IBM App Connect Enterprise V12

Change Data Captureノード

日本アイ・ビー・エム システムズ・エンジニアリング

はじめに

- 当資料では、App Connect Enterprise V12のFix Pack 12.0.10.0で追加されたChange Data Captureノードについて解説します。
- 記載内容は、ACE V12.0.11.0およびV12.0.12.2をベースにしており、今後のFixpackやバージョンで変更される場合があります。
- Db2とPostgreSQLで検証した結果を記載しています。

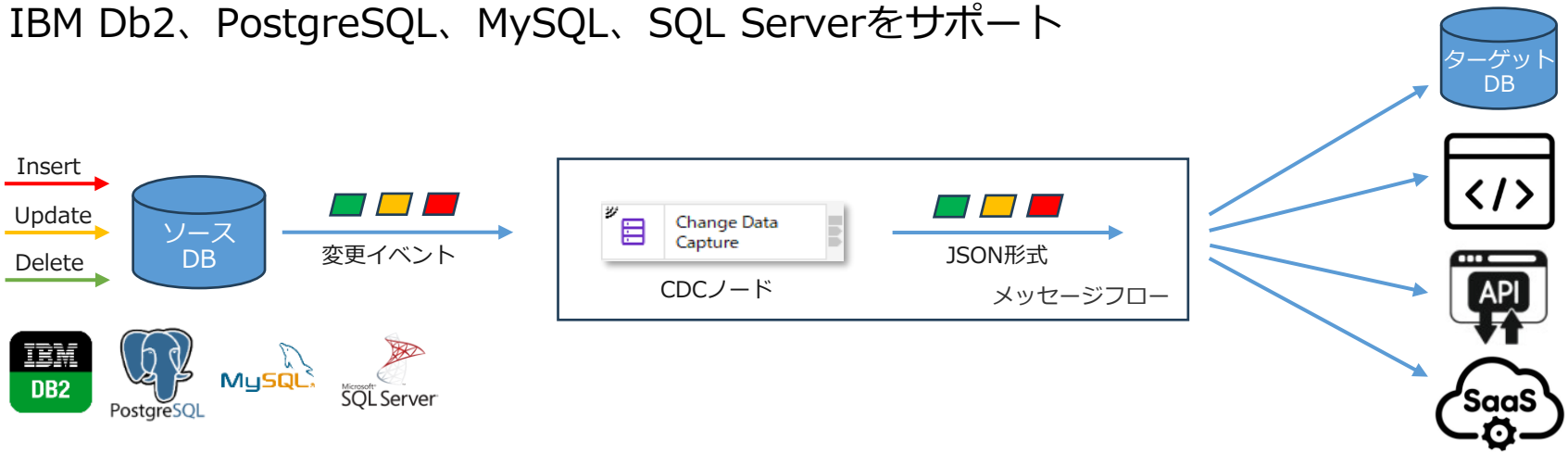
内容

- Change Data Captureノード
  - Change Data Captureノード
  - CDCノードの基本動作
  - 【補足】ソースDBとしてDb2を利用する際の考慮事項
- 環境セットアップ
  - セットアップの流れ
  - 1. データベース環境のセットアップ
  - 2. ACE環境のセットアップ
- フロー開発
  - ポリシー定義
  - CDCノードの設定
- データ・フォーマット
  - レコード・データのフォーマット
  - DDLデータのフォーマット
  - 実際のスナップショットのレコード・データ
  - 実際の変更イベントのレコード・データ

Change Data Captureノード

Change Data Captureノード

- ACE v12.0.10からChange Data Capture(CDC)ノードを提供
  - CDCとは、データベースの変更をリアルタイムにキャプチャする機能
- 実体はDebeziumが提供するコネクターを利用
  - Debeziumとは、主要なデータベースに対するCDC機能を提供するオープンソース・プラットフォーム
    - https://debezium.io/
- ソースDBに加えられた変更(Insert/Update/Delete)をリアルタイムにキャプチャし、フローにレコード・データを含む変更イベント(JSON形式)を取り込むことが可能
  - 変更イベントを加工し、様々なターゲット・システムにニア・リアルタイムに連携することが可能となる
- IBM Db2、PostgreSQL、MySQL、SQL Serverをサポート



CDCノードの基本動作

- キャプチャ対象のテーブルの1レコード・データを1メッセージとしてフローに取り込む
- データを取り込むタイミング
  - 新規フローの初回デプロイ時
    - デプロイした時点のテーブルの全レコード・データを1件ずつフローに取り込む(スナップショット)
    - 同じフローを再デプロイした場合は、スナップショットのレコード・データは流れない
  - 初回デプロイ以降
    - テーブルに対するINSERT/UPDATE/DELETE処理のたびに変更されたレコード・データを1件ずつフローに取り込む（変更イベント）
- DDLデータが取り込まれる場合がある
  - レコード・データの前にテーブルのDDL情報を含むデータ（DDLデータ）が取り込まれる場合がある

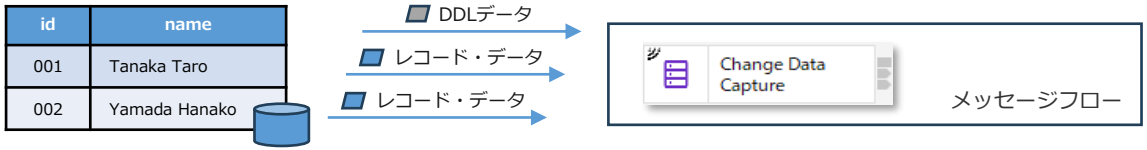
※ソースDBの種類やバージョンによって動作が変わる場合があります。
DDLデータは基本的にフロー内では利用しないため、後段のノードでDDLデータを読み捨てるフィルタリング処理を実装する必要があります。

- レコード・データはJSON形式
  - CDCノードではレコード・データをJSONドメインで処理
  - データ・フォーマットの詳細は後述

CDCノードの基本動作

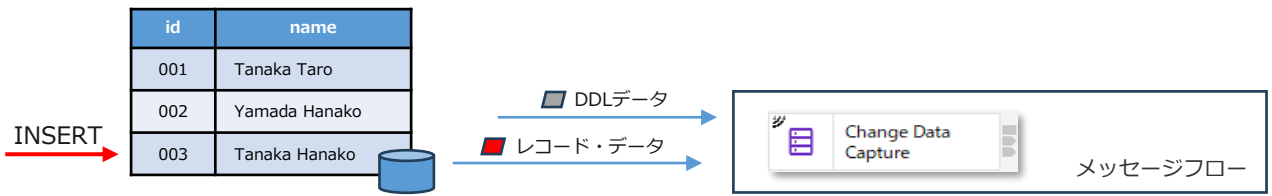
- Db2をソースDBとした場合の動作イメージ
  - 新規フローの初回デプロイ時
    - その時点の全レコード・データがフローに流れる（**スナップショット**）
      - ✓ 1件目には、DDLデータが入力される
      - ✓ 2件目以降に、テーブルの各レコード・データが入力される

<デプロイ時点のテーブルの状態>



  - 初回デプロイ以降
    - INSERT/UPDATE/DELETE処理の都度、変更情報を含むレコード・データが1件ずつフローに流れる（**変更イベント**）
      - ✓ 1件目にDDLデータ、2件目にレコード・データが入力される

<テーブルに対してINSERT処理を行う場合>



【補足】ソースDBとしてDb2を利用する際の考慮事項

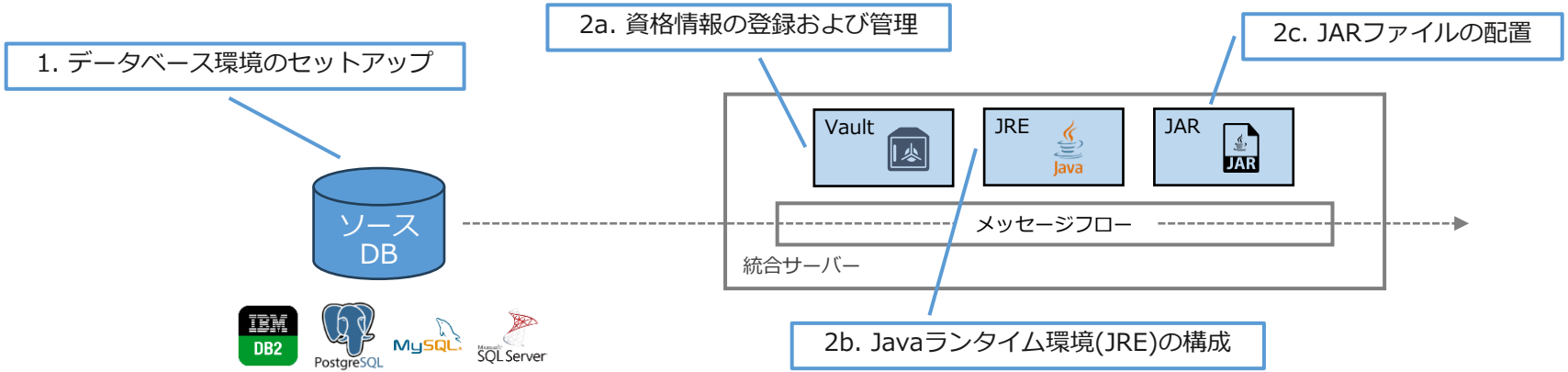
- IIDRのライセンスが必要
  - Db2からデータをキャプチャする場合は、IBM InfoSphere Data Replication(IIDR)のライセンスが必要
    - DebeziumのDb2用コネクターは、内部的にIIDR SQL Replicationのキャプチャ機能を利用
- ソースDBへの負荷がかかる
  - SQL Replicationでは、ソース表への更新処理と共に、Capture ProgramによるCD表(*1)への更新処理やCDCノードからのポーリング処理のSQL負荷がソースDBにかかる
  - Db2で利用可能な他のレプリケーション方式（Q Replication、CDC Replication）と比較して、ソースDBへの負荷が高い

- 変更イベントとともに付随情報も取り込まれる
  - Db2をソースとした構成では、変更イベントの取り込みの都度、DDL情報も別メッセージとしてフローに取り込まれるため、フロー内でフィルタリング処理を実装する必要がある
- Db2に対する更新処理をメッセージフローに取り込む方法として以下の方法も検討
  - DatabaseInputノード
    - Db2のトリガー機能を利用してソース表の変更イベントをメッセージフローに取り込む方法
    - トリガー機能を利用した場合も、ソース表の更新情報がイベント・ストア（通常、データベース表）に記録され、DatabaseInputノードからイベント・ストアをポーリングすることになるため、ソースDBに対する負荷の観点ではCDCノードと同等
    - https://www.ibm.com/docs/ja/app-connect/12.0?topic=databases-event-based-database-integration
  - Q ReplicationのEvent Publisher機能＋MQInputノード
    - Q Replicationのキャプチャ機能を利用して、ソース表の変更イベントをMQに連携できる機能
    - MQInputノードを利用してメッセージフローに取り込むことができる
    - Capture Programがソース表の更新をログから取得し、直接MQキューに格納するため、上記他の方法と比較して、ソースDBに対する負荷を抑えられるメリットがある
    - https://www.ibm.com/docs/en/idr/11.4.0?topic=po-event-publishing

環境セットアップ

セットアップの流れ

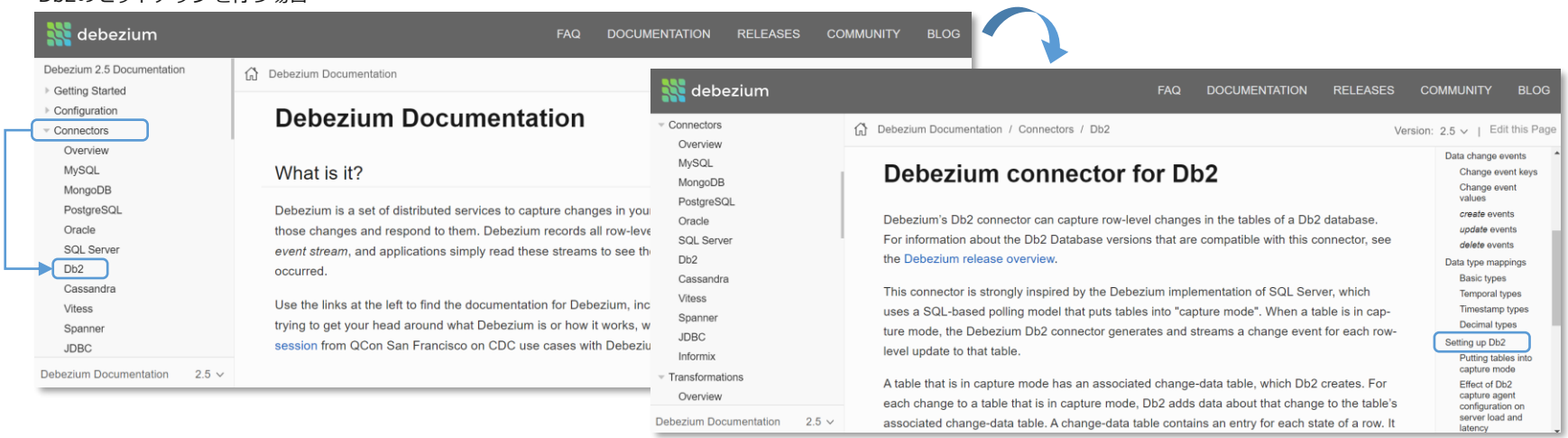
1. データベース環境のセットアップ
   - ◆ CDCを実行するための設定
2. ACE環境のセットアップ
   a. 資格情報の登録および管理
      - ◆ データベース接続時に使用する資格情報の登録
   b. Javaランタイム環境(JRE)の構成
      - ◆ CDCノードが機能するJREのバージョンを指定
   c. JARファイルの配置
      - ◆ Debeziumコネクター JARファイル
      - ◆ Db2 JDBCドライバ JARファイル（Db2のみ）
        - ※ライセンス要件により、Debezium Db2 コネクター・アーカイブにDb2 JDBCドライバが含まれていないため



1. データベース環境のセットアップ

- CDCノードを使用するために必要な設定を確認する
  - Debezium(https://debezium.io/documentation/)にアクセスし、使用するDebeziumのバージョンを選択
    - ACE(V12.0.12.2時点)がサポートするDebeziumのバージョンは、V2.5.*x*
    - Debeziumの各バージョンと互換性のあるデータベースおよびバージョンは以下を参照
      - ✓ https://debezium.io/releases/
  - 画面左側のConnectorsタブより、使用するデータベースを選択
  - 画面右側の見出しより「Setting up <データベース>」をクリックし、指示に沿ってセットアップを行う

Db2のセットアップを行う場合



- その他参考資料：Red HatのDebeziumユーザーガイド
  - https://access.redhat.com/documentation/ja-jp/red_hat_integration/2023.q4/html/debezium_user_guide/index

2. ACE環境のセットアップ

a. 資格情報の登録および管理

- データベース接続に必要な資格情報の登録、および資格情報を管理するための構成が必要
  - 資格情報を管理するためのVaultを構成
    - 資格情報管理の方法はVault以外でも可能
      - ✓ 詳細は「App Connect Enterprise 資格情報管理」を参照
        - https://www.ibm.com/support/pages/node/7108203

作成済みの統合ノードに対してVaultを構成する例

```
$ mqsivault NODE1 --create --vault-key myvaultkey
BIP15293I: Node or workdir vault key obtained from --vault-key command line argument.

BIP8071I: コマンドは正常終了しました。
```

  - データベースの接続に必要な資格情報を登録

資格情報のタイプにCDCを指定して資格情報を登録する例

```
$ mqsicredentials NODE1 --create --integration-server SERVER1 --credential-type cdc --credential-name db2inst1
--username <username> --password <password>
BIP15119I: タイプ 'cdc' の資格情報名 'db2inst1' の 'create' アクションは正常に終了しました。

BIP8071I: コマンドは正常終了しました。
```

2. ACE環境のセットアップ

b. Javaランタイム環境(JRE)の構成

- CDCノードでは、Java 11を指定して統合サーバーを構成する必要がある
  - 指定されたバージョンのJREを使用するように統合サーバーを構成可能
  - デフォルトでは、統合サーバーはACEで提供されるJava 8を使用
    - Java 11も同梱されている
  - ACEの一部機能は、Java 11で動作しない点に注意
    - 詳細は以下を参照
      - ✓ https://www.ibm.com/docs/en/app-connect/12.0?topic=cis-configuring-integration-server-use-specified-java-runtime-environment-jre
    - Java 11で利用できない機能とCDCノードを併用する場合は、統合サーバーを分けて構成
- 構成方法
  - 統合サーバーを停止
  - ibmint specify jreコマンドを実行して、ランタイムで使用するJREを指定
    - ibmint specify jreコマンドは指定されたJREから詳細を抽出し、 server.java.yamlファイルを生成
    - server.java.yamlファイルとserver.components.yamlファイルより、起動時に必要な情報が統合サーバーに提供される
    - ibmint specify jreコマンドの詳細は以下を参照
      - ✓ https://www.ibm.com/docs/en/app-connect/12.0?topic=commands-ibmint-specify-jre-command

Java 11を指定して統合サーバーを構成する例

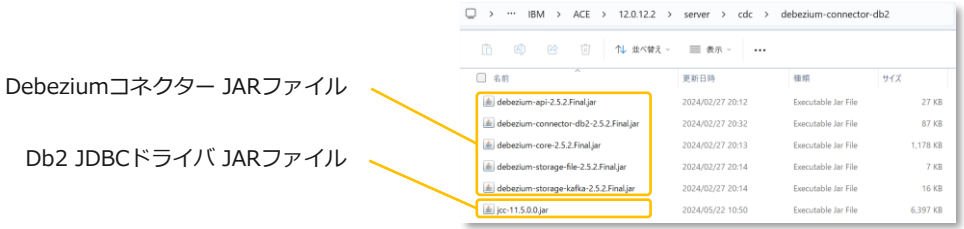
```
$ ibmint specify jre --version 11 --integration-node NODE1 --integration-server SERVER1

BIP15273I: The JRE to be used by this Integration Server has changed. This will take effect next time
the Integration Server is started.
```

  - 統合サーバーを起動

2. ACE環境のセットアップ

c. JARファイルの配置

- CDCノードには、使用するデータベースに固有のJARファイルが必要
  - Debeziumが提供するコネクターJARファイル
- JARファイルの取得方法
  - 「1. データベース環境のセットアップ」(p11)と同様、Debezium(https://debezium.io/documentation/)にアクセスし、使用するDebeziumのバージョンを選択
  - 画面左側のConnectorsタブより、使用するデータベースを選択
  - 画面右側の見出しより「Deployment」をクリックし、[Procedure]の1~3を参考に以下を実施
    - Debeziumに記載のリンクまたはMaven Central(https://central.sonatype.com/)より、使用するバージョンのDebeziumコネクターのプラグイン・アーカイブをダウンロード
    - ダウンロードしたファイルを稼働環境にコピーして展開
      - ✓ 複数のJARファイルを含むディレクトリを以下の所定の場所に配置
        - Linux : <*INSTALLATION_PATH*>/server/cdc
        - Windows : <*INSTALLATION_PATH*>¥server¥cdc
    - ※データベースにDb2を使用する場合は、併せて以下も行う
      - ✓ Debeziumに記載のリンクまたはMaven Central(https://central.sonatype.com/)より、Db2のJDBCドライバをダウンロード
        - ダウンロードしたJDBCドライバ JARファイルを、先に取得したDebeziumコネクターのJARファイルと同じディレクトリに配置



ディレクトリ構成例

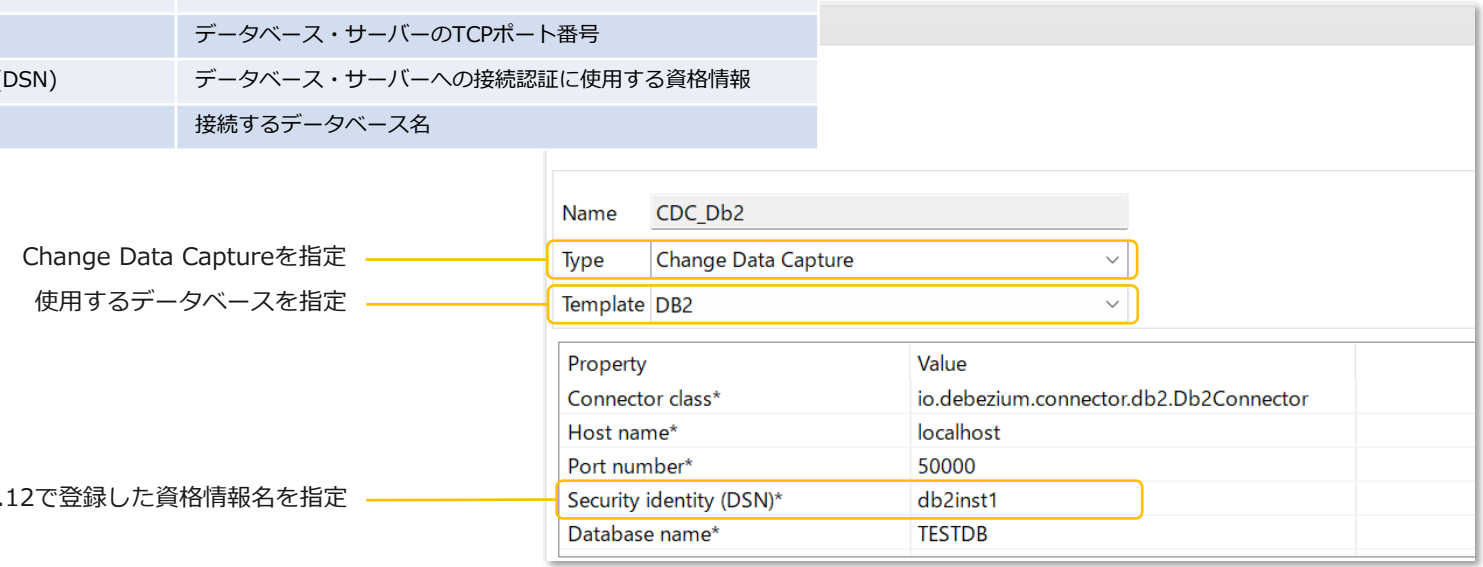
フロー開発

ポリシー定義

- データベース接続時に使用するChange Data Captureタイプのポリシーを作成
  - ポリシーのプロパティ情報

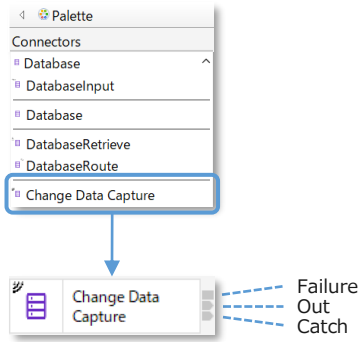
プロパティ	説明
Connector class	Debeziumコネクター名　※自動定義
Host name	データベース・サーバーのホスト名またはIPアドレス
Port number	データベース・サーバーのTCPポート番号
Security identity (DSN)	データベース・サーバーへの接続認証に使用する資格情報
Database name	接続するデータベース名

ポリシー定義例



CDCノードの設定

- パレットのConnectors->DatabaseからCDCノードをフローに配置し、プロパティを設定

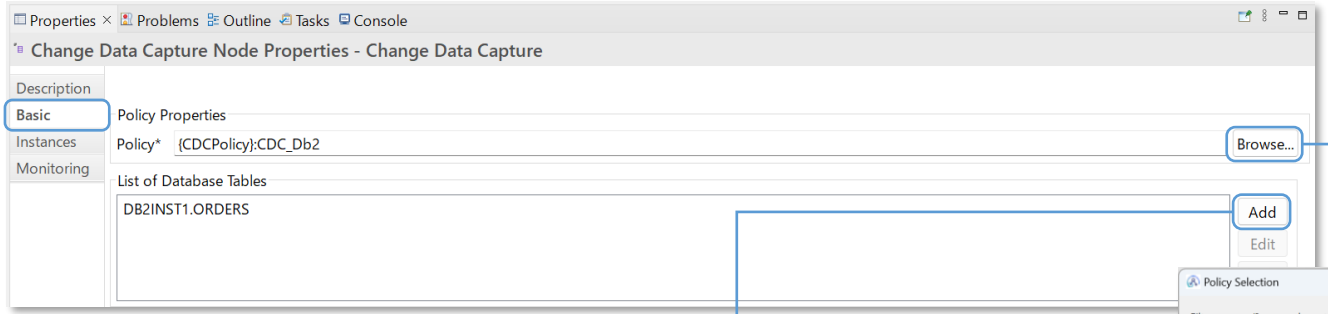


タブ	プロパティ	説明
Description	Node Name	ノード名
	Short Description	ノードについての簡潔な説明
	Long Description	メッセージフローにおけるノードの目的の説明
Basic	Policy	接続時に使用するポリシー
	List of Database Tables	キャプチャ対象のテーブルリスト
Instances	Additional instances pool	追加インスタンスを取得するプール - [Use Pool Associated with Message Flow]: 追加インスタンスはメッセージフロー・プールから取得 - [Use Pool Associated with Node]: Additional instancesプロパティで指定された数に基づいて、ノードの追加インスタンスから追加インスタンスを割り当て
	Additional instances	Additional instances poolプロパティがUse Pool Associated with Nodeに設定されている場合に、ノードが起動できる追加インスタンスの数
Monitoring	Events	ノードに対して定義するイベント

CDCノードの設定

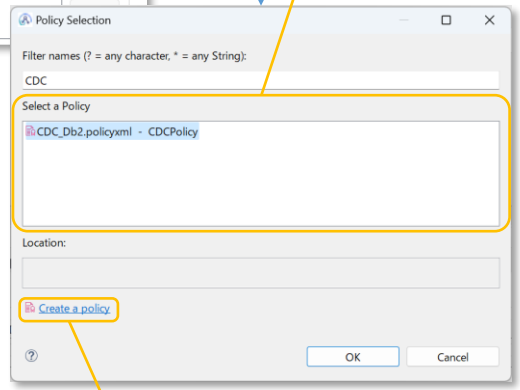
- 設定例

CDCノードの設定例

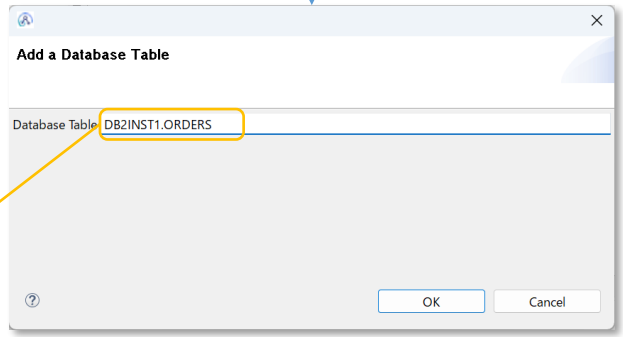


p.16で定義したポリシーを指定

ポリシーの指定画面



テーブルの指定画面



<スキーマ名>.<テーブル名>の形式で、
キャプチャ対象のテーブルを指定

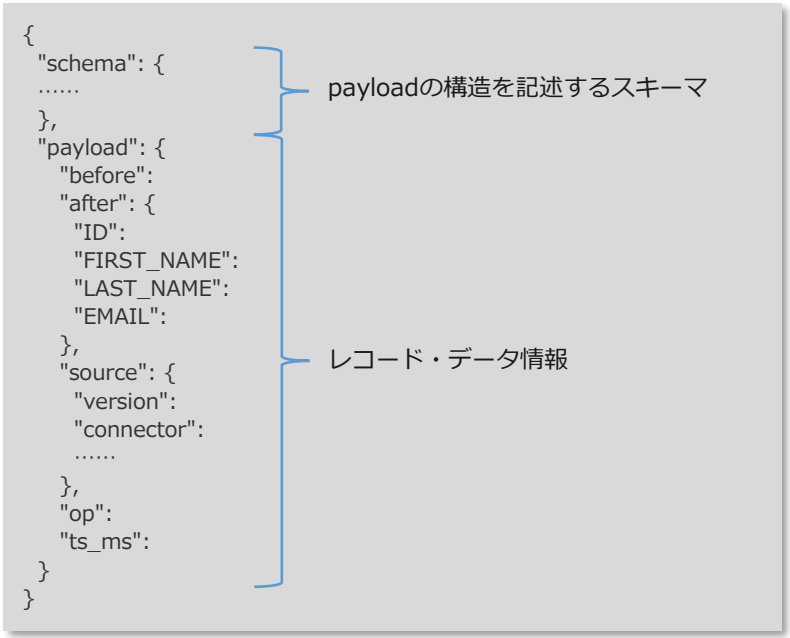
新規にポリシーを作成することも可能

データ・フォーマット

レコード・データのフォーマット

- レコード・データのフォーマット
  - schemaフィールドとpayloadフィールドで構成される

レコード・データのフォーマット



レコード・データのフォーマット

- 主要なフィールド

レコード・データの内容

```
{
 “schema”: { ····· ①
  "type": "struct",
  "fields": [
   {
    "type": "struct",
    "fields": [
     {
      "type": "int32",
      "optional": false,
      "field": "ID"
     },
     {
      "type": “string”,
      "optional": false,
      "field": “FIRST_NAME"
     },
     {
      "type": “string”,
      "optional": false,
      "field": “LAST_NAME"
     },
     {
      "type": “string”,
      "optional": false,
      "field": “EMAIL"
     }
    ],
    "optional": true,
    "name": “mydatabase.MYSCHEMA.CUSTOMERS.Value", ····· ②
    "field": "before"
   },
```

番号	フィールド	説明
①	schema	payloadの構造を記述するスキーマ
②	name	payloadのbeforeまたはafterフィールドのスキーマ ※スキーマの形式は <*logicalName.schemaName.tableName*.Value>で、 スキーマ名はデータベース内で一意

レコード・データのフォーマット

- 主要なフィールド（続き）

レコード・データの内容（続き）

```
……
    {
      "type": "struct",
      "fields": [
        {
          "type": "string",
          "optional": false,
          "field": "version"
        },
……
      ],
      "optional": false,
      "name": "io.debezium.connector.db2.Source", ····· ③
      "field": "source"
    },
    {
      "type": "string",
      "optional": false,
      "field": “op"
    },
    {
      "type": “int64",
      "optional": true,
      "field": “ts_ms"
    },
  ],
  "optional": false,
  "name": “mydatabase.MYSCHEMA.CUSTOMERS.Envelope" ····· ④
},
```

番号	フィールド	説明
③	name	payloadのsourceフィールドのスキーマ
④	name	payload全体のスキーマ

レコード・データのフォーマット

- 主要なフィールド（続き）

レコード・データの内容（続き）

```
"payload": { ····· ⑤
 "before": null, ····· ⑥
 "after": { ····· ⑦
   "ID": 10001,
   "FIRST_NAME": "john",
   "LAST_NAME": "doe",
   "EMAIL": "john.doe@example.org",
 },
 "source": { ····· ⑧
   "version": "2.3.7.Final",
   "connector": "db2",
   "name": "myconnector",
   "ts_ms": 1559729468470,
   "snapshot": "false",
   "db": "mydatabase",
   "schema": "MYSCHEMA",
   "table": "CUSTOMERS",
   "change_lsn": "00000027:00000758:0003",
   "commit_lsn": "00000027:00000758:0005“,
 },
 "op": “c”, ····· ⑨
 "ts_ms": 1559729471739 ····· ⑩
}
}
```

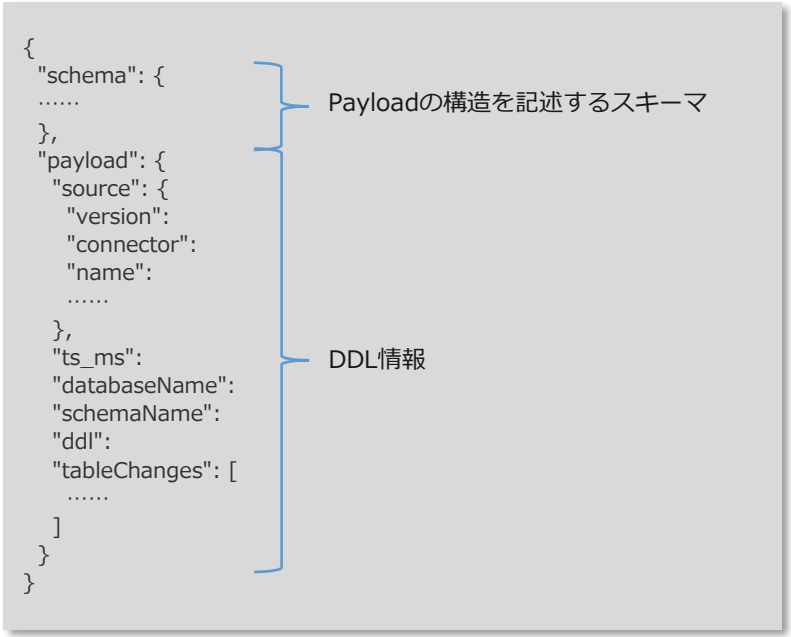
番号	フィールド	説明
⑤	payload	実際のデータ
⑥	before	イベント発生前のレコードの状態
⑦	after	イベント発生後のレコードの状態
⑧	source	イベントソースのメタデータ
⑨	op	コネクターがイベントを生成する原因となった操作タイプ ・c = create（作成） ・u = update (更新) ・d = delete (削除) ・r = read (スナップショット)
⑩	ts_ms	コネクターがイベントを処理した時刻

＜各フィールドについての詳細はDebeziumを参照＞
https://debezium.io/documentation/reference/2.5/connectors/db2.html#db2-change-event-values

DDLデータのフォーマット

- DDLデータのフォーマット
  - schemaフィールドとpayloadフィールドで構成される

DDLデータのフォーマット



DDLデータのフォーマット

- 主要なフィールド

DDLデータの内容

```
{
 "schema": {
  ……
 },
 "payload": {
  "source": {
   "version": "2.5.4.Final",
   "connector": "db2",
   "name": "db2",
   "ts_ms": 0,
   "snapshot": "true",
   "db": "testdb",
   "schema": "DB2INST1",
   "table": "CUSTOMERS",
   "change_lsn": null,
   "commit_lsn": "00000025:00000d98:00a2“,
   "event_serial_no": null
  },
  "ts_ms": 1588252618953,  ①
  "databaseName": "TESTDB",  ┐
  "schemaName": "DB2INST1",  ┘ ②
  "ddl": null,  ③
  "tableChanges": [  ④
   {
    "type": "CREATE",  ⑤
    "id": "¥"DB2INST1¥".¥"CUSTOMERS¥"",  ⑥
    "table": {  ⑦
     "defaultCharsetName": null,
     "primaryKeyColumnNames": [  ⑧
      "ID"
     ],
```

番号	フィールド	説明
①	ts_ms	コネクターがイベントを処理した時刻
②	databaseName schemaName	データベース名 スキーマ名
③	ddl	スキーマ変更の原因となったDDL
④	tableChanges	DDLコマンドの実行によって生成された、 スキーマ変更を含む1つ以上の項目の配列
⑤	type	変更タイプ - CREATE (テーブルの作成) - ALTER (テーブルの変更) - DROP (テーブルの削除)
⑥	id	作成、変更、削除されたテーブルの識別子
⑦	table	変更後のテーブルのメタデータ
⑧	primaryKeyColumnName	テーブルの主キーを構成するカラムのリスト

DDLデータのフォーマット

- 主要なフィールド（続き）

DDLデータの内容（続き）

```
        "columns": [ ·····  ⑨
          {
            "name": "ID",
            "jdbcType": 4,
            "nativeType": null,
            "typeName": "int identity",
            "typeExpression": "int identity",
            "charsetName": null,
            "length": 10,
            "scale": 0,
            "position": 1,
            "optional": false,
            "autoIncremented": false,
            "generated": false,
          },
          ·····
        ],
        "attributes": [
          {
            "customAttribute": "attributeValue“  ·····  ⑩
          }
        ]
      }
    }
  ]
 }
}
```

番号	フィールド	説明
⑨	columns	変更されたテーブルの各カラムのメタデータ
⑩	attributes	各テーブルの変更に対するカスタム属性のメタデータ

<各フィールドについての詳細はDebeziumを参照>
https://debezium.io/documentation/reference/2.5/connectors/db2.html#about-the-debezium-db2-connector-schema-change-topic

実際のスナップショットのレコード・データ

- キャプチャ対象テーブルのスナップショット・データ

スナップショットのレコード・データ内容


```
{
  "schema": {
    ……
  },
  "payload": {
    "before": null,
    "after": {
      "ID": 10001,
      "ORDER_DATE": 16816,
      "PURCHASER": 1001,
      "QUANTITY": 1,
      "PRODUCT_ID": 102
    },
    "source": {
      "version": "2.5.2.Final",
      "connector": "db2",
      "name": "orders1212.FCMComposite_1_1",
      "ts_ms": 1716323443362,
      "snapshot": "first",
      "db": "TESTDB",
      "sequence": null,
      "schema": "DB2INST1",
      "table": "ORDERS",
      "change_lsn": null,
      "commit_lsn": "00000000:000025d3:0000000000049337"
    },
    "op": "r",
    "ts_ms": 1716355843372,
    "transaction": null
  }
}
```


デプロイ時点のテーブルのレコードデータがafterにセットされる
beforeはnull

操作タイプ（r: read）※スナップショット

実際の変更イベントのレコード・データ

- キャプチャ対象テーブルの変更イベント(INSERT)のフォーマット

レコードINSERT時の変更イベントのレコード・データ内容


```
{
  "schema": {
    ……
  },
  "payload": {
    "before": null,
    "after": {
      "ID": 10005,
      "ORDER_DATE": 19865,
      "PURCHASER": 1004,
      "QUANTITY": 1,
      "PRODUCT_ID": 104
    },
    "source": {
      "version": "2.5.2.Final",
      "connector": "db2",
      "name": "orders1212.FCMComposite_1_1",
      "ts_ms": 1716323584270,
      "snapshot": "false",
      "db": "TESTDB",
      "sequence": null,
      "schema": "DB2INST1",
      "table": "ORDERS",
      "change_lsn": "00000000:00000000:00000000046280b3",
      "commit_lsn": "00000000:00002714:000000000004cd13"
    },
    "op": "c",
    "ts_ms": 1716355984284,
    "transaction": null
  }
}
```


INSERTしたレコード・データがafterにセットされる
beforeはnull

操作タイプ（c: create)
※INSERT処理により、新たにレコードが追加されたことを指す

実際の変更イベントのレコード・データ

- キャプチャ対象テーブルの変更イベント(UPDATE)のフォーマット

レコードUPDATE時の変更イベントのレコード・データ内容


```
{
 "schema": {
  ……
 },
 "payload": {
  "before": {
   "ID": 10005,
   "ORDER_DATE": 19865,
   "PURCHASER": 1004,
   "QUANTITY": 1,
   "PRODUCT_ID": 104
  }
  "after": {
   "ID": 10005,
   "ORDER_DATE": 19864,
   "PURCHASER": 1004,
   "QUANTITY": 1,
   "PRODUCT_ID": 104
  },
  "source": {
   "version": "2.5.2.Final",
   "connector": "db2",
   "name": "corders1212.FCMComposite_1_1",
   "ts_ms": 1716323786027,
   "snapshot": "false",
   "db": "TESTDB",
   "sequence": null,
   "schema": "DB2INST1",
   "table": "ORDERS",
   "change_lsn": "00000000:00000000:0000000004641d50",
   "commit_lsn": "00000000:000027d0:000000000004ce8b"
  },
  "op": "u",
  "ts_ms": 1716356186029,
  "transaction": null
 }
}
```


beforeにはUPDATE前のレコード・データ、
afterにはUPDATE後のレコード・データがセットされる

操作タイプ（u: update)

実際の変更イベントのレコード・データ

- キャプチャ対象テーブルの変更イベント(DELETE)のフォーマット

レコードDELETE時の変更イベントのレコード・データ内容

```
{
  "schema": {
    ……
  },
  "payload": {
    "before": {
      "ID": 10005,
      "ORDER_DATE": 19864,
      "PURCHASER": 1004,
      "QUANTITY": 1,
      "PRODUCT_ID": 104
    }
    "after": null
    "source": {
      "version": "2.5.2.Final",
      "connector": "db2",
      "name": "orders1212.FCMComposite_1_1",
      "ts_ms": 1716323887056,
      "snapshot": "false",
      "db": "TESTDB",
      "sequence": null,
      "schema": "DB2INST1",
      "table": "ORDERS",
      "change_lsn": "00000000:00000000:00000000046589d3",
      "commit_lsn": "00000000:000027dc:000000000004d0c9"
    },
    "op": "d",
    "ts_ms": 1716356287059,
    "transaction": null
  }
}
```

beforeにDELETE前のレコード・データがセットされる
afterはnull

操作タイプ（d: delete)