DevAx
connect

How to Test your Events?

金森政雄
DevAx ソリューションアーキテクト
Amazon Web Services Japan K,K
kanamasa@amazon.co.jp

アジェンダ

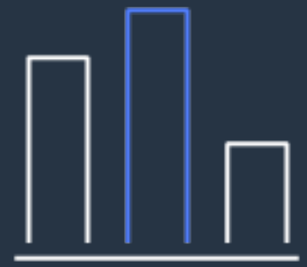
1. テストとは
2. イベント駆動におけるテストの難しさ
3. 分散システムにおけるテストの考え方とテクニック
4. まとめ

1.テストとは

なぜテストが必要か

作成したシステムに不具合があると、、、

コスト
(経済的/時間的)



機会の損失

信頼の失墜

重大な事故

テストはビジネスを成功させるためにある

- 要件レビューによる要件の正確性向上、余計な工数の排除
- テストしやすい設計による拡張性の高いソフトウェア
- 不具合の少ないソフトウェアによりビジネスの成功の可能性をアップ

だから制約もある

テストにおける制約

- 時間的制約
- リソース的制約
    - テスト人員
    - テスト環境
    - ソフトウェア的制約

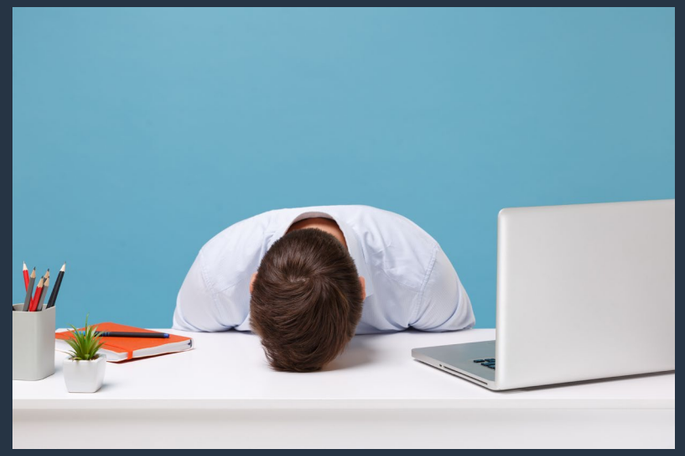
→**「いかに効率よく重要な欠陥を見つけるか」**
がテスト成功の鍵

我々開発者にとっては、

○ テストがある

- フィードバックを早く得る
- 安心して/自信を持って開発できる

✕ テストがない/遅い/不安定



- 不具合が後からわかる
- 正しく動くか不安

はやく、正確に、簡単に動く

テストが必要

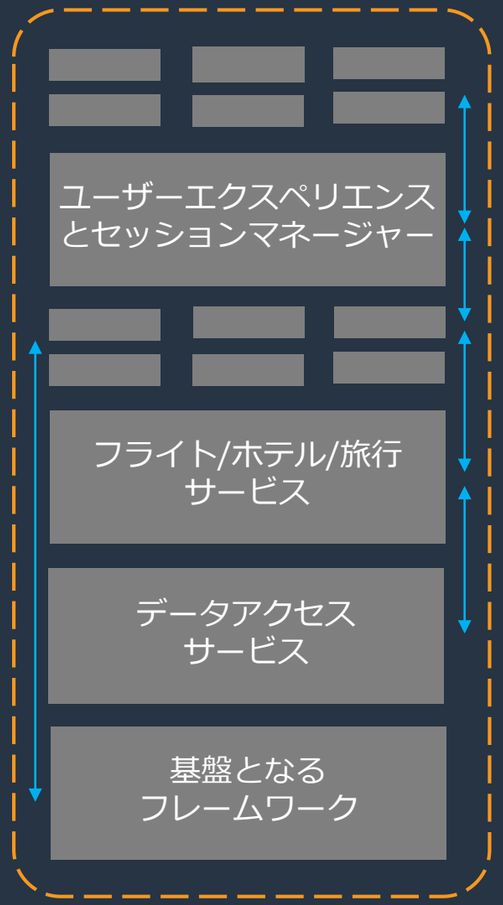
「テスト」と言う言葉は幅が広い

本日のセッションでは、システムが**期待する機能を提供することを確認する行為**を「テスト」とします。

システムにおいて、単体テスト、結合テスト、受入テストなどと呼ばれるテストを想定します。(非機能は含みません)

2.イベント駆動における テストの難しさ

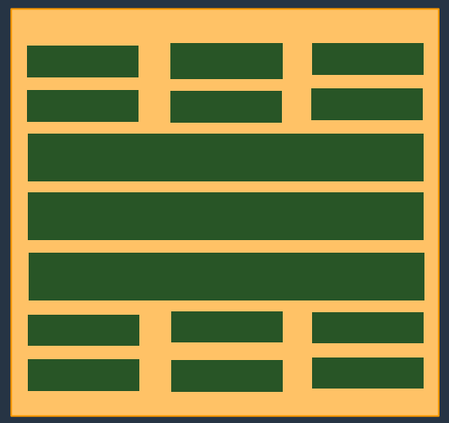
モノリスアーキテクチャ



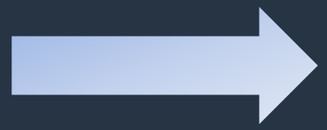
コードベースの
内部に複雑性が
隠蔽されていた

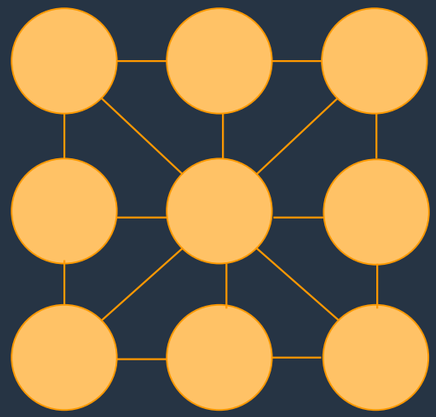
フライト、ホテル、旅行のデータ
がモノリスなデータベース
に格納される

複雑さが内から外へ移行している



コードベース
の複雑さ





コンポーネント間
の連携の複雑さ

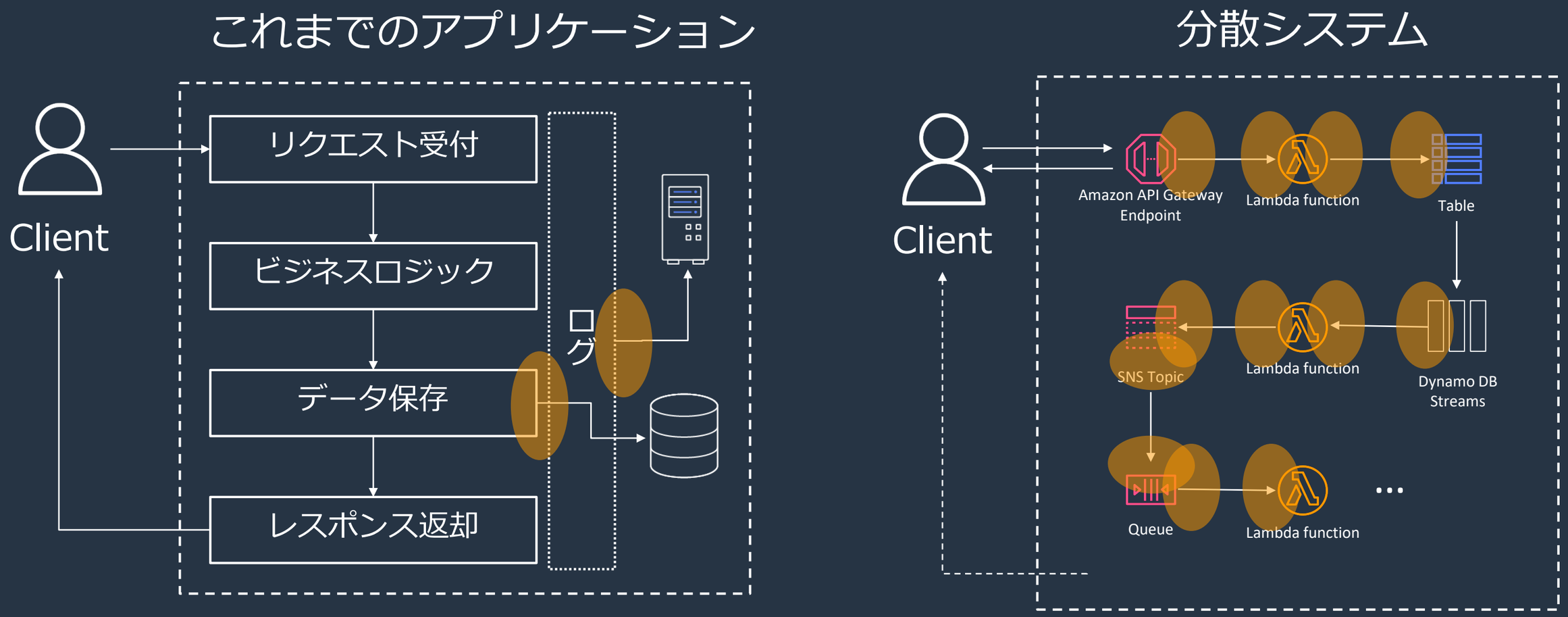
イベント駆動/分散システムのテストの課題

1. 外部の**依存関係が増える**
2. **非同期処理**の結果確認の手間
3. 早い変化の中で**整合性**を取らないといけない

イベント駆動/分散システムのテストの課題

1. 外部の**依存関係が増える**
2. 非同期処理の結果確認の手間
3. 早い変化の中で整合性を取らないといけない

コード上でも外部の依存関係が多くなる



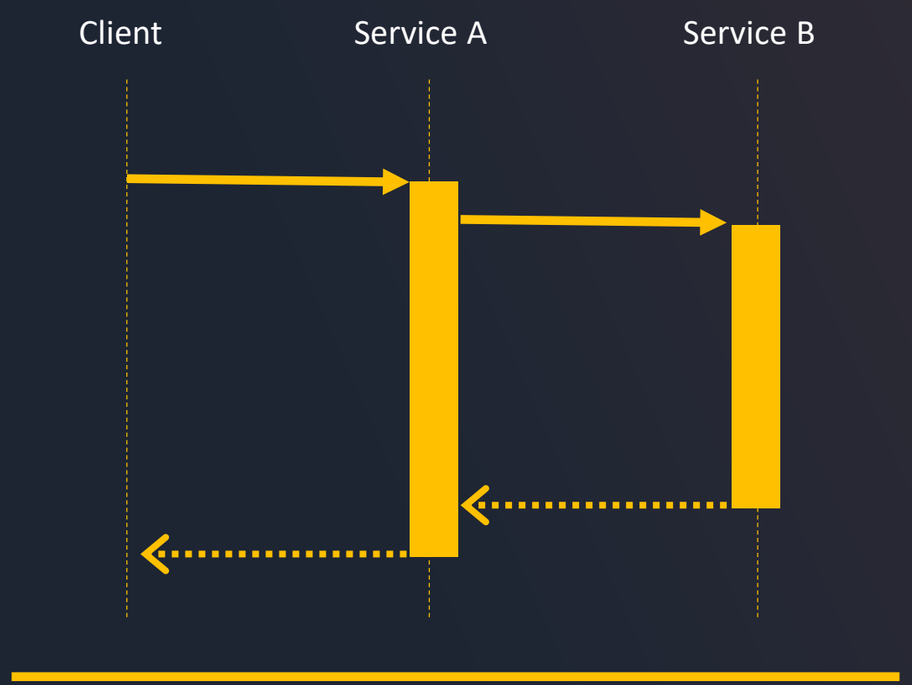
aws DevAx connect

イベント駆動/分散システムのテストの課題

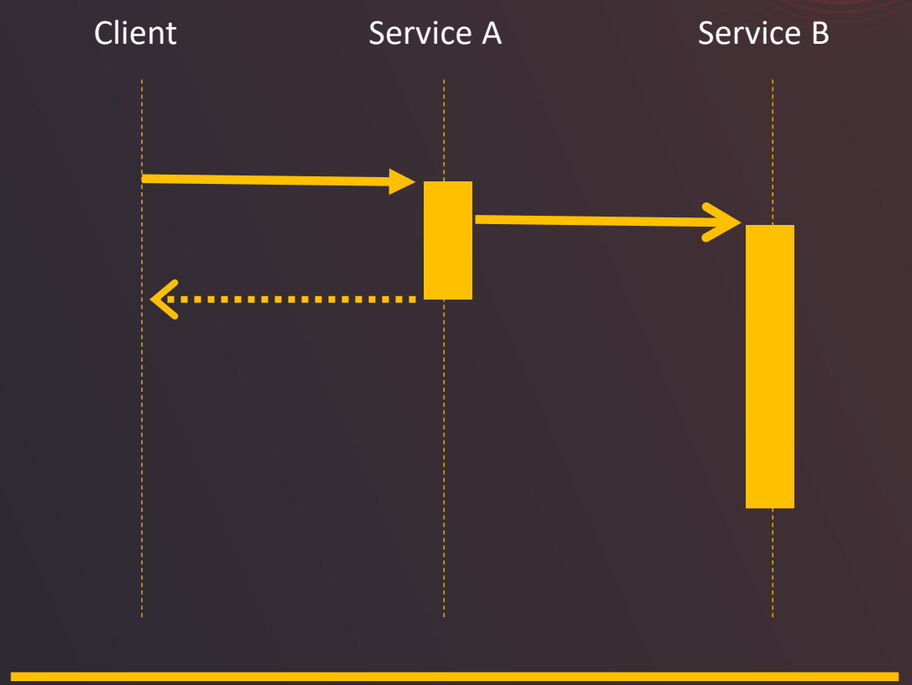
1. 外部の依存関係が増える
2. **非同期処理**の結果確認の手間
3. 早い変化の中で整合性を取らないといけない

非同期化による応答性の改善と依存性の削減

第一回資料より



同期コマンド



非同期イベント

aws DevAx

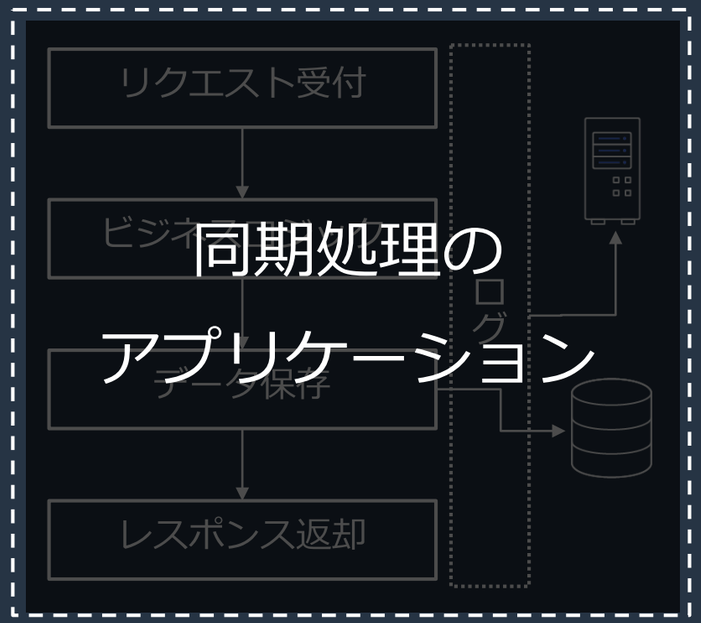
非同期処理のテスト

{x:1, y:2} を入力すると、{result: 3} が返る

{x:1, y:2}

Client

{result: 3}

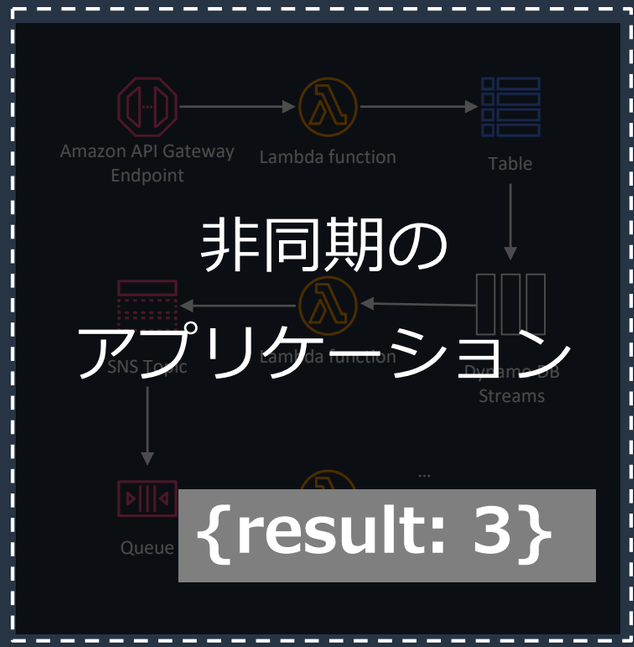


- 結果がすぐに確認できる
- どこで問題が起きたかもわかる

{x:1, y:2}

Client

202:Accepted



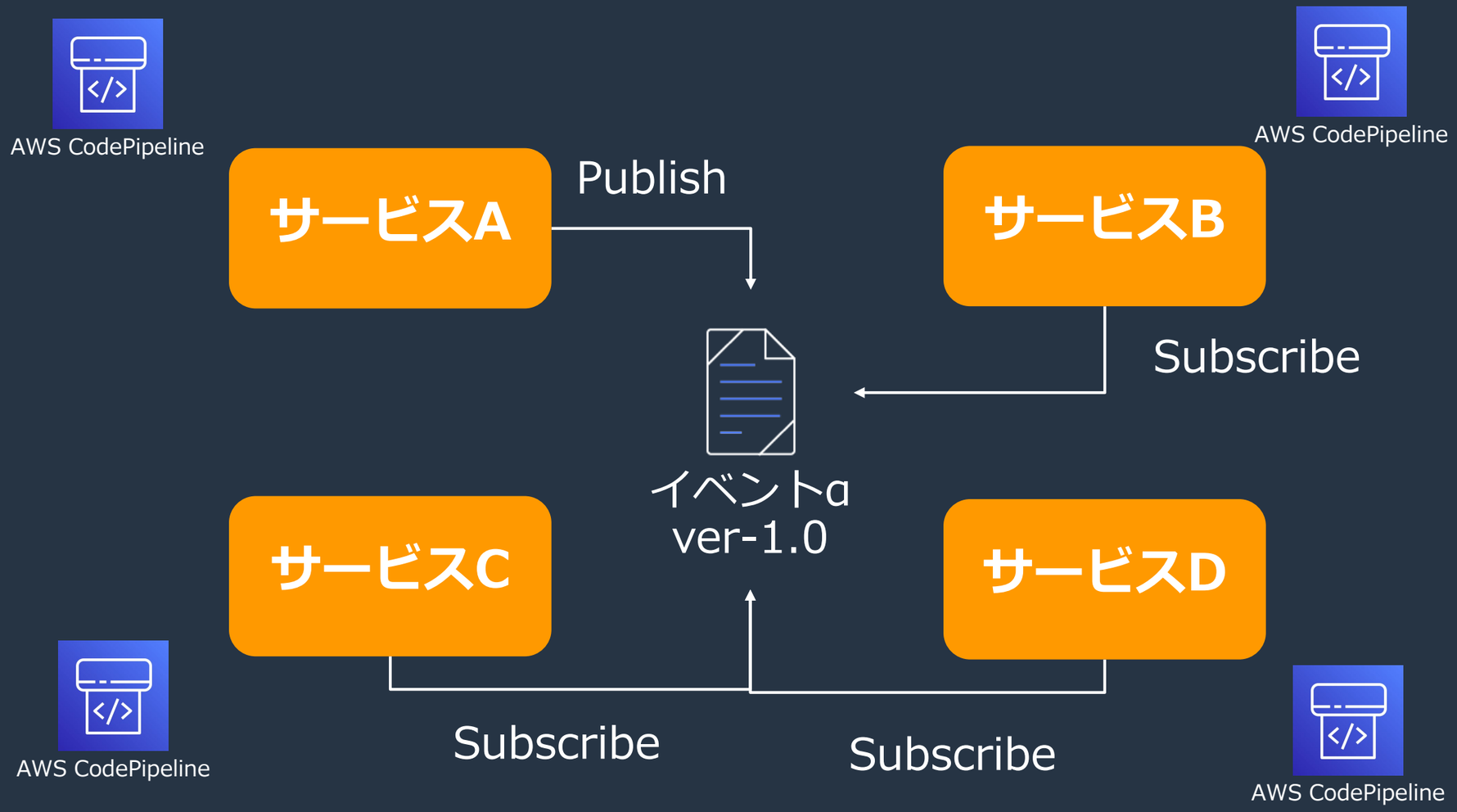
- 結果の確認が手間
- デバックが難しい

aws DevAx connect

イベント駆動/分散システムのテストの課題

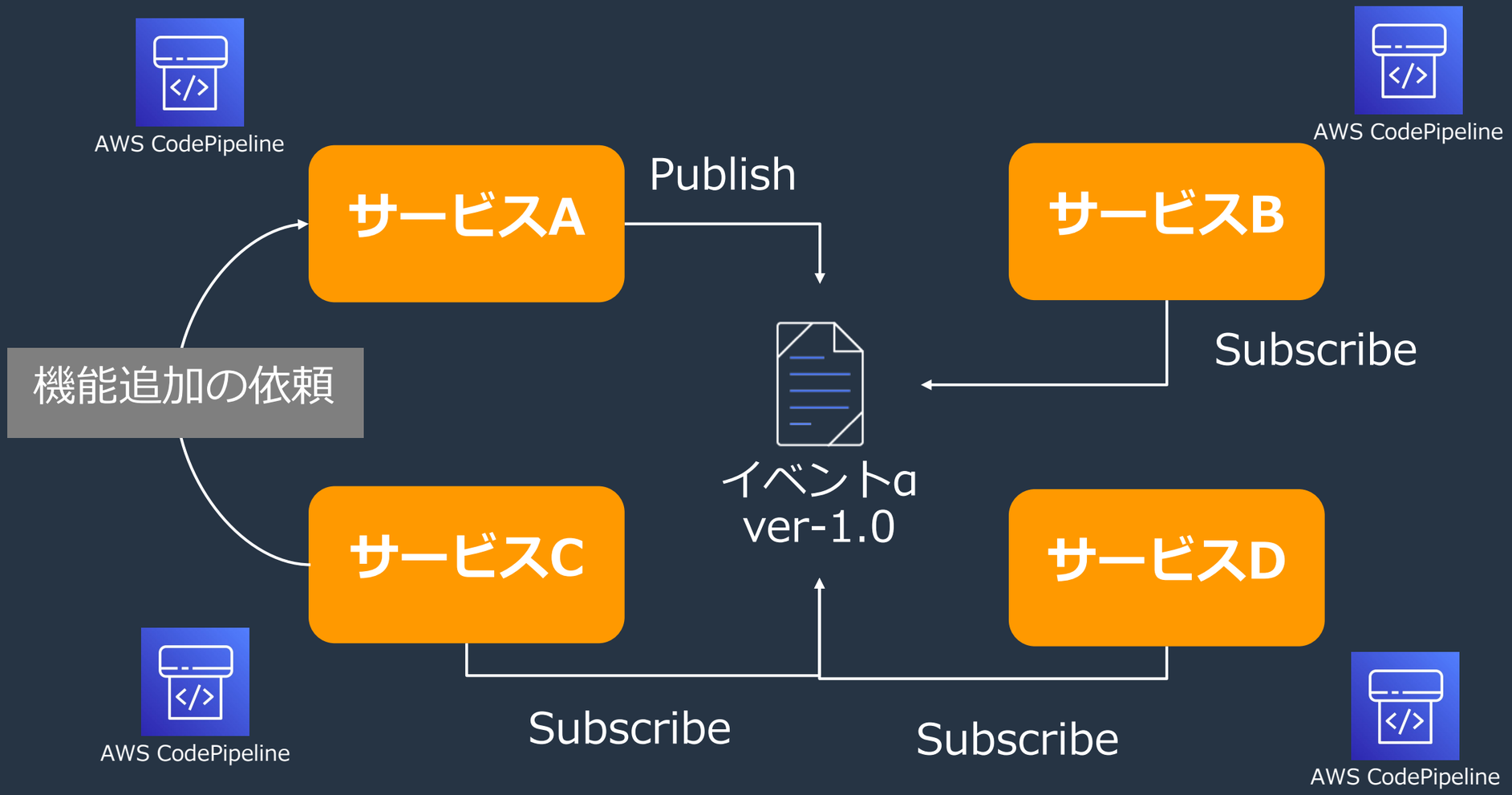
1. 外部の依存関係が増える
2. 非同期処理の結果確認の手間
3. 早い変化の中で**整合性**を取らないといけない

早い変化の中で協調していく必要がある



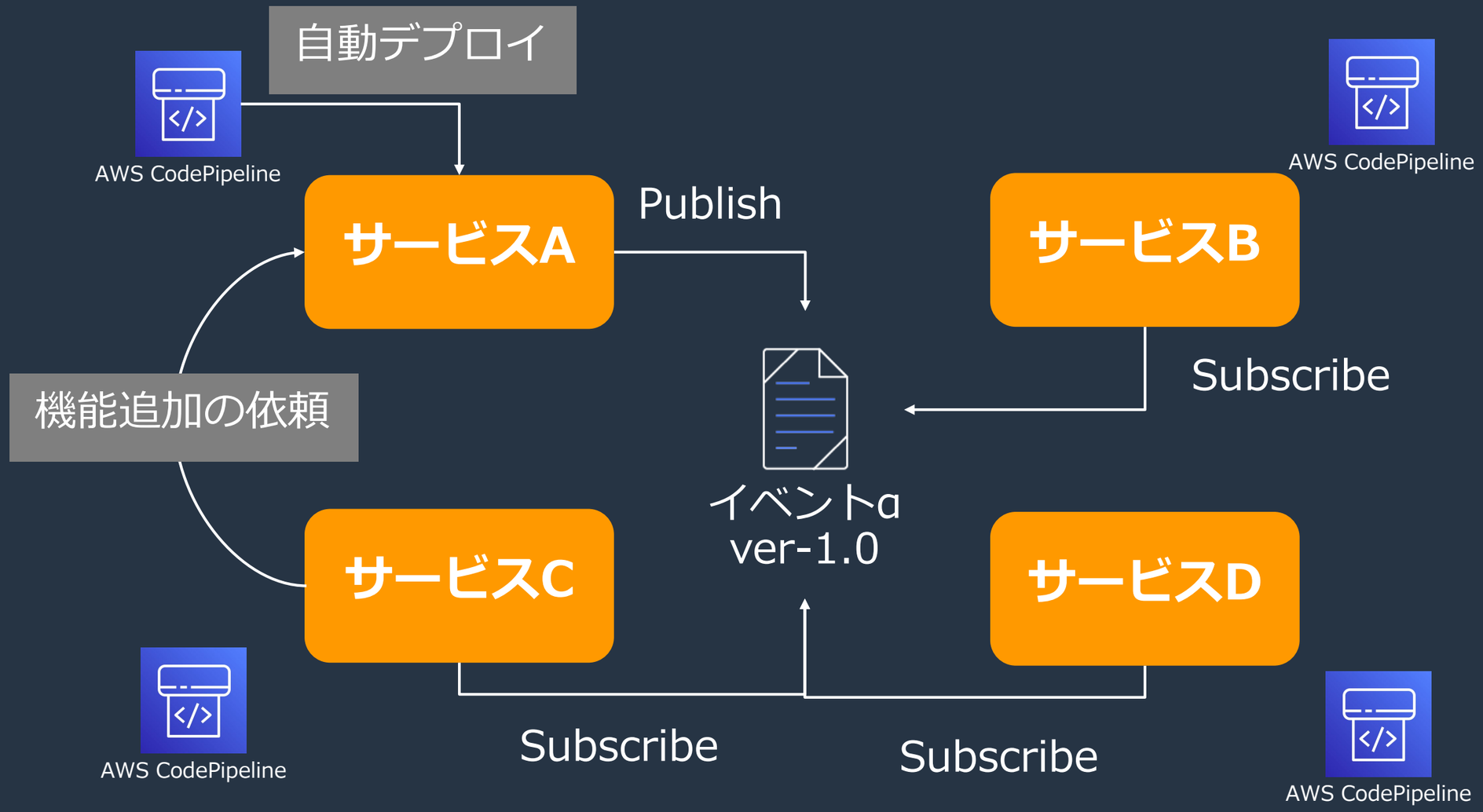
aws DevAx connect

早い変化の中で協調していく必要がある



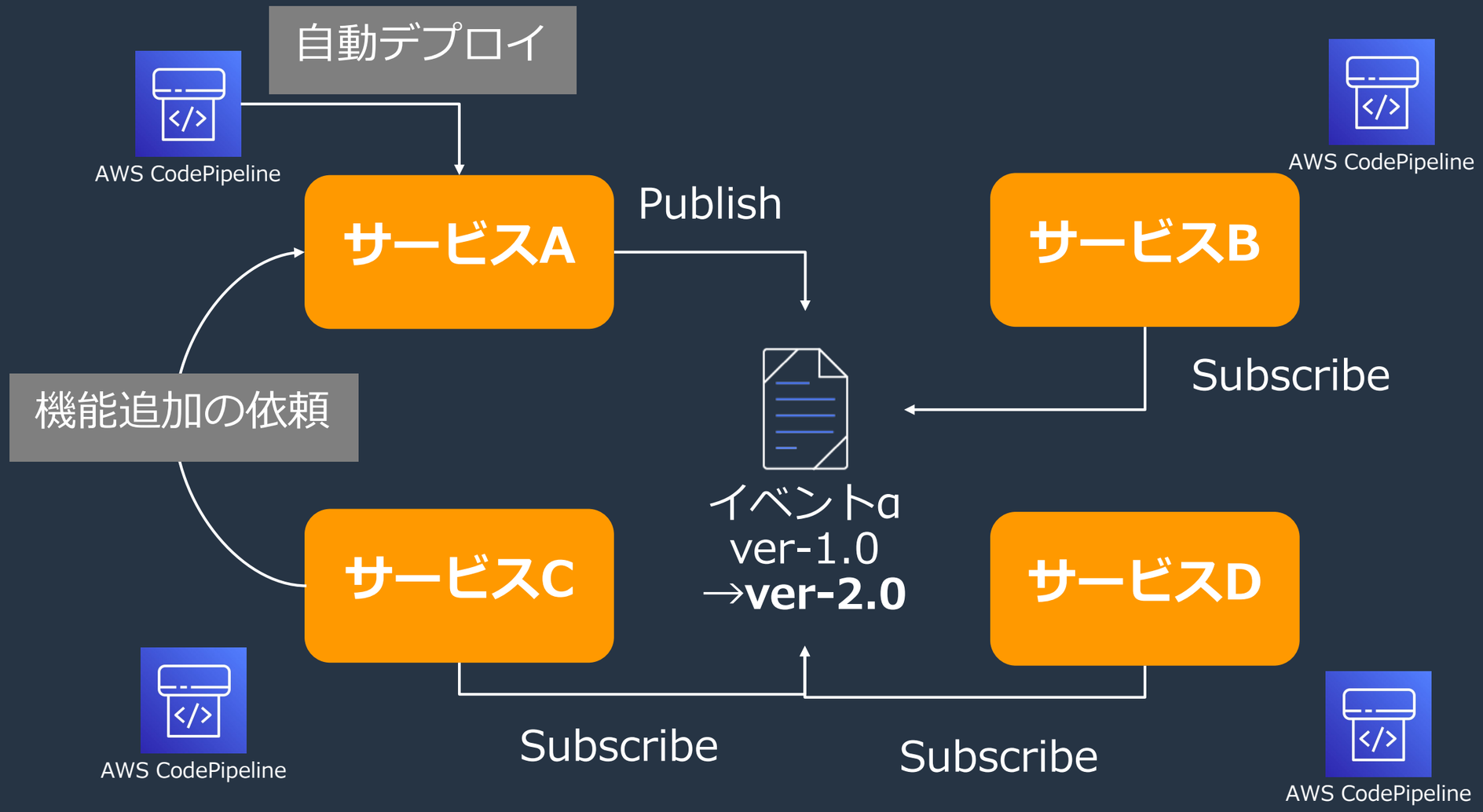
aws DevAx connect

早い変化の中で協調していく必要がある



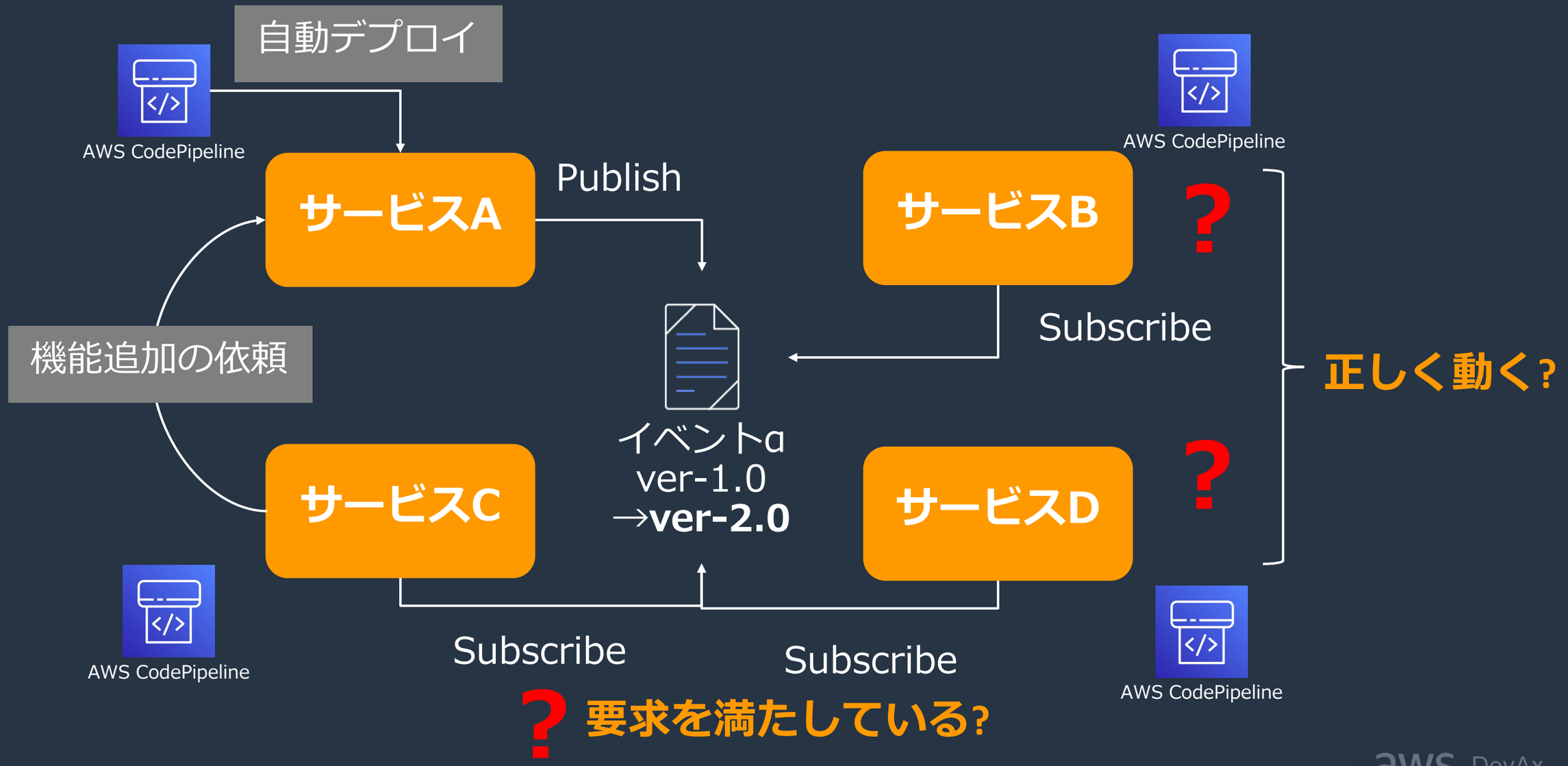
aws DevAx connect

早い変化の中で協調していく必要がある



aws DevAx connect

早い変化の中で協調していく必要がある



aws DevAx connect

ここまでのまとめ

分散システムのテストの難しさ

- 外部の**依存関係が増える**
- **非同期処理**の結果確認の手間
- 早い変化の中で**整合性**を
  取らないといけない

開発者には、、、

- はやく、
- 正確に、
- 簡単に動く

テストが必要

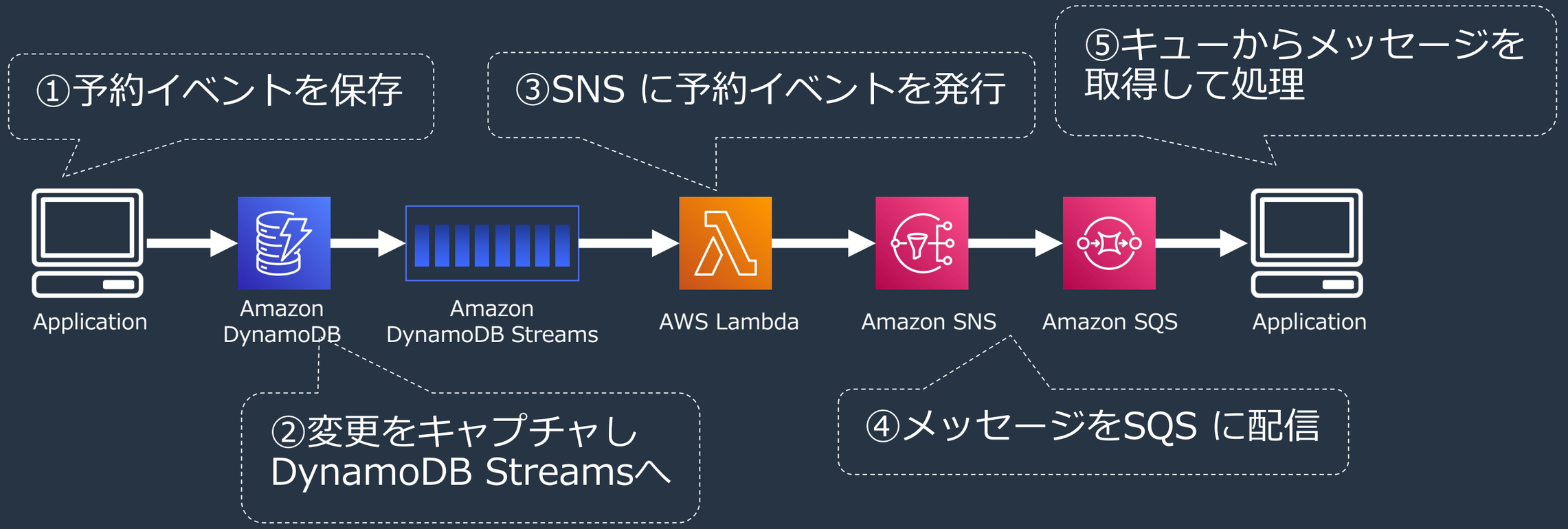
3.分散システムにおける テストの考え方とテクニック

aws DevAx connect

よくあるご質問

サーバーレスでシステムを構築しているんですが、
AWS のマネージドサービスの呼び出しを
どうやってテストすればよいでしょうか?

TravelBuddy を例に考えてみる



どこをテストしようとしている?

自分たちで実装する部分(コードの記述が必要)

AWS がサービスとして提供する機能による連携
(設定のみで実現可能)

AWSのサービスの機能自体

(図では色をつけていない部分)

aws DevAx connect

どこをテストしようとしている?

自分たちで実装する部分(コードの記述が必要)

AWS がサービスとして提供する機能による連携
(設定のみで実現可能)

AWSのサービスの機能自体 → **ここのテストは必要??**

(図では色をつけていない部分)

aws DevAx connect

AWS or 外部のサービスをテストする必要性

全てのメディアタイプのファイルをS3 にput できる

このテストの問題点

- かなりのコスト(工数/金額）が必要な上にビジネスに貢献しない
- 本来の仕様が自分たちではわからない
- バグが見つかっても自分たちでは改修できない

サービスを使う理由

- ビジネスに重要な機能に集中するため
- 「差別化できない重労働」をオフロードする

どちらにも貢献しない

開発者にとってのテスト

- 素早くフィードバックを得て開発を加速する
- 自信を持って変更できるようにする

※動作確認やPoC は必要

❌「AWS や3rd Party の サービスの動作確認をしない」

- 選定時のPoC や動作確認は必須

- 設定が期待する性能 や動作を提供することの確認も必要
  例: FIFO の順序制御、DynamoDB のR/WCU の設定値 etc…

- E2E テストおいて、期待した結果になっていることの確認
  (ただし、数を増やしすぎない。後述)

- 実運用ではリアルユーザーモニタリングと合成モニタリングなどで、
  全体として期待した動作を提供できているかを確認し続ける

どこをテストしようとしている?

自分たちで実装する部分(コードの記述が必要)

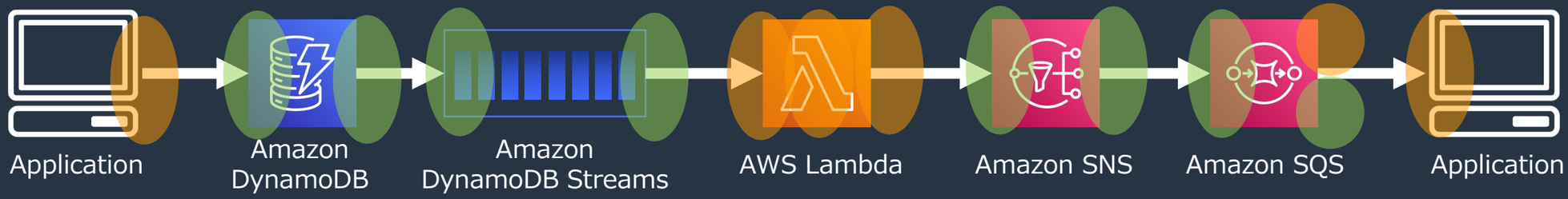
AWS がサービスとして提供する機能による連携
(設定のみで実現可能)

AWSのサービスの機能自体

(図では色をつけていない部分)

aws DevAx connect

どこをテストしようとしている?



自分たちで実装する部分(コードの記述が必要)

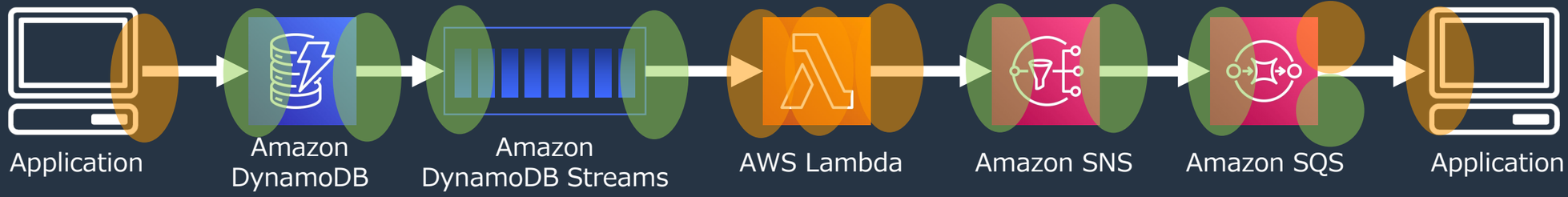
AWS がサービスとして提供する機能による連携 (設定のみで実現可能)

- 連携設定の不備

- IAM の権限設定不備

etc

どこをテストしようとしている?



自分たちで実装する部分(コードの記述が必要)

AWS がサービスとして提供する機能による連携 (設定のみで実現可能)

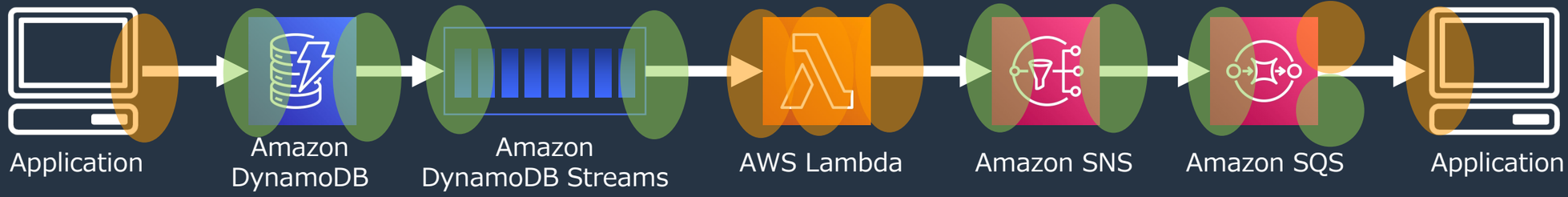
- 連携設定の不備

- IAM の権限設定不備

etc

- 解決策:Policy Validation の自動化や、AWS Config, CDK のテスト など
- 今回の話とはズレてくるのでスキップ

どこをテストしようとしている?



自分たちで実装する部分(コードの記述が必要)

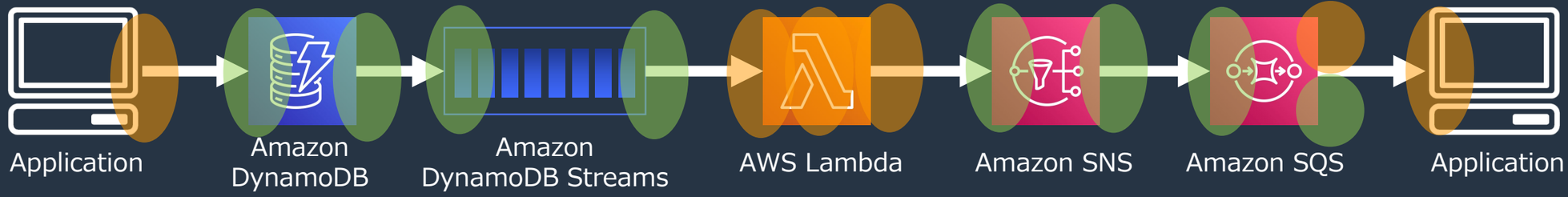
AWS がサービスとして提供する機能による連携 (設定のみで実現可能)

- 連携設定の不備

- IAM の権限設定不備

etc

どこをテストしようとしている?



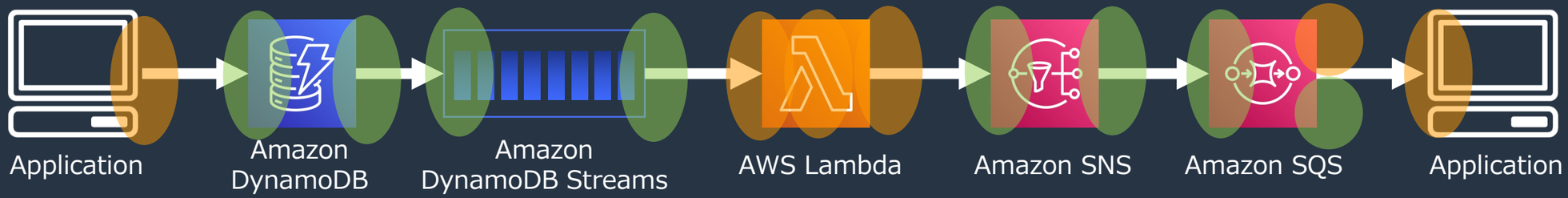
自分たちで実装する部分(コードの記述が必要)

- イベントの受け取り方
- AWS API の呼び出し方

AWS がサービスとして提供する機能による連携
(設定のみで実現可能)

- 連携設定の不備

どうやってテストする?



- イベントの受け取り方
- AWS API の呼び出し方
- 連携設定の不備

ローカルでの実行

クラウド上で実施

どうやってテストする?

- イベントの受け取り方
- AWS API の呼び出し方
- 連携設定の不備

ローカルでの実行

クラウド上で実施

aws DevAx connect

ローカルでの実行

例: SAM CLI を使ったローカルでのサーバーレスアプリケーション実行

ローカル起動用の Dummy Event データの生成

```
$  sam local generate-event ¥
       dynamodb update ¥
       --table reservation > event.json
```

生成したEvent データを使ったLambda の実行

```
$ sam local invoke -e
event.json
```

API Gatewayから、Lambdaを呼び出し実行

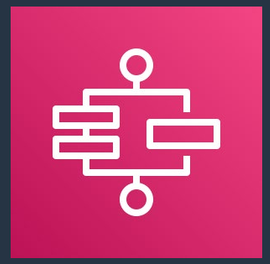
```
$ sam local start-api
```

→ http://127.0.0.1:3000 で起動

SAM CLI

ローカルでの実行

SAM CLI 以外のローカルでのAWS サービスの実行



```
$ docker run -p 8083:8083 ¥
  amazon/aws-stepfunctions-local
```

Step Functions Local

```
$ docker run -d -p 8000:8000 ¥
  amazon/dynamodb-local ¥
  -jar DynamoDBLocal.jar  ¥
  -sharedDb
```

DynamoDB Local

```
$ amplify mock
```

AWS Amplify

```
$ ./codebuild_build.sh
  -i aws/codebuild/standard:5.0 ¥
  -a output
```

AWS CodeBuild agent

ローカルでの実行

今回のサンプルは全てローカルで実行することも可能

ツールなどのインストール
設定/コードの書換

クラウド上での動作を
全て再現はできない

連携設定の確認にはならない

aws DevAx connect

ローカルでの実行
メリット/デメリット

※ここではクラウドの動作と一致すること

○ はやく　△ 正確に※　✕ 簡単に動く

メリット

- 高速に動作確認が可能
- デプロイ不要、AWS利用料なし
- ブレークポイントのサポート

デメリット

- 完全に再現できるわけではない
- IAM など検証できない部分も
- 環境構築、維持にコストがかかる

ブレークポイントを使いたい場合など
コーディング中の動作確認、デバッグに便利

どうやってテストする?

- イベントの受け取り方
- AWS API の呼び出し方
- 連携設定の不備

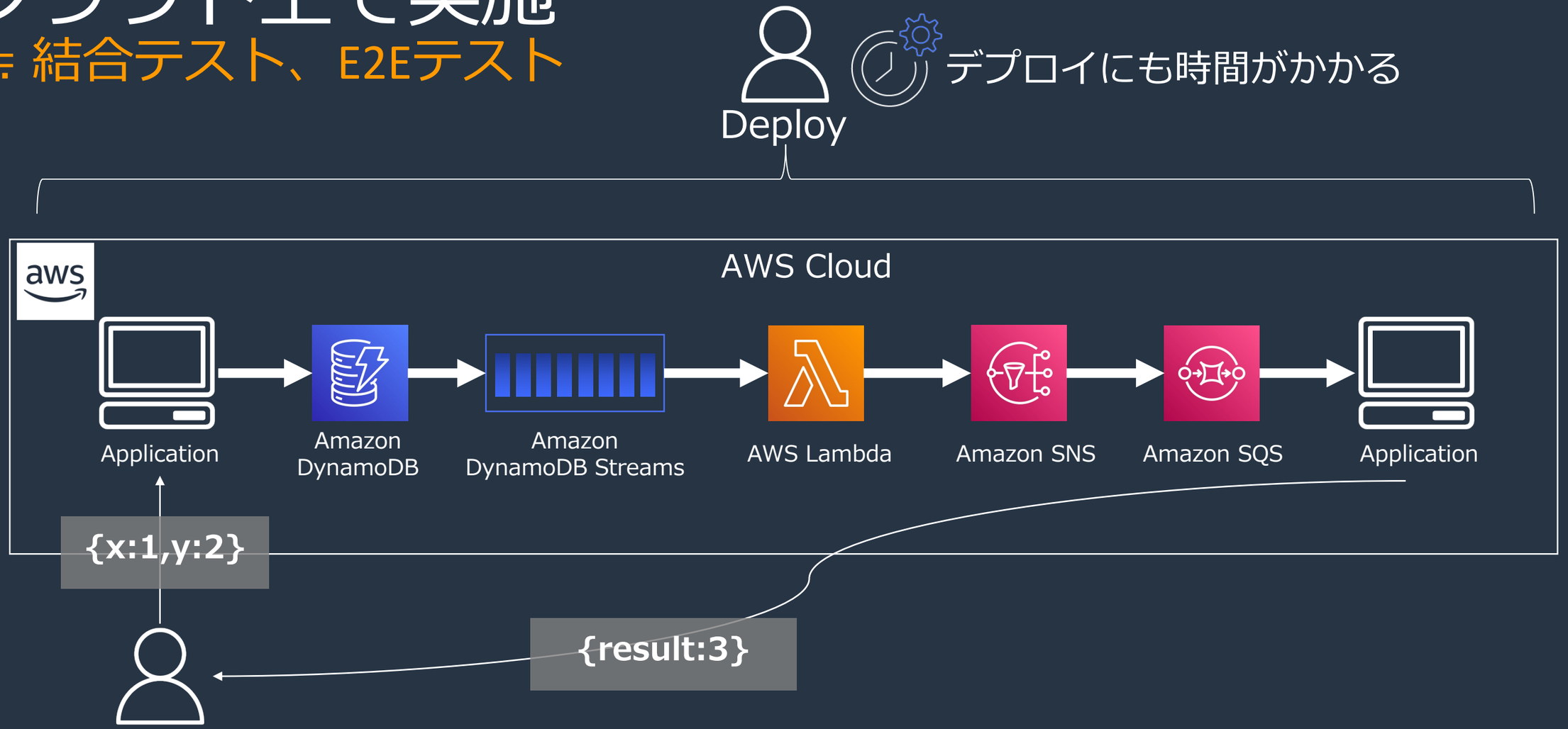
ローカルでの実行

クラウド上で実施

aws DevAx connect

クラウド上で実施

≒ 結合テスト、E2Eテスト



クラウド上で実施
メリット/デメリット

※ここではクラウドの動作と一致すること

✖はやく　　○正確に※　　△簡単に動く

メリット

- 正確なサービスの動作を確認
- 権限設定の確認も可能

デメリット

- 時間がかかる
- デプロイなど毎回準備が必要
- AWS利用料もかかる

結合テスト~E2E テストなど
実施回数を最小化する

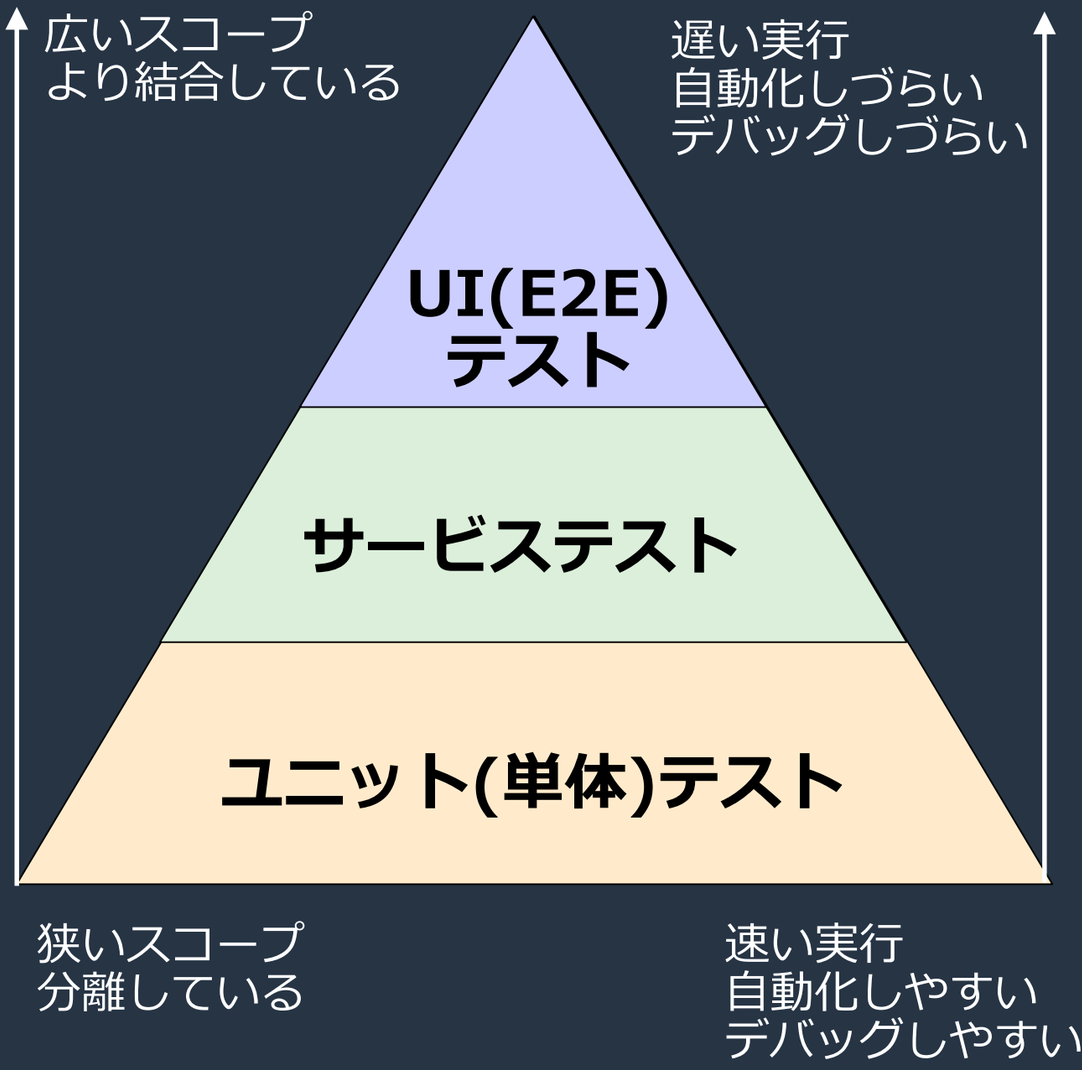
テストフェーズについて

テストピラミッド

- Mike Cohn が
  『Succeeding with Agile』で説明

- 上に行くほどスコープと実行時間↑
  逆に下に行くとスコープ/実行時間↓

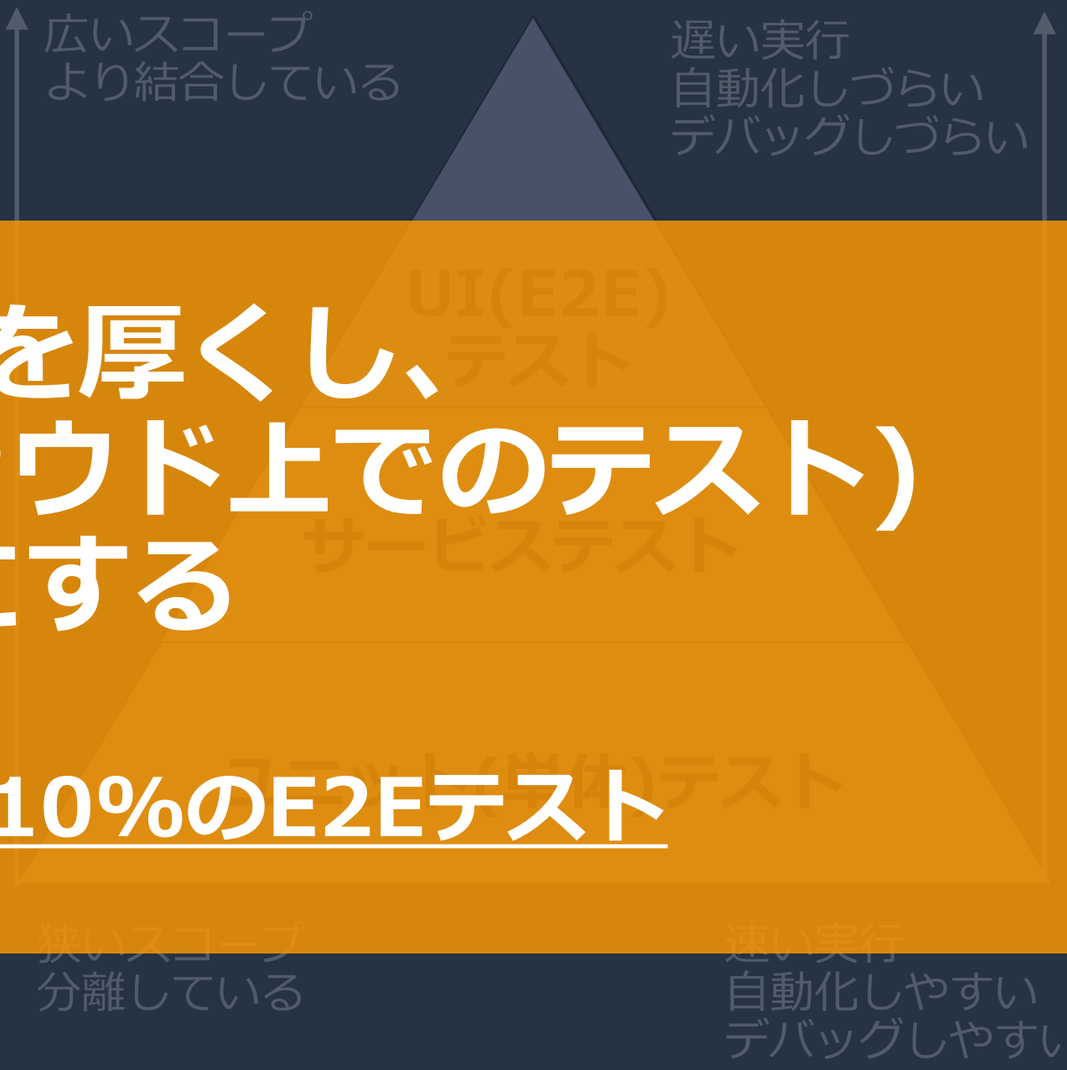
- ユニットテストを多くし、
  E2Eテストは重要なジャーニーに絞る
  (逆ピラミッドにならないようにする)



テストフェーズについて

テストピラミッド

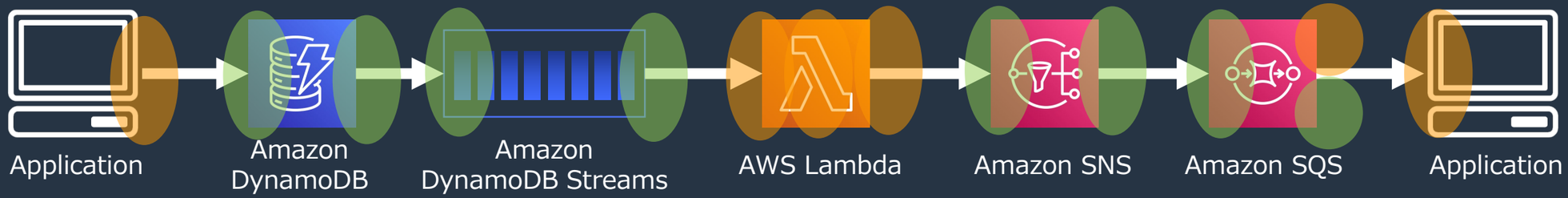
- Mike Cohn が『Succeeding with Agile』で説明
- 上に行くほどスコープと実行時間が増加、逆に下に行くとスコープが減少傾向
- ユニットテストは多く、UIテストやE2Eテストは重要なジャーニーに絞る(逆ピラミッドにならないようにする)

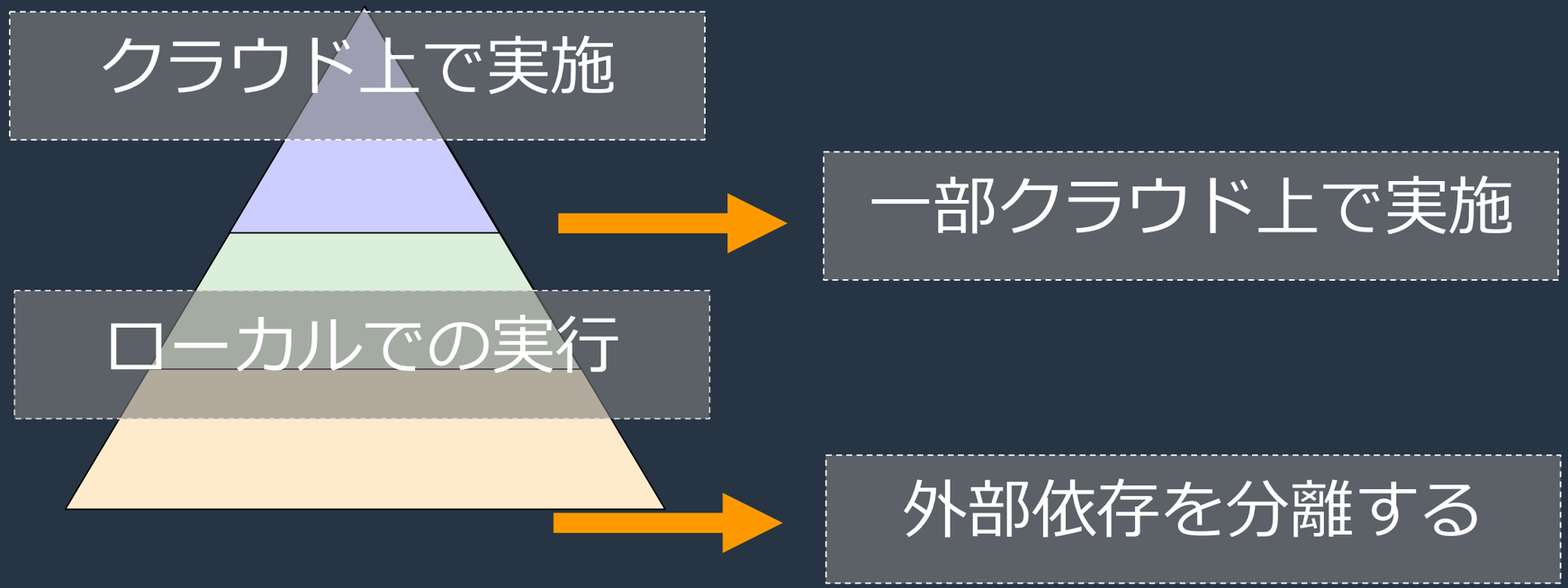


ユニットテストを厚くし、
UI(E2E)テスト(≒ クラウド上でのテスト)
は最小限にする

70%のユニットテスト、10%のE2Eテスト

この二択しかないのか

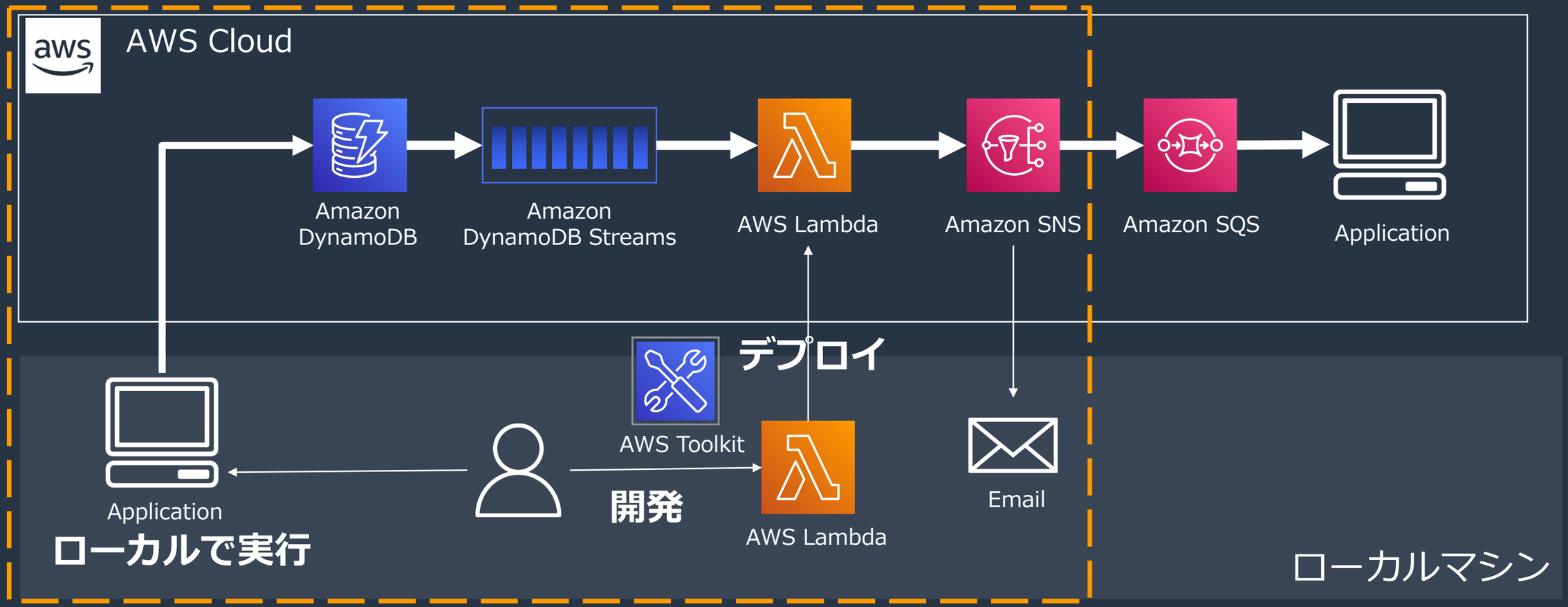




aws DevAx connect

一部クラウド上で実施

スコープを限定して実施



aws DevAx connect

一部クラウド上で実施
メリット/デメリット

△はやく　　○正確に　　△簡単に動く

メリット

- 正確なサービスの動作を確認
- デプロイも全体を実施するより高速

etc.

デメリット

- 自動化には適さない
- ローカル実施よりは遅い
  (回数が限定されるなら許容範囲)

etc.

サービス間の連携の動作確認や結合テストなど
頻度の少ないテスト,動作確認に

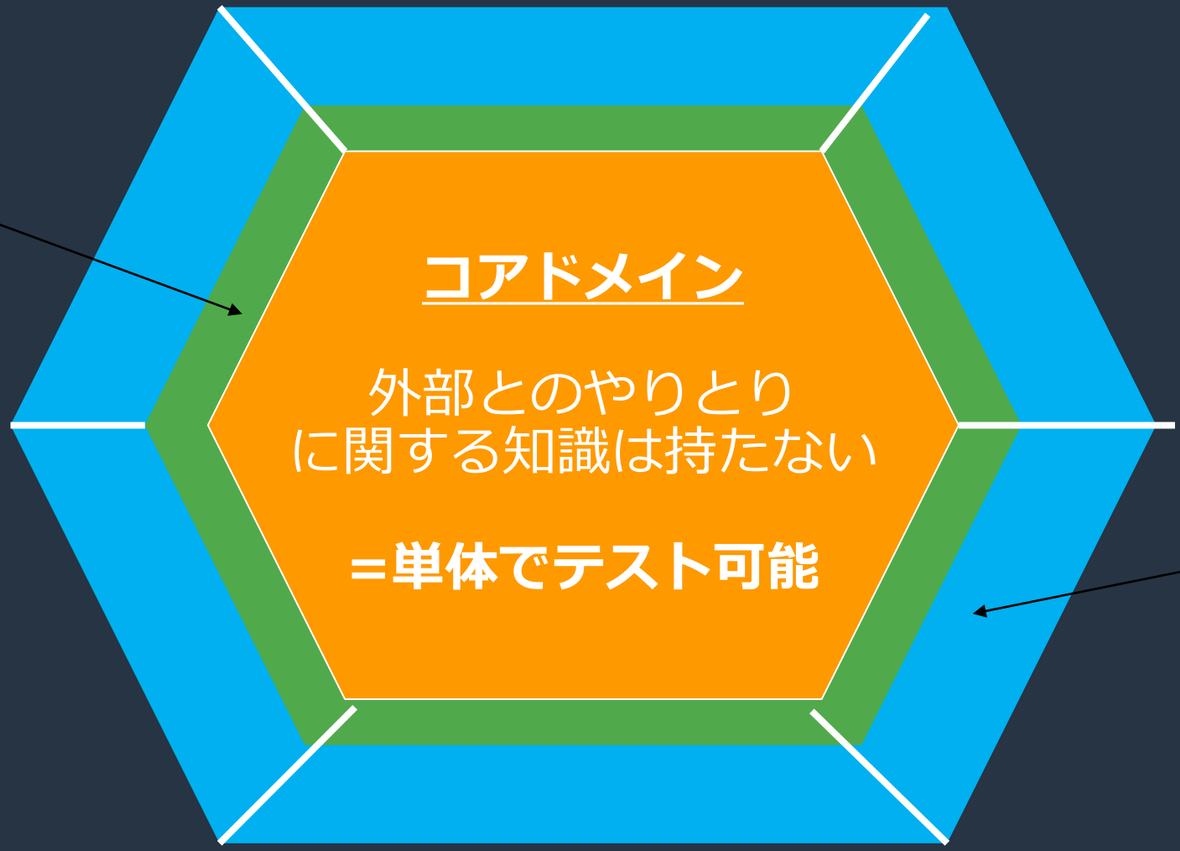
外部依存を分離する
ポート&アダプター (ヘキサゴナルアーキテクチャ)

ポート

ドメインが外部と
やりとりするIFを提供

外部やモック呼び出し
のエントリーポイント

ドメインと外部の境界



アダプタ

ポートのIFに応じて
特定の外部システム
とのやりとりを変換

例: SQL アダプタ

モックアダプタ

aws DevAx connect

Lambda関数の実装例

```
def lambda_handler(event, context):
    """ Unit TestしやすいLambda関数のサンプル
    """
    id = get_parameter(event, "id")
    is_takeout = get_parameter(event, "is_takeout")

    item = get_item(id)

    tax = calc_tax(item, is_takeout)

    return {
        "statusCode": 200,
        "body": json.dumps({
            'id': item['id'],
            'item_name': item['item_name'],
            'price': item['price'],
            'tax': tax
        }),
    }
```

Lambda関数の実装例

```
def lambda_handler(event, context):
    """ Unit Testしやすい
    """
    id = get_parameter
    is_takeout = get_p

    item = get_item(id)

    tax = calc_tax(item, is_takeout)

    return {
        "statusCode": 200,
        "body": json.dumps({
            'id': item['id'],
            'item_name': item['item_name'],
            'price': item['price'],
            'tax': tax
        }),
    }
```

```
def calc_tax(item, is_takeout):
    price = Decimal(item['price'])
    tax_rate = Decimal(0.1)
    if is_takeout is not None and is_takeout.lower() == "true":
        tax_rate = Decimal(0.08)
    return int(math.floor(price * tax_rate))
```

コアドメイン

- PureなロジックとしてUnit Testを実施可能
- AWS など外部の知識は不要

Lambda関数の実装例

```python
def lambda_handler(event, context):
    """ Unit TestしやすいLambda関数のサンプル
    """
    id = get_parameter(event, "id")
    is_takeout = get_parameter(event, "is_takeout")

    item = get_item(id)

    tax = calc_tax(item, is_takeout)

    return {
        "statusCode": 200,
        "body": json.dumps({
            'id': item['id'],
            'item_name': item['item_name'],
            'price': item['price'],
            'tax': tax
        }),
    }
```

ポート
- ドメインとアダプタのIF
- get_item() の先にはDynamoDB とのアダプタ

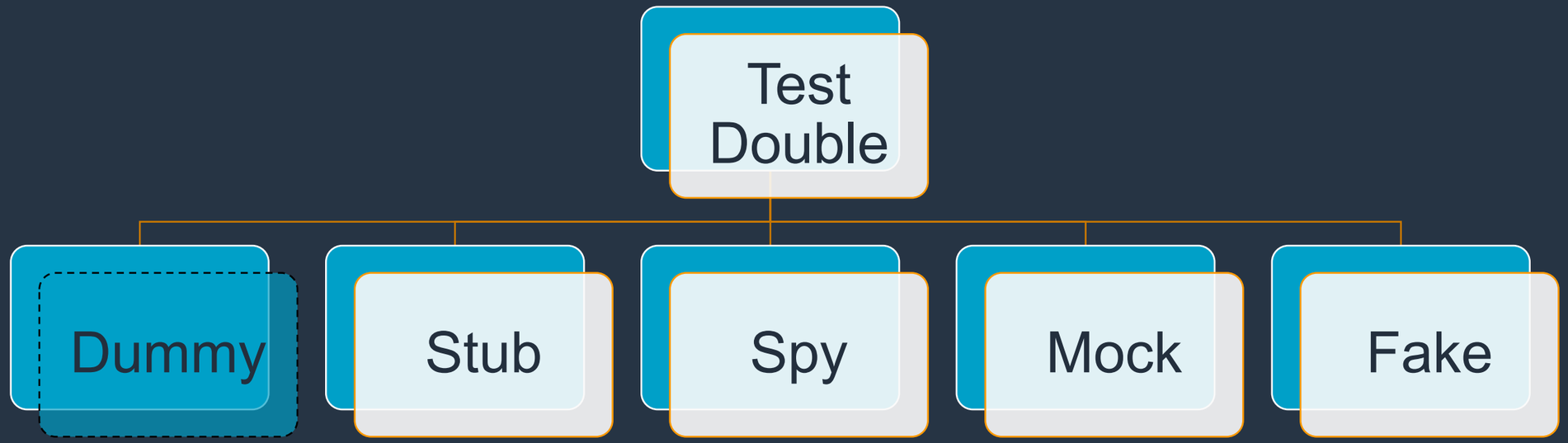
アダプタ
- この場合handler()がアダプタ
- Lambda サービスとの変換を担当

コアドメインに対する単体テスト

- 純粋な要件に対してテストケースを書く
  - 他のAWSサービスとの結合方法に対する知識を含まない
- テストが軽くなるため、開発やデリバリーを高速化できる
- 外部のアーキテクチャが変更された時にも、テストケースが影響を受けにくい
  - テストの陳腐化が防げる
  - テストのメンテナンスコストの軽減
- Pure Logicなので意図が伝わりやすい、理解しやすい
  - テストケースから仕様を理解出来る

テストダブル(Double＝代役)

https://martinfowler.com/bliki/TestDouble.html



テストダブルは、テスト目的で本番オブジェクトを
置き換える場合の置き換えデザインパターンを表す用語。

さまざまな種類のdoubleがある

システムの一部を差し替えてテストする手法

ダミーオブジェクト	パラメータとして受け渡しするが利用されることのないオブジェクト
スタブ	テスト時に呼び出され、あらかじめ容易した結果を返すオブジェクト
スパイ	テスト対象からの出力を記録しておくことでテストコード実行後に値を取り出して検証する
モック	期待した通りに呼び出されることを確認するためのオブジェクト。期待されない呼び出しが行われた場合は例外を返す。
フェイク	動作するように実装されているがエラー処理などを手抜きしたテスト用オブジェクト

外部依存を分離する
メリット/デメリット

○ はやく　　ー 正確に　　○ 簡単に動く

メリット

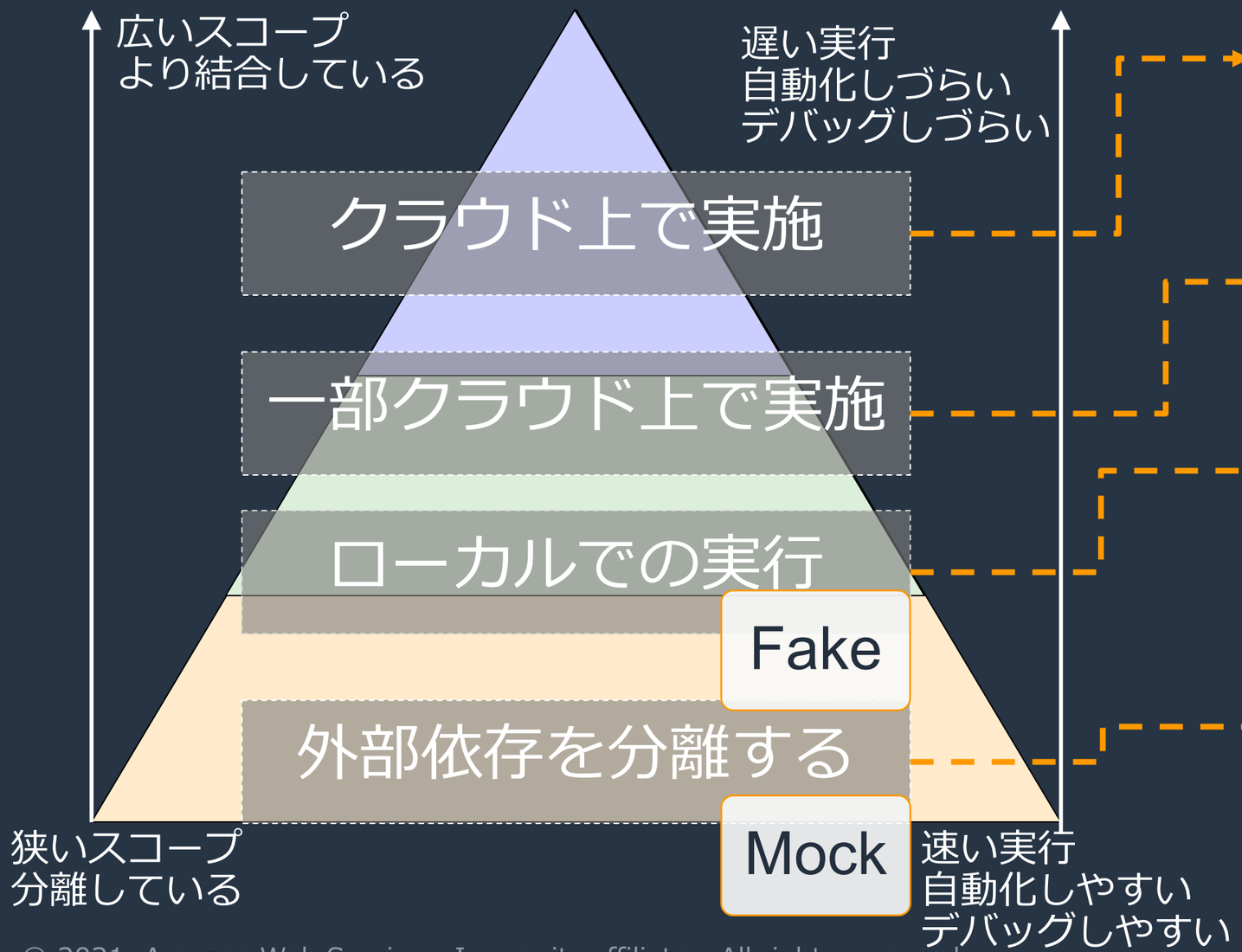
- 高速に開発内容へのフィードバックが得られる
- 外部との依存関係がないので、影響を受けづらく維持しやすい
- Mock などの活用で連携部分のロジックのテストも

デメリット

- 連携部分は実際の接続ではないので、結合テストなどで補完が必要

単体テストを充実させる

ここまでの振り返り



E2Eテストは最小限にし、
重要なストーリーのみにする

接続部分の開発や動作確認
で利用

ブレークポイントを利用した
デバッグなど

テストしやすい設計、
テストダブルの活用などで
充実させていく

aws DevAx connect

イベント駆動/分散システムのテストの課題

1. 外部の**依存関係が増える** ✓
2. **非同期処理**の結果確認の手間
3. 早い変化の中で**整合性**を取らないといけない

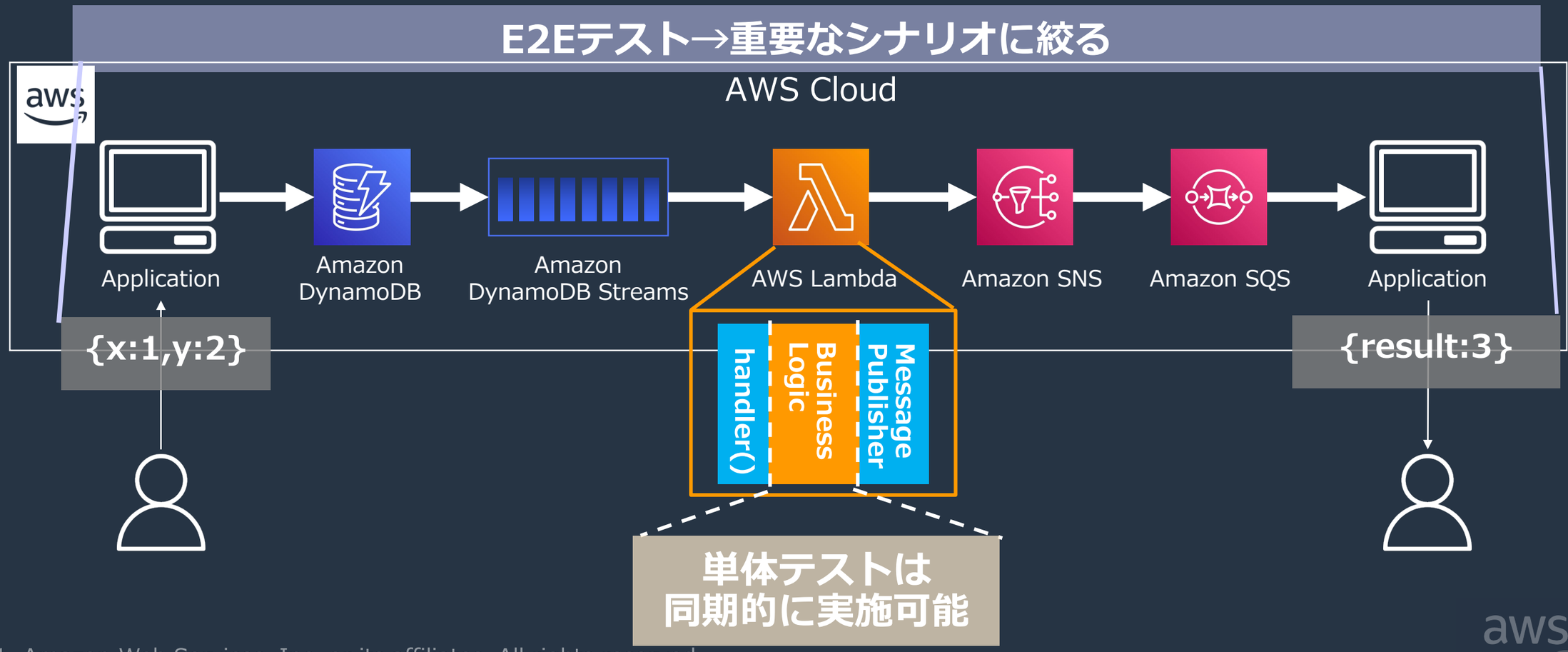
イベント駆動/分散システムのテストの課題

1. 外部の**依存関係が増える** ✓
2. **非同期処理**の結果確認の手間
3. 早い変化の中で**整合性**を取らないといけない

非同期テストの問題を見直す

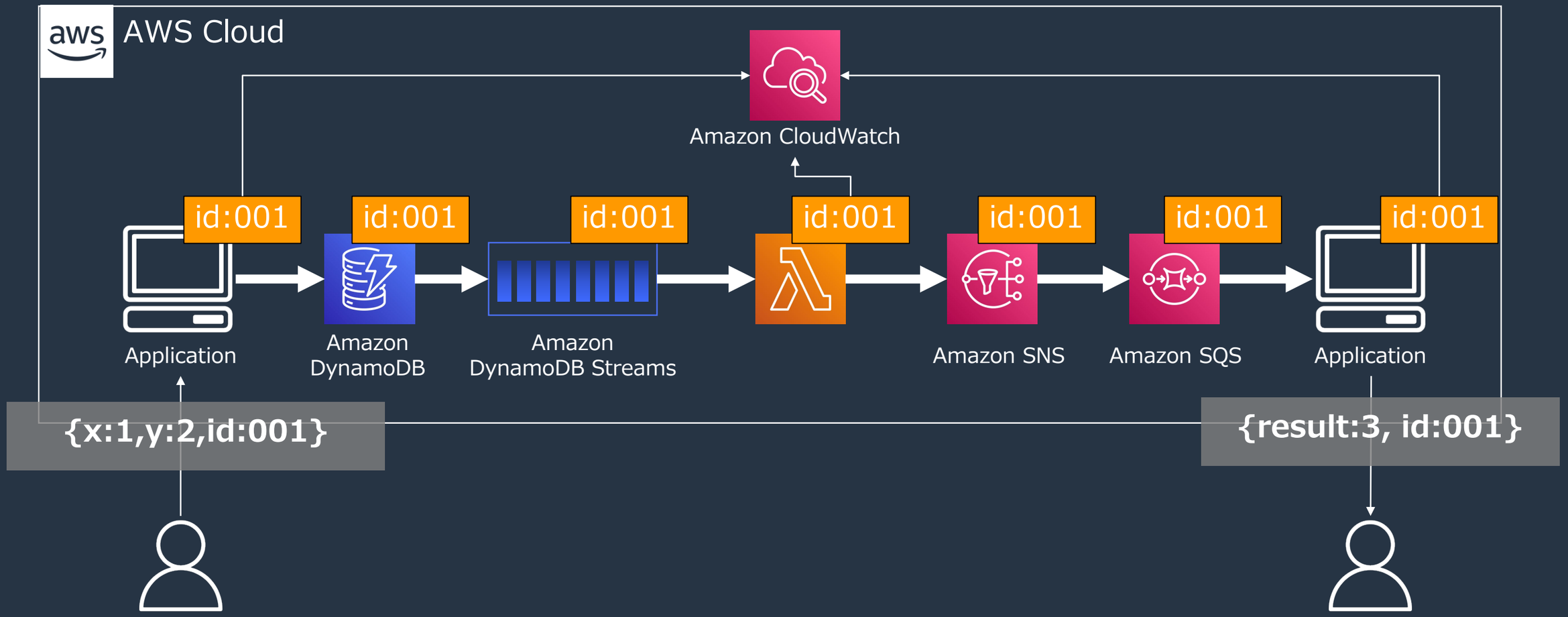
最小化したE2E テストと、ユニットテストでカバー

{x:1, y:2} を入力すると、{result: 3} が返る



設計段階からテスト/調査のしやすさを組み込む

相関ID を利用してイベントを識別できるように



イベント駆動/分散システムのテストの課題

1. 外部の**依存関係が増える** ✓
2. **非同期処理**の結果確認の手間 ✓
3. 早い変化の中で**整合性**を取らないといけない

イベント駆動/分散システムのテストの課題

1. 外部の**依存関係が増える** ✓
2. **非同期処理**の結果確認の手間 ✓
3. 早い変化の中で**整合性**を取らないといけない

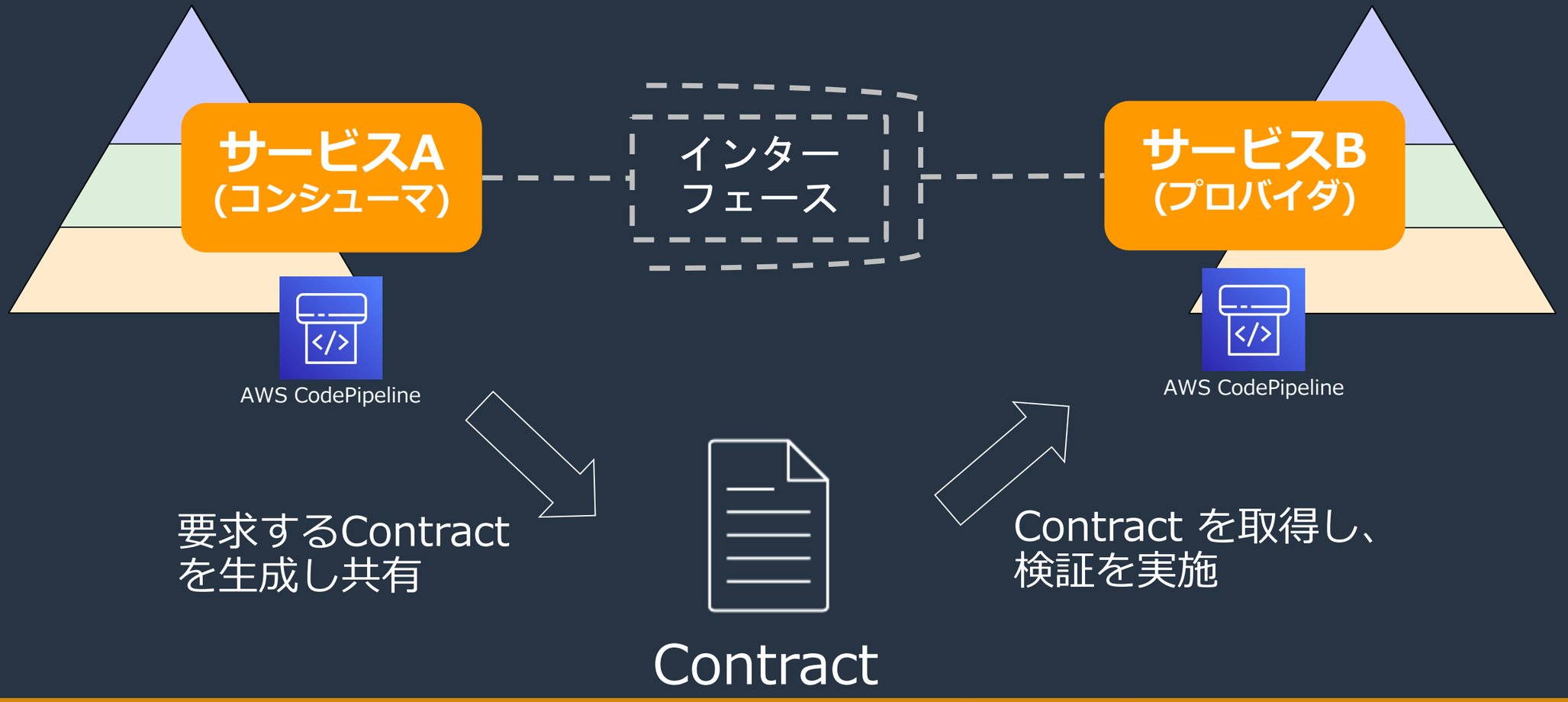
改めて課題の整理

- マイクロサービスにおいて、サービス間のインターフェースは「契約」
- サービス間のテストは「結合テスト」以降になる
- それぞれのサービスが独自に変更、デプロイしていきたい

契約の整合性を簡単にテストしたい

コンシューマー駆動契約

Consumer-Driven Contracts (CDC)



インターフェースの整合性をそれぞれで自動テストできる

PACT 

- コンシューマ駆動契約テストのためのツール
- PACT の基本的な動作
  1. コンシューマのテスト(CI)の中で”pact file” (JSON) を生成
  2. プロバイダは”pact file” を利用して、契約を壊していないかテストする
  3. それぞれのテストでMock の機能も提供
- ”pact file”を中継するブローカーも提供
- 対応言語:Ruby, Javascript,JVM,.NET(C#),Go,Python,PHP など主要な言語に対応

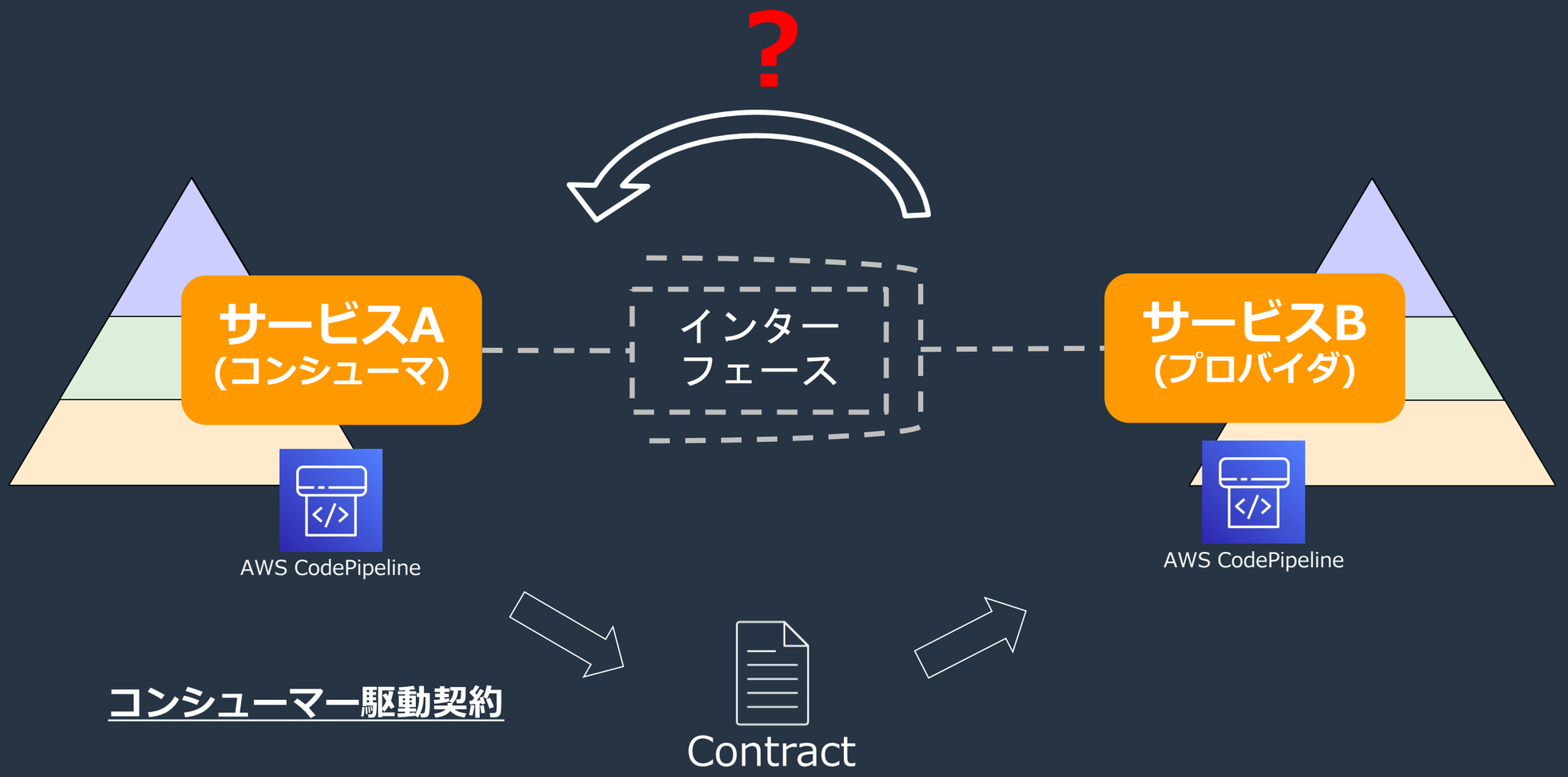
Message Pact Specification

- Pact Specification 3.0 にて導入
  https://github.com/pact-foundation/pact-specification/tree/version-3
- メッセージングサービスに関係なくメッセージの仕様をチェック可能

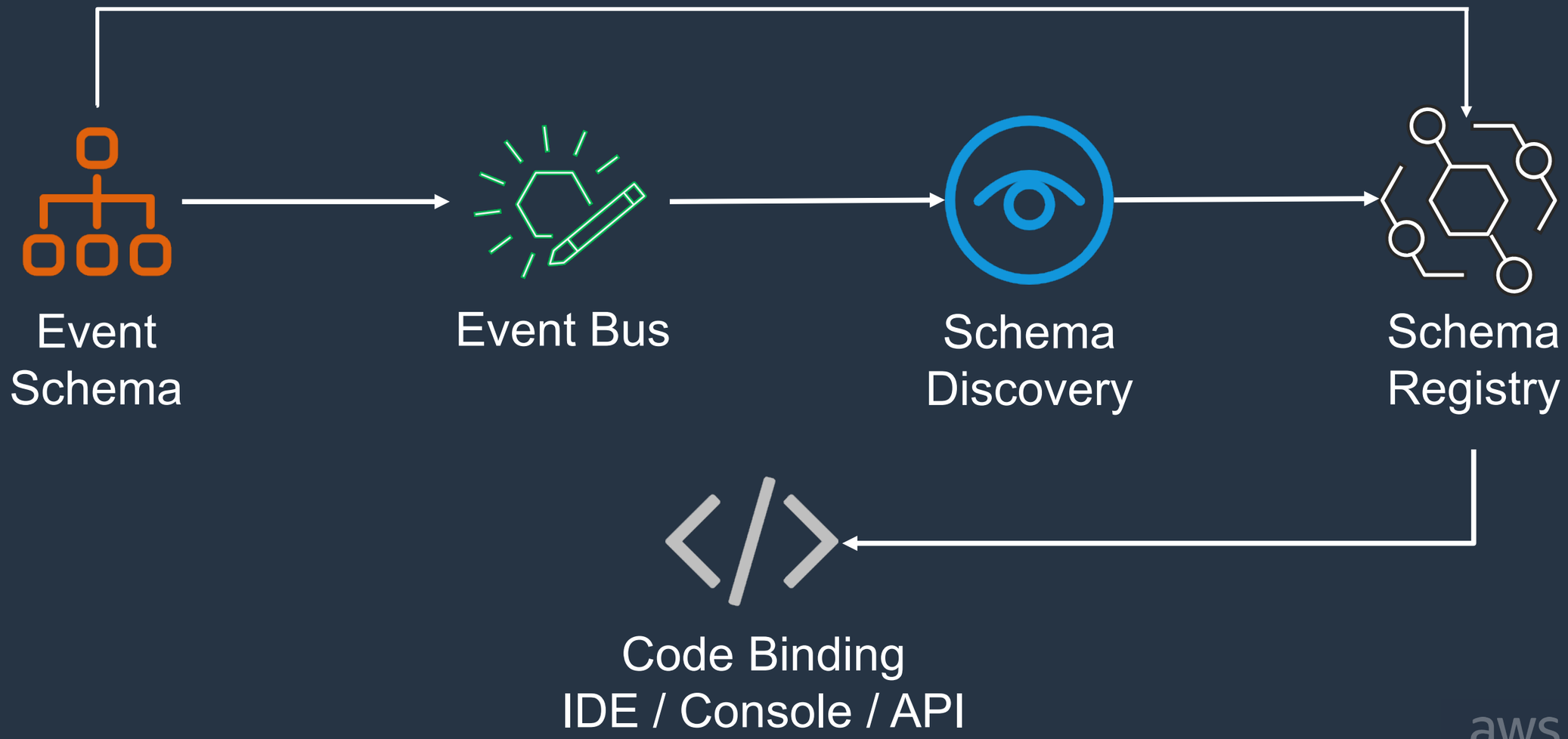
Pact fileのサンプル

```
{
    "consumer": {
        "name": "Consumer"
    },
    "provider": {
        "name": "Provider"
    },
    "messages": [
        {
            "description": "Published credit data",
            "providerState": "or maybe 'scenario'? not sure about this",
            "contents": {
                "foo": "bar"
            },
            "metaData": {
                "contentType": "application/json"
            }
        }
    ]
}
```

プロバイダからの変更はどうする?



Amazon EventBridge
- Schema Registry and Discovery



aws DevAx connect

Schema Registry & Schema Discovery

- Schema Registry - EventBridge (CloudWatch Events) でやり取りされる様々なデータのスキーマを、コンソールからまとめて公開
  - (例) EC2 インスタンスの状態変化通知のスキーマ

aws.ec2@EC2InstanceStateChangeNotification

Version 1 Created on Dec 1, 2019, 09:11 AM GMT+9

Action

```
"components": {
  "schemas": {
    "AWSEvent": {
      "type": "object",
      "required": ["detail-type", "resources", "id", "source", "time", "detail", "region", "version", "account"],
      "x-amazon-events-detail-type": "EC2 Instance State-change Notification",
      "x-amazon-events-source": "aws.ec2",
      "properties": {
        "detail": {
          "$ref": "#/components/schemas/EC2InstanceStateChangeNotification"
        },
        "detail-type": {
          "type": "string"
        },
        "resources": {
```

プロパティの一覧とそれぞれの型 (String や独自の型など)

Version 1 Created on Dec 1, 2019, 09:11 AM GMT+9

```
    },
    "EC2InstanceStateChangeNotification": {
      "type": "object",
      "required": ["instance-id", "state"],
      "properties": {
        "instance-id": {
          "type": "string"
        },
        "state": {
          "type": "string"
        }
      }
    }
```

独自の型の定義も確認できる

aws DevAx connect

Schema Registry & Schema Discovery

- Schema Discovery
  - EventBridge へ送信されたイベントからスキーマ情報を自動生成
  - (例) MongoDB Atlas (MongoDB 社にて提供されるマネージドサービス) へデータの CRUD 処理が行われた場合のスキーマ

aws.partner-mongodb.com-stitch.trigger-5dee176ba96a577ca57e00a8@MongoDB...

Version 4 Created on Dec 9, 2019, 07:46 PM GMT+9 | Version 4 | Save as a new version | Action


```
"schemas": {
  "AWSEvent": {
    "type": "object",
    "required": ["detail-type", "resources", "detail", "id", "source", "time", "region", "version", "account"],
    "x-amazon-events-detail-type": "MongoDB Database Trigger for sample_airbnb.listingsAndReviews",
    "x-amazon-events-source": "aws.partner/mongodb.com/stitch.trigger/5dee176ba96a577ca57e00a8",
    "properties": {
      "detail": {
        "$ref": "#/components/schemas/MongoDBDatabaseTriggerForSample_airbnb.listingsAndReviews"
      },
      "account": {
        "type": "string"
      },
      "detail-type": {
        "type": "string"
```

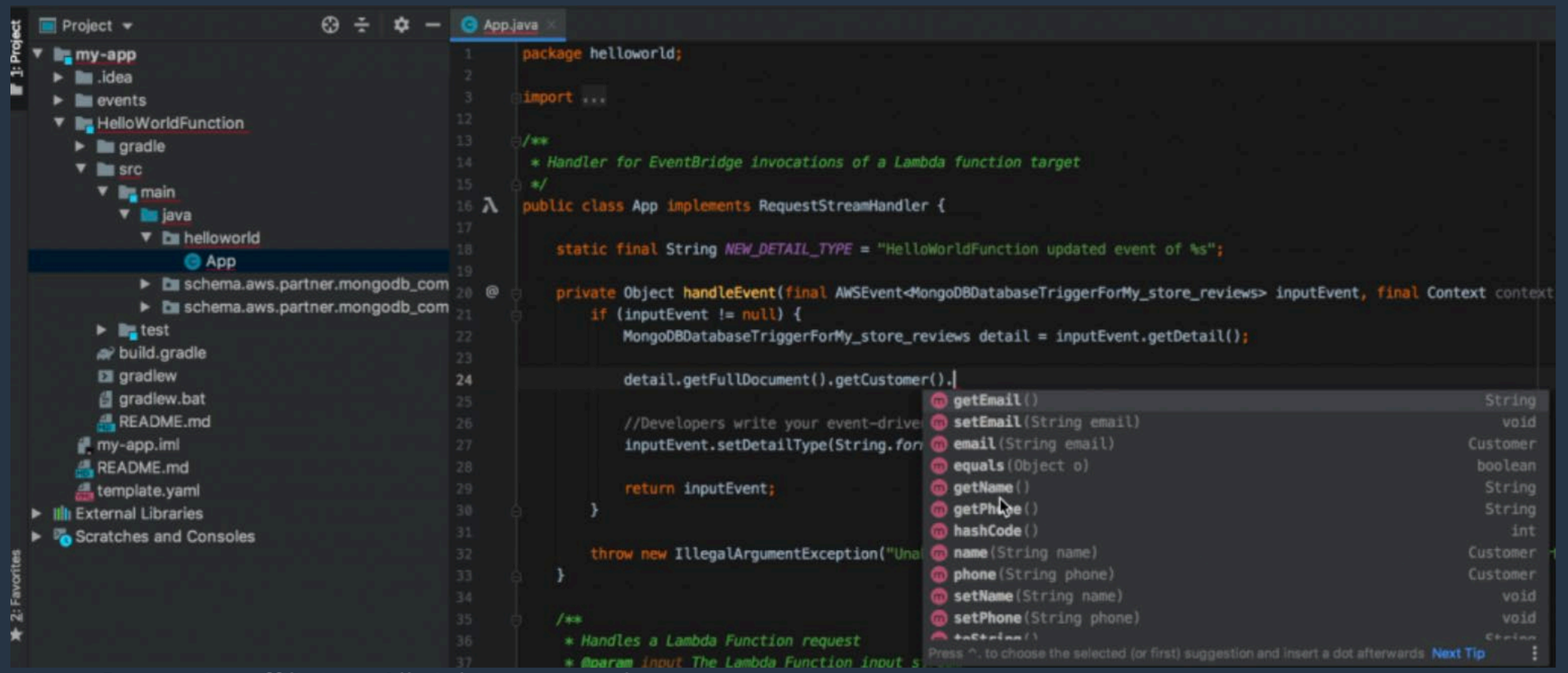

プロパティの一覧とそれぞれの型 (String や独自の型など)

AWS のサービスと全く同じ形で、パートナーから送信されるデータやカスタムイベントのスキーマも確認できる

aws DevAx connect

Schema Registry & Schema Discovery

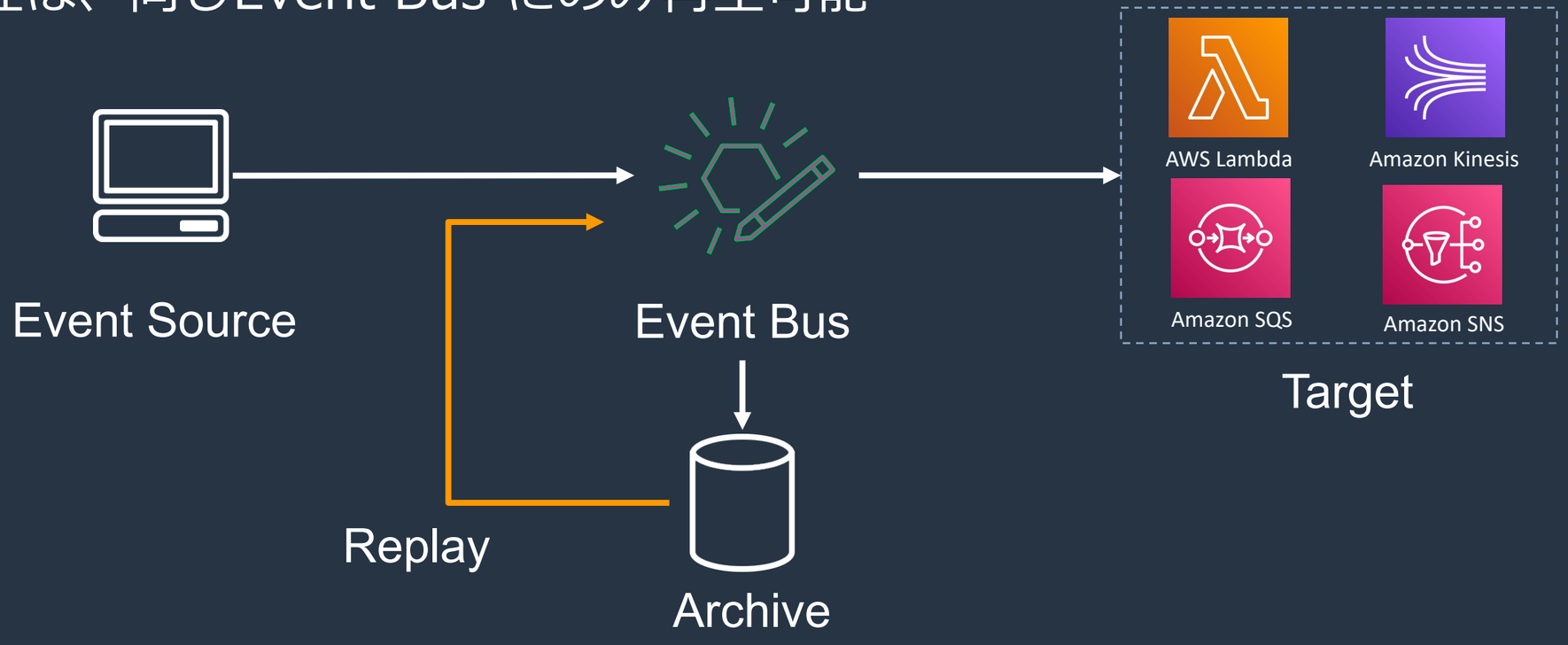
- Code Bindings – スキーマ定義を言語ごとにダウンロード可能
  - 2021 年 7 月時点で Java 8+、Python 3.6+、TypeScript 3+ に対応
  - アプリケーション開発の際に、ダウンロードしたスキーマ定義をインポートしておくことで、コード補完やコンパイル時のエラー検出に役立てることができる



aws DevAx connect

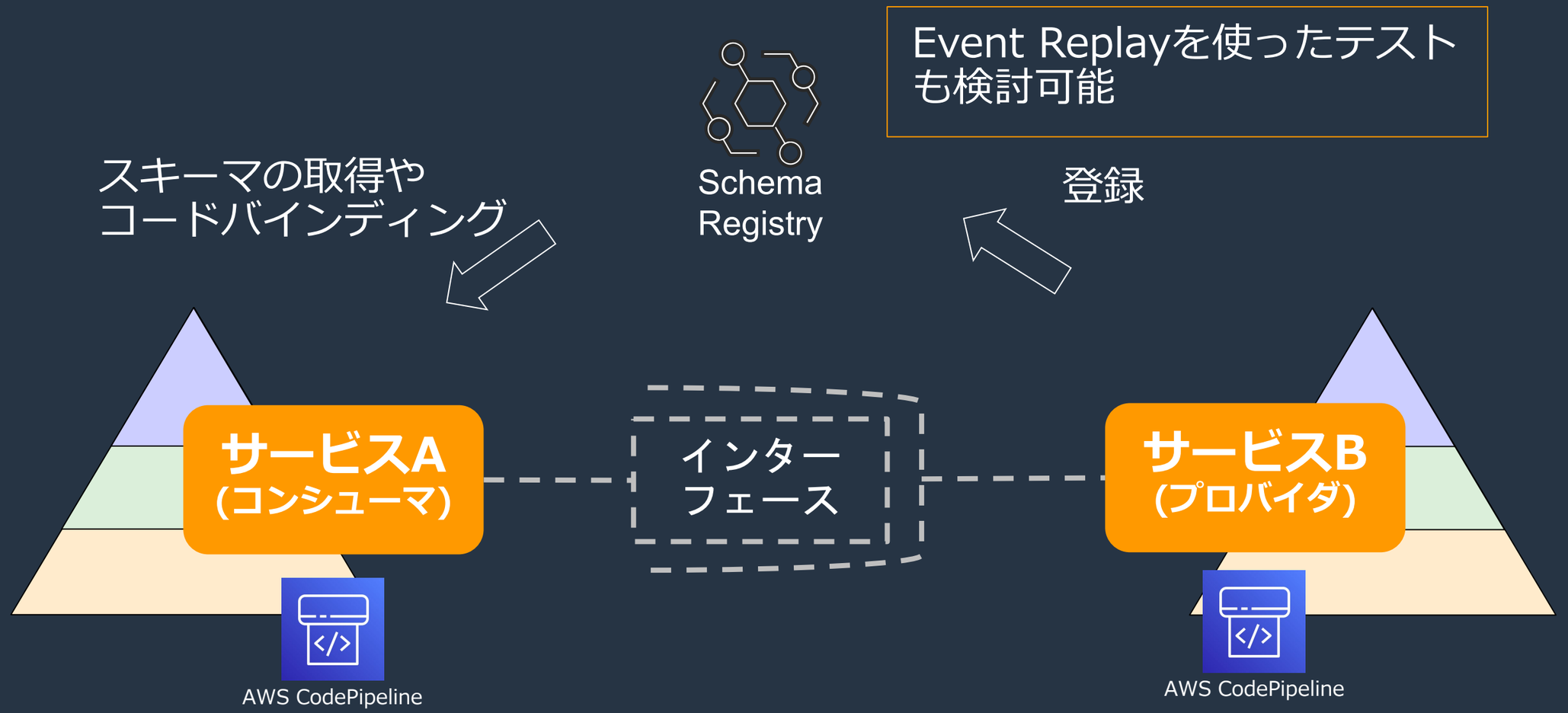
Event Replay

- Archive – 指定した(全ても可能)パターンのイベントを保存する
- Replay – Archive したイベントを再生できる
  - 開始/終了時間とルール(全ても可能)して再生する
  - 順番は保存された順序を保証しない
  - 現在は、同じEvent Bus にのみ再生可能



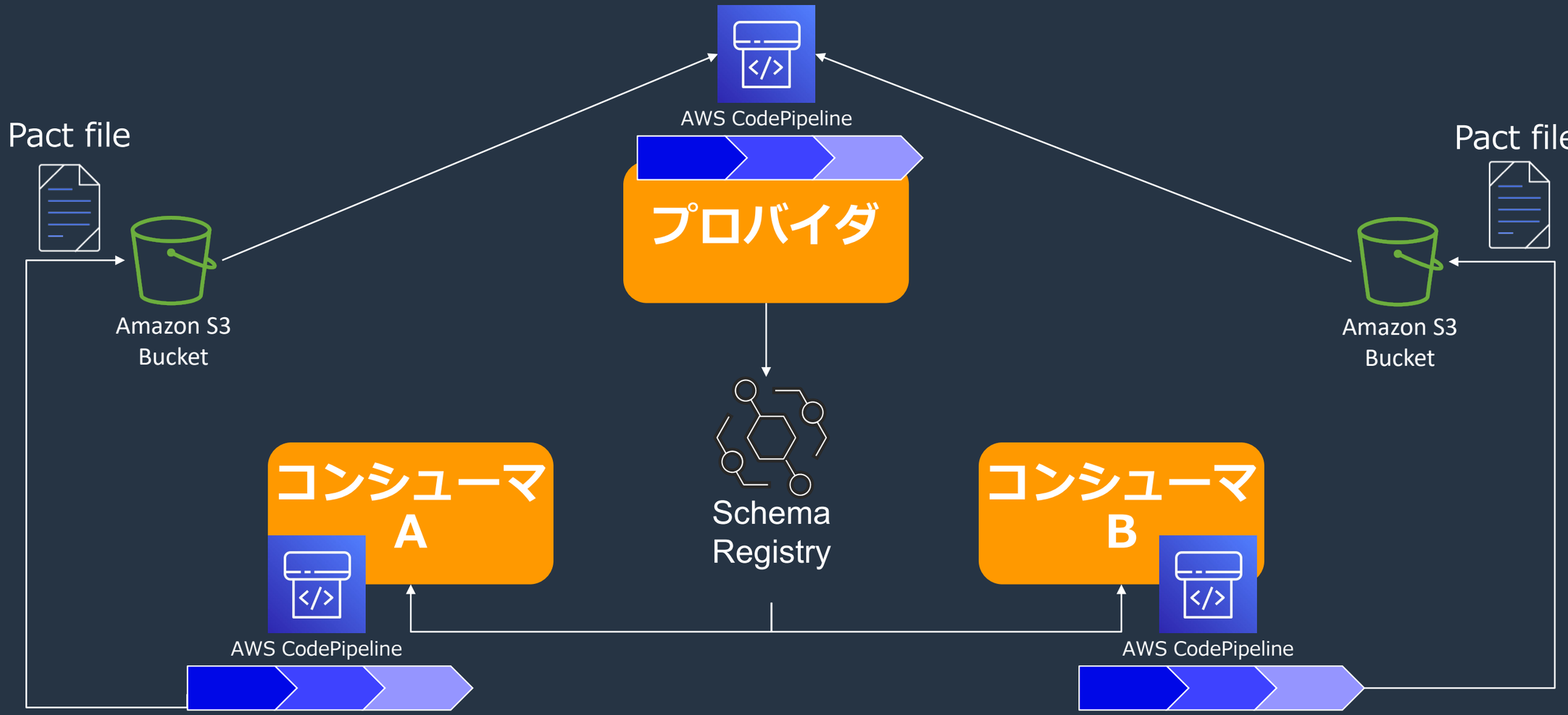
aws DevAx connect

EventBridge でイベントの仕様を連携



aws DevAx connect

実装例:PACT とEventBridge による連携



aws DevAx connect

5.まとめ

まとめ

イベント駆動のテストの難しさ
→ **イベント駆動を導入するための税金**

(今後、成熟していく領域であるはず)

<u>難しさを軽減するために</u>

- コアドメインをPureに保ち、単体テストを充実させる
- E2Eテストを過度に増やしすぎない
- 協調しやすい方法を検討(CDC やSchema の共有)

Q&A

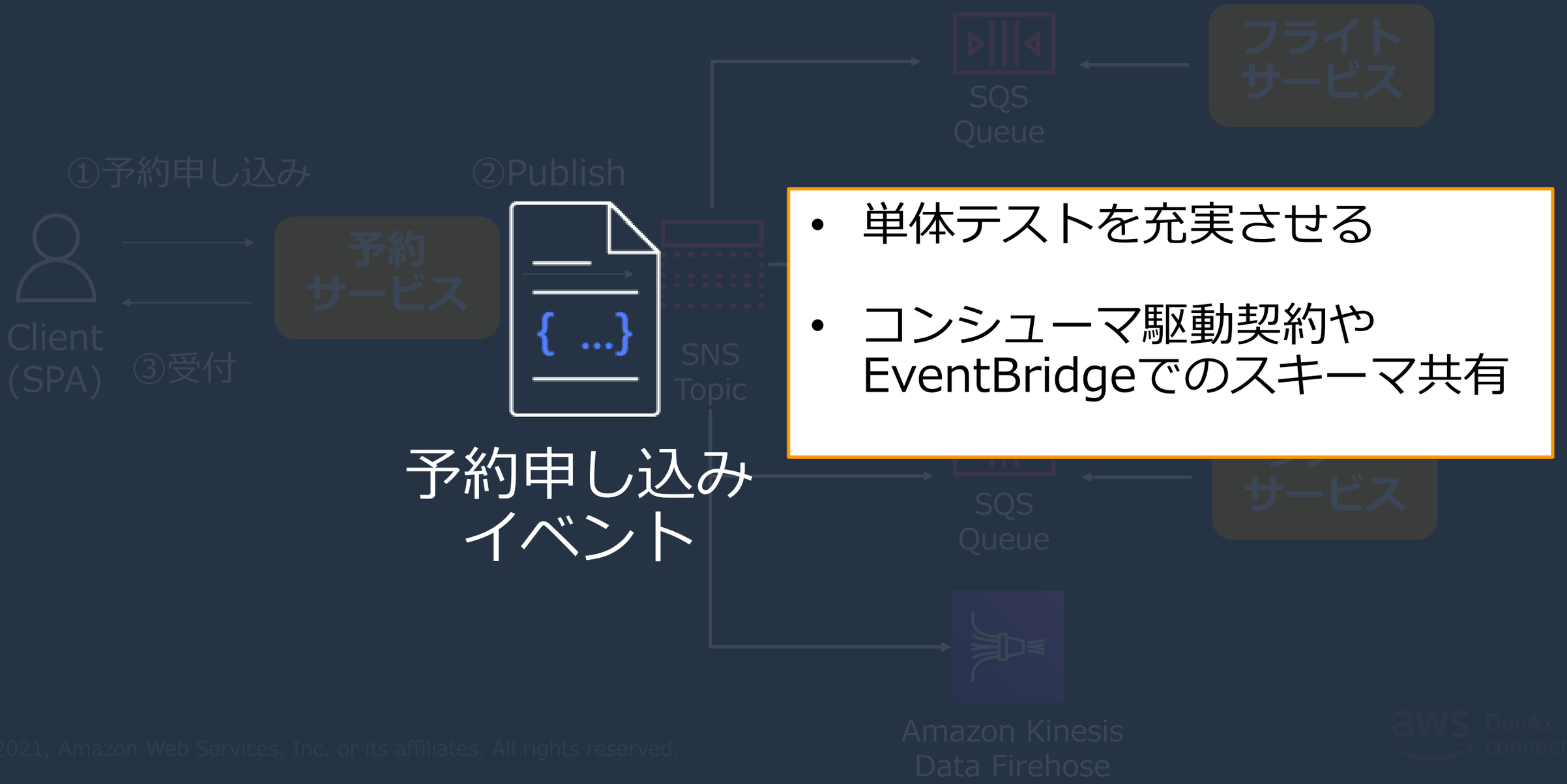
aws DevAx connect

お疲れ様でした!!
最後に少しだけ次回予告

aws DevAx connect

イベントのテスト

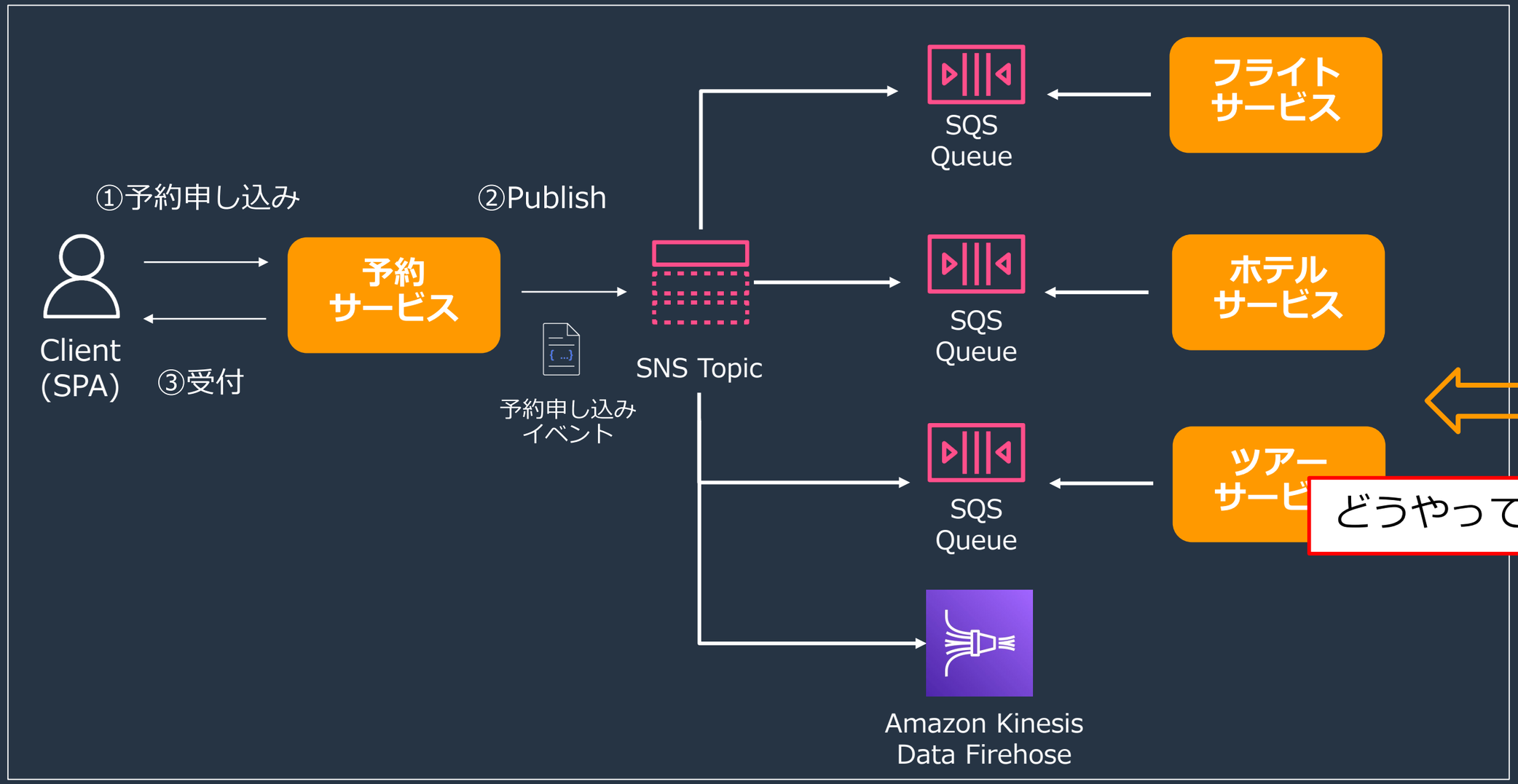
#devaxconnect



- 単体テストを充実させる
- コンシューマ駆動契約や EventBridgeでのスキーマ共有

次のお題:外部のSaaSを使いたい

#devaxconnect



aws DevAx connect

第7回
Amazon EventBridge を使って イベントドリブンな外部サービス連携を実現する

櫻谷 広人
アマゾン ウェブ サービス ジャパン株式会社
パートナーソリューションアーキテクト

SaaS やそのほかの外部サービスとの連携はモダンなアプリケーション開発において、コア機能に集中しアジリティ高くサービスを成長させるために必須になってきています。しかし、SaaS との連携は、考慮/調整事項が多く、結果的に工数が大きくなってしまうことも。Amazon EventBridge の SaaS 連携機能を活用し、SaaS との連携もイベント駆動でシンプルに実現する方法を解説します

aws DevAx connect

アンケートご協力のお願い

アンケートにご協力ください!!

- 登壇者へのフィードバック

- 次回以降のDevAx::connect について
  - 開催方法や時間帯など
  - 取り上げて欲しいテーマ

  DevAx::connect は**皆さんと共に作っていきたい**と考えています!!

aws DevAx connect

Thank you!

aws DevAx connect