Red Hat 3scale API Management 2.5

デベロッパーポータルの使用

デベロッパーポータルを適切に設定して API 管理機能を向上させる

Last Updated: 2023-03-26

Red Hat 3scale API Management 2.5 デベロッパーポータルの使用

デベロッパーポータルを適切に設定して API 管理機能を向上させる

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本ガイドでは、Red Hat 3scale API Management 2.5 でのデベロッパーポータルの使用について説明します。

目次

パート I. API ドキュメント

第1章 3SCALE への仕様の追加

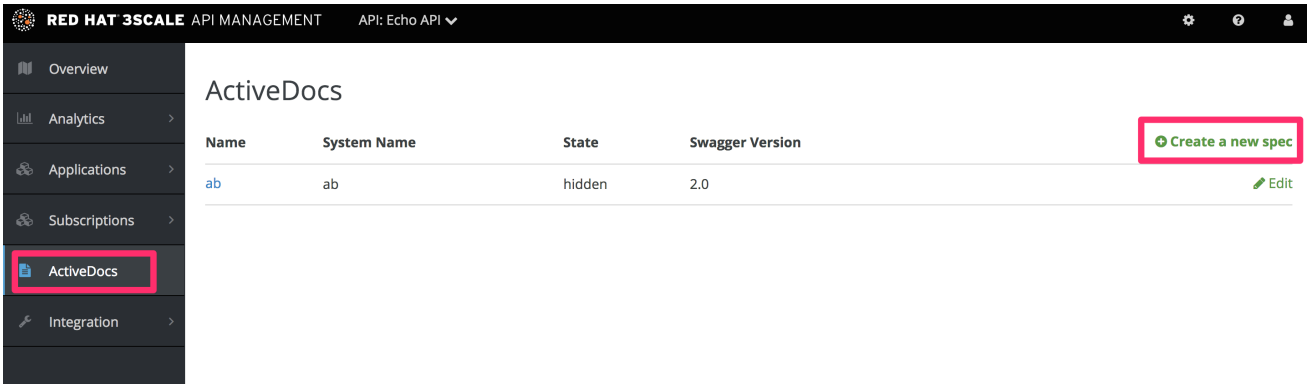
本章では、API に ActiveDocs を設定する手順について説明します。

3scale は、API のインタラクティブドキュメントを作成するためのフレームワークを提供します。

OpenAPI Specification (OAS) 2.0 (OpenAPI Specification (OAS) をベース) を使用することで、API に機能的なドキュメントを設定することができます。これは、開発者が API を調べ、テストを行い、統合するのに役立ちます。

1.1. ACTIVEDOCS のサービス仕様への移動

管理ポータルで **[your_API_name] → ActiveDocs** の順に移動します。これにより、API のサービス仕様のリストが表示されます (初期段階は空)。



サービス仕様は、必要なだけ追加することができます。通常、各サービス仕様は API のいずれか 1 つに対応しています。たとえば、3scale では、それぞれの 3scale API に仕様があります (Service Management、Account Management、Analytics、および Billing)。

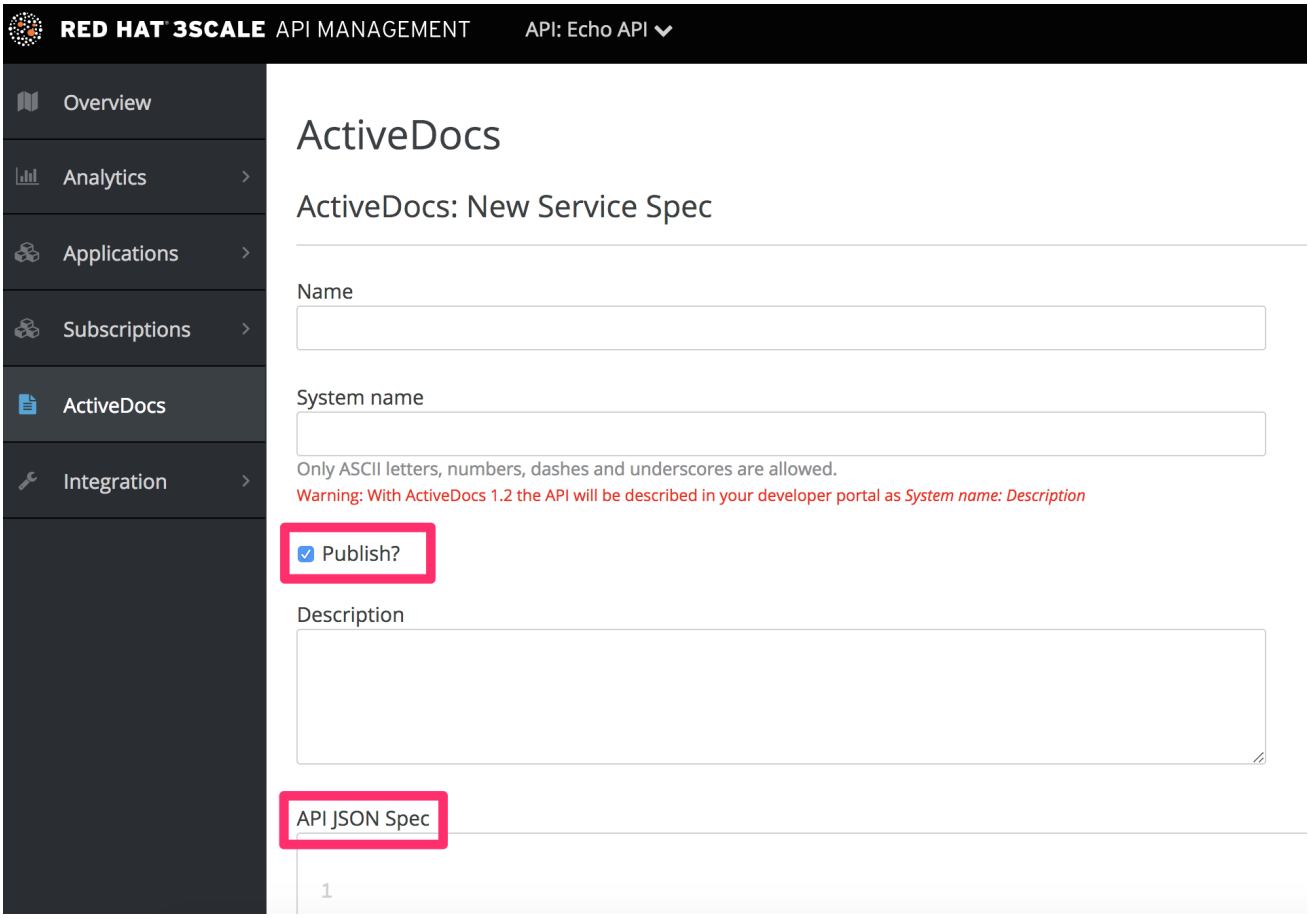
1.2. サービス仕様の作成

新しいサービス仕様を追加する場合は、以下の項目を指定する必要があります。

- 名前
- システム名 (デベロッパーポータルからサービス仕様を参照するために必要)
- 仕様を公開するかどうか
- 説明 (内部的にのみ使用される)
- API JSON 仕様 (以下の図を参照)

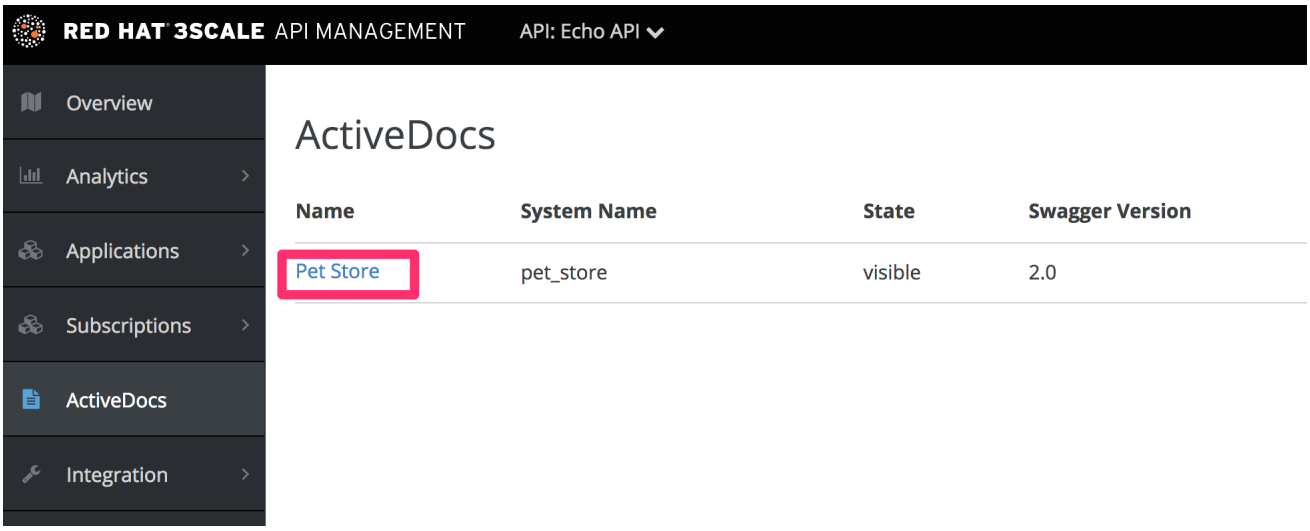
API JSON 仕様は、ActiveDocs の秘密の成分です。

API の仕様は、OpenAPI Specification (OAS) が提案する仕様に従って生成する必要があります。本チュートリアルは、API 用に有効な OpenAPI Specification (OAS) 2.0 準拠の仕様 がすでに用意されていることを前提としています。



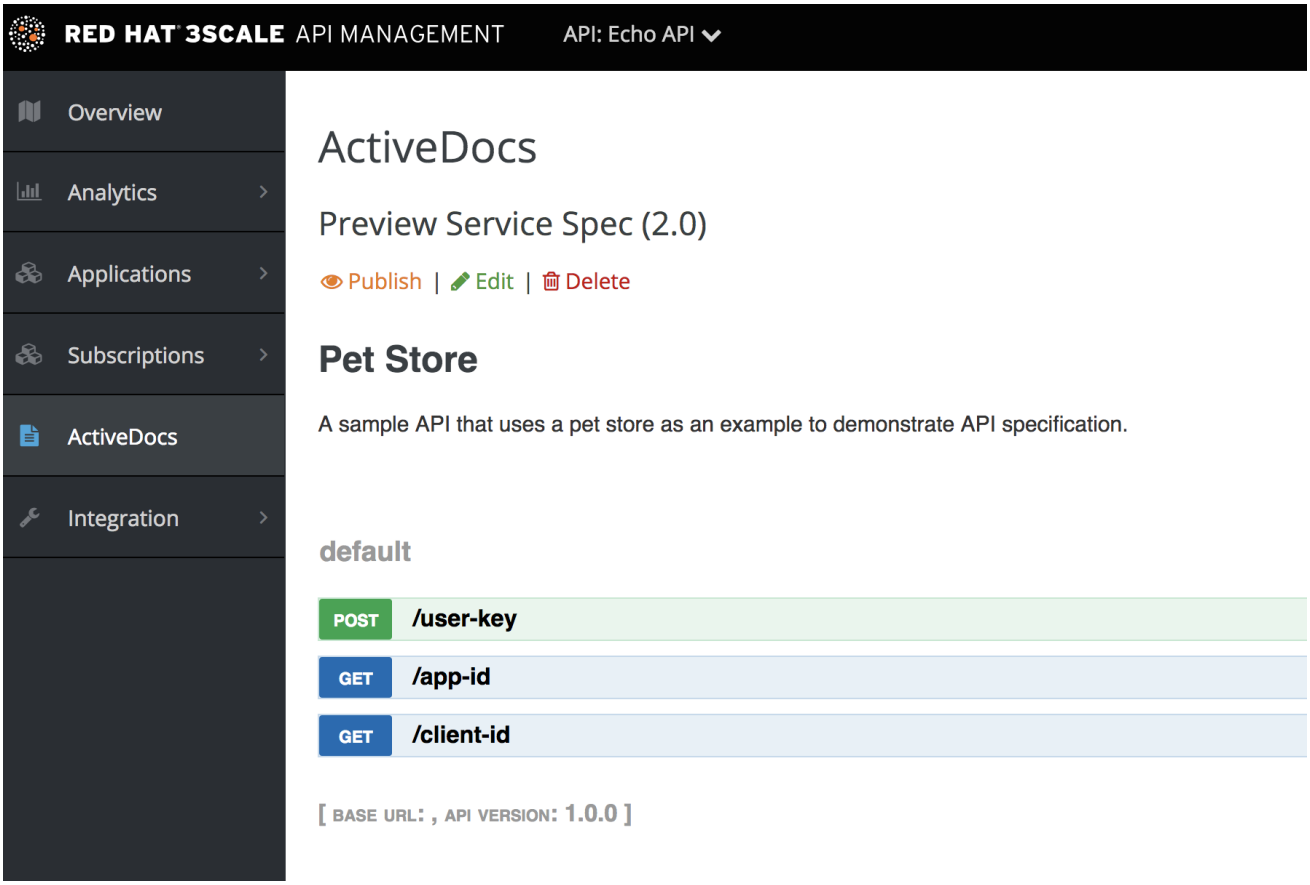
1.3. 最初の ACTIVEDOC に関する操作

最初の ActiveDoc を追加すると、これが **[your_API_name] → ActiveDocs** のリストに表示されます。必要に応じて編集、削除、または公開から非公開への切り替えが可能です。また、これを API から切り離したり、他の API にアタッチしたりできます。**Audience → Developer Portal → ActiveDocs**の順に移動して、(API にアタッチされているかどうかに関わらず) すべての ActiveDocs を確認できます。



また、サービス仕様に指定した名前 (この例では Pet Store) をクリックして、ActiveDocs がどう見えるかをプレビューできます。まだ仕様を公開していない場合でも、この操作を実行できます。

以下は、ActiveDoc の見え方の例です。


RED HAT 3SCALE API MANAGEMENT
API: Echo API
Overview
Analytics
Applications
Subscriptions
ActiveDocs
Integration
ActiveDocs
Preview Service Spec (2.0)
Publish | Edit | Delete
Pet Store
A sample API that uses a pet store as an example to demonstrate API specification.
default
POST /user-key
GET /app-id
GET /client-id
[BASE URL: , API VERSION: 1.0.0]

第2章 OAS 仕様の作成

本章は、RESTful API 用に OpenAPI Specification (OAS) 2.0 準拠の仕様を作成するのに役立ちます。この仕様は、デベロッパーポータルで ActiveDocs を動作させるのに必要です。コードを読むだけであれば、すべての例は OAS Petstore のソースコード例 に記載されています。

2.1. OPENAPI SPECIFICATION (OAS) について

3scale ActiveDocs は、Swagger と呼ばれる RESTful Web サービスの仕様をベースにしています (Wordnik より)。この例は、拡張された OpenAPi Specification の Petstore の例 をベースにしており、すべての仕様データを OpenAPi Specification Specification 2.0 のドキュメント から引用しています。

OAS は単なる仕様ではありません。これに関連するあらゆる機能のフレームワークも提供します。

1. 複数の言語 (NodeJS、Scala、その他) によるリソース仕様のサーバー
2. 仕様ファイルを使用して開発者を引き付ける UI を生成する、さまざまな HTML/CSS/Javascripts アセット
3. Swagger 準拠サーバーからクライアントライブラリーを自動的に生成することのできる、OAS codegen プロジェクト。数多くの現代的な言語によるクライアント側のライブラリー作成をサポートします。

2.2. 3SCALE ACTIVEDOCS と OAS

ActiveDocs は OAS に置き換わるものではなく、OAS をインスタンス化したものです。ActiveDocs を使用する場合、独自の OAS サーバーを実行したり、インタラクティブドキュメントの UI コンポーネントを扱ったりする必要がありません。インタラクティブドキュメントは、3scale のデベロッパーポータルから提供され、レンダリングされます。

Swagger 準拠の API 仕様をビルドし管理ポータルに追加する以外に、するべきことはありません。これで、インタラクティブドキュメントが利用可能になります。開発者は、デベロッパーポータルを通じて API に対するリクエストを行うことができます。

すでに Swagger 準拠の API 仕様がある場合は、それをデベロッパーポータルに追加できます (ActiveDocs の設定に関するチュートリアル を参照)。

3scale では OAS 仕様をさまざまな方法で拡張し、独自のインタラクティブな API ドキュメント作成で必要となった特定の機能に対応しました。

- API キーの自動入力
- CORS 非対応 API への呼び出しを許可する OAS プロキシー

2.3. API 仕様の作成

まず、オリジナルのソース OAS 仕様 から、オリジナルの仕様を確認することを推奨します。

OAS サイトには、仕様の例が複数あります。例を使って学習したい場合は、OAS API チームが作成した Petstore API の例に従うことができます。

2.3.1. Petstore API の例を使った学習

Petstore API は非常にシンプルな API です。これは、学習を目的とするものであって、実稼働用ではありません。

Petstore API は、4 つのメソッドで設定されています。

- **GET /api/pets**: システムからすべてのペットを返す
- **POST /api/pets**: ストアに新しいペットを作成する
- **GET /api/pets/{id}**: ID に基づき 1 つのペットを返す
- **DELETE /api/pets/{id}**: ID に基づき 1 つのペットを削除する

Petstore API は 3scale API Management と統合されます。このため、認証用に新たなパラメーターを追加する必要があります。たとえば、ユーザーキー認証方法では、このパラメーターはヘッダーで送信されます。他の認証方法については、API ゲートウェイの管理の 認証パターン を参照してください。

以下のパラメーターを追加する必要があります。

user_key: {user_key}

user_key は、開発者により API へのリクエストで送信されます。開発者は、デベロッパーポータルでこれらのキーを取得します。キーの受信時に、Service Management API を使用して 3scale に対する承認チェックを行う必要があります。

開発者にとって、cURL 呼び出しで表される API のドキュメントは、以下のようになります。

```
curl -X GET "http://example.com/api/pets?tags=TAGS&limit=LIMIT" -H "user_key: {user_key}"
curl -X POST "http://example.com/api/pets" -H "user_key: {user_key}" -d "{ "name": "NAME", "tag": "TAG", "id": ID }"
curl -X GET "http://example.com/api/pets/{id}" -H "user_key: {user_key}"
curl -X DELETE "http://example.com/api/pets/{id}" -H "user_key: {user_key}"
```

ただし、ドキュメントを OAS Petstore のドキュメント のようにしたい場合は、関連の Petstore **swagger.json** ファイルのような Swagger 準拠の仕様を作成する必要があります。この仕様をそのまま使用して、ActiveDocs をテストすることができます。ただし、これは実際の API ではないことに注意してください。詳細は、次のセクションで説明します。

2.3.2. OAS 仕様の詳細について

OAS 仕様は、JSON でエンコードされたハッシュにマッピングするリソース宣言に依存します。Petstore **swagger.json** ファイルを例にして、ステップごとに手順を説明します。

2.3.2.1. OAS オブジェクト

これは API 仕様のルートドキュメントオブジェクトです。最上位レベルのフィールドをすべて一覧表示します。

警告

ホストは、IP アドレスではなくドメインでなければなりません。3scale は、デベロッパーポータルに対して行われたリクエストをプロキシーとしてホストに中継し、結果をレンダリングします。そのためには、セキュリティー上の理由から、ホストおよび basePath エンドポイントがホワイトリストに登録されている必要があります。宣言できるのは、ご自分のホストだけです。API プロバイダーが自身に属さないドメインを中継していることが確認された場合、3scale は API プロバイダーのアカウントを終了する権利を有します。つまり、ローカルホストまたはその他のワイルドカードドメインは機能しません。

2.3.2.2. Info オブジェクト

Info オブジェクトは API に関するメタデータを提供します。これは ActiveDocs ページに提示されます。

2.3.2.3. paths オブジェクト

paths オブジェクトは、個々のエンドポイントへの相対パスを保持します。パスが basePath に追加され、完全な URL となります。ACL の制約により、パスは空である場合があります。

オブジェクトではないパラメーターは、プリミティブデータタイプを使用します。Swagger では、これらは JSON スキーマドラフト 4 でサポートされるデータタイプに基づきます。プリミティブデータタイプには file もありますが、これが機能するのは、API エンドポイントで CORS が有効な場合 (したがって、アップロードは api-docs ゲートウェイを経由しない) のみです。そうでない場合は、ゲートウェイレベルで停止します。

サポートされるデータタイプ

現在、OAS は以下の **データタイプ** をサポートしています。

- 整数型フォーマット: int32 および int64。どちらのフォーマットも符号付きです。
- 数値型フォーマット: float および double
- プレーンテキストおよび文字列型フォーマット byte、date、date-time、password、および binary
- boolean

2.3.3. 有用なツール

JSON 表記を十分に理解している場合は、JSON Editor Online が便利です。JSON を縮小化する整形フォーマットを提供すると共に、JSON オブジェクトブラウザーも提供します。

OAS Editor も便利なツールです。このツールを使用すると、YAML で書かれた OAS API 仕様をブラウザーで作成および編集し、リアルタイムでプレビュー表示することができます。有効な JSON 仕様を生成することもできます。これは、3scale 管理ポータルで後からアップロードできます。機能が限定されたライブデモ バージョンを使用することや、独自の OAS エディターをデプロイすることができます。

2.3.3.1. OAS 仕様の拡張機能: API キーの自動入力

API キーの自動入力は、3scale ActiveDocs の OAS 仕様に対する有用な拡張機能です。parameters セクションで、API 認証モードに応じて、**x-data-threescale-name** フィールドに以下の値を定義できます。

- **user_keys**: API キー認証のみを使用するサービスのアプリケーションのユーザーキーを返します。
- **app_ids**: アプリケーション ID/アプリケーションキーを使用するサービスのアプリケーションの ID を返します (後方互換のために、OAuth と OpenID Connect もサポートされています)。
- **app_keys**: アプリケーション ID/アプリケーションキーを使用するサービスのアプリケーションのキーを返します (後方互換のために、OAuth と OpenID Connect もサポートされています)。
- **client_ids**: OAuth/OpenID Connect 認証のみを使用するサービスのアプリケーションのクライアント ID を返します。
- **client_sercrets**: OAuth/OpenID Connect 認証のみを使用するサービスのアプリケーションのクライアントシークレットを返します。

API キー認証の例

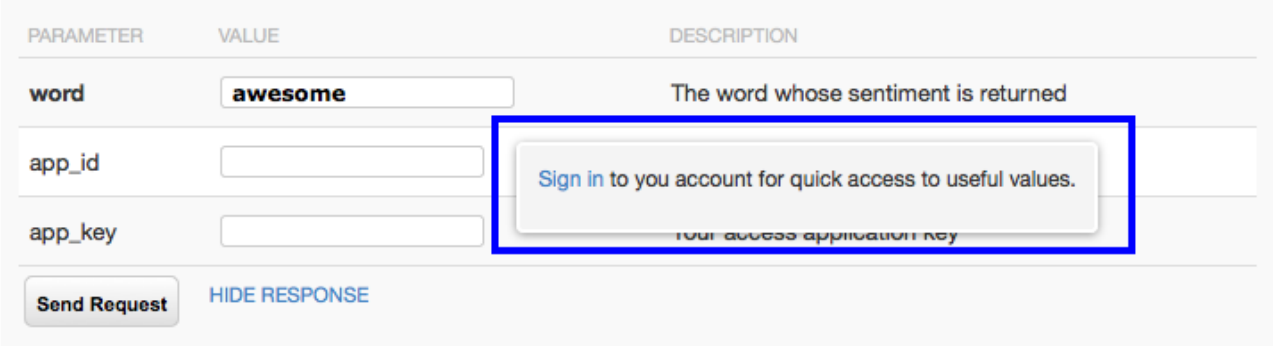
API キー認証のみの場合に **"x-data-threescale-name": "user_keys"** を使用する例を以下に示します。

```
"parameters": [
  {
    "name": "user_key",
    "description": "Your access API Key",
    "type": "string",
    "in": "query",
    "x-data-threescale-name": "user_keys",
    "required": true
  },
]
```

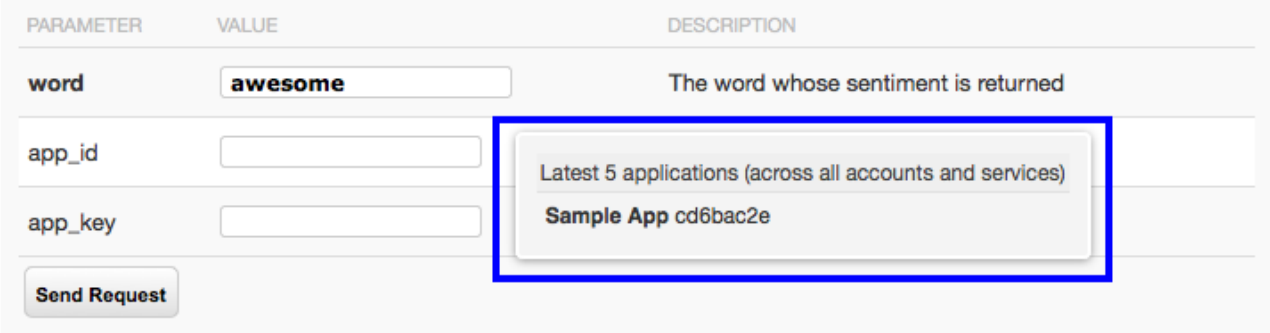
アプリケーション ID/アプリケーションキー認証の例

アプリケーション ID/アプリケーションキー認証モードの場合には、アプリケーション ID を表すパラメーターには **"x-data-threescale-name": "app_ids"** を指定し、アプリケーションキーを表すパラメーターには **"x-data-threescale-name": "app_keys"** を指定します。

パラメーターを宣言していると、ActiveDocs は自動的に、ActiveDocs ユーザーにデベロッパーポータルにログインしてキーを取得するよう求めます (以下のスクリーンショットを参照)。



ユーザーがすでにログインしている場合は、ActiveDocs は該当する可能性のある直近 5 つのキーを表示します。これにより、キーをコピー/ペーストせずにすぐテストすることができます。



注記

x-data-threescale-name フィールドは OAS 仕様の拡張機能で、ActiveDocs 以外のユースケースでは無視されます。

第3章 ACTIVEDOCS と OAUTH

本チュートリアルでは、ユーザーが簡単に OAuth 対応の API をテストして呼び出せるように、一元的にさまざまな ActiveDocs を設定する方法について説明します。

OAuth 対応の API がある場合、ユーザーにその機能に注目してもらいたい場合があります。ActiveDocs を使用してこれを行うには、どうしたらよいでしょうか。これは通常の場合より複雑ですが、可能ではあります。

3.1. 前提条件

設定を始める前に、Red Hat Single Sign-On インスタンスおよび OpenID Connect インテグレーションを設定しておく必要があります。設定方法の詳細は、OpenID Connect インテグレーション に関するドキュメントを参照してください。また、ActiveDocs の設定方法を十分理解する必要もあります。3scale への仕様の追加 および OAS 仕様の作成 を参照してください。

3.2. クライアントクレデンシャルフローとリソースオーナーフロー

この最初の例は、OAuth 2.0 クライアントクレデンシャルフローを使用する API に関するものです。この API は任意のパスを受け入れ、リクエストに関する情報 (パス、リクエストパラメーター、ヘッダー等) を返します。Echo API には、有効なアクセストークンがなければアクセスできません。API のユーザーは、クレデンシャル (**client_id** と **client_secret**) を交換してアクセストークンを取得するまで、API を呼び出すことができません。

ユーザーが ActiveDocs から API を呼び出せるようにするには、アクセストークンをリクエストする必要があります。これは単なる OAuth 承認サーバーへの呼び出しなので、OAuth トークンエンドポイント用の ActiveDocs 仕様を作成できます。これにより、ActiveDocs 内からこのエンドポイントを呼び出すことができます。この例のクライアントクレデンシャルフローの場合、Swagger JSON 仕様は以下のようになります。

```
{
  "swagger": "2.0",
  "info": {
    "version": "v1",
    "title": "OAuth for Echo API",
    "description": "OAuth2.0 Client Credentails Flow for authentication of our Echo API.",
    "contact": {
      "name": "API Support",
      "url": "http://www.swagger.io/support",
      "email": "support@swagger.io"
    }
  },
  "host": "red-hat-sso-instance.example.com",
  "basePath": "/auth/realms/realm-example/protocol/openid-connect",
  "schemes": [
    "http"
  ],
  "paths": {
    "/token": {
      "post": {
        "description": "This operation returns the access token for the API. You must call this before
calling any other endpoints.",
        "operationId": "oauth",
        "parameters": [
```

```
        {
          "name": "client_id",
          "description": "Your client id",
          "type": "string",
          "in": "query",
          "required": true
        },
        {
          "name": "client_secret",
          "description": "Your client secret",
          "type": "string",
          "in": "query",
          "required": true
        },
        {
          "name": "grant_type",
          "description": "OAuth2 Grant Type",
          "type": "string",
          "default": "client_credentials",
          "required": true,
          "in": "query",
          "enum": [
            "client_credentials",
            "authorization_code",
            "refresh_token",
            "password"
          ]
        }
      ]
    }
  }
}
```

リソースオーナー OAuth フローでは、ユーザー名とパスワードのパラメーターに加え、アクセストークンを発行するために必要なその他のパラメーターも追加する必要があります。このクライアントクレデンシャルフローの例では、client_id と client_secret (3scale からのサインイン済みユーザーの値を代入可能)、ならびに grant_type を送信するだけです。

次に、Echo API の ActiveDocs 仕様で、**client_id** と **client_secret** の代わりに **access_token** パラメーターを追加する必要があります。

```
{
  "swagger": "2.0",
  "info": {
    "version": "v1",
    "title": "Echo API",
    "description": "A simple API that accepts any path and returns information about the request",
    "contact": {
      "name": "API Support",
      "url": "http://www.swagger.io/support",
      "email": "support@swagger.io"
    }
  },
  "host": "echo-api.3scale.net",
  "basePath": "/v1/words",
```

```
  "schemes": [
    "http"
  ],
  "produces": [
    "application/json"
  ],
  "paths": {
    "/{word}.json": {
      "get": {
        "description": "This operation returns information about the request (path, request parameters,
headers, etc.),
        "operationId": "wordsGet",
        "summary": "Returns the path of a given word",
        "parameters": [
          {
            "name": "word",
            "description": "The word related to the path",
            "type": "string",
            "in": "path",
            "required": true
          },
          {
            "name": "access_token",
            "description": "Your access token",
            "type": "string",
            "in": "query",
            "required": true
          }
        ]
      }
    }
  }
}
```

その後は、通常どおりデベロッパーポータルに ActiveDocs を含めることができます。この場合、OAuth エンドポイントが最初に表示されるよう順番を指定する必要があるため、以下のようになります。

```
{% active_docs version: "2.0" services: "oauth" %}




<script type="text/javascript">
  $(function () {
    window.swaggerUi.load(); // <-- loads first swagger-ui

    // do second swagger-ui

    var url = "/swagger/spec/echo-api.json";
    window.anotherSwaggerUi = new SwaggerUi({
      url: url,
      dom_id: "another-swagger-ui-container",
      supportedSubmitMethods: ['get', 'post', 'put', 'delete', 'patch'],
      onComplete: function(swaggerApi, swaggerUi) {
```

```
        $('#another-swagger-ui-container pre code').each(function(i, e) {hljs.highlightBlock(e)});
      },
      onFailure: function(data) {
        log("Unable to Load Echo-API-SwaggerUI");
      },
      docExpansion: "list",
      transport: function(httpClient, obj) {
        log("[swagger-ui]>>> custom transport.");
        return ApiDocsProxy.execute(httpClient, obj);
      }
    });

    window.anotherSwaggerUi.load();

  });
</script>
```

第4章 デベロッパーポータルでの ACTIVEDOCS の公開

本チュートリアルでは、デベロッパーポータルで ActiveDocs を公開する方法について説明します。

Swagger が満足できるものになり、それを 3scale に追加 したら、続いてこれを公開してデベロッパーポータルにリンクを設定し、API 開発者が使用できるようにします。

以下のスニペットをデベロッパーポータルの任意のページのコンテンツに追加する必要があります。この操作は、デベロッパーポータルの CMS で行わなければなりません。SERVICE_NAME はサービス仕様のシステム名 (この例では pet_store) であることに注意してください。

```
<h1>Documentation</h1>
<p>Use our live documentation to learn about Echo API</p>
{% active_docs version: "2.0" services: "SERVICE_NAME" %}
<script type="text/javascript">
  $(function () {
    {% comment %}
      // you have access to swaggerUi.options object to customize its behaviour
      // such as setting a different docExpansion mode
      window.swaggerUi.options['docExpansion'] = 'none';
      // or even getting the swagger specification loaded from a different url
      window.swaggerUi.options['url'] = "http://petstore.swagger.io/v2/swagger.json";
    {% endcomment %}
    window.swaggerUi.load();
  });
</script>
```

注記

- 1 ページに指定できるサービスは 1 つだけです。複数の仕様を表示する場合は、別々のページで表示するのが最善の方法です。
- このスニペットには jQuery が必要ですが、デフォルトでデベロッパーポータルの main layout に含まれています。main layout から jQuery 依存関係を削除する場合は、この依存関係を ActiveDocs が含まれるページに追加する必要があります。
- CMS ページで Liquid タグが有効になっていることを確認します。
- Liquid タグで使用されるバージョン **{{ '{% active_docs version: "2.0" ' }}%}** は、Swagger 仕様のバージョンに対応している必要があります。
- 外部ソースから仕様を取得する場合は、以下のように JavaScript コードを変更します。

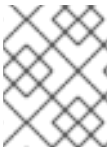
```
$(function () {
 window.swaggerUi.options['url'] = "SWAGGER_JSON_URL";
 window.swaggerUi.load();
});
```

スニペット 14 行目の例を確認してください。この行は、コメントブロックの中には置かないでください。

第5章 SWAGGER UI 2.1.3 から 2.2.10 へのアップグレード

使用している 3scale が Swagger UI 2.1.3 の組み込まれているバージョンである場合、Swagger UI バージョン 2.2.10 にアップグレードできます。

3scale デベロッパーポータルで以前実装されていた Swagger UI 2.1.3 は、**Documentation** ページに 1 つの **{% active_docs version: "2.0" %}** Liquid タグがあることが条件です。3scale で Swagger 2.2.10 がサポートされたことに伴い、実装メソッドは複数の Liquid タグ **cdn_asset** と **include** に変わっています。



注記

3scale の以前のバージョンの Swagger UI は、引き続き、従来の **active_docs** Liquid タグメソッドを使用して呼び出されます。

Swagger UI 2.1.3 を 2.2.10 にアップグレードするには、以下の手順を実施します。

1. 3scale AMP 管理ポータルにログインします。
2. **Developer Portal → Documentation** ページの順に移動するか、Swagger UI 実装を更新するページに移動します。
3. コードペインで、**{% active_docs version: "2.0" %}** Liquid タグを **cdn_asset** Liquid タグの以下のアセットと新しいパーシャル **shared/swagger_ui** に置き換えます。

```
{% cdn_asset /swagger-ui/2.2.10/swagger-ui.js %}
{% cdn_asset /swagger-ui/2.2.10/swagger-ui.css %}

{% include 'shared/swagger_ui' %}
```

4. デフォルトでは、Swagger UI は **APIs > ActiveDocs** に公開された ActiveDocs 仕様を読み込みます。別の仕様を読み込むには、**window.swaggerUi.load();** の行の前に以下の **window.swaggerUi.options** の行を追加します。ここで、**<SPEC_SYSTEM_NAME>** は読み込む仕様のシステム名です。

```
window.swaggerUi.options['url'] = "{{provider.api_specs.<SPEC_SYSTEM_NAME>.url}}";
```

パート II. API のバージョン管理

第6章 API のバージョン管理

3scale API Management Platform では、API のバージョン管理が可能です。3scale で API を管理する際、3 とおりの方法でご自分の API のバージョンを正しく管理することができます。3scale ゲートウェイで API のバージョンを管理する方法の例を以下に示します。3scale ゲートウェイは、3scale アーキテクチャーにより新たな機能を提供します。

6.1. 目的

本項では、3scale に API バージョン管理システムを実装するための詳細な情報を説明します。

あなたは楽曲を検索するための API を所有しているとします。ユーザーは、さまざまなキーワード (例: アーティスト、作詞家、曲のタイトル、アルバムタイトル等) を使用して好みの楽曲を検索することができます。API の初期バージョン (v1) があり、新たな改良バージョン (v2) を開発したと仮定します。

以降のセクションで、3scale を使用して API バージョン管理システムを実装する最も一般的な方法を 3 通り説明します。

- URL によるバージョン管理
- エンドポイントによるバージョン管理
- カスタムヘッダーによるバージョン管理

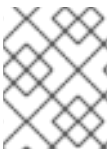
6.2. 前提条件

以下に示すクイックスタートガイドを使用する前に、API を 3scale に接続する ための基本手順を完了してください。

6.3. URL によるバージョン管理

楽曲の検索用に異なるエンドポイントがある場合には (アーティストで検索、曲のタイトルで検索など)、URL によるバージョン管理では URI の一部に API バージョンを含めてください。以下に例を示します。

1. api.songs.com/v1/songwriter
2. api.songs.com/v2/songwriter
3. api.songs.com/v1/song
4. api.songs.com/v2/song
5. 以下、同様



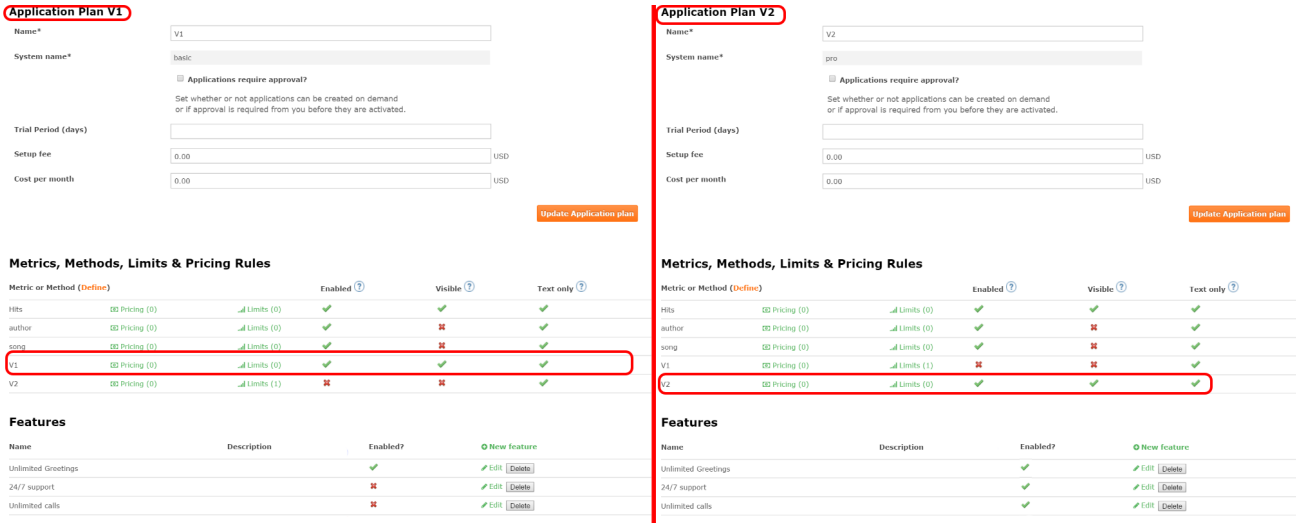
注記

この手法を用いる場合には、v1 の時点で API のバージョン管理を行うことを計画している必要があります。

この場合、3scale ゲートウェイは URI からエンドポイントおよびバージョンを抽出します。このアプローチでは、バージョンとエンドポイントの任意の組み合わせで、アプリケーションプランを設定することができます。続いて、メトリクスをこれらのプランおよびエンドポイントに関連付け、それぞれのエンドポイントおよびバージョンごとに、使用状況を把握することができます。

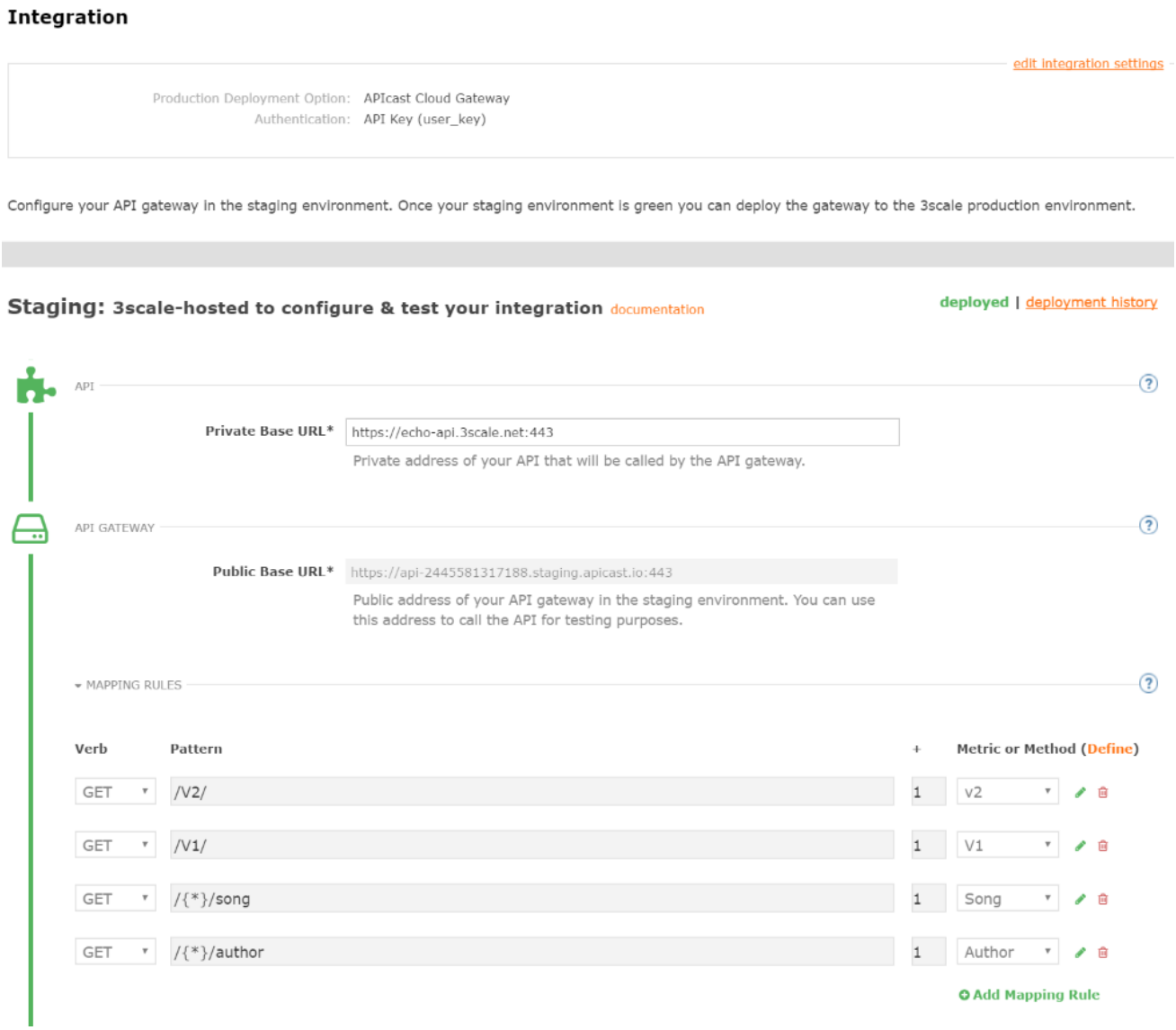
3scale 機能の柔軟性を以下のスクリーンショットに示します。

図6.1 バージョン管理計画機能



後は、以下の図に示すように、3scale 管理ポータルの **[your_API_name] > Integration > Configuration** の順に移動し、URI をメトリクスにマッピングするだけです。

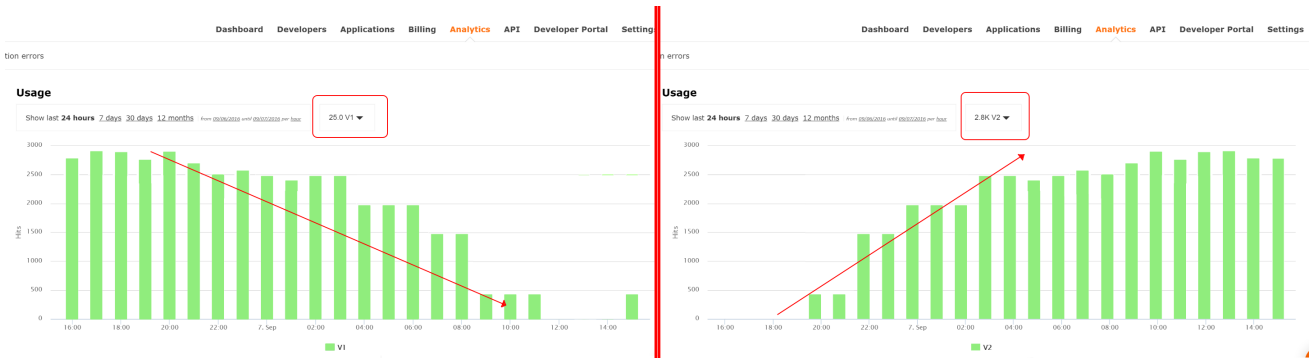
図6.2 メトリクスへの URI のマッピング



これで、異なる機能が有効になった 2 つの異なる API バージョンを使用することができます。それぞれの API の使用状況を、完全に管理し把握することができます。

API v2 に移行する必要があることをすべてのユーザーに伝える場合には、移行を依頼するメッセージを送信することができます。どのユーザーが移行したかを監視し、v1 の使用が減少して v2 の使用が増加する様子を確認することができます。3scale への承認呼び出しにメトリクスを追加して、v1 エンドポイントと v2 エンドポイントにアクセスする全トラフィック量を比較し、v1 を廃止しても問題ない時期を判断することができます。

図6.3 バージョン管理



一部のユーザーが v1 を使用し続けている場合には、それらのユーザーを絞り込み、v2 への切り替えに関する新たなメッセージを送信することができます。

3scale では、3 段階のステップにより廃止の通知を送信することができます。

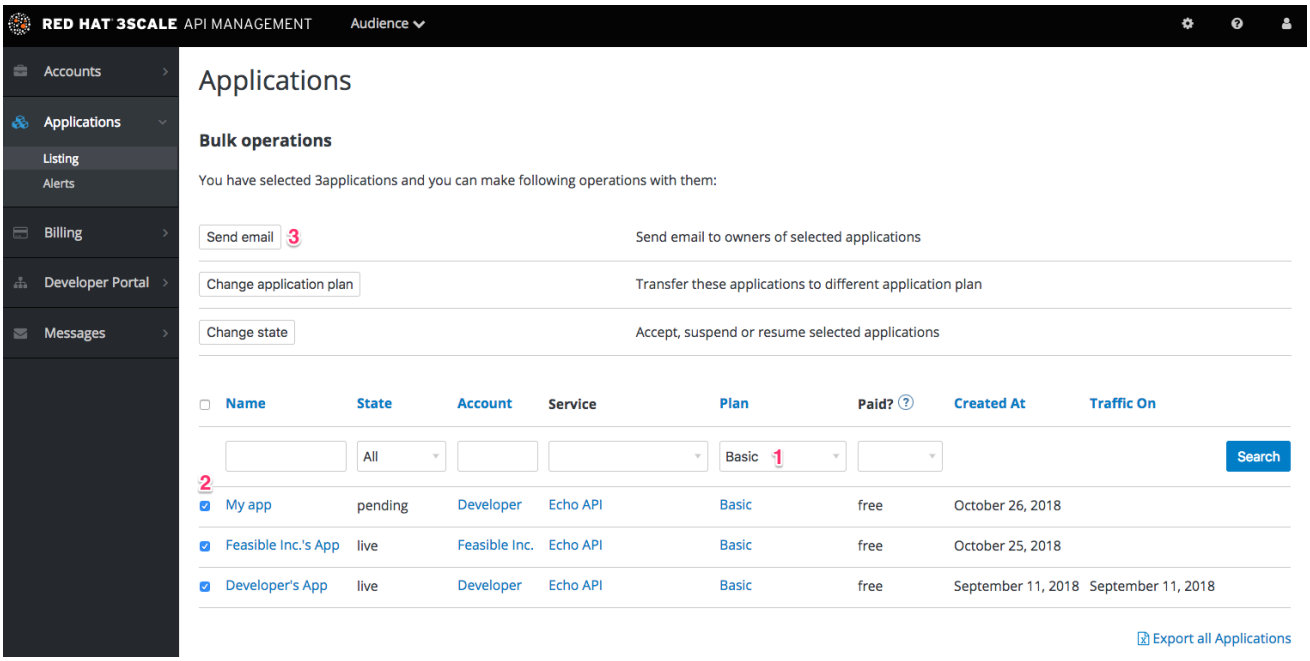
1. **Audience** > **Applications** > **Listing**の順に移動し、廃止の通知を送信するアプリケーションプランを選択して **Search** をクリックし、リストを絞り込みます。

2. チェックボックスをクリックし (複数選択可能)、問題の特定バージョンのユーザーをすべて選択します。新たなオプションが表示され、一括操作 (**電子メールの送信**、**アプリケーションプランの変更**、および **ステータスの変更**) を行うことができます。

3. **Send email** をクリックし、手順に従って、非推奨バージョンをまだ使用しているお客様に廃止の通知を送信します。

以下の図を参照してください。

図6.4 廃止通知の送信



エンドポイントに authrep 呼び出しが行われるたびに、認証は 1 度だけですが報告は 2 度行われます (エンドポイント用と API バージョン用にそれぞれ 1 度ずつ)。呼び出しは 1 度しか認証されないので、二重に請求することはありません。ある API バージョンの任意のエンドポイントに呼び出しを行うたびに、バージョン番号 (v1、v2 等) から名前を取ったメトリクスのヒット数が積算されます。これを使用して、バージョンごとの全トラフィックを相互に比較することができます。

6.4. エンドポイントによるバージョン管理

エンドポイントのバージョン管理を使用して、各バージョン (api.cons.com/author_v1) のエンドポイントを変更します。ゲートウェイは、エンドポイントそのものからエンドポイントおよびバージョンを抽出します。本手法および前項の手法では、API プロバイダーは外部 URL を内部 URL にマッピングすることができます。

エンドポイントによるバージョン管理手法は、オンプレミスデプロイメントでのみ実施することができます。この手法に必要な URL の書き換えに使用する LUA スクリプトが、オンプレミス設定の一環として提供されるためです。

外部 URL		内部 URL
api.songs.com/songwriter_v1	は、次のように書き換えられます。	internal.songs.com/search_by_songwriter
api.songs.com/songwriter_v2	は、次のように書き換えられます。	internal.songs.com/songwriter

ほとんどすべて (マッピング、アプリケーションプランの機能など) が、前項の手法と全く同じように機能します。

6.5. カスタムヘッダーによるバージョン管理

カスタムヘッダーによるバージョン管理では、バージョンを指定するのに URI ではなくヘッダー (x-api-version) を使用します。

この場合、ゲートウェイはパスからエンドポイントを、ヘッダーからバージョンを、それぞれ抽出します。前述の手法とまったく同様に、パス/バージョンの任意の組み合わせを解析して可視化することができます。このアプローチには、使用する API 管理システムにかかわらず、いくつかのデメリットがあります。詳細については、API versioning methods: A brief reference を参照してください。3scale が機能する仕組みを以下に示します。

- 前項の手法とまったく同様に、カスタムヘッダーによるバージョン管理は、オンプレミスのホスト型 API にのみ適用可能です。authrep 呼び出しを正しくルーティングするには、リクエストヘッダーの解析/処理が必要だからです。このタイプのカスタム処理の実施には、Lua スクリプトの使用が不可欠です。

- この手法では、前項の手法のようなきめ細かな機能分離を実現するのが非常に困難です。

- この方法を使用する最大の利点の 1 つであり、一部の API プロバイダーがこの方法を選択する主な理由は、顧客の URL とエンドポイントが変更されないためです。開発者がある API バージョンから別の API バージョンに切り替える場合、変更する必要があるのはヘッダーだけです。それ以外はすべて同じままで機能します。

パート III. API の認証

第7章 認証パターン

本チュートリアルでは、API に認証パターンを設定する方法、およびアプリケーションと API 間の通信への影響について説明します。

API にアクセスするためのクレデンシャルを発行するのに、API に応じて異なる認証パターンを使用しなければならない場合があります。これには、API キー、penAuth トークン、およびカスタム設定が含まれます。本チュートリアルでは、利用可能な標準的な認証パターンから適切なパターンを選択する方法について説明します。

7.1. サポートされる認証パターン

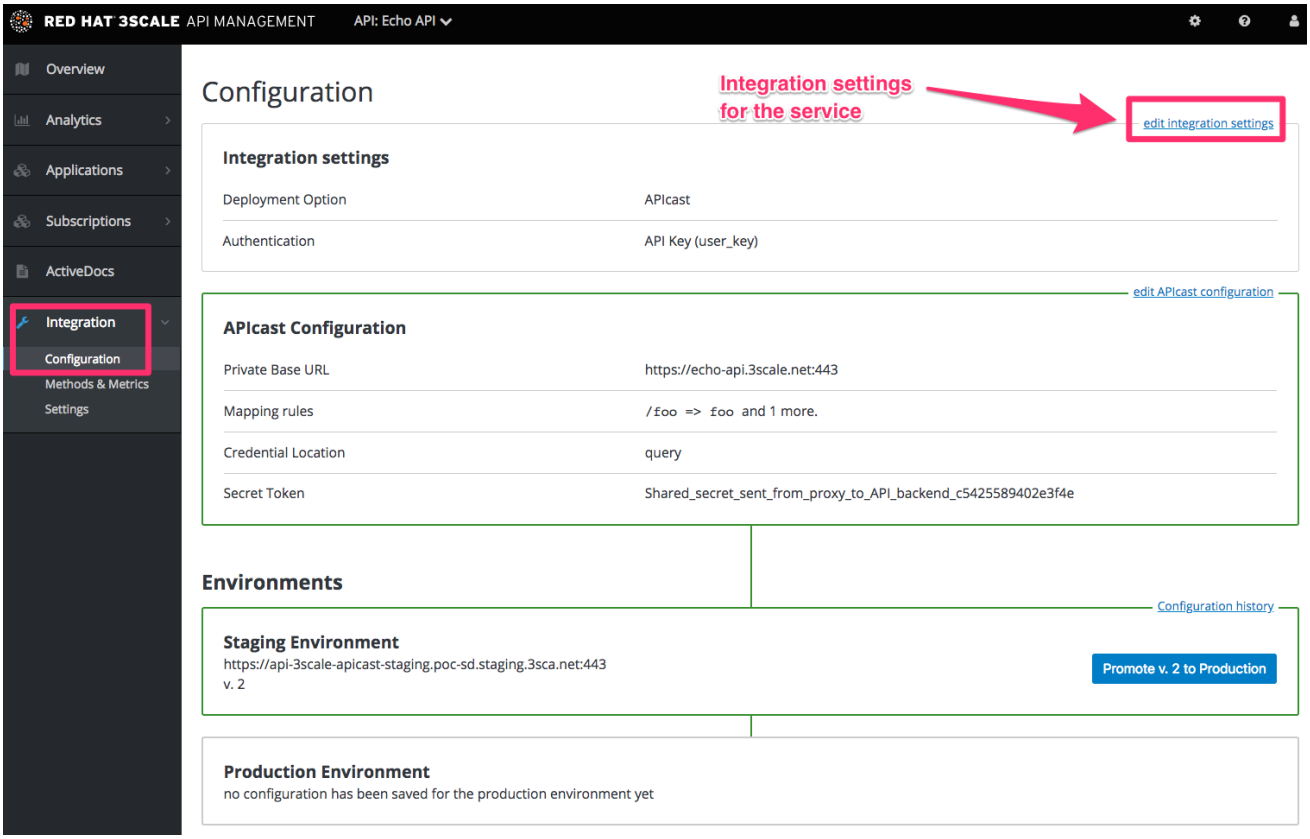
3scale では、以下の認証パターンが標準でサポートされます。

- **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、1 つの無作為な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とミュータブルなシークレットキー文字列
- OpenID Connect

7.2. 認証パターンのセットアップ

7.2.1. サービスに設定する認証モードの選択

設定を行う API サービスに移動します (API という名称のサービスが 1 つしかない場合には、それを選択します)。**Integration** セクションに移動します。



運用するサービスごとに異なる認証パターンを使用できますが、1つのサービスに使用することのできるパターンは1つだけです。

重要

サービスの動作が予測不能になる可能性があるため、クレデンシャルを登録したら認証パターンを変更しないでください。認証パターンを変更するには、新しいサービスを作成して顧客を移行することを推奨します。

7.2.2. 使用する認証モードの選択

認証モードを選択するには、AUTHENTICATION セクションまでスクロールします。ここで、以下のオプションのいずれかを選択することができます。

- API Key (user_key)
- App_ID and App_Key Pair
- OpenID Connect

7.2.3. API が正しいクレデンシャルタイプを受け入れることの確認

選択したクレデンシャルのタイプごとに、API の呼び出しで異なるパラメーター (キーフィールド、ID 等) を受け入れなければならない場合があります。これらのパラメーターの名前は、3scale 内部で使用されているものとは異なる場合があります。3scale バックエンドへの呼び出しで正しいパラメーター名が使用されている場合に、3scale の認証は正しく機能します。

7.2.4. クレデンシャルをテストするためのアプリケーションの作成

クレデンシャルセットが機能していることを確認するには、新しいアプリケーションを作成して API を使用するためのクレデンシャルを発行します。管理ポータル Dashboard の Accounts 領域に移動し、使用するアカウントをクリックして **new application** をクリックします。

フォームに入力して save をクリックすると、API を使用するためのクレデンシャルと共に新しいアプリケーションが作成されます。これで、これらのクレデンシャルを使用して API を呼び出すことができ、3scale に登録されたアプリケーションのリストに対して記録が照合されます。

7.3. 標準の認証パターン

3scale でサポートされる認証パターンの詳細を、以下のセクションで説明します。

7.3.1. API キー

サポートされるクレデンシャルの中で最もシンプルな形態は、単一の API モデルです。API へのアクセス権限を持つアプリケーションは、それぞれ1つの (一意の) 長い文字列を持ちます。以下に例を示します。

```
API-key = 853a76f7c8d5f4a1ee8bf10a4e0d1f13
```

デフォルトでは、キーパラメーターの名前は **user_key** です。このラベルを使用するか、**API-key** 等の別のラベルを選択することができます。別のラベルを選択する場合には、3scale への承認呼び出しを行う前に値をマッピングする必要があります。この文字列は、API を使用するための識別子およびシーク

レットトークンの両方として機能します。このパターンは、セキュリティーに対する要件が低い環境または API の呼び出しに SSL セキュリティーが使用される環境でのみ使用することを推奨します。トークンおよびアプリケーションに対して実行される操作は、以下のとおりです。

- アプリケーションの一時中止: アプリケーションの API へのアクセスが一時中止され、実質的に、該当するキーを使用する API への呼び出しがすべて一時中止されます。
- アプリケーションの再開: アプリケーションの一時中止アクションの効力を解除します。
- キーの再生成: アプリケーション用に新しい無作為な文字列キーが生成され、それがアプリケーションに関連付けられます。このアクションが実行されると、直ちに以前のトークンを使用した呼び出しは受け入れられなくなります。

最後のアクションのトリガーとなるのは、管理ポータルでの API 管理および API の開発者ユーザーのコンソール (許可される場合) です。

7.3.2. App_ID と App_Key のペア

API キーの認証パターンでは、アプリケーションの識別子とシークレットトークンが 1 つのトークンに組み合わされます。一方、この認証パターンでは両者が分離されます。API を使用する各アプリケーションは、**アプリケーション ID** (App_ID) と呼ばれるイミュータブルな初期 ID を発行します。App_ID は不変で、秘密である場合とそうでない場合があります。さらに、各アプリケーションは 1~n 個の **アプリケーションキー** (App_Keys) を持つことができます。それぞれのキーは App_ID に直接関連付けられ、秘密として扱う必要があります。

```
app_id = 80a4e03 app_key = a1ee8bf10a4e0d1f13853a76f7c8d5f4
```

デフォルトの設定では、開発者はアプリケーションごとに最大 5 つのキーを作成することができます。これにより、開発者は新しいキーを作成してコードに追加し、アプリケーションを再デプロイして古いキーを無効にすることができます。そのため、API キーを再生成する場合のようなアプリケーションのダウンタイムが発生することはありません。

統計値および流量制御は、API キーごとではなく、常にアプリケーション ID レベルで維持されます。開発者が 2 組の統計値を追跡するためには、2 つのキーではなく 2 つのアプリケーションを作成する必要があります。

システムのモードを変更し、アプリケーションキーを持たないアプリケーションを作成することもできます。この場合には、3scale システムは App_ID のみに基づいてアクセスを認証します (キーの確認は行われません)。このモードは、ウィジェットタイプのシナリオ、またはアプリケーションではなくユーザーに流量制御が適用される場合に役立ちます。ほとんどの API では、アプリケーションごとに少なくとも 1 つのアプリケーションキーを持つ必要があります。この設定は、**[your_API_name] > Integration > Settings** で可能です。

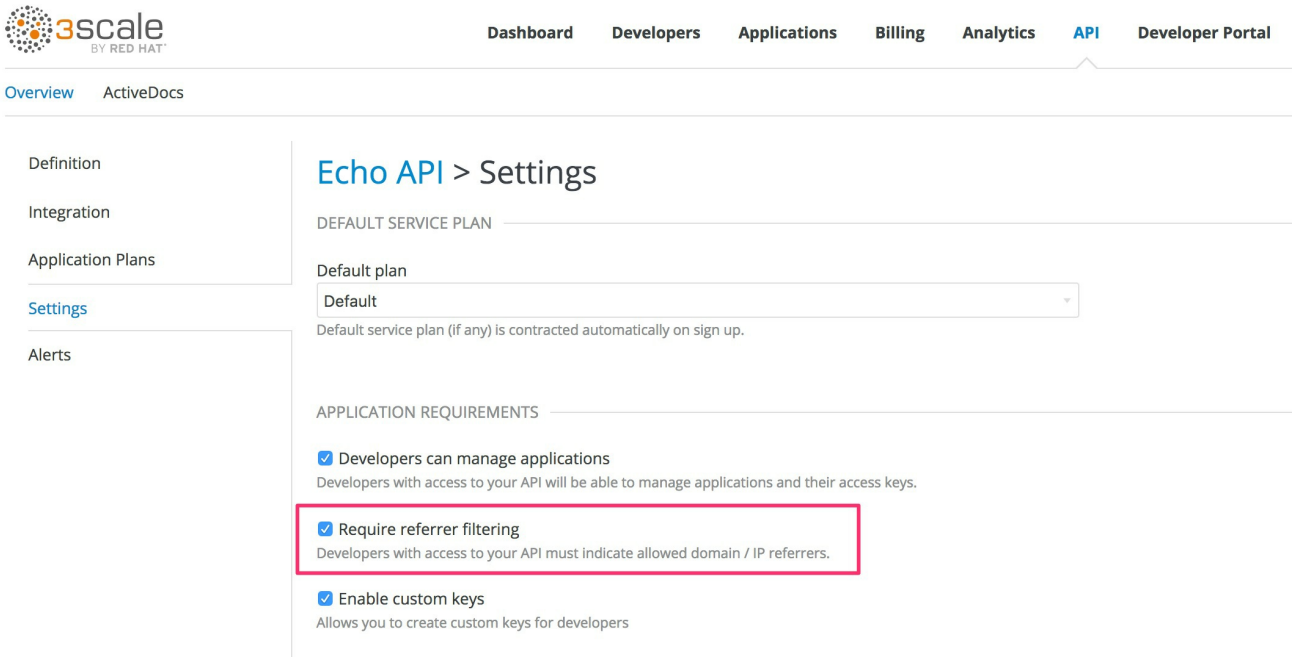
7.3.3. OpenID Connect

OpenID Connect による認証に関する情報は、OpenID Connect インテグレーション のセクションを参照してください。

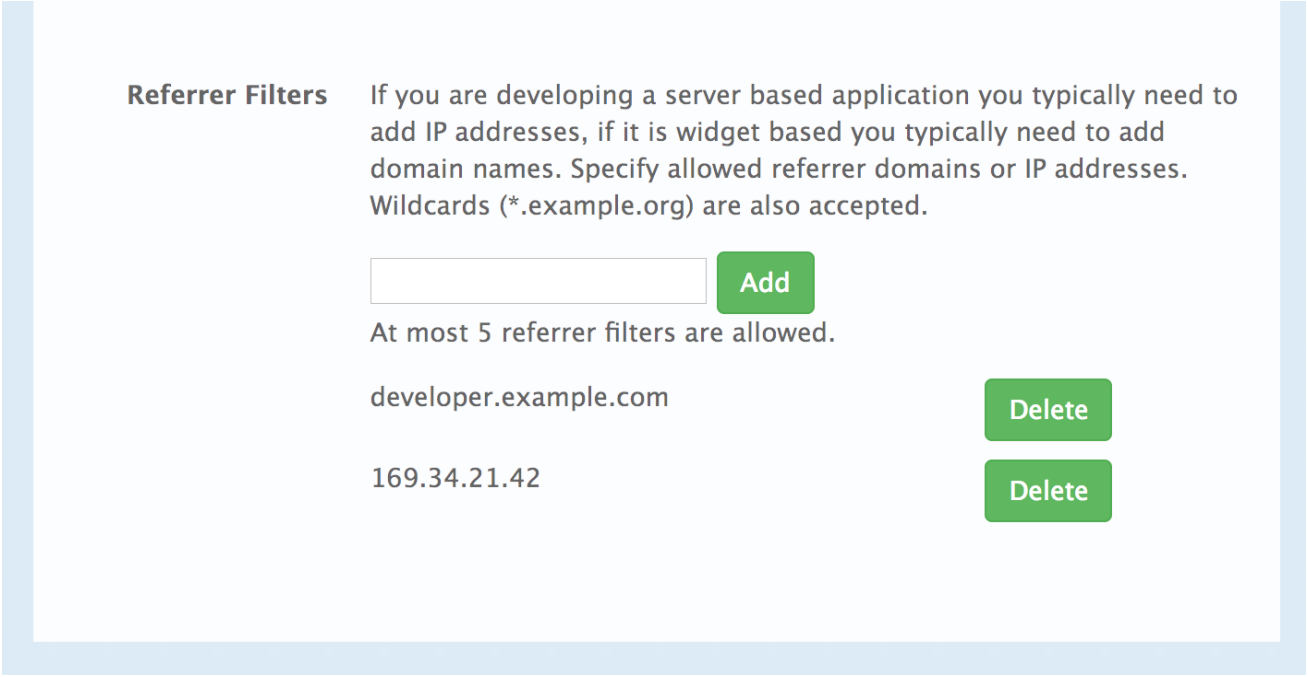
7.4. 参照元フィルター機能

3scale では参照元フィルター機能がサポートされ、API へのアクセスが許可されるアプリケーションの IP アドレスまたはドメイン名をホワイトリストに登録することができます。API クライアントでは、**Referrer** ヘッダーで参照元の値を指定します。Referrer ヘッダーの目的および使用法については、RFC 7231 のセクション 5.5.2Referer に説明があります。

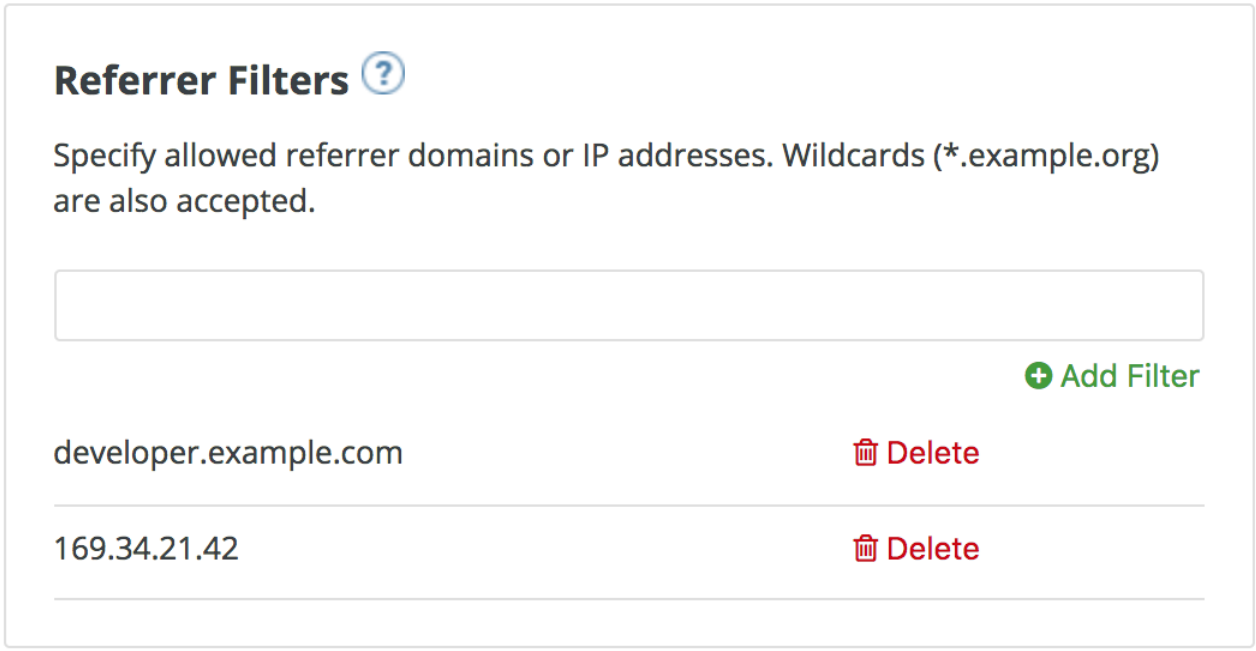
参照元フィルター機能を有効にするには、[your_API_name] > Integration > Settingsの順に移動し、**Require referrer filtering** チェックボックスを選択して **Update Service** をクリックします。



ご自分の API にアクセスする開発者は、デベロッパーポータルから許可される参照元のドメイン/IP を設定する必要があります。



管理ポータルでは、このサービスに属する全アプリケーションの詳細ページに、新たな **Referrer Filters** セクションが表示されます。ここで、管理者は、このアプリケーションの許可される Referrer ヘッダー値のホワイトリストを設定することもできます。



アプリケーションごとに、最大 5 つの参照元の値を設定することができます。

値に使用することができるのは、ラテン文字、数字、ならびに特殊文字 *、.、および - だけです。* はワイルドカードの値として使用することができます。値を * に設定するとあらゆる参照元の値が許可され、参照元の確認はバイパスされます。

リファラーフィルタリング機能を機能させるには、サービスポリシーチェーンで APIcast リファラーポリシー を有効にする必要があります。

Require referrer filtering 機能および **3scale Referrer** ポリシーが有効な場合には、承認は以下のように行われます。

1. 参照元フィルターが指定されていないアプリケーションは、提供されたクレデンシャルだけを使用して通常どおり承認されます。
2. 参照元フィルターの値が設定されたアプリケーションの場合には、APIcast はリクエストの **Referrer** ヘッダーから参照元の値を抽出し、それを AuthRep (承認およびレポート) リクエストの **referrer** パラメーターとして Service Management API に送信します。以下の表は、参照元フィルター機能のパラメーターのさまざまな組み合わせに対する AuthRep の応答をまとめています。

referrer パラメーターが渡されるか	アプリケーションに参照元フィルターが設定されているか	referrer パラメーターの値	HTTP レスポンス	レスポンスのボディー
はい	はい	参照元フィルターと一致する	200 OK	**<status><authorized>true</authorized></status>**

referrer パラメーターが渡されるか	アプリケーションに参照元フィルターが設定されているか	referrer パラメーターの値	HTTP レスポンス	レスポンスのボディー
はい	いいえ	参照元フィルターと一致する	200 OK	**<status><authorized>true</authorized></status>**
はい	はい	参照元フィルターと一致しない	409 Conflict	**<status><authorized>false</authorized><reason>referrer "test.example.com" is not allowed</reason>** (**test.example.com** は例です)
はい	いいえ	参照元フィルターと一致しない	200 OK	**<status><authorized>true</authorized></status>**
はい	はい	*	200 OK	**<status><authorized>true</authorized></status>**
はい	いいえ	*	200 OK	**<status><authorized>true</authorized></status>**
いいえ	はい	–	409 Conflict	**<status><authorized>false</authorized><reason>referrer is missing</reason>**
いいえ	いいえ	–	200 OK	**<status><authorized>true</authorized></status>**

AuthRep により承認されない呼び出しは、APIcast により拒否され Authorization Failed エラーが返されます。サービスの Integration ページで、ステータスコードおよびエラーメッセージを具体的に設定することができます。

第8章 OPENID CONNECT インテグレーション

3scale では、OpenID Connect 仕様を使用して、API リクエストの認証用にサードパーティーアイデンティティープロバイダー (IdP) を統合します。OpenID Connect は OAuth 2.0 に追加する形で構築され、OAuth 2.0 の承認フレームワークを認証メカニズムにより補完します。OpenID Connect の認証オプションが使用されると、API リクエストは JSON Web Token (JWT) 形式 (RFC 7519) のアクセストークンを使用して認証されます。

インテグレーションは、以下に示す 2 つの部分で設定されます。

- 「APIcast による JWT の検証および解析」
- 「Zync によるクライアントクレデンシャルの同期」

Red Hat 3scale API Management は、OpenID プロバイダーとして機能する Red Hat Single Sign-On (RH-SSO) により、両方のインテグレーションポイントを完全にサポートしています。サポートされる RH-SSO のバージョンについては、Red Hat 3scale API Management のサポート対象設定 のアーティクルを参照してください。APIcast のインテグレーションは、ForgeRock でもテストされています。

どちらのケースでも、OpenID Connect の認証オプションを使用するサービスの Integration ページにおいて、APIcast Configuration の **OpenID Connect Issuer** フィールドを指定して、インテグレーションを設定することができます。手順は、「Red Hat Single Sign-On インテグレーションの設定」 を参照してください。

8.1. APICAST による JWT の検証および解析

OpenID Connect の認証モードを使用するサービスに対する API リクエストは、OpenID プロバイダーにより発行された JWT 形式のアクセストークンを提供する必要があります。このトークンは、**Bearer** スキーマを使用して **Authorization** ヘッダーで提供されます。ヘッダーの例を以下に示します。

```
Authorization: Bearer <JWK>
```

例:

```
Authorization: Bearer:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2lkcC5leGFtcGxlLmNvbSIsInN1YiI
6ImFiYzEyMyIsIm5iZiI6MTUzNzg5MjQ5NCwiZXhwIjoxNTM3ODk2MDk0LCJpYXQiOjE1Mzc4OTI0OT
QsImp0aSI6ImlkMTIzNDU2IiwidHlwIjoiQmVhcmVyIn0.LM2PSmQ0k8mR7eDS_Z8iRdGta-Ea-
pJRrf4C6bAiKz-Nzhxpm7fF7oV3BOipFmimwkQ_-mw3kN--
oOc3vU1RE4FTCQGbzO1SAWHOZqG5ZUx5ugaASY-
hUHIohy6PC7dQl0e2NlAeqqg4MuZtEwrpESJW-
VnGdljrAS0HsXzd6nENM0Z_ofo4ZdTKvIKsk2KrdyVBOcjgVjYongtppR0cw30FwnpqfeCkuATeINN5OK
HXOibRA24pQyIF1s81nnmxLnjnVbu24SFE34aMGRXYzs4icMI8sK65eKxbvwV3PIG3mM0C4ilZPO26d
oP0YrLfVwFcqEirmENUAcHXz7NuvA
```

JWT トークンの受領者はトークンに含まれる署名を検証し、トークンが既知の発行者により署名されていること、およびその内容が変更されていないことを確認します。3scale では、公開/秘密鍵のペアに基づく RSA 署名がサポートされます。なお、発行者は秘密鍵を使用して JWT トークンに署名します。APIcast は公開鍵を使用してこのトークンを検証します。

APIcast では、JWT 署名の検証に使用する JSON Web Key (JWK) の取得に、OpenID Connect Discovery が使用されます。

それぞれのリクエストについて、APIcast は以下の処理を行います。

1. 公開鍵を使用して JWT トークンを検証する。
2. クレーム **nbf** および **exp** を検証する。
3. クレーム **iss** (Issuer) で指定される発行者が **OpenID Connect Issuer** フィールドで設定される発行者と同じであることを検証する。
4. **azp** または **aud** クレームの値を抽出し、それをクライアント ID (3scale 内のアプリケーションを識別する) として使用して Service Management API を通じて呼び出しを承認する。

JWT の検証または承認の確認のいずれかが失敗すると、APIcast は Authenication failed エラーを返します。そうでなければ、APIcast は API バックエンドにリクエストをプロキシー処理します。**Authorization** ヘッダーはリクエストにそのまま残るので、API バックエンドは JWT トークンを使用してユーザーおよびクライアントの ID を確認することもできます。

8.2. ZYNC によるクライアントクレデンシャルの同期

3scale では、Zync コンポーネントを使用して、3scale と **OpenID Connect Issuer** 設定で指定した Red Hat Single Sign-On サーバー間で、クライアント (アプリケーション) のクレデンシャルを同期します。OpenID Connect を使用するように設定されたサービスで新たなアプリケーションが作成/更新/削除されるたびに、Zync は該当するイベントを受け取り、RH-SSO API を使用してその変更を Red Hat Single Sign-On インスタンスに通知します。「Red Hat Single Sign-On インテグレーションの設定」 セクションで、Zync が RH-SSO API を使用するための正しいクレデンシャルを取得するのに必要な手順を説明します。

8.3. RED HAT SINGLE SIGN-ON インテグレーションの設定

8.3.1. カスタム CA 証明書を使用する Zync の設定

Zync と Red Hat Single Sign-On 間に、SSL 接続を確立する必要があります。3scale 2.2 以降は、**SSL_CERT_FILE** 環境変数により、Red Hat Single Sign-On 用カスタム CA 証明書をサポートしています。この変数は、証明書バンドルのローカルパスをポイントします。この変数を以下のように設定します。

1. 以下の cURL コマンドを使用して、新しい証明書を検証します。レルムの JSON 設定が返されるはずです。検証の失敗は、証明書が正しくないことを意味します。

```
curl -v https://<secure-sso-host>/auth/realms/master --cacert customCA.pem
```

2. 証明書バンドルを Zync Pod に追加します。
   a. Zync Pod 上の **/etc/pki/tls/cert.pem** ファイルの既存コンテンツを収集します。以下を実行します。

```
oc exec <zync-pod-id> cat /etc/pki/tls/cert.pem > zync.pem
```

   b. カスタム CA 証明書ファイルの内容を **zync.pem** に追加します。

```
cat customCA.pem >> zync.pem
```

   c. 新たなファイルを ConfigMap として Zync Pod に追加します。

```
oc create configmap zync-ca-bundle --from-file=./zync.pem
```

```
oc set volume dc/zync --add --name=zync-ca-bundle --mount-path
/etc/pki/tls/zync/zync.pem --sub-path zync.pem --source='{"configMap":{"name":"zync-ca-
bundle","items":[{"key":"zync.pem","path":"zync.pem"}]}}'
```

```
oc patch dc/zync --type=json -p '[{"op": "add", "path":
"/spec/template/spec/containers/0/volumeMounts/0/subPath", "value":"zync.pem"}]'
```

3. デプロイメント後に、証明書が追加され内容が正しいことを確認します。

```
oc exec <zync-pod-id> cat /etc/pki/tls/zync/zync.pem
```

4. 新たな CA 証明書のバンドルをポイントするように、Zync の **SSL_CERT_FILE** 環境変数を設定します。

```
oc set env dc/zync SSL_CERT_FILE=/etc/pki/tls/zync/zync.pem
```

8.3.2. Red Hat Single Sign-On の設定

Red Hat Single Sign-On を設定するには、以下の手順を実施します。

1. レルム (**<REALM_NAME>**) を作成します。
2. クライアントを作成します。
   a. クライアント ID を指定します。
   b. **Client Protocol** フィールドで、**openid-connect** を選択します。
3. 以下のように値を設定して、クライアントのアクセス権限を設定します。
   a. **Access Type**: **confidential**
   b. **Standard Flow Enabled**: **OFF**
   c. **Direct Access Grants Enabled**: **OFF**
   d. **Service Accounts Enabled**: **ON**
4. クライアントのサービスアカウントロールを設定します。
   a. クライアントの **Service Account Roles** タブに移動します。
   b. **Client Roles** ドロップダウンリストで、**realm-management** をクリックします。
   c. **Available Roles** ペインで **manage-clients** リスト項目を選択し、**Add selected >>** をクリックしてロールを割り当てます。
5. クライアントのクレデンシャルを書き留めます。
   a. クライアント ID (**<CLIENT_ID>**) を書き留めます。
   b. クライアントの **Credentials** タブに移動し、**Secret** フィールド (**<CLIENT_SECRET>**) の値を書き留めます。
6. レルムにユーザーを追加します。

a. ウィンドウ左側の **Users** メニューをクリックします。

b. **Add user** をクリックします。

c. ユーザー名を入力して **Email Verified** スイッチを **ON** に設定し、**Save** をクリックします。

d. **Credentials** タブでパスワードを設定します。両方のフィールドにパスワードを入力し、**Temporary** スイッチを **OFF** に設定して次回ログイン時のパスワードリセットを回避します。続いて **Reset Password** をクリックします。

e. ポップアップウィンドウが表示されたら、**Change password** をクリックします。

8.3.3. 3scale の設定

Red Hat Single Sign-On にクライアントを作成して設定したら、Red Hat Single Sign-On と協調するように 3scale を設定する必要があります。

3scale を設定するには、以下の手順を実施します。

1. OpenID Connect を有効にします。

   a. OpenID Connect による認証を有効にするサービスを選択し、**[your_API_name] > Integration > Configuration** の順に移動します。

   b. **edit integration settings** を選択します。

   c. **Authentication** デプロイメントオプションで、**OpenID Connect** を選択します。

   d. **Update Service** をクリックして、設定を保存します。

2. APIcast の設定を編集します。

   a. **[your_API_name] > Integration > Configuration**の順に移動します。

   b. **edit APIcast configuration** を選択します。

   c. **AUTHENTICATION SETTINGS** セクションの **OpenID Connect Issuer** フィールドに、Red Hat Single Sign-On サーバーの URL (ホスト: **<RHSSO_HOST>**、ポート: **<RHSSO_PORT>**) と共に前項の手順で書き留めたクライアントのクレデンシャルを入力します。

   ```
   https://<CLIENT_ID>:<CLIENT_SECRET>@<RHSSO_HOST>:
   <RHSSO_PORT>/auth/realms/<REALM_NAME>
   ```

   d. **Update the Staging Environment**をクリックして、設定を保存します。

8.4. OAUTH 2.0 SUPPORTED FLOWS

API クライアントは、3scale で設定した OpenID Connect 発行者からアクセストークンを取得する必要があります。この場合、その OpenID プロバイダーのサポートする任意の OAuth 2.0 フローが使用されます。Red Hat Single Sign-On の場合には、以下のフローがサポートされます (RH-SSO クライアントで使用される用語をかっこ書きで指定します)。

- 認可コード (**標準フロー**)
- リソースオーナーパスワードクレデンシャル (**直接アクセスグラントフロー**)

- インプリシット (**インプリシットフロー**)
- クライアントクレデンシャル (**サービスアカウントフロー**)

OpenID Connect (OIDC)下のクライアントが 3scale に作成されると、Red Hat Single Sign-On (RH SSO)の Zync で作成された対応するクライアントでは、Authorization Code フローのみが有効になります。このフローは、ほとんどすべてのケースについて最もセキュアで適切なフローとして推奨されます。ただし、他のフローを有効にすることが可能です。

8.4.1. OAuth 2.0 対応の認証フローの仕組み

クライアントは、承認リクエストもしくはトークンリクエスト、またはその両方を使用してアクセストークンを取得します。これらのリクエストを受け取る URL は、OpenID プロバイダーの **.well-known/openid-configuration** エンドポイントを使用して把握することができます。それぞれ、**"authorization_endpoint"** および **"token_endpoint"** が該当します。たとえば、**https://<RHSSO_HOST>:<RHSSO_PORT>/auth/realms/<REALM_NAME>/.well-known/openid-configuration**。

8.4.2. OAuth 2.0 対応の認証フローの設定

管理ポータルで、3scale API 用に許可される OAuth 2.0 フローを設定することができます。新たなアプリケーションを作成する際には、OpenId Connect (OIDC) の設定を含めて、基本的なインテグレーションは完了しています。

OAuth 2.0 対応の認証フローを設定するには、以下の手順を実施します。

1. AUTHENTICATION セクションに移動します (**[your_API_name] > Integration > Configuration > edit integration settings > AUTHENTICATION**)。
2. **OpenID Connect** を選択します。
3. RH SSO 側のクライアントで、対応するフローが有効になります。**[your_API_name] > Integration > Configuration > edit APIcast configuration > AUTHENTICATION SETTINGS** の順に移動して、それらを確認することができます。
   - **standardFlowEnabled** (Authorization Code Flow) [デフォルトで選択済み]
   - **implicitFlowEnabled** (Implicit Flow)
   - **serviceAccountsEnabled** (Service Accounts Flow)
   - **directAccessGrantsEnabled** (Direct Access Grant Flow)
4. 1つまたは複数のフローを選択します。
5. **Update the Staging Environment**をクリックして変更を保存します。

8.5. インテグレーションのテスト

インテグレーションをテストするには、以降のセクションに記載する手順を実施する必要があります。

8.5.1. クライアントの同期に関するテスト

クライアントの同期をテストするには、以下の手順を実施します。

1. OpenID Connect インテグレーションを設定したサービスにアプリケーションを作成します。

2. 生成したアプリケーションのクライアント ID およびクライアントシークレットを書き留めます。

3. 同じクライアント ID およびクライアントシークレットを持つクライアントが、設定された Red Hat Single Sign-On レルムに存在することを確認します。

4. 3scale 管理ポータルでアプリケーションのリダイレクト URL を更新します。リダイレクト用 URL は、可能な限り正確に指定する必要があります。

5. これに対応して、Red Hat Single Sign-On のクライアントの **Valid Redirect URIs** フィールドが更新されることを確認します。

8.5.2. API 承認フローのテスト

APT 承認フローをテストするには、以下の手順を実施します。

1. 対応する RH-SSO クライアントで有効な OAuth 2.0 フローを使用して、Red Hat Single Sign-On サーバーからアクセストークンを取得します。

2. **Authorization** ヘッダーで、RH-SSO から取得した **access_token** の値を使用します (例: **Authorization: Bearer <access_token>**)。

トークンが正しく対応する 3scale のアプリケーションが承認されると、APIcast ゲートウェイは API バックエンドからのレスポンスを返します。

8.6. インテグレーションの例

3scale のサービス API は、OpenID Connect による認証を使用するように設定されています。サービス API の **公開ベース URL** は **https://api.example.com** に、**プライベートベース URL** は **https://internal-api.example.com** に、それぞれ設定されています。

API のインテグレーションで **OpenID Connect Issuer** フィールドは **https://zync:41dbb98b-e4e9-4a89-84a3-91d1d19c4207@idp.example.com/auth/realms/myrealm** に設定され、レルム **myrealm** のクライアント **zync** には正しい Service Account ロールが設定されています。

3scale には、クライアント ID、クライアントシークレット、およびリダイレクト URL がそれぞれ **myclientid**、**myclientsecret**、および **https://myapp.example.com** に設定されたアプリケーションがあります。Red Hat Single Sign-On 側にも、レルム **myrealm** に、クライアント ID、クライアントシークレット、および **Valid Redirect URIs** がそれぞれ **myclientid**、**myclientsecret**、および **https://myapp.example.com** に設定されたクライアントが存在しています。このクライアントでは、標準フローが有効です。レルム **myrealm** には 1 人のユーザーが設定され、そのユーザー名およびパスワードは **myuser** および **mypassword** です。

フローは以下のようになります。

1. エンドポイント **https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/auth** を使用して、アプリケーションは RH-SSO に承認リクエストを送信します。アプリケーションは、リクエストでクライアント ID **myclientsecret** とリダイレクト URL **https://myapp.example.com** を提供する必要があります。

2. RH-SSO はログインウィンドウを表示し、ユーザーはここで自分のクレデンシャル (ユーザー名 **myuser** およびパスワード **mypassword**) を入力する必要があります。

3. 設定およびユーザーがこのアプリケーションで初めて認証されるかどうかに応じて、同意に関するウィンドウが表示されます。

4. ユーザーが認証されると、アプリケーションはエンドポイント **https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/token** を使用して、クライアント ID **myclientid**、クライアントシークレット **myclientsecret**、およびリダイレクト URL **https://myapp.example.com** と共に、トークンリクエストを RH-SSO に送信します。

5. RH-SSO は、access_token フィールドが **eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A** と設定された JSON を返します。

6. アプリケーションは、ヘッダー **Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A** と共に API リクエストを **https://api.example.com** に送信します。

7. アプリケーションは、**https://internal-api.example.com** から正常なレスポンスを受け取るはずです。

パート IV. OPENAPI SPECIFICATION (OAS)

第9章 OPENAPI 仕様 (OAS) に基づく新しいサービスの作成

9.1. はじめに

本章では、Red Hat 3scale 2.5 での OpenAPI Specification (OAS) 2.0 機能の概要、および既存サービスの更新や新規サービスの作成を行う手順について説明します。

9.2. 前提条件

- OpenAPI Specification (OAS)
- 3scale 2.5 インスタンステナントのクレデンシャル (**トークン** または **provider_key**)

9.3. OPENAPI SPECIFICATION の機能

注記

ActiveDocs は、OpenAPI (OAS) のインポート時に作成、更新されます。

- サービスの **system_name** をオプションのパラメーターとして渡すことができ、そのデフォルトは OAS からの **info.title** フィールドです。
- メソッドは OAS からの各操作に対して作成されます。
  - **メソッド** 名は **operation.operationId** フィールドから取得されます。
- 既存の **マッピングルール** は、新規 API 定義をインポートする前にすべて削除されます。
  - コマンドの実行前から存在するメソッドは削除されません。
- マッピングルールは、OAS からの各操作に対して作成されます。
- OpenAPI 定義リソースは、以下のチャンネルのいずれかで提供することができます。
  - 利用可能なパスの **ファイル名**
  - **URL** フォーマット (toolbox は所定のアドレスからダウンロードを試みます)
  - **stdin** 標準入力ストリームから読み込む

9.4. OPENAPI SPECIFICATION の使用

```
NAME
   openapi - Import API definition in OpenAPI specification

USAGE
   3scale import openapi [opts] -d <dst> <spec>

DESCRIPTION
   Using an API definition format like OpenAPI, import to your 3scale API

OPTIONS
      -d --destination=<value>                3scale target instance.
```

```
                                Format: "http[s]://<authentication>@3scale_domain"

 -t --target_system_name=<value>          Target system name

OPTIONS FOR IMPORT
   -c --config-file=<value>                  3scale toolbox
                                configuration file
                                (default:
                                $HOME/.3scalerc.yaml)
   -h --help                        show help for this command
   -k --insecure                      Proceed and operate even
                                for server connections
                                otherwise considered
                                insecure
   -v --version                      Prints the version of this
                                command
```

9.4.1. ファイル名のパスからの OpenAPI 定義の検出

使用できるフォーマットは **json** と **yaml** です。フォーマットは、ファイル名の拡張子から自動的に検出されます。

```
$ 3scale import openapi -d <destination> /path/to/your/spec/file.[json|yaml|yml]
```

9.4.2. URL からの OpenAPI 定義の検出

使用できるフォーマットは **json** と **yaml** です。フォーマットは、URL のパスの拡張子から自動的に検出されます。

```
$ 3scale import openapi -d <destination> http[s]://domain/resource/path.[json|yaml|yml]
```

9.4.3. stdin からの OpenAPI 定義の検出

OpenAPI リソースのコマンドラインパラメーターは **-** です。

使用できるフォーマットは **json** と **yaml** です。形式はパーサーにより内部的に自動検出されます。

```
$ tool_to_read_openapi_from_source | 3scale import openapi -d <destination> -
```