ZOZOTOWN の大規模マーケティング メール配信を支えるアーキテクチャ

Google Cloud Next Tokyo

田島 克哉

株式会社ZOZO
データ・AIシステム本部 MA部
MA基盤ブロック
ブロック長

アジェンダ

01. ZOZOTOWN 概要

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

マーケティング メール配信

バッチ配信

多数の会員に対して一斉に配信。

配信時間はある程度決まっているものの、配信開始から終了まで数時間の幅が許容される。

例:「ブラックフライデーのセール開始」を通知する配信など

時間最適化配信

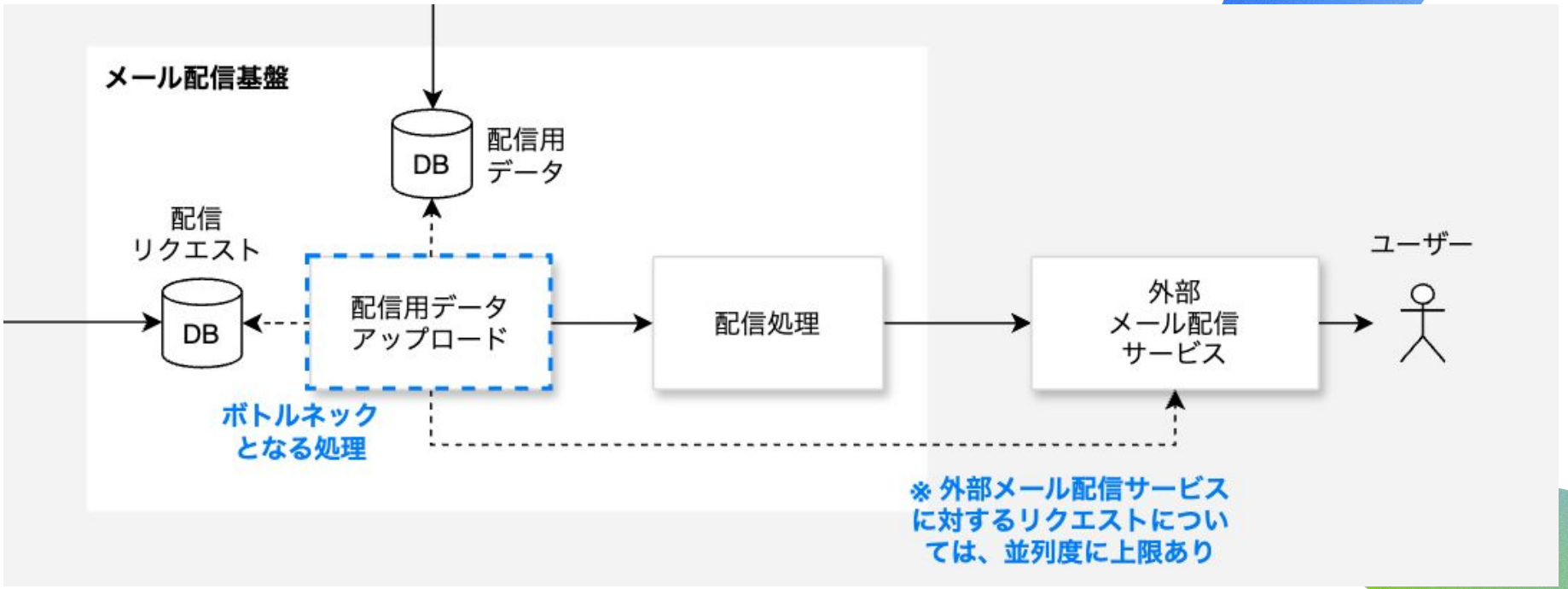
会員にとって最適な時間に配信。

配信が遅れると、ユーザーへのメリットが薄れる。

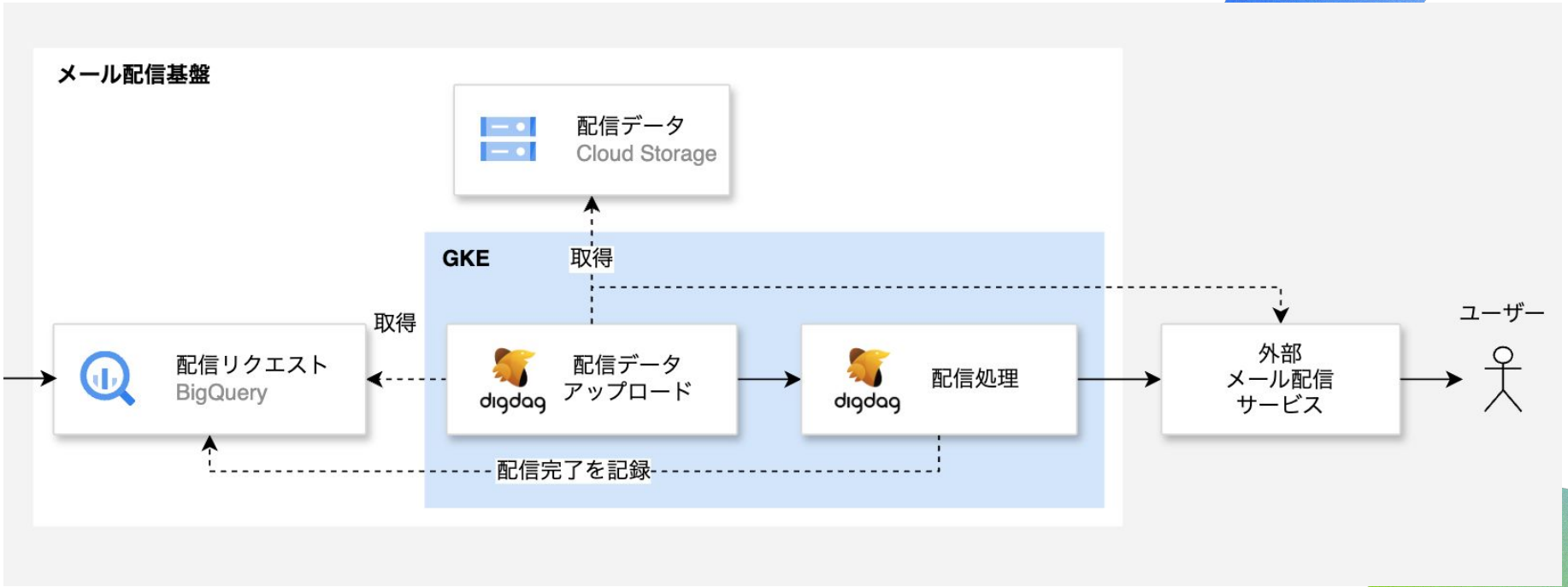
例:「お気に入りした商品の在庫がラスト1点になった」ことを通知する配信など

02. メール配信基盤

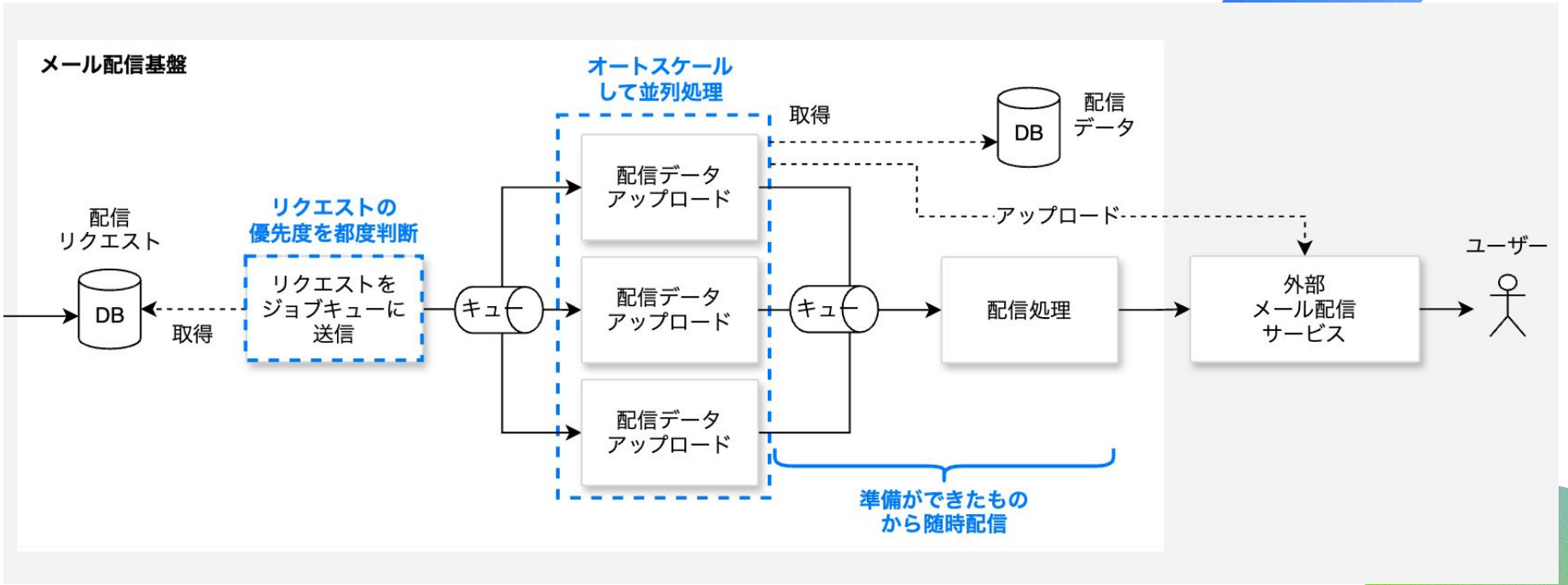
メール配信の流れ


メール配信基盤
DB
配信用
データ
配信
リクエスト
DB
配信用データ
アップロード
配信処理
外部
メール配信
サービス
ユーザー
ボトルネック
となる処理
※ 外部メール配信サービス
に対するリクエストについ
ては、並列度に上限あり

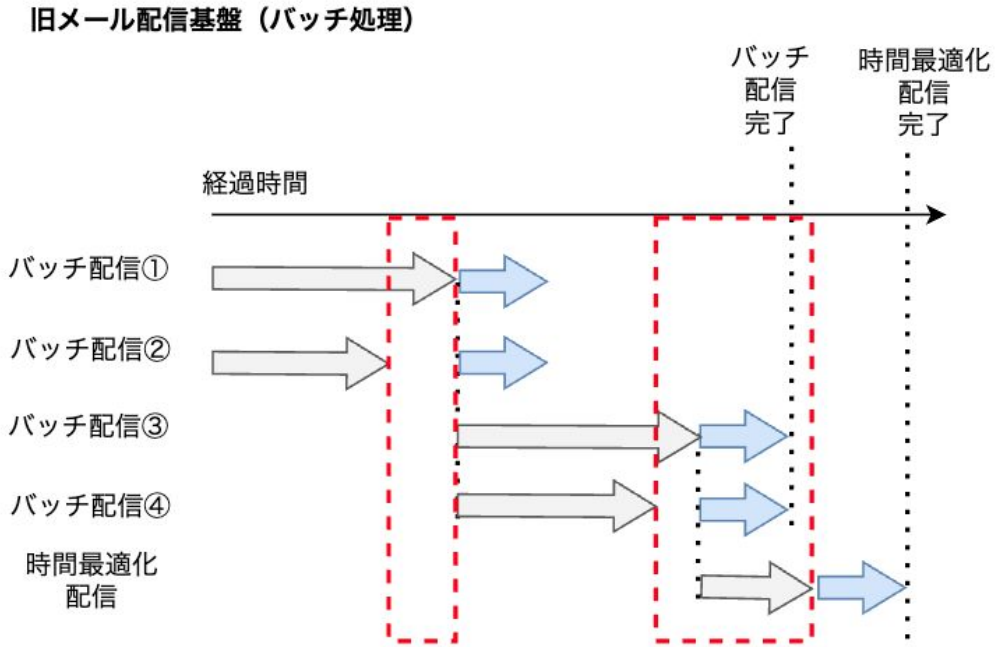
旧アーキテクチャ


メール配信基盤
配信データ
Cloud Storage
GKE
取得
取得
ユーザー
配信リクエスト
BigQuery
配信データ
アップロード
配信処理
外部
メール配信
サービス
配信完了を記録

新アーキテクチャの考え方



リアーキテクチャ前の処理



※ 簡略化のため、並列上限：2と仮定

配信データアップロード処理

配信処理

処理が1並列になる時間帯

リアーキテクチャによる処理の変化

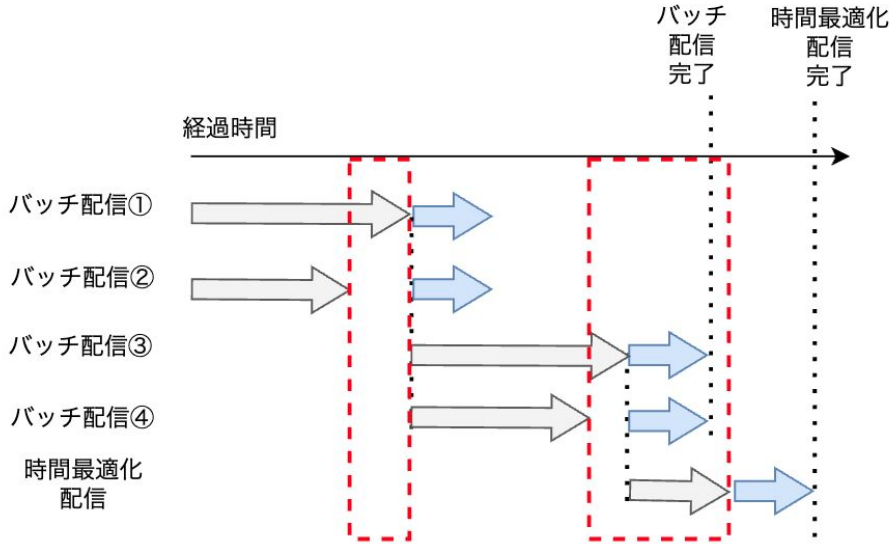
※簡略化のため、並列上限：2と仮定

配信データアップロード処理

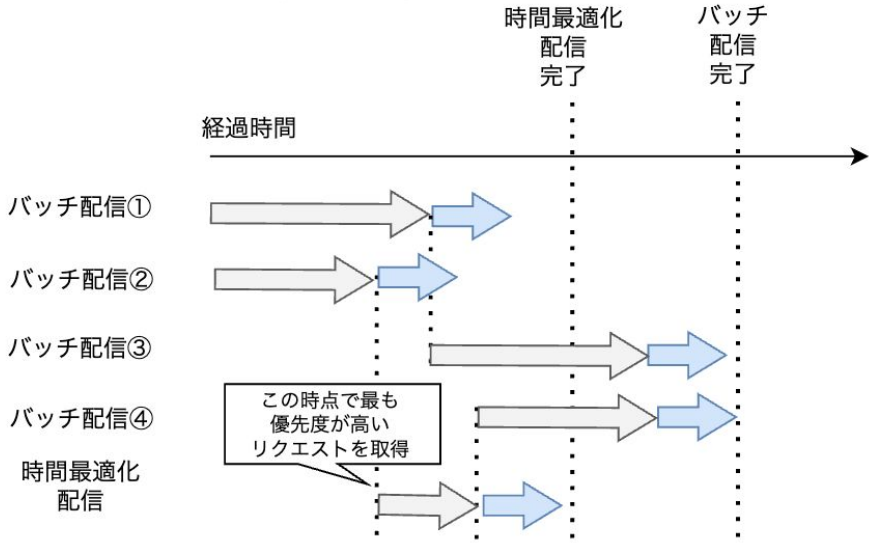
配信処理

処理が1並列になる時間帯

旧メール配信基盤（バッチ処理）



新メール配信基盤（非同期処理）

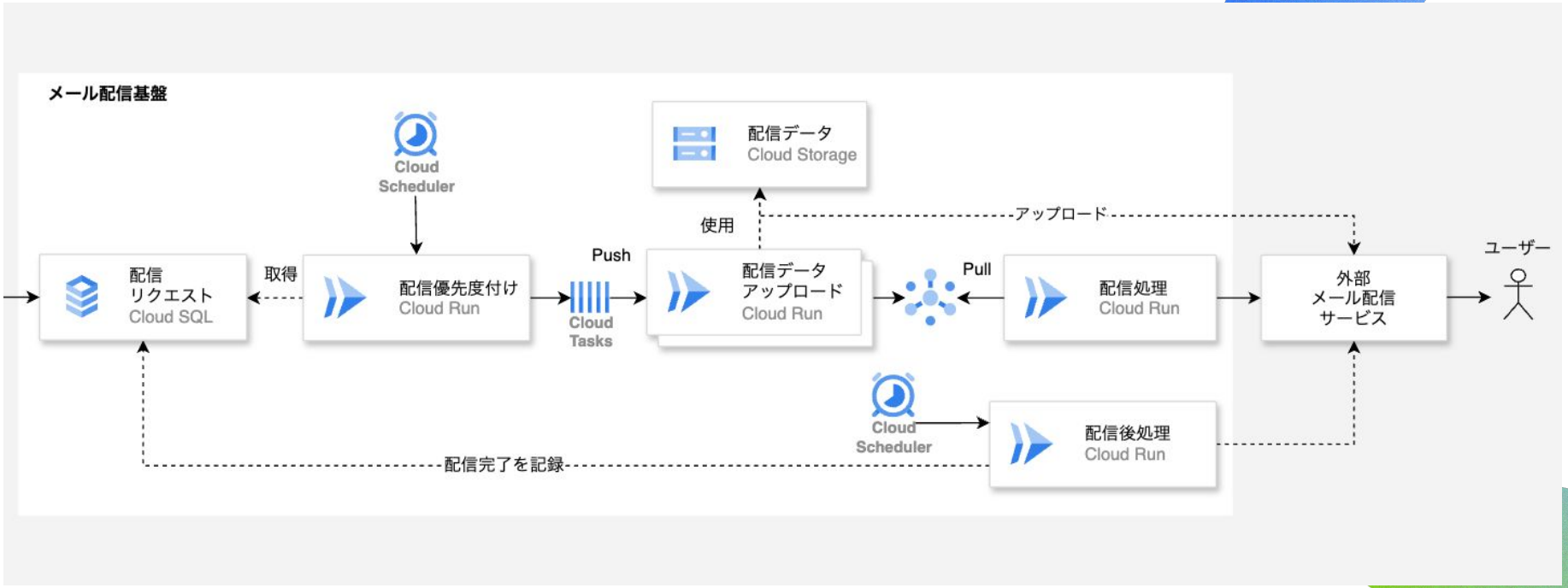


03. 技術選定

富永 良子

株式会社 ZOZO
データ・AIシステム本部 MA部
MA基盤ブロック
バックエンド エンジニア

新アーキテクチャ



ワーカーの選定

	GKE Standard	GKE Autopilot	Cloud Run
学習コスト	高：Kubernetes + インフラ設計・運用知識	中：Kubernetes 基本 + GKE 制約の理解	低：ほぼアプリケーション開発のみ
管理コスト	高：ユーザーがノードの作成・スケーリングなどを管理。クラスタ管理の運用負荷がかかる。	中：Google がノードを管理。Pod 単位でリソースを割り当てるため、クラスタ管理の負担がほぼない。	低：ソースコード / コンテナ イメージのみでデプロイ可能。インフラストラクチャのプロビジョニング、管理が不要。
柔軟性	高：ノード構成、ネットワーク ポリシーなどに対して高度なカスタマイズが可能。	中：Google が管理する制限があるため、一部のアドオンやカスタマイズができない。	低：基盤となるインフラや環境のカスタマイズはできない。
開発	ステートレス / ステートフル アプリケーションの両方をサポート。		ステートレス、リクエストドリブンのサービス、関数実行などに最適。
料金	クラスタ管理手数料 + ノード単位での課金	クラスタ管理手数料 + Pod 単位でリソース使用量に基づいて課金。	実行リクエスト単位の秒単位の課金。

各種ワークロードに対応する ジョブキューと Cloud Run の設定

Cloud Scheduler + Cloud Run Service (リクエストベース)/ Cloud Run Jobs

→ 配信優先度付け、配信後処理

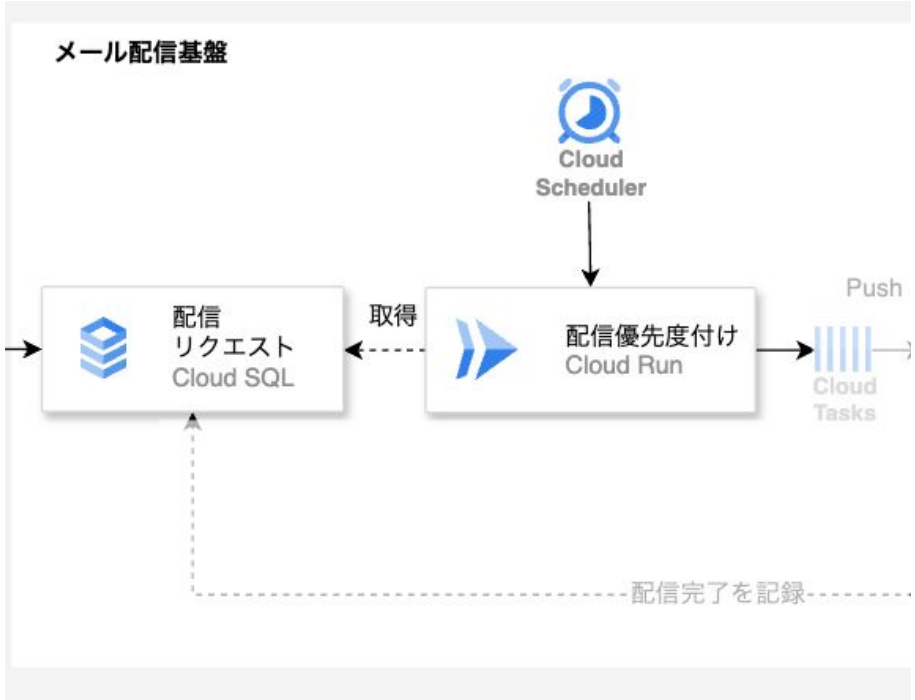
Cloud Tasks + Cloud Run Service (リクエストベース)

→ 配信データアップロード

Pub/Sub + Cloud Run Service (常時稼働 CPU)

→ 配信処理

Cloud Scheduler + Cloud Run Service (リクエストベース)



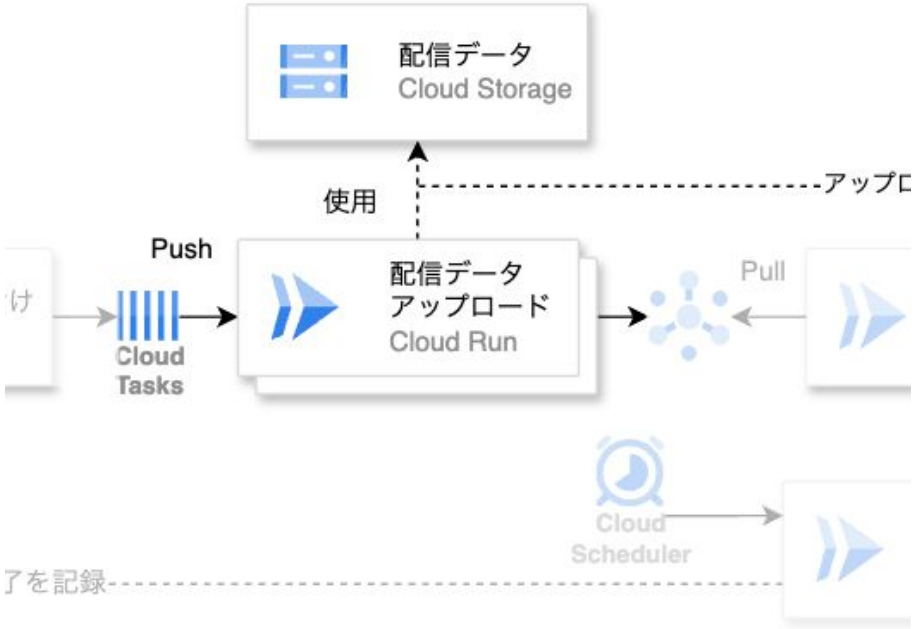
配信優先度付けサービス

最初の処理である、優先度の高い配信リクエストを取得する処理に使用。

Cloud Scheduler からは比較的短い間隔でリクエストを送り、処理が頻繁に実行されるようにしている。

Cloud Run Service のリクエスト最大同時処理数を 1 に設定し、デプロイ時のアクティブインスタンス数が 1 になるようにすることで、処理が重複しないようにしている。

Cloud Tasks + Cloud Run Service (リクエストベース)



配信データ アップロード サービス

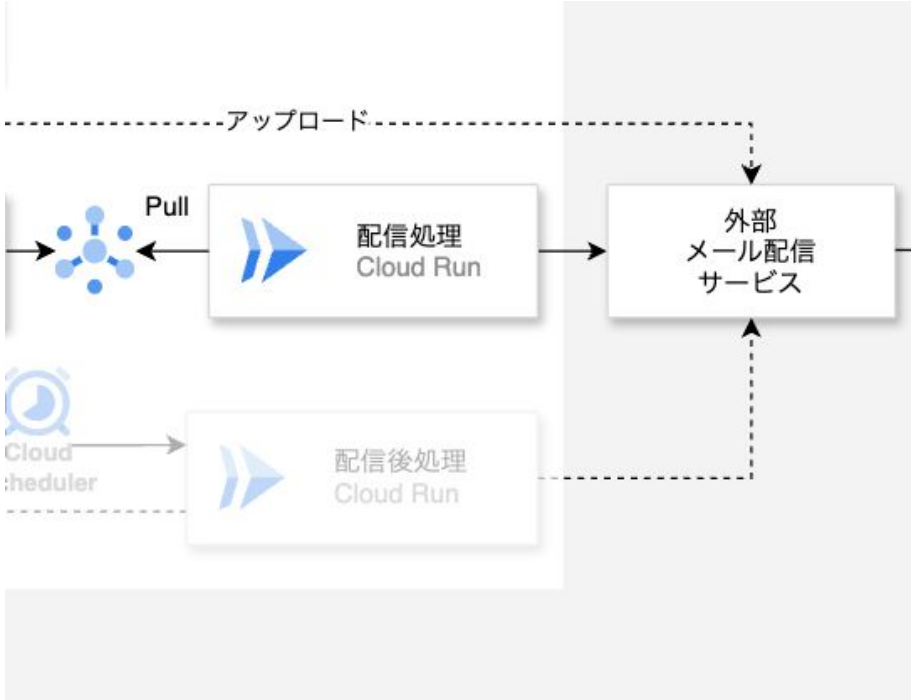
配信データを外部メール配信サービスへアップロードする処理に使用。

Cloud Tasks を使用し配信レートを管理することで、流量制限に対応しつつ処理を最適化。

ボトルネックとなっていた処理のため、並列に処理を行い効率化。Push 型の HTTP リクエストでリクエスト量に応じてオートスケールさせ、配信がない時間帯はゼロスケールすることでコストメリットを実現。

また、Cloud Storage をマウントすることで、配信データを参照している。

Pub/Sub + Cloud Run Service (常時稼働 CPU)



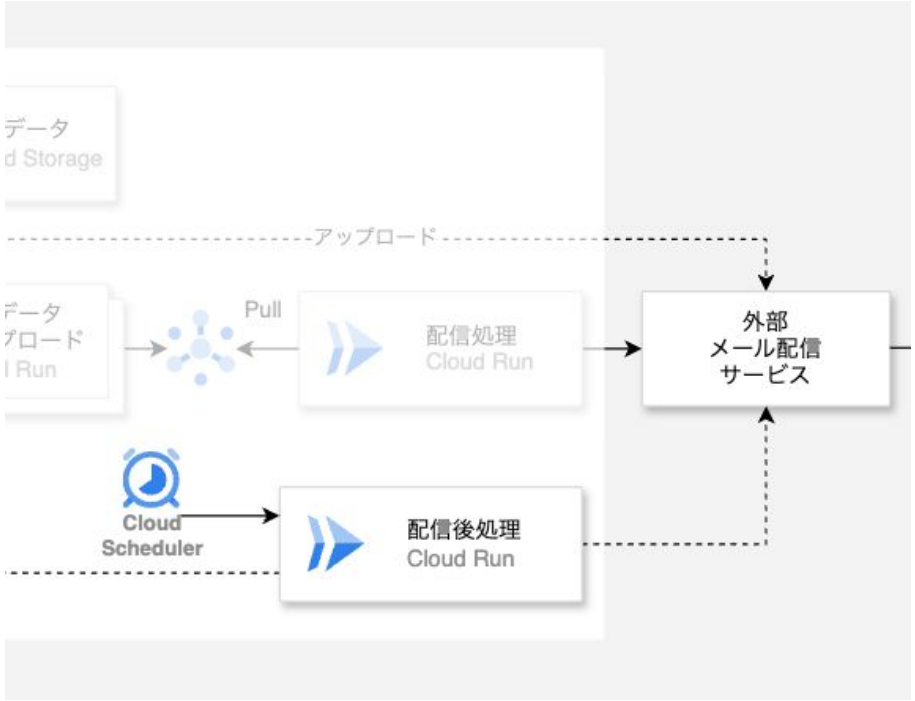
配信処理サービス

Pub/Sub のメッセージを取得し、外部メール配信サービスに配信リクエストを行う。

配信が重複しないよう、配信処理は exactly once で実行する必要がある。そのため、配信処理を行う Cloud Run は pull サブスクリプションでメッセージを取得する。

将来的には、Cloud Run Worker Pools を使用する可能性あり。

Cloud Scheduler + Cloud Run Jobs



配信後処理サービス

外部メール配信サービスでの配信完了を確認し、配信実績ログを書き込んだり、配信リクエストのステータスを変更するなどの後処理を行う。

Cloud Run Jobs は 5 分おきに起動。全ての処理が冪等になるようにし、処理失敗時にもジョブを再実行すれば良い状態にしている。

04. 成果

リアーキテクチャの成果

ビジネス要件への対応

配信リクエストの並列処理を最適化することで、リソースを最大限に活用しつつ**配信の優先順位を柔軟に決定して処理できる**ようになった。

バッチ配信と時間最適化配信の両方を、それぞれ最適なタイミングで配信できるようになった。

配信速度の向上

配信量のピーク時に**1,400 万通程度の配信にかかる時間が、リアーキテクチャ前より 40% 短縮**。より多くの配信可能時間を確保できるようになった。

1 つの大きいリクエストがボトルネックとなり他のリクエストの処理が待ち合わせをする、といった影響がなくなった。

運用負荷の軽減

GKE によるワークフローシステムの運用からフルマネージドな Cloud Run へ移行することで、**システムの運用負荷が大幅に軽減**。Kubernetes の学習コストも不要に。

Pub/Sub や Cloud Tasks が持つリトライなどの機能により、低い開発コストで保守が容易なアプリケーションを実現。

Cloud Run を使用して感じたメリット

開発アジリティの向上

コンテナイメージを用意するだけで、高速に開発とデプロイを繰り返せる。

オートスケーリングのほか、Cloud Storage のマウントやCloud SQL との接続、並列処理数の上限の設定など、必要な機能を簡単に実現。

2025 年には IAP のビルトインなども発表され、より便利に進化し続けている。

スケーラビリティとコスト効率

軽量なコンテナを瞬時に起動し、高トラフィックにも即時対応。リクエストや CPU 使用率に応じて自動で高速にスケール。

従量課金制であることに加え、最小インスタンス数を 0 に設定することでリクエストがないときは課金されないようにすることも可能。

運用負荷が低い

フルマネージドなサービスであるため、インフラの構築やサーバーの保守がほぼ不要。

スケーリング、ネットワーク、セキュリティなどを設定で管理でき、複雑な構成管理や手作業の運用を大幅に削減。

高可用で障害に強い。

ログやモニタリングが統合されている。