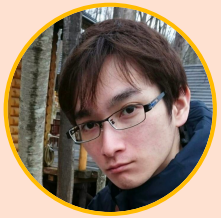
BigQuery で実現する ユーザーの傾向に合わせた 不動産物件レコメンドシステム

菊地 慧

LIFULL HOME'S 事業本部
プロダクトエンジニアリング部
Tech Lead



LIFULL

人々の暮らしや人生を満たすサービスを作り続ける。

あらゆる人が安心と喜びをもって未来へと進んでいくためのサポートをしたい。
そんな想いがLIFULL HOME'Sをはじめとした、暮らし全般に関わるさまざまな情報サービスの開発へと広がっていきました。

LIFULL HOME'S

LIFULL HOME’S アプリ

住まいの物件検索から、
住み替え完了までをサポート！

プラットフォーム
Android、iOS

アプリ独自の機能

- やることリスト
  - 引越し完了までの ToDo も管理
- かざして検索
  - 良さそうな物件や気になるエリアの街並みから検索

かざして検索

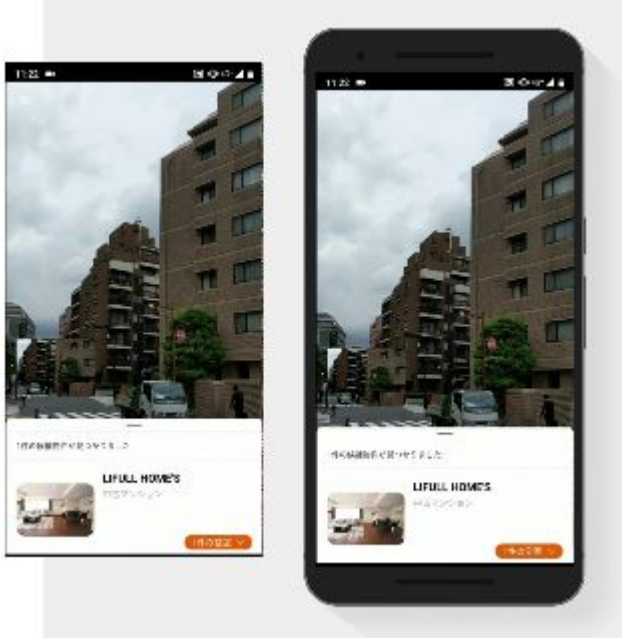
直感的に住まい探しを始められる

例えば

- 散歩中、良さそうなマンションから
- 気になるエリアの街並みから
- ここに住んだら・・・というふとした思いつきから

体験の流れ

1. 街並みなどにカメラをかざす
2. カメラ映像から「建物」を検知し「領域を判定」
3. 検知した領域を選択する
4. 該当する物件情報があれば表示。
   なければ方位などからカメラに映る可能性がある周辺の物件情報を表示



不動産レコメンドシステム

今回のお話

- Google Cloud Platform でレコメンドを作るための選択肢
- BigQuery の優秀さ
- アプリと Google Cloud Platform の相性
- 時間と費用

不動産レコメンドシステム

レコメンド内容

お気に入り傾向が似ているほかのユーザーがお気に入りに入れている物件をレコメンドする

想定対象ユーザー

- 自分にあった検索条件を見つけられていない
  - 住まい探しを始めたばかり
  - 条件を絞り込めていない
  - 条件を絞り込みすぎている

開発の経緯

きっかけ

ユーザーに対してパーソナライズされた機能を何か作れないか？
例えば、条件を絞り込めていないユーザーに物件をレコメンドとか？

- 既存のレコメンドシステム
  - ユーザーの検索履歴などから検索条件でレコメンドしているものが多い
  - 純粋に物件に対する好みなどのものは提供されていない
- ECでよく見る「よく一緒に購入されている商品」や「この商品を買った人はこんな商品も買っています」みたいなこと？
  - 不動産だと「よく一緒に問合せされている物件」や「この物件をお気に入り（もしくは見た）人はこんな物件もお気に入り（見た）しています」になる？

開発する上での課題

1. **学習データの収集**
   - レコメンドをするために必要となる「ユーザーが物件をお気に入りに追加した」というデータをどう収集するか？

2. **レコメンドエンジンの構築**
   - どうやってレコメンドエンジンを構築するか？

3. **アプリからの利用方法**
   - ユーザー毎に生成されたレコメンドをどのようにしてアプリ側で利用するのか？

学習データの収集

01

学習データの収集方法

課題

レコメンドを生成するためのデータセット**「ユーザーが物件をお気に入りに追加した」**というデータをどう収集するか？

方法

- API 経由でデータを収集する
  - データの収集をするためには費用がかかる
  - 任意のタイミング、構造でデータの取得ができる
  - 任意の場所にデータを保存できる

- Firebase Analytics でデータを収集する
  - 無料でデータの収集が可能
  - **「ユーザーが物件をお気に入りに追加した」**というイベントを取得済み
  - 何らかの方法で外部にデータをエクスポートする必要がある

学習データの収集方法

課題

レコメンドを生成するためのデータセット**「ユーザーが物件をお気に入りに追加した」**というデータをどう収集するか？

方法

- API 経由でデータを収集する
  - データの収集をするためには費用がかかる
  - 任意のタイミング、構造でデータの取得ができる
  - 任意の場所にデータを保存できる

- **Firebase Analytics でデータを収集する**
  - 無料でデータの収集が可能
  - **「ユーザーが物件をお気に入りに追加した」**というイベントを取得済み
  - 何らかの方法で外部にデータをエクスポートする必要がある

Firebase Analytics イベント

イベント名

add_to_wishlist

計測タイミング

ユーザーがある物件をお気に入りに追加する際に計測

パラメータ

- お気に入りに追加した物件の ID
- イベント発生時刻

※ Firebase Analytics にユーザー ID を自社 DB のユーザー ID を設定している

Firebase Analytics イベントのエクスポート

課題

Firebase Analytics で計測したイベントを学習データとして使用するためには、そのままでは使えないため、外部にデータを出力する必要がある

方法

- Cloud Functions
  - Firebase 向け Google アナリティクストリガー
- Firebase BigQuery Export

Firebase 向け Google アナリティクストリガーとは

Cloud Functions により Google アナリティクスを拡張することができる。
イベントごとに関連するすべてのパラメータとユーザープロパティへのアクセスができる。

Cloud Functions の呼び出し

ユーザーアクティビティでコンバージョンイベントが生成されるたびにトリガーされる

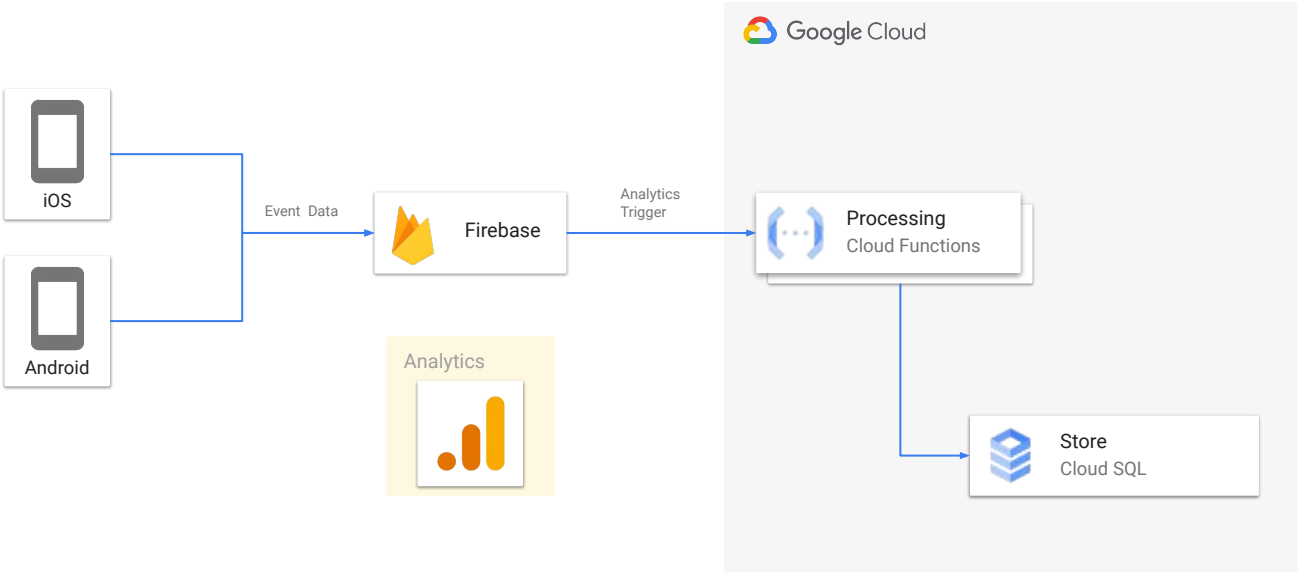
サポートされているイベント

コンバージョンイベントのみ

※ Firebase コンソールの [アナリティクス] > [イベント] から指定する

Architecture
Cloud Functions Google Analytics For Firebase Triggers

Architecture: Mobile -> Firebase Analytics -> Cloud Functions -> Cloud SQL



Firebase 向け Google アナリティクストリガー

メリット

- 任意のストレージに出力することができる
  - BigQuery、Cloud SQL、Firestore など
- イベントの発生タイミングで出力することができる
  - リアルタイムなレコメンドにも対応することができる

デメリット

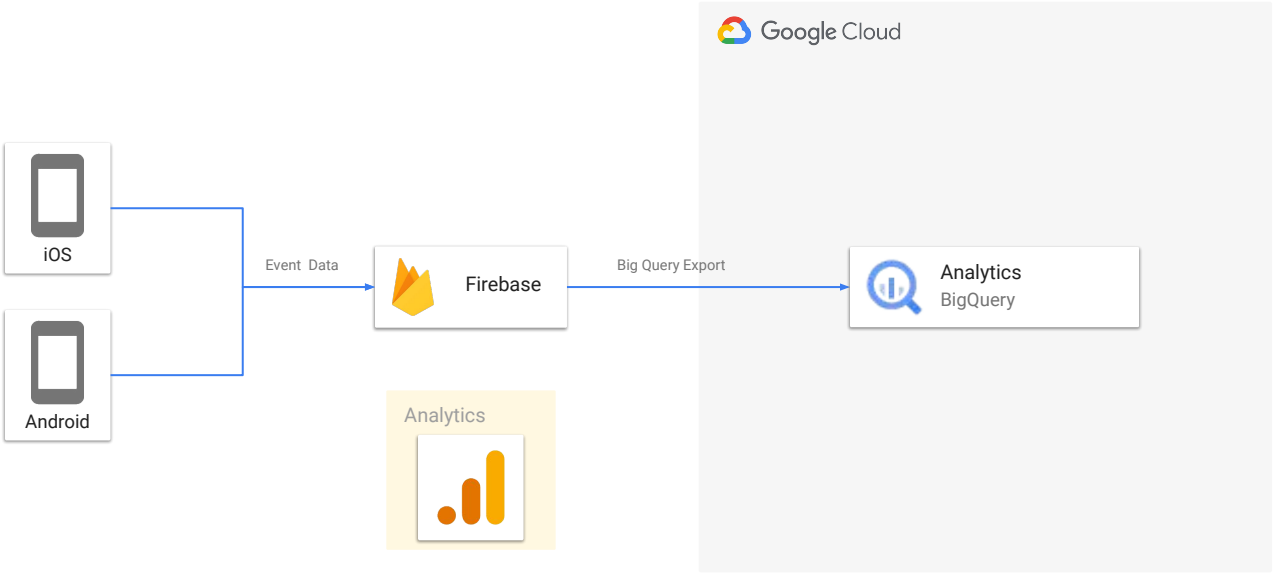
- Cloud Functions の起動に制限がある
  - Firebase Analytics で該当イベントをコンバージョン指定する必要がある
  - コンバージョン指定可能なイベント数には制限がある
- Cloud Functions 及びストレージの開発・運用コストが発生する

Firebase BigQuery Export とは

Firebase Analytics のサンプリングしていないイベントすべてを BigQuery にエクスポートすることができる

Architecture
Firebase BigQuery Export

Architecture: Mobile -> Firebase Analytics -> Big Query



Firebase BigQuery Export

メリット

- Firebase コンソールでプロジェクトをリンクさせるだけで利用が可能
- Firebase Analytics に対して BigQuery のクエリによる分析などが行える
- イベントのすべてのデータがエクスポートされている
- Google Analytics 4 Propery を利用していると、エクスポート頻度を指定可能（リアルタイムもしくは一定間隔）

デメリット

- BigQuery の利用料金がかかる

学習データの収集方法

ユーザーが物件をお気に入りに追加した というデータの収集方法

ユーザーの行動

- Firebase Analytics でお気に入り追加イベントを記録する
  - 無料でイベントを記録できる
  - レコメンドだけでなく他の Firebase の機能にも利用が可能

データの外部への出力

- **Firebase BigQuery Export**
  - データを BigQuery にためることで、クエリによる分析なども行える
  - Google Cloud Platform の各製品と組み合わせやすい

レコメンド
エンジンの構築

02

レコメンドエンジンの構築について

学習データ

学習データは Firebase Analytics 経由で BigQuery に蓄積することにした

課題

どのような手段で BigQuery にある学習データを利用して機械学習を行うか？

検討した方法

- Recommendations AI
- BigQuery ML
- DataProc
- TensorFlow + AI Platform

Recommendations AI とは

Google が提供する EC 向けのレコメンドシステム。一人ひとりの趣味や好みに合わせてパーソナライズされたおすすめ情報をあらゆるタッチポイントを通じて提供できる

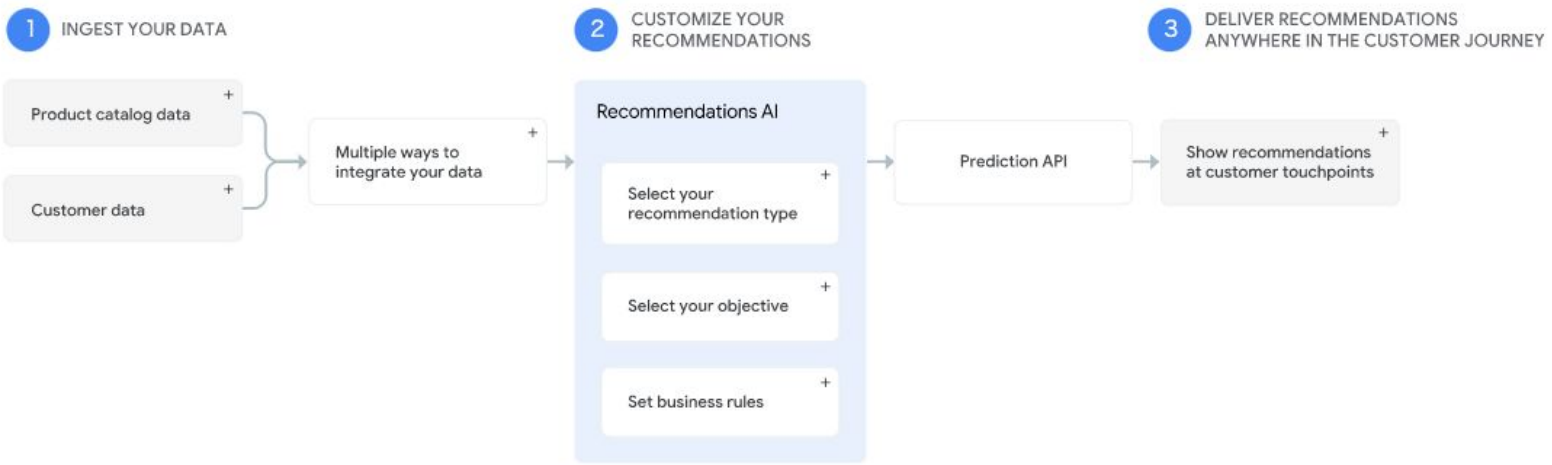
プロダクトカタログ（商品）とユーザーのイベントを元にして、レコメンドを行う

特徴

- フルマネージドサービス
- GUI ベースで操作が行える
- シナリオに合わせてバイアスや季節的な変動を補正できる
- ビジネスルールを適用することができる

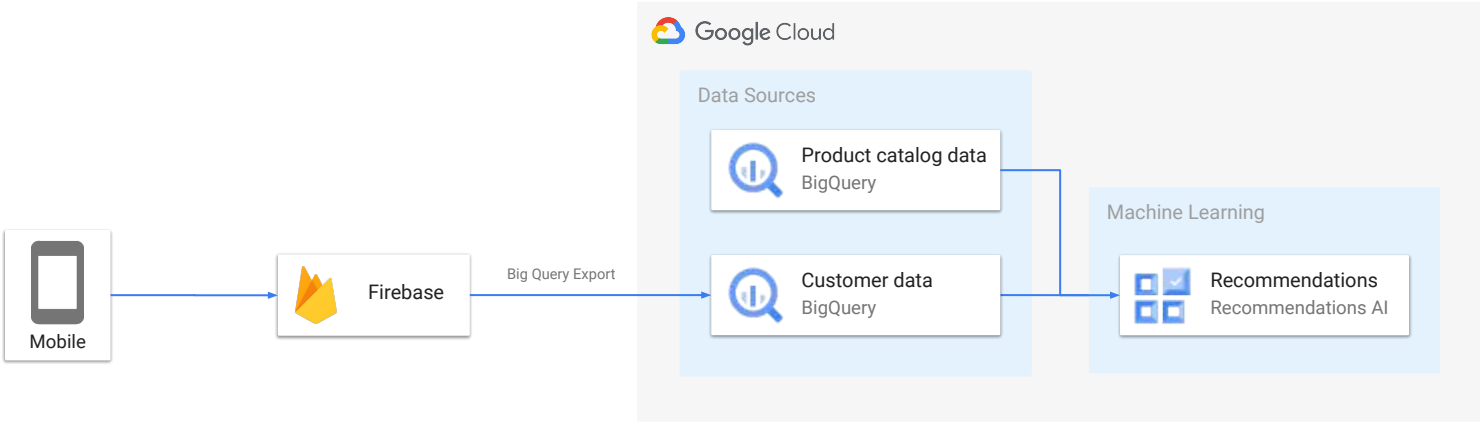
Recommendations AI の仕組み

仕組み



Architecture
Recommendations AI

Architecture: Mobile -> Firebase Analytics -> Big Query -> Recommendations AI



Recommendations AI

メリット

- EC 向けに非常に特化している
- コンソールから作業をするだけでレコメンドが可能

デメリット

- 事前にカタログ情報を用意する必要がある
  - リアルタイムに更新される物件をカタログとして用意するのは難しい
- ウェブサイトでの利用 が想定されている

今回のケースには合わないため断念

BigQuery ML

BigQuery で標準 SQL クエリを使用して、機械学習モデルを作成し実行できる

- 既存の SQL ツールやスキルが活用できる
  - python や Java でプログラミングする必要がない
  - 標準 SQL により、BigQuery でトレーニングができる
- BigQuery に格納されているデータをトレーニングデータとして扱える
  - データウェアハウスからエクスポートする必要がない
- サポートされているモデルの種類が多い
  - BigQuery にあるデータから様々な ML モデルを作成できる
  - 作成されたモデルは BigQuery データセットに格納される

サポートされるモデル

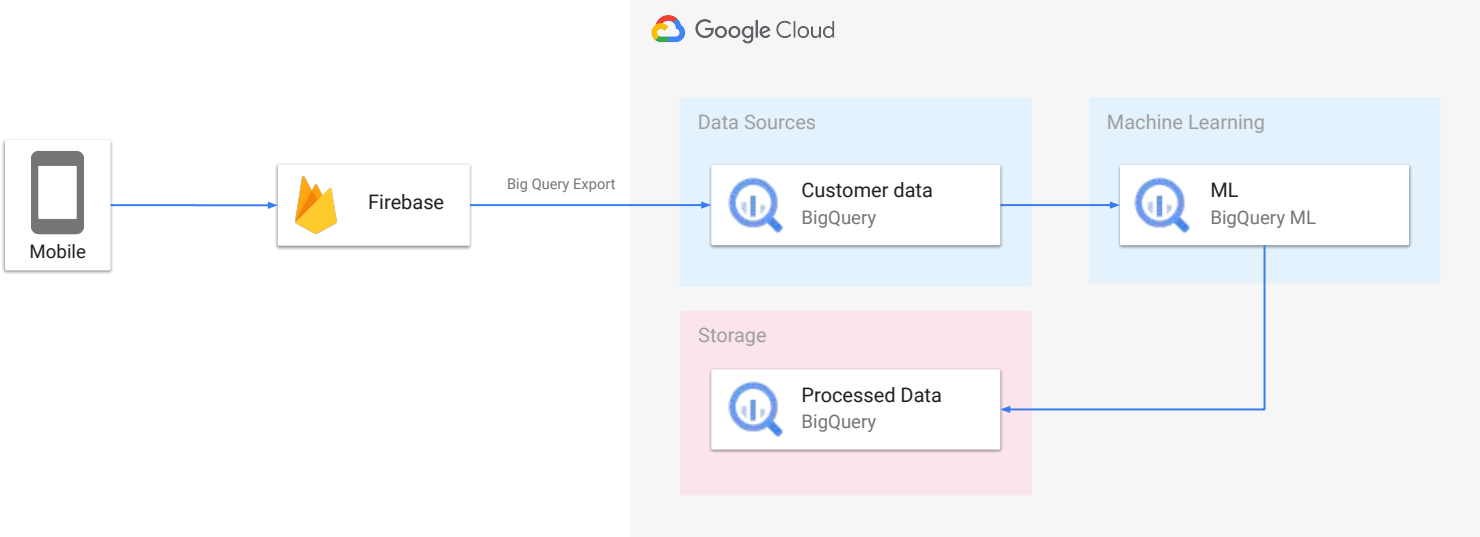
- 線形回帰（予測）
- 2 項ロジスティック回帰（分類）
- 多項ロジスティック回帰（分類）
- K 平均法クラスタリング（データセグメンテーション）
- 行列分解（商品のレコメンデーションシステムの作成）
- 時系列（時系列予測）
- ブーストツリー（XGBoost ベースの分類モデルと回帰モデルの作成）
- ディープニューラルネットワーク
- AutoML Tables
- TensorFlow モデルからのインポート

サポートされるモデル

- 線形回帰（予測）
- 2 項ロジスティック回帰（分類）
- 多項ロジスティック回帰（分類）
- K 平均法クラスタリング（データセグメンテーション）
- 行列分解（商品のレコメンデーションシステムの作成）
- 時系列（時系列予測）
- ブーストツリー（XGBoost ベースの分類モデルと回帰モデルの作成）
- ディープニューラルネットワーク
- AutoML Tables
- TensorFlow モデルからのインポート

Architecture
BigQuery ML

Architecture: Mobile -> Firebase Analytics -> Big Query ML



BigQuery ML

メリット

- 大量のデータセットから学習、予測ができる
- 標準 SQL クエリを用いることができる
- データセットが用意できれば素早く機械学習を導入できる

デメリット

- 行列分解モデルを利用するためのプラン
  - 定額制、もしくは Reservations 利用のみ。オンデマンドの場合は Flex Slots
- 検討段階では、データ処理料などの見積もりが難しいため、予算としてたてにくい

AI Platform

機械学習モデルを構築、デプロイ、管理できる One Platform

AI Platform Prediction

TensorFlow などのトレーニング済みモデルをホストし、クラウドから予測を取得できる。

- モデルのトレーニング
  - AI Platform でモデルのトレーニングができる
- 予測リクエスト
  - ホストしたトレーニング済みモデルに対して、リクエストを投げることができる
- モデルの管理
  - バージョン管理などが可能

AI Platform

メリット

- 高機能で今回の要件を超えるようなことも実現できる
- モデルのバージョン管理なども可能となるため、今後の拡張性が高い

デメリット

- AI Platform Training、AI Platform Prediction、Cloud Storage などの費用がかかる
- TensorFlow などの機械学習の前提知識が多少なりとも必要となる

下記の理由から断念

- 想定よりも費用が少しかかってしまいそう
- 今回の要件にはちょっとオーバースペック？

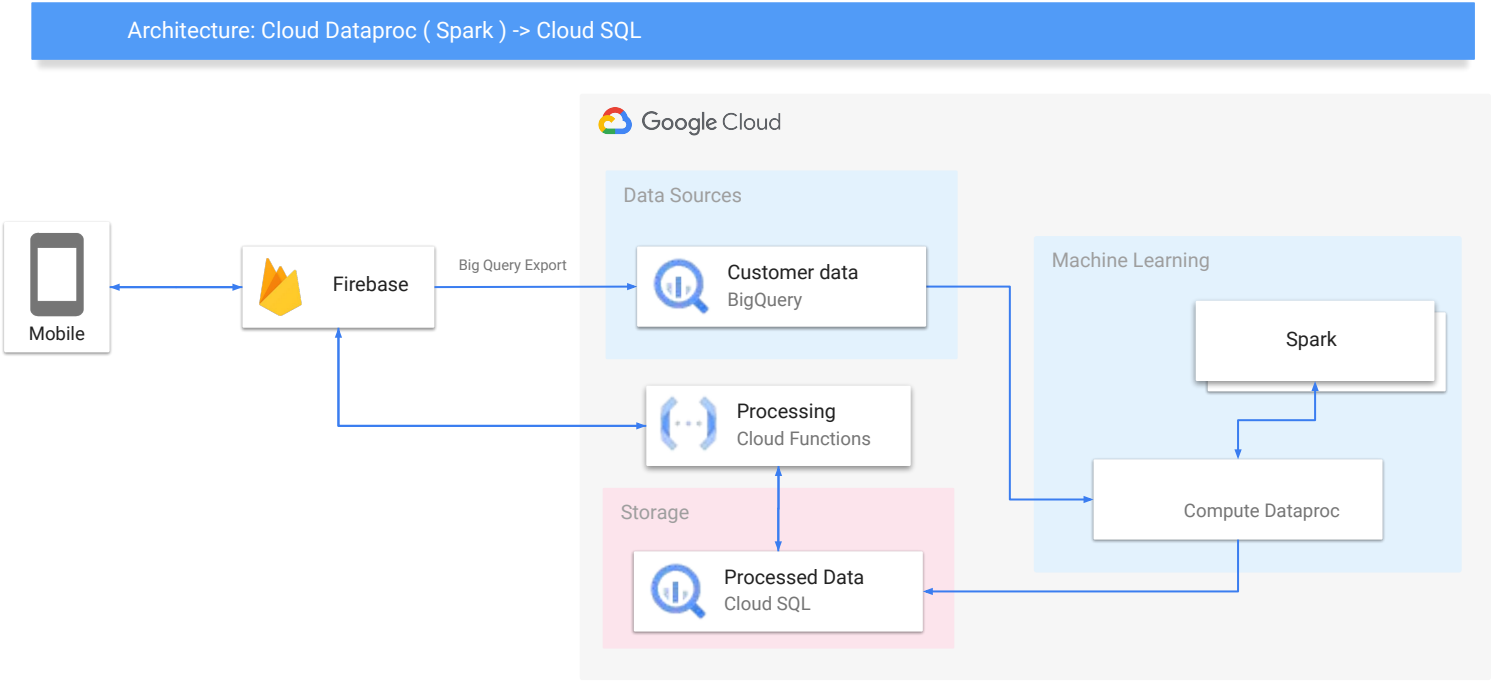
Cloud Dataproc

Dataproc はクラウド上で高速、簡単かつ安全にオープンソースデータと分析を実行することを可能にします。

主な機能

- 自動クラスタ管理
  - デプロイメント、ロギング、モニタリングが管理されるため、クラスタではなくデータに集中できます。
- OSS ジョブのコンテナ化
  - Dataproc 上に OSS ジョブ（Apache Spark など）を構築すると、Kubernetes を使ってそれらを素早くコンテナ化して、GKE クラスタがある場所ならどこでもデプロイできます
- エンタープライズセキュリティ

Architecture
Cloud Dataproc (Spark)



Cloud Dataproc

メリット

- App engine が常に立ち上がっている
  - リアルタイムレコメンドに向いている

デメリット

- App engine（Cloud Dataproc）の費用
  - 継続的に費用がかかる
- Cloud SQL の費用
- Spark の構築や運用で手間がかかる

レコメンドエンジンの構築方法

- Recommendations AI
  - EC 向けで今回の用途には・・・
- BigQuery ML
  - 理想的だが予算が・・・
- AI Platform
  - 今回の要件以上のことが実現できるが費用感が少し合わない
- Cloud Dataproc
  - 今回の要件にはちょうどいいが構築・運用に少し手間がかかる

レコメンドエンジンの構築方法

- Recommendations AI
  - EC 向けで今回の用途には・・・
- BigQuery ML
  - 理想的だが予算が・・・
- AI Platform
  - 今回の要件以上のことが実現できるが費用感が少し合わない
- Cloud Dataproc
  - 今回の要件にはちょうどいいが構築・運用に少し手間がかかる
- BigQuery
  - 協調フィルタリングをクエリだけで実現できる

BigQuery

BigQuery は Google のインフラの処理能力を活用して、SQL クエリを超高速に実行できる

クエリ内容

過去7日間におけるユーザー毎のお気に入りに追加した物件を抽出し、ユーザー毎に似た傾向を持つユーザーがお気に入りに追加した物件を最大 3件まで抽出する

懸念

- 超高速と言っても処理時間がどのくらいかかるか？
- データ量がどのくらいになるか？
- 最終的な費用はどの程度になるか？

BigQuery クエリ

過去 7 日間のイベントデータを抽出する

```
WITH today AS (
  SELECT
    event_date, user_pseudo_id, event_timestamp, event_params, user_id,
  FROM `{dataset}.events_intraday_{YYYYMMDD}`
)
, base AS (
  SELECT *
  FROM (
    SELECT
      event_date, user_pseudo_id, event_timestamp, event_params, user_id,
    FROM `{dataset}.events_20*`
    WHERE PARSE_DATE('%Y%m%d', CONCAT('20', _TABLE_SUFFIX)) > DATE_SUB(CURRENT_DATE, INTERVAL 7 day)
  )
  UNION ALL (SELECT * FROM today)
)
```

BigQuery クエリ

過去 7 日間のイベントデータからユーザー毎に物件 ID を抽出する

```
, wish AS (
  SELECT
      PARSE_DATE('%Y%m%d', event_date) AS event_date,
      user_pseudo_id AS user_id, event_timestamp AS et,
      (SELECT value.string_value FROM UNNEST(event_params) WHERE key = 'pkey') AS pkey
  FROM base
  WHERE (SELECT value.string_value FROM UNNEST(event_params) WHERE key = 'pkey') IS NOT NULL
)
```

BigQuery クエリ

過去 7 日間のイベントデータからユーザー毎に物件 ID のリストを作る

```
, user AS (
  SELECT
    user_id, ARRAY_AGG(DISTINCT pkey) AS pkeys,
  FROM wish
  GROUP BY user_id
)
, target_user AS (
  SELECT
    user_id AS target_id,
    ARRAY_AGG(DISTINCT pkey) AS target_pkeys,
  FROM wish
  WHERE user_id IN (SELECT user_pseudo_id FROM today)
  GROUP BY user_id
)
```

BigQuery クエリ

今日イベントを計測したユーザーに対して、過去 7 日間にイベントを計測したユーザーの組み合わせを抽出する

```
, coop AS (
  SELECT
    target_id,
    user_id,
    sim(target_pkeys, pkeys) AS simulality
  FROM target_user
  CROSS JOIN user
)
```

関数：ユーザー間の物件 ID の類似度を生成して返す

```
CREATE TEMP FUNCTION sim(x ANY TYPE, y ANY TYPE) AS ((
  SELECT
    SUM((SELECT 1 FROM UNNEST(y) AS py WHERE px = py))
  FROM UNNEST(x) AS px
));
```

BigQuery クエリ

似ているユーザーのリストを作成する

```
, sim_user AS (
  SELECT
    target_id, ARRAY_AGG(STRUCT(user_id, simulality) ORDER BY simulality DESC) AS users
  FROM coop
  WHERE TRUE
    AND simulality IS NOT NULL
    AND target_id != user_id
  GROUP BY target_id
)
```

BigQuery クエリ

レコメンドを行いたいユーザーに対して、重複している物件の数を数えてリスト化する

```
, joint AS (
  SELECT
    target_id, array_count(ARRAY_AGG(pkey)) AS recomend_pkyes
  FROM sim_user, UNNEST(users)
  LEFT JOIN target_user USING (target_id)
  LEFT JOIN user USING (user_id), UNNEST(pkeys) AS pkey
  WHERE pkey NOT IN UNNEST(target_pkeys)
  GROUP BY target_id
)
```

関数：ユーザー同士の似ている同じ物件の数を数えて、上位3件を抽出する

```
CREATE TEMP FUNCTION array_count(pkeys ARRAY<STRING>) AS (ARRAY(
  SELECT AS STRUCT
    pkey, COUNT(*) AS cnt
  FROM UNNEST(pkeys) AS pkey
  GROUP BY pkey
  ORDER BY cnt DESC
  LIMIT 3 # レコメンド数
));
```

BigQuery クエリ

Firebase Analytics で付与した user_id を持っているユーザーのみを対象ユーザーとする

```
SELECT
  user_id, joint.*
FROM joint
INNER JOIN (
  SELECT DISTINCT
    user_id, user_pseudo_id
  FROM today
  WHERE user_id IS NOT NULL
)
ON user_pseudo_id = target_id
```

BigQuery の呼び出し方法

課題

レコメンドエンジンとして使う場合には定期的に実行する必要がある。また ML モデルは作成されないため、クエリの結果をどこかに格納しておく必要がある。

検討案

	クエリの定期実行	レコメンド結果の格納
BigQuery	クエリのスケジュール	BigQuery
Cloud Functions	Cloud Scheuler	任意のストレージ
Cloud Composer		任意のストレージ

BigQuery の呼び出し方法

課題

レコメンドエンジンとして使う場合には定期的に実行する必要がある。また ML モデルは作成されないため、クエリの結果をどこかに格納しておく必要がある。

検討案

	クエリの定期実行	レコメンド結果の格納
BigQuery	クエリのスケジュール	BigQuery
Cloud Functions	**Cloud Scheuler**	**任意のストレージ**
Cloud Composer		任意のストレージ

Cloud Functions による BigQuery 定期実行

構成

- Cloud Scheduler
  - 定期的に Cloud Pub/Sub の Topic に対してメッセージを発行するジョブを設定
- Cloud Pub/Sub
  - メッセージをキューイングする
- Cloud Functions
  - Topic の subscriber として設定することで、Pub/Sub メッセージをトリガーに起動

費用

毎日一回だけ定期的に実行される Cloud Functions 内で BigQuery による協調フィルタリングまで試したところ月数百円（データ量により変動）

Cloud Functions による BigQuery 定期実行

課題

BigQuery のクエリ発行とデータストレージへの保存する必要があるが、 Cloud Functions の起動時間内で収まるか？

検証結果

- クエリ自体は数秒で処理が完了した
- データストレージへの保存も1分を超えるようなデータ量にならないため採用

※ Cloud Functions で実行時間が足りなければ Cloud Run も検討しました

データストレージについて

課題

BigQuery のクエリ結果の格納先をどうするか？

検討案

- BigQuery
- Cloud Firestore

データストレージについて

課題

BigQuery のクエリ結果の格納先をどうするか？

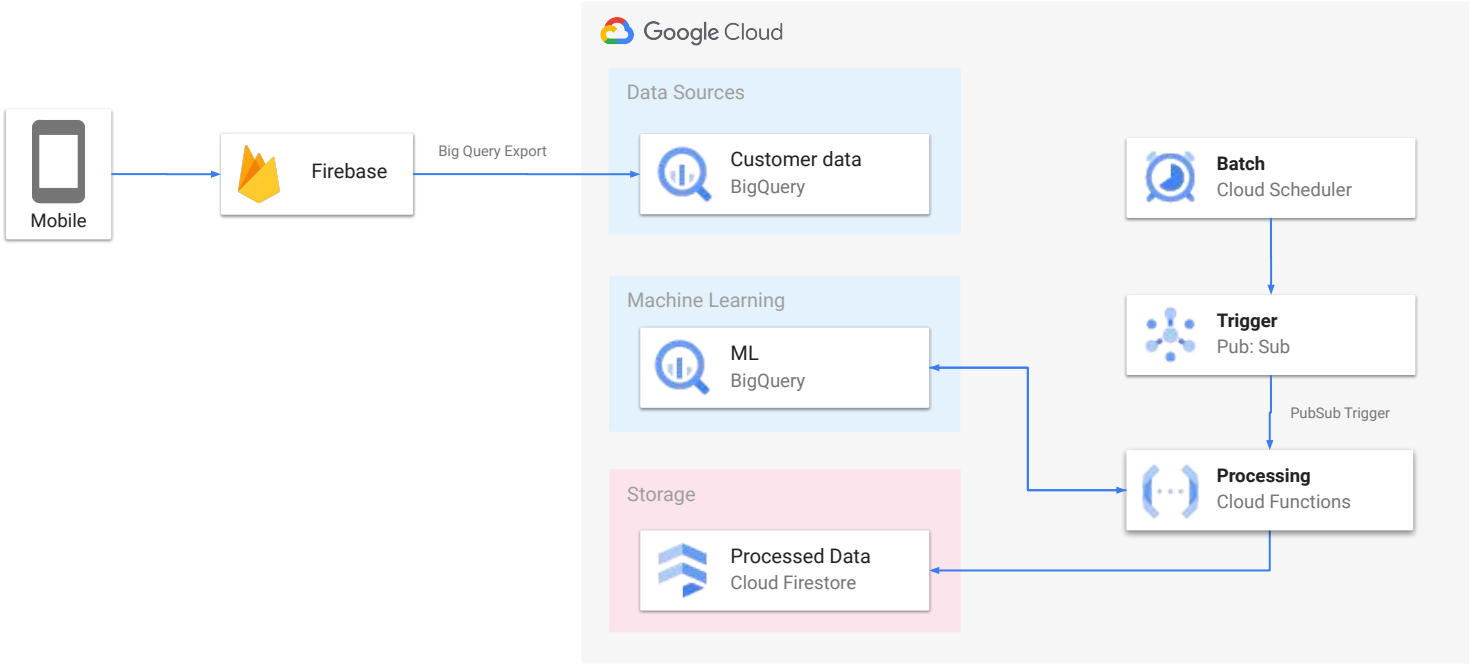
検討案

- BigQuery
  - ストレージ料金は安い
  - リアルタイムレコメンドでも対応が可能
- Cloud Firestore
  - アプリでは既に導入済みで、ユーザー単位でドキュメントが存在している
  - バッチレコメンドを検討しているが、リアルタイムレコメンドになったら厳しい

アプリからの利用方法を検討した上で決定することに

Architecture
Cloud Scheduler -> Cloud Firestore

Architecture: Cloud Scheduler -> Cloud Pub: Sub -> Cloud Functions -> Cloud Firestore



アプリからの利用方法

03

アプリからの利用方法

課題

データストアとして Cloud Firestore を選択したがアプリからのアクセス方法をどうするか？

検討案

- Cloud Functions
  - API として Cloud Functions を経由して、Cloud Firestore にアクセスする
- Cloud Firestore
  - アプリからクライアント SDK 経由で直接アクセスする

Cloud Functions 経由の場合

関数の呼び出し方法

HTTP リクエスト、クライアント SDK

メリット

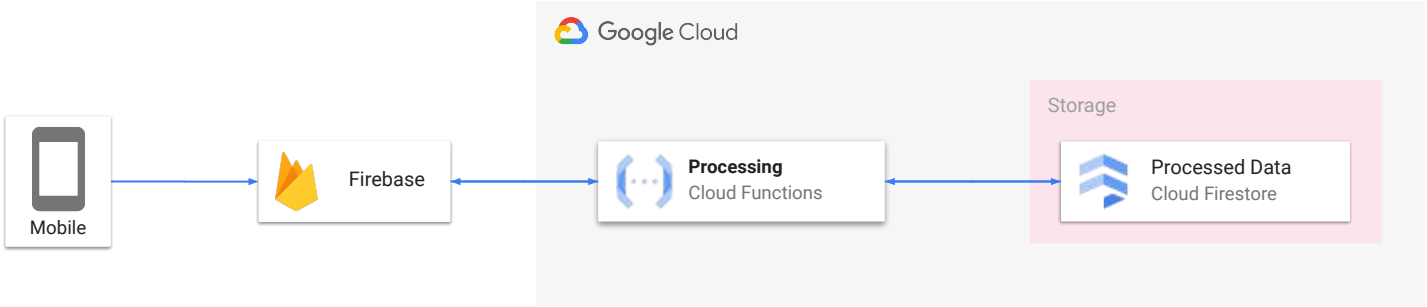
- 認証には Firebase Auth が利用できる
- クライアント SDK 経由を利用することで手軽に利用できる
- データストレージが Cloud Firestore 以外に変更することもできる

デメリット

- Cloud Firestore のほかに Cloud Functions の費用がかかる
- Cloud Functions 自体の開発や運用などが必要となる

Architecture
Cloud Functions -> Cloud Firestore

Architecture: Mobile -> Firebase Cloud Functions -> Cloud Firestore



Cloud Firestore クライアント SDK 経由の場合

呼び出し方法

クライアント SDK 経由

メリット

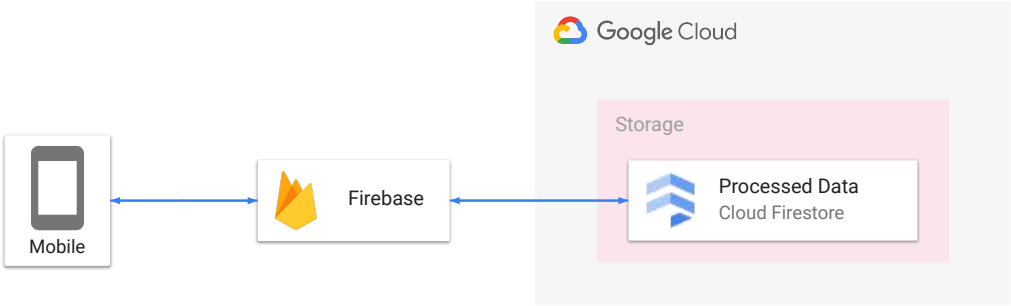
- アプリからクライアント SDK 経由で直接アクセスできる
- セキュリティについてもルールベースで定義ができる
- 全体の構成がシンプルになる

デメリット

- ストレージ保存料金については BigQuery より少し高価

Architecture
Firestore

Architecture: Mobile -> Cloud Firestore



検討結果

Cloud Functions

- 認証には Firebase Auth が利用できる
- クライアント SDK 経由を利用することで手軽に利用できる
- データストレージが Cloud Firestore 以外に変更することもできる
- Cloud Firestore のほかに Cloud Functions の費用がかかる
- Cloud Functions 自体の開発や運用などが必要となる

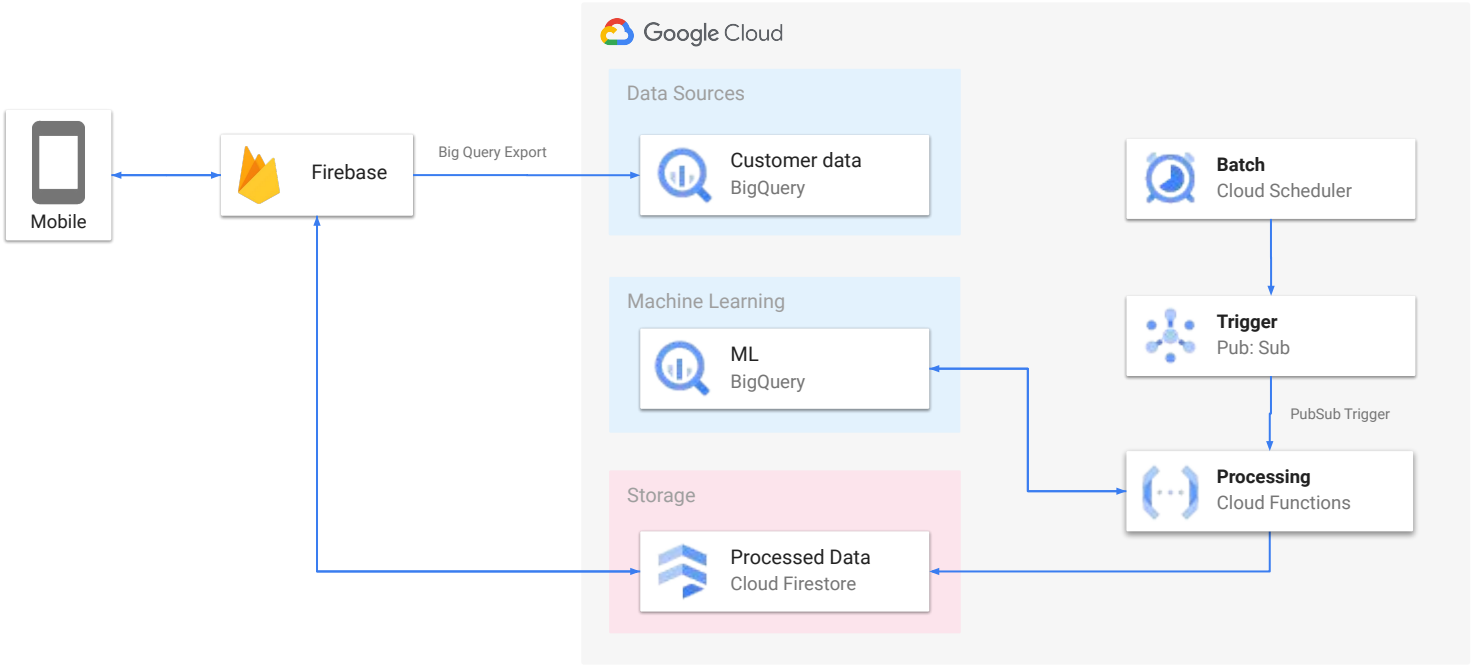
Cloud Firestore

- アプリからクライアント SDK 経由で直接アクセスできる
- セキュリティについてもルールベースで定義ができる
- 全体の構成がシンプルになる
- ストレージ保存料金については BigQuery より少し高価

まとめ

Architecture
不動産レコメンドシステム

Architecture: Mobile -> Firebase Analytics -> Big Query



まとめ

- Big Query は優秀
  - クエリだけでも現実的な速度で協調フィルタリングを行うことができる
  - BigQuery ML はすべての手間を自動化してくれる

- アプリと Google Cloud Platform の相性の良さ
  - Firebase を導入しているとより手軽に利用できる
  - 様々な機能を組み合わせることや、組み合わせられるものが多い

- 時間と費用
  - 試行錯誤することで運用コストを下げることはできる
  - Google Cloud Platform には手間をかけずにレコメンドシステムを構築する手段がたくさん存在している

Thank you