Kafka ビジネス価値プレイブック

企業の神経システム

Kafkaで何が変わる？

- 出来事（イベント）をそのまま会社の“神経”に流します。
- バッチ待ちを減らし、今起きたことに合わせて動けます。
- システムどうしをゆるくつないで、改修しやすく、安全に拡張できます。

ーー技術の背景については別の資料の紹介した通り。

身近な事例（一覧）

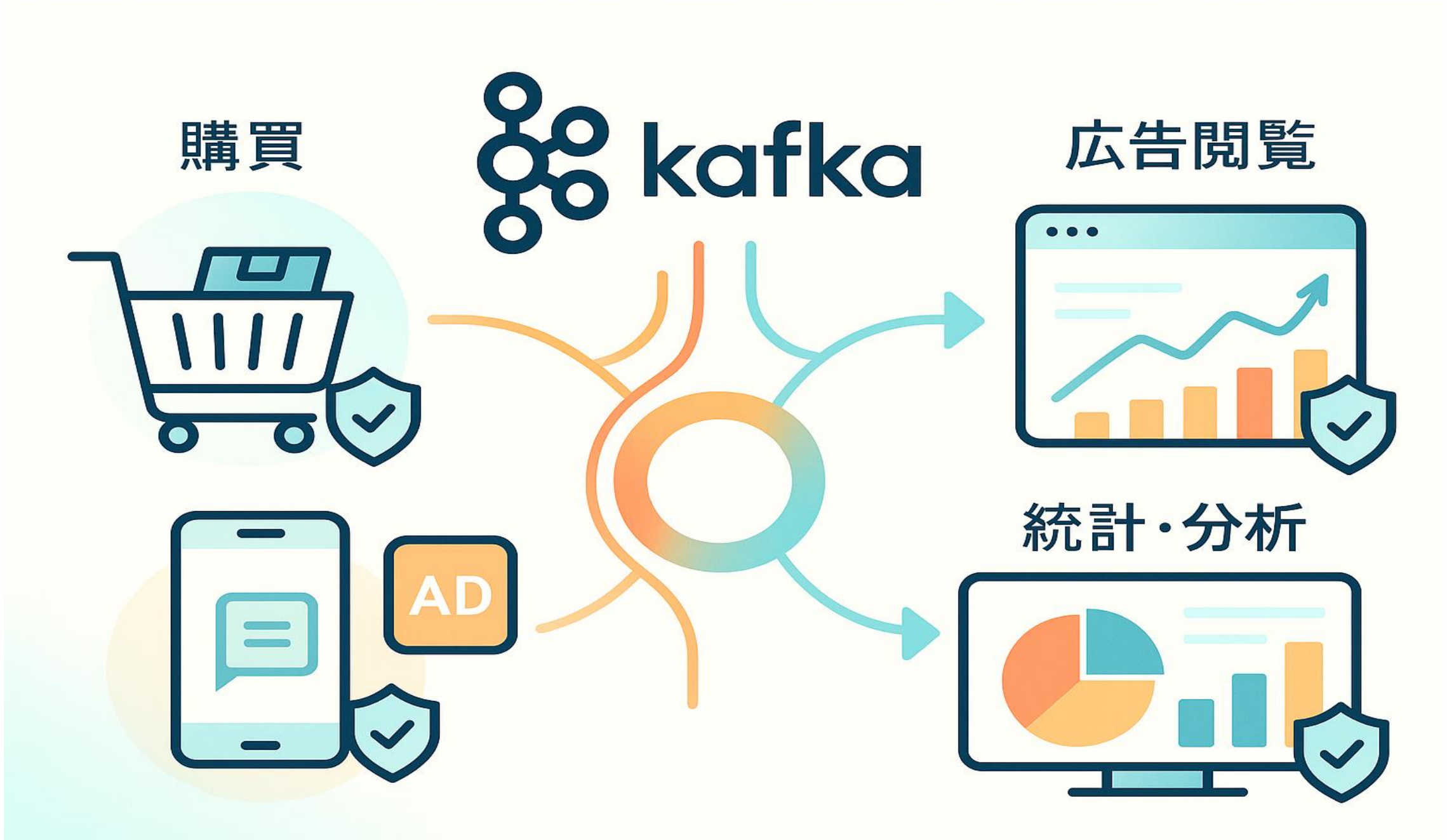
- ヤフー（Yahoo! JAPAN）（ポータル / 広告）：ニュース/検索/広告で生まれる出来事（クリック・閲覧・入札）をそのまま流す → Kafkaを中核にしたデータパイプラインで欠損防止・疎結合化
- ZOZOTOWN（ZOZO）（EC / 小売）：商品在庫の変化を即時に反映して、欠品や誤表示を防ぐ → SQL Server→KafkaのCDCでリアルタイム連携（Qlik Replicate）
- メルペイ（Mercariグループ）（キャッシュレス / 金融）：支払・チャージなどの明細を“その場”で分析、異常検知・CRMにも活用 → Kafka＋Flink/CDC（Spanner→Kafka）でストリーム基盤

身近な事例（一覧）

- トヨタ自動車（クルマのデータを都内DCと北海道DCで双方向同期し、
  処理を近い場所に寄せる。）
- デンソー（設備の停止/稼働/品質データを“そのまま流し”、
  異常の早期検知や品質トレーサビリティ。）
- コマツ（KOMATSU）（現場の稼働データをストックせず“流し込んで”分析に回す。）
- JR東日本物流（最適化の取り組み）（工場→センター→駅店舗の出荷/入荷/在庫/配送を、イベントでつなげて遅延や欠品を抑止。）
- 次世代スマートメーター（経産省資料）（家・街の電力状態を“数分単位”で把握、余剰電力に合わせて制御。）

（Yahoo! JAPAN） のケース

- 業種：ポータル / 広告
- ニュース/検索/広告で生まれる出来事（クリック・閲覧・入札）をそのまま流す
- 使い方：Kafkaを中核にしたデータパイプラインで欠損防止・疎結合化
- 期待できる効果：データ欠損率↓、配信/分析の遅延↓、改修スピード↑
- 出典：Yahoo! Tech Blog（2019）


購買
kafka
広告閲覧
AD
統計・分析

ZOZOTOWN（ZOZO） のケース

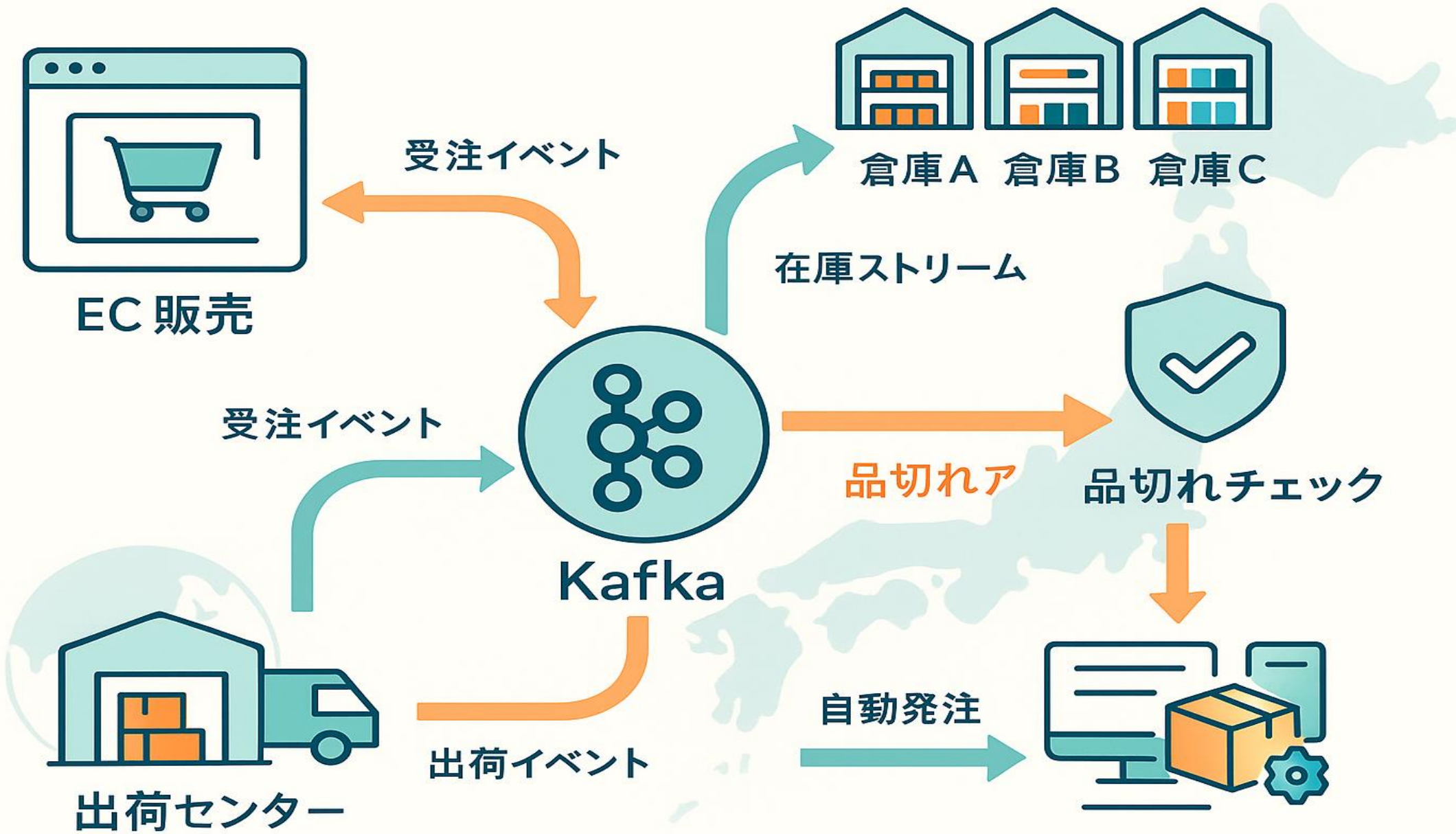
業種：EC / 小売

身近な例：商品在庫の変化を即時に反映して、欠品や誤表示を防ぐ

使い方：SQL Server→KafkaのCDCでリアルタイム連携（Qlik Replicate）

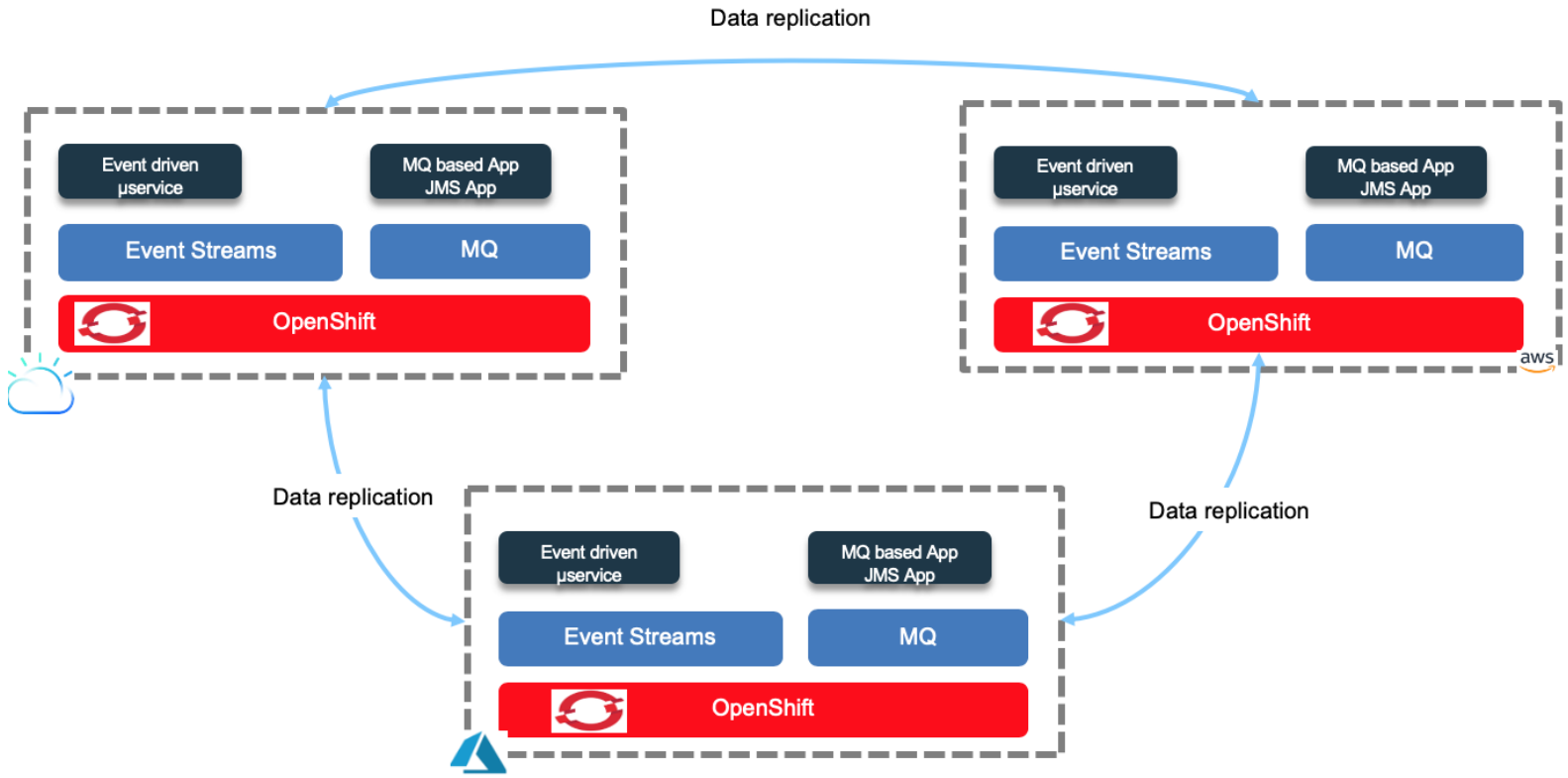
期待できる効果：在庫反映時間↓、品切れ率↓、在庫評価精度↑

出典：ZOZO Tech Blog（2020）


EC 販売
受注イベント
倉庫A 倉庫B 倉庫C
在庫ストリーム
受注イベント
Kafka
品切れア
品切れチェック
出荷センター
出荷イベント
自動発注

トヨタ自動車 の例

- 自動車
- クルマのデータを都内DCと北海道DCで双方向同期し、処理を近い場所に寄せる。
- 使い方：広域データ同期（MirrorMaker2）とエッジ連携の実験。走行データやアプリ処理を負荷・遅延に応じて分散。
- 技術の要点：Kafka / MirrorMaker2, OpenShift, エッジ負荷分散, 監視
- 参考：トヨタ『コネクティッドカーとエッジコンピューティングによるグリーンモビリティへの挑戦』(2023)



出典: https://ibm-cloud-architecture.github.io/refarch-eda/technology/kafka-mirrormaker/

観点	昔（Kafkaなし）	Kafka導入後
情報の届く速さ	夜や定時にまとめて反映。遅れが出やすい	ほぼその場で共有。遅れが小さい
情報のズレ	画面・部署ごとに数字が合わないことが多い	同じ出来事が同時に共有されやすい
つながり方	システム同士を1対1で多数接続（配線だらけ）	真ん中（Kafka）に一度だけ渡し、各自が受け取る
止まりやすさ	どこかが止まると連鎖して全体が止まりがち	誰かが遅くても他は動ける。影響が広がりにくい
ピーク対応	売上集中時に処理詰まり・取りこぼし	いったん受け取り、後から順にさばける
拠点またぎ	拠点ごとに連携の作り込みが必要	拠点間へ同じ流れを配れる（展開が楽）
変更・追加	新しい連携を足すたびに既存に手戻り	新しい受け手を追加しやすい（既存に最小影響）
見える化	集計は翌日・翌週になりがち	いま起きている状況をそのままグラフ化しやすい
障害復旧	どこまで処理したか不明でやり直しが大変	途中から再開しやすい（履歴が残る）
セキュリティ運用	いろんな所がDBに直接つながる	受け渡し口を一本化して管理しやすい

具体シナリオで見ると

EC在庫：旧来は“夜バッチで在庫同期”→昼の特売で売り越し＆キャンセル多発。
Kafka後は注文、生産、在庫等のイベントが即時連鎖し、在庫確保→出荷→残量閾値アラート→自動発注
までをリアルタイム化。

広告×購買計測：旧来はWebログ集計が翌日→広告効果の最適化が常に後手。
Kafka後は閲覧・クリック・購買が同じ時間軸のストリームに載り、配信入札やレコメンドを即時に調整。

以前のシステムが抱えた根本原因（まとめ）

バッチ/同期主義で時間軸がズレる

点対点連携の増殖で変更に脆い

中央DB依存でスケール/可用性ボトルネック

履歴不在で再処理・検証が難しい

地理分散とスキーマ進化に弱い

Kafkaは「書いた順の耐久イベントログ」を中核に据えることで、上記を設計レベルで解消します。