Amazon ECS / AWS Lambdaを組合せた
類似商品検索及びリアルタイム在庫検索システムのご紹介

登壇者紹介

南沙佳

ライフスタイル事業本部

吉田明広

ライフスタイル事業本部

VISION：ニューラルポケットが描くAIスマートシティの姿

当社が実現するスマートシティの姿1

待ちのない街

AI技術を活用したデータ解析で
待ち時間から楽しさ・安心時間に

当社が実現するスマートシティの姿2

情報に出逢える街

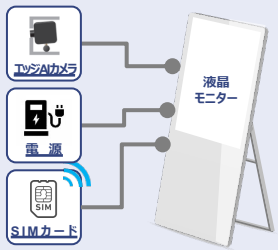
地域や個人に密接した情報を通して人々の行動変容を促す
AIで視聴分析や配信自動化を支援

会社沿革

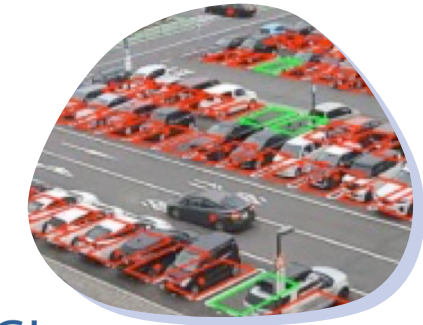
’18/8 ファッショントレンド分析の
AIMDの提供開始

’18/7 エッジAI技術の
開発開始

SIGN DIGI



’19/5 広告搭載のAIサイネージ開発開始



’20/8 駐車場満空把握の
「デジパーク」を開発開始

AI MEDIA PROVIDER
FOCUS CHANNEL
by Neural Pocket

’21/11 マンション広告サイネージ大手の
『フォーカスチャネル社』を完全子会社化

Neural Engineering

’21/10『ニューラルエンジニアリング』社
の設立、建設業許可の取得

JPX
東証マザーズ上場

NETTEN
by Neural Pocket

’22/2 屋外LEDサイネージ販売大手の
『ネットテン社』を完全子会社化

創業

2018年 2019 2020 2021 2022

小売店在庫分析では、データ集約~行動変容を繋げ、大きな価値創出を狙う

- アパレル
- 人流・交通量、駐車場
- リアル広告
- **小売店在庫**

- 非構造化データ(例:画像)→構造化(深層学習 etc.)
- **データの集約**

リアルタイム在庫計算システム

- **データ分析によるインサイトの抽出**
  - 機械学習 / 深層学習 etc.
- **意思決定のサポート**
  - 数理最適化 etc.

類似商品検索システム

- 情報の可視化
- 最適候補の提案

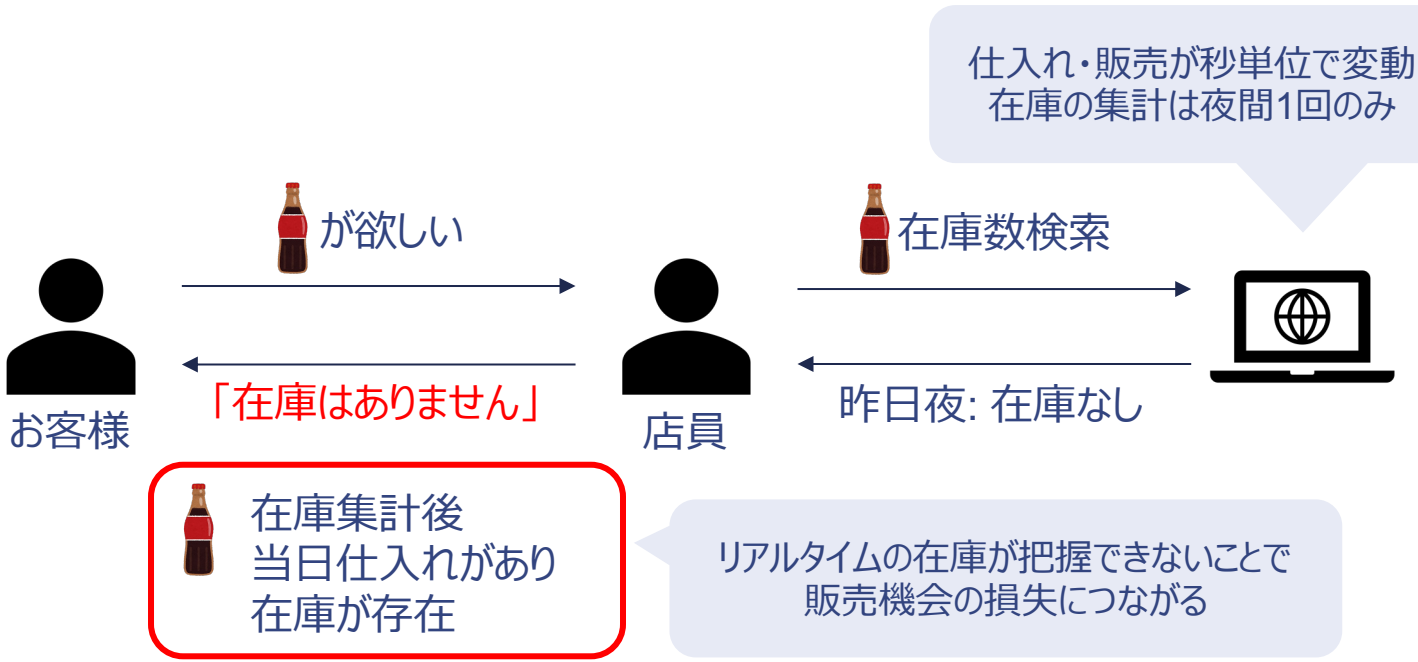
ユーザーに対する候補提案UI

- ムダの削減 (在庫破棄削減 / 混雑解消 etc.)
- 利便性向上 (新しい選択肢の認知)

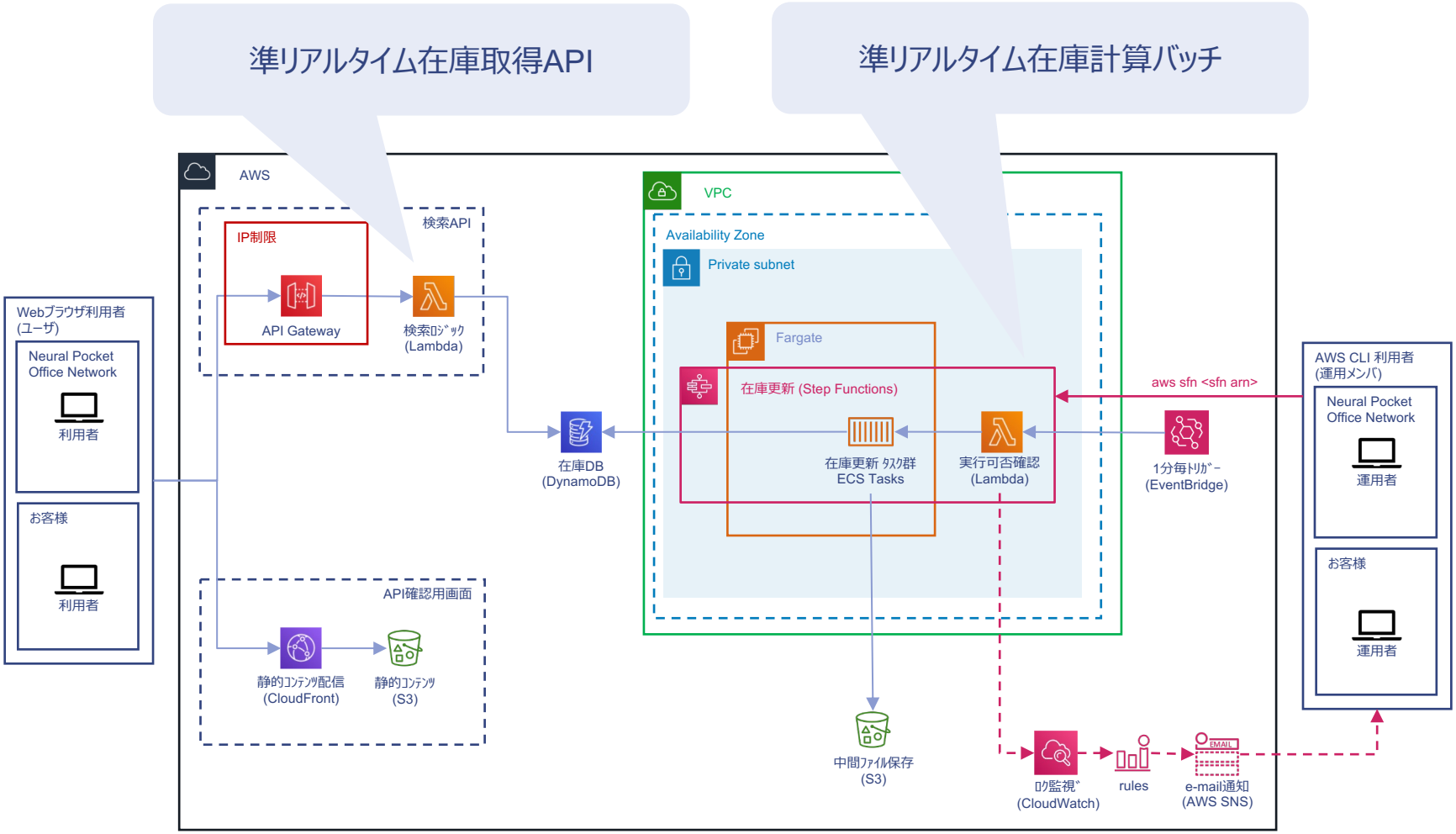
- リアルタイム在庫計算システム
- 類似商品検索システム

小売業界のビジネス課題

商品数・店舗数が多く、リアルタイムの在庫把握が困難



システム構成図


準リアルタイム在庫取得API
準リアルタイム在庫計算バッチ
AWS
検索API
IP制限
API Gateway
検索ﾛｼﾞｯｸ
(Lambda)
VPC
Availability Zone
Private subnet
Fargate
在庫更新 (Step Functions)
在庫更新 ﾀｽｸ群
ECS Tasks
実行可否確認
(Lambda)
在庫DB
(DynamoDB)
1分毎ﾄﾘｶﾞｰ
(EventBridge)
aws sfn <sfn arn>
Webブラウザ利用者
(ユーザ)
Neural Pocket
Office Network
利用者
お客様
利用者
AWS CLI 利用者
(運用メンバ)
Neural Pocket
Office Network
運用者
お客様
運用者
API確認用画面
静的ｺﾝﾃﾝﾂ配信
(CloudFront)
静的ｺﾝﾃﾝﾂ
(S3)
中間ﾌｧｲﾙ保存
(S3)
ﾛｸ監視
(CloudWatch)
rules
e-mail通知
(AWS SNS)
EMAIL

準リアルタイム在庫取得API ~API紹介~

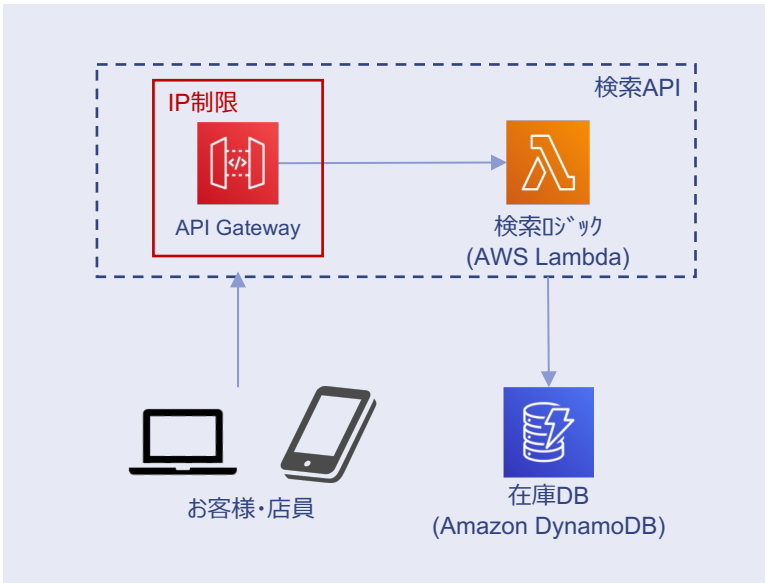
お客様や店員の方が準リアルタイムで在庫情報を取得できるようなAPIを作成

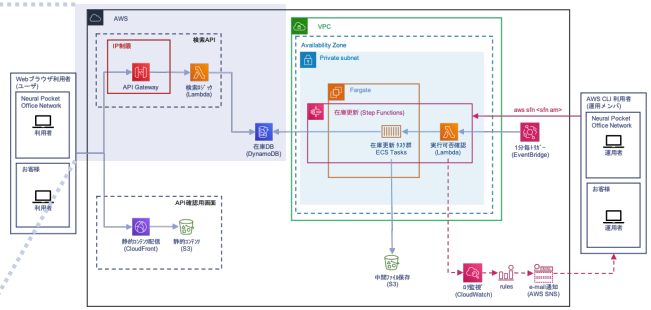
- API種類
  - GET
    - {itemCd}/{storeId}
      - 単一アイテム & 単一店舗の在庫数
    - {itemCd}
      - 単一アイテムに対する全店舗の在庫数
  - POST
    - Body
      - itemCdList
      - storeCdList
    - 複数アイテム×複数店舗の在庫をまとめて取得

複数のアイテム・店舗を同時検索可能にすることで、
APIの呼び出し回数を削減

準リアルタイム在庫取得API ~技術紹介~

お客様が利用することを想定し、スケーリングを考慮可能





全国からのアクセスが想定され、
スケーリング可能なAWS Lambdaを使用

Serverlessで環境構築/メンテナンスが不要

フロントも合わせて開発したため、
Node.js + TypeScriptで実装

連携されるデータと在庫計算の流れ

月1回連携
棚卸しデータ

1日10^7 ~10^8件程度

随時連携
POSデータ

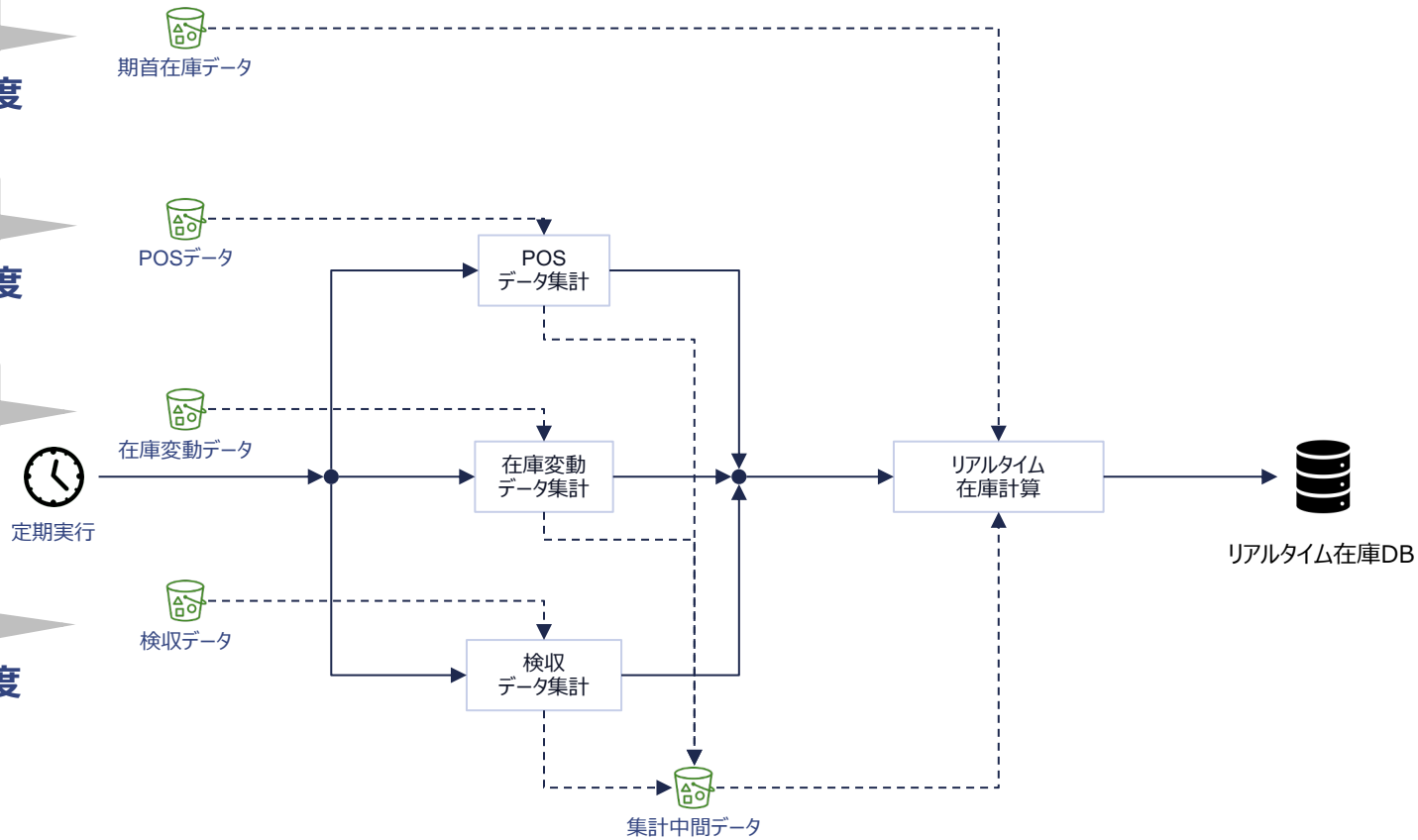
1日10^5 ~10^6件程度

随時連携
廃棄/店舗移動データ

1日10^4件程度

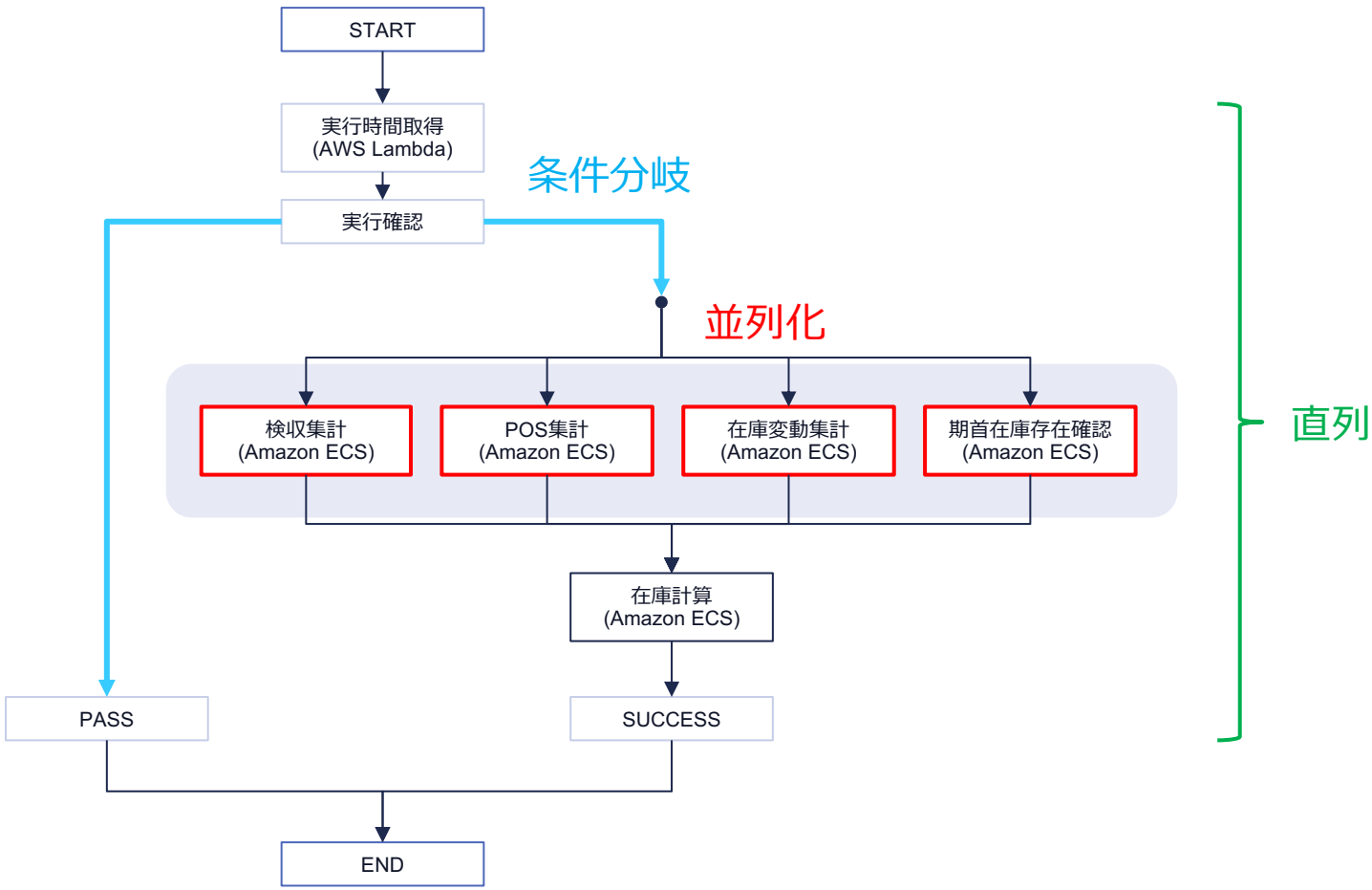
随時連携
納品データ

1日10^5 ~10^6件程度



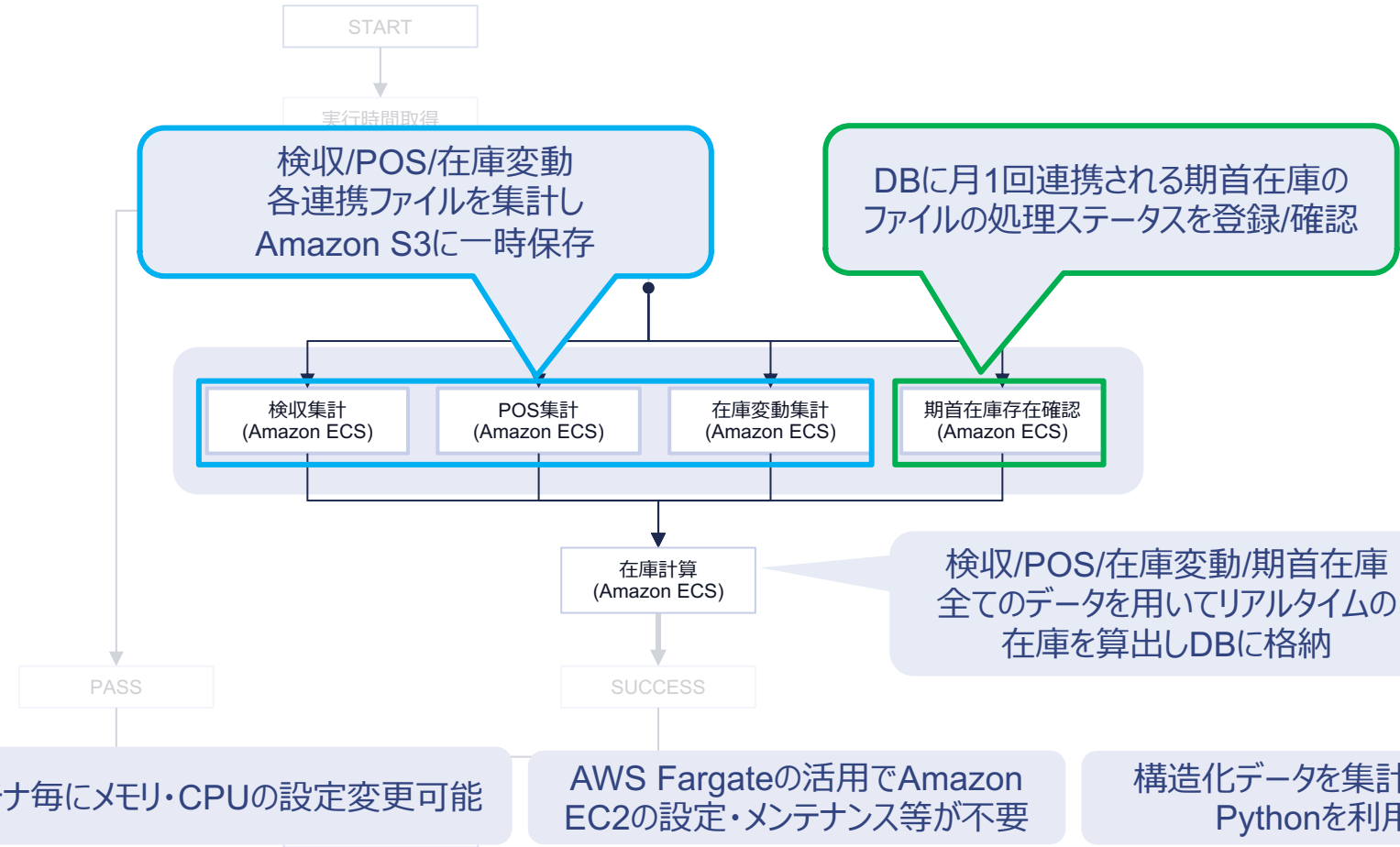
準リアルタイム計算バッチ ~AWS Step Functionsによる処理順序指定~

タスクの条件分岐・並列・直列の実行順序を定義可能



準リアルタイム計算バッチ ~AWS Step Functionsによる処理順序指定~

ホストマシンの設定が不要となり、複数タスクをそれぞれのコンテナとして起動可能
(タスク失敗時はタスク単位で自動で規定回数まで再実行)



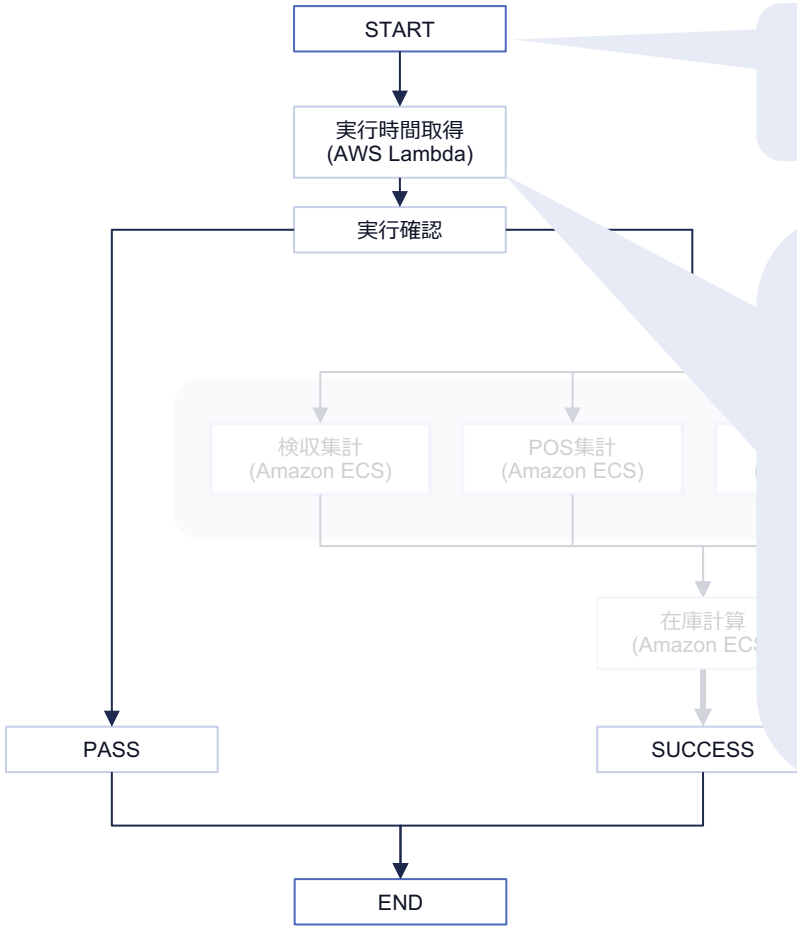
コンテナ毎にメモリ・CPUの設定変更可能

AWS Fargateの活用でAmazon EC2の設定・メンテナンス等が不要

構造化データを集計しやすいPythonを利用

準リアルタイム計算バッチ ~AWS Step Functionsによる処理順序指定~

タスク実行の重複・抜け漏れを防ぎ、準リアルタイムで在庫を更新し続ける



Event Bridgeで1分ごとに実行
→遅延が発生しても自己修正可能

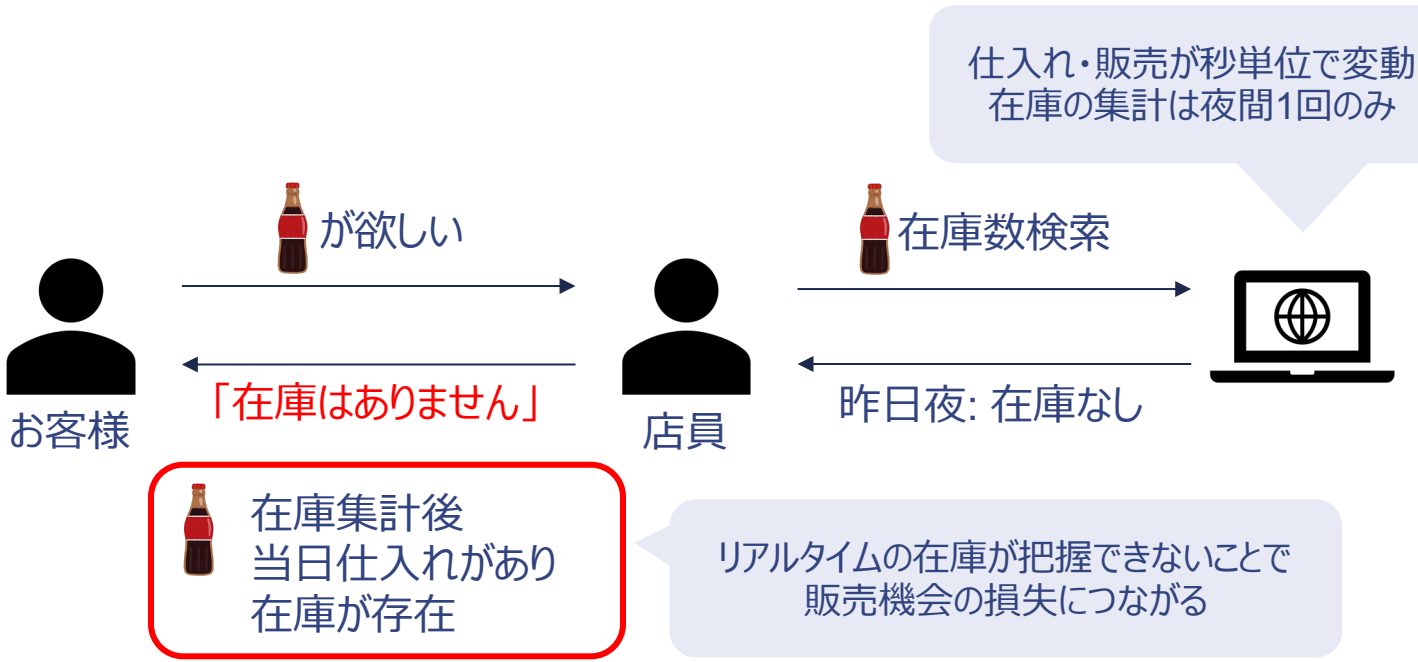
- 実行要否確認
  - 実行中のタスクが存在する場合PASS
  - 前のジョブでエラー発生の場合PASS
  - 次回処理時間を取得
    - 営業時間外/未来の時間の場合PASS
    - 過去の時間の場合
      その時間帯のデータ集計を実施
  - →重複・抜け漏れの防止
- AWS Lambdaを使用しServerlessで瞬時にAmazon DynamoDB にアクセスし
  ステータスを確認

- リアルタイム在庫計算システム

- **類似商品検索システム**

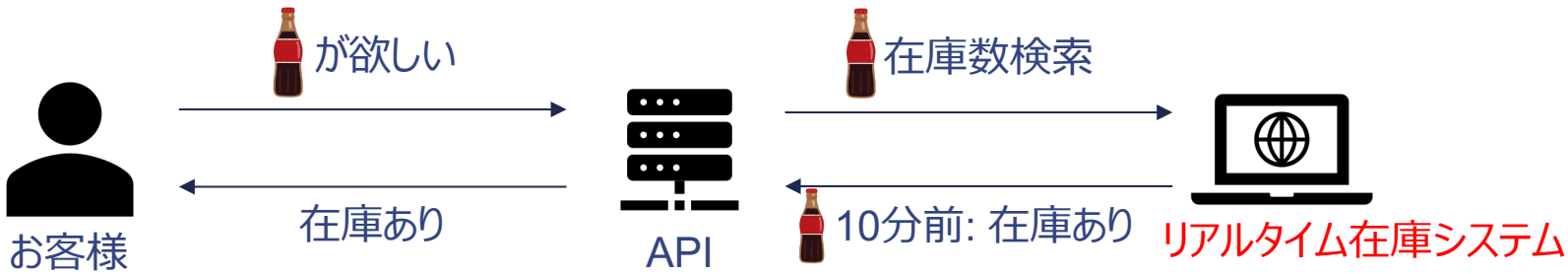
小売業界のビジネス課題

商品数・店舗数が多く、リアルタイムの在庫把握が困難



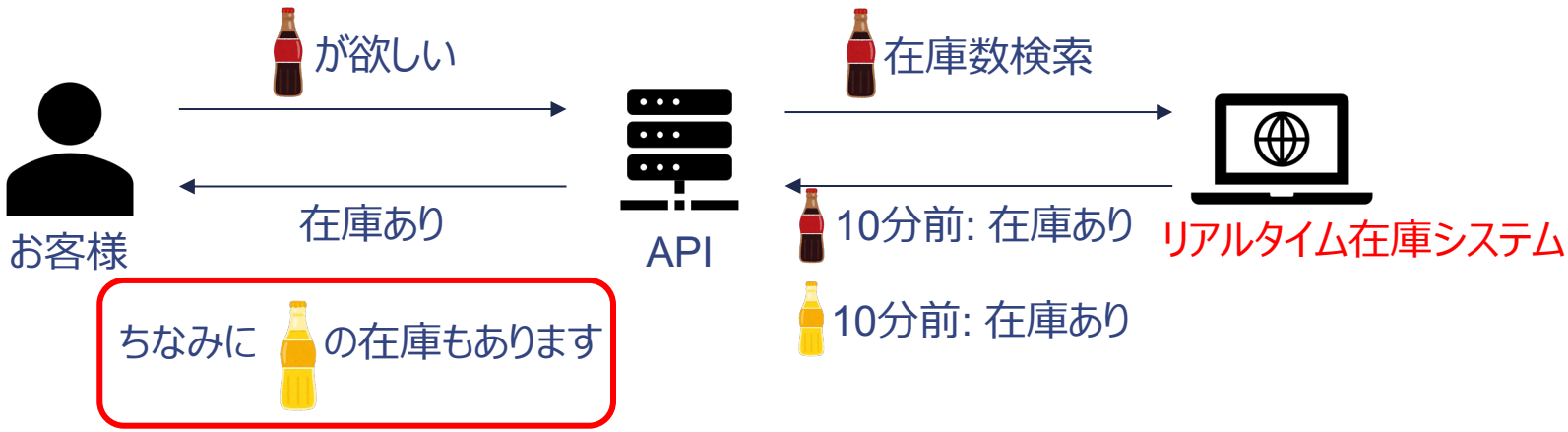
小売業界のビジネス課題

リアルタイム在庫システムの構築により誰でも在庫の把握が可能に

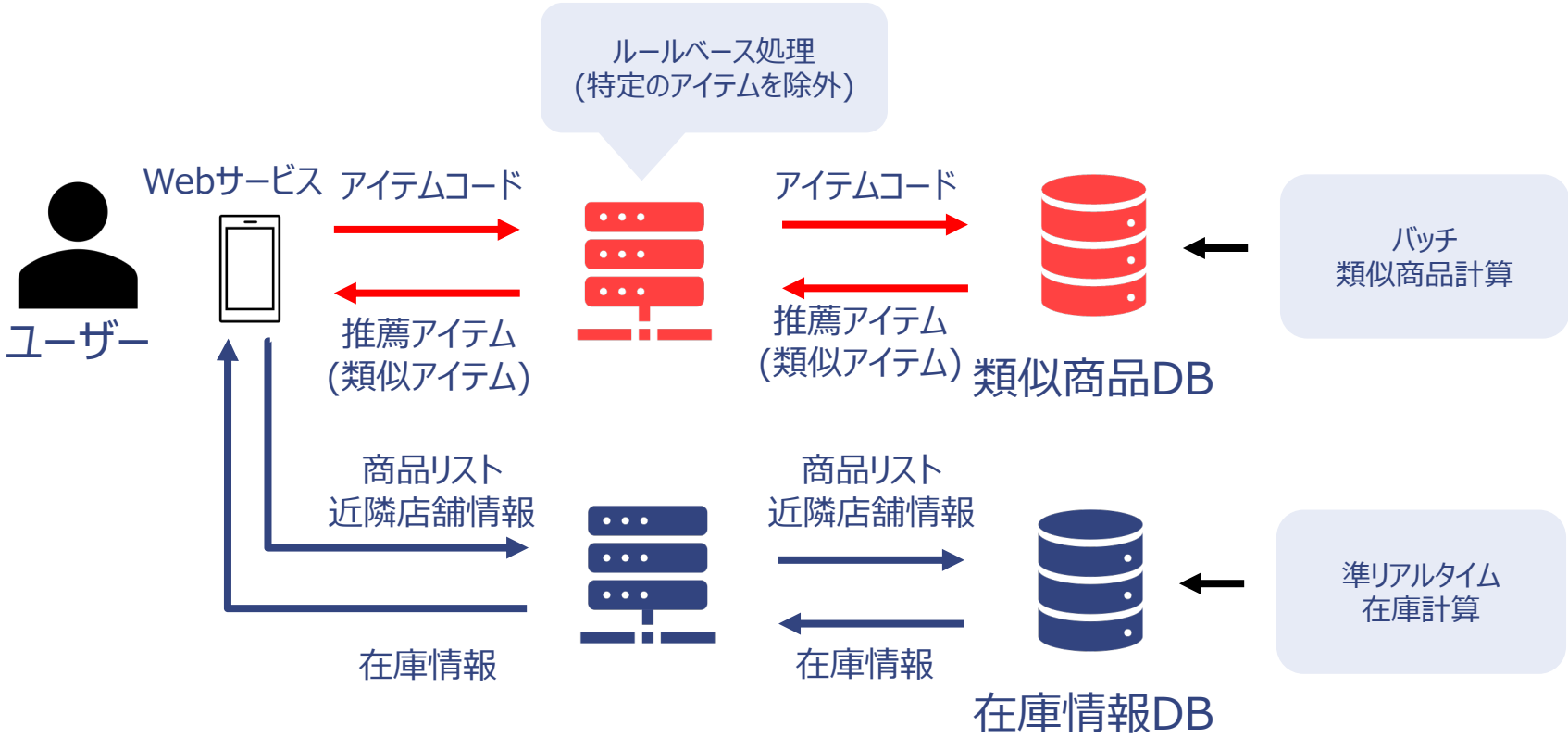


小売業界のビジネス課題

類似商品を含めた在庫情報を提供することで商品認知の機会を増やしたい

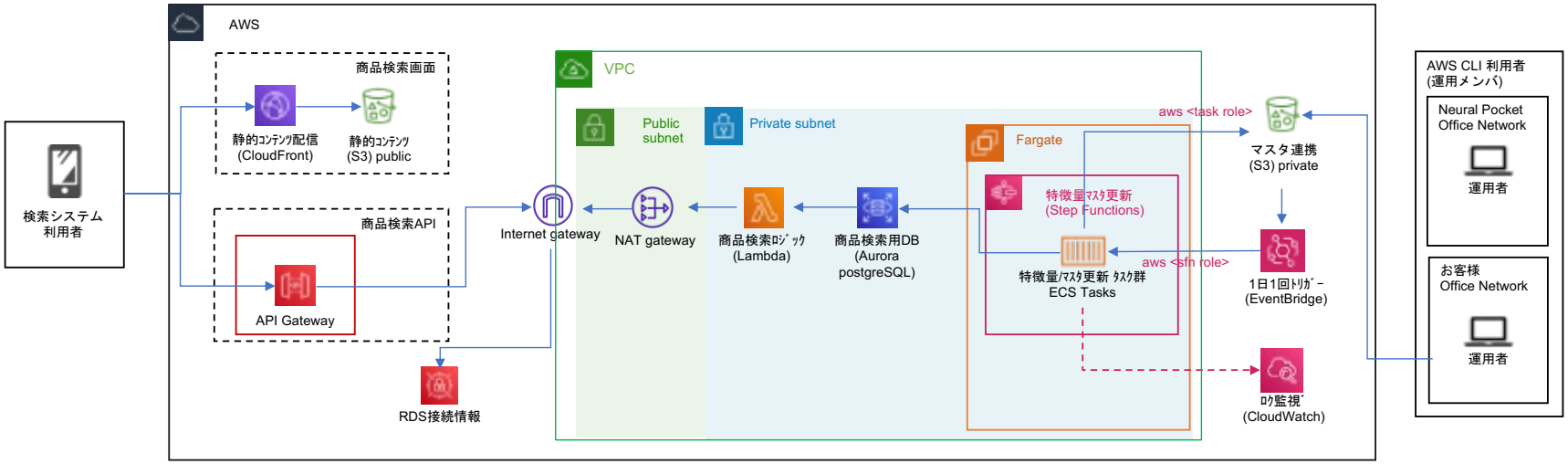


類似商品検索を含む在庫検索の流れ

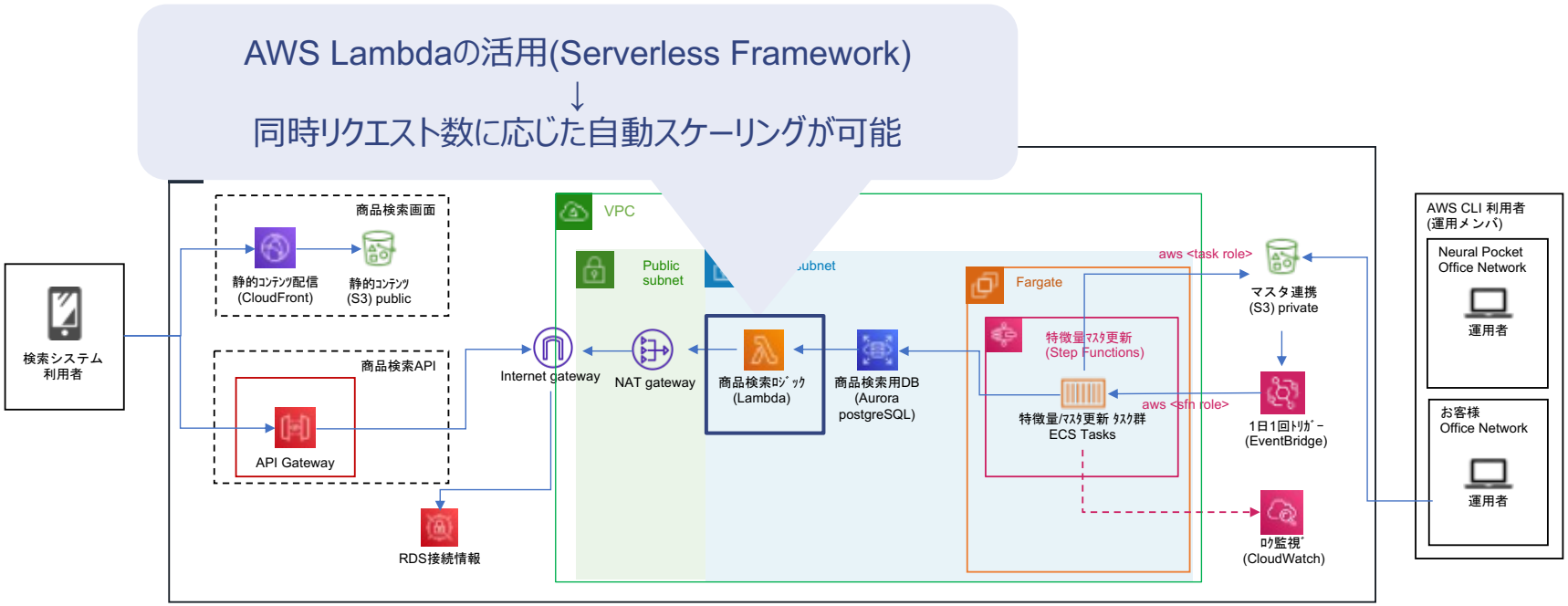


①類似アイテム検索 / ②在庫情報取得

類似商品検索システムの構成

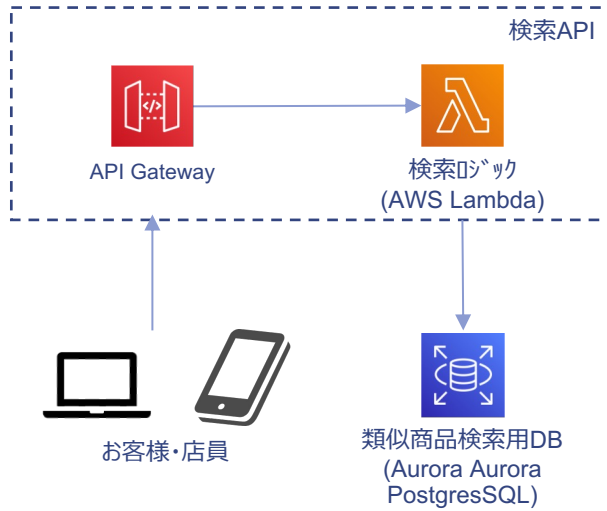


類似商品検索システムの構成



API (AWS Lambda + Amazon API Gateway) タスク紹介 / 技術選定

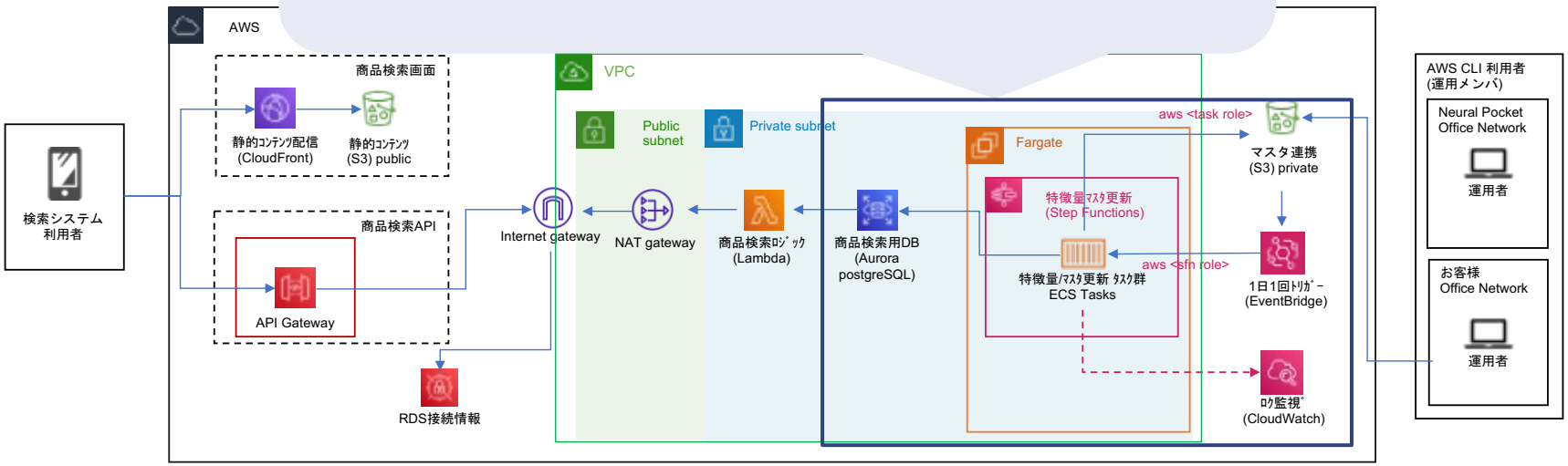
- 類似商品の検索
  - API種類
    - GET
      - {itemCd}
        - アイテム毎に類似商品を返すAPI



- AWS Lambda (Node.js + TypeScript)
  - Serverless Frameworkで環境構築が必要ない
  - 日本中のお客様からのアクセスを想定しているため、アクセス増加の際のスケーリング可能
- APIGateway (REST API)

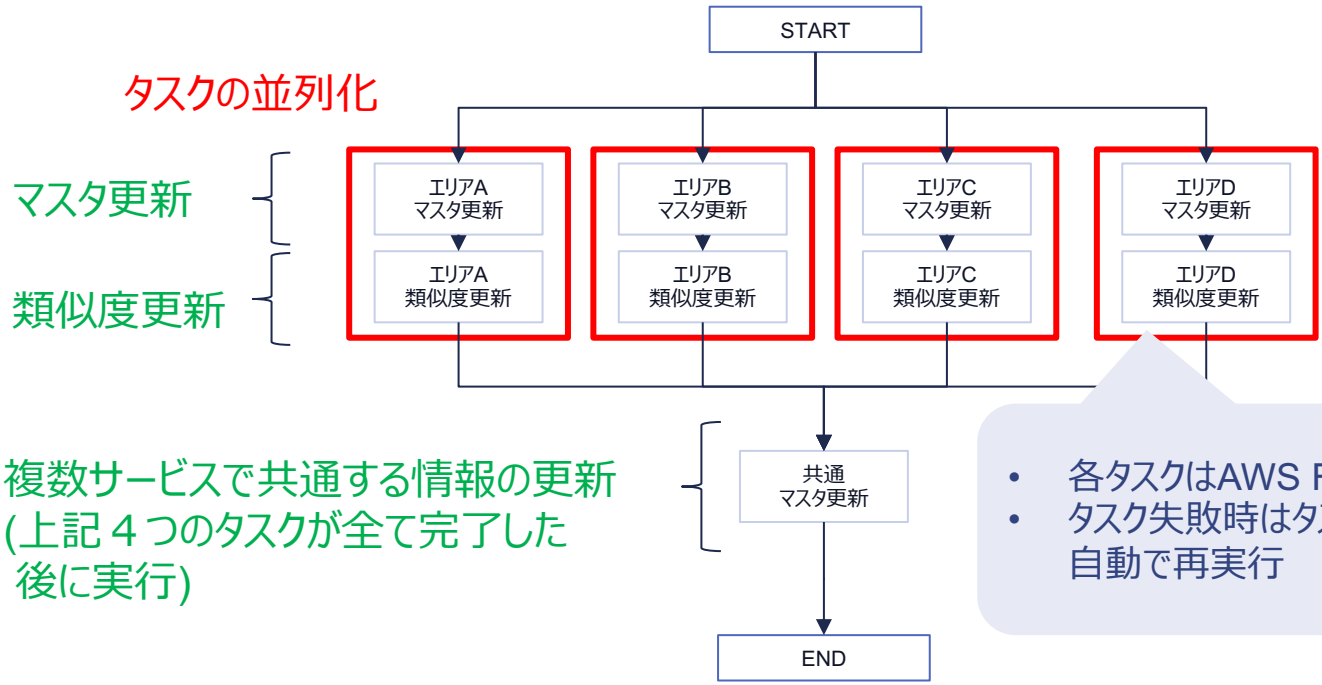
類似商品検索システムの構成

- AWS Fargate上でマスタ更新及び
  商品間の類似度*計算(後述) (1 日 1 回定期実行)
- 複数のタスクを直列・並列実行するために
  AWS Step Functionsの活用(次頁)



類似商品検索システムの構成 ～バッチ実行のAWS Step Functionsの構成～

AWS Step Functionsの活用により、複数のタスクの並列・直列実行を簡易的に実行可能に

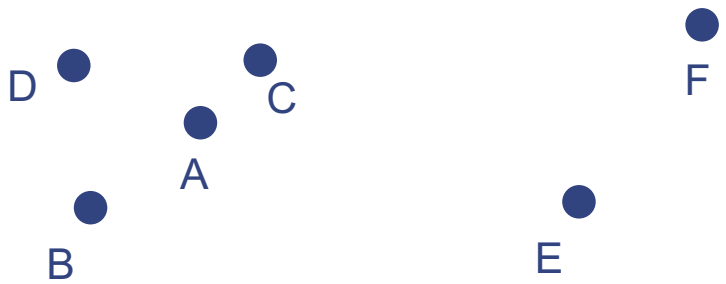


- 各タスクはAWS Fargateで定義
- タスク失敗時はタスク単位で
自動で再実行

効率的な類似度計算

組合せ数が非常に多い状況で、全組合せの類似度を計算することなく、
一定以上の類似度のペアを全て抽出したい

力任せに計算を行うと $\frac{n(n-1)}{2}$ 回の計算が必要 (n: データ数, $n > 10^5$)



距離行列

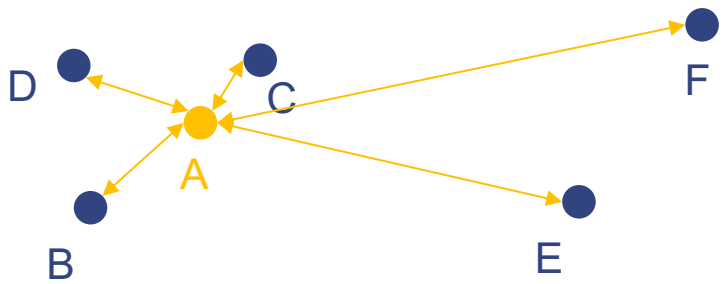
	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

$\frac{n(n-1)}{2}$ 回

効率的な類似度計算

組合せ数が非常に多い状況で、全組合せの類似度を計算することなく、
一定以上の類似度のペアを全て抽出したい

①一部のデータ(アンカー)を抽出し、アンカーと他の全アイテムとの類似度の計算を行う



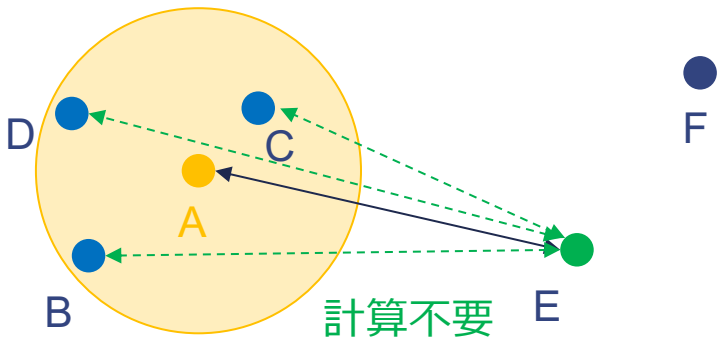
距離行列

	A	B	C	D	E	F
A	0	0.2	0.1	0.3	0.7	0.8
B		0				
C			0			
D				0		
E					0	
F						0

効率的な類似度計算

組合せ数が非常に多い状況で、全組合せの類似度を計算することなく、
一定以上の類似度のペアを全て抽出したい

①一部のデータ(アンカー)を抽出し、アンカーと他の全アイテムとの類似度の計算を行う
②アンカー以外のデータ(クエリ)を１つ抽出
クエリとアンカーの距離が遠い場合、アンカーに近いデータとクエリの類似度計算は不要



距離行列

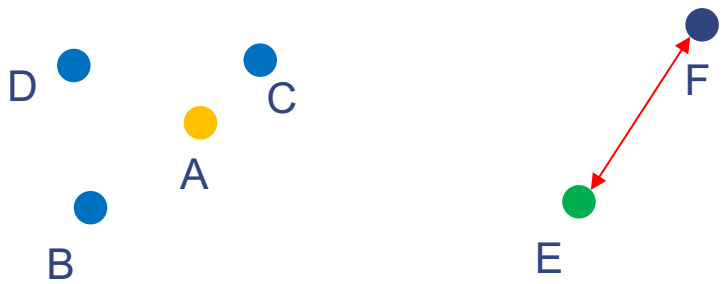
	A	B	C	D	E	F
A	0	0.2	0.1	0.3	**0.7**	0.8
B		0			計算不要	
C			0			
D				0		
E					0	
F						0

効率的な類似度計算

組合せ数が非常に多い状況で、全組合せの類似度を計算することなく、
一定以上の類似度のペアを全て抽出したい

①一部のデータ(アンカー)を抽出し、アンカーと他の全アイテムとの類似度の計算を行う
②アンカー以外のデータ(クエリ)を１つ抽出
クエリとアンカーの距離が遠い場合、アンカーに近いデータとクエリの類似度計算は不要
③残ったペアのみ計算

※実際はアンカーを複数用意する



距離行列

	A	B	C	D	E	F
A	0	0.2	0.1	0.3	0.7	0.8
B		0			計算不要	
C			0			
D				0		
E					0	
F					0.4	0

テストについて

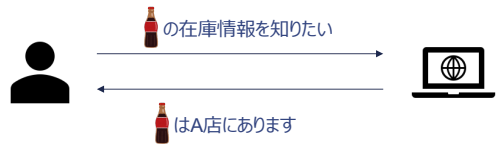
	ツール	選定理由
API単体テスト	Postman	• アプリ上でテストケースの追加、実行が容易 • Jsonファイルを出力して共有が容易
API負荷テスト	Taurus	• jmxファイルの共有のみで他の端末でも簡単にテスト実行可能 • 複数のテストツールのラッパー
Batch単体テスト	PyTest	• 書きやすさ
UIテスト	Cypress	• コマンド毎に画面のスナップショットを保存可能

本プロジェクトで使用した技術スタックについて

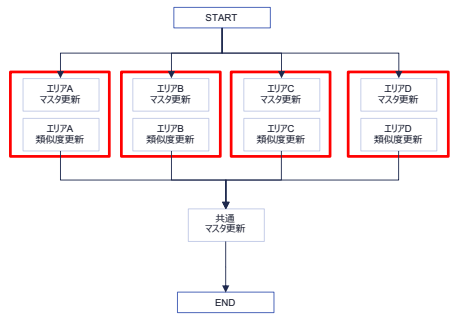
	技術スタック
バックエンド	• TypeScript / Serverless / Node.js (API) • Python (Batch)
インフラ	• 各種AWSサービス (Amazon S3 / AWS Fargate / AWS Step Functions / AWS Lambda / Amazon DynamoDB / Amazon Aurora PostgresSQL etc..) • Terraform
フロントエンド	• React • TypeScript

まとめ

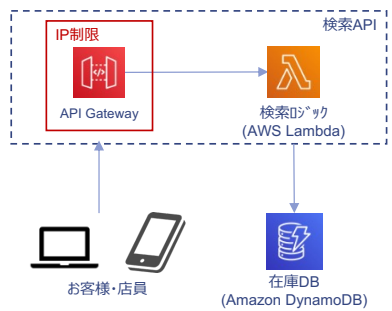
- 商品数・購買数が多い小売業態において
  準リアルタイムで在庫計算システム及び
  類似商品検索システムについてご紹介

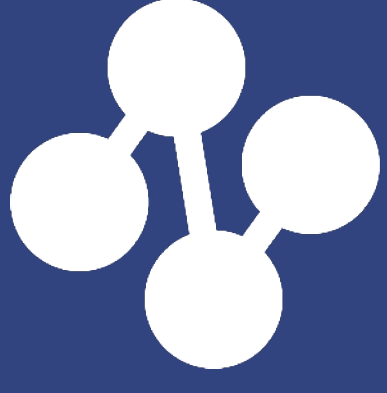


- バッチ処理において、複数のECSタスクを
  直列・並列に実行するための
  AWS Step Functionsの実例についてご紹介



- リアルタイム処理において、AWS Lambdaのような
  サーバーレスアーキテクチャを活用することで
  簡便な実行環境管理及びスケーリングを可能に





Neural Pocket