イベント

ここでは、次の内容について説明します。

イベント処理の概要

イベント処理メカニズムにより、CWM はアウトバウンドおよびインバウンドのイベントを処理するために外部ブローカと連携できます。ワークフローは、新しいワークフローを開始したり、既存のワークフローを通知したりするために使用できるイベントのコンシューマまたはプロデューサとして機能できます。さらに、CWM で定義するイベントタイプごとに、イベントをフィルタ処理し、特定の属性値を含むイベントを待機しているワークフローにイベントをルーティングするための相関属性を追加できます。

イベントメッセージは、クラウドイベント仕様に従って定義する必要があります。詳細については、イベントフォーマットのセクションを参照してください。

ブローカとプロトコル

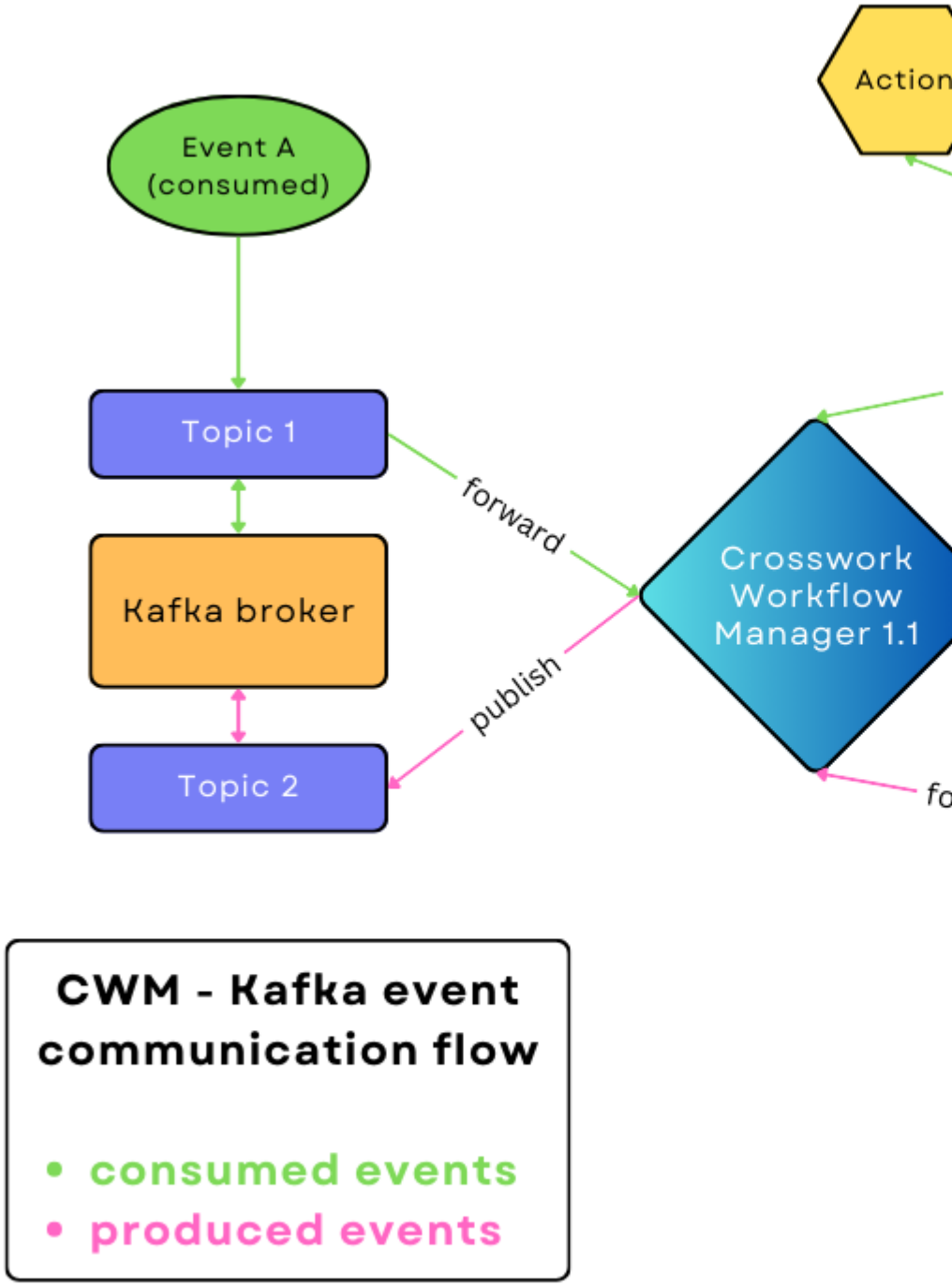
Crosswork Workflow Manager 1.1 は、イベントを処理するために Kafka ブローカと AMQP および HTTP プロトコルをサポートしています。イベントは、CWM 内で実行されているワークフローによって**消費**されるか、実行中のワークフローによって**生成**（ブローカによって転送される着信イベント）され、外部システムに転送（ブローカによって受信される発信イベント）されます。

（注） CWM はイベントブローカ自体として機能しないことに注意してください。外部ブローカに接続してメッセージ/イベントを転送する手段を提供します。

Kafka ブローカ

consume イベントの種類の場合、CWM は Kafka ブローカに接続し、トピックの特定のイベントタイプをリッスンします。特定のタイプのイベントが適切なトピックに登録されると、CWM はイベントデータを取得し、実行中のワークフローに転送します。次に、ワークフローはイベントステート内で定義されたアクションを実行するか、別のワークフロー実行を実行します（選択されている場合）。

produce イベントの種類の場合、実行中のワークフローは単一のイベントまたは一連のイベントを生成し、CWM がブローカに転送し、それらは適切なトピックで公開されます。



Kafkaブローカは、有効なコンテンツタイプが送信される限り、言語固有のSDKでサポートされているすべてのイベント メッセージ フォーマットを受け入れます。サポートされているフォーマットのリストは、https://github.com/cloudevents/spec?tab=readme-ov-file にあります。

AMQP プロトコル（RabbitMQ ブローカなど）

consume イベントの種類の場合、CWM は AMQP ブローカに接続し、キューで特定のイベントタイプをリッスンします。Kafka ブローカと同様に、特定のタイプのイベントが適切なキューに登録されると、CWM はイベントデータを取得し、実行中のワークフローに転送します。次に、ワークフローはイベントステート内で定義されたアクションを実行するか、別のワークフロー実行を実行します（選択されている場合）。

produce イベントの種類の場合、実行中のワークフローは単一のイベントまたは一連のイベントを生成し、CWM がブローカに転送し、それらは適切なキューで公開されます。

AMQP ブローカは、有効なコンテンツタイプが送信される限り、特定の SDK でサポートされているすべてのイベントメッセージフォーマットを受け入れます。サポートされているフォーマットのリストは、https://github.com/cloudevents/spec?tab=readme-ov-file にあります。

HTTP プロトコル

consume イベントの種類の場合、CWM は着信イベントをリッスンする HTTP エンドポイントを公開します。特定のタイプのイベントが発生すると、このイベントタイプを待機する実行中のワークフローに転送されます。

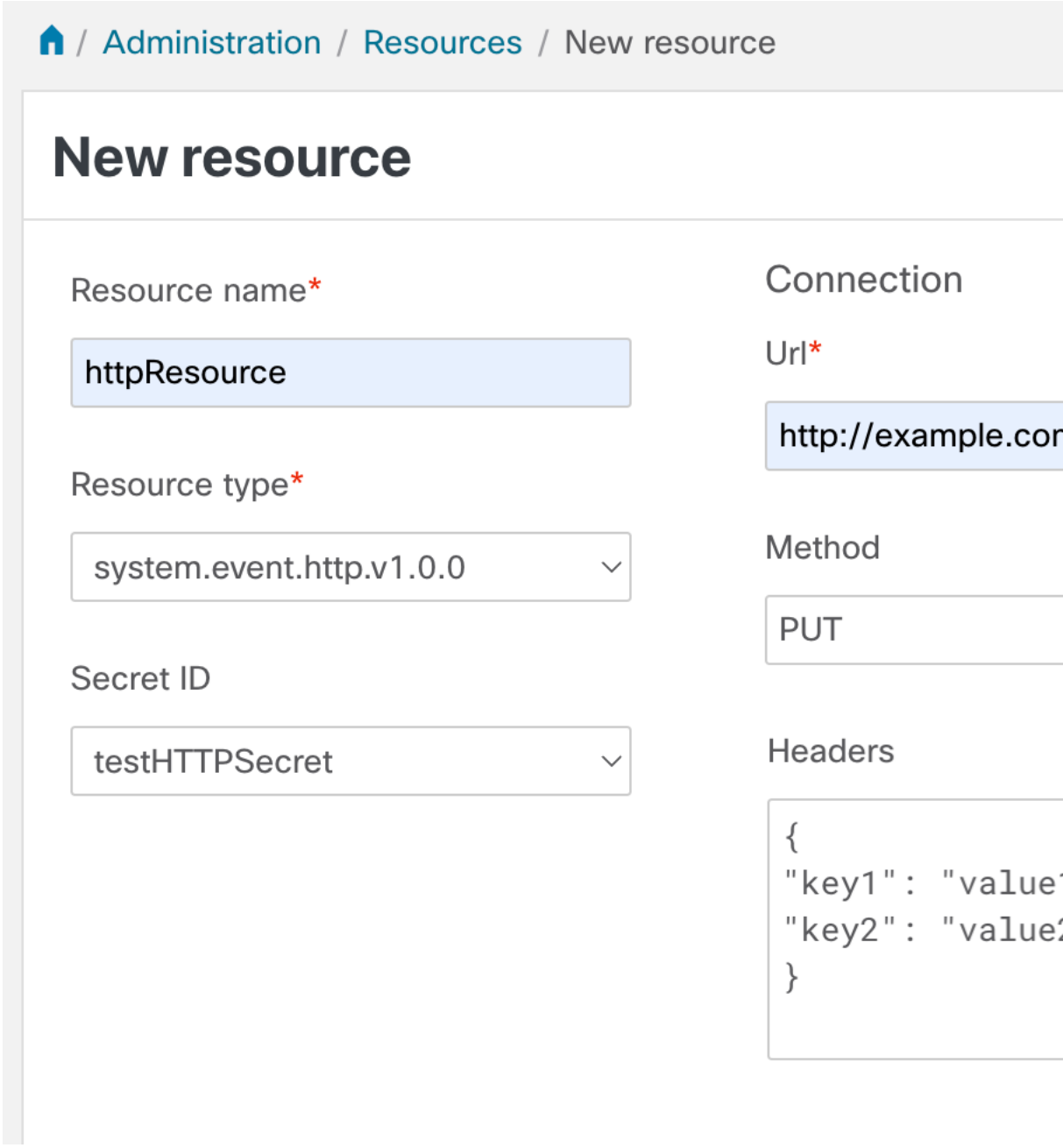
（注） イベントが消費されると、CWM は接続先 HTTP サーバーとして機能します。したがって、CWM サーバーの URL は、特定の HTTP イベントタイプのリソースとして効果的に指定されます。

イベントメッセージは HTTP *POST* リクエストである必要があり、メッセージ本文はクラウドイベントを表す JSON フォーマットである必要があります。

（注） 例： `{ "specversion": "1.0", "id": "2763482-4-324-32-4", "type": "com.github.pull_request.opened", "source": "/sensors/tn-1234567/alerts", "datacontenttype": "text/xml", "data": "<test=\"xml\"/>", "contextAttrName": "contextAttrValue" }`

produceイベントの種類の場合、ワークフローはクラウドイベントフォーマットでイベントを生成し、CWM はそれを HTTP *POST* リクエストとして外部システムによって公開されている HTTP エンドポイントに転送します。HTTP エンドポイントアドレスは、CWM のリソース構成で定義されたホスト **URL** と、ワークフロー定義内のイベント定義の [エンドポイント（End point）] フィールドを連結したものです。リソース構成内で、リクエストメソッドを *PUT* また

はその他に変更し、キーと値のペアをヘッダー（JSON フォーマット）として追加できます。



イベントシステム構成

シークレット

イベント構成では、イベントを送信または受信するサードパーティサービスによって公開されているブローカまたはエンドポイントへの接続を有効にするために必要なログイン情報がシー

クレットに保存されます。これには、基本認証（ユーザー名とパスワード）が含まれます。シークレットの作成時に指定したシークレット ID は、リソースの作成時に参照されるため、事前にシークレットを追加する必要があります。その方法については、シークレットの追加に関するセクションを参照してください。

リソース

リソースは、サードパーティサービスによって公開されたイベントブローカまたはエンドポイントに到達するために必要なすべての接続の詳細（シークレットを含む）を提供する場所です。使用するブローカ/プロトコルに応じて、次の 3 つのデフォルトのイベントリソースタイプから選択できます。

- `system.event.amqp.v1.0.0`
- `system.event.kafka.v1.0.0`
- `system.event.http.v1.0.0`

それぞれに異なる構成フィールドのセットがあることに注意してください。

- AMQP の場合は、 `amqp://localhost:5723` のフォーマットで **ServerDSN** を指定します。
- Kafka の場合：
  - [KafkaVersion]：Kafka のバージョンを指定します。Kafka のバージョンを確認する標準的な方法は、端末で `bin/kafka-topics.sh --version` を実行することです。
  - [ブローカ（Brokers）]：`["localhost:9092", "192.168.10.9:9092"]` のフォーマットで Kafka ブローカアドレスを指定します。
  - [OtherSettings]：デフォルトの Kafka 設定値を含む編集可能なリスト。これらの値は必要に応じて変更できます。
- HTTP の場合：
  - **produce** イベントの種類： [URL] フィールドに入力し、必要に応じて [メソッド（Method）] と [ヘッダー（Headers）]（たとえば、JSON オブジェクトとしてのクライアント ID ヘッダー名と値）を入力します。

（注） **URL** は接続先 HTTP サーバーのアドレスである必要がありますが、URL パスは不要です。イベントタイプを設定するときに、URL パスを**エンドポイント**として指定します。

  - **consume** イベントの種類：[URL] フィールドに CWM インスタンスのサーバー URL を入力します（例：`192.168.10.9:9092`）。

（注） URL パス（/event/http）なしで CWM インスタンスの URL を指定することを忘れないでください。後でイベントタイプを設定するときに、URL パスを**エンドポイント**として使用します。

Event type

（注） 新しいイベントタイプを作成するには、リソースとシークレットを CWM に追加する必要があります。

イベントタイプを追加する場合は、次のフィールドを使用できます。

- [イベントタイプ名（Event type name）]：イベントタイプの名前。後でワークフロー定義内で参照されます。
- [リソース（Resource）]：以前に CWM に追加されたリソースのリスト。
  - [イベントソース（Event source）]：ワークフロー定義で参照される完全にユーザー定義のエントリ。produce イベントの種類の場合は必須です。
  - [エンドポイント（End point）]：Kafka トピック（イベントストリーム）、AMQP エンドポイント（終端）、または HTTP URL（ホスト）パスの名前。

（注） HTTP **consume** イベントの種類の場合は、**エンドポイント**として /event/http を指定します。

- [種類の選択（Select kind）]：consume または produce のイベントの種類の 2 つのオプションから構成されるリスト。

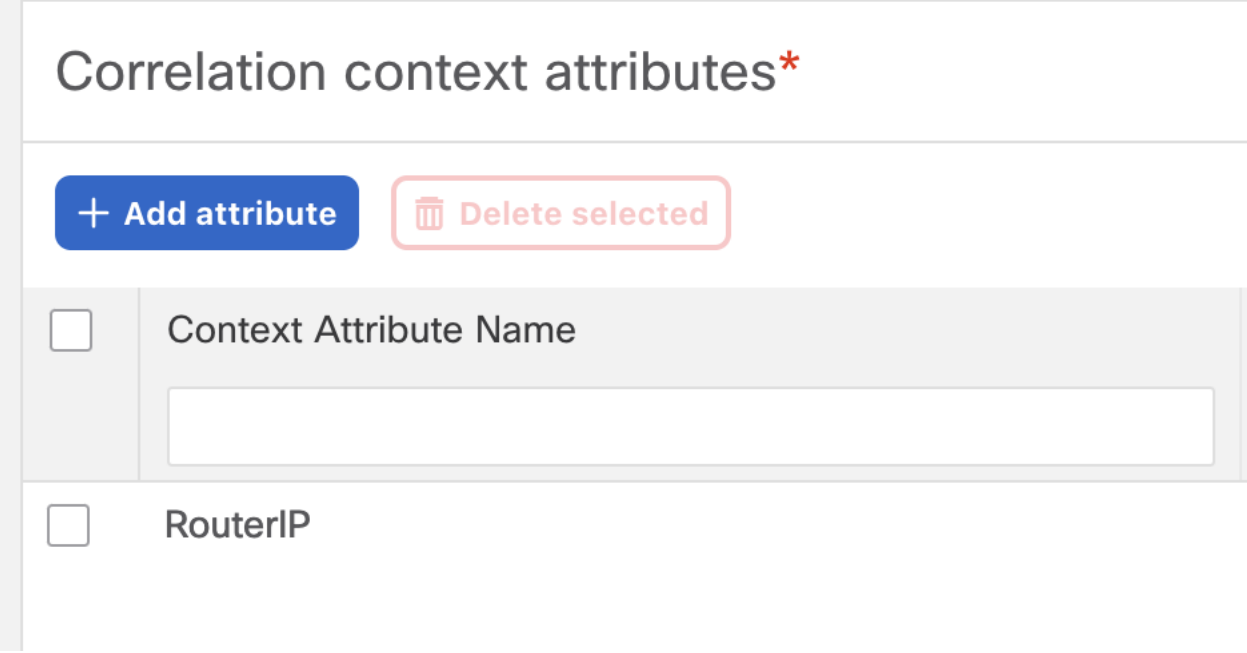
（注） both オプションは、{{ version.CWM }} ではまだサポートされていません。

- [リスナーの開始（Start listener）]（consume の種類の場合のみ）：クリックすると、定義されたイベントタイプのリスニングが開始されます。
- [ジョブの実行（Run job）]（consume の種類の場合のみ）：イベントの受信時にワークフローをトリガーする場合は、このチェックボックスをオンにします。その後、リストから目的のワークフローを選択します。

相関属性

必要に応じて、イベントのコンテキスト属性を設定できます。これらは consume イベントの種類にのみ適用され、ワークフローを選択的にトリガーするために使用されます。それらは、インバウンドイベントデータを絞り込み、相関属性の特定の値を持つイベントタイプをリッスンする適切なワークフローにルーティングする一種のカスタムフィルタとして表示できます。

イベントタイプに属性を追加するには、[属性の追加（Add attribute）] をクリックし、属性名と値を指定して、[追加（Add）] をクリックします。



（注） 相関属性は完全にユーザー定義です。特定のワークフローにルーティングされるクラウドイベントメッセージ内に記載されている JSON キーと値のペアと一致する必要があります。

イベントメッセージのフォーマット

イベントメッセージは、クラウドイベント仕様のフォーマットに従う必要があります。最小実行可能イベントメッセージには、次のパラメータが含まれます。

```
{
  "specversion": "1.0",
  "id": "00001",
  "type": "com.github.pull_request.opened",
  "source": "/sensors/tn-1234567/alerts"
}
```

メッセージには、"datacontenttype"、"data"、相関コンテキスト属性名（この例では contextAttrName）などの追加のパラメータを含めることができます。

```
  {
  "specversion": "1.0",
  "id": "2763482-4-324-32-4",
```

```
  "type": "com.github.pull_request.opened",
  "source": "/sensors/tn-1234567/alerts",
  "datacontenttype": "text/xml",
  "data": "<test data=\"xml\"/>",
  "contextAttrName": "contextAttrValue"
}
```

ワークフローイベントの定義とステート

ワークフロー定義には、ワークフローが待機するイベントを処理するために使用する2つの主要な構文要素があります。次のものがあります。

- イベント定義：イベントタイプとそのプロパティを定義するために使用されます。

```
{
   "name": "applicant-info",
   "type": "org.application.info",
   "source": "applicationssource",
   "correlation": [
    {
      "contextAttrName": "applicantId"
    }
   ]
}
```

- イベントステート：イベントが発生したときに実行するアクションを定義するために使用されます。

```
{
"name": "MonitorVitals",
"type": "event",
"onEvents": [
        {
          "actions": [
            {
              "functionRef": {
                "refName": "uppercase",
                "arguments": {
                  "input": {
                    "in": "patient ${ .patient } has high temperature"
                  }
                }
              }
            }
          ],
          "eventRefs": [
            "HighBodyTemperature"
          ]
        }
      ]
}
```

Kafka イベントの定義

前提条件

- セットアップされた Kafka サービス（または AMQP の場合は AMQP 1.0 プラグインを使用した RabbitMQ、または任意の HTTP クライアント）。
- CWM 1.1 が OVA を使用してインストールされている。

ステップ 1：Kafka シークレットとリソースの作成

Kafka サービスへのセキュアな接続を有効にするには、Kafka ログイン情報を使用してシークレットを作成し、接続の詳細を含むリソースを作成する必要があります。作成方法は次のとおりです。

シークレットの作成

ステップ 1 CWM で、[管理（Admin）] -> [シークレット（Secrets）] タブに移動します。

ステップ 2 [シークレットの追加（Add Secret）] をクリックします。

ステップ 3 [新しいシークレット（New secret）] ビューで、次を指定します。

a) [シークレット ID（Secret ID）]：`KafkaSecret`

b) [シークレットタイプ（Secret type）]：`basicAuth`

ステップ 4 シークレットタイプを選択すると、[シークレットタイプの詳細（Secret type details）] セクションに一連の追加フィールドが表示されます。フィールドに入力します。

a) [パスワード（password）]：Kafka へのログインに使用されるパスワード。

b) [ユーザー名（username）]：Kafka へのログインに使用されるユーザー名。

ステップ 5 [シークレットの作成（Create Secret）] をクリックします。

リソースの作成

ステップ 1 CWM で、[管理（Admin）] -> [リソース（Resources）] タブに移動します。

ステップ 2 [リソースの追加（Add Resource）] をクリックします。

ステップ 3 [新しいリソース（New resource）] ウィンドウで、次を指定します。

a) [リソース名（Resource name）]：`KafkaResource`

b) [リソースの種類（Resource name）]：`cisco.cwm.kafka.v1.0.0`（または、代わりにこれらのプロトコルを使用する場合は `cisco.cwm.amqp.v1.0.0` または `cisco.cwm.http.v1.0.0`）

c) [シークレット ID（Secret ID）]：`KafkaSecret`

d) 接続：

- [KafkaVersion]：Kafka のバージョンを指定します。これを確認する標準的な方法は、端末で `bin/kafka-topics.sh --version` を実行することです。
- [ブローカ（Brokers）]：`["localhost:9092"]` の形式で Kafka ブローカアドレスを指定します。
- [OtherSettings]：デフォルトの Kafka 設定値を含む編集可能なリスト。これらの値は必要に応じて変更できます。

（注） 接続設定は、**AMQP** と **HTTP** リソースタイプの場合は異なります。

- AMQP の場合は、**ServerDNS** を「amqp://localhost:5723」形式で指定します。- HTTP の場合は、**URL** と追加の **headers**（クライアント ID ヘッダー名と値など）を指定します。URL はホストアドレスである必要がありますが、URL パスは不要である点に注意してください。これは、リソースの種類の設定時に **End point** として指定します。

ステップ 4 [リソースの作成（Create a Resource）] をクリックします。

ステップ 2：CWM へのイベントタイプの追加

シークレットとリソースを用意したら、CWM によって消費または生成されるイベントのタイプを指定します。

ステップ 1 CWM UI で、左側のナビゲーションメニューから [管理（Admin）] タイルを選択します。

ステップ 2 [イベントシステム（Event system）] パネルで、[イベントタイプの追加（Add event type）] をクリックします。

ステップ 3 [新しいイベントタイプ（New event type）] モーダルで、必要な入力を行います。

a) [イベントタイプ名（Event type name）]：イベントタイプの名前を指定します。後でワークフロー定義内で参照します。

b) [リソース（Resource）]：リストから `KafkaResource` を選択します。

c) [イベントソース（Event source）]：イベントソースを定義します。これは完全にユーザー定義であり、ワークフロー定義で参照されます。`produce` イベントの種類の場合は必須です。

d) [エンドポイント（End point）]：Kafka の場合、Kafka トピック（イベントストリーム）を指定します。AMQP の場合は、エンドポイント（終端）を指定します。HTTP の場合は、URL（ホスト）パスを指定します。

e) [種類の選択（Select kind）]：リストから `consume` を選択します。

（注） ワークフローによって生成され、別のシステムによって消費されるイベントを定義するには、`Produce` を使用します。この場合、これ以降に示されている残りの **ステップ 2** の設定は適用されません。`both` オプションは、CWM 1.1 ではまだサポートされていません。

f) [リスナーの開始（Start listener）]：クリックすると、定義されたイベントタイプのリスニングが開始されます。

g) [ジョブの実行（Run job）]：イベントの受信時にワークフローをトリガーする場合は、このチェックボックスをオンにします。その後、リストから目的のワークフローを選択します。

h) [相関コンテキスト属性（Correlation context attributes）]：必要に応じて、イベントのコンテキスト属性を設定できます。これらは `consume` イベントの種類にのみ適用され、ワークフローを選択的にトリガーするために使用されます。それらは、インバウンドイベントデータを絞り込み、コンテキスト属性に基づいて「リスニング」ワークフロー内で定義されたアクションをトリガーする一種のカスタムフィルタとして表示できます。

i) [属性の追加（Add attribute）] をクリックし、属性名と値（完全にユーザー定義）を指定します。

ステップ 4 [イベントタイプの作成（Create Event type）] をクリックします。

ステップ 3：ワークフローでのイベントの定義

イベントタイプが追加されたので、このイベントタイプに登録して、イベントが CWM で受信されたときにアクションを実行するワークフローを作成できます。そのためには、イベント定義を使用してイベントを定義し、イベントステートを指定し、イベントが発生したときに実行

するアクションを定義する必要があります。例として、ルータ過熱アラーム（インバウンドイベント）がワークフローイベントステートをトリガーし、2つの修復アクションが実行されるシナリオを取り上げてみましょう。

```
{
  "id": "HighRouterTempWorkflow",
  "name": "Router Overheating Alarm Workflow",
  "start": "RemediateHighTemp",
  "events": [
    {
      "kind": "consumed",
      "name": "HighRouterTemp",
      "type": "HighRouterTemp",
      "source": "monitoring.app"
    }
  ],
  "states": [
    {
      "end": {
        "terminate": true
      },
      "name": "RemediateHighTemp",
      "type": "event",
      "onEvents": [
        {
          "actions": [
            {
              "functionRef": {
                "refName": "DispatchTech",
                "contextAttributes": {
                  "RouterIP": "${ .RouterIP }"
                },
                "resultEventTimeout": "PT30M"
              }
            },
          ],
          "eventRefs": ["HighRouterTemp"],
           },
           {
           "actions": [
            {
              "functionRef": {
                "refName": "MoveTraffic",
                "contextAttributes": {
                  "RouterIP": "${ .RouterIP }"
                },
                "resultEventTimeout": "PT30M"
              }
            }
          ],
          "timeouts": {
            "actionExecTimeout": "PT60M"
          }
        }
      ]
    }
  ],
  "version": "1.0.0",
  "description": "Remediate router overheating",
  "specVersion": "0.8"
}
```

（注） この例は完全なワークフローではないことに注意してください。ワークフロー内でイベントを定義し、それに基づいてアクションを実行する方法の例を示します。

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。