ファッション コーディネート アプリ「WEAR」における Vertex AI Vector Search を利用したレコメンド機能の開発・運用で得られたノウハウの紹介

Google Cloud Next Tokyo

岡本 侑都

ZOZO, Inc.

データ・AI システム本部

データシステム部　MLOps ブロック

目次

01. サービス紹介

ZOZOTOWN

https://zozo.jp/

- ファッションEC
- 1,600 以上のショップ、9,000 以上のブランドの取り扱い
- 常時 107 万点以上の商品アイテム数と毎日平均2,700 点以上の新着商品を掲載（2025 年 3 月末時点）
- ブランド古着のファッションゾーン「ZOZOUSED」や
  コスメ専門モール「ZOZOCOSME」、ラグジュアリー＆デザイナーズゾーン
  「ZOZOVILLA」を展開
- 即日配送サービス
- ギフトラッピング サービス
- ツケ払いなど

WEAR

by ZOZO

https://wear.jp/

- あなたの「似合う」が探せるファッションコーディネートアプリ
- 1,800 万ダウンロード突破、コーディネート投稿総数は1,400 万件以上（2025 年 3 月末時点）
- コーディネートや最新トレンド、メイクなど豊富なファッション情報をチェック
- AI を活用したファッションジャンル診断や、フルメイクをAR で試せる「WEAR お試しメイク」を提供
- コーディネート着用アイテムを公式サイトで購入可能
- WEAR 公認の人気ユーザーをWEARISTA と認定。モデル・タレント・デザイナー・インフルエンサーといった各界著名人も参加

02. コーディネート レコメンド機能の紹介

コーディネート詳細画面のレコメンド

関連枠におけるコーディネートの表示ロジックを
ルールベースから ML に置き換える

以下を満たすコーディネートをレコメンド

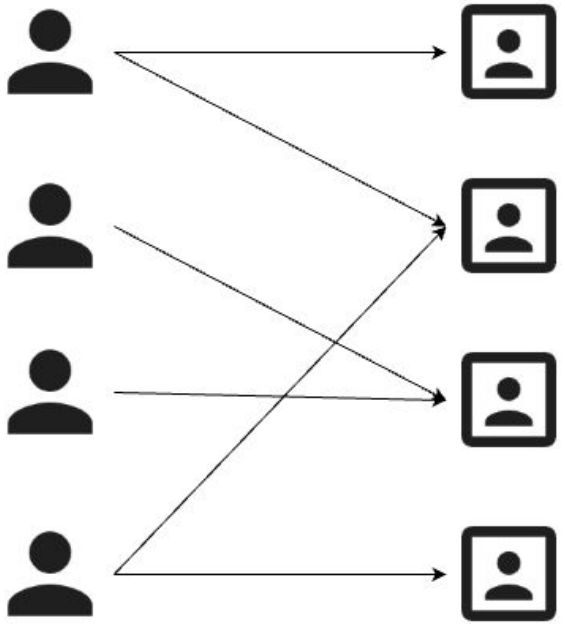
- 閲覧中のコーディネートに類似
- ユーザーが興味を持ちそう

レコメンド ロジック

既存のユーザー、ユーザー行動履歴、コーディネートのデータを使って GNN モデルを学習

- Node
  - ユーザー
  - コーディネート
- Edge
  - ユーザー行動履歴

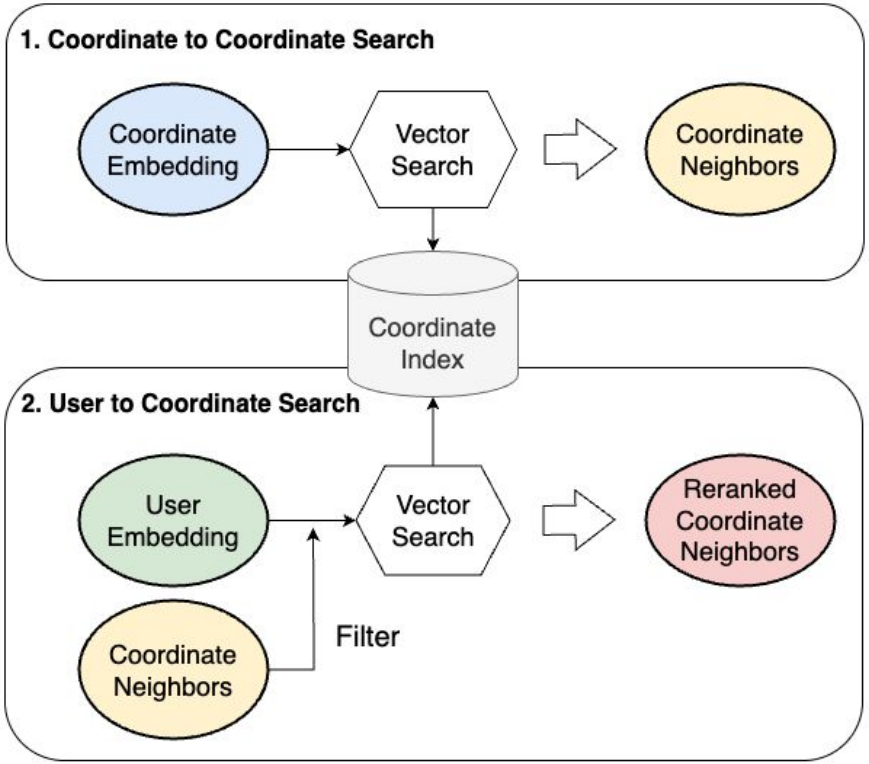
Node 自身の特徴量と Edge で紐づく周辺 Node の特徴量を集約し、Embedding を抽出



類似コーディネート & パーソナライズ

2 段階でベクトル検索を実行

1. コーディネート群の絞り込み
   - コーディネート同士で近傍探索
2. コーディネート群のリランキング
   - 近傍探索対象を 1 で取得したコーデ群に制限
   - ユーザーとコーディネートで近傍探索



レコメンドの導入効果

関連枠のコーディネートの閲覧数・クリック数ともに大幅に向上

- 関連枠における UU あたりのクリック数は**約 2 倍**
- 関連枠における UU あたりのインプレッション数は**約 1.6 倍**
- 関連枠に表示されたコーディネートの詳細画面への次遷移率は**約 1.3 倍**

03. Vertex AI Vector Search の選定理由

既存のベクトル検索機能の実装

従来はベクトル検索機能の多くを自前で実装していた

- ベクトル検索 API の実装
- ベクトル検索 Index
  - ビルド / 更新
  - デプロイ

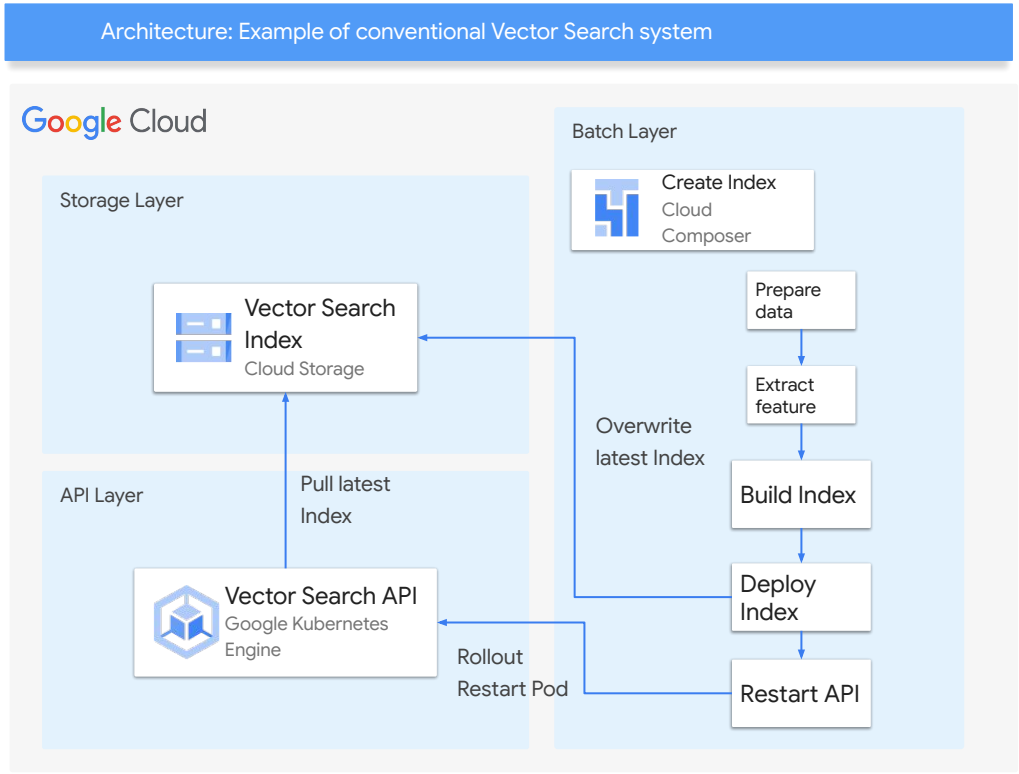
ex. ZOZOTOWN の類似アイテム検索（画像検索）機能

引用元 : https://techblog.zozo.com/entry/cloudnext19tokyo-imagesearch

既存のベクトル検索機能の構成例

Index は ベクトル検索 API で
オンメモリに保持

Index 更新の反映には ベクトル検索 API の再起動が必要



Vertex AI Vector Search の選定理由

マネージドサービス

- 実装が容易
- サーバレス

高速なクエリ実行

- ScaNN アルゴリズムにより大規模データでも高速

将来需要に備えた知見獲得

- マネージド サービスでのベクトル検索機能の開発・運用事例の獲得

Alloy DB for PostgreSQL との比較

マネージドサービスかつ ScaNN アルゴリズムも利用可能な点は共通

Vertex AI Vector Search に感じた優位性

（本番だけでなく）実験環境での利用しやすさ

- 環境構築が容易
  - DB 構築が不要
  - GCS のデータを元に Index をビルド可能
- Index の切り替えが容易
- Vertex AI Pipelines と合わせて利用しやすい

04. レコメンド システムのインフラ構成

ベクトル検索以外のサービス選定

Google Kubernetes Engine (GKE)

レコメンド API の稼働

- 他案件の API も含め、マルチテナント構成で運用

Vertex AI Pipelines

Embedding 抽出等のバッチ処理の実行

- マネージドかつサーバレスでインフラの運用・保守が不要
- Vertex AI Pipelines の CI / CD 、監視、スクリプトが
  まとまったテンプレートリポジトリを社内開発している

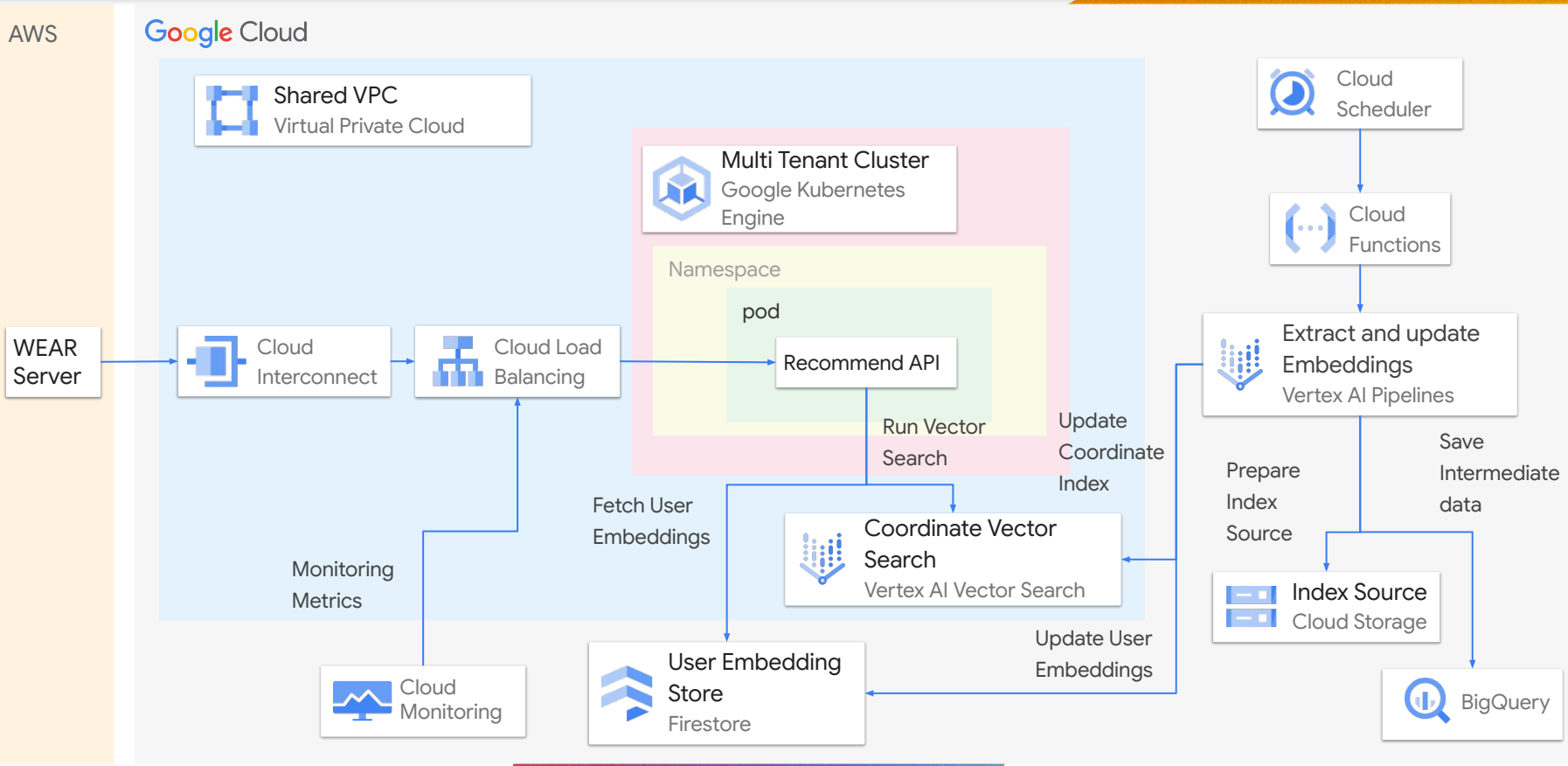
ユーザー Embedding の保存先 DB

レイテンシを抑えるため、ユーザー Embedding はパイプラインでオフラインに抽出し、DB に保存

Cloud Firestore

- インスタンス課金ではないため比較的安価
- クエリは軽量でパフォーマンス要件もそこまで高くない

Architecture: wear related coordinate recommendations



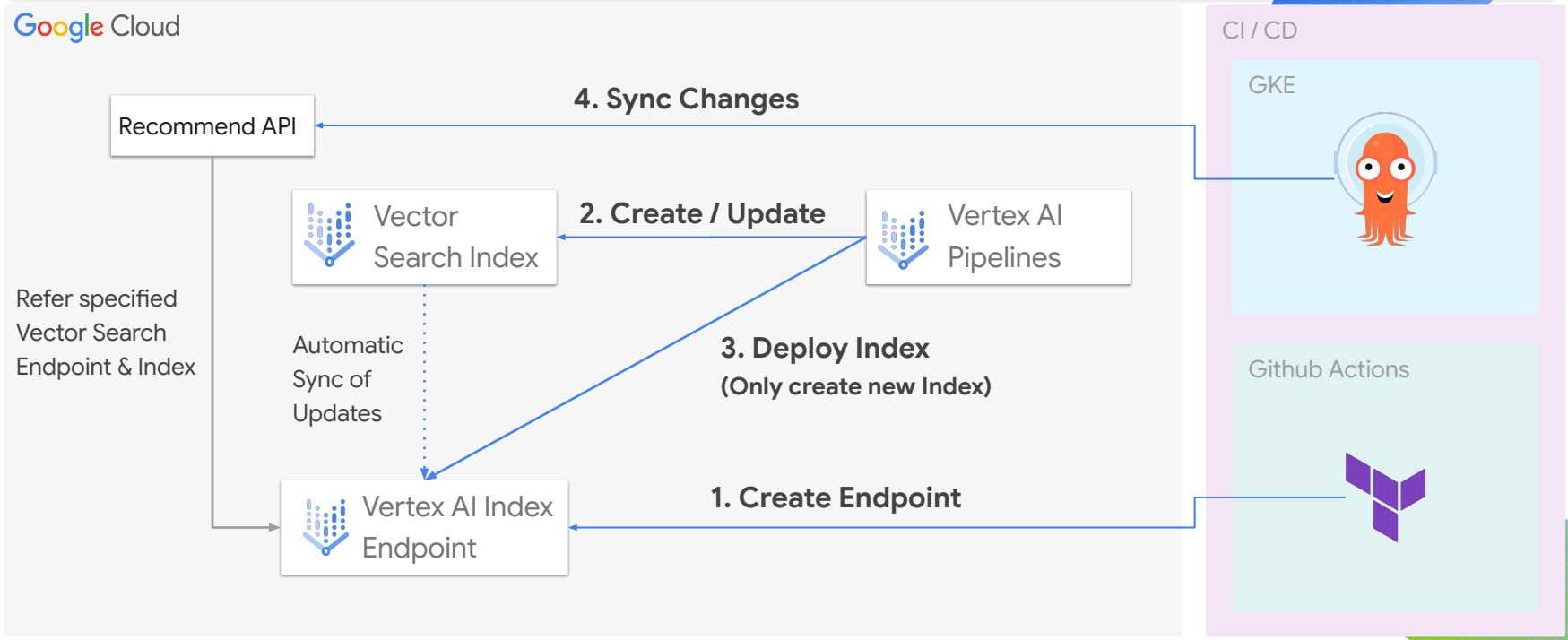
Vertex AI Vector Search のデプロイ

Index Endpoint の事前作成と Index Endpoint への Index のデプロイが必要

- **Index Endpoint 作成時の設定項目**
  - リージョン
  - ネットワーク設定
- **Index のデプロイ時の設定項目**
  - ビルド済み Index 名
  - Vertex AI Vector Search のマシンタイプ
  - 最小・最大レプリカ数
  - デプロイの timeout 値

Vertex AI Vector Search のリリースフロー

Architecture: Vertex AI Vector Search release flow



05. 実装で得た知見

Vertex AI Index Endpoint の作成

- Endpoint 作成時に VPC Peering 設定可能（Public IP の付与も可能）
- IP（gRPC）アドレスは Index デプロイ時に自動割り当て
  - デプロイ済みの Index 更新では IP は変わらない
  - IP の割り当て範囲の指定も可能
- Index と Index Endpoint のリージョンは一致する必要がある
  - Index ソースデータの GCS バケットと Index のリージョンも一致する必要がある
  - Vertex AI Pipelines のアーティファクトをソースデータにする場合、アーティファクト用のバケット・パイプライン実行のリージョンにも注意

ベクトル検索クエリの実行

- ベクトル検索時の入力パラメータには Embedding だけでなく、Index 内のベクトルの識別子である データポイント IDが利用可能
- Index レコードに restricts フィールドを持たせることで、ベクトル一致のフィルタリングが可能
  - restricts でサブセットを絞り込んでベクトル検索

クエリ実行のフィルタリング

クエリ実行時に allow list・deny list を指定することで、条件に一致するデータポイント ID を対象にベクトル検索可能

- restricts フィールドは複数付与可能
- numeric_restricts フィールドも利用可能
  - 数値の大小比較等も可能

```
[
  {
    "id": "2020",
    "embedding": [
      0.5,
      0.5,
      ...
    ],
    "restricts": [
      {
        "namespace": "snap_id",
        "allow": [
          "2020"
        ]
      }
    ]
  },
  ...
]
```

Index 構成パラメータの指定

Index 構成時に指定できるパラメータは精度・パフォーマンスに大きく影響する

パラメータの一覧は右のドキュメントに記載があるが、パフォーマンス影響については、次に説明するドキュメントに記述が分かれている

Vertex AI > Documentation

この情報は役に立ちましたか？

インデックス構成パラメータ

フィードバックを送信

類似度検索のインデックスを構成するには、次のフィールドを構成する必要があります。

インデックスの構成方法については、インデックス パラメータを構成するをご覧ください。

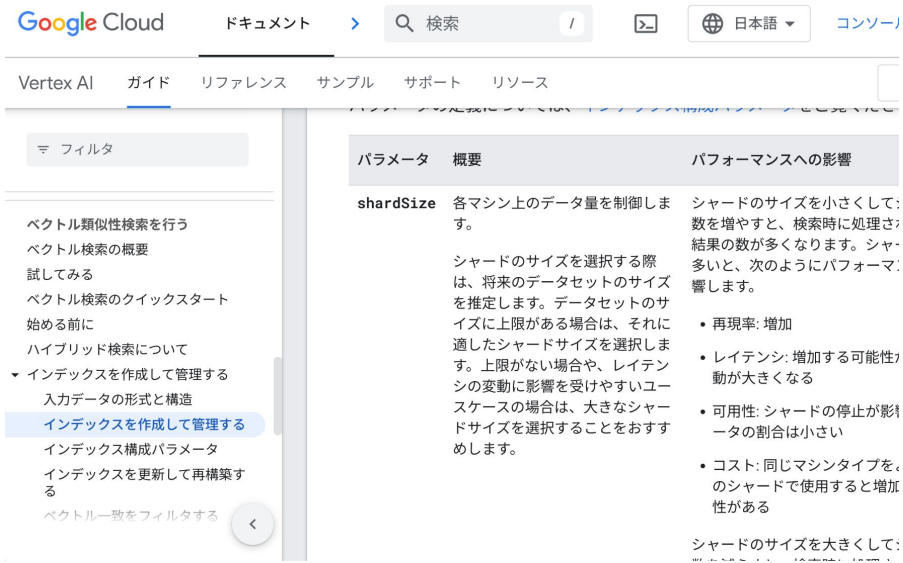
NearestNeighborSearch

フィールド	
contentsDeltaUri	string ベクトル検索 Index の内容を挿入、更新、削除できるようにします。文字列は、有効な Cloud Storage ディレクトリ パス（gs://BUCKET_NAME/PATH_TO_INDEX_DIR/ など）でなければなりません。 IndexService.UpdateIndex を呼び出す際にこのフィールドを設定した場合、同じ呼び出しで他の Index フィールドは更新できません。個々のデータファイルを構造化する方法をご覧ください。
isCompleteOverwrite	boolean IndexService.UpdateIndex を呼び出す際にこのフィールドを contentsDeltaUri とともに設定すると、Index の既存の内容が contentsDeltaUri のデータに置き換えられます。このフィールドを true に設定すると、インデックス全体が、指定した新しいメタデータ ファイルで完全に上書きされます。
config	NearestNeighborSearchConfig ベクトル検索 Index の構成。

引用元: https://cloud.google.com/vertex-ai/docs/vector-search/configuring-indexes?hl=ja

Index 構成パラメータ チューニングのヒント

再現率・レイテンシ・可用性・コストのトレードオフを記載したドキュメントがパラメーター一覧ページと別に用意されている



引用元: https://cloud.google.com/vertex-ai/docs/vector-search/create-manage-index?hl=ja
https://cloud.google.com/vertex-ai/docs/vector-search/query-index-vpc?hl=ja

Indexing 時に指定可能なパラメータ

特にクエリ実行時のレイテンシ影響が大きかったパラメータ

各リーフノードの埋め込み数

- **leafNodeEmbeddingCount**
  - 当初仮で 100 としていたところ 1s 以上かかっていた
  - 15,000 に変更後、99%tile 120ms 程度に向上
    - Embedding 総数 14,000,000

※ レイテンシは条件によって変わるためあくまで参考値です

クエリ実行時に指定可能なパラメータ

クエリ実行時に指定可能なパラメータ値の調整例

クエリが検索されるリーフノードのデフォルト割合

- fractionLeafNodesToSearch
  - コーデ → コーデの検索ではレイテンシ重視
  - ユーザー → コーデの検索では再現率重視

クエリで返す結果数

- setNeighborCount
  - コーデ → コーデの検索では母数を取るため多め
  - ユーザー → コーデの検索では必要分だけ

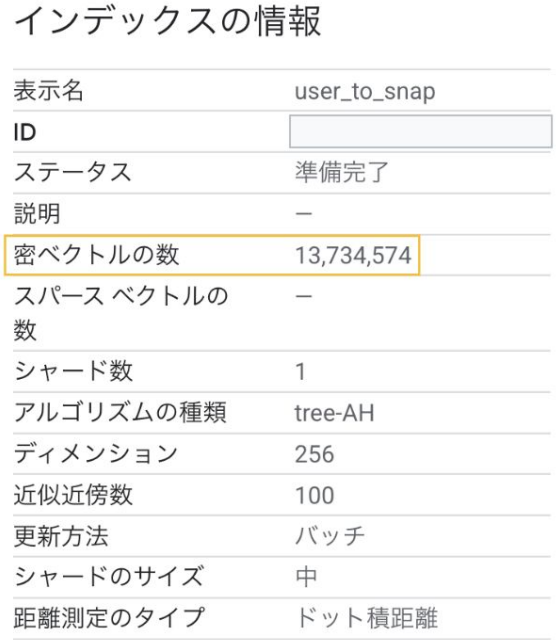
Index の更新

IndexService.UpdateIndex で既存のバッチ Index の内容を更新できる

- contentsDeltaUri
  - Index 作成時と同様にデータソースを用意してパスを指定
- isCompleteOverwrite
  - true にすると既存の Index レコードは全て失われるため注意

Index データポイント更新時の挙動

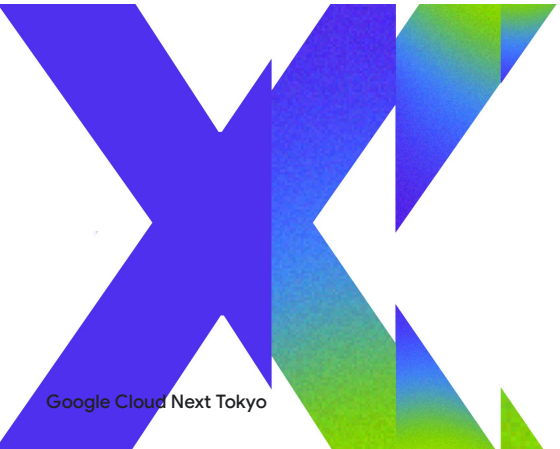
- データソースに含まれるデータポイントのうち、ID が重複する場合は新しいデータで上書きされる
  - 新規データポイントの場合は追加される
- Index の更新後は、Index が再構築される
  - 再構築中は増分 Index が作成される
    - 増分 Index により、一時的に密ベクトル数の表示値が多くなる場合がある
    - 再構築中にクエリ実行した場合、新規更新した embedding を使ってベクトル検索を実行する



Index データポイント更新の反映

デプロイ済み Index の更新は2つのフェーズで実行される

1. Index の更新
2. デプロイ済み Index の同期



デプロイ済み Index の更新が終わっていても同期が完了していない場合は、クエリ実行時に更新前の Index でベクトル検索される

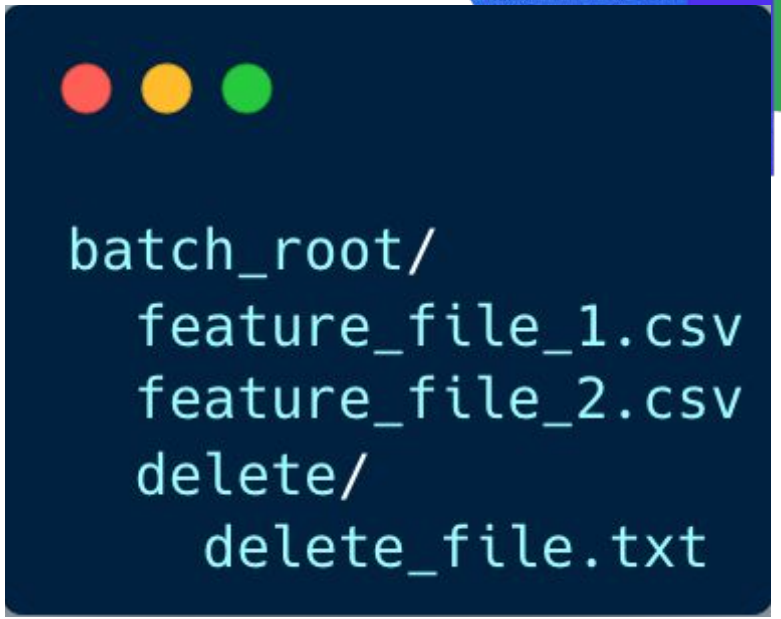
同期タイミングは明示されていないが、更新に関わらず数分おきに同期されている

Index データポイント削除時の注意点

更新と削除は一度にできないことに注意

削除対象のデータポイント ID をテキスト形式で delete ディレクトリ配下に保存

Index 更新と同様のメソッドを実行すると削除できる



ファイル構成の引用元：https://cloud.google.com/vertex-ai/docs/vector-search/setup/format-structure?hl=ja

06. 運用で得た知見

システムの安定性

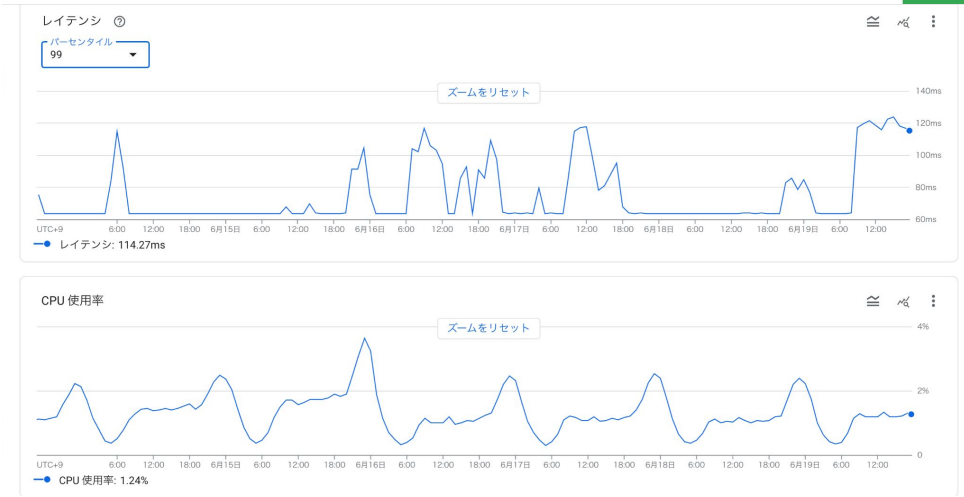
約 4 ヶ月本番運用し、かなり安定している

- 低レイテンシ
  - 99%tile レイテンシで 60~120ms 程度
    - ピーク帯 30rps
    - マシンタイプ e2-standard-16
    - シャード数 1
    - レプリカ数 2（冗長化のため 2 で構成）
- 高い安定感
  - リクエスト時の timeout 等のエラーもかなり少ない
    - 1~2 件 / week 程度

メトリクス確認がしやすい

メトリクス ダッシュボードがデフォルトで提供されており、メトリクスを確認しやすい

- ノード数
- シャード数
- 秒間クエリ数
- レイテンシ（50, 95, 99%tile）
- CPU 使用率
- メモリ使用率



Index データポイントの更新時間が長い

Index の更新・削除タスクの実行にはそれぞれ 1~2 時間程度かかっており、レコメンドのリアルタイム性を上げる際のボトルネックになる

- 更新：1~2 時間前後
  - 更新数 3,500,000 件程度
- 削除：30~40 分前後
  - 更新数 30 ~ 60 件程度

※ Embedding 総数は 14,000,000 件程度、Index のバッチ アップデートを利用

ストリーミング アップデートの利用検討

ストリーミング アップデートにより数秒以内に Index の更新が可能

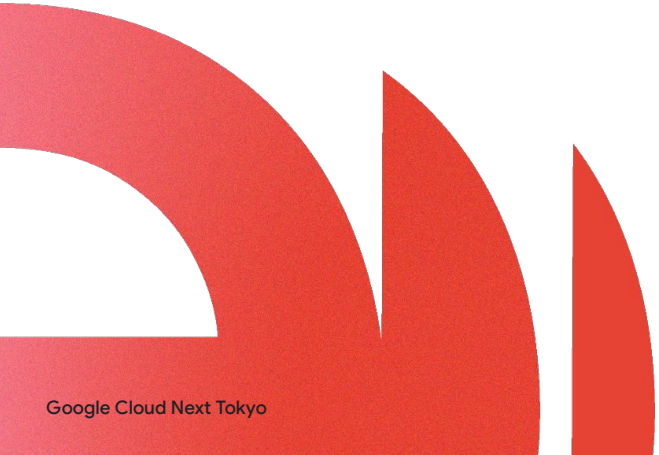
利用時には以下の制約に注意する必要がある

- バッチ アップデート Index に対してストリーミング アップデートは使用できない（新規作成が必要）
- 更新リクエストの制限 / Quota
  - データポイント数の上限 1,000 / request
  - スループット上限 120,000KB / min
  - リクエスト数上限 6,000 / min

Vertex AI Vector Search の費用

料金の要素

- シャード数 x レプリカ数 x マシンタイプ x 稼働時間
- Index 作成と更新時に処理されるデータ量



シャードサイズとマシンタイプ

選択可能なシャードサイズは3 種類

それぞれサポートされるデータサイズ・利用可能なマシンタイプが決まっている

- SHARD_SIZE_SMALL：2 GiB / シャード
- SHARD_SIZE_MEDIUM：20 GiB / シャード
- SHARD_SIZE_LARGE：50 GiB / シャード

マシンタイプ	SHARD SMALL	SHARD MEDIUM	SHARD LARGE
n1-standard-16	✔	✔	
n1-standard-32	✔	✔	✔
e2-standard-2	✔		
e2-standard-16	✔	✔	
e2-highmem-16	✔	✔	✔
n2d-standard-32	✔	✔	✔

データポイントの作成・更新量と費用

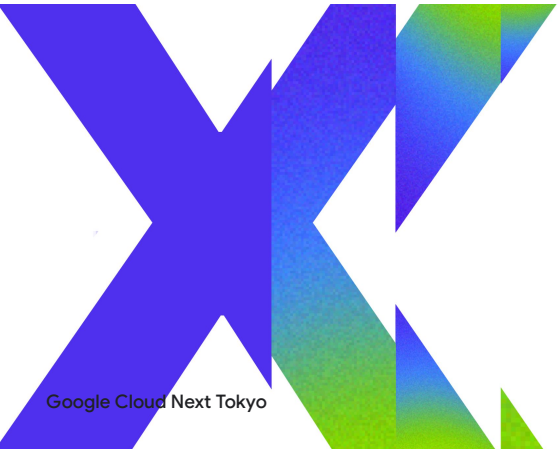
全てのリージョンに一律で、処理されたデータに対して **$3.00 / GiB** が課金される

※ 2025 年 7 月現在の費用

コーディネートレコメンドの場合

- **データ更新費用：$42 / day**
- 更新データ量：3.83GiB / day
  - 約 3,500,000 件の embedding
- Index サイズ：14GiB

※ 2025 年 7 月現在の費用



データ更新の費用を減らす対策

データ更新にかかる費用が想定よりもかなり多くなってしまった

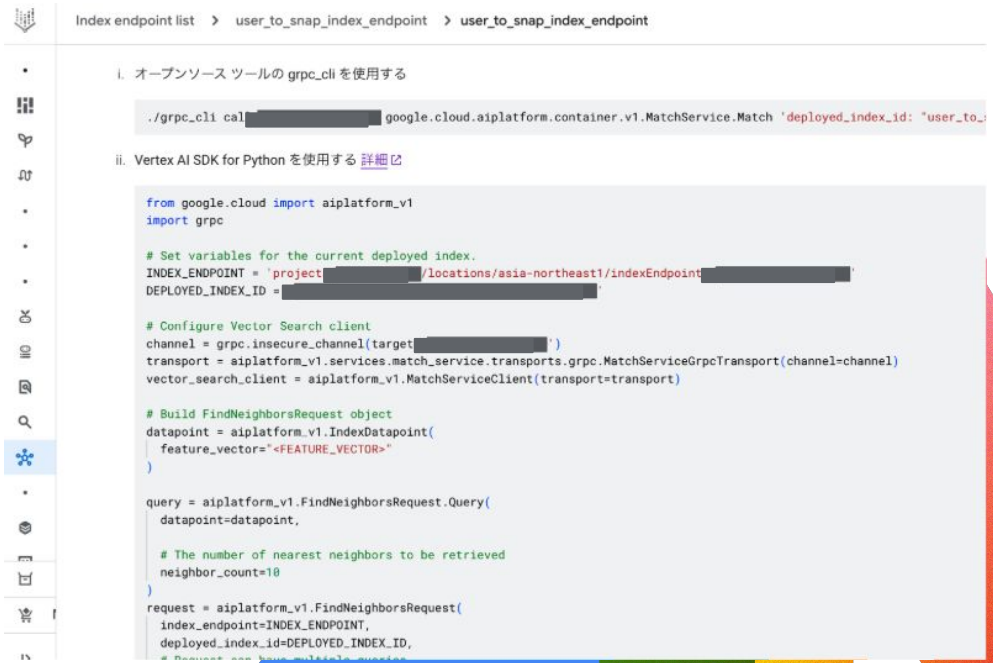
コーデ・ユーザー Embedding 合わせて **$80 / day 程度**

- Vertex AI Vector Search
- Cloud Firestore

対策として、Embedding 更新に閾値を設けて更新対象のデータ量を減らすことで費用を削減できないか改善中

その他 Tips

- クライアントライブラリによる
  リクエスト実装のサンプルコードはコンソールにある（Python）
- 表現が曖昧な箇所は English 版のドキュメントを見た方が良い
- ドキュメントで不足する部分は、サポートケースを活用
- デプロイ済み Index に含まれるデータは別途保存しておくと確認しやすい（BigQuery など）



07. まとめ

まとめ

- WEAR のコーディネート レコメンド機能に Vertex AI Vector Search を採用した
- 4 ヶ月程度運用し、今のところレイテンシ・パフォーマンス面で安定している
- 実装時のハマりどころは多いが、一度経験すれば次回からは開発・運用工数を抑えられそう
- 作成したい機能のシステム要件にマッチすれば今後も利用可能性は高い