手動で作成しなければならない、型安全性もない

Example: Flight SQL

例：Flight SQL

- Lots of boilerplate 定型コードが多い、例えば…
 - Routing requests リクエストルーター
 - Parsing/serializing messages リクエストを解析し、シリアライズする
 - Handwritten client 手動で書いたクライアント
 - Hard to maintain, extend! メンテナンスも開発も難しい
- Caused trouble for [InfluxDB](#) InfluxDBはFlightに問題があったことある
- Doesn't even use many of the predefined RPC methods
Flight SQLはFlightの定義したRPCメソッドをほとんど使ってない
 - Wants to define lots of its own RPC methods, but can't
反面に自分のRPCメソッドを定義したい、しかしできない

If we fix these problems, we can
make Flight better for expert users!

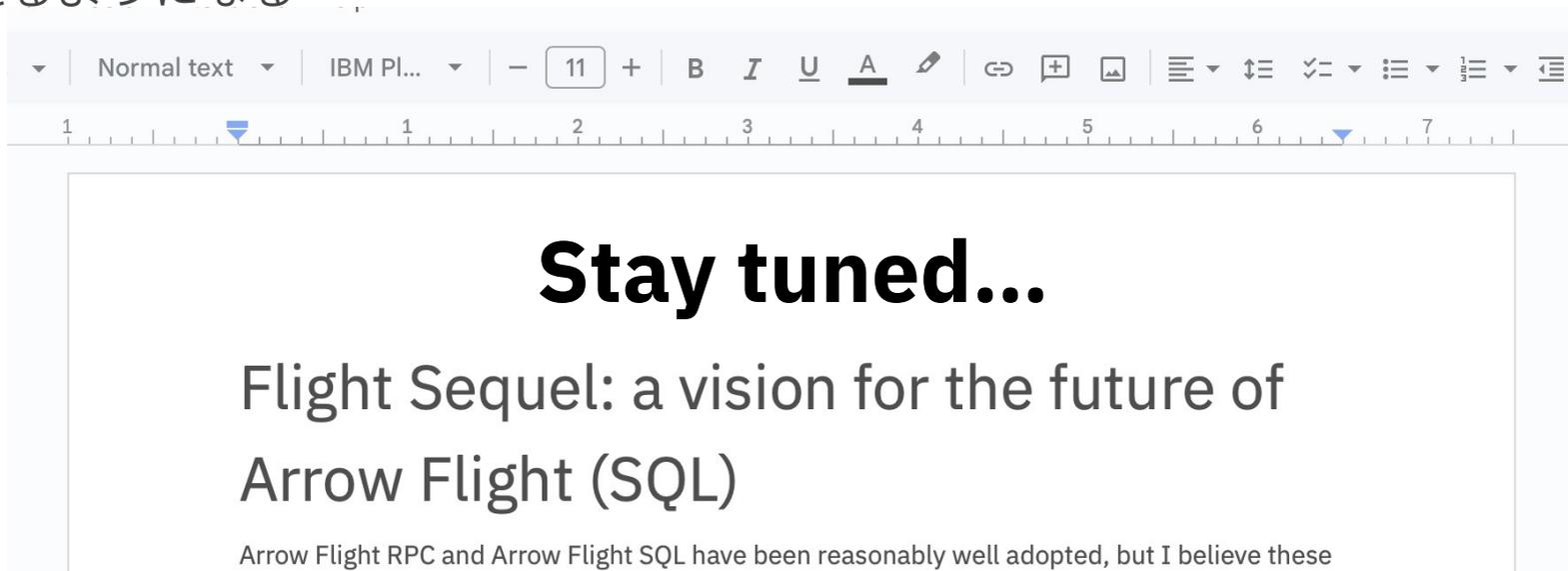
問題を解決すれば、上級ユーザーのために
より良いFlightにできる！

What can we do?

どうすればいいのでしょうか？

Goal: make it easy to use Arrow + gRPC + Protobuf without **having** to use wrappers like Flight

目的：Flightみたいなラッパーを使わなくてもArrowとgRPCとProtobufを簡単に使えるようになる



The image shows a presentation slide with a light blue background. At the top, there is a navigation bar with various icons and a font size of 11. The main content of the slide is centered and reads:

Stay tuned...

Flight Sequel: a vision for the future of Arrow Flight (SQL)

Arrow Flight RPC and Arrow Flight SQL have been reasonably well adopted, but I believe these

Thanks for listening!
Questions?