aws DevAx connect

「疎結合」を実現する メッセージングサービスの選択と利用

Shingo Sugimoto
Solutions Architect, Industry Solutions
Amazon Web Services Japan K.K.
sugishin@amazon.co.jp

自己紹介

杉本 晋吾　Shingo Sugimoto

技術統括本部 インダストリーソリューション部

ソリューションアーキテクト (SA)

略歴

現在 ← 海外開発製品の技術営業 ← ITコンサル会社のCTO

← 小さいソフトハウス経営 ← アパレル事業会社のCIO

← ECパッケージベンダーのCTO ← ネットプロバイダー管理者

好きなAWSサービス

- AWS App Runner, AWS Fargate
- Amazon DocumentDB

aws DevAx connect

Agenda

1. 「疎結合」である必要性と根本的な動機を考えてみる
2. メッセージングサービスの相違点
3. AWSのメッセージングサービスの適用とユースケース
4. 設計や開発だけではなく 運用も考慮する

このセッションでお伝えしたいこと

- なぜ、色々なメッセージングサービスがあるのか？自分にとって最適なものは何か？を見極めるポイント
- 「イベント駆動」をキーワードとした、メッセージングサービスに関する幾つかの視点
- (良くない意味での) プロダクトアウトに ならないための注意点

このセッションでお伝えしたいこと

aws DevAx connect

このセッションの主な対象者

システムの開発、設計に携わる方

- モダンなアプリやシステムを開発していきたい
- メッセージングサービスに興味がある、触れたことがある
- 概ね理解しているが、色んなサービスがあって整理できていない
- 昔から分散処理やイベントドリブンに触れてきたけど、最近少し混乱してきました

このセッションでは触れないこと

- 特定のメッセージングサービスにフォーカスした詳細な説明
- AWSが提供するサービスや機能における具体的な説明 (概念的なお話程度の予定)

AWSサービス資料

https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-service-cut/

1. 「疎結合」である必要性と根本的な動機を考えてみる

1. 「疎結合」である必要性と根本的な動機を考えてみる
2. メッセージングサービスの相違点
3. AWSのメッセージングサービスの適用とユースケース
4. 設計や開発だけではなく 運用も考慮する

aws DevAx connect

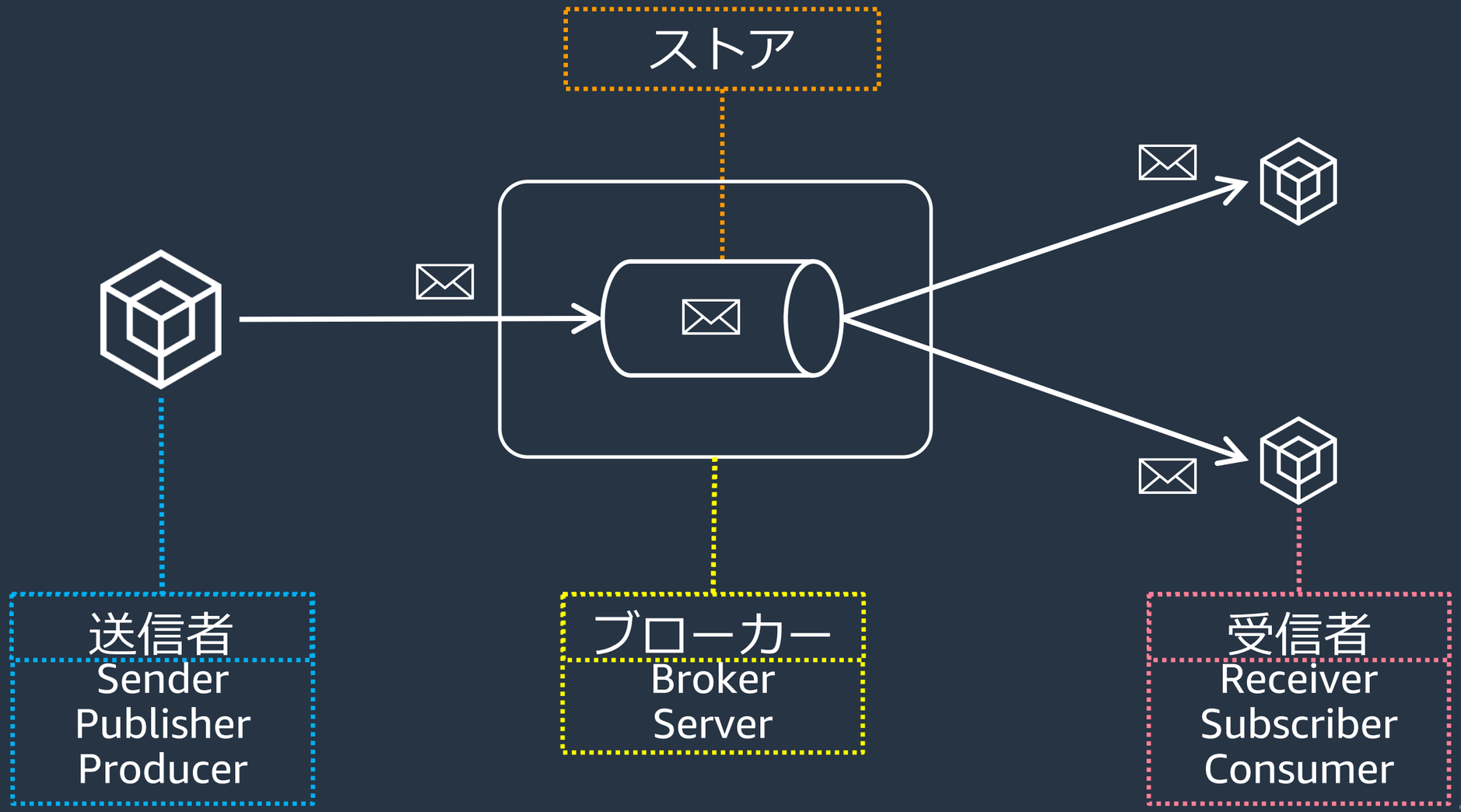
このセクションの話題

- メッセージングサービスとは？
- メッセージングサービスのモデル大別
- 疎結合にする動機を考えてみる
- メッセージングサービスの利用方法と弱点

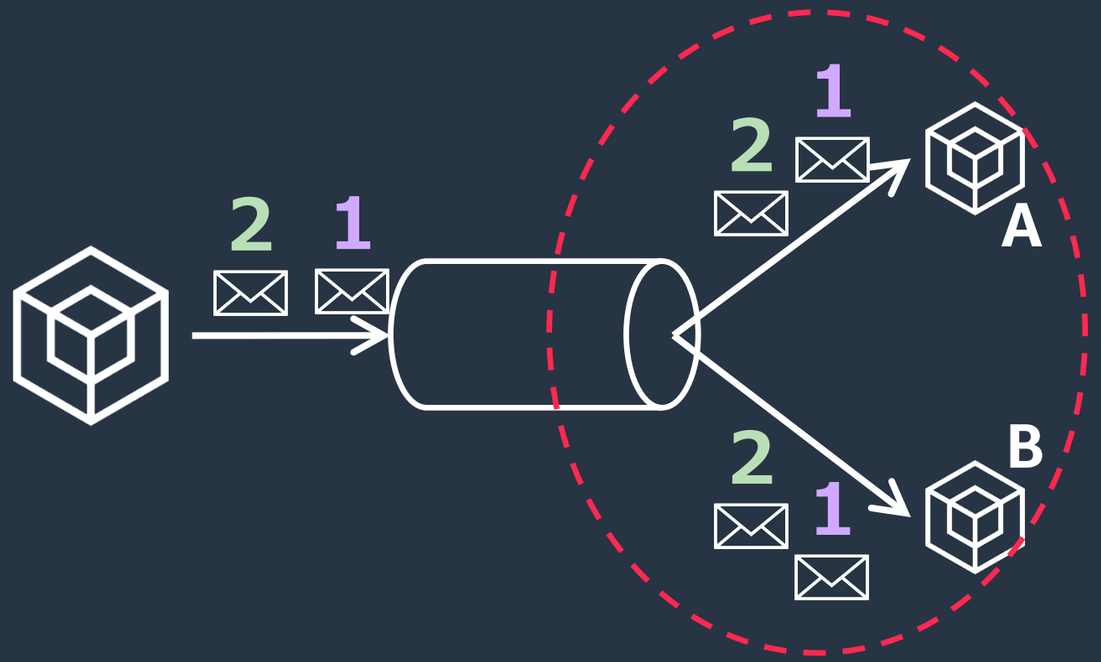
メッセージングサービスとは？

- 複数の異なるアプリケーション、システム、ドメインにおいて、(主に非同期で) 相互に通信を行うための1つの方法
- 様々なプログラミング言語やプラットフォームを超えて、相互にデータを交換できる基盤として動作
- データ形式やプロトコルの相互運用を実現するために、変換機能を有するものもある

メッセージングサービスとは？



ファンアウト



ブローカーは、

送信側からのメッセージを、

待ち受けているすべての受信側に
メッセージを配送

いわゆるブロードキャスト

メッセージングサービスの4つのメリット

1. パフォーマンスの向上

   送信側は受信側の処理を待たずに、受信側は使用可能なときに処理できる

2. 信頼性の向上

   データの一貫性を保ち、分離によって耐障害性を向上

3. 精度の高い拡張性

   送信側も受信側もブローカーも、ニーズに合わせたスケーリングを可能にする

4. 簡単な分散

   双方の依存関係を排除して、「分解したシステムの構築」を大幅に簡略化

aws DevAx connect

メッセージングサービスのモデル大別

モデル1: キュー

→ ポイントツーポイント、ワーカーキュー

モデル2: ストリーム

→ データストリーム、イベントストリーム

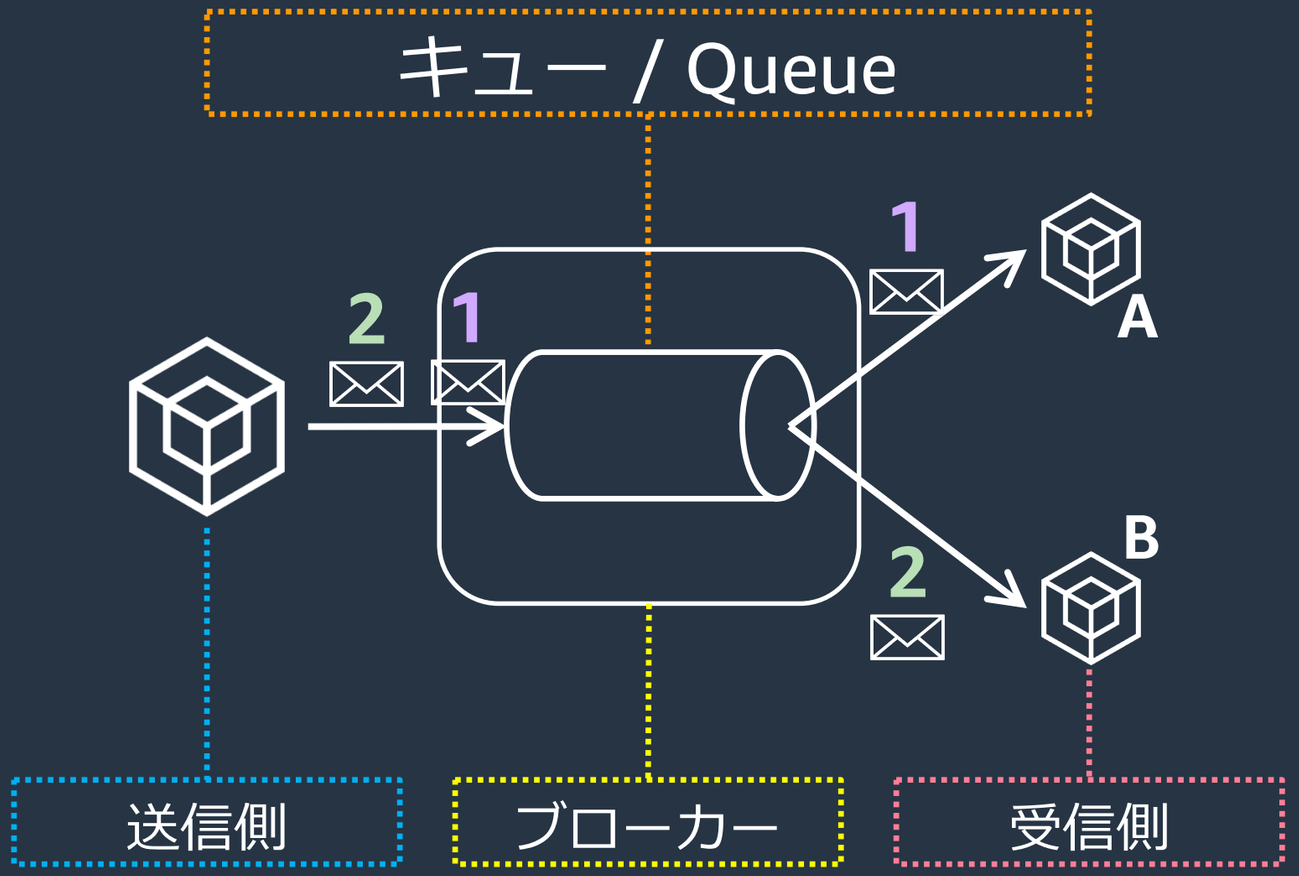
モデル3: トピック

→ ルーティング

モデル4: バス

→ ハブ、イベントバス、エンタープライズサービスバス (ESB)

モデル1: キュー



ブローカーは、

送信側からのメッセージを、

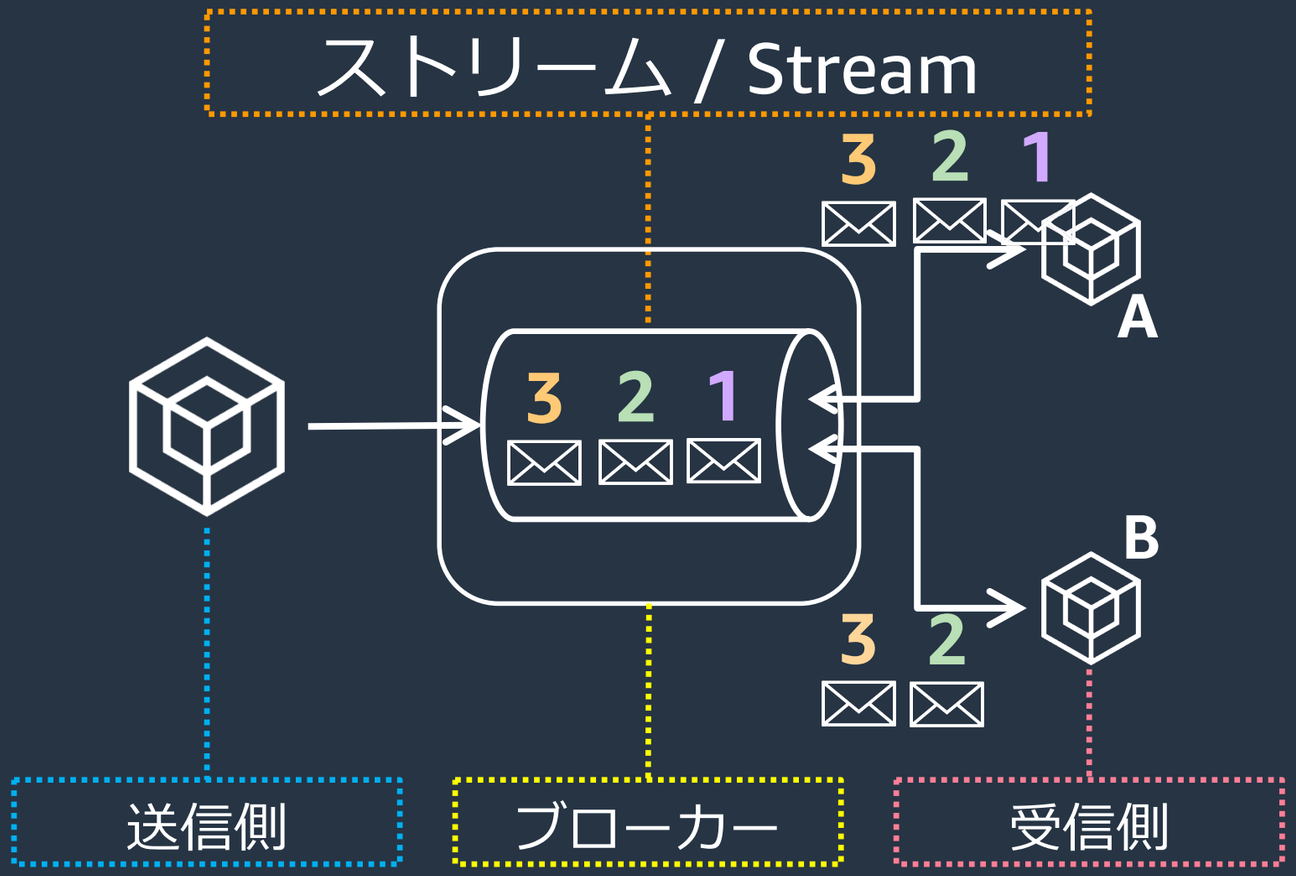
直列に蓄積 (待ち行列化) し、

受信側に配送し処理完了まで管理

待ちメッセージは処理可能な受信者に順次配信

→ イベント駆動など、送受信が独立し確実な配送が必要な場合

モデル2: ストリーム



ブローカーは送信側からのメッセージを、

時系列に蓄積し、ブローカーが受信側を意識せず、保持期間によって管理

メッセージの追跡は受信側の責任

→ ビッグデータの分散処理など、連続データを複数同時に処理する場合

キュー vs ストリーム

```
// キューの例 : amqplib (RabbitMQ)
var conn = await amqplib.connect("amqp://localhost:5672");
var channel = await conn.createChannel();
await channel.assertQueue("queue_name", { durable: true });

await channel.consume("queue_name", function (msg) {
  var ok = 何かの処理(msg);
  if (ok)
    channel.ack(msg);
  else
    channel.reject(msg, true);
});
```

メッセージの処理が正常終了か、異常終了かを
ブローカーに通知する

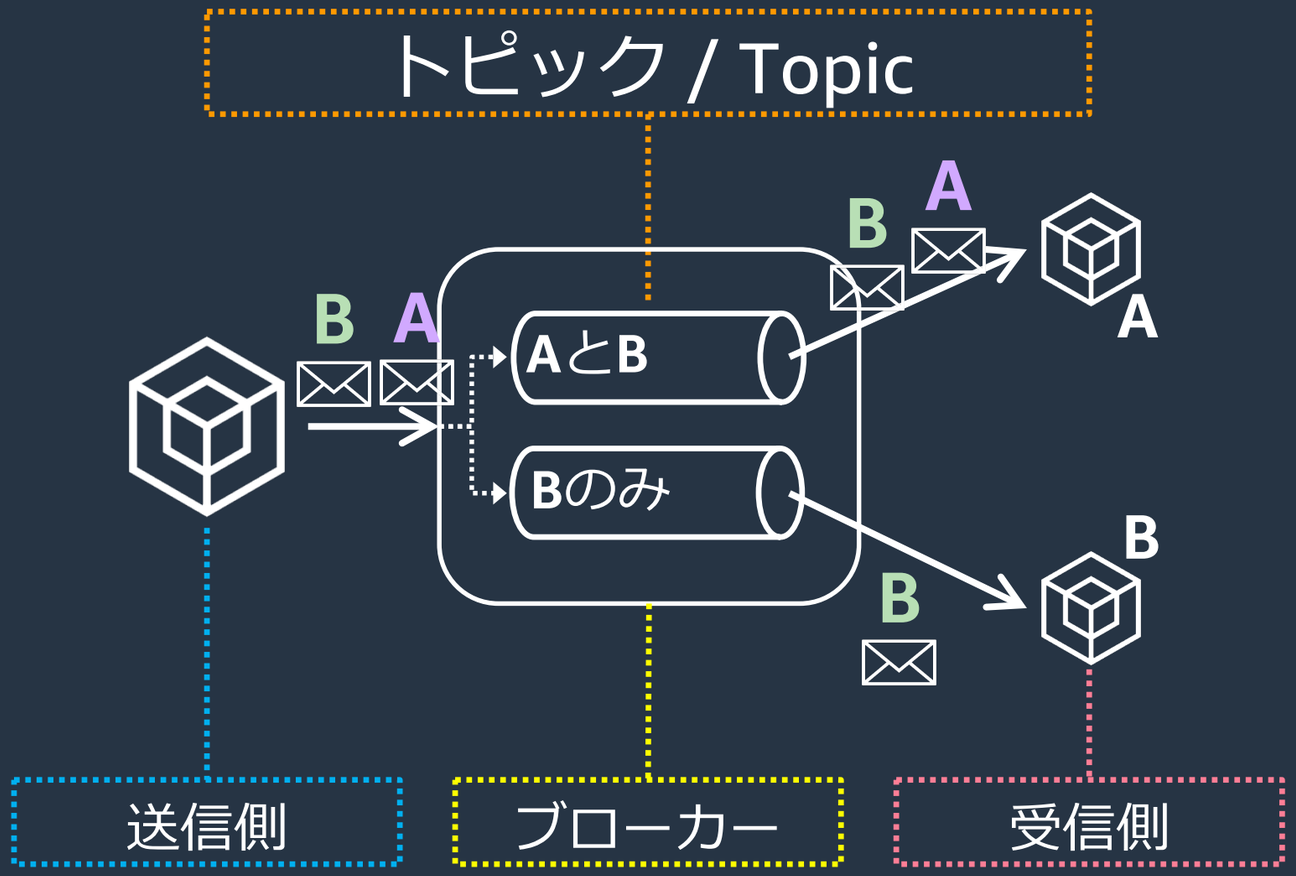
```
// ストリームの例 : kafka-node (Apache kafka)
var client = new kafka.KafkaClient({ kafkaHost: "localhost:9092" });
var consumer = new kafka.Consumer(client, [
  { topic: "topic_name", partition: 1, offset: 0 }
]);

consumer.on("message", function (msg) {
  何かの処理(msg);
});
```

← 読み出す位置 (offset) を指定

← メッセージ受信時は、処理に徹する

モデル3: トピック



ブローカーは、

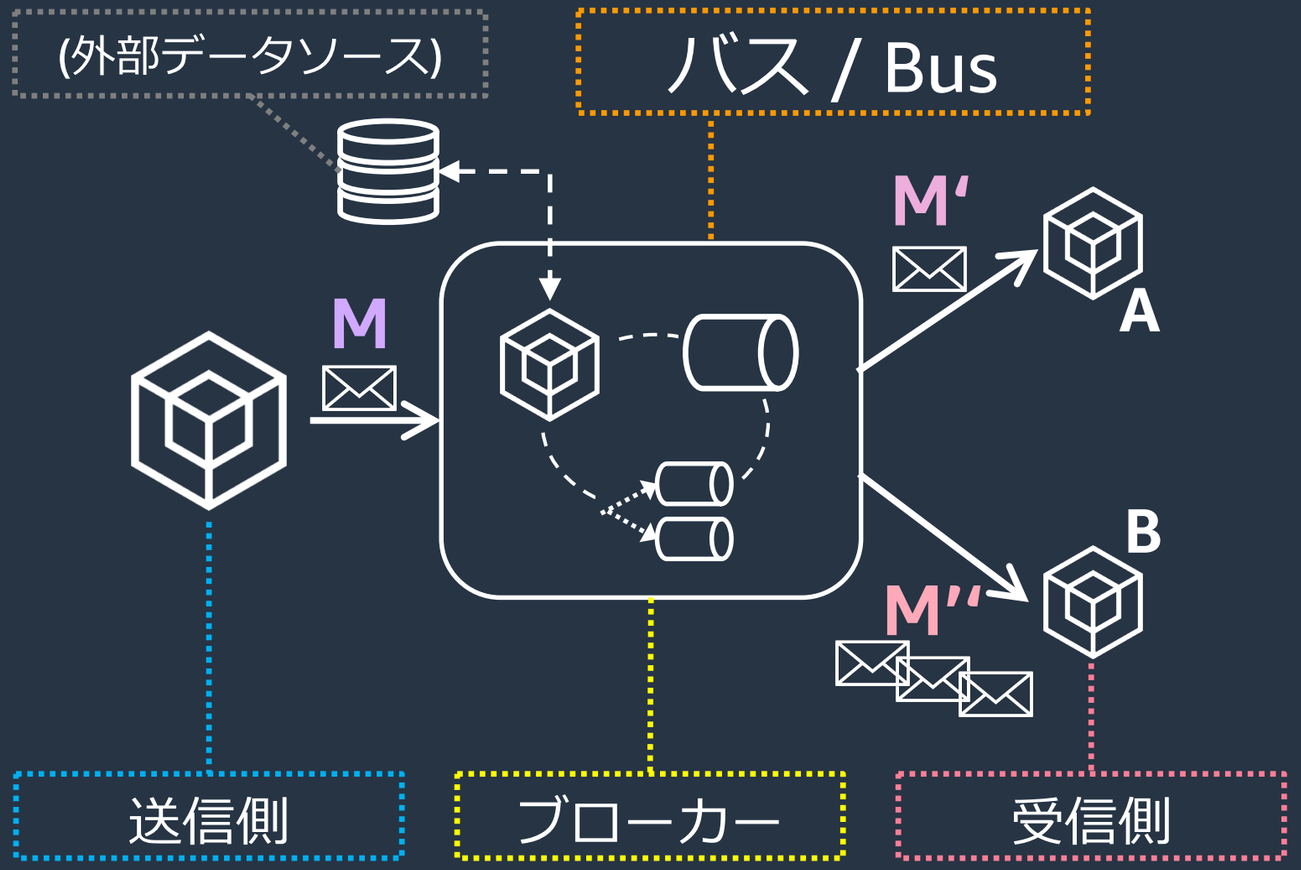
送信側からのメッセージを、

論理的な送信先にマップ/ルーティングし、

それぞれの受信側にメッセージを配送

→ 複数受信者のエイリアス、メッセージによる配送先分岐

モデル4: バス



ブローカーは、

(一時的な) データストアやルーティング、変換の仕組みを有し、

送信側からのメッセージを、受信側が期待するようなメッセージに加工して配信することが出来る

→ サービスやアプリケーション同士を相互接続するための仕組み

再掲

メッセージングサービスのモデル大別

モデル1: キュー

→ ポイントツーポイント、ワーカーキュー

モデル2: ストリーム

→ データストリーム、イベントストリーム

モデル3: トピック

→ ルーティング

モデル4: バス

→ ハブ、イベントバス、エンタープライズサービスバス (ESB)

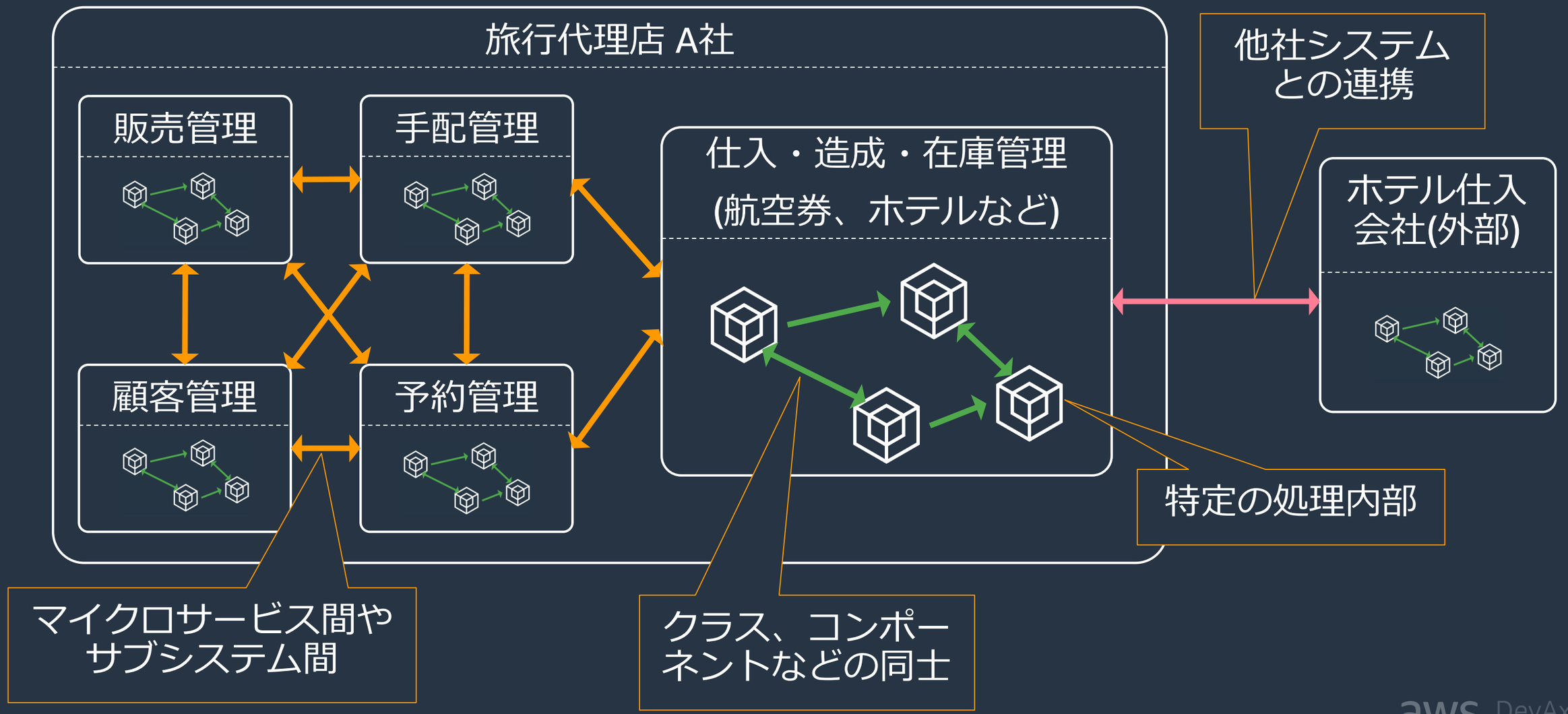
なぜ様々なモデルがあるのか？

- ブローカーの管理による、不達制御や期限切れ、配送優先度などの要否
- ExactlyOnceの実現やメッセージルーティング等の高度な機能よりも、とにかく高速配信であることが大事
- ブローカーの能力によって配信性能が影響されにくい構成の実現

など...

→ 単なる進化や改善だけでなく、それぞれにトレードオフがある場合も

何処を、疎結合にしたい？



何処を、疎結合にしたい？

- 特定の処理内部
- (プログラミング概念としての) オブジェクトやクラス、コンポーネントなどの抽象概念同士
- マイクロサービス間やサブシステム間
- IoTデバイスとネットワーク上のサーバーとの区間
- (自社内や 他社との) システム同士の連携

イベント駆動の範囲

→ 疎結合を期待するものの粒度には幅がある

→ 設計や開発の段階、責任範囲の影響

メッセージングサービスの利用方法

- クラウドサービスの利用
  - クラウド環境のマネージドサービスや、クラウドネイティブなサービスを利用する
  - クラウド環境の仮想サーバー環境に、 OSS やプロプライエタリなパッケージを導入
- ローカルサービス（サーバー）を構築
  - OSS やプロプライエタリなパッケージを、オンプレのサーバー環境に構築
- フレームワーク/ライブラリの導入
  - プログラミング言語やフレームワークが備えている機能を利用する

メッセージングサービスへの接続

- メッセージングサービスが提供するライブラリを利用
  - プログラミング言語毎の提供
  - AWS の場合は、SDK for C++ / Go / Java / JavaScript / .NET / PHP / Python(Boto3) / Ruby
- メッセージングサービスが対応するプロトコルを利用
  - AMQP や STOMP、MQTT などのクライアントを利用
- Webhook (REST API)
  - プログラミング言語やフレームワークが備えている機能を利用する

何故に、疎結合にしたい？

- コンポーネント間の切り離しで設計を容易にするため
- チームサイズ (機能やサービス毎の人員) を小さくしたい
- 容易なメンテナンスやテストを実現
- 負荷分散やスパイクの均一化を目指す
- 処理性能のスケーラビリティを享受
- 障害や性能低下時、全体的なダウンを避けるための緩衝を期待

→ 処理性能、設計、運用、組織など、動機は様々

メッセージングサービスの弱点

- 送信→受信 は一方通行

  → トランザクション処理の中で利用する場合は、例外時の補正トランザクション処理が必要
- 多用することによって、複雑で見通しの悪い結合になってしまう恐れ
- ブローカーの管理、運用の手間

  → マネージドサービスの利用
- セキュリティの問題 (騙したメッセージ、大量メッセージ投入)

  → 暗号化やクライアント認証を行う
- 処理において問題が発生した際のデバッグ、追跡、原因究明

  → ログを工夫する、AWS X-Ray を利用する

このセクションのまとめ

- メッセージングサービスとは？

  → 一般的なメッセージングサービスの概念とメリットのご紹介
- メッセージングサービスのモデル大別

  → キュー、ストリーム と、トピック、バス
- 疎結合にする動機を考えてみる

  → 何処をしたいのか？何故なのか？原点の動機は忘れないようにしておく
- メッセージングサービスの利用方法と弱点

  → 正しく理解して、対策や計画、準備をしておく

2. メッセージングサービスの相違点

1. 「疎結合」である必要性と根本的な動機を考えてみる
2. メッセージングサービスの相違点
3. AWSのメッセージングサービスの適用とユースケース
4. 設計や開発だけではなく 運用も考慮する

aws DevAx connect

このセクションの話題

- 様々なメッセージングサービス
- メッセージングサービスが有する特徴的な機能

様々なメッセージングサービス

OSS のメッセージングサービス

- Apache ActiveMQ
- RabbitMQ
- Apache Kafka
- Apache Pulsar
- ZeroMQ
- Mosquitto
- Redis

など...

様々なメッセージングサービス

AWS におけるメッセージングサービス

-  Amazon Simple Queue Service (Amazon SQS)
-  Amazon Simple Notification Service (Amazon SNS)
-  Amazon MQ (for ActiveMQ / for RabbitMQ)
-  Amazon Managed Streaming for Apache Kafka (Amazon MSK)
-  Amazon EventBridge
-  Amazon Kinesis
-  AWS IoT Core
-  Amazon Pinpoint

様々なメッセージングサービス

プロプライエタリなメッセージングサービス

- IBM MQ
- Microsoft Message Queue (MSMQ)
- Oracle Advanced Queuing (Oracle AQ)
- TP1/MessageQueue

など...

様々なメッセージングサービスの相違点

- 動作するメッセージングモデルの違い
- 受信側がメッセージを能動的に取得するか (Pull/Peak)、待受けるか (Subscribe) ?
- 永続化のサポート (メモリ、ディスク、DBMS など)
- ブローカー自体の冗長化や分散 ができるか ?
- サポートするQoS (At-least-once/Exactly-once/At-most-once)
- サポートするプロトコル
- ブローカー自体の有無 (ライブラリがその機能を有する、など)

メッセージングサービスが有する特徴的な機能

- メッセージの永続化
- 冗長化や分散のサポート
- メッセージの高度な配布、ルーティング
- セキュリティや認証
- サポートする通信プロトコル

メッセージの永続化

- メッセージをディスク、DBMS などに格納し、不測の再起動などが発生しても、処理が完了していないメッセージが失われない
- メモリ使用量が減る分、ディスクI/Oが増加してスループットが低下するリスクもある

(ディスクやDBなどに永続化する能力を備えているメッセージングサービスの例)

→ Apache ActiveMQ, RabbitMQ, Apache Kafka, Redis, Mosquitto , Apache Pulsar

→ AWS におけるメッセージングサービス全般

冗長化や分散のサポート

- キューの冗長化や負荷分散のよる、耐障害性の向上
- 複数ブローカーを動的に構成出来るか (システム稼働中の増減)
- 冗長化や分散は、メッセージ配信のスループットが低下する

(冗長化やシャーディングなどの分散をサポートするサービスの例)

→ Apache ActiveMQ, RabbitMQ, Apache Kafka, Apache Pulsar, Redis

→ AWS におけるメッセージングサービス全般

メッセージの高度な管理、ルーティング

- トピックの階層化やワイルドカードなどのマッチングなど、高度なルーティング用の仕組みの搭載
- デッドレターキュー、遅延キュー、メッセージ優先順位、キューの最大長制限、メッセージ単位のTTL設定などの機能
- 殆どの場合、動作速度とトレードオフ

(高度なメッセージング設定が可能なサービスの例)

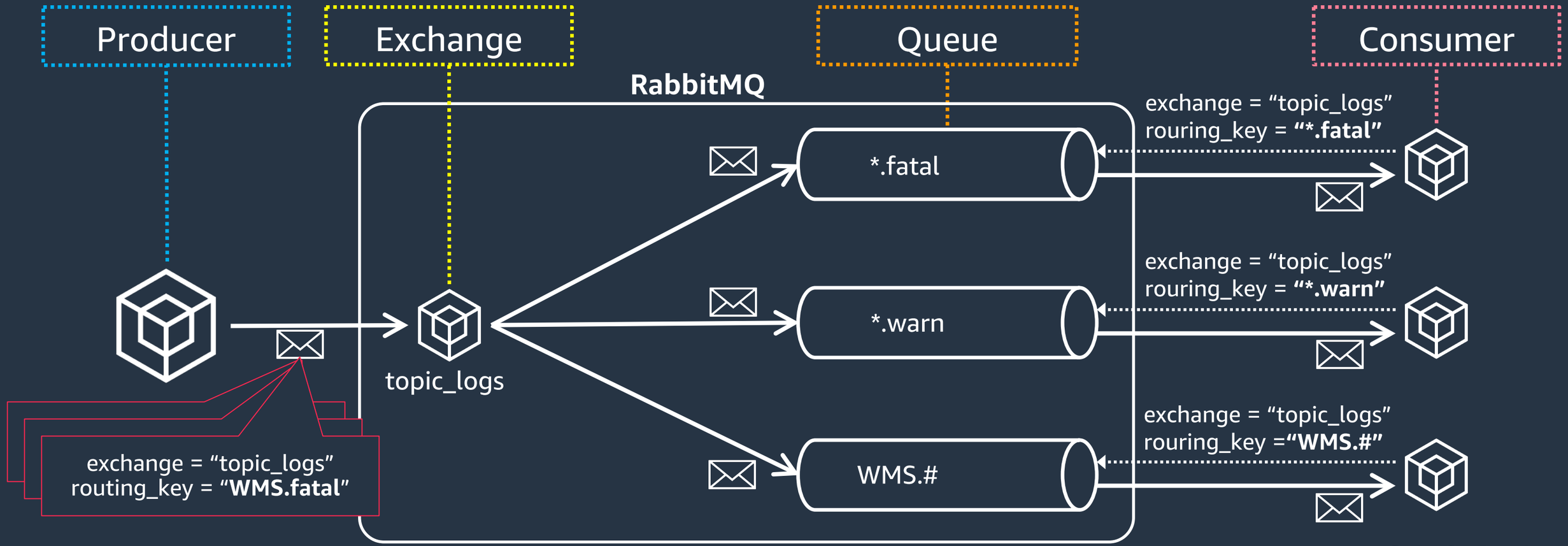
→ Apache ActiveMQ, RabbitMQ

→ Amazon MQ, Amazon EventBridge, AWS IoT Core, Amazon SNS

→ Amazon SQS (ルーティングを除く)

(参考) RabbitMQにおけるトピックルーティング

メッセージの形式: RoutingKey: “(system).(level)” + Body: “(Log message)”



セキュリティや認証

- サーバーに接続するクライアントやキュー単位での認証 (ID/パスワード、証明書、トークン、署名付き要求、環境における認証情報など)
- TLS/SSL などのトランスポートレベルの暗号化
- ネットワークポリシーによる制御

(高度なセキュリティ設定が可能なサービスの例)

→ Apache ActiveMQ, RabbitMQ, Apache Kafka , Apache Pulsar

→ AWS のメッセージングサービス全般 (IAMやアクセスポリシー設定による)

サポートする通信プロトコル

- AMQP、STOMP、MQTT、HTTP、JMS、独自方式 ...
- TCP，WebSocket (xxx over WebSocket)

(AMQPをサポートするサービスの例)

→ Amazon MQ, Apache ActiveMQ, RabbitMQ, Apache Pulsar

(MQTTをサポートするサービスの例)

→ Amazon MQ, AWS IoT Core, Mosquitto

(JMSをサポートするサービスの例)

→ Amazon SQS, Amazon MQ , Apache MSK, Apache ActiveMQ, RabbitMQ, Apache Kafka

3. AWSのメッセージングサービスの適用とユースケース

1. 「疎結合」である必要性と根本的な動機を考えてみる
2. メッセージングサービスの相違点
3. AWSのメッセージングサービスの適用とユースケース
4. 設計や開発だけではなく 運用も考慮する

aws DevAx connect

このセクションの話題

- AWSメッセージングサービスの整理
- メッセージングサービスにおけるTips

AWSのメッセージングとイベントサービス

第一回資料より

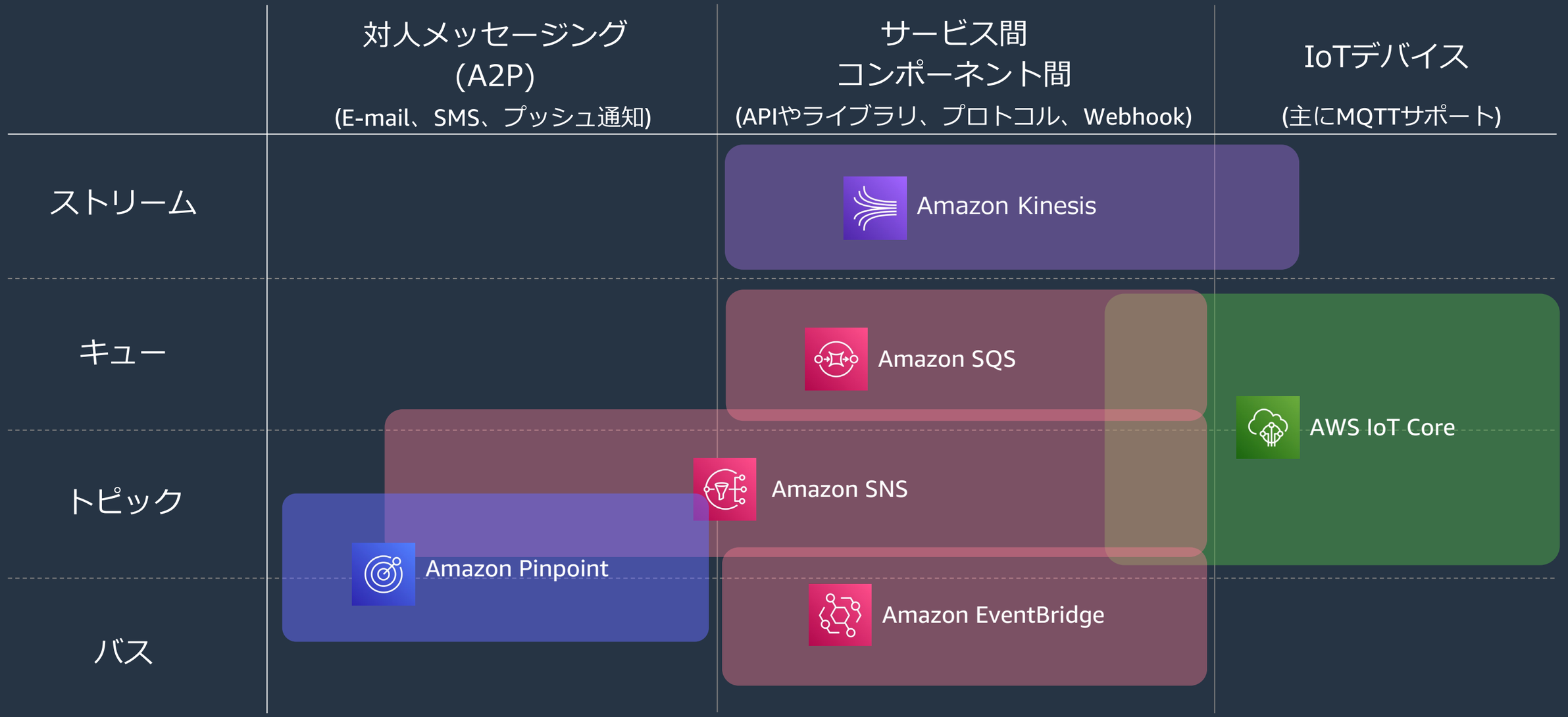
	イベントストア		イベントルーター	
	キュー	ストリーム	トピック	イベントバス
AWSネイティブ	 Amazon SQS	 Amazon Kinesis	 Amazon SNS	 Amazon EventBridge
マネージドオープンソース(マネージドOSS)	 Amazon MQ	 Amazon MSK	 Amazon MQ	

クラウド環境に、より最適化されたメッセージングサービス

OSSを、AWS環境でマネージドサービス化したもの

aws DevAx

AWS メッセージングサービスの整理 (AWSネイティブ)



AWS メッセージングサービスの整理 (マネージドOSS)

	対人メッセージング (A2P) (E-mail、SMS、プッシュ通知)	サービス間 コンポーネント間 (APIやライブラリ、プロトコル、Webhook)	IoTデバイス (主にMQTTサポート)
ストリーム		Amazon MSK	
キュー			
トピック		Amazon MQ	
バス			

aws DevAx connect

配信セマンティクス

第一回資料より

At-least once 配信

イベントは、ターゲットに複数回配信されることも。処理されたイベントの状態を追跡することによって重複イベントを検出するロジックを含める(冪等性)

- Amazon EventBridge
- Amazon SNS 標準トピック
- Amazon SQS 標準 キュー

Exactly-once 配信*

AWSサービスに重複排除に使用する識別子を渡すことで重複排除

- Amazon Kinesis
- Amazon SNS FIFO トピック
- Amazon SQS FIFO キュー

* Lambdaでリトライ設定している場合は、At-least once

aws DevAx

順序セマンティクス*

第一回資料より

順序保証なし

イベント/メッセージが順序保証なしで配信

- Amazon EventBridge
- Amazon SNS 標準トピック
- Amazon SQS 標準 キュー

順序保証

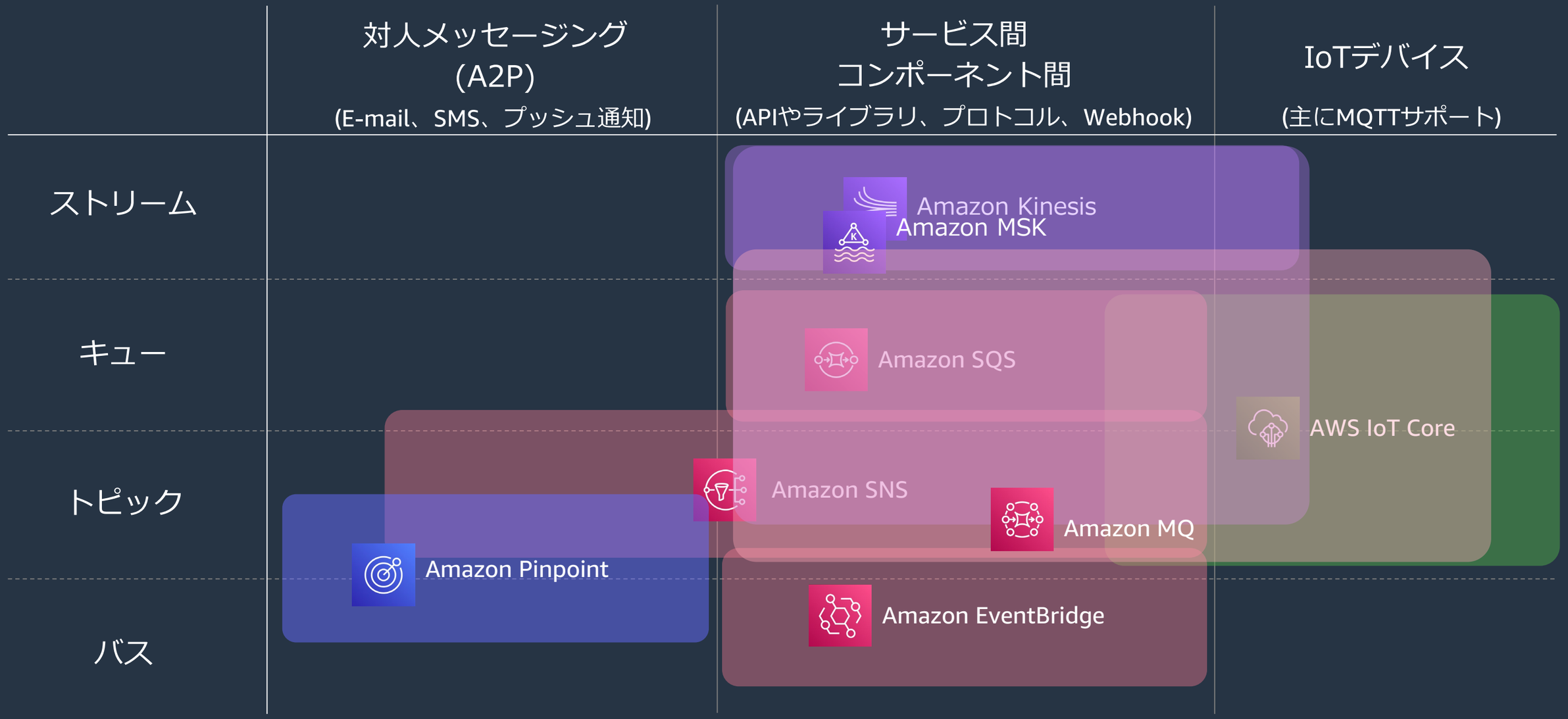
イベントは、パーティション、メッセージグループなどの中で順番に配信
（グローバルな順序保証はなし）

- Amazon Kinesis
- Amazon SNS FIFO トピック
- Amazon SQS FIFO キュー

＊ 順不同のイベント処理ロジックが受入可能かは、ビジネス要件次第。順不動なイベントを処理するようにアプリケーションを設計できない場合は、代わりにオーケストレーションも検討

aws DevAx

AWS メッセージングサービスの整理 (全体)



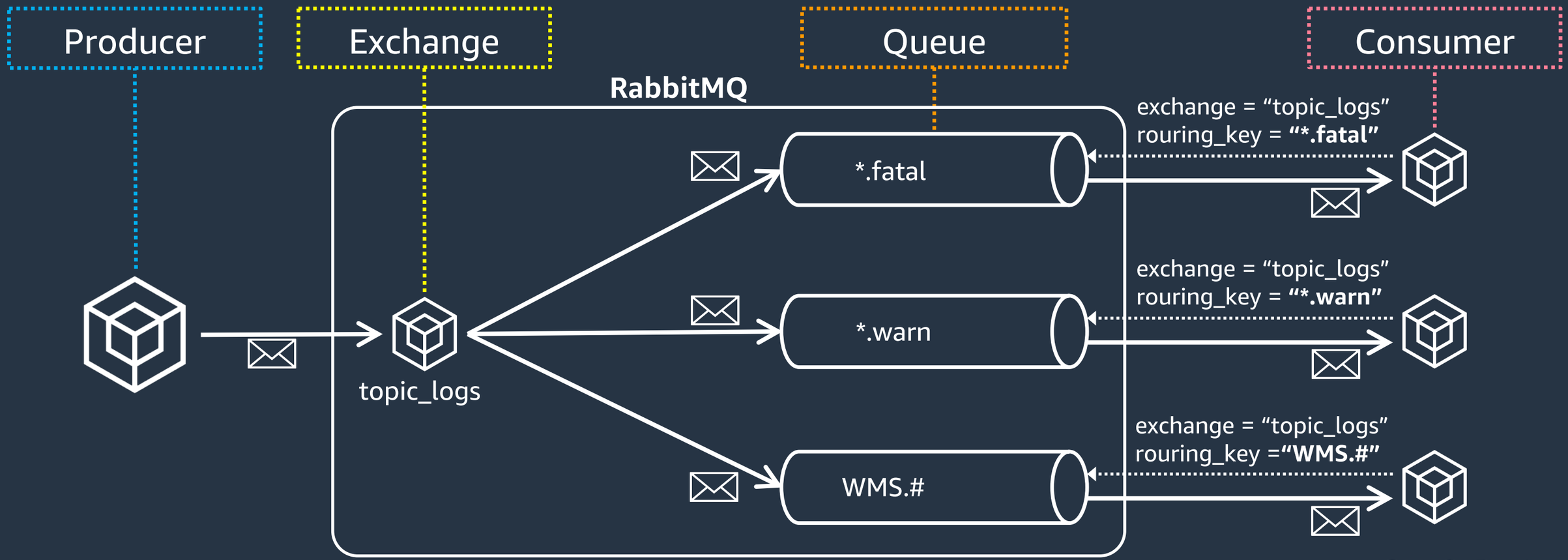
aws DevAx connect

メッセージングサービスにおけるTips

- Amazon SQS と Amazon SNS の違い
- At-least once の送受信処理 (その1)
- At-least once の送受信処理 (その2)
- 対人メッセージング(A2P): Amazon SNS と Amazon Pinpoint
- IoT でのメッセージング: 限定的な通信帯域で、サイズの小さい大量のメッセージを処理する

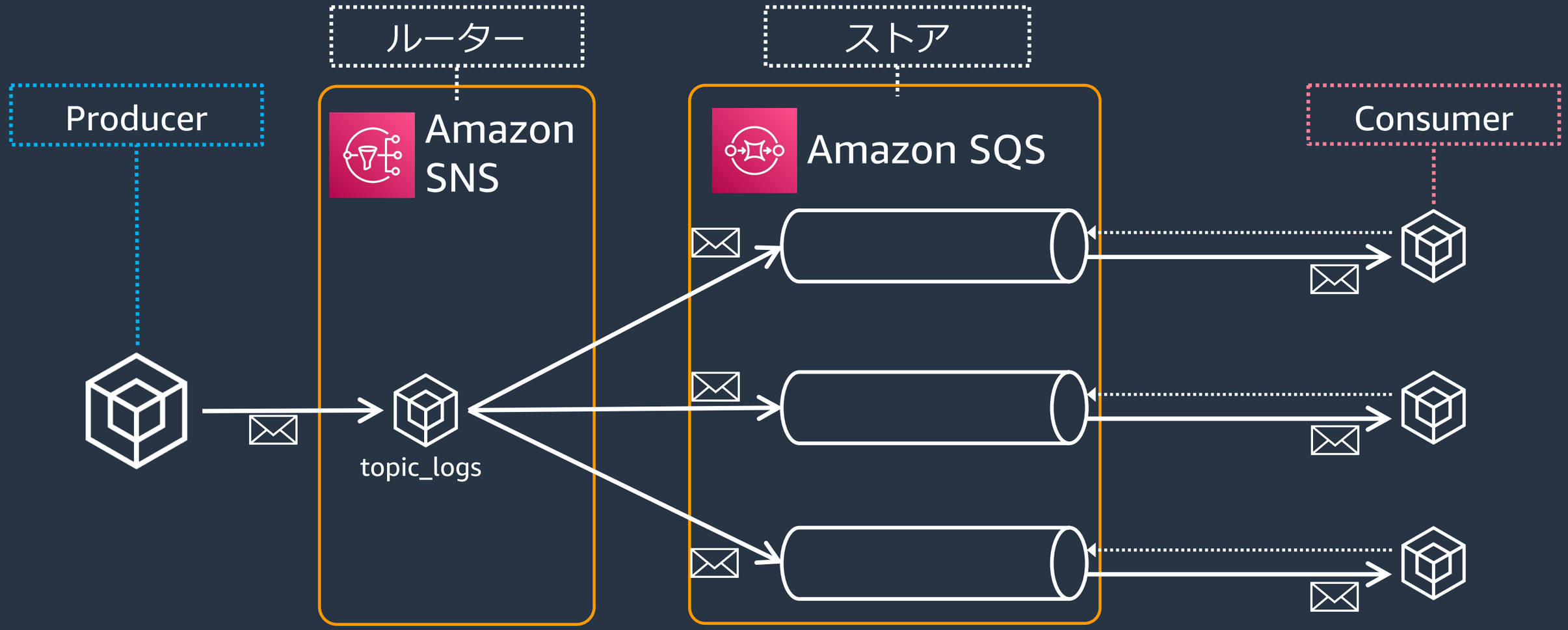
再掲

Amazon SQS と Amazon SNS の違い: RabbitMQにおけるトピックルーティング



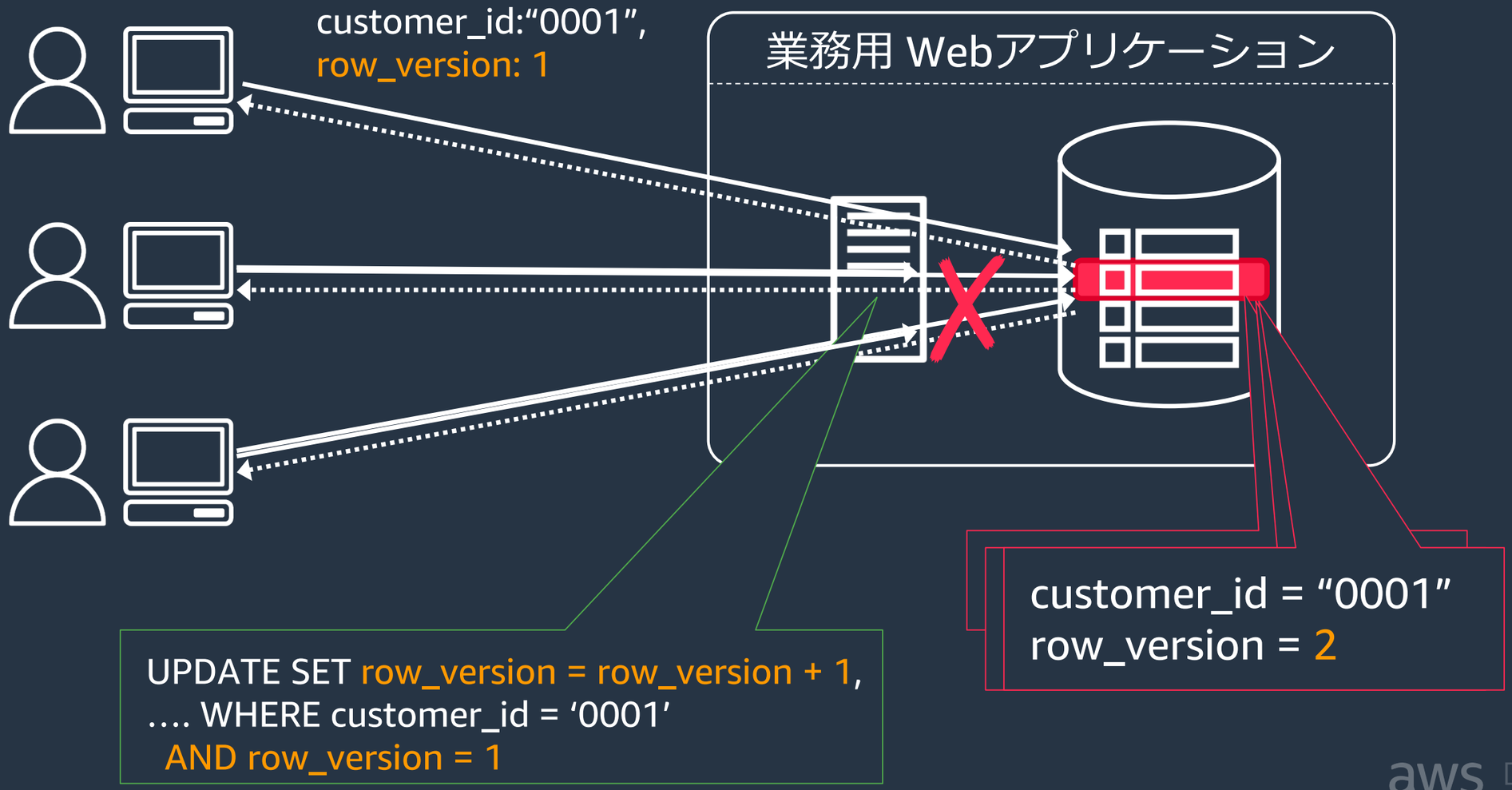
aws DevAx connect

Amazon SQS と Amazon SNS の違い:
Amazon SNS と Amazon SQS で表現

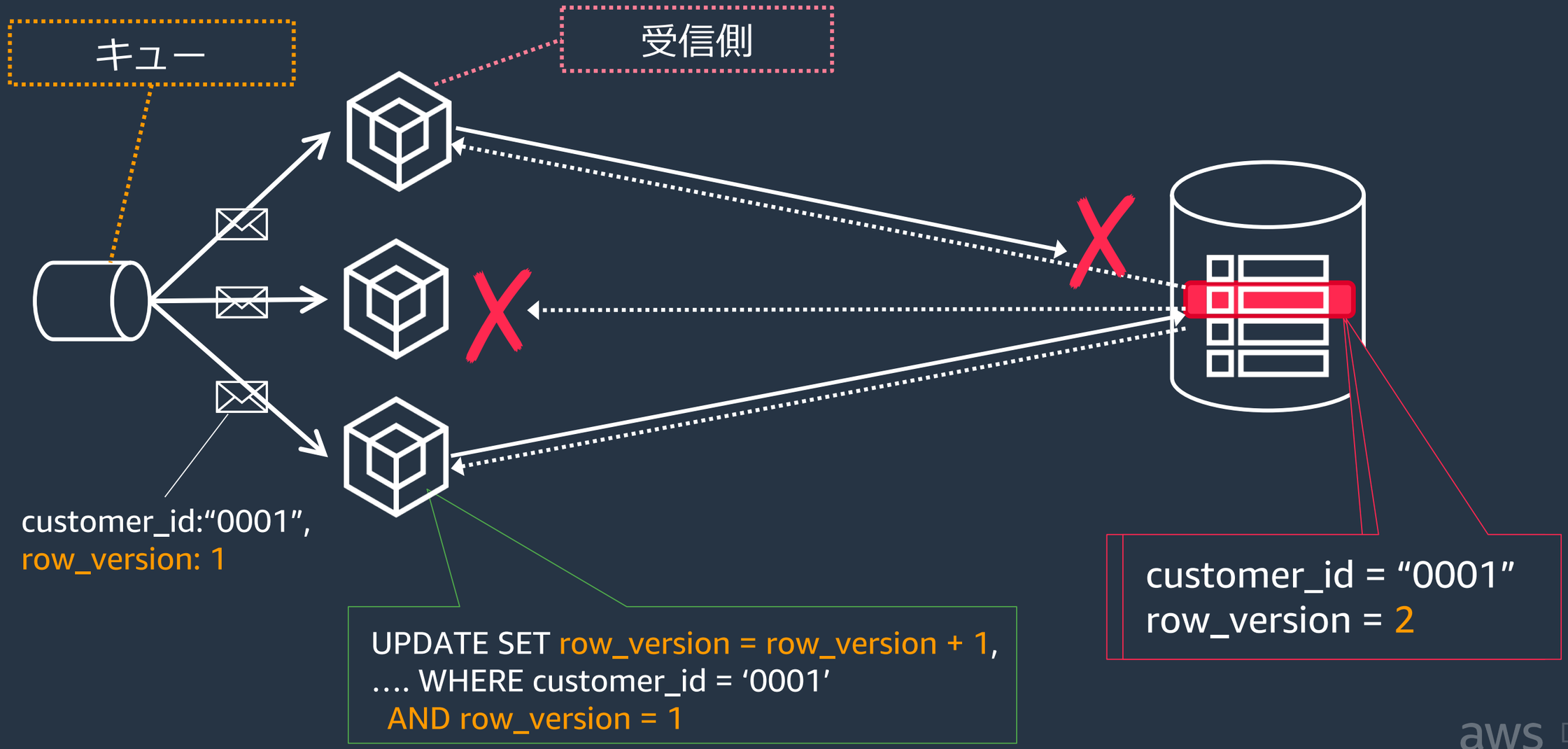


At-least once の送受信処理 (その1)

Webアプリケーションにおけるカジュアルな排他制御

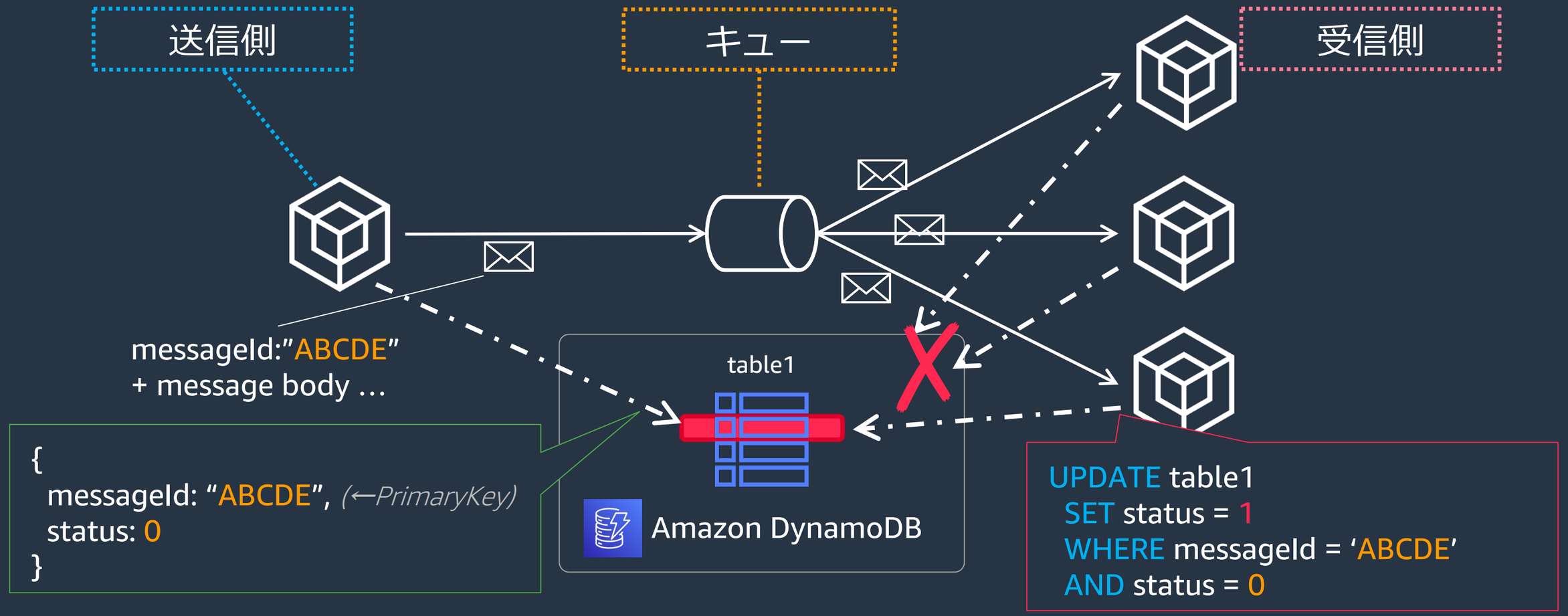


At-least once の送受信処理 (その1)

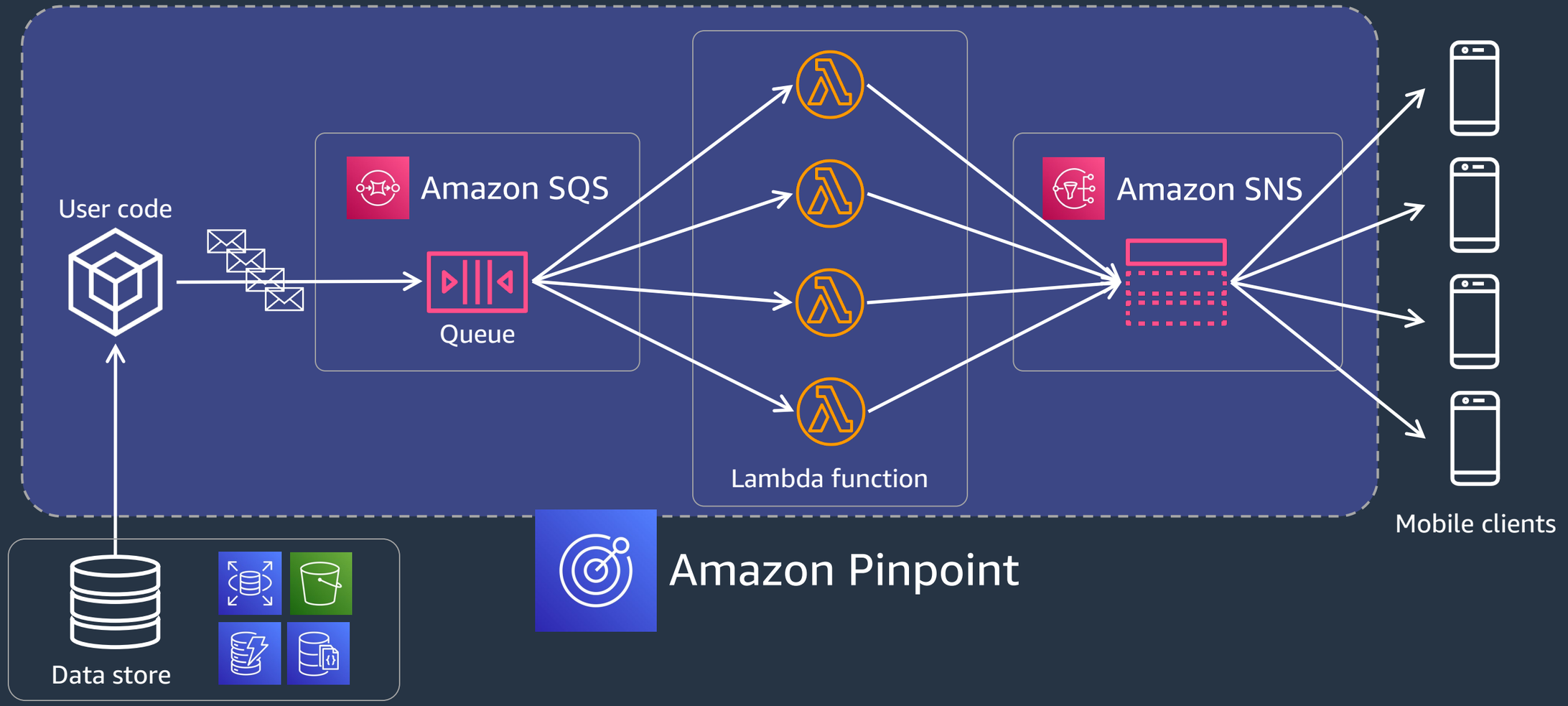


At-least once の送受信処理 (その2)

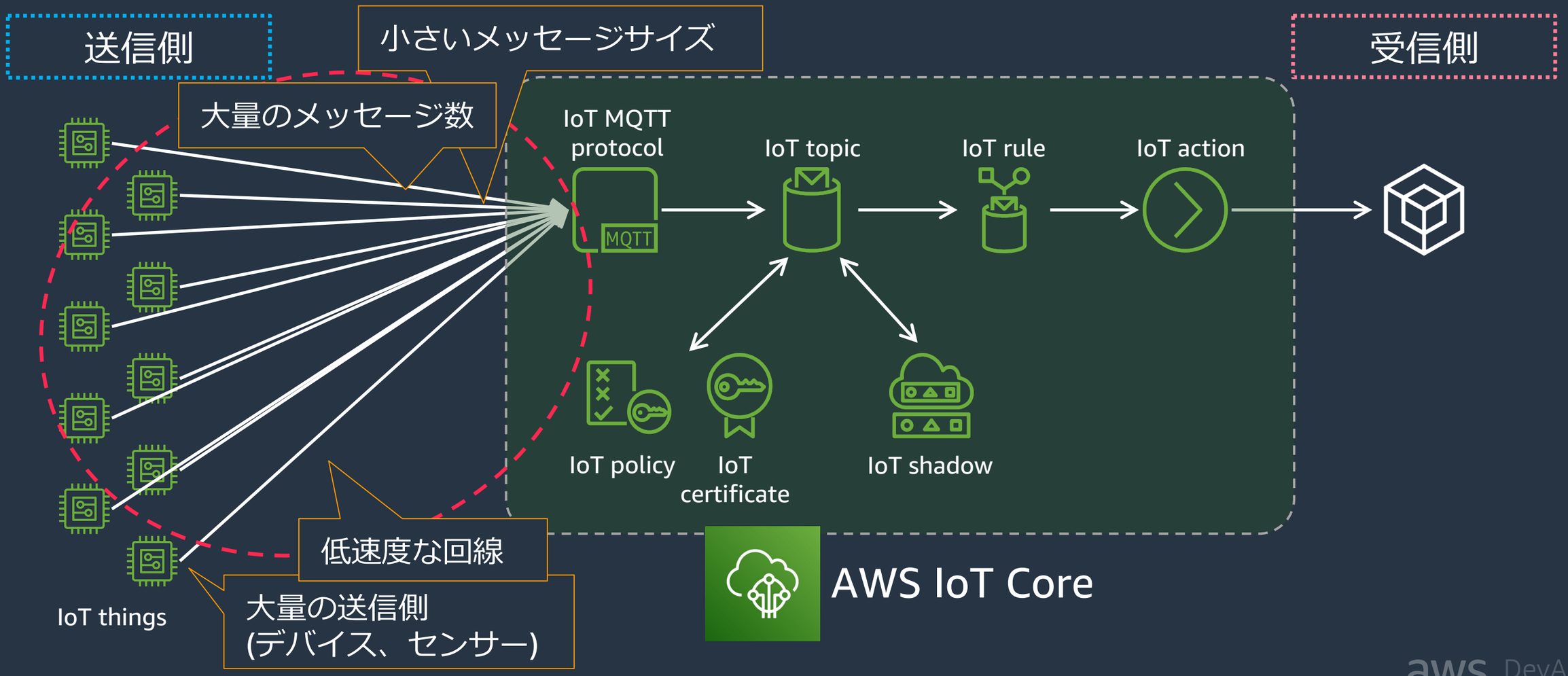
DynamoDB を利用した排他制御



対人メッセージング(A2P): Amazon SNS と Amazon Pinpoint

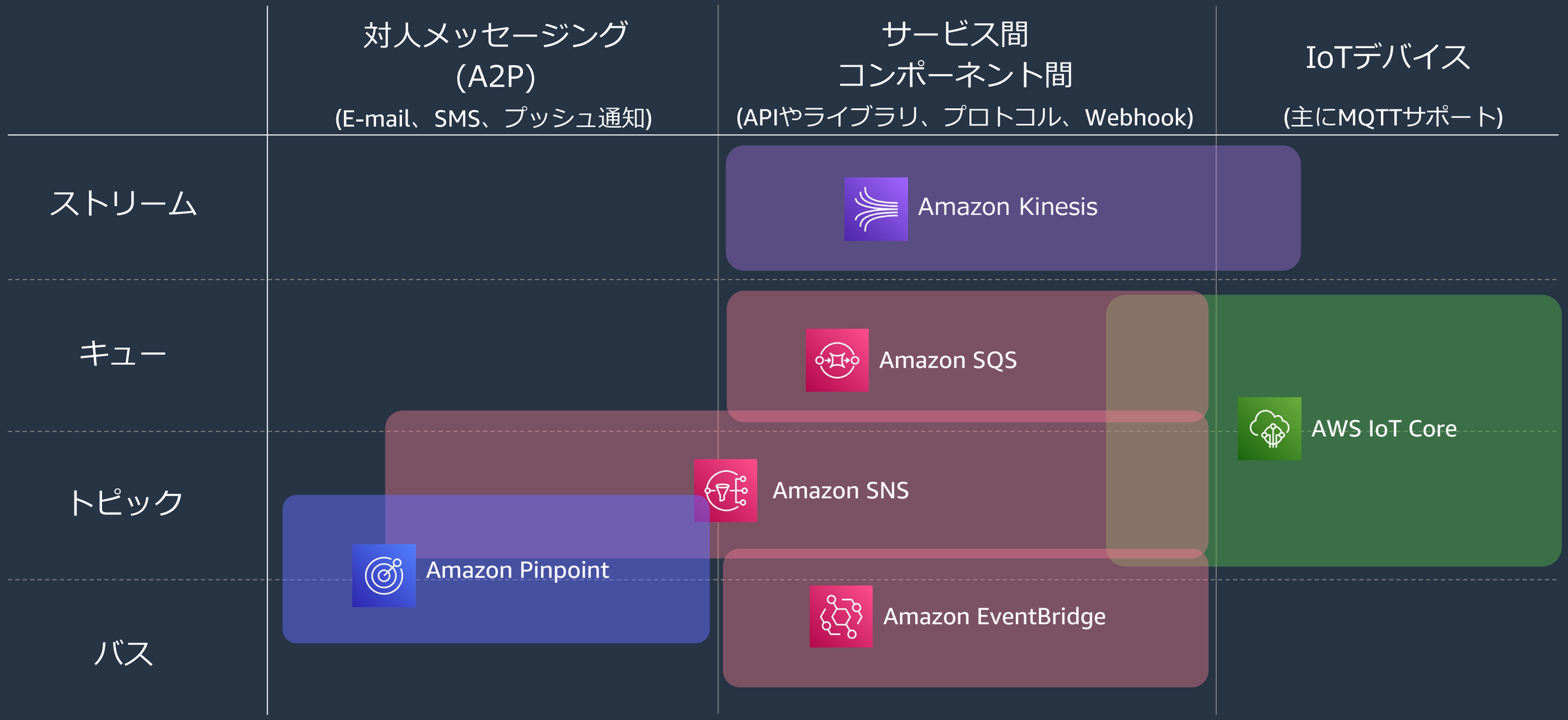


IoT でのメッセージング: 限定的な通信帯域で、サイズの小さい大量のメッセージを処理する



再掲

AWS メッセージングサービスの整理 (AWSネイティブ)



4. 設計や開発だけではなく運用も考慮する

1. 「疎結合」である必要性と根本的な動機を考えてみる
2. メッセージングサービスの相違点
3. AWSのメッセージングサービスの適用とユースケース
4. 設計や開発だけではなく 運用も考慮する

aws DevAx connect

このセクションの話題

- 「付加価値を生まない重労働」
- オンプレOSS？マネージドOSS？AWSネイティブ？
- メッセージングサービスの多重化によるループに注意

「付加価値を生まない重労働」

イベント駆動としてのメッセージングサービスの利用において、

メッセージ (イベント) の配送と、受信側における確実な処理実行は最重要

- メッセージングサービスにおける耐障害性向上はパフォーマンスや運用負担とトレードオフ

→ マネージドサービス(AWSネイティブ、マネージドOSS) を利用して、不要な運用負担を避ける

オンプレや EC2 に OSS のメッセージングを導入

メリット

- マネージドOSSではサポートしていないプラグインや拡張の導入
- マネージドOSSではサポートしていないバージョンの利用

デメリット

- 自身で運用管理する必要がある
- 冗長化、分散等の設定、適合するネットワークの構築

マネージド OSS のメッセージングを導入

メリット

- OSS ならではの機能と弾力性、柔軟性の享受
- (既存の利用がある場合) コード変更が殆ど必要ない、素早い移行

デメリット

- マネージド環境特有の設定や運用知識＋OSSパッケージの知識
- AWS ネイティブのメッセージングに比べて複雑な機能
- 高可用性にする場合、Multi-AZ 構成などの構築が必要

AWS ネイティブのメッセージングを導入

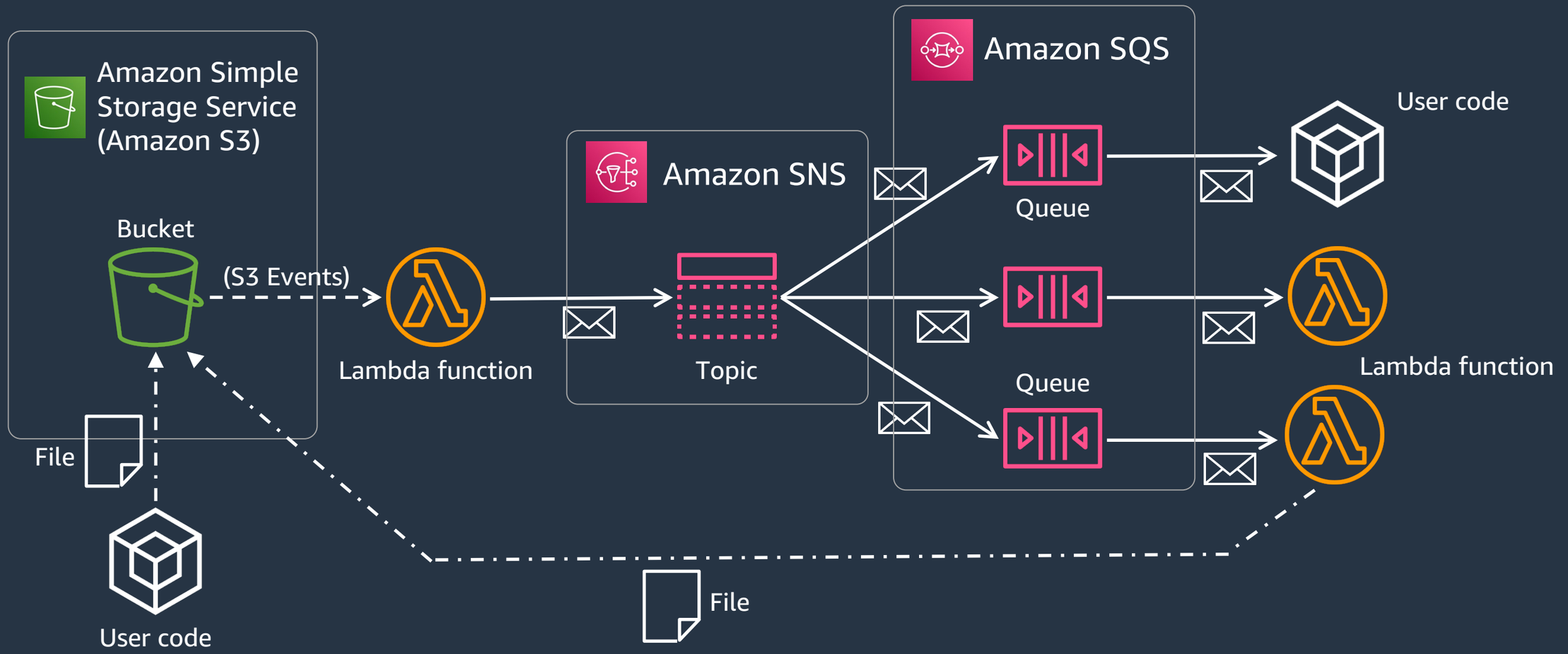
メリット

- 冗長化や分散構成を意識しないスケーラビリティ
- サーバーレス
- 従量課金、シンプルな機能

デメリット

- メッセージングサービス自体を AWS 環境以外に持ち出せない (完全に一致する代替が無い) ← AWS 外からの接続・利用は可能

メッセージングの多重化によるループに注意



まとめ

aws DevAx connect

再掲

セクション1 のまとめ

- メッセージングサービスとは？

  → 一般的なメッセージングサービスの概念とメリットのご紹介
- メッセージングサービスのモデル大別

  → キュー、ストリーム と、トピック、バス
- 疎結合にする動機を考えてみる

  → 何処をしたいのか？何故なのか？原点の動機は忘れないようにしておく
- メッセージングサービスの利用方法と弱点

  → 正しく理解して、対策や計画、準備をしておく

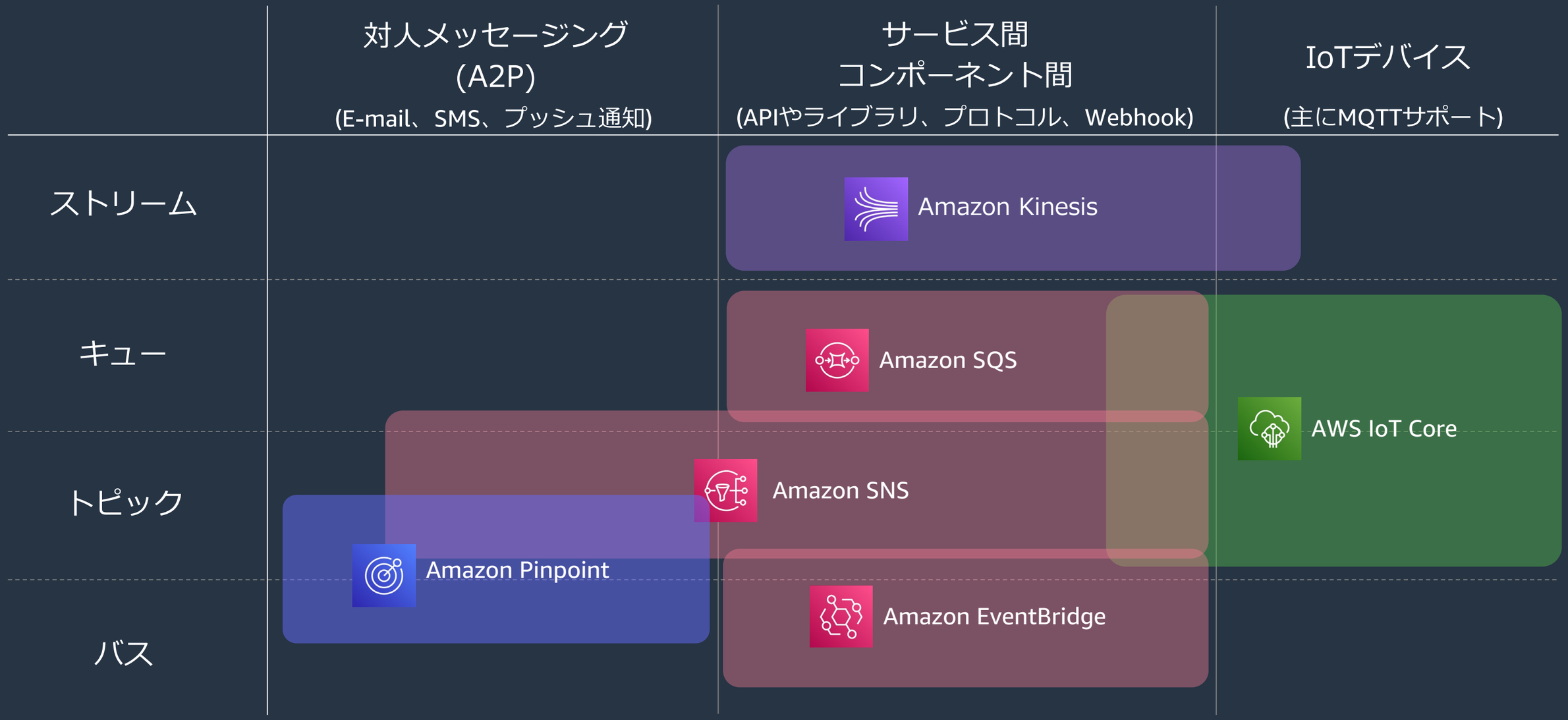
様々なメッセージングサービスの相違点

再掲

- 動作するメッセージングモデルの違い
- 受信側がメッセージを能動的に取得するか (Pull/Peak)、待受けるか (Subscribe) ?
- 永続化のサポート (メモリ、ディスク、DBMS など)
- ブローカー自体の冗長化や分散 ができるか ?
- サポートするQoS (At-least-once/Exactly-once/At-most-once)
- サポートするプロトコル
- ブローカー自体の有無 (ライブラリがその機能を有する、など)

再掲

AWS メッセージングサービスの整理 (AWSネイティブ)



再掲

AWS メッセージングサービスの整理 (マネージドOSS)

	対人メッセージング (A2P) (E-mail、SMS、プッシュ通知)	サービス間 コンポーネント間 (APIやライブラリ、プロトコル、Webhook)	IoTデバイス (主にMQTTサポート)
ストリーム		Amazon MSK	
キュー			
トピック		Amazon MQ	
バス			

aws DevAx connect

「付加価値を生まない重労働」

再掲

イベント駆動としてのメッセージングサービスの利用において、

メッセージ (イベント) の配送と、受信側における確実な処理実行は最重要

- メッセージングサービスにおける耐障害性向上はパフォーマンスや運用負担とトレードオフ

→ マネージドサービス(AWSネイティブ、マネージドOSS) を利用して、不要な運用負担を避ける

まとめ

サービスの選定方法 ≒ 求める要件の優先順位

- メッセージングの概念を理解し、導入したい動機をよく考えてみる
- 製品比較の ○ (マル) が多いものではなく、要件を満たすものかどうか
- PoC を十分に行う、特にスループットや障害発生時の動作確認など
- 運用やトラブル時の手順、シナリオを確認しておく
- 付加価値を生まない重労働は避けるべき

AWS Startup Loft 提供メニュー

Loft 入場時に利用する
ウェブアプリケーション

AWS アカウント ID 必須

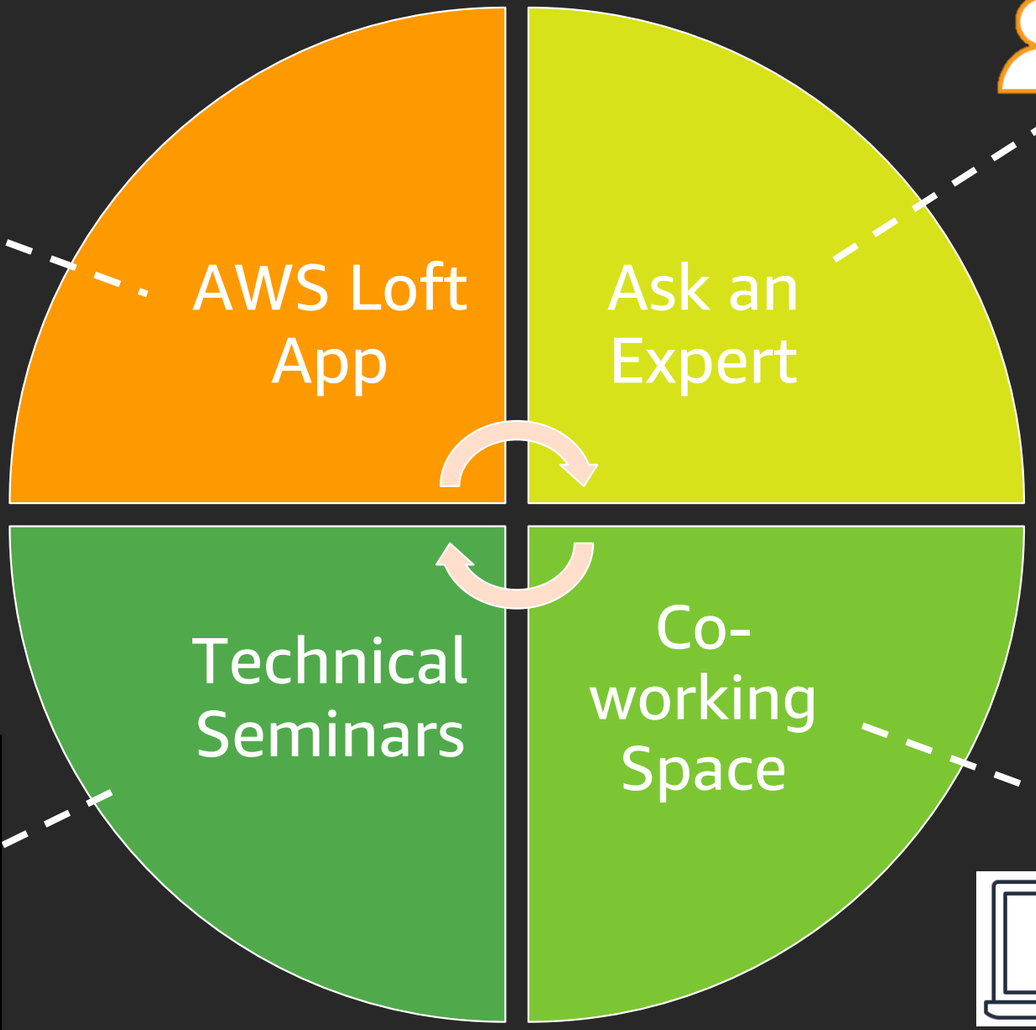
イベント情報なども確認可能

AWS に関する技術的な相談が可能

AWS アカウント ID 必須

Loft App にご登録された方は、
Online Ask an Expert をご利用可能

相談例は[こちらのブログより](#)ご覧ください



平日夜間 18:00-21:00 に実施

- セミナー
- ワークショップ
- Meet-up

etc

ウェビナーで実施中

平日10:00-18:00 に利用可能

- 電話ボックス（2台）
- 会議室（1室）
- Wi-Fi
- 電源

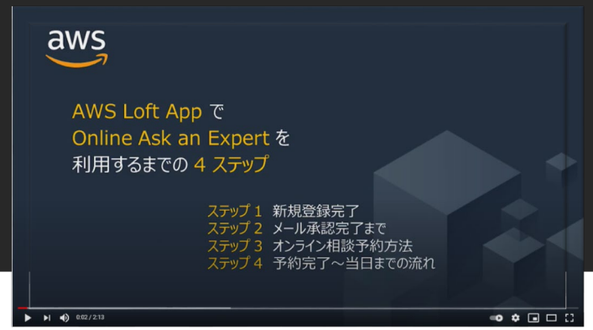
AWS アカウント ID 必須

2021 年 6 月現在休館中

aws DevAx connect

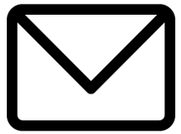
Ask an Expert 利用の 4 つのステップ

登録方法を動画で見る場合はこちら



https://awsloft.tokyo
へアクセス

承認申請



Loft App に
ログイン＆予約

予約日に相談

aws DevAx connect

Thank you!

Shingo Sugimoto
Solutions Architect, Industry Solutions
Amazon Web Services Japan K.K.
sugishin@amazon.co.jp

aws DevAx connect