API

ここでは、次の内容について説明します。

CWM API の概要

CWM API は、Representational State Transfer（REST）の設計原則に従って開発されました。API には、JSON データ形式を使用した HTTP を使用してアクセスします。関連する HTTP 応答コードで、要求の処理の成否が示されます。データ取得メソッドには GET 要求が必要です。一方、データを追加、変更、または削除するメソッドには POST、PUT、PATCH、または DELETE メソッドが必要です。要求が誤ったリクエストタイプで送信されると、エラーが返されます。

CWM API の使用方法

CWM API は次の 2 つの方法で使用できます。

- Swagger インターフェイスを介して、または
- Postman コレクションを介して

CWM UI からアクセスする Swagger インターフェイスが製品に直接組み込まれていますが、使いやすさを考慮して、サンプル要求を含む Postman コレクションも提供されています。

[API を介した管理（Manage via API ）] セクションのすべてのチュートリアルでは、Swagger の使用を前提としています。

Swagger の使用

CWM Swagger API にアクセスするには、左側のナビゲーションメニューから、[swagger] アイコンをクリックします。

CWM Postman コレクションの使用

前提条件

- Postman Web アプリケーションアカウントまたは Postman デスクトップがインストールされている。

JSON コレクションファイルのダウンロード

このリンクをクリックして、JSON 形式の Postman コレクションをダウンロードします。zip アーカイブを解凍します。

コレクションのインポートと環境の設定

ステップ 1 Postman を開き、[コレクション（Collections）] に移動します。

ステップ 2 [インポート（Import）] をクリックし、[インポートするには任意の場所にドロップ（DropAnywhere to Import）] 画面からフォルダを選択し、zip アーカイブから解凍したフォルダをポイントします。

ステップ 3 [環境（Environment）] に移動し、新しくインポートした**テスト**環境を選択します。

ステップ 4 CWM の IP アドレスとポートに合わせて **baseUrl** 変数と **port** 変数の現在の値を指定し、変更を保存します。

これで設定が完了し、コレクションを使用する準備が整いました。

アダプタの管理

外部ターゲットシステムと連携するには、CWM にアダプタが必要です。アダプタは、**オペレータ**ガイドで説明されているように CWM UI で、または CWM API を使用して管理できます。アダプタの処理には、次の API エンドポイントを使用できます。

- `GET/adapter`：CWM アプリケーションに存在するアダプタのリストを取得します。
- `POST/adapter`：アダプタの **.tar** ファイルを CWM ストレージにアップロードします。
- `GET/adapter/{adapterId}`：CWM アプリケーションに存在する特定のアダプタの詳細を取得します。これには、アダプタで使用可能なすべてのアクティビティの一覧表示が含まれます。

- `PUT/adapter/{adapterId}`：既存のアダプタファイルを新しいアダプタバージョンで更新します。
- `DELETE/adapter/{adapterId}`：CWM アプリケーションからアダプタを削除します。
- `POST/adapter/{adapterId}/deploy`：アップロードされたアダプタファイルに基づいてシステムにアダプタを展開します。
- `PATCH/adapter/{adapterId}`：アダプタのデフォルトステータスを更新します。

アダプタのインストール

CWM アダプタは、**.tar** インストールファイルに含まれています。ワークフローで使用するには、事前にストレージにアップロードしてシステムに展開する必要があります。その方法について説明します。

アダプタファイルのアップロード

アダプタを展開する前に、アダプタの **.tar** ファイルを CWM ストレージにアップロードする必要があります。

ステップ 1 最新のアダプタ インストール ファイルを取得するか、独自のアダプタを作成します。

ステップ 2 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 3 [adapters] セクションで、`POST/adapter` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] をクリックします。

ステップ 4 表示されるサブセクションで、[ファイルの選択（Choose File）] をクリックし、アダプタの **.tar** インストールファイルを選択して [アップロード（Upload）] をクリックし、[実行（Execute）] をクリックします。

サーバーの応答コードが `201` の場合、アダプタファイルは CWM データベースに正常にアップロードされています。

アダプタの展開

ステップ 1 CWM API の [アダプタ（adapters）] セクションで、`GET/adapter` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] と[実行（Execute）] をクリックします。

ステップ 2 サーバーの応答本文から、アップロードしたアダプタの `id` フィールドの値をコピーします。

ステップ 3 CWM API の [アダプタ（adapters）] セクションで、`POST/adapter/{adapterId}/deploy` エンドポイントをクリックして展開します。

ステップ 4 エンドポイント内で、[試す（Try it out）] をクリックします。[アダプタID（Adapter ID）] フィールドにアダプタ ID を貼り付けます。

ステップ 5 [createWorker] フィールドで、`createWorker` パラメータを `true` に設定できます。アダプタ ID と同じ名前のワーカーが作成されます。

ステップ 6 [実行（Execute）] をクリックします。

サーバーの応答コードが `201` の場合、アダプタプラグインは正常にインストールされており、問題なく続行できます。

アダプタの削除

アダプタをストレージから完全に削除してアンインストールするには、次の手順を実行します。

ステップ 1 CWM API の [アダプタ（adapters）] セクションで、`GET/adapter` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] と[実行（Execute）] をクリックします。

ステップ 2 サーバーの応答本文から、アップロードしたアダプタの `id` フィールドの値をコピーします。

ステップ 3 CWM API の [アダプタ（adapters）] セクションで、`DELETE/adapter/{adapterId}` エンドポイントをクリックして展開します。

ステップ 4 エンドポイント内で、[試す（Try it out）] をクリックします。[アダプタID（Adapter ID）] フィールドにアダプタ ID を貼り付けます。

ステップ 5 [実行（Execute）] をクリックします。

ワーカーの管理

ワーカーは、ワークフロー定義とアダプタコードで定義されたアクションを実行するプロセスです。**オペレータ**ガイドで説明されているように CWM UI を使用してワーカーを管理できます。または以下で説明しているように CWM API を使用して管理できます。

ワーカーを管理するための次のアクションを使用できます。

- `GET/worker`：CWM アプリケーションに存在するワーカーのリストを取得します。
- `POST/worker`：CWM アプリケーション内に新しいワーカーを作成します。
- `GET/worker/{workerName}`：CWM アプリケーションに存在する特定のワーカーの詳細を取得します。
- `PUT/worker/{workerName}`：既存のワーカーを新しいパラメータ値で更新します。
- `DELETE/worker/{workerName}`：CWM アプリケーションからワーカーを削除します。
- `POST/worker/{workerName}/start`：アプリケーションで作成されたワーカーをアクティブにします。
- `POST/worker/{workerName}/stop`：アプリケーションで作成されたワーカーを非アクティブ化します。

ワーカーの作成

ステップ 1 CWM にログインし、左側のナビゲーションメニューから [Swagger] アイコンをクリックします。

ステップ 2 CWM API の [ワーカー（workers）] セクションで、`POST/worker` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] をクリックします。

ステップ 3 [ワーカーデータ（Worker data）] フィールドに、必要な値を入力します。

a) [activities] : 展開されたアダプタの ID または特定のアダプタアクティビティを貼り付けます。

b) [startWorker] : `true` に設定します。

c) [workerName] : ワーカーの名前を指定します。

ステップ 4 [実行（Execute）] をクリックします。

ワーカーの開始

ステップ 1 CWM API の [ワーカー（workers）] セクションで、`POST/{workerName}/start` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] をクリックします。

ステップ 2 次の**パラメータ**のフィールドに、必要な値を入力します。

a) [開始するワーカーの名前（Name of a worker to start）] : 開始するワーカーの名前を貼り付けます。

b) [forceReload]: ワーカーを強制的に起動する場合は `true` に設定します。

ステップ 3 [実行（Execute）] をクリックします。

ワーカーの停止

ステップ 1 CWM API の [ワーカー（workers）] セクションで、`POST/{workerName}/stop` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] をクリックします。

ステップ 2 次の**パラメータ**のフィールドに、必要な値を入力します。

a) [停止するワーカーの名前（Name of a worker to stop）] : 停止するワーカーの名前を貼り付けます。

b) [forceStop]: ワーカーを強制的に停止する場合は `true` に設定します。

ステップ 3 [実行（Execute）] をクリックします。

ワークフローの管理

ワークフロー定義は、**オペレータ**ガイドで説明されているように CWM UI で、または CWM API を使用して管理できます。

- `GET/workflow`：CWM アプリケーションに存在するワークフロー定義のリストを取得します。
- `POST/workflow`：CWM アプリケーション内に新しいワークフロー定義を作成します。
- `GET/workflow/{workflowId}`：CWM アプリケーションに存在する特定のワークフローの詳細を取得します。
- `PUT/workflow/{workflowId}`：CWM アプリケーション内に存在するワークフロー定義を更新します。
- `DELETE/workflow/{workflowId}`：選択したワークフロー定義を CWM アプリケーションから削除します。
- `GET/workflowExport`：指定されたワークフロー定義 ID の配列に基づいてワークフロー定義をエクスポートします。
- `POST/workflowImport`：CWM アプリケーションにワークフロー定義を一括でインポートします。

（注） ワークフローを管理するには、CWM UI を使用することが推奨されます。詳細については、**オペレータ**ガイドを参照してください。

リソースとシークレットの管理

概要

CWM では、アダプタは、他のシステムやアプリケーションなどの外部エンティティでアクションを実行できるようにするアクティビティを定義します。これらのエンティティは、ほとんどの場合、通常は接続と認証データを必要とする API を介して統合されます。CWM は、アクティビティがワークフローで使用されるときに、接続エンドポイントの詳細と認証データを実行時に渡すことができるフレームワークを提供します。したがって、ワークフローを実行するオペレータは、IP アドレス、ポート、ユーザー名、パスワードなど、これらのシステム（リソース）の詳細を知らない場合があります。

CWM は、データベース内のリソースとシークレットを安全に処理し、それぞれの ID でそれらを識別するためのフレームワークを提供します。ワークフローを実行する場合は、リソース ID のみを渡す必要があり、残りのデータはリソースマネージャによってアダプタに送信され

ます。オペレータの介入やアダプタ開発者の追加開発は必要ありません。シークレットとリソースは、**オペレータ**ガイドで説明されているようにCWM UIで、またはCWM APIを使用して管理できます。

リソースおよびシークレットのタイプ

リソースおよびシークレットのタイプは、ユーザーが作成したリソースとシークレットをタイプ別に整理するために使用される入れ物と考えることができます。タイプは特定のアダプタ内で定義され、アダプタのインストール時に自動的にシステムに追加されます。
`GET/secretType/{secretTypeId}` APIエンドポイントを使用して、特定のタイプに属するシークレットを一覧表示できます。

シークレット API エンドポイント

シークレットを管理するための次のアクションを使用できます。

- `GET/secret`：CWMアプリケーションに存在するシークレットのリストを取得します。
- `POST/secret`：CWMアプリケーション内に新しいシークレットを作成します。
- `GET/secretType/{secretTypeId}`：CWMアプリケーションに存在する、特定のタイプに属するシークレットを一覧表示します。
- `GET/secretType`：CWMアプリケーションに存在するシークレットのタイプのリストを取得します。
- `GET/secret/{secretId}`：既存のシークレットの詳細を取得します。
- `DELETE/secret/{secretId}`：CWMアプリケーションからシークレットを削除します。
- `PATCH/secret/{secretId}`：CWMアプリケーションに存在するシークレットを新しいパラメータ値で更新します。

リソース API エンドポイント

リソースを管理するための次のアクションを使用できます。

- `GET/resource`：CWMアプリケーションに存在するリソースのリストを取得します。
- `POST/resource`：CWMアプリケーションに新しいリソースを作成します。
- `GET/resource/{resourceId}`：CWMアプリケーションに存在する特定のリソースの詳細を取得します。
- `PATCH/resource/{resourceId}`：既存のリソースを新しいパラメータ値で更新します。
- `DELETE/resource/{resourceId}`：CWMアプリケーションからリソースを削除します。
- `GET/resourceType`：CWMアプリケーションに存在するリソースタイプのリストを取得します。
- `GET/resourceType/{resourceTypeId}`：既存のリソースの種類の詳細を取得します。

シークレットの作成

ステップ 1 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 2 CWM API の [シークレット（secrets）] セクションで、[POST /secret] エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で、[試す（Try it out）] をクリックし、[シークレット入力（Secret input）] フィールドにデータを入力します。入力の例を次に示します。

```
{
"secret": {
"username": "admin",
"password": "admin"
},
"secretId": "NSOSecret",
"secretType": "basicAuth"
}
```

ステップ 4 [実行（Execute）] をクリックします。

サーバーの応答コードが 201 の場合、シークレットは正常に作成されており、シークレットを関連付けるリソースの作成を開始できます。

リソースの作成

ステップ 1 [CWM APIリソース（CWM API resources）] セクションで、[POST /resource] エンドポイントをクリックして展開します。

ステップ 2 エンドポイント内で、[試す（Try it out）] をクリックし、[リソース入力（Resource input）] フィールドにデータを入力します。入力の例を次に示します。

```
{
    "resource": {
        "scheme": "http",
        "host": "127.0.0.1",
        "port": 8080
    },
    "resourceId": "NSOLocal",
    "resourceType": "cisco.nso.resource.v1.0.0",
    "secretId": "NSOSecret"
}
```

ステップ 3 [実行（Execute）] をクリックします。

サーバー応答コードが 201 の場合、リソースは正常に作成されています。

スケジュールの管理

繰り返し発生する操作を自動化したり、将来の特定の日時に確定したりするために、スケジュールを作成できます。CWM には、次の 2 種類のスケジュールがあります。

- ワンタイム：単一のジョブを実行する日時を定義します。
- 繰り返し：ジョブの実行が繰り返されるときのルール（間隔、カレンダー、または cron 式に基づいて）を定義します。

CWM スケジューラ API を使用すると、スケジュールを作成、更新、一時停止/一時停止を解除することができます。{{ version.CWM }} で基本スケジュールを（スケジュールの削除などの他の操作とともに）作成することは UI を介して可能ですが、スケジューラのより多くの機能を使用して高度なスケジュールを定義することは API を介してのみ可能です。

スケジュールの処理には、次の API エンドポイントを使用できます。

- `GET/schedule`：CWM アプリケーションに存在するスケジュールのリストを取得します。
- `POST/schedule`：CWM アプリケーション内に新しいスケジュールを作成します。
- `GET/schedule/{scheduleId}`：CWM アプリケーションに存在する特定のスケジュールの詳細を取得します。
- `PATCH/schedule/{scheduleId}`：既存のスケジュールを新しい詳細で更新します。
- `DELETE/schedule/{scheduleId}`：CWM アプリケーションからスケジュールを削除します。

スケジュールの作成

スケジュールを作成するには、次の手順を実行します。

ステップ 1 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 2 CWM API の [スケジューラ（scheduler）] セクションで、`POST/schedule` エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で、[試す（Try it out）] をクリックし、[スケジュールリクエスト（Schedule request）] で選択した値を指定します。

ステップ 4 [実行（Execute）] をクリックします。サーバー応答コードが `201` の場合、スケジュールは正常に作成されています。

スケジュールの更新

スケジュールを編集するには、次の手順を実行します。

ステップ 1 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 2 CWM API の [スケジューラ（scheduler）] セクションで、`PATCH/schedule` エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で、[試す（Try it out）] をクリックし、[スケジュール ID（Schedule ID）] を指定し、[スケジュール更新リクエスト（Schedule update request）] で選択した値を編集します。

ステップ 4 [実行（Execute）] をクリックします。サーバー応答コードが `200` の場合、スケジュールは正常に更新されています。

スケジュールを更新すると、構成全体が置き換えられます。つまり、既存の値を 1 つだけ変更する場合、それ以外は同じであっても、すべての詳細を再度渡す必要があります。

スケジュールの一時停止

メンテナンス期間など、選択した時間のスケジュールを一時停止できます。スケジュールが一時停止されると、実行されると想定されていた実行がスキップされます。スケジュールを一時停止するには、次の手順を実行します。

ステップ 1 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 2 CWM API の [スケジューラ（scheduler）] セクションで、`PATCH/schedule` エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で、[試す（Try it out）] をクリックし、[スケジュール ID（Schedule ID）] を指定し、[スケジュール更新リクエスト（Schedule update request）] で [一時停止（paused）] フィールドの値を `true` に設定します。

ステップ 4 [実行（Execute）] をクリックします。サーバー応答コードが `200` の場合、スケジュールは正常に一時停止されています。

次のリクエストで再開するまでの間、スケジュールは一時停止されたままになります。

スケジュールの一時停止の解除

スケジュールを再開するには、次の手順を実行します。

ステップ 1 CWM にログインし、左側のナビゲーションメニューから [swagger] アイコンをクリックします。

ステップ 2 CWM API の [スケジューラ（scheduler）] セクションで、`PATCH/schedule` エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で、[試す（Try it out）] をクリックし、[スケジュール ID（Schedule ID）] を指定し、[スケジュール更新リクエスト（Schedule update request）] で [一時停止（paused）] フィールドの値を `false` に設定します。

ステップ 4 [実行（Execute）] をクリックします。サーバー応答コードが `200` の場合、スケジュールは正常に再開されています。

失敗時の一時停止

`pauseOnFailure` フィールドを `true` に設定すると、ジョブのいずれかが失敗した後にスケジュールが自動的に一時停止されます。たとえば、スケジュールに関連付けられたワークフロー定義が削除される場合など、問題に対処する機会が与えられます。失敗時の一時停止の値を変更するには 、「**スケジュールの更新**」の一般的な手順に従います。

オーバーラップポリシーの変更

オーバーラップポリシーは、前の実行が引き続き実行されているときに、同時にスケジュールによって次のジョブを開始する必要がある場合の動作を制御します。デフォルトのポリシーは [スキップ（Skip）] です（[オーバーラップ（overlap）] フィールドの値は `1` に設定される）。つまり、前のジョブがまだ実行中の場合、次にスケジュールされた実行は開始されず、スキップされます。既存の実行が完了すると、その時間より後の次にスケジュールされた実行のみが考慮されます。オーバーラップポリシーを変更するには、「**スケジュールの更新**」の一般的な手順に従います。

使用可能なポリシー：

ポリシータイプ	[オーバーラップ（**overlap**）] フィールド値	説明
スキップ（**Skip**）	1	これは、実行の重複を防ぐデフォルトのポリシーです。スケジュールの前の実行がまだ実行されているときに、次の実行を実行する必要がある場合、次の実行はスキップされます。
1 つをバッファ（Buffer One）	2	現在の実行が完了するとすぐに次の実行を開始します。1 つの実行のみがバッファリングされます。現在のジョブの実行中に発生することが想定されている複数の実行である場合、それらはスキップされ、実行中のジョブの終了後にキュー内の最初の実行のみが実行されます。

ポリシータイプ	[オーバーラップ（**overlap**）]フィールド値	説明
すべてをバッファ（**Buffer All**）	3	現在のジョブの実行中に発生することが想定されているすべての実行をバッファリングします。バッファリングされた実行はすべて、実行中のジョブが完了した直後に順番に実行されます。
その他をキャンセル（**Cancel Other**）	4	実行中のジョブをキャンセルし、古いジョブのキャンセルが完了した後に新しいジョブを開始します。
その他を終了（**Terminate Other**）	5	実行中のジョブを終了し、即座に新しいジョブを開始します。
すべて許可（**Allow All**）	[6]	任意の数の同時実行を開始します。

ジョブの管理

ジョブは、オペレータガイドで説明されているようにCWM UIで、またはCWM APIを使用して管理できます。

!!!「重要」複数のタグに基づくジョブのクエリなど、{{version.CWM}}の一部の機能は、APIを介してのみ使用できます。

ジョブを管理するための次のアクションを使用できます。

- `GET/job`：CWMアプリケーションに存在するジョブのリストを取得します。
- `POST/job`：指定されたパラメータに基づいてCWMアプリケーションで実行される新しいジョブを作成します。
- `GET/job/{jobId}/runs/{runId}`：CWMアプリケーションに存在する特定のジョブの詳細を返します。
- `POST/job/{jobId}/runs/{runId}/cancel`：ワークフローワーカーがワークフロー定義から進行中のタスクの実行を完了した後、実行中のジョブの実行をキャンセルします。
- `GET/job/{jobId}/runs/{runId}/events`：特定のジョブのイベント履歴を返します。
- `POST/job/{jobId}/runs/{runId}/terminate`：実行中のジョブの実行をただちに終了します。

ジョブの実行

ジョブを実行するには、次の手順を実行します。

ステップ 1 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 2 CWM API の [ジョブ（jobs）] セクションで、`POST/job` エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で [試す（Try it out）] をクリックし、[ジョブ実行リクエスト（Job Execution Request）] で選択した値を指定します。次に例を示します。

```
{
"data": {},
"jobName": "test API job",
"tags": [
    "test", "API"
],
"workflowName": "Test cisco workflow",
"workflowVersion": "1.1"
}
```

（注） ジョブタグはオプションですが、将来的に特定のジョブのフィルタリングが容易になる場合があります。

ステップ 4 [実行（Execute）] をクリックします。

ジョブ実行が正常に作成された場合は、コード `200` とジョブ ID と実行 ID が返されるサーバー応答を取得する必要があります。

複数タグによるジョブのフィルタ処理

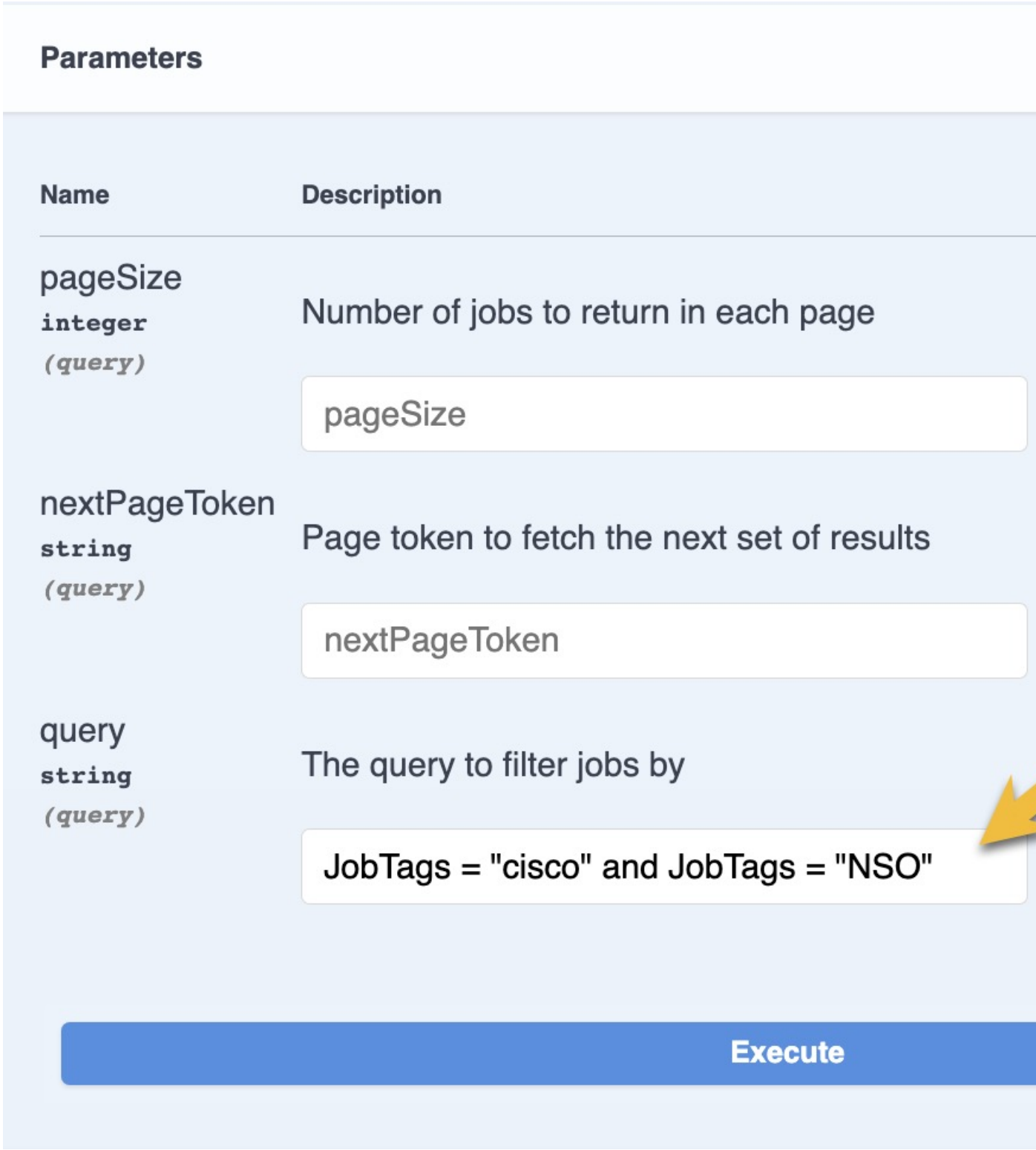
関連付けられたタグなど、特定のクエリでフィルタ処理されたCWMに存在するジョブのリストを取得できます。リストを取得するには、次の手順を実行します。

ステップ 1 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 2 CWM API の [ジョブ（jobs）] セクションで、`GET/job/{jobId}/runs/{runId}` エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で [試す（Try it out）] をクリックし、次の例のように、`query` パラメータで `JobTags = "tag_name1" and JobTags = "tag_name2"`というスキーマに従って、ジョブをフィルタ処理するタグを指定します。

図 *1:* ジョブイベントログ



ステップ 4 [実行（Execute）] をクリックします。

フィルタ処理されたジョブの詳細とともに、ステータスコード 200 のサーバー応答が表示されます。

既知の問題：ジョブの詳細を含む応答本文で、workflowId という名前のフィールドは、実際にはワークフロー定義 ID ではなくジョブ ID です。

イベント履歴の取得

特定のジョブのイベント履歴（すべてまたはフィルタ処理済み）のリストを取得するには、次の手順を実行します。

ステップ 1 CWM にログインし、左側のナビゲーションメニューから [swagger] アイコンをクリックします。

ステップ 2 CWM API の [ジョブ（jobs）] セクションで、 GET/job/{jobId}/runs/{runId}/events エンドポイントをクリックして展開します。

ステップ 3 エンドポイント内で、[試す（Try it out）] をクリックし、イベント履歴を取得するジョブの実行 ID とジョブ ID を指定します。

ステップ 4 必要に応じて、現在実行中のジョブをクエリする場合は、isLongPoll パラメータを true に設定できます。その後、特定のジョブの実行が終了するまで接続が開かれ、ジョブの実行が完了すると応答が返されます。false に設定すると、リクエストの送信後すぐに、リクエストの時点までに完了したイベントで構成されるイベント履歴が表示されます。

ステップ 5 必要に応じて、filterType パラメータを選択した値（all または close_event）に設定できます。

（注） close_event 値は、WorkflowExecutionFailed や WorkflowExecutionCompleted など、すでに終了したジョブのクローズイベントのみをフィルタ処理します。フィルタとして close_event を選択し、ジョブが現在実行中の場合は、400 エラーが表示されます。

ステップ 6 [実行（Execute）] をクリックします。

フィルタ処理されたイベントとともに、ステータスコード 200 のサーバー応答が表示されます。

イベントタイプの管理

イベントタイプは、CWM によって受信または生成され、ワークフロー定義で参照される信号のカテゴリです。**オペレータ**ガイドで説明されているように CWM UI を使用してワーカーを管理できます。または以下で説明しているように CWM API を使用して管理できます。

イベントタイプを管理するための次のアクションを使用できます。

- GET/eventType：CWM アプリケーションに存在するイベントタイプのリストを取得します。

- `POST/eventType`：CWM アプリケーションで新しいイベントタイプを作成します。
- `GET/eventType/{name}`：CWM アプリケーションに存在する特定のイベントタイプの詳細を取得します。
- `PUT/eventType/{name}`：既存のイベントタイプを新しいパラメータ値で更新します。
- `DELETE/eventType/{name}`：CWM アプリケーションからイベントタイプを削除します。
- `POST/eventType/{name}/start`：イベントリスナーを開始または停止します。

イベントタイプの作成

ステップ 1 CWM にログインし、左側のナビゲーションメニューから :simple-swagger: アイコンをクリックします。

ステップ 2 CWM API の [eventType] セクションで、`POST/eventType` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] をクリックします。

ステップ 3 [eventType データ（eventType data）] フィールドで、必要な値を変更します。

```
    {
"correlation": [
    {
    "contextAttributeName": "string",
    "contextAttributeValue": "string"
    }
],
"createWorkflow": false,
"dataOnly": true,
"endpoint": "string",
"kind": "string",
"name": "string",
"resourceId": "string",
"source": "string",
"type": "string",
"workflowName": "string",
"workflowVersion": "string"
}
```

ステップ 4 [実行（Execute）] をクリックします。

イベントタイプリスナーの開始/停止

ステップ 1 CWM API の [eventType] セクションで、`POST/{name}/{action}` エンドポイントをクリックして展開します。エンドポイント内で、[試す（Try it out）] をクリックします。

ステップ 2 次の**パラメータ**のフィールドに、必要な値を入力します。

a) "Name"：リスナーを開始/停止する必要があるイベントタイプの名前を貼り付けます。

b) "action"：リスナーを開始する場合は `start` に設定し、実行中のリスナーを停止する場合は `stop` に設定します。

ステップ 3 [実行（Execute）] をクリックします。

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。