Red Hat Streams for Apache Kafka 2.7

Streams for Apache Kafka Bridge での 3scale API Management の使用

3scale で利用可能な機能を使用する

Last Updated: 2024-05-29

Red Hat Streams for Apache Kafka 2.7 Streams for Apache Kafka Bridge での 3scale API Management の使用

3scale で利用可能な機能を使用する

法律上の通知

概要

Red Hat 3scale API Management をデプロイし、OpenShift Container Platform に AMQ Streams Kafka Bridge をデプロイして統合します。

目次

はじめに

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。

改善を提案するには、Jira 課題を作成し、変更案についてご説明ください。ご要望に迅速に対応できるよう、できるだけ詳細にご記入ください。

前提条件

- Red Hat カスタマーポータルのアカウントがある。このアカウントを使用すると、Red Hat Jira Software インスタンスにログインできます。
  アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. 以下の Create issue をクリックします。
2. **Summary** テキストボックスに、問題の簡単な説明を入力します。
3. **Description** テキストボックスに、次の情報を入力します。
   - 問題が見つかったページの URL
   - 問題の詳細情報
     他のフィールドの情報はデフォルト値のままにすることができます。
4. レポーター名を追加します。
5. **Create** をクリックして、Jira 課題をドキュメントチームに送信します。

フィードバックの提供にご協力いただきありがとうございました。

第1章 3SCALE API MANAGEMENT

Kafka Bridge を OpenShift Container Platform にデプロイした場合は、3scale で使用できます。

Kafka Bridge のプレーンデプロイメントでは、認証または認可のプロビジョニングがなく、TLS 暗号化による外部クライアントへの接続はサポートされません。3scale API Management は、TLS を使用して Kafka Bridge を保護し、認証および認可を提供できます。また、3scale との統合により、メトリック、流量制御、請求などの追加機能も利用できるようになります。

3scale では、Streams for Apache Kafka にアクセスする外部クライアントからのリクエストに対して、さまざまな種類の認証を使用できます。3scale では、以下のタイプの認証がサポートされます。

標準 API キー

識別子およびシークレットトークンとして機能する、ランダムな単一文字列またはハッシュ。

アプリケーション ID とキーのペア

イミュータブルな識別子およびミュータブルなシークレットキー文字列。

OpenID Connect

委譲された認証のプロトコル。

1.1. KAFKA BRIDGE のサービス検出

3scale は、サービス検出を使用して統合されます。これには、3scale を Streams for Apache Kafka および Kafka Bridge と同じ OpenShift クラスターにデプロイする必要があります。

Streams for Apache Kafka Cluster Operator デプロイメントには、以下の環境変数が設定されている必要があります。

- STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_LABELS
- STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_ANNOTATIONS

Kafka Bridge をデプロイすると、Kafka Bridge の REST インターフェイスを公開するサービスは、3scale による検出にアノテーションとラベルを使用します。

- 3scale によって **discovery.3scale.net=true** ラベルが使用され、サービスが検出されます。
- アノテーションによってサービスに関する情報が提供されます。

OpenShift コンソールで設定を確認するには、Kafka Bridge インスタンスの **Services** に移動します。**Annotations** に、Kafka Bridge の OpenAPI 仕様へのエンドポイントが表示されます。

1.2. 3SCALE APICAST ゲートウェイポリシー

3scale は 3scale APIcast と組み合わせて使用されます。これは、3scale と共にデプロイされた API ゲートウェイであり、Kafka Bridge への単一のエントリーポイントを提供します。

APIcast ポリシーは、ゲートウェイの動作をカスタマイズするメカニズムを提供します。3scale には、ゲートウェイ設定のための標準ポリシーのセットが含まれています。また、独自のポリシーを作成することもできます。

APIcast ポリシーに関する詳細は、 Red Hat 3scale のドキュメント を参照してください。

Kafka Bridge の APIcast ポリシー

3scale と Kafka Bridge との統合のポリシー設定例は **policies_config.json** ファイルに含まれており、このファイルでは以下を定義します。

- Anonymous Access (匿名アクセス)
- Header Modification (ヘッダー変更)
- Routing (ルーティング)
- URL Rewriting (URL の書き換え)

ゲートウェイポリシーは、このファイルを使用して有効または無効に設定します。

この例をひな形として使用し、独自のポリシーを定義できます。

Anonymous Access (匿名アクセス)

Anonymous Access ポリシーでは、認証をせずにサービスが公開され、HTTP クライアントがデフォルトのクレデンシャル (匿名アクセス用) を提供しない場合に、このポリシーによって提供されます。このポリシーは必須ではなく、認証が常に必要であれば無効または削除できます。

Header Modification (ヘッダー変更)

Header Modification ポリシーを使用すると、既存の HTTP ヘッダーを変更したり、ゲートウェイを通過するリクエストまたはレスポンスへ新規ヘッダーを追加したりすることができます。3scale の統合では、このポリシーによって、HTTP クライアントから Kafka Bridge までゲートウェイを通過するすべてのリクエストにヘッダーが追加されます。
Kafka Bridge は、新規コンシューマー作成のリクエストを受信すると、URI のある **base_uri** フィールドが含まれる JSON ペイロードを返します。コンシューマーは後続のすべてのリクエストにこの URI を使用する必要があります。以下に例を示します。

```
{
  "instance_id": "consumer-1",
  "base_uri":"http://my-bridge:8080/consumers/my-group/instances/consumer1"
}
```

APIcast を使用する場合、クライアントは以降のリクエストをすべてゲートウェイに送信し、Kafka Bridge には直接送信しません。そのため URI には、ゲートウェイの背後にある Kafka Bridge のアドレスではなく、ゲートウェイのホスト名が必要です。

Header Modification ポリシーを使用すると、ヘッダーが HTTP クライアントからリクエストに追加されるので、Kafka Bridge はゲートウェイホスト名を使用します。

たとえば、**Forwarded: host=my-gateway:80;proto=http** ヘッダーを適用すると、Kafka Bridge は以下をコンシューマーに提供します。

```
{
   "instance_id": "consumer-1",
   "base_uri":"http://my-gateway:80/consumers/my-group/instances/consumer1"
}
```

X-Forwarded-Path ヘッダーには、クライアントからゲートウェイへのリクエストに含まれる元のパスが含まれています。このヘッダーは、ゲートウェイが複数の Kafka Bridge インスタンスをサポートする場合に適用される Routing ポリシーに密接に関連します。

Routing (ルーティング)

Routing ポリシーは、複数の Kafka Bridge インスタンスがある場合に適用されます。コンシュー

マーが最初に作成された Kafka Bridge インスタンスにリクエストを送信する必要があるため、適切な Kafka Bridge インスタンスにリクエストを転送するようゲートウェイのルートをリクエストに指定する必要があります。
Routing ポリシーは各ブリッジインスタンスに名前を付け、ルーティングはその名前を使用して実行されます。Kafka Bridge のデプロイ時に、**KafkaBridge** カスタムリソースで名前を指定します。

たとえば、コンシューマーから以下への各リクエスト (**X-Forwarded-Path** を使用) について考えてみましょう。

http://my-gateway:80/my-bridge-1/consumers/my-group/instances/consumer1

この場合、各リクエストは以下に転送されます。

http://my-bridge-1-bridge-service:8080/consumers/my-group/instances/consumer1

URL Rewriting ポリシーはブリッジ名を削除しますが、これは、リクエストをゲートウェイから Kafka Bridge に転送するときにこのポリシーが使用されないからです。

URL Rewriting (URL の書き換え)

URL Rewiring ポリシーは、ゲートウェイから Kafka Bridge にリクエストが転送されるとき、クライアントから特定の Kafka Bridge インスタンスへのリクエストにブリッジ名が含まれないようにします。
ブリッジ名は、ブリッジが公開するエンドポイントで使用されません。

1.3. TLS 検証用の 3SCALE APICAST

TLS の検証用に APIcast を設定できます。これにはテンプレートを使用した APIcast の自己管理によるデプロイメントが必要になります。**apicast** サービスがルートとして公開されます。

TLS ポリシーを Kafka Bridge API に適用することもできます。

1.4. 既存の 3SCALE デプロイメントを使用する場合

3scale がすでに OpenShift にデプロイされており、Kafka Bridge と併用する場合は、Kafka Bridge を使用するための 3scale のデプロイメント に記載されているとおりに設定されていることを確認してください。

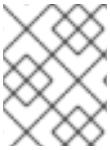
第2章 KAFKA BRIDGE を使用するための 3SCALE のデプロイメント

3scale を Kafka Bridge で使用するには、まず 3scale をデプロイし、次に Kafka Bridge API の検出を設定します。

このシナリオでは、Streams for Apache Kafka、Kafka、Kafka Bridge、および 3scale API Management コンポーネントは、同じ OpenShift クラスターで実行されます。

次の 3scale コンポーネントは、Kafka Bridge の検出に役立ちます。

- 3scale APIcast は、HTTP クライアントが Kafka Bridge API サービスに接続するための NGINX ベースの API ゲートウェイを提供します。
- 3scale toolbox は、Kafka Bridge サービスの OpenAPI 仕様を 3scale にインポートするために使用されます。



注記

3scale が Kafka Bridge と同じクラスターにすでにデプロイされている場合は、デプロイメント手順を省略して、現在のデプロイメントを使用できます。

前提条件

- デプロイメントに際して 3scale コンポーネントを理解している。
- Streams for Apache Kafka および Kafka が稼働している 。
- Kafka Bridge がデプロイされている 。

3scale デプロイメントの場合:

- Red Hat 3scale API Management Supported Configurations を確認する
- インストール用に **cluster-admin** ロール (**system:admin** など) を持つユーザーを用意する
- 以下が記述されている JSON ファイルにアクセスできる。
  - Kafka Bridge OpenAPI 仕様 (**openAPIV2.json**)
  - Kafka Bridge のヘッダー変更および Routing ポリシー (**policies_config.json**)
    GitHub で JSON ファイルを探します。

手順

1. Red Hat 3scale ドキュメント に記載の手順に従って、3scale API Management を設定します。
   a. 3scale API Management Operator を使用して 3scale API Manager と APIcast をインストールします。
   API Manager をデプロイする前に、**API Manager** カスタムリソースの **wildcardDomain** プロパティーを、OpenShift クラスターをホストするドメインに更新します。

   ドメインは、3scale 管理ポータルにアクセスするための URL (**http[s]://<authentication_token>@3scale-admin.<cluster_domain>**) で使用されます。

b. **API Manager** カスタムリソースのステータスを確認して、3scale が正常にデプロイされたことを確認します。

2. 3scale API Manager が Kafka Bridge サービスを検出するように承認を付与します。

```
oc adm policy add-cluster-role-to-user view system:serviceaccount:
<my_bridge_namespace>:amp
```

このコマンドは、指定した namespace (**<my_bridge_namespace>**) の Kafka Bridge リソースへの読み取りアクセス (**view**) を、API Manager (**amp**) に付与します。

3. 3scale API Management toolbox をセットアップします。

a. Red Hat 3scale のドキュメント に記載の手順に従って、3scale toolbox をインストールします。

b. 3scale と対話できるように環境変数を設定します。

Kafka Bridge の設定例

```
export REMOTE_NAME=strimzi-kafka-bridge (1)
export SYSTEM_NAME=strimzi_http_bridge_for_apache_kafka (2)
export TENANT=strimzi-kafka-bridge-admin (3)
export PORTAL_ENDPOINT=$TENANT.3scale.net (4)
export TOKEN=<3scale_authentication_token> (5)
```

(1) **REMOTE_NAME** は、3scale 管理ポータルのリモートアドレスに割り当てられた名前です。

(2) **SYSTEM_NAME** は、3scale toolbox で OpenAPI 仕様をインポートして作成される 3scale サービス/API の名前です。

(3) **TENANT** は、3scale 管理ポータルのテナント名です (**https://$TENANT.3scale.net**)。

(4) **PORTAL_ENDPOINT** は、3scale 管理ポータルを実行するエンドポイントです。

(5) **TOKEN** は、3scale toolbox または HTTP リクエストを介して対話するために 3scale 管理ポータルによって提供される認証トークンです。

c. 3scale toolbox のリモート Web アドレスを設定します。

```
podman run -v /path/to/openapiv2.json:/tmp/oas/openapiv2.json registry.redhat.io/3scale-
amp2/toolbox-rhel8:3scale2.14 3scale import openapi -d <admin_portal_url>
/tmp/oas/openapiv2.json
```

3scale toolbox のコンテナーイメージを指定します。3scale のコンテナーイメージは Red Hat Ecosystem Catalog にあります。

<admin_portal_url> は、3scale 管理ポータルのエンドポイントへのパス (**https://$TOKEN@$PORTAL_ENDPOINT/**) に置き換えます。これで、toolbox を実行するたびに、3scale 管理ポータルのエンドポイントアドレスを指定する必要がなくなりました。

4. Cluster Operator デプロイメントに、3scale が Kafka Bridge サービスを検出するために必要なラベルプロパティーおよびアノテーションプロパティーがあることを確認します。

```
#...
env:
- name: STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_LABELS
    value: |
    discovery.3scale.net=true
- name: STRIMZI_CUSTOM_KAFKA_BRIDGE_SERVICE_ANNOTATIONS
    value: |
    discovery.3scale.net/scheme=http
    discovery.3scale.net/port=8080
    discovery.3scale.net/path=/
    discovery.3scale.net/description-path=/openapi
#...
```

そうでない場合は、OpenShift コンソールを使用してプロパティーを追加するか、Cluster Operator と Kafka Bridge を 再デプロイし てみてください。

5. 3scale 管理ポータルから、Red Hat 3scale のドキュメント で説明するように、OpenShift から Kafka Bridge API サービスをインポートします。

6. OpenAPI 仕様 (JSON ファイル) の **Host** フィールドを編集して、Kafka Bridge サービスのベース URL を使用します。
以下に例を示します。

```
"host": "my-bridge-bridge-service.my-project.svc.cluster.local:8080"
```

host URL に以下が含まれていることを確認します。

- Kafka Bridge 名 (**my-bridge**)
- プロジェクト名 (**my-project**)
- Kafka Bridge のポート (**8080**)

7. 更新した OpenAPI 仕様をローカルファイルから 3scale toolbox にインポートします。

```
podman run -v /path/to/openapiv2.json:/tmp/oas/openapiv2.json registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale import openapi [opts] -d=<admin_portal_url> -t 3scale-kafka-bridge /tmp/oas/openapiv2.json
```

ここで、システム名は OpenAPI 仕様から生成するのではなく、**3scale-kafka-bridge** として指定します。**/path/to/openapiv2.json** を OpenAPI 仕様ファイルへのパスに置き換え、**<admin_portal_url>** を 3scale 管理ポータルのエンドポイントへのパスに置き換えます。

8. サービスの Header Modification および Routing ポリシー (JSON ファイル) をインポートします。

   a. 3scale で作成したサービスの ID を見つけます。これは、ポリシーをインポートするときに必要です。

```
export SERVICE_ID=$(curl -k -s -X GET "https://$PORTAL_ENDPOINT/admin/api/services.json?access_token=$TOKEN" | jq ".services[] | select(.service.system_name | contains(\"$SYSTEM_NAME\")) | .service.id")
```

ここでは、リクエストで **jq** コマンドライン JSON パーサーツールを 使用します。

b. ポリシーをインポートします。

```
3scale policies import -f /path/to/policies_config.json -d=<admin_portal_url> 3scale-kafka-bridge
```

/path/to/policies_config.json をポリシー設定ファイルへのパスに置き換え、**<admin_portal_url>** を 3scale 管理ポータルのエンドポイントへのパスに置き換えます。

9. 3scale 管理ポータルから、Kafka Bridge サービスのエンドポイントとポリシーがロードされていることを確認します。

10. 3scale Toolbox から、アプリケーションプランとアプリケーションを作成します。
認証のユーザーキーを取得するためにアプリケーションが必要になります。

11. 実稼働環境用の手順: 実稼働環境のゲートウェイで API を利用可能にするには、設定をプロモートします。

```
3scale proxy-config promote $REMOTE_NAME $SERVICE_ID
```

12. API テストツールを使用して、コンシューマーの作成に呼び出しを使用する APIcast ゲートウェイと、アプリケーションに作成されたユーザーキーで、Kafka Bridge にアクセスできることを検証します。
以下に例を示します。

```
https//my-project-my-bridge-bridge-service-3scale-apicast-staging.example.com:443/consumers/my-group?user_key=3dfc188650101010ecd7fdc56098ce95
```

Kafka Bridge からペイロードが返されれば、コンシューマーが正常に作成されています。

```
{
  "instance_id": "consumer1",
  "base uri": "https//my-project-my-bridge-bridge-service-3scale-apicast-staging.example.com:443/consumers/my-group/instances/consumer1"
}
```

ベース URI は、クライアントが以降のリクエストで使用するアドレスです。

付録A サブスクリプションの使用

Streams for Apache Kafka は、ソフトウェアサブスクリプションから提供されます。サブスクリプションを管理するには、Red Hat カスタマーポータルでアカウントにアクセスします。

アカウントへのアクセス

1. access.redhat.com に移動します。
2. アカウントがない場合は作成します。
3. アカウントにログインします。

サブスクリプションのアクティベート

1. access.redhat.com に移動します。
2. **My Subscriptions** に移動します。
3. **Activate a subscription** に移動し、16 桁のアクティベーション番号を入力します。

Zip および Tar ファイルのダウンロード

zip または tar ファイルにアクセスするには、カスタマーポータルを使用して、ダウンロードする関連ファイルを検索します。RPM パッケージを使用している場合、この手順は必要ありません。

1. ブラウザーを開き、access.redhat.com/downloads で Red Hat カスタマーポータルの **Product Downloads** ページにログインします。
2. **Streams for Apache Kafka** エントリーの場所を **INTEGRATION AND AUTOMATION** カテゴリーで特定します。
3. 必要な Streams for Apache Kafka 製品を選択します。**Software Downloads** ページが開きます。
4. コンポーネントの **Download** リンクをクリックします。

DNF を使用したパッケージのインストール

パッケージとすべてのパッケージ依存関係をインストールするには、以下を使用します。

```
dnf install <package_name>
```

ローカルディレクトリーからダウンロード済みのパッケージをインストールするには、以下を使用します。

```
dnf install <path_to_download_package>
```

改訂日時: 2024-04-30