Red Hat 3scale API Management 2.12

API ゲートウェイの管理

3scale インストール環境のより詳細な設定方法

Last Updated: 2022-12-04

Red Hat 3scale API Management 2.12 API ゲートウェイの管理

3scale インストール環境のより詳細な設定方法

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、基本インストールの後に実施する設定タスクについて説明します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、CTO である Chris Wright のメッセージ をご覧ください。

API ゲートウェイの管理は、3scale インストール環境により詳細な設定を適用するのに役立ちます。インストールに関する基本的な情報は、3scale のインストールを参照してください。

パート I. API ゲートウェイ

第1章 3SCALE APICAST API ゲートウェイの高度な操作について

本章で説明する 3scale APIcast の高度な操作は、API へのアクセスの設定を調整するのに役立ちます。

1.1. 3SCALE API への呼び出しの公開ベース URL

公開ベース URL は API 利用者がご自分の API プロダクトにリクエストを行うのに使用する URL で、3scale により公開されます。これは、ご自分の APIcast インスタンスの URL になります。

Self-managed デプロイメントオプションのいずれかを使用している場合には、それぞれの環境 (ステージングおよび実稼働環境) について、ご自分の管理するドメインに属する専用の公開ベース URL を選択することができます。この URL はご自分の API バックエンドの URL とは別で、たとえば **https://api.yourdomain.com:443** のようになります。ここで、**yourdomain.com** はご自分のドメインです。公開ベース URL を設定したら必ず変更を保存し、必要に応じてステージング環境の変更を実稼働環境にプロモートしてください。

注記

指定する公開ベース URL は、OpenShift クラスターで利用可能なポートを使用する必要があります。デフォルトでは、OpenShift のルーターは標準の HTTP および HTTPS ポート (80 および 443) の接続しかリッスンしません。ユーザーに他のポート経由で API に接続してもらう場合は、OpenShift の管理者と連携してそのポートを有効にします。

APIcast は公開ベース URL で指定したホスト名に対する呼び出ししか受け付けません。たとえば、公開ベース URL を **https://echo-api.3scale.net:443** と指定した場合には、正しい呼び出しは以下のようになります。

```
curl "https://echo-api.3scale.net:443/hello?user_key=you_user_key"
```

API に公開ドメインがない場合には、リクエストに APIcast の IP アドレスを使用することができます。ただし、実際のドメインではなくても良いので、**Public Base URL** フィールドには値を指定する必要があります。その場合には、Host ヘッダーでホストを指定するようにしてください。以下に例を示します。

```
curl "http://192.0.2.12:80/hello?user_key=your_user_key" -H "Host: echo-api.3scale.net"
```

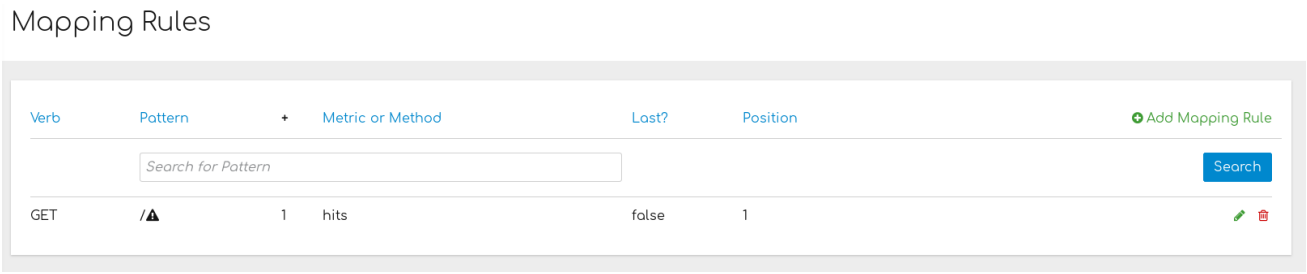
ローカルマシンにデプロイしている場合には、ドメインを localhost と指定することができます。この場合には公開ベース URL は **http://localhost:80** のようになり、以下のようにリクエストを行うことができます。

```
curl "http://localhost:80/hello?user_key=your_user_key"
```

API プロダクトが複数ある場合には、それぞれのプロダクトについて公開ベース URL を適切に設定します。APIcast はホスト名に基づきリクエストをルーティングします。

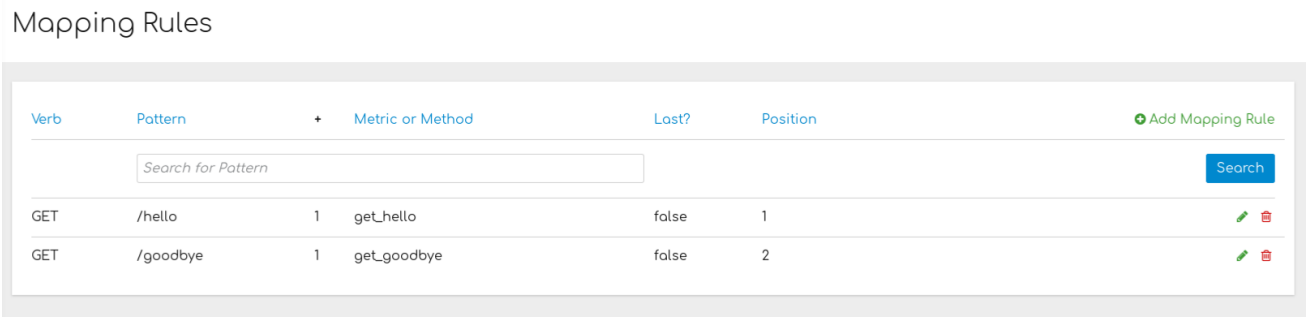
1.2. 3SCALE API の使用状況を把握するために APICAST がマッピングルールをどのように適用するか

API に対するリクエストに基づいて、マッピングルールにより API の使用状況を把握するメトリクスまたはメソッドを定義、指定します。マッピングルールの例を以下に示します。



このルールは、/ で始まるすべての **GET** リクエストがメトリクス **hits** のカウントを 1 つ増やす、という意味です。このルールは API に対するすべてのリクエストにマッチします。これは有効なマッピングルールですが、一般的すぎるので、より具体的なルールが追加された時に、2 倍にカウントされることになります。

より具体的なルールの例として、Echo API マッピングルールを以下に示します。



マッピングルールは、API プロダクトレベルおよび API バックエンドレベルで機能します。

- プロダクトレベルでのマッピングルール
  - このマッピングルールは、バックエンドレベルのマッピングルールに優先します。つまり、プロダクトレベルのマッピングルールが先に評価されます。
  - どのバックエンドがリダイレクトされたトラフィックを受信するかにかかわらず、このマッピングルールは常に評価されます。
- バックエンドレベルでのマッピングルール
  - バックエンドにマッピングルールを追加すると、それらのマッピングルールはそのバックエンドと結び付くすべてのプロダクトに追加されます。
  - このマッピングルールは、プロダクトレベルで定義されたマッピングルールの後に評価されます。
  - このマッピングルールは、マッピングルールが設定されたバックエンドにトラフィックがリダイレクトされた場合にのみ評価されます。
  - プロダクトに結び付けられるバックエンドのパスが、バックエンドの各マッピングルールの前に自動的に追加されます。

プロダクトレベルおよびバックエンドレベルのマッピングルールの例

1 つのバックエンドが設定されたプロダクトのマッピングルールの例を以下に示します。

- **Echo API** バックエンド:
  - プライベートエンドポイント: **https://echo-api.3scale.net**

    - 以下のパターンの 2 つのマッピングルールが含まれる

```
/hello
/bye
```

- Cool API プロダクト:
    - 公開エンドポイント: **https://cool.api**
    - ルーティングパス **/echo** により Echo API バックエンドを使用する
- Cool API プロダクトには、自動的に以下のパターンのマッピングルールが含まれます。

```
/echo/hello
/echo/bye
```

    - これは、公開 URL **https://cool.api/echo/hello** に送信されたリクエストが、**https://echo-api.3scale.net/hello** にリダイレクトされることを意味します。
    - 同様に、**https://cool.api/echo/bye** に送信されたリクエストは、**https://echo-api.3scale.net/bye** にリダイレクトされます。

ここで、同じ Echo API バックエンドを使用する Tools For Devs というプロダクトを追加することを考えてみます。

- Tools For Devs プロダクト:
    - 公開エンドポイント: **https://dev-tools.api**
    - ルーティングパス **/tellmeback** により Echo API バックエンドを使用する
- Tools For Devs プロダクトには、自動的に以下のパターンのマッピングルールが含まれます。

```
/tellmeback/hello
/tellmeback/bye
```

    - したがって、公開 URL **https://dev-tools.api/tellmeback/hello** に送信されたリクエストは、**https://echo-api.3scale.net/hello** にリダイレクトされます。
    - 同様に、**https://dev-tools.api/tellmeback/bye** に送信されたリクエストは、**https://echo-api.3scale.net/bye** にリダイレクトされます。
- パターン **/ping** のマッピングルールを Echo API バックエンドに追加すると、Cool API プロダクトおよび Tools For Devs プロダクトの両方に変更が加えられます。
    - Cool API には、**/echo/ping** というパターンのマッピングルールが含まれます。
    - Tools For Devs には、**/tellmeback/ping** というパターンのマッピングルールが含まれます。

マッピングルールの照合

3scale では、前方一致に基づきマッピングルールを適用します。表記法は OpenAPI および ActiveDocs の仕様に準じます。

- マッピングルールは、スラッシュ (/) で始める必要があります。

- URL の文字列 (例: **/hello**) を使用してパスの照合を行います。
  - マッピングルールを保存すると、設定した URL 文字列にリクエストが送付され、それぞれのマッピングルールで定義したメトリクスまたはメソッドが呼び出されます。
- マッピングルールでは、**/{word}?value={value}** のように、クエリー文字列またはボディーにパラメーターを含めることができます。
- APIcast は以下のようにパラメーターを取得します。
  - **GET** メソッド:クエリー文字列から。
  - **POST**、**DELETE**、または **PUT** メソッド:ボディーから。
- マッピングルールには、**/{word}** のように名前付きワイルドカードを含めることができます。このルールは、プレースホルダー **{word}** で定義する任意の文字にマッチします。たとえば、**/morning** のようなリクエストがマッピングルールにマッチします。ワイルドカードは、スラッシュとスラッシュの間またはスラッシュとピリオドの間に使用することができます。パラメーターにもワイルドカードを含めることができます。
- デフォルトでは、指定したソート順に従ってすべてのマッピングルールが上から評価されます。ルール **/v1** を追加した場合には、パスが **/v1** で始まるリクエストにマッチします (例: **/v1/word** または **/v1/sentence**)。
- パターンの最後にドル記号 (**$**) を追加して、完全一致を指定することができます。たとえば、**/v1/word** は、**/v1/word** のリクエストにだけマッチし、**/v1/word/hello** のリクエストにはマッチしません。完全一致を指定する場合には、すべてにマッチするデフォルトのマッピングルール (/) を必ず無効にする必要があります。
- 複数のマッピングルールがリクエストのパスにマッチする場合がありますが、マッチするルールがなければ、リクエストは無視され HTTP 404 ステータスコードが返されます。

マッピングルールのワークフロー

マッピングルールに関するワークフローを以下に示します。

- いつでも新たなマッピングルールを定義することができます。マッピングルールの定義 を参照してください。
- 誤って変更されないように、次回のリロード時にマッピングルールはグレーアウト表示されます。
- 既存のマッピングルールを編集するには、右側にある鉛筆アイコンをクリックして、まずそのルールを有効にする必要があります。
- ルールを削除するには、ゴミ箱アイコンをクリックします。
- **Integration > Configuration** で変更をプロモートすると、すべての変更および削除が保存されます。

他のマッピングルールの停止

1 つまたは複数のマッピングルールを処理した後、他のマッピングルールの処理を停止するには、新規マッピングルールの作成時に **Last?** を選択します。たとえば、**API Integration Settings** で以下のマッピングルールを定義し、それぞれのルールに異なるメトリクスが関連付けられていると仮定します。

```
(get) /path/to/example/search
(get) /path/to/example/{id}
```

(get) /path/to/example/search の呼び出しを行う場合、ルールがマッチした後は、APIcast は残りのマッピングルールの処理を停止し、そのメトリクスのカウントを増やすのを停止します。

1.3. 特殊な要求を持つ API を APICAST がどのように処理するか

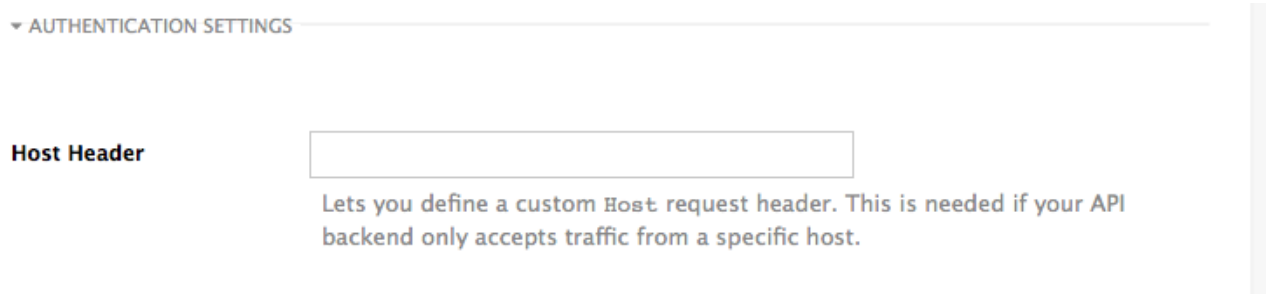
API 利用者が API を適切に呼び出すことができるように、カスタム APIcast 設定が必要となる特殊なケースがあります。

Host ヘッダー

このオプションは、**Host** ヘッダーが指定した値にマッチしない限りトラフィックを拒否する API プロダクトにのみ必要です。このような場合には、**Host** がゲートウェイの 1 つなので、ゲートウェイが API プロダクトの前にあることで問題が生じます (例: **xxx-yyy.staging.apicast.io**)。

この問題を避けるには、**AUTHENTICATION SETTINGS** (**[Your_product_name] > Integration > Settings**) の **Host Header** フィールドで、API プロダクトが想定するホストを定義します。

これにより、Hosted APIcast インスタンスはリクエストの呼び出しのホストに関する定義を書き換えます。



API バックエンドの保護

APIcast が実稼動環境で動作するようになった後、API プロダクトへの直接アクセスを、指定したシークレットトークンを持つ呼び出しだけに制限することが望まれる場合があります。そのためには、APIcast の **シークレットトークン** を設定します。シークレットトークンの設定方法については、高度な APIcast 設定 を参照してください。

プライベート API を扱う APIcast の使用

APIcast を使用して、インターネット上に公開されていない API を保護することができます。そのための条件は以下のとおりです。

- デプロイメントオプションとして、Self-managed APIcast を使用している。
- 一般のインターネットから APIcast へのアクセスが可能で、APIcast が 3scale Service Management API に発信の呼び出しを行うことができる。
- APIcast が API プロダクトにアクセスすることができる。
  この場合には、**Private Base URL** フィールドに API の内部ドメイン名または IP アドレスを設定し、他の手順は通常どおりに実施することができます。ただし、この設定により、ステージング環境のメリットを活用することができなくなります。ステージング用 APIcast インスタンスは 3scale がホストしていて、プライベート API バックエンドにはアクセスすることができないため、テストコールは成功しません。APIcast を実稼働環境にデプロイして正しく設定すれば、APIcast は想定どおりに機能します。

1.4. OPENTRACING を使用する APICAST の設定

OpenTracing は API の仕様で、マイクロサービスのプロファイリングおよびモニターリングに使用されるメソッドです。バージョン 3.3 以降の APIcast には、OpenTracing ライブラリーおよび Jaeger Tracer ライブラリー が含まれています。

前提条件

- それぞれの外部リクエストに、固有のリクエスト ID がアタッチされている。通常、これには HTTP ヘッダーが使用されます。
- それぞれのサービスがリクエスト ID を他のサービスに転送する。
- それぞれのサービスがリクエスト ID をログに出力する。
- それぞれのサービスがリクエストの開始/終了時刻等の補足情報を記録する。
- ログが集約され、HTTP リクエスト ID を使用して解析する手段を提供する。

手順

1. **OPENTRACING_TRACER** 環境変数が **jaeger** に設定されるようにします。この変数が空欄の場合には、OpenTracing は無効になります。
2. **OPENTRACING_CONFIG** 環境変数を設定して、使用するトレーサーのデフォルト設定ファイルを指定します。例として、以下の **jaeger.example.json** ファイルを参照してください。
3. オプション:実際の OpenTracing 設定に応じて、**OPENTRACING_HEADER_FORWARD** 環境変数を設定します。

検証

インテグレーションが適切に機能しているかどうかをテストするには、トレースが Jaeger トレースインターフェイスで報告されるかどうかを確認します。

関連情報

- APIcast の環境変数
- OpenTracing および Jaeger インテグレーションのアップストリームプロジェクト

1.5. OPENSHIFT インスタンスへの JAEGER のインストール

3scale API プロバイダーは、Jaeger と共に OpenTracing を使用して、API への呼び出しのトレースおよびトラブルシューティングを行うことができます。そのためには、3scale を実行中の OpenShift インスタンスに Jaeger をインストールします。

警告

Jaeger はサードパーティーコンポーネントであり、APIcast と共に使用する場合を除き 3scale はサポートを提供しません。以下の手順は参考例としてのみ提供され、実稼働環境での使用には適しません。

手順

1. 現在の namespace に Jaeger オールインワンテンプレートをインストールします。

```
oc process -f https://raw.githubusercontent.com/jaegertracing/jaeger-openshift/master/all-in-one/jaeger-all-in-one-template.yml | oc create -f -
```

2. 以下の内容で Jaeger 設定ファイル **jaeger_config.json** を作成します。

```
{
   "service_name": "apicast",
   "disabled": false,
   "sampler": {
     "type": "const",
     "param": 1
   },
   "reporter": {
     "queueSize": 100,
     "bufferFlushInterval": 10,
     "logSpans": false,
     "localAgentHostPort": "jaeger-agent:6831"
   },
   "headers": {
     "jaegerDebugHeader": "debug-id",
     "jaegerBaggageHeader": "baggage",
     "TraceContextHeaderName": "uber-trace-id",
     "traceBaggageHeaderPrefix": "testctx-"
   },
   "baggage_restrictions": {
       "denyBaggageOnInitializationFailure": false,
       "hostPort": "127.0.0.1:5778",
       "refreshInterval": 60
   }
}
```

- **sampler** 定数を 1 に設定すると、すべてのリクエストがサンプリングされます。
- **reporter** の場所およびキューサイズは必須です。
- リクエストを追跡するためには、**headers** セクションの **TraceContextHeaderName** エントリーは必須です。

3. Jaeger 設定ファイルから ConfigMap を作成し、それを APIcast にマウントします。

```
oc create configmap jaeger-config --from-file=jaeger_config.json
oc set volumes dc/apicast --add -m /tmp/jaeger/ --configmap-name jaeger-config
```

4. 前のステップで追加した設定で、OpenTracing および Jaeger を有効にします。

```
oc set env deploymentConfig/apicast OPENTRACING_TRACER=jaeger
OPENTRACING_CONFIG=/tmp/jaeger/jaeger_config.json
```

5. Jaeger インターフェイスが動作している URL を確認します。

```
oc get route
(…) jaeger-query-myproject.127.0.0.1.nip.io
```

6. 前のステップで確認した URL から Jaeger インターフェイスを開きます。Openshift のヘルスチェックから読み込まれているデータが表示されます。

7. リクエストのトレースをすべて表示できるように、OpenTracing および Jaeger のサポートをバックエンド API に追加します。この操作は、使用されるフレームワークおよび言語に応じてバックエンドごとに異なります。例として Using OpenTracing with Jaeger to collect Application Metrics in Kubernetes を参照してください。

関連情報

- Jaeger on OpenShift Development setup
- Jaeger on OpenShift Production setup
- Distributed tracing on OpenShift Service Mesh

第2章 DOCKER コンテナー環境の操作

2.1. DOCKER コンテナー環境上の APICAST に関するトラブルシューティング

本セクションでは、Docker コンテナー環境上で APIcast を使用する際に直面する、もっとも典型的な問題について説明します。

2.1.1. Cannot connect to the Docker daemon エラー

docker:Cannot connect to the Docker daemon.Is the docker daemon running on this host?というエラーメッセージが表示された場合には、Docker サービスが起動していない可能性があります。**sudo systemctl status docker.service** コマンドを実行して、Docker デーモンのステータスを確認することができます。

RHEL で Docker コンテナー環境の操作を行う場合、デフォルトでは root 権限が必要なので、このコマンドは必ず **root** ユーザーとして実行してください。詳細については、このドキュメント を参照してください。

2.1.2. 基本的な Docker コマンドラインインターフェイスコマンド

デタッチモードでコンテナーを起動し (**-d** オプション)、実行中の APIcast インスタンスのログを確認する場合には、**log** コマンド (**sudo docker logs <container>**) を使用することができます。ここで、**<container>** はコンテナー名 (上記の例では**apicast**) またはコンテナー ID です。**sudo docker ps** コマンドを使用して、実行中のコンテナーならびにその ID および名前を一覧表示することができます。

コンテナーを停止するには、**sudo docker stop <container>** コマンドを実行します。**sudo docker rm <container>** コマンドを実行して、コンテナーを削除することもできます。

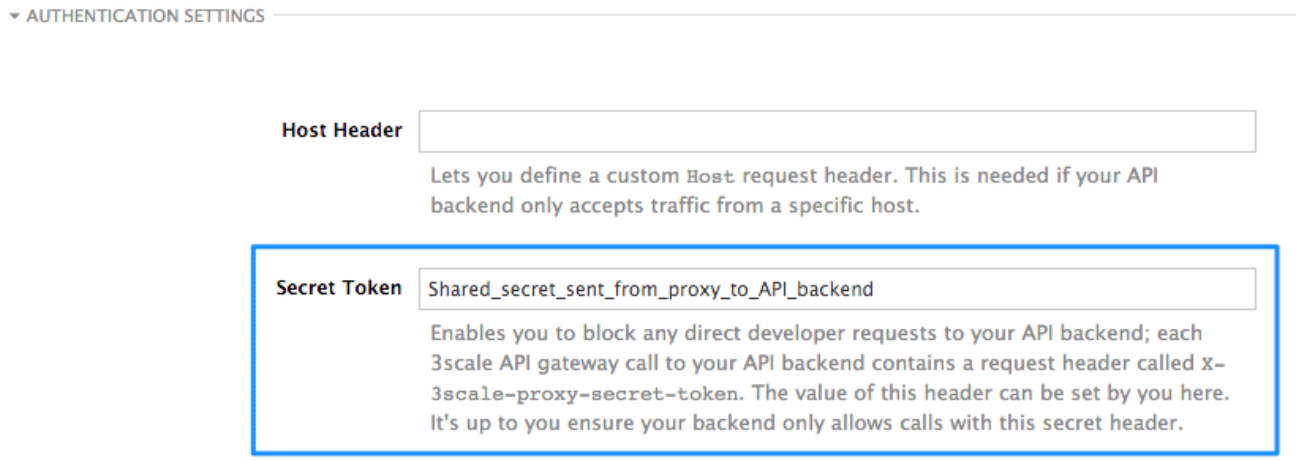
使用できるコマンドの詳細は、Docker コマンドのリファレンス を参照してください。

第3章 高度な APICAST 設定

本セクションでは、ステージング環境における 3scale の API ゲートウェイの高度な設定オプションについて説明します。

3.1. シークレットトークンの定義

セキュリティー上の理由から、3scale ゲートウェイから API バックエンドに送信されるすべてのリクエストには、**X-3scale-proxy-secret-token** というヘッダーが含まれます。Integration ページの **AUTHENTICATION SETTINGS** で、このヘッダーの値を設定することができます。



設定したシークレットトークンは、プロキシーと API 間の共有シークレットとして機能し、ゲートウェイから送信されていない API リクエストを拒否したい場合には、すべてブロックすることができます。これにより、サンドボックスゲートウェイを使用してトラフィック管理ポリシーをセットアップする際に、公開エンドポイントを保護するための新たなセキュリティーレイヤーを追加することができます。

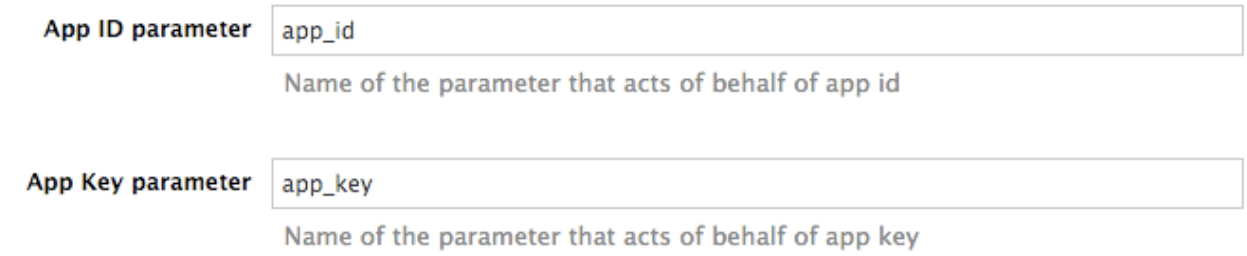
ゲートウェイが機能するためには、API バックエンドには公開されている解決可能なドメインが必要です。したがって、API バックエンドを知っていれば、誰でもクレデンシャルの確認を迂回することができます。ステージング環境の API ゲートウェイは実稼働環境での使用を想定していないので、このことが問題になることはありません。ただし、アクセスを防げるようにしておいた方が望ましいと言えます。

3.2. クレデンシャル

3scale における API クレデンシャルは、**user_key** または **app_id/app_key** のどちらかです (使用する認証モードによります)。ステージング環境の API ゲートウェイでは OpenID Connect が有効です。ただし、Integration ページでテストすることはできません。

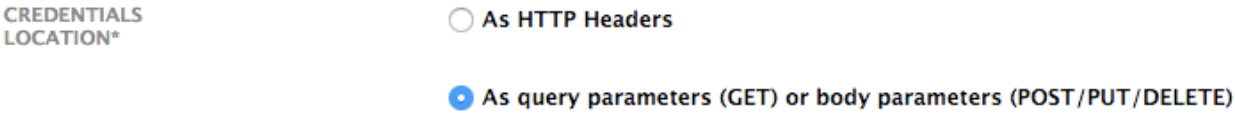
なお、API で異なるクレデンシャル名を使用することが望ましい場合もあります。この場合、API キーモードを使用していれば **user_key** にカスタムな名前を設定する必要があります。

あるいは、以下のように **app_id** および **app_key** を設定します。



たとえば、**app_id** を **key** に変えることが API にとってより適切であれば、名前を変更することができます。ゲートウェイは名前 **key** を取得し、3scale バックエンドに対して承認呼び出しを行う前に **app_id** に変換します。新しいクレデンシャル名には、英数字を使用しなければならない点に注意してください。

API がクレデンシャルを渡す際に、クエリー文字列 (GET 以外ではボディー) またはヘッダーのどちらを使用するかを定義することができます。



注記

クレデンシャルを抽出する際に、APIcast はヘッダー名を正規化します。つまり、大文字と小文字の区別をなくし、アンダースコアとハイフンを等価に扱います。たとえば、アプリケーションキーのパラメーターを **App_Key** と設定した場合には、**app-key** 等の他の値も有効なアプリケーションキーヘッダーとして受け入れられます。

3.3. エラーメッセージの設定

本セクションでは、APIcast のエラーメッセージを設定する方法について説明します。

プロキシーとして、3scale APIcast はリクエストを以下のように管理します。

- エラーがなければ、APIcast はクライアントからのリクエストを API バックエンドサーバーに渡し、API のレスポンスを変更せずにクライアントに返します。レスポンスを変更する場合には、Header Modification ポリシー を使用することができます。
- API のレスポンスが **404 Not Found** または **400 Bad Request** 等のエラーメッセージの場合には、APIcast はそのメッセージをクライアントに返します。ただし、APIcast が **Authentication missing** 等の他のエラーを検出した場合には、APIcast はエラーメッセージを送信してリクエストを終了させます。

したがって、APIcast がこれらのエラーメッセージを返すように設定することができます。

- Authentication failed:このエラーは、クレデンシャルが偽造されているか、アプリケーションが一時的に保留されているかのいずれかにより、API リクエストに有効なクレデンシャルが含まれていないことを意味します。また、このエラーはメトリクスが無効 (その値が **0**) な場合に生成されます。
- Authentication missing:API リクエストにクレデンシャルが含まれていない場合には、必ずこのエラーが生成されます。ユーザーが API リクエストにクレデンシャルを追加しなかった場合に起こります。

- No match:このエラーは、リクエストがどのマッピングルールにもマッチしなかったため、メトリクスが更新されないことを意味します。この状況は必ずしもエラーとは限りませんが、ユーザーが無効なパスを試みているか、マッピングルールが正当なケースに対応していないかのいずれかを意味します。

- Usage limit exceeded:このエラーは、リクエストしたエンドポイントに関して、クライアントが流量制御の上限に達したことを意味します。リクエストが複数のマッピングルールにマッチする場合には、クライアントは複数の流量制御の上限に達する可能性があります。

エラーを設定するには、以下の手順に従います。

1. **[Your_product_name] > Integration > Settings**から操作を行います。
2. **GATEWAY RESPONSE** セクションで設定するエラーのタイプを選択します。
3. 以下のフィールドの値を指定します。
    - Response Code:3 桁の HTTP レスポンスコード
    - Content-type:**Content-Type** ヘッダーの値
    - Response Body:レスポンスメッセージのボディーの値
4. 変更を保存するには、**Update Product** をクリックします。

3.4. 設定履歴

Promote v.[n] to Staging APIcastをクリックするたびに (**[n]** はバージョン番号を表します)、その時点での設定が JSON ファイルに保存されます。ステージング環境のゲートウェイは、新規リクエストのたびに最新の設定を取得します。それぞれの環境 (ステージングまたは実稼働環境) について、それまでの設定ファイルの履歴をすべて確認することができます。

1. **[Your_product_name] > Integration > Configuration**から操作を行います。
2. 次に挙げる該当する環境の横にある **Configuration history** のリンクをクリックします。**Staging APIcast** または **Production APIcast**

以前のバージョンに自動的にロールバックすることはできない点に注意してください。その代わりに、対応する JSON ファイルによりすべての設定バージョンの履歴にアクセスすることができます。これらのファイルを使用して、それぞれの時点でデプロイした設定を確認することができます。必要であれば、どのデプロイメントでも手動で作成し直すことができます。

3.5. デバッグ

ゲートウェイ設定のセットアップは簡単ですが、それでもエラーが発生する可能性があります。そのような状況のために、ゲートウェイはエラーを追跡するのに有用なデバッグ情報を返すことができます。

APIcast からデバッグ情報を取得するには、次に挙げるヘッダーを API リクエストに追加する必要があります。**X-3scale-debug: {SERVICE_TOKEN}**。この時、アクセスする API サービスに対応するサービストークンを指定します。

ヘッダーが検出されサービストークンが有効であれば、ゲートウェイはレスポンスヘッダーに以下の情報を追加します。

```
X-3scale-matched-rules: /v1/word/{word}.json, /v1
X-3scale-credentials: app_key=APP_KEY&app_id=APP_ID
X-3scale-usage: usage%5Bversion_1%5D=1&usage%5Bword%5D=1
```

X-3scale-matched-rules には、リクエストにマッチしているマッピングルールのコンマ区切りリストが表示されます。

ヘッダー **X-3scale-credentials** は、3scale バックエンドに渡されたクレデンシャルを返します。

X-3scale-usage には、3scale バックエンドに報告された使用状況が表示されます。**usage%5Bversion_1%5D=1&usage%5Bword%5D=1** は **usage[version_1]=1&usage[word]=1** を URL エンコードしたもので、API リクエストによりメソッド (メトリクス) **version_1** および **word** のカウントがそれぞれ 1 増えたこと意味します。

3.6. パスルーティング

APIcast は、3scale アカウント (または **APICAST_SERVICES_LIST** 環境変数が設定されている場合にはサービスのサブセット) で設定されているすべての API サービスを処理します。通常、APIcast はリクエストのホスト名に基づいて API リクエストを適切な API サービスにルーティングします。そのために、**公開ベース URL** との照合を行います。最初にマッチしたサービスが承認に使用されます。

パスルーティング機能により、複数のサービスで同じ **公開ベース URL** を使用することができ、リクエストはリクエストのパスによりルーティングされます。この機能を有効にするには、**APICAST_PATH_ROUTING** 環境変数を **true** または **1** に設定します。有効にすると、APIcast はホスト名とパスの両方に基づいて受信したリクエストをサービスにマッピングします。

同じ **公開ベース URL** を使用して 1 つのゲートウェイを通じて異なるドメインでホストされる複数のバックエンドサービスを公開する場合に、この機能を使用することができます。そのためには、API バックエンド (つまり **プライベートベース URL**) ごとに複数の API サービスを設定し、パスルーティング機能を有効にします。

たとえば、3 つのサービスが以下のように設定されている場合、

- サービス A 公開ベース URL: **api.example.com** マッピングルール: **/a**
- サービス B 公開ベース URL: **api2.example.com** マッピングルール: **/b**
- サービス C 公開ベース URL: **api.example.com** マッピングルール: **/c**

パスルーティングが **無効** な場合には (**APICAST_PATH_ROUTING=false**)、**api.example.com** に対するすべての呼び出しはサービス A との照合を試みます。したがって、呼び出し **api.example.com/c** および **api.example.com/b** はNo Mapping Rule matchedエラーと共に失敗します。

パスルーティングが **有効** な場合には (**APICAST_PATH_ROUTING=true**)、呼び出しはホストおよびパスの両方に照合されます。したがって、

- **api.example.com/a** はサービス A にルーティングされます。
- **api.example.com/c** はサービス C にルーティングされます。
- **api.example.com/b** は No Mapping Rule matched エラーと共に失敗します。つまり、 **公開ベース URL** がマッチしないので、サービス B とはマッチしません。

パスルーティングを使用する場合には、同じ **公開ベース URL** を使用する複数のサービスにおいて、マッピングルールが競合しないようにする必要があります。つまり、メソッドとパスパターンの組み合わせは、それぞれ 1 つのサービスでしか使用することはできません。

第4章 APICAST ポリシー

APIcast ポリシーとは、APIcast の動作を変更する機能の単位です。ポリシーを有効、無効、および設定して、APIcast 動作の変更を制御することができます。ポリシーを使用して、デフォルトの APIcast デプロイメントでは利用することのできない機能を追加します。自分専用のポリシーを作成することや、Red Hat 3scale の提供する 標準ポリシー を使用することができます。

以下のトピックでは、標準 APIcast ポリシー、ポリシーチェーンの作成、およびカスタム APIcast ポリシーの作成について説明します。

4.1. 3SCALE APICAST のデフォルト動作を変更するための標準ポリシー

3scale では、組み込みの標準ポリシーが提供されます。これらのポリシーは、APIcast がリクエストとレスポンスをどのように処理するかを変更する機能の単位です。ポリシーを有効、無効、または設定して、APIcast を変更する方法を制御することができます。

詳細は、3scale 管理ポータルでのポリシーの有効化を参照してください。3scale では、以下の標準ポリシーが利用可能です。

- 3scale Auth Caching
- 3scale Batcher
- 3scale Referrer
- Anonymous Access
- Camel Service
- Conditional ポリシー
- Content Caching
- CORS Request Handling
- Custom Metrics
- Echo
- Edge Limiting
- Header Modification
- HTTP Status Code Overwrite
- HTTP2 Endpoint
- IP Check
- JWT Claim Check
- Liquid Context Debug
- Logging
- Maintenance Mode

- NGINX Filter
- OAuth 2.0 Mutual TLS Client Authentication
- OAuth 2.0 Token Introspection
- On Fail
- Proxy Service
- Rate Limit Headers
- Response Request Content Limits
- Retry
- RH-SSO/Keycloak Role Check
- Routing
- SOAP
- TLS Client Certificate Validation
- TLS Termination
- Upstream
- Upstream Connection
- Upstream Mutual TLS
- URL Rewriting
- URL Rewriting With Captures
- Websocket

4.1.1. 3scale 管理ポータルでのポリシーの有効化

管理ポータルでは、3scale API プロダクトごとに 1 つ以上のポリシーを有効にすることができます。

前提条件

- 3scale API プロダクト

手順

1. 3scale にログインします。
2. 管理ポータルでのダッシュボードで、ポリシーを有効にする API プロダクトを選択します。
3. **[your_product_name]** から **Integration > Policies** の順に移動します。
4. **POLICIES** セクションで **Add policy** をクリックします。
5. 追加するポリシーを選択し、必須フィールドに値を入力します。

6. **Update Policy Chain** をクリックし、ポリシーチェーンを保存します。

4.1.2. 3scale Auth Caching

3scale Auth Caching ポリシーは、APIcast に送信された認証呼び出しをキャシュします。動作モードを選択して、キャッシュ操作を設定することができます。

3scale Auth Caching では、以下のモードを使用することができます。

1.strict: 承認された呼び出しだけをキャッシュします。

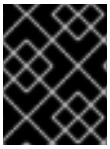
strict モードでは、承認された呼び出しだけがキャッシュされます。ポリシーを strict モードで実行中に呼び出しが失敗または拒否されると、ポリシーはキャッシュエントリーを無効にします。バックエンドにアクセスできなくなると、キャッシュステータスにかかわらず、キャッシュされたすべての呼び出しが拒否されます。

2.resilient: バックエンドの機能が停止している場合には、最後のリクエストに基づいて承認します。

resilient モードでは、承認された呼び出しおよび拒否された呼び出しの両方がキャッシュされます。ポリシーを resilient モードで実行中は、呼び出しに失敗しても既存のキャッシュエントリーは無効になりません。バックエンドにアクセスできなくなると、キャッシュにヒットする呼び出しは、キャッシュステータスに応じて承認/拒否の状態を維持します。

3.allow: バックエンドの機能が停止している場合には、過去に拒否されていない限りすべてを許可します。

allow モードでは、承認された呼び出しおよび拒否された呼び出しの両方がキャッシュされます。ポリシーを allow モードで実行中は、キャッシュされた呼び出しはキャッシュステータスに応じて拒否/許可の状態を維持します。ただし、新規の呼び出しは承認された呼び出しとしてキャッシュされます。



重要

allow モードで運用した場合には、セキュリティーの低下が懸念されます。これらの懸念に念頭に置き、注意した上で allow モードを使用してください。

4.none: キャッシュを無効にします。

none モードでは、キャッシュが無効になります。ポリシーを有効にしたままキャッシュの使用を止める場合に、このモードが役立ちます。

設定プロパティー

プロパティー	説明	値	必須/任意
caching_type	**caching_type** プロパティーにより、キャッシュの動作モードを定義することができます。	データタイプ: 列挙文字列 [resilient, strict, allow, none]	必須

ポリシーオブジェクトの例

```
{
  "name": "caching",
```

```
    "version": "builtin",
    "configuration": {
      "caching_type": "allow"
    }
  }
```

ポリシーの設定方法に関する情報は、このドキュメントの 3scale でのポリシーチェーンの作成 セクションを参照してください。

4.1.3. 3scale Batcher

標準の APIcast 承認メカニズムでは、APIcast が API リクエストを受け取るたびに、3scale バックエンド (Service Management API) に対して 1 回呼び出しが行われます。3scale Batcher ポリシーは、この標準メカニズムの代替手段を提供します。

重要

このポリシーを使用するには、ポリシーチェーンの **3scale APIcast** ポリシーの前に **3scale Batcher** を配置する必要があります。

3scale Batcher ポリシーを使用すると、承認ステータスがキャッシュされ使用状況レポートがバッチ処理されます。これにより、3scale バックエンドへのリクエスト数が大幅に削減されます。3scale Batcher ポリシーにより、レイテンシーを短縮しスループットを増やすことで、API キャストのパフォーマンスを改善できます。

3scale Batcher ポリシーが有効な場合には、APIcast は以下のフローを使用して承認を行います。

1. それぞれのリクエストにおいて、ポリシーはクレデンシャルがキャッシュされているかどうかを確認します。
    - クレデンシャルがキャッシュされている場合には、ポリシーは 3scale バックエンドに呼び出しを行う代わりに、キャッシュされた承認ステータスを使用します。
    - クレデンシャルがキャッシュされていない場合には、ポリシーはバックエンドに呼び出しを行い、残存期間 (TTL) を設定して承認ステータスをキャッシュします。
2. リクエストに対応する使用状況を直ちに 3scale バックエンドに報告する代わりに、ポリシーは使用状況のカウンターを累積して、バックエンドにバッチで報告します。累積した使用状況のカウンターは、独立したスレッドにより 1 回の呼び出しで 3scale バックエンドに報告されます (報告の頻度は設定が可能です)。

3scale Batcher ポリシーではスループットが向上しますが、精度は低下します。使用上限および現在の使用状況は 3scale に保存され、APIcast は 3scale バックエンドに呼び出しを行う時にのみ正しい承認ステータスを取得することができます。3scale Batcher ポリシーが有効な場合には、APIcast が 3scale に呼び出しを送信しない期間があります。この期間中に、呼び出しを行うアプリケーションが定義された限度を超える可能性があります。

流量制御の精度よりもスループットが重要な場合には、高負荷の API に対してこのポリシーを使用します。報告頻度および承認の TTL が流量制御の期間よりはるかに短い場合には、3scale Batcher ポリシーにより優れた精度が得られます。たとえば、流量制御が 1 日あたりで、報告頻度および承認の TTL が数分に設定されている場合などです。

3scale Batcher ポリシーでは、以下の設定設定がサポートされます。

- **auths_ttl**:承認キャッシュの有効期限が切れる TTL を秒単位で設定します。

- 現在の呼び出しの承認がキャッシュされている場合には、APIcast はキャッシュされた値を使用します。**auths_ttl** パラメーターで設定した時間が経過すると、APIcast はキャッシュを削除し、3scale バックエンドに呼び出しを行い承認のステータスを取得します。
- **auths_ttl** パラメーターを **0** 以外の値に設定します。**auths_ttl** の値を **0** に設定すると、リクエストの初回キャッシュ時に承認カウンターが更新されます。その結果、流量制御が有効になりません。

- **batch_report_seconds**: APIcast が 3scale バックエンドにバッチレポートを送信する頻度を設定します。デフォルト値は **10** 秒です。

4.1.4. 3scale Referrer

3scale Referrer ポリシーを使用すると、**参照元フィルター** 機能が有効になります。サービスポリシーチェーンでこのポリシーが有効な場合には、APIcast は上流への **AuthRep** コールとして 3scale Referrer ポリシーの値を**Service Management API**に送信します。3scale Referrer ポリシーの値は、呼び出しの **referrer** パラメーターで送信されます。

参照元フィルター機能の仕組みの詳細については、**認証パターン**の 参照元フィルター セクションを参照してください。

4.1.5. Anonymous Access

Anonymous Access ポリシーでは、認証をせずにサービスが公開されます。このポリシーは、認証パラメーターを送信するように変更することのできないレガシーアプリケーション等の場合に有用です。Anonymous Access ポリシーは、API キーおよびアプリケーション ID/アプリケーションキーによる認証オプションだけを使用するサービスをサポートします。クレデンシャルをまったく持たない API リクエストに対してこのポリシーを有効にした場合には、APIcast はポリシーで設定されたデフォルトのクレデンシャルを使用して呼び出しを承認します。API の呼び出しが承認されるためには、クレデンシャルが設定されたアプリケーションが存在しアクティブでなければなりません。

アプリケーションプランを使用して、デフォルトのクレデンシャルに使用されるアプリケーションに流量制御を設定することができます。

注記

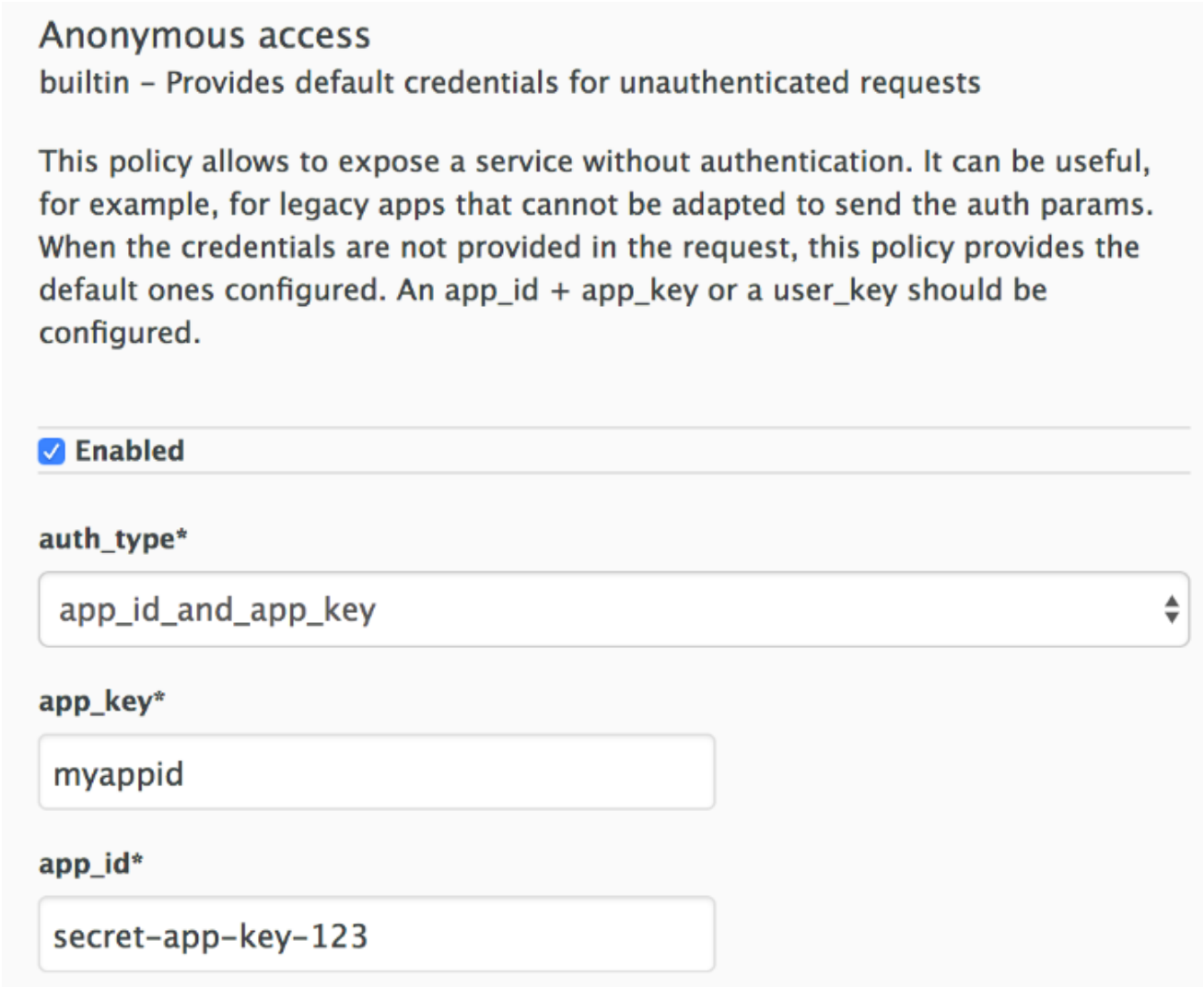
APIcast ポリシーおよび Anonymous Access ポリシーをポリシーチェーンで併用する場合には、APIcast ポリシーの前に Anonymous Access ポリシーを設定する必要があります。

ポリシーに必要な設定プロパティーを以下に示します。

- 認証タイプ以下に示す選択肢のいずれかの値を選択し、プロパティーが API に設定された認証オプションに対応している状態にします。
  - **app_id_and_app_key**:アプリケーション ID/アプリケーションキーによる認証オプション用
  - **user_key**: API キーによる認証オプション用
- **app_id** (**app_id_and_app_key** 認証タイプにのみ有効): API の呼び出しにクレデンシャルが提供されていない場合に、承認に使用するアプリケーションのアプリケーション ID。
- **app_key** (**app_id_and_app_key** 認証タイプにのみ有効): API の呼び出しにクレデンシャルが提供されていない場合に、承認に使用するアプリケーションのアプリケーションキー。

- **user_key** (**user_key** 認証タイプにのみ有効): API の呼び出しにクレデンシャルが提供されていない場合に、承認に使用するアプリケーションの API キー。

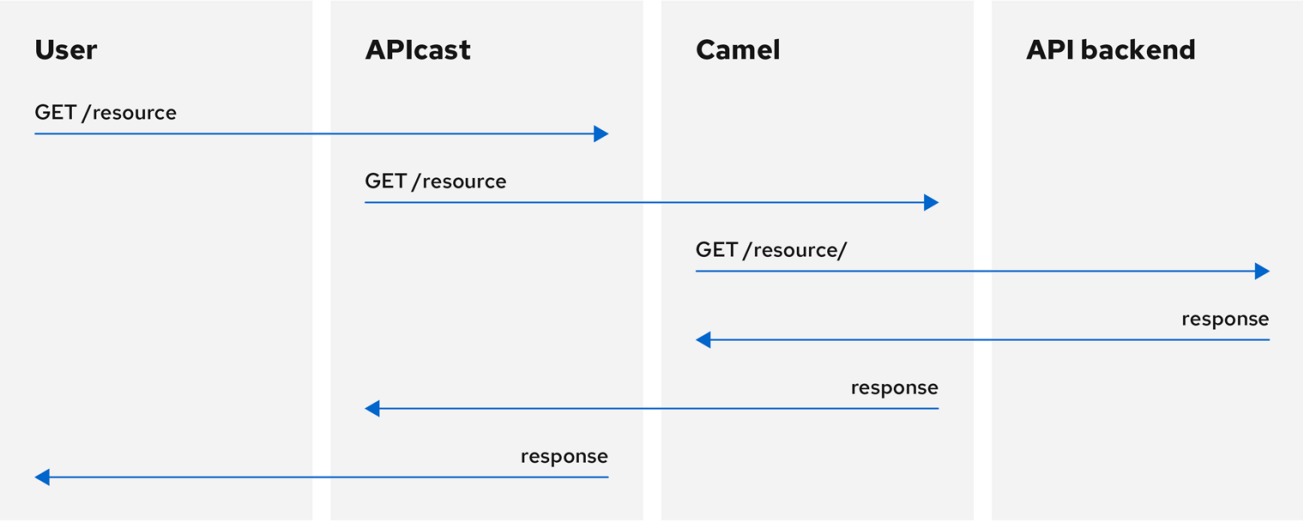
図4.1 Anonymous Access ポリシー



4.1.6. Camel Service

Camel Service ポリシーを使用して、定義した Apache Camel プロキシーを使用して 3scale のトラフィックを送信する **HTTP プロキシー** を定義することができます。この場合、Camel はリバース HTTP プロキシーとして機能し、APIcast は Camel にトラフィックを送信し、Camel がそのトラフィックを API バックエンドに送信します。

トラフィックフローの例を以下に示します。



3scale バックエンドに送信された APIcast トラフィックは、すべて Camel プロキシーを使用しません。このポリシーは、Camel プロキシーおよび APIcast と API バックエンド間の通信にのみ適用されます。

すべてのトラフィックをプロキシー経由で送信するには、**HTTP_PROXY** 環境変数を使用する必要があります。

注記

- Camel Service ポリシーはすべての負荷分散ポリシーを無効にし、トラフィックは Camel プロキシーに送信されます。
- **HTTP_PROXY**、**HTTPS_PROXY**、または **ALL_PROXY** パラメーターが定義されている場合には、このポリシーによりこれらのパラメーターの値が上書きされます。
- プロキシー接続では、認証はサポートされません。認証には Header Modification ポリシーを使用します。

設定

ポリシーチェーンの設定例を以下に示します。

```
"policy_chain": [
  {
    "name": "apicast.policy.apicast"
  },
  {
    "name": "apicast.policy.camel",
    "configuration": {
        "all_proxy": "http://192.168.15.103:8080/",
        "http_proxy": "http://192.168.15.103:8080/",
        "https_proxy": "http://192.168.15.103:8443/"
    }
  }
]
```

http_proxy または **https_proxy** が定義されていない場合は、**all_proxy** の値が使用されます。

ユースケースの例

Camel Service ポリシーは、Apache Camel を使用して、3scale でより粒度の細かいポリシーおよび変換を適用できるように作られています。このポリシーは、HTTP および HTTPS を使用した Apache Camel とのインテグレーションをサポートします。詳細は、*6章Fuse のポリシーエクステンションを使用した 3scale メッセージコンテンツの変換*を参照してください。

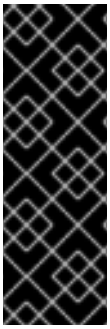
一般的な HTTP プロキシーポリシー使用の詳細は、「Proxy Service」を参照してください。

プロジェクトの例

GitHub の Camel proxy policy で **camel-netty-proxy** の例を参照してください。このプロジェクト例には、API バックエンドからのレスポンスボディーを大文字に変換する HTTP プロキシーが示されています。

4.1.7. Conditional ポリシー

Conditional ポリシーは、ポリシーチェーンが含まれるため、他の APIcast ポリシーとは異なります。このポリシーは、**access**、**rewrite**、**log** など、各 **nginx フェーズ** で評価される条件を定義します。条件が true の場合には、Conditional ポリシーは、チェーンに含まれるポリシーごとにフェーズを実行します。



重要

APIcast の Conditional ポリシーはテクノロジープレビュー機能です。テクノロジープレビューの機能は、Red Hat の実稼働環境のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

以下の例では、Conditional ポリシーが **the request method is POST** の条件を定義していると仮定しています。

```
APIcast --> Caching --> Conditional --> Upstream

                    |
                    v

                Headers

                    |
                    v

              URL Rewriting
```

今回の例では、リクエストが **POST** の場合に、各フェーズの実行順序は以下のようになります。

1. APIcast
2. Caching
3. Headers

4. URL Rewriting
5. Upstream

リクエストが **POST** でない場合には、各フェーズの実行順序は以下のようになります。

1. APIcast
2. Caching
3. Upstream

条件

Conditional ポリシーチェーンでポリシーを実行するかどうかを判断する条件は、**JSON** を使用して表現でき、Liquid テンプレートを使用します。

以下の例では、リクエストパスが **/example_path** かどうかを確認します。

```
{
  "left": "{{ uri }}",
  "left_type": "liquid",
  "op": "==",
  "right": "/example_path",
  "right_type": "plain"
}
```

左側のオペランドと右のオペランドの両方を Liquid または plain 文字列として評価できます。デフォルトは、Plain 文字列です。

and または **or** を使用した操作を組み合わせることができます。この設定では、以前の例の内容に加え、**Backend** ヘッダーの値を確認します。

```
{
  "operations": [
    {
      "left": "{{ uri }}",
      "left_type": "liquid",
      "op": "==",
      "right": "/example_path",
      "right_type": "plain"
    },
    {
      "left": "{{ headers['Backend'] }}",
      "left_type": "liquid",
      "op": "==",
      "right": "test_upstream",
      "right_type": "plain"
    }
  ],
  "combine_op": "and"
}
```

詳細は、policy config schema を参照してください。

Liquid でサポートされている変数

- uri
- host
- remote_addr
- headers['Some-Header']

変数の更新リストは ngx_variable.lua を参照してください。

以下の例では、リクエストの **Backend** ヘッダーが **staging** の場合に、アップストリームポリシーを実行します。

```
{
  "name":"conditional",
  "version":"builtin",
  "configuration":{
    "condition":{
      "operations":[
        {
          "left":"{{ headers['Backend'] }}",
          "left_type":"liquid",
          "op":"==",
          "right":"staging"
        }
      ]
    },
    "policy_chain":[
      {
        "name":"upstream",
        "version": "builtin",
        "configuration":{
          "rules":[
            {
              "regex":"/",
              "url":"http://my_staging_environment"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.8. Content Caching

Content Caching ポリシーにより、カスタムの条件に基づいてキャシングを有効および無効にすることができます。アップストリームレスポンスをポリシーに使用することができないクライアントリクエストにのみ、これらの条件を適用することができます。

Content Caching ポリシーがポリシーチェーンにある場合に、APIcast は要求をアップストリームに送信する前に **HEAD** 要求を **GET** 要求に変換します。この変換を希望しない場合は、ポリシーチェーンに Content Caching ポリシーを追加しないでください。

cache-control ヘッダーが送付される場合は、APIcast が設定するタイムアウトに優先します。

以下の設定例の場合、メソッドが GET の場合にレスポンスがキャッシュされます。

設定例

```
{
  "name": "apicast.policy.content_caching",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "cache": true,
        "header": "X-Cache-Status-POLICY",
        "condition": {
          "combine_op": "and",
          "operations": [
            {
              "left": "{{method}}",
              "left_type": "liquid",
              "op": "==",
              "right": "GET"
            }
          ]
        }
      }
    ]
  }
}
```

サポートされる設定

- 次に挙げるメソッドのいずれかについて、Content Caching ポリシーを disabled に設定します。**POST**、**PUT**、または **DELETE**
- あるルールがマッチし、そのルールがキャッシュを有効にする場合、実行は停止しキャッシングは無効になりません。ここでは、優先度の順に設定することが重要です。

アップストリームレスポンスヘッダー

NGINX の **proxy_cache_valid** ディレクティブ情報は、**APICAST_CACHE_STATUS_CODES** および **APICAST_CACHE_MAX_TIME** ではグローバルにしか設定することができません。タイムアウトに関してアップストリームが異なる動作を要求する場合は、**Cache-Control** ヘッダーを使用します。

4.1.9. CORS Request Handling

Cross Origin Resource Sharing (CORS) Request Handling ポリシーを使用すると、以下の項目を指定することができるので CORS の動作の制御が可能です。

- 許可されるヘッダー
- 許可されるメソッド
- 許可されるオリジンヘッダー
- 許可されるクレデンシャル
- 最大エージ

CORS Request Handling ポリシーにより、指定されていない CORS リクエストはすべてブロックされます。

注記

APIcast ポリシーおよび CORS Request Handling ポリシーをポリシーチェーンで併用する場合には、APIcast ポリシーの前に CORS Request Handling ポリシーを設定する必要があります。

設定プロパティー

プロパティー	説明	値	必須/任意
allow_headers	**allow_headers** プロパティーでは、APIcast が許可する CORS ヘッダーを配列として指定することができます。	データタイプ: 文字列の配列 (CORS ヘッダーでなければなりません)	任意
allow_methods	**allow_methods** プロパティーでは、APIcast が許可する CORS メソッドを配列として指定することができます。	データタイプ: 列挙文字列の配列 [GET, HEAD, POST, PUT, DELETE, PATCH, OPTIONS, TRACE, CONNECT]	任意
allow_origin	**allow_origin** プロパティーでは、APIcast が許可するオリジンのドメインを指定することができます。	データタイプ: 文字列	任意
allow_credentials	**allow_credentials** プロパティーでは、APIcast がクレデンシャルの設定された CORS リクエストを許可するかどうかを指定することができます。	データタイプ: ブール値	いいえ
max_age	**max_age** プロパティーでは、プリフライトリクエストの結果をキャッシュできる期間を設定できます。	データタイプ: 整数	いいえ

ポリシーオブジェクトの例

```
{
  "name": "cors",
 "version": "builtin",
 "configuration": {
   "allow_headers": [
```

```
      "App-Id", "App-Key",
      "Content-Type", "Accept"
    ],
    "allow_credentials": true,
    "allow_methods": [
      "GET", "POST"
    ],
    "allow_origin": "https://example.com",
    "max_age" : 200
   }
}
```

ポリシーの設定方法に関する情報は、3scale 管理ポータルでのポリシーチェーンの変更を参照してください。

4.1.10. Custom Metrics

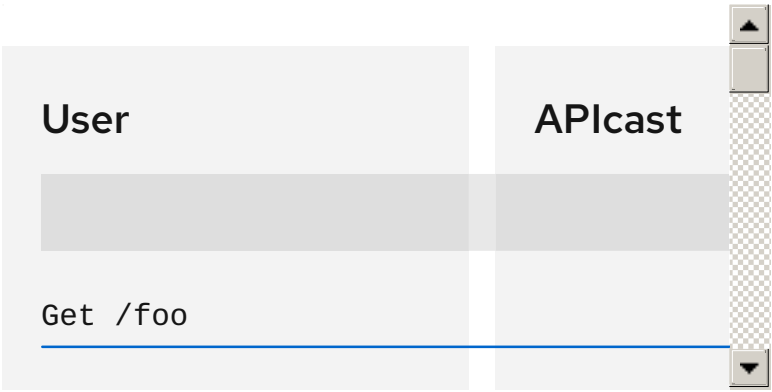
Custom Metrics ポリシーにより、アップストリーム API によって送信されるレスポンスの後にメトリクスを追加することができます。このポリシーの主なユースケースは、レスポンスコードのステータス、ヘッダー、またはさまざまな NGINX 変数に基づいてメトリクスを追加する場合です。

Custom metrics の制限

- リクエストがアップストリーム API に送信される前に認証が行われると、新しいメトリクスをアップストリーム API に報告するために、バックエンドに 2 番目の呼び出しが行われます。
- このポリシーは、バッチ処理のポリシーとは機能しません。
- ポリシーがメトリクスの値をプッシュする前に、管理ポータルでメトリクスを作成する必要があります。

リクエストフローの例

以下のチャートは、認証がキャッシュされていない場合のリクエストフローの例と、認証がキャッシュされている場合のフローを示しています。



設定例

このポリシーは、アップストリーム API が 400 のステータスを返すと、ヘッダーインクリメントだけメトリクス error のカウントを増やします。

```
{
  "name": "apicast.policy.custom_metrics",
  "configuration": {
```

```
    "rules": [
     {
       "metric": "error",
       "increment": "{{ resp.headers['increment'] }}",
       "condition": {
         "operations": [
           {
             "right": "{{status}}",
             "right_type": "liquid",
             "left": "400",
             "op": "=="
           }
         ],
         "combine_op": "and"
       }
     }
   ]
  }
}
```

このポリシーは、アップストリーム API が 200 のステータスを返すと、status_code 情報により hits メトリクスのカウントを増やします。

```
{
  "name": "apicast.policy.custom_metrics",
  "configuration": {
    "rules": [
     {
       "metric": "hits_{{status}}",
       "increment": "1",
       "condition": {
         "operations": [
           {
             "right": "{{status}}",
             "right_type": "liquid",
             "left": "200",
             "op": "=="
           }
         ],
         "combine_op": "and"
       }
     }
   ]
  }
}
```

4.1.11. Echo

Echo ポリシーは、受信したリクエストを出力してクライアントに返します。また、オプションで任意の HTTP ステータスコードを返します。

設定プロパティー

プロパティー	説明	値	必須/任意
status	Echo ポリシーがクライアントに返す HTTP ステータスコード	データタイプ: 整数	任意
exit	Echo ポリシーが使用する終了モードを指定します。**request** 終了モードは、受信したリクエストの処理を停止します。**set** 終了モードは書き換えフェーズを省略します。	データタイプ: 列挙文字列 [request, set]	必須

ポリシーオブジェクトの例

```
{
  "name": "echo",
  "version": "builtin",
  "configuration": {
    "status": 404,
    "exit": "request"
  }
}
```

ポリシーの設定方法に関する情報は、本書の3scale でのポリシーチェーンの作成セクションを参照してください。

4.1.12. Edge Limiting

Edge Limiting ポリシーの目的は、バックエンド API に送信されるトラフィックに対する柔軟な流量制御を提供することで、このポリシーをデフォルトの 3scale 承認メカニズムと共に使用することができます。このポリシーでサポートされるユースケースの例を以下に示します。

- エンドユーザーの流量制御:エンドユーザーの流量制御: リクエストの認証ヘッダーで渡される JWT トークンの **sub** (subject) クレームの値による流量制御。 **{{ jwt.sub }}** として設定されます。
- 1秒あたりのリクエスト数 (RPS) 流量制御
- サービスごとのグローバル流量制御:アプリケーションごとではなく、サービスごとに流量制御を適用します。
- 同時接続上限:許容される同時接続の数を設定します。

制限のタイプ

このポリシーでは、lua-resty-limit-traffic ライブラリーにより提供される以下の制限タイプがサポートされます。

- **leaky_bucket_limiters**: 平均リクエスト数および最大バーストサイズをベースとする、リーキーバケットアルゴリズムに基づきます。

- **fixed_window_limiters**:特定の期間に基づきます (最後の n 秒)。
- **connection_limiters**:同時接続の数に基づきます。

すべての制限をサービスごとまたはグローバルに適用することができます。

制限の定義

リミッターはキーを持ち、制限を定義するのに使用するエンティティーをこのキーでエンコードします (IP アドレス、サービス、エンドポイント、ID、特定ヘッダーの値、その他のエンティティー等)。このキーは、リミッターの **key** パラメーターで指定します。

key は以下のプロパティーで定義されるオブジェクトです。

- **name**:キーの名前を定義します。スコープ内で一意でなければなりません。
- **scope**:キーのスコープを定義します。サポートされるスコープは以下のとおりです。
  - **service**: 1 つのサービスに影響を及ぼすサービスごとのスコープ
  - **global**: すべてのサービスに影響を及ぼすグローバルスコープ
- **name_type**: **name** の値がどのように評価されるかを定義します。
  - **plain**: プレーンテキストとして評価される。
  - **liquid**: Liquid として評価される。

それぞれの制限は、そのタイプに応じたパラメーターも持ちます。

- **leaky_bucket_limiters**: **rate** および **burst**
  - **rate**:遅延を生じることなく実行できる 1 秒あたりのリクエスト数を定義します。
  - **burst**:許容レートを超えることのできる 1 秒あたりのリクエスト数を定義します。**rate** で指定される許容レートを超えるリクエストには、人為的な遅延が適用されます。1 秒あたりのリクエスト数が **burst** で定義されるレートを超えると、リクエストは拒否されます。
- **fixed_window_limiters**: **count** および **window**。**count** は、**window** で定義される秒数あたりに実行できるリクエスト数を定義します。
- **connection_limiters**: **conn**、**burst**、および **delay**
  - **conn**:許容される同時接続の最大数を定義します。**burst** で定義される 1 秒あたりの接続数までこの値を超えることができます。
  - **delay**: 制限を超えた接続を遅延させる秒数を定義します。

例

- service_A に対するリクエストを毎分 10 回許可します。

```
{
  "key": { "name": "service_A" },
  "count": 10,
  "window": 60
}
```

- burst を 10、delay を 1 秒に設定して、100 の同時接続を許可します。

```
{
  "key": { "name": "service_A" },
  "conn": 100,
  "burst": 10,
  "delay": 1
}
```

サービスごとにさまざまな制限を定義することができます。複数の制限を定義した場合には、少なくとも 1 つの制限に達すると、リクエストは拒否または遅延されます。

Liquid テンプレート

Edge Limiting ポリシーではキーに Liquid 変数がサポートされるので、動的なキーに制限を指定することができます。そのためには、キーの **name_type** パラメーターを **liquid** に設定する必要があります。これにより、**name** パラメーターに Liquid 変数を使用することができます。たとえば、クライアント IP アドレスの場合は **{{ remote_addr }}**、JWT トークンの **sub** クレームの場合は **{{ jwt.sub }}** を設定します。

例

```
{
  "key": { "name": "{{ jwt.sub }}", "name_type": "liquid" },
  "count": 10,
  "window": 60
}
```

Liquid のサポートに関する詳細な情報は、「ポリシーでの変数およびフィルターの使用」を参照してください。

条件の適用

各リミッターには、リミッターが適用されるタイミングを定義する条件がなければなりません。条件は、リミッターの **condition** プロパティーで指定します。

以下のプロパティーで **condition** を定義します。

- **combine_op**: 演算の一覧に適用されるブール演算子です。**or** および **and** の値がサポートされます。
- **operations**:評価する必要がある条件のリストです。各演算は、以下のプロパティーを持つオブジェクトにより表されます。
  - **left**:演算の左側部分
  - **left_type**: **left** プロパティーがどのように評価されるか (plain または liquid)。
  - **right**:演算の右側部分
  - **right_type**: **right** プロパティーがどのように評価されるか (plain または liquid)。
  - **op**:左右の部分の間に適用される演算子。**==** (等しい) および **!=** (等しくない) の 2 つの値がサポートされます。

例

```
"condition": {
  "combine_op": "and",
  "operations": [
    {
      "op": "==",
      "right": "GET",
      "left_type": "liquid",
      "left": "{{ http_method }}",
      "right_type": "plain"
    }
  ]
}
```

流量制御カウンター用ストレージの設定

デフォルトでは、Edge Limiting ポリシーは流量制御カウンターに OpenResty 共有ディクショナリーを使用します。ただし、共有ディクショナリーの代わりに外部の Redis サーバーを使用することができます。この設定は、複数の APIcast インスタンスがデプロイされている場合に役立ちます。**redis_url** パラメーターを使用して Redis サーバーを設定することができます。

エラー処理

リミッターでは、エラーの処理方法を設定するために以下のパラメーターがサポートされます。

- **limits_exceeded_error**: 設定した上限を超えた際にクライアントに返されるエラーステータスコードおよびメッセージを指定します。以下のパラメーターを設定する必要があります。
  - **status_code**:上限を超えた際のリクエストのステータスコード。デフォルト: **429**
  - **error_handling**: 以下のオプションを使用して、エラーの処理方法を指定します。
    - **exit**:リクエストの処理を中止し、エラーメッセージを返します。
    - **log**:リクエストの処理を完了し、出力ログを返します。
- **configuration_error**: 設定が正しくない場合にクライアントに返されるエラーステータスコードおよびメッセージを指定します。以下のパラメーターを設定する必要があります。
  - **status_code**:設定に問題がある場合のステータスコード。デフォルト: **500**
  - **error_handling**: 以下のオプションを使用して、エラーの処理方法を指定します。
    - **exit**:リクエストの処理を中止し、エラーメッセージを返します。
    - **log**:リクエストの処理を完了し、出力ログを返します。

4.1.13. Header Modification

Header Modification ポリシーを使用すると、受信したリクエストまたはレスポンスに関して、既存のヘッダーを変更する、補足のヘッダーを定義して追加する、またはヘッダーを削除することができます。レスポンスヘッダーおよびリクエストヘッダーの両方を変更することができます。

Header Modification ポリシーでは、以下の設定パラメーターがサポートされます。

- **request**: リクエストヘッダーに適用する操作のリスト
- **response**: レスポンスヘッダーに適用する操作のリスト

それぞれの操作は、以下のパラメーターで設定されます。

- **op**:適用する操作を指定します。**add** 操作は既存のヘッダーに値を追加します。**set** 操作はヘッダーおよび値を作成し、既存のヘッダーの値が既に存在する場合はその値を上書きします。**push** 操作はヘッダーおよび値を作成しますが、既存のヘッダーの値が既に存在していてもその値を上書きしません。その代わりに、**push** は既存のヘッダーに値を追加します。**delete** 操作はヘッダーを削除します。

- **header**:作成または変更するヘッダーを指定します。ヘッダー名には任意の文字列を使用することができます (例: **Custom-Header**)。

- **value_type**:ヘッダーの値がどのように評価されるかを定義します。プレーンテキストの場合の **plain** または Liquid テンプレートとして評価する場合の **liquid** いずれかに設定します。詳細は、「ポリシーでの変数およびフィルターの使用」 を参照してください。

- **value**:ヘッダーに使用される値を指定します。値のタイプが liquid の場合には、値は **{{ variable_from_context }}** の形式にする必要があります。削除する場合には不要です。

ポリシーオブジェクトの例

```
{
  "name": "headers",
  "version": "builtin",
  "configuration": {
    "response": [
      {
        "op": "add",
        "header": "Custom-Header",
        "value_type": "plain",
        "value": "any-value"
      }
    ],
    "request": [
      {
        "op": "set",
        "header": "Authorization",
        "value_type": "plain",
        "value": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="
      },
      {
        "op": "set",
        "header": "Service-ID",
        "value_type": "liquid",
        "value": "{{service.id}}"
      }
    ]
  }
}
```

ポリシーの設定方法に関する情報は、本書の3scale でのポリシーチェーンの作成セクションを参照してください。

4.1.14. HTTP Status Code Overwrite

API プロバイダーは、HTTP Status Code Overwrite ポリシーを API プロダクトに追加できます。このポリシーにより、アップストリームのレスポンスコードを、指定したレスポンスコードに変更できます。

3scale は、HTTP Status Code Overwrite ポリシーをアップストリームサービスから送信されたレスポンスコードに適用します。つまり、3scale が公開する API が状況に合わないコードを返す場合には、そのコードをアプリケーションに意味のあるレスポンスコードに変更するために、HTTP Status Code Overwrite ポリシーを設定することができます。

ポリシーチェーンでは、変更するレスポンスコードを生成するポリシーが HTTP Status Code Overwrite ポリシーの前にある必要があります。変更するステータスコードを生成するポリシーがない場合、HTTP Status Code Overwrite ポリシーのポリシーチェーンでの位置は重要ではありません。

管理ポータルで、HTTP Status Code Overwrite ポリシーをプロダクトのポリシーチェーンに追加します。ポリシーチェーンでポリシーをクリックし、変更するアップストリームレスポンスコードと、代わりに返したいレスポンスコードを指定します。上書きする追加のアップストリームレスポンスコードごとに正符号をクリックします。たとえば、HTTP Status Code Overwrite ポリシーを使用して、アップストリームの **201**Created レスポンスコードを **200**OK レスポンスコードに変更できます。

変更するレスポンスコードのもう 1 つの例は、コンテンツ制限を超える場合のレスポンスです。アップストリームが **413** payload too large を返す場合に、 **414** request-URI too long のレスポンスコードであれば役立ちます。

管理ポータルで HTTP Status Code Overwrite ポリシーを追加する代わりに、ポリシーチェーン設定ファイルと共に 3scale API を使用できます。

例

ポリシーチェーン設定ファイルの以下の JSON 設定は、2 つのアップストリームのレスポンスコードを上書きします。

```
{
  "name": "statuscode_overwrite",
  "version": "builtin",
  "configuration": {
    "http_statuses": [
      {
        "upstream": 200,
        "apicast": 201
      },
      {
        "upstream": 413,
        "apicast": 414
      }
    ]
  }
}
```

4.1.15. HTTP2 Endpoint

HTTP2 エンドポイントポリシーは、リクエストを送信するコンシューマーアプリケーションと APIcast 間の HTTP/2 プロトコル接続を有効にします。HTTP2 エンドポイントポリシーがプロダクトのポリシーチェーンにある場合、API キャストにリクエストを行うコンシューマーアプリケーションからアップストリームサービスへの通信フロー全体が HTTP/2 プロトコルを使用することができます。

HTTP2 エンドポイントポリシーがポリシーチェーンにある場合:

- リクエスト認証は、JSON Web トークンまたは **App_ID** と **App_Key** のペアを使用して指定する必要があります。API キー認証はサポートされていません。

- HTTP2 エンドポイントポリシーは、3scale APIcast ポリシーの前に指定する必要があります。
- アップストリームサービスのバックエンドは、HTTP/1.1 プレーンテキストまたは Transport Layer Security (TLS) を実装できます。
- ポリシーチェーンには、TLS Termination ポリシーも含まれる必要があります。

4.1.16. IP Check

IP Check ポリシーは、IP のリストに基づいてリクエストを拒否または許可するために使用します。

設定プロパティー

プロパティー	説明	データタイプ	必須/任意
check_type	**check_type** プロパティーには、**whitelist** または **blacklist** の 2 つの値を指定することができます。**blacklist** は、リスト上の IP からのリクエストをすべて拒否します。**whitelist** は、リスト上に **ない** IP からのリクエストをすべて拒否します。	文字列 (**whitelist** または **blacklist** のどちらかでなければなりません)	必須
ips	**ips** プロパティーでは、ホワイトリストまたはブラックリストに登録する IP アドレスのリストを指定することができます。個別の IP および CIDR 範囲の両方を使用することができます。	文字列の配列 (有効な IP アドレスでなければなりません)	必須
error_msg	**error_msg** プロパティーを使用して、リクエストが拒否された時に返されるエラーメッセージを設定することができます。	文字列	任意

プロパティー	説明	データタイプ	必須/任意
client_ip_sources	**client_ip_sources** プロパティーでは、クライアント IP の取得方法を設定することができます。デフォルトでは、最後に呼び出しを実施した IP が使用されます。これ以外のオプションは、**X-Forwarded-For** および **X-Real-IP** です。	文字列の配列。有効なオプションは、**X-Forwarded-For**、**X-Real-IP**、および **last_caller** です (複数の選択が可能です)。	任意

ポリシーオブジェクトの例

```
{
  "name": "ip_check",
  "configuration": {
    "ips": [ "3.4.5.6", "1.2.3.0/4" ],
    "check_type": "blacklist",
    "client_ip_sources": ["X-Forwarded-For", "X-Real-IP", "last_caller"],
    "error_msg": "A custom error message"
  }
}
```

ポリシーの設定方法に関する情報は、本書の3scale でのポリシーチェーンの作成セクションを参照してください。

4.1.17. JWT Claim Check

JWT Claim Check ポリシーを使用すると、JSON Web Token (JWT) クレームに基づきリソースターゲットおよびメソッドをブロックする、新たなルールを定義することができます。

JWT Claim Check ポリシーの概要

JWT クレームの値に基づいてルーティングするためには、ポリシーチェーンには、ポリシーが共有するコンテキストで JWT を検証してクレームを格納するポリシーが必要です。

JWT Claim Check ポリシーがリソースおよびメソッドをブロックしている場合には、ポリシーは JWT 操作も検証します。一方、メソッドおよびリソースがマッチしない場合には、リクエストはバックエンド API に送信されます。

以下に例を示します。以下の GET リクエストの例では、JWT には管理者として role クレームが必要です。もしなければ、リクエストは拒否されます。一方、GET 以外のリクエストは JWT 操作を検証しないため、POST リソースは JWT の制約なしに許可されます。

```
{
  "name": "apicast.policy.jwt_claim_check",
  "configuration": {
     "error_message": "Invalid JWT check",
     "rules": [
        {
           "operations": [
```

```
                {"op": "==", "jwt_claim": "role", "jwt_claim_type": "plain", "value": "admin"}
            ],
            "combine_op":"and",
            "methods": ["GET"],
            "resource": "/resource",
            "resource_type": "plain"
        }
    ]
  }
}
```

ポリシーチェーンへの JWT Claim Check ポリシーの設定

ポリシーチェーンで JWT Claim Check ポリシーを設定するには、以下の条件を満たす必要があります。

- 3scale システムにアクセスできること。
- すべてのデプロイメントが完了していること。

ポリシーの設定

1. 3scale 管理ポータルでのポリシーの有効化に記載の手順に従って JWT Claim Check を選択し、API に JWT Claim Check ポリシーを追加します。
2. **JWT Claim Check**のリンクをクリックします。
3. **Enabled** のチェックボックスを選択し、ポリシーを有効にします。
4. ルールを追加するには、プラス (**+**) アイコンをクリックします。
5. **resource_type** を指定します。
6. 演算子を選択します。
7. ルールによって制御される **resource** を指定します。
8. 許可されるメソッドを追加するには、プラス (**+**) アイコンをクリックします。
9. トラフィックがブロックされた時にユーザーに表示するエラーメッセージを入力します。
10. API への JWT Claim Check ポリシーの設定が完了したら、**Update Policy** をクリックします。該当するセクションのプラス (**+**) アイコンをクリックして、さらにリソースタイプおよび許可されるメソッドを追加することができます。
11. **Update Policy Chain** をクリックして変更を保存します。

4.1.18. Liquid Context Debug

注記

Liquid Context Debug ポリシーの想定は、開発環境でのデバッグ用途のみで、実稼働環境での使用は意図されていません。

このポリシーは **JSON** を使用して API リクエストに応答します。コンテキストで利用可能なオブジェ

クトおよび値を含み、Liquid テンプレートの評価に使用することができます。3scale APIcast または Upstream ポリシーと組み合わせる場合には、正常な動作のためにポリシーチェーンではこれらのポリシーの前に Liquid Context Debug ポリシーを設定する必要があります。循環参照を避けるために、ポリシーには重複するオブジェクトは 1 回だけ含め、それをスタブ値で置き換えます。

このポリシーが有効な時に APIcast が返す値の例を以下に示します。

```
{
  "jwt": {
    "azp": "972f7b4f",
    "iat": 1537538097,
    ...
    "exp": 1537574096,
    "typ": "Bearer"
  },
  "credentials": {
    "app_id": "972f7b4f"
  },
  "usage": {
    "deltas": {
      "hits": 1
    },
    "metrics": [
      "hits"
    ]
  },
  "service": {
    "id": "2",
    ...
  }
  ...
}
```

4.1.19. Logging

Logging ポリシーには 2 つの目的があります。

- アクセスログの出力を有効および無効にする。
- それぞれのサービスごとにカスタムアクセスログのフォーマットを作成し、カスタムアクセスログ書き込みの条件を設定できるようにする。

Logging ポリシーとアクセスログの場所のグローバル設定を組み合わせることができます。APIcast アクセスログの場所を設定するには、**APICAST_ACCESS_LOG_FILE** 環境変数を設定します。デフォルトでは、この変数は標準出力デバイスの **/dev/stdout** に設定されています。グローバル APIcast パラメーターに関する詳細な情報は、*7章APIcast の環境変数*を参照してください。

また、Logging ポリシー機能に関する補足情報を以下に示します。

- このポリシーは、**enable_access_logs** 設定パラメーターしかサポートしません。
- アクセスログを有効にするには、**enable_access_logs** パラメーターを選択するか、Logging ポリシーを無効にします。
- API のアクセスログを無効にするには、以下の手順を実施します。

1. ポリシーを有効にします。
2. **enable_access_logs** パラメーターの選択を解除します。
3. **Submit** ボタンをクリックします。

- デフォルトでは、このポリシーはポリシーチェーンで有効になっていません。

4.1.19.1. すべての API のロギングポリシーの設定

APICAST_ENVIRONMENT を使用すると、ポリシーがすべての API プロダクトのシステム全体に適用される設定をロードできます。これを実現する方法の例を以下に示します。APICAST_ENVIRONMENT は、デプロイメント、テンプレート、または演算子のタイプに応じて異なるファイルのパスを参照するために使用されます。

システム全体でロギングポリシーを設定するには、デプロイメントタイプに応じて以下の点を考慮してください。

- テンプレートベースのデプロイメントの場合、ConfigMap および VolumeMount を使用してコンテナーにファイルをマウントする必要があります。
- 3scale operator ベースのデプロイメントの場合:
  - 3scale 2.11 より前のバージョンでは、ConfigMap および VolumeMount を使用してコンテナーにファイルをマウントする必要があります。
  - 3scale 2.11 の時点で、APIManager カスタムリソース (CR) で参照されるシークレットを使用する必要があります。
- APIcast Operator のデプロイメントの場合:
  - 3scale 2.11 以前は設定できませんでした。
  - 3scale 2.11 の時点で、APIManager カスタムリソース (CR) で参照されるシークレットを使用する必要があります。
- Docker にデプロイされた Self-managed APIcast の場合には、ファイルをコンテナーにマウントする必要があります。

Logging オプションは、API でログが正しくフォーマットされない問題を避けるのに役立ちます。

以下は、すべてのサービスでロードされるポリシーの例です。

custom_env.lua ファイル

```
local cjson = require('cjson')
local PolicyChain = require('apicast.policy_chain')
local policy_chain = context.policy_chain

local logging_policy_config = cjson.decode([[
{
  "enable_access_logs": false,
  "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}"
}
]])

policy_chain:insert( PolicyChain.load_policy('logging', 'builtin', logging_policy_config), 1)
```

```
return {
  policy_chain = policy_chain,
  port = { metrics = 9421 },
}
```

4.1.19.1.1. ConfigMap および VolumeMount を使用してコンテナーにファイルをマウントし、すべての API のロギングポリシーを設定する方法

1. **custom_env.lua** ファイルで ConfigMap を作成します。

```
oc create configmap logging --from-file=/path/to/custom_env.lua
```

2. ConfigMap のボリュームをマウントします (例: **apicast-staging**)。

```
oc set volume dc/apicast-staging --add --name=logging --mount-path=/opt/app-root/src/config/custom_env.lua --sub-path=custom_env.lua -t configmap --configmap-name=logging
```

3. 環境変数を設定します。

```
oc set env dc/apicast-staging APICAST_ENVIRONMENT=/opt/app-root/src/config/custom_env.lua
```

4.1.19.1.2. APIManager カスタムリソース (CR) で参照されるシークレットを使用してすべての API のロギングポリシーを設定する方法

operator ベースのデプロイメントの 3scale 2.11 以降では、ロギングポリシーをシークレットとして設定し、APIManager カスタムリソース (CR) でシークレットを参照します。

注記

以下の手順は、3scale operator にのみ有効です。ただし、以下の手順に従って APIcast operator を設定することができます。

前提条件

- Lua でコーディングされる 1 つ以上のカスタム環境。

手順

1. カスタム環境コンテンツでシークレットを作成します。

```
$ oc create secret generic custom-env --from-file=./custom_env.lua
```

2. APIcast カスタム環境を使用して APIManager CR を設定およびデプロイします。

apimanager.yaml content:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
```

```
  name: apimanager-apicast-custom-environment
spec:
  apicast:
    productionSpec:
      customEnvironments:
        - secretRef:
            name: custom-env
    stagingSpec:
      customEnvironments:
        - secretRef:
            name: custom-env
```

3. APIManager CR をデプロイします。

```
$ oc apply -f apimanager.yaml
```

シークレットが存在しない場合、Operator は CR を failed とマークします。シークレットを変更するには、APIcast に反映するために Pod/コンテナーを再デプロイする必要があります。

カスタム環境の更新

カスタムの環境コンテンツを変更する必要がある場合は、以下の 2 つのオプションがあります。

- 推奨:別の名前で別のシークレットを作成し、APIManager CR フィールドを更新します。

  ```
  customEnvironments[].secretRef.name
  ```

  Operator は、新しいカスタム環境コンテンツをロードするローリング更新をトリガーします。

- 既存のシークレットコンテンツを更新し、APIcast を再デプロイして **spec.apicast.productionSpec.replicas** または **spec.apicast.stagingSpec.replicas** を 0 にしてから、以前の値に戻します。

4.1.19.1.3. Docker 上にデプロイされた Self-managed APIcast のすべての API のロギングポリシーの設定

以下の Docker コマンドを使用し、custom_env.lua をマウントして、この特定の環境で APIcast を実行します。

```
docker run --name apicast --rm -p 8080:8080 \
    -v $(pwd):/config \
    -e APICAST_ENVIRONMENT=/config/custom_env.lua \
    -e THREESCALE_PORTAL_ENDPOINT=https://ACCESS_TOKEN@ADMIN_PORTAL_DOMAIN \
    quay.io/3scale/apicast:master
```

考慮すべき Docker コマンドの重要な概念を以下に示します。

- 現在の Lua ファイルをコンテナー **-v $(pwd):/config** と共有します。
- APICAST_ENVIRONMENT 変数を、**/config** ディレクトリーに保存されている Lua ファイルに設定します。

4.1.19.2. ロギングポリシーの例

これらは Logging ポリシーの例で、以下の点に注意してください。

- **custom_logging** または **enable_json_logs** プロパティーが有効な場合には、デフォルトのアクセスログは無効になる。
- **enable_json_logs** が有効な場合には、**custom_logging** フィールドは省略される。

アクセスログの無効化

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false
  }
}
```

カスタムアクセスログの有効化

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "[{{time_local}}] {{host}}:{{server_port}} {{remote_addr}}:{{remote_port}} \"
{{request}}\" {{status}} {{body_bytes_sent}} ({{request_time}}) {{post_action_impact}}",
  }
}
```

サービスを識別した上でのカスタムアクセスログの有効化

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.serializable.name}}",
  }
}
```

JSON 形式でのアクセスログの設定

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "enable_json_logs": true,
    "json_object_config": [
      {
        "key": "host",
        "value": "{{host}}",
        "value_type": "liquid"
      },
      {
        "key": "time",
        "value": "{{time_local}}",
        "value_type": "liquid"
      },
      {
```

```
        "key": "custom",
        "value": "custom_method",
        "value_type": "plain"
      }
    ]
  }
}
```

正常なリクエストのみに対するカスタムアクセスログの設定

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}",
    "condition": {
      "operations": [
        {"op": "==", "match": "{{status}}", "match_type": "liquid", "value": "200"}
      ],
      "combine_op": "and"
    }
  }
}
```

レスポンスのステータスが 200 または 500 のいずれかにマッチする場合のアクセスログのカスタマイズ

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}",
    "condition": {
      "operations": [
        {"op": "==", "match": "{{status}}", "match_type": "liquid", "value": "200"},
        {"op": "==", "match": "{{status}}", "match_type": "liquid", "value": "500"}
      ],
      "combine_op": "or"
    }
  }
}
```

4.1.19.3. カスタムロギングに関する補足情報

カスタムロギングでは、エクスポートした変数と共に Liquid テンプレートを使用することができます。この変数の例を以下に示します。

- NGINX デフォルトディレクティブ変数: log_format。たとえば **{{remote_addr}}**。
- レスポンスおよびリクエストヘッダー:
  - **{{req.headers.FOO}}**:リクエストの FOO ヘッダーを取得する。
  - **{{res.headers.FOO}}**:レスポンスの FOO ヘッダーを取得する。

- サービス情報 (例: **{{service.id}}**)、および以下のパラメーターにより提供されるすべてのサービスプロパティー:
  - THREESCALE_CONFIG_FILE
  - THREESCALE_PORTAL_ENDPOINT

4.1.20. Maintenance Mode

Maintenance Mode ポリシーを使用すると、指定したステータスコードおよびメッセージと共に受信したリクエストを拒否することができます。このポリシーは、メンテナーンス期間または API を一時的にブロックする場合に役立ちます。

設定プロパティー

設定可能なプロパティーおよびデフォルト値を、以下のリストに示します。

プロパティー	値	デフォルト	説明
status	整数 (**オプション**)	503	レスポンスコード
message	文字列 (**オプション**)	503 Service Unavailable - Maintenance	レスポンスのメッセージ

Maintenance Mode ポリシーの例

```
{
  "policy_chain": [
    {"name": "maintenance-mode", "version": "1.0.0",
    "configuration": {"message": "Be back soon..", "status": 503} },
  ]
}
```

特定のアップストリームでのメンテナーンスモードの適用

```
{
    "name": "maintenance_mode",
    "version": "builtin",
    "configuration": {
        "message_content_type": "text/plain; charset=utf-8",
        "message": "Echo API /test is currently Unavailable",
        "condition": {
            "combine_op": "and",
            "operations": [
                {
                    "left_type": "liquid",
                    "right_type": "plain",
                    "op": "==",
                    "left": "{{ original_request.path }}",
                    "right": "/test"
                }
            ]
        },
```

```
        "status": 503
    }
}
```

ポリシーの設定方法に関する情報は、本書の3scale でのポリシーチェーンの作成セクションを参照してください。

4.1.21. NGINX Filter

NGINX は一部のリクエストヘッダーを自動的にチェックし、これらのヘッダーを検証できない場合にリクエストを拒否します。たとえば、NGINX は NGINX が検証できない **If-Match** ヘッダーのあるリクエストを拒否します。NGINX が特定のヘッダーの検証を省略するようにする場合は、NGINX Filter ポリシーを追加します。

NGINX Filter ポリシーを追加する場合、NGINX が検証をスキップするリクエストヘッダーを 1 つ以上指定します。指定するヘッダーごとに、ヘッダーをリクエストに残すかどうかを指定します。たとえば、以下の JSON コードは NGINX Filter ポリシーを追加して、**If-Match** ヘッダーの検証をスキップするが、アップストリームサーバーに転送されるリクエストに **If-Match** ヘッダーを保持するように設定することができます。

```
{ "name": "apicast.policy.nginx_filters",
  "configuration": {
    "headers": [
      {"name": "If-Match", "append": true}
    ]
  }
}
```

以下の例でも **If-Match** ヘッダーの検証をスキップしますが、このコードは、リクエストをアップストリームサーバーに送信する前に **If-Match** ヘッダーを削除するように NGINX に指示します。

```
{ "name": "apicast.policy.nginx_filters",
  "configuration": {
    "headers": [
      {"name": "If-Match", "append": false}
    ]
  }
}
```

アップストリームサーバーに送信されるリクエストに指定のヘッダーを追加するかどうかに関係なく、NGINX が指定したヘッダーを検証できない場合に NGINX **412** 応答コードを回避します。

重要

Header Modification ポリシーと NGINX Filter ポリシー に同じヘッダーを指定すると、競合の原因となる可能性があります。

4.1.22. OAuth 2.0 Mutual TLS Client Authentication

このポリシーは、全 API コールに対して OAuth 2.0 相互 TLS クライアント認証を実行します。

OAuth 2.0 Mutual TLS Client Authentication ポリシーの **JSON** の例を以下に示します。

```
{
```

```
  "$schema": "http://apicast.io/policy-v1/schema#manifest#",
  "name": "OAuth 2.0 Mutual TLS Client Authentication",
  "summary": "Configure OAuth 2.0 Mutual TLS Client Authentication.",
  "description": ["This policy executes OAuth 2.0 Mutual TLS Client Authentication ",
    "(https://tools.ietf.org/html/draft-ietf-oauth-mtls-12) for every API call."
  ],
  "version": "builtin",
  "configuration": {
    "type": "object",
    "properties": { }
  }
}
```

4.1.23. OAuth 2.0 Token Introspection

OAuth 2.0 Token Introspection ポリシーを使用すると、OpenID Connect (OIDC) 認証オプションを用いるサービスに使用される JSON Web Token (JWT) トークンを検証することができます。この場合、トークン発行者 (Red Hat Single Sign-On) のトークンイントロスペクションエンドポイントを使用します。

トークンイントロスペクションエンドポイントおよびそのエンドポイントに呼び出しを行う際に APIcast が使用するクレデンシャルを定義する場合、**auth_type** フィールドでは以下の認証タイプがサポートされます。

- **use_3scale_oidc_issuer_endpoint**: APIcast は、クライアントのクレデンシャル (**クライアント ID** および **クライアントシークレット**) ならびに Service Integration ページで定義した OIDC 発行者設定からのトークンイントロスペクションエンドポイントを使用します。APIcast は、**token_introspection_endpoint** フィールドからトークンイントロスペクションエンドポイントを検出します。このフィールドは、OIDC 発行者が返す **.well-known/openid-configuration** エンドポイントにあります。
  use_3scale_oidc_issuer_endpoint に設定された認証タイプ:

  ```
  "policy_chain": [
  …
    {
      "name": "apicast.policy.token_introspection",
      "configuration": {
        "auth_type": "use_3scale_oidc_issuer_endpoint"
      }
    }
  …
  ],
  ```

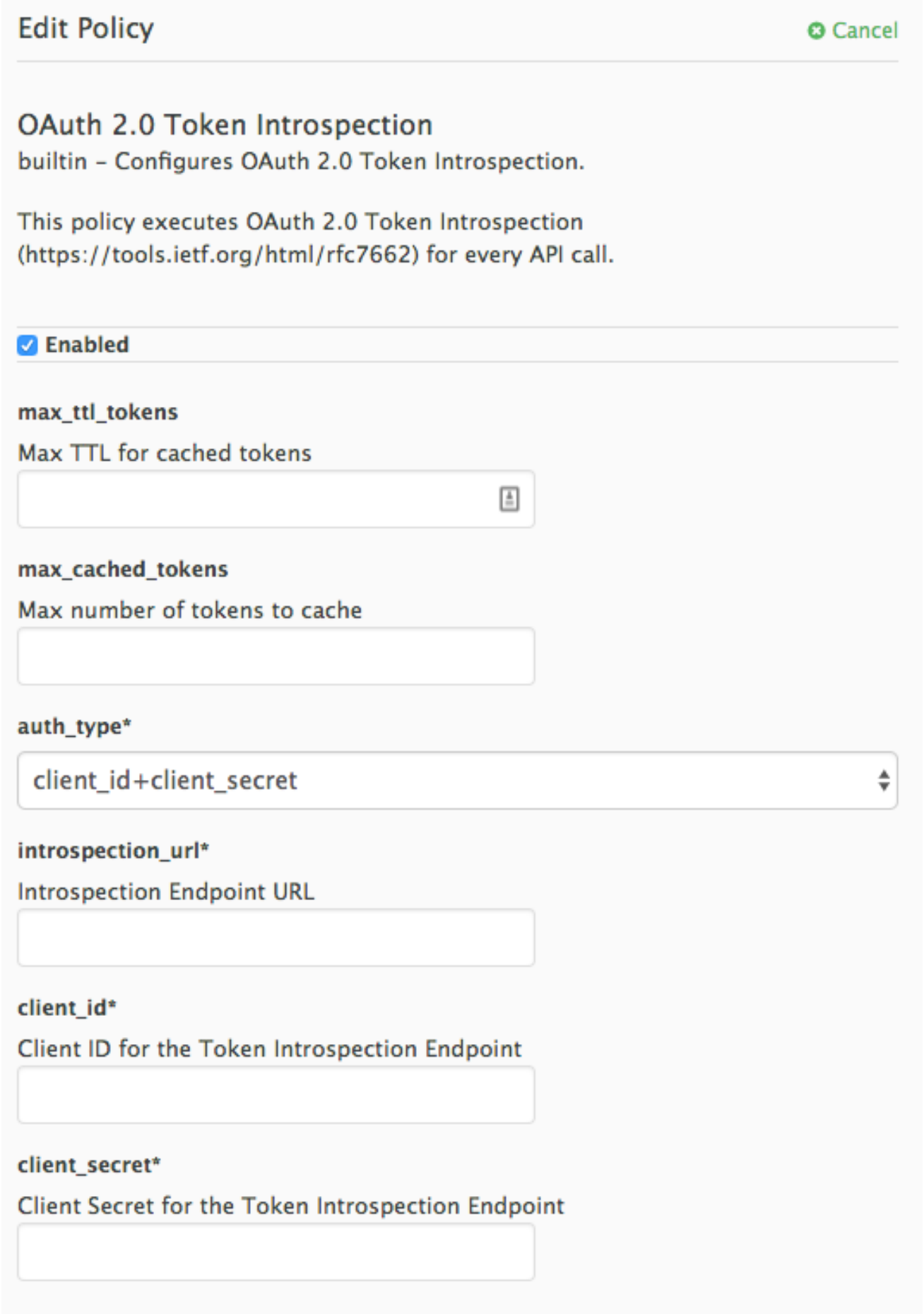
- **client_id+client_secret**:このオプションでは、APIcast がトークン情報を要求するのに使用する **クライアント ID** および **クライアントシークレット** と共に、異なるトークンイントロスペクションエンドポイントを指定することができます。このオプションを使用する場合には、以下の設定パラメーターを定義します。

  - **client_id**:トークンイントロスペクションエンドポイント用のクライアント ID を設定します。
  - **client_secret**:トークンイントロスペクションエンドポイント用のクライアントシークレットを設定します。
  - **introspection_url**: イントロスペクションエンドポイントの URL を設定します。

client_id+client_secret に設定された認証タイプ:

```
"policy_chain": [
…
  {
    "name": "apicast.policy.token_introspection",
    "configuration": {
      "auth_type": "client_id+client_secret",
      "client_id": "myclient",
      "client_secret": "mysecret",
      "introspection_url": "http://red_hat_single_sign-on/token/introspection"
    }
  }
…
],
```

auth_type フィールドの設定にかかわらず、APIcast は Basic 認証を使用してトークンイントロスペクションの呼び出しを承認します (**Authorization: Basic <token>** ヘッダー、ここで **<token>** は Base64 でエンコードされた **<client_id>:<client_secret>** 設定です)。



トークンイントロスペクションエンドポイントのレスポンスには、**active** 属性が含まれます。APIcast はこの属性の値を確認します。属性の値に応じて、APIcast は呼び出しを承認または拒否します。

- **true**:呼び出しを承認します
- **false**: **Authentication Failed** エラーと共に呼び出しを拒否します

このポリシーを使用すると、トークンのキャッシュを有効にして、同じ JWT トークンに対する呼び出しのたびにトークンイントロスペクションエンドポイントに呼び出しを行うのを避けることができます。Token Introspection ポリシーのトークンキャッシュを有効にするには、**max_cached_tokens** フィールドを **0** (機能は無効) から **10000** までの値に設定します。さらに、**max_ttl_tokens** フィールドで、トークンの残存期間 (TTL) の値を **1** から **3600** 秒に設定することができます。

4.1.24. On Fail

API プロバイダーは、On Fail ポリシーを API プロダクトに追加できます。On Fail ポリシーがポリシーチェーンにあり、特定の API コンシューマーリクエストに対してポリシーの実行に失敗すると、APIcast は以下の処理を行います。

- リクエストの処理を停止します。
- リクエストを送信するアプリケーションに指定したステータスコードを返します。

On Fail ポリシーは、不適切な設定やカスタムポリシーのコンプライアンス違反のコードが原因で、APIcast がポリシーを処理できない場合に役立ちます。ポリシーチェーンに On Fail ポリシーがないと、APIcast はポリシーを適用できないポリシーをスキップし、チェーン内の他のポリシーを処理し、リクエストをアップストリーム API に送信します。ポリシーチェーンに On Fail ポリシーがあると、APIcast はリクエストを拒否します。

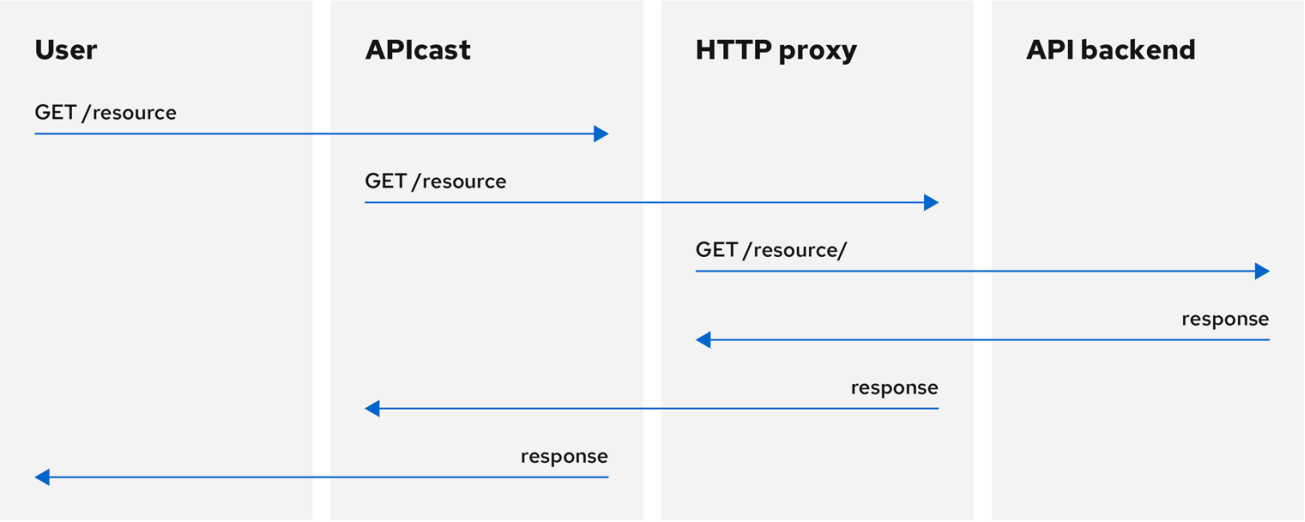
ポリシーチェーンでは、On Fail ポリシーを任意の場所に配置できます。

管理ポータルで、On Fail ポリシーをプロダクトのポリシーチェーンに追加します。ポリシーチェーンでポリシーをクリックし、On Fail ポリシーを適用する際に APIcast が返すステータスコードを指定します。たとえば、クライアントからの不適切なリクエストを示す **400** を指定できます。

4.1.25. Proxy Service

Proxy Service ポリシーを使用して、定義したプロキシーを使用して 3scale のトラフィックを送信する一般的な **HTTP プロキシー** を定義することができます。この場合、プロキシーサービスはリバース HTTP プロキシーとして機能し、APIcast は HTTP プロキシーにトラフィックを送信し、プロキシーがそのトラフィックを API バックエンドに送信します。

トラフィックフローの例を以下に示します。



3scale バックエンドに送信された APIcast トラフィックは、すべてプロキシーを使用しません。このポリシーは、プロキシーおよび APIcast と API バックエンド間の通信にのみ適用されます。

すべてのトラフィックをプロキシー経由で送信するには、**HTTP_PROXY** 環境変数を使用する必要があります。

注記

- Proxy Service ポリシーはすべての負荷分散ポリシーを無効にし、トラフィックはプロキシーに送信されます。
- **HTTP_PROXY**、**HTTPS_PROXY**、または **ALL_PROXY** パラメーターが定義されている場合には、このポリシーによりこれらのパラメーターの値が上書きされます。
- プロキシー接続では、認証はサポートされません。認証には Header Modification ポリシーを使用します。

設定

ポリシーチェーンの設定例を以下に示します。

```
"policy_chain": [
  {
    "name": "apicast.policy.apicast"
  },
  {
    "name": "apicast.policy.http_proxy",
    "configuration": {
        "all_proxy": "http://192.168.15.103:8888/",
        "https_proxy": "https://192.168.15.103:8888/",
        "http_proxy": "https://192.168.15.103:8888/"
    }
  }
]
```

http_proxy または **https_proxy** が定義されていない場合は、**all_proxy** の値が使用されます。

ユースケースの例

Proxy Service ポリシーは、HTTP を使用した Apache Camel を用いて、3scale でより粒度の細かいポリシーおよび変換を適用できるように作られました。ただし、Proxy Service ポリシーを一般的な HTTP プロキシーサービスとして使用することもできます。HTTPS を使用した Apache Camel とのインテグレーションについては、「Camel Service」を参照してください。

プロジェクトの例

GitHub の camel-netty-proxy の例を参照してください。このプロジェクトには、API バックエンドからのレスポンスボディーを大文字に変換する HTTP プロキシーが示されています。

4.1.26. Rate Limit Headers

アプリケーションが流量制御の設定されたアプリケーションプランにサブスクライブする際に、Rate Limit Headers ポリシーはレスポンスメッセージに **RateLimit** ヘッダーを追加します。これらのヘッダーは、設定されたリクエストクォータ制限、ならびに現在の時間枠での残りのリクエストクォータおよび秒数に関する有用な情報を提供します。

プロダクトのポリシーチェーンで、Rate Limit Headers ポリシーを追加する場合、3scale APIcast ポリシーの前に設定する必要があります。3scale APIcast ポリシーが Rate Limit Headers ポリシーの前にあると、Rate Limit Headers ポリシーは機能しません。

RateLimit ヘッダー

以下の **RateLimit** ヘッダーがそれぞれのメッセージに追加されます。

- **RateLimit-Limit**:設定された時間枠での合計リクエストクォータを表示します (例: **10** リクエスト)。
- **RateLimit-Remaining**:現在の時間枠での残りのリクエストクォータを表示します (例: **5** リクエスト)。
- **RateLimit-Reset**:現在の時間枠での残り時間を秒数で表示します (例: **30** 秒)。このヘッダーの動作は、**Retry-After** ヘッダーの **delta-seconds** 表記と互換性があります。

デフォルトでは、Rate Limit Headers ポリシーが設定されていない場合やアプリケーションプランに流量制御が設定されていない場合、レスポンスメッセージには流量制御に関するヘッダーがありません。

注記

流量制御を設定せずに API メトリクスを要求しているが、親メトリクスに制限が設定されている場合、親の制限が適用されるため、流量制御に関するヘッダーがレスポンスに含まれます。

関連情報

- インターネットドラフト: RateLimit Header Fields for HTTP
- Configuring 3scale application plans and rate limits
- Configuring 3scale API metrics

4.1.27. Response/Request Content Limits

API プロバイダーは、Response/Request Content Limits ポリシーを API プロダクトに追加できます。このポリシーにより、アップストリーム API へのリクエストのサイズやアップストリーム API からのレ

スポンスのサイズを制限できます。このポリシーがない場合、リクエスト/レスポンスのサイズは無制限になります。

このポリシーは、以下の項目のオーバーロードを防ぐのに役立ちます。

- バックエンド。大きすぎるペイロードに対応する必要があるため。
- エンドユーザー (API コンシューマー)。処理能力を超えるデータを受け取るため。

3scale が Response/Request Content Limits ポリシーを適用するには、リクエストまたはレスポンスに **content-length** ヘッダーが必要です。

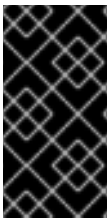
管理ポータルで Response/Request Content Limits ポリシーをプロダクトに追加したら、それをクリックして制限をバイト単位で指定します。リクエストの制限またはレスポンスの制限のいずれか、またはその両方を指定することができます。デフォルト値は **0** で、サイズが無制限であることを意味します。

または、以下のようにポリシーチェーン設定ファイルを更新して、このポリシーを追加することができます。

```
{
  "name": "apicast.policy.limits",
  "configuration":
  {
    "request": 100,
    "response": 100
  }
}
```

4.1.28. Retry

Retry ポリシーは、アップストリーム API へのリトライリクエストの回数を設定します。Retry ポリシーはサービスごとに設定されるため、ユーザーは必要な数のサービスに対してリトライを有効にすることができ、またそれぞれのサービスに異なるリトライ数を設定することもできます。



重要

3scale 2.12 では、リトライするケースをポリシーから設定することはできません。この動作を制御する環境変数 **APICAST_UPSTREAM_RETRY_CASES** は、すべてのサービスにリトライリクエストを適用します。この点についての詳細は、APICAST_UPSTREAM_RETRY_CASES をご確認ください。

Retry ポリシー **JSON** の例を以下に示します。

```
{
  "$schema": "http://apicast.io/policy-v1/schema#manifest#",
  "name": "Retry",
  "summary": "Allows retry requests to the upstream",
  "description": "Allows retry requests to the upstream",
  "version": "builtin",
  "configuration": {
    "type": "object",
    "properties": {
      "retries": {
        "description": "Number of retries",
```

```
        "type": "integer",
        "minimum": 1,
        "maximum": 10
      }
    }
  }
}
```

4.1.29. RH-SSO/Keycloak Role Check

OpenID Connect 認証オプションと共に使用した場合、このポリシーによりロールの確認機能が追加されます。このポリシーは、Red Hat Single Sign-On (RH-SSO) により発行されたアクセストークンのレルムロールおよびクライアントロールを確認します。レルムロールを指定すると、3scale のすべてのクライアントリソースにロールの確認機能が追加されます。

ポリシー設定の **type** プロパティーで指定するロールの確認には、2 つのタイプがあります。

- **whitelist**:これがデフォルトです。**whitelist** が使用されると、APIcast は指定したスコープが JWT トークンに含まれるかどうかを確認し、JWT にそのスコープがなければ呼び出しを拒否します。
- **blacklist**: **blacklist** が使用された場合には、JWT トークンにブラックリスト登録したスコープが含まれていれば APIcast は呼び出しを拒否します。

同じポリシーに **blacklist** および **whitelist** 両方のチェックを設定することはできませんが、APIcast ポリシーチェーンに複数の **RH-SSO/Keycloak Role Check** ポリシーインスタンスを追加することができます。

ポリシー設定の **scopes** プロパティーを使用して、スコープのリストを設定することができます。

それぞれの **scope** オブジェクトには、以下のプロパティーがあります。

- **resource**:ロールによって制御されるリソースエンドポイント。これは、マッピングルールと同じフォーマットです。文字列の先頭からパターンの照合を行います。完全一致の場合には、最後に **$** を追加する必要があります。
- **resource_type**:このプロパティーで、**resource** の値がどのように評価されるかを定義します。
  - プレーンテキストとして (**plain**): **resource** の値をプレーンテキストとして評価します。たとえば **/api/v1/products$**。
  - Liquid テキストとして (**liquid**): **resource** の値に Liquid を使用することを許可します。たとえば、**/resource_{{ jwt.aud }}**は、クライアント ID が含まれるリソースへのアクセスを管理します。
- **methods**: RH-SSO のユーザーロールに基づいて APIcast で許可される HTTP メソッドを一覧表示する場合に、このパラメーターを使用します。たとえば、以下のロールを持つメソッドを許可することができます。
  - /**resource1** にアクセスするための **role1** レルムロール。このレルムロールを持たないメソッドの場合には、**blacklist** を指定する必要があります。
  - /**resource1** にアクセスするための **role1** という **client1** ロール
  - /**resource1** にアクセスするための **role1** および **role2** レルムロール。**realm_roles** でロールを指定します。各ロールのスコープを指定することもできます。

- /**resource1** にアクセスするための、アプリケーションクライアントの **role1** というクライアントロール (アクセストークンの受領者)。クライアントに JSON Web Token (JWT) 情報を指定するには、**liquid** クライアントタイプを使用します。

- /**resource1** にアクセスするための、アプリケーションクライアントのクライアント ID が含まれるクライアントロール (アクセストークンの受領者)。クライアントロールの **name** に JWT 情報を指定するには、**liquid** クライアントタイプを使用します。

- アプリケーションのクライアント ID が含まれるリソースにアクセスするための、**role1** というクライアントロール。**resource** に JWT 情報を指定するには、**liquid** クライアントタイプを使用します。

- **realm_roles**: レルムロールを確認するのに使用します。Red Hat Single Sign-On のレルムロールに関するドキュメントを参照してください。
  レルムロールは、Red Hat Single Sign-On により発行された JWT にあります。

```
"realm_access": {
  "roles": [
    "<realm_role_A>", "<realm_role_B>"
  ]
}
```

  実際のロールは、ポリシーで指定する必要があります。

```
"realm_roles": [
  { "name": "<realm_role_A>" }, { "name": "<realm_role_B>" }
]
```

  realm_roles 配列の各オブジェクトでは、以下のプロパティーを使用することができます。

  - **name**: ロールの名前を指定します。

  - **name_type**: **plain** または **liquid** どちらかの値を指定して、name がどのように評価されるかを定義します。これは **resource_type** の場合と同じように機能します。

- **client_roles**:クライアント namespace に特定のアクセ出力ルがあるかどうかを確認する場合に、**client_roles** を使用します。Red Hat Single Sign-On のクライアントロール に関するドキュメントを参照してください。
  クライアントロールは、JWT の **resource_access** クレームのセクションにあります。

```
"resource_access": {
  "<client_A>": {
    "roles": [
      "<client_role_A>", "<client_role_B>"
    ]
  },
  "<client_B>": {
    "roles": [
      "<client_role_A>", "<client_role_B>"
    ]
  }
}
```

  ポリシーでクライアントロールを指定します。

```
"client_roles": [
  { "name": "<client_role_A>", "client": "<client_A>" },
  { "name": "<client_role_B>", "client": "<client_A>" },
  { "name": "<client_role_A>", "client": "<client_B>" },
  { "name": "<client_role_B>", "client": "<client_B>" }
]
```

client_roles 配列の各オブジェクトでは、以下のプロパティーを使用することができます。

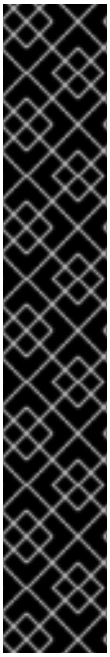
- **name**: ロールの名前を指定します。
- **name_type**: **plain** または **liquid** どちらかの値を指定して、**name** の値がどのように評価されるかを定義します。これは **resource_type** の場合と同じように機能します。
- **client**: ロールのクライアントを指定します。定義しなければ、このポリシーはクライアントに **aud** クレームを使用します。
- **client_type**: **plain** または **liquid** どちらかの値を指定して、**client** の値がどのように評価されるかを定義します。これは **resource_type** の場合と同じように機能します。

4.1.30. Routing

Routing ポリシーを使用すると、リクエストをさまざまなターゲットエンドポイントにルーティングすることができます。ターゲットエンドポイントを定義すると、正規表現を使用して UI から受信したリクエストをターゲットエンドポイントにルーティングできるようになります。

ルーティングは、以下のルールに基づきます。

- リクエストパスルール
- ヘッダールール
- クエリー引数ルール
- JSON Web Token (JWT) クレームルール



重要

Routing ポリシーをポリシーチェーンに追加する場合、Routing ポリシーは常に標準の 3scale APIcast ポリシーの直前に指定する必要があります。つまり、Routing ポリシーと 3scale APIcast ポリシー間にポリシーを設定することはできません。これにより、APIcast がアップストリーム API に送信するリクエストで、APIcast の出力が正しくなります。正しいポリシーチェーンの例を以下に 2 つ示します。

```
Liquid Context Debug
JWT Claim Check
Routing
3scale APIcast
```

```
Liquid Context Debug
Routing
3scale APIcast
JWT Claim Check
```

ルーティングルール

- 複数のルールが存在する場合には、Routing ポリシーは最初のマッチに適用されます。これらのルールは並べ替えることができます。
- マッチするルールがなければ、ポリシーはアップストリームを変更せず、サービス設定で定義済みのプライベートベース URL を使用します。

リクエストパスルール

以下の設定では、パスが **/accounts** の場合に **http://example.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
            }
          ]
        }
      }
    ]
  }
}
```

ヘッダールール

以下の設定では、ヘッダー **Test-Header** の値が **123** の場合に **http://example.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "header",
              "header_name": "Test-Header",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
```

```
    ]
  }
}
```

クエリー引数ルール

以下の設定では、クエリー引数 **test_query_arg** の値が **123** の場合に **http://example.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "query_arg",
              "query_arg_name": "test_query_arg",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
    ]
  }
}
```

JWT クレームルール

JWT クレームの値に基づいてルーティングするためには、ポリシーチェーンには、ポリシーが共有するコンテキストで JWT を検証して格納するポリシーが必要です。

以下の設定では、JWT クレーム **test_claim** の値が **123** の場合に **http://example.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "jwt_claim",
              "jwt_claim_name": "test_claim",
              "op": "==",
              "value": "123"
            }
          ]
        }
```

```
      }
    ]
  }
}
```

複数演算ルール

ルールに複数の演算を設定し、and**combine_op** を使用してすべてが true と評価される場合にだけ指定したアップストリームにルーティングすることや、or**combine_op** を使用して少なくとも 1 つが true と評価される場合に指定したアップストリームにルーティングすることができます。**combine_op** のデフォルト値は and です。

以下の設定では、リクエストのパスが /**accounts** で、かつヘッダー **Test-Header** の値が **123** の場合に、**http://example.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "combine_op": "and",
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
            },
            {
              "match": "header",
              "header_name": "Test-Header",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
    ]
  }
}
```

以下の設定では、リクエストのパスが /**accounts** であるか、またはヘッダー **Test-Header** の値が **123** の場合に、**http://example.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "combine_op": "or",
          "operations": [
```

```
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
            },
            {
              "match": "header",
              "header_name": "Test-Header",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
    ]
  }
}
```

ルールの結合

ルールを組み合わせることができます。複数のルールがある場合、選択されるアップストリームは、true と評価される最初のルールです。

複数のルールで設定される設定を以下に示します。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://some_upstream.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
            }
          ]
        }
      },
      {
        "url": "http://another_upstream.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/users"
            }
          ]
        }
      }
```

```
    ]
   }
  }
```

キャッチオールルール

演算のないルールは常にマッチします。これは、キャッチオールルールを定義するのに役立ちます。

以下の設定では、パスが **/abc** の場合にはリクエストを **http://some_upstream.com** にルーティングし、パスが **/def** の場合には **http://another_upstream.com** にルーティングし、上記ルールのどちらも true と評価されない場合には **http://default_upstream.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://some_upstream.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/abc"
            }
          ]
        }
      },
      {
        "url": "http://another_upstream.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/def"
            }
          ]
        }
      },
      {
        "url": "http://default_upstream.com",
        "condition": {
          "operations": []
        }
      }
    ]
  }
}
```

サポートされる操作

サポートされる演算は、**==**、**!=**、および **matches** です。最後の演算は正規表現を使用する文字列との照合を行い、ngx.re.match を使用して実装されます。

以下の設定では、**!=** が使用されています。パスが **/accounts** ではない場合に **http://example.com** にルーティングします。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "!=",
              "value": "/accounts"
            }
          ]
        }
      }
    ]
  }
}
```

Liquid テンプレート

設定の値に liquid テンプレートを使用することができます。これにより、チェーン内のポリシーがコンテキストにキー **my_var** を保存する場合には、動的な値を使用してルールを定義することができます。

以下の設定では、リクエストをルーティングするのにその値が使用されています。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "header",
              "header_name": "Test-Header",
              "op": "==",
              "value": "{{ my_var }}",
              "value_type": "liquid"
            }
          ]
        }
      }
    ]
  }
}
```

host_header で使用されるホストの設定

リクエストがルーティングされる際に、デフォルトでは、ポリシーはマッチしたルールの URL のホストを使用して Host ヘッダーを設定します。**host_header** 属性を使用して別のホストを指定することができます。

以下の設定では、Host ヘッダーのホストに **some_host.com** が指定されています。

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "host_header": "some_host.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.31. SOAP

SOAP ポリシーでは、ポリシーで指定したマッピングルールを使用して、HTTP リクエストの SOAPAction または Content-Type ヘッダーにより提供される SOAP アクション URI との照合を行います。

設定プロパティー

プロパティー	説明	値	必須/任意
pattern	**pattern** プロパティーにより、APIcast がマッチを探す SOAPAction URI の文字列を指定することができます。	データタイプ: 文字列	必須
metric_system_name	**metric_system_name** プロパティーを使用して、マッチしたパターンがヒットを登録する 3scale バックエンドメトリクスを指定することができます。	データタイプ: 文字列 (有効な メトリクス でなければなりません)	必須

ポリシーオブジェクトの例

```
{
  "name": "soap",
  "version": "builtin",
  "configuration": {
    "mapping_rules": [
      {
        "pattern": "http://example.com/soap#request",
        "metric_system_name": "soap",
        "delta": 1
      }
    ]
  }
}
```

ポリシーの設定方法に関する情報は、このドキュメントの 3scale でのポリシーチェーンの作成 セクションを参照してください。

4.1.32. TLS Client Certificate Validation

TLS Client Certificate Validation ポリシーでは、APIcast は TLS ハンドシェイクを実装し、ホワイトリストに対してクライアント証明書を検証します。ホワイトリストには、認証局 (CA) によって署名された証明書、または単なるクライアント証明書が含まれます。証明書の有効期限が切れている、または証明書が無効な場合には、リクエストは拒否され他のポリシーは処理されません。

クライアントは APIcast に接続してリクエストを送信し、クライアント証明書を提供します。APIcast は、ポリシー設定に従って受信したリクエストで提供された証明書の信ぴょう性を検証します。アップストリームに接続する際に独自のクライアント証明書を使用するように、APIcast を設定することもできます。

TLS Client Certificate Validation ポリシーに対応する APIcast の設定

TLS を終端するように APIcast を設定する必要があります。以下の手順に従って、Client Certificate Validation ポリシーが設定された APIcast でユーザーが提供するクライアント証明書を検証するように設定します。

3scale システムにアクセスできること。すべてのデプロイメントが完了していること。

ポリシーに対応する APIcast のセットアップ

APIcast をセットアップし TLS を終端するように設定するには、以下の手順に従います。

1. OpenShift テンプレートを使用した APIcast のデプロイ に記載の手順に従って、アクセストークンを取得し Self-managed APIcast をデプロイする必要があります。

注記

ゲートウェイ全体で特定の証明書を使用するように APIcast インスタンスを再設定しなければならないので、Self-managed APIcast のデプロイメントが必要です。

2. テストを簡素化するために、**--param** フラグを指定してレイジーローダー、キャッシュなし、およびステージング環境を使用することができます (ただし、テスト目的の場合のみ)。

```
oc new-app -f https://raw.githubusercontent.com/3scale/3scale-amp-openshift-templates/master/apicast-gateway/apicast.yml --param CONFIGURATION_LOADER=lazy --param DEPLOYMENT_ENVIRONMENT=staging --param CONFIGURATION_CACHE=0
```

3. テスト用途の証明書を生成します。なお、実稼働環境のデプロイメントの場合には、認証局から提供された証明書を使用することができます。

4. TLS 証明書を使用してシークレットを作成します。

```
oc create secret tls apicast-tls
--cert=ca/certs/server.crt
--key=ca/keys/server.key
```

5. APIcast デプロイメント内にシークレットをマウントします。

```
oc set volume dc/apicast --add --name=certificates --mount-path=/var/run/secrets/apicast --secret-name=apicast-tls
```

6. HTTPS 用ポート 8443 のリッスンを開始するように APIcast を設定します。

```
oc set env dc/apicast APICAST_HTTPS_PORT=8443
APICAST_HTTPS_CERTIFICATE=/var/run/secrets/apicast/tls.crt
APICAST_HTTPS_CERTIFICATE_KEY=/var/run/secrets/apicast/tls.key
```

7. サービスでポート 8443 を公開します。

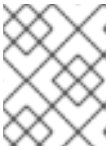
```
oc patch service apicast -p '{"spec":{"ports":[{"name":"https","port":8443,"protocol":"TCP"}]}}'
```

8. デフォルトルートを削除します。

```
oc delete route api-apicast-staging
```

9. **apicast** サービスをルートとして公開します。

```
oc create route passthrough --service=apicast --port=https --hostname=api-3scale-apicast-staging.$WILDCARD_DOMAIN
```



注記

このステップは、使用するすべての API で実施する必要があります (それぞれの API のドメインに変更してください)。

10. プレースホルダーの [Your_user_key] に実際のキーを指定して、上記のステップでデプロイしたゲートウェイが機能し、設定が保存されていることを確認します。

```
curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN?user_key=[Your_user_key] -v --cacert ca/certs/ca.crt
```

ポリシーチェーンへの TLS Client Certificate Validation ポリシーの設定

ポリシーチェーンで TLS Client Certificate Validation を設定するには、3scale のログインクレデンシャルが必要です。TLS Client Certificate Validation ポリシーに対応するように APIcast を設定 している必要もあります。

1. 3scale 管理ポータルでのポリシーの有効化に記載の手順に従って TLS Client Certificate Validation を選択し、API に TLS Client Certificate Validation ポリシーを追加します。
2. **TLS Client Certificate Validation** のリンクをクリックします。
3. **Enabled** のチェックボックスを選択し、ポリシーを有効にします。
4. 証明書をホワイトリストに追加するには、プラス (**+**) アイコンをクリックします。
5. **-----BEGIN CERTIFICATE-----** および **-----END CERTIFICATE-----** を含めて、証明書を指定します。
6. API への TLS Client Certificate Validation ポリシーの設定が完了したら、**Update Policy** をクリックします。

付加手順:

- プラス (**+**) アイコンをクリックして、さらに証明書を追加することができます。
- 上/下矢印を使用して、証明書の順番を入れ替えることもできます。

変更を保存するには、**Update Policy Chain** をクリックします。

TLS Client Certificate Validation ポリシー機能の確認

TLS Client Certificate Validation ポリシーの機能を確認するには、3scale のログインクレデンシャルが必要です。TLS Client Certificate Validation ポリシーに対応するように APIcast を設定 している必要もあります。

プレースホルダーの **[Your_user_key]** に実際のキーを指定して、適用したポリシーを確認することができます。

```
curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN\?user_key\=[Your_user_key] -v --cacert ca/certs/ca.crt --cert ca/certs/client.crt --key ca/keys/client.key

curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN\?user_key\=[Your_user_key] -v --cacert ca/certs/ca.crt --cert ca/certs/server.crt --key ca/keys/server.key

curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN\?user_key\=[Your_user_key] -v --cacert ca/certs/ca.crt
```

ホワイトリストからの証明書の削除

ホワイトリストから証明書を削除するには、3scale のログインクレデンシャルが必要です。TLS Client Certificate Validation ポリシーに対応するように APIcast を設定 している必要があります。ポリシーチェーンに TLS Client Certificate Validation ポリシーを設定 して、証明書をホワイトリストに追加している必要があります。

1. **TLS Client Certificate Validation** のリンクをクリックします。
2. **x** アイコンをクリックし、ホワイトリストから証明書を削除します。
3. 証明書の削除が完了したら、**Update Policy** をクリックします。

変更を保存するには、**Update Policy Chain** をクリックします。

証明書の操作についての詳細は、Red Hat Certificate System のドキュメント を参照してください。

4.1.33. TLS Termination

本セクションでは、Transport Layer Security (TLS) Termination ポリシーに関して、その概要、設定、検証、およびポリシーからのファイル削除について説明します。

TLS Termination ポリシーを使用すると、全 API 用の単一の証明書を使用せずにそれぞれの API ごとに TLS リクエストを終端するように、APIcast を設定することができます。APIcast は、クライアントへの接続を確立する前に設定設定をプルします。このようにして、APIcast はポリシーからの証明書を使用して TLS を終端させます。このポリシーは、以下のソースと共に機能します。

- ポリシー設定内に保管されるソース
- ファイルシステム上に保管されるソース

デフォルトでは、このポリシーはポリシーチェーンで有効になっていません。

ポリシーチェーンへの TLS Termination ポリシーの設定

本セクションでは、ポリシーチェーンに TLS Termination ポリシーを設定するための前提条件および手順について説明します。なお、設定には Privacy Enhanced Mail (PEM) 形式の証明書を使用します。前提条件は以下のとおりです。

- ユーザーの発行した証明書
- PEM 形式のサーバー証明書
- PEM 形式の証明書の秘密鍵

以下の手順に従います。

1. 管理ポータルでのポリシーの有効化 に記載の手順に従って TLS Termination を選択し、API に TLS Termination ポリシーを追加します。
2. **TLS Termination** のリンクをクリックします。
3. **Enabled** のチェックボックスを選択し、ポリシーを有効にします。
4. TLS 証明書をポリシーに追加するには、プラス (**+**) アイコンをクリックします。
5. 証明書のソースを選択します。
    - デフォルトでは、**Embedded certificate** が選択されます。これらの証明書をアップロードします。
        - **PEM 形式の証明書の秘密鍵**: **Browse** をクリックして選択およびアップロードします。
        - **PEM 形式の証明書**: **Browse** をクリックして選択およびアップロードします。
    - **Certificate from filesystem** - これらの証明書パスを選択して指定します。
        - **証明書へのパス**
        - **証明書の秘密鍵へのパス**

6. API への TLS Termination ポリシーの設定が完了したら、**Update Policy** をクリックします。

付加手順:

- プラス (**+**) アイコンをクリックして、さらに証明書を追加することができます。
- 上/下矢印を使用して、証明書の順番を入れ替えることもできます。

変更を保存するには、**Update Policy Chain** をクリックします。

TLS Termination ポリシー機能の確認

3scale のログインクレデンシャルが必要です。APIcast に TLS Termination ポリシーを設定 している必要があります。

以下のコマンドを使用して、ポリシーが機能するかどうかをコマンドラインでテストすることができます。

```
curl “${public_URL}:${port}/?user_key=${user_key}" --cacert ${path_to_certificate}/ca.pem -v
```

ここで、

- **public_URL**: ステージング環境用の公開ベース URL
- **port**: ポート番号
- **user_key**: 認証に使用するユーザーキー
- **path_to_certificate**: ローカルファイルシステムに保管された CA 証明書へのパス

TLS Termination ポリシーからのファイルの削除

本セクションでは、TLS Termination ポリシーから証明書およびキーファイルを削除する手順について説明します。

- 3scale へのログインクレデンシャルが必要です。
- APIcast に TLS Termination ポリシーを設定 して、証明書をポリシーに追加している必要があります。

証明書を削除するには、以下の手順を実施します。

1. **TLS Termination** のリンクをクリックします。
2. **x** アイコンをクリックして、証明書およびキーを削除します。
3. 証明書の削除が完了したら、**Update Policy** をクリックします。

変更を保存するには、**Update Policy Chain** をクリックします。

4.1.34. Upstream

Upstream ポリシーでは、正規表現を使用して Host リクエストヘッダーを解析し、プライベートベース URL で定義されたアップストリーム URL を別の URL に置き換えることができます。

例:

正規表現 **/foo** および URL フィールド **newexample.com** が設定されたポリシーが、URL **https://www.example.com/foo/123/** を **newexample.com** に置き換える

ポリシーチェーンの参照

プロパティー	説明	値	必須/任意
regex	**regex** プロパティーを使用して、Upstream ポリシーがリクエストパスを照合する際に使用する正規表現を指定することができます。	データタイプ: 文字列 (有効な正規表現の構文でなければなりません)	必須
url	**url** プロパティーを使用して、マッチした場合に置き換える URL を指定することができます。Upstream ポリシーはこの URL が有効かどうかを確認しない点に注意してください。	データタイプ: 文字列 (有効な URL でなければなりません)	必須

ポリシーオブジェクトの例

```
{
  "name": "upstream",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "regex": "^/v1/.*",
        "url": "https://api-v1.example.com",

      }
    ]
  }
}
```

ポリシーの設定方法に関する情報は、本書の 3scale でのポリシーチェーンの作成 セクションを参照してください。

4.1.35. Upstream Connection

Upstream Connection ポリシーを使用すると、3scale システムでの API バックエンドサーバーの設定に応じて、API ごとに以下のディレクティブのデフォルト値を変更することができます。

- **proxy_connect_timeout**
- **proxy_send_timeout**
- **proxy_read_timeout**

Upstream Connection ポリシーを設定するには、以下の条件を満たす必要があります。

- 3scale システムにアクセスできること。
- すべてのデプロイメントが完了していること。

以下の手順に従います。

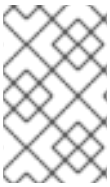
1. 3scale 管理ポータルでのポリシーの有効化に記載の手順に従って **Upstream Connection** を選択し、API に Upstream Connection ポリシーを追加します。
2. **Upstream Connection** のリンクをクリックします。
3. **Enabled** のチェックボックスを選択し、ポリシーを有効にします。
4. アップストリームへの接続に関するオプションを設定します。
    - **send_timeout**
    - **connect_timeout**
    - **read_timeout**
5. API への Upstream Connection ポリシーの設定が完了したら、**Update Policy** をクリックします。

変更を保存するには、**Update Policy Chain** をクリックします。

4.1.36. Upstream Mutual TLS

Upstream Mutual TLS ポリシーを使用すると、設定で指定した証明書に基づいて、APIcast とアップストリーム API との間で相互 TLS 接続を確立して検証することができます。

verify フィールドを有効にすると、ポリシーはアップストリーム API からのサーバー証明書も検証します。**ca_certificates** には、APIcast がサーバーの検証に使用する **-----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----** を含めた Privacy Enhanced Mail (PEM) 形式の証明書が含まれます。



注記

アップストリーム API の証明書の検証を行うには、**verify** フィールドを有効にし、**ca_certificates** を入力する必要があります。**verify** フィールドが有効になっていない場合は、アップストリーム API での APIcast 証明書のチェックのみが行われます。

ポリシーチェーンで Upstream Mutual TLS を設定するには、3scale システムにアクセスできる必要があります。

1. 3scale 管理ポータルでのポリシーの有効化に記載の手順に従って **Upstream Mutual TLS** を選択し、API に Upstream Mutual TLS ポリシーを追加します。
2. **Upstream Mutual TLS** のリンクをクリックします。
3. **Enabled** のチェックボックスを選択し、ポリシーを有効にします。
4. **Certificate type** を選択します。
    - **path**: OpenShift によって生成された証明書などのパスを指定する場合。

- **embedded**:サードパーティーが生成した証明書をファイルシステムからアップロードして使用する場合。

5. **Certificate** でクライアント証明書を指定します。
6. **Certificate key** でキーを指定します。
7. API への Upstream Mutual TLS ポリシーの設定が完了したら、**Update Policy Chain** をクリックします。

変更をプロモートするには、以下の手順を実施します。

1. **[Your_product] page > Integration > Configuration**の順に移動します。
2. **APIcast Configuration** セクションで、**Promote v# to Staging APIcast**をクリックします。

v# は、プロモートされる設定のバージョン番号を表します。

パス設定

以下のように、OpenShift および Kubernetes シークレットの証明書パスを使用します。

```
{
  "name": "apicast.policy.upstream_mtls",
  "configuration": {
      "certificate": "/secrets/client.cer",
      "certificate_type": "path",
      "certificate_key": "/secrets/client.key",
      "certificate_key_type": "path"
  }
}
```

組み込み設定

http フォームとファイルアップロードには、以下の設定を使用します。

```
{
  "name": "apicast.policy.upstream_mtls",
  "configuration": {
    "certificate_type": "embedded",
    "certificate_key_type": "embedded",
    "certificate": "data:application/pkix-cert;name=client.cer;base64,XXXXXXXXXXX",
    "certificate_key": "data:application/x-iwork-keynote-sffkey;name=client.key;base64,XXXXXXXX"
  }
}
```

Upstream Mutual TLS に関する追加のフィールド、**ca_certificates**、および **verify** についての詳細は、ポリシー設定スキーマ を参照してください。

その他の考慮事項

Upstream mutual TLS ポリシーは、**APICAST_PROXY_HTTPS_CERTIFICATE_KEY** および **APICAST_PROXY_HTTPS_CERTIFICATE** 環境変数の値を上書きします。ポリシーで設定された証明書を使用するため、これらの環境変数は影響を受けません。

4.1.37. URL Rewriting

URL Rewriting ポリシーを使用すると、リクエストのパスおよびクエリー文字列を変更することができます。

3scale APIcast ポリシーと組み合わせる場合には、ポリシーチェーンで URL Rewriting ポリシーを APIcast ポリシーの前に設定すると、APIcast のマッピングルールは変更したパスに適用されます。ポリシーチェーンで URL Rewriting ポリシーを APIcast ポリシーの後に設定すると、マッピングルールは元のパスに適用されます。

このポリシーでは、以下の 2 つの操作セットがサポートされます。

- **commands**:リクエストのパスを書き換えるために適用されるコマンドのリスト。
- **query_args_commands**:リクエストのクエリー文字列を書き換えるために適用されるコマンドのリスト。

パスを書き換えるためのコマンド

commands リストの各コマンドは、以下の設定パラメーターで設定されます。

- **op**:適用される操作。設定可能なオプションは **sub** および **gsub** です。**sub** 操作では、指定した正規表現との最初のマッチだけが置き換えられます。**gsub** 操作では、指定した正規表現とのマッチがすべて置き換えられます。sub および gsub 操作に関するドキュメントを参照してください。
- **regex**:照合される Perl 互換の正規表現
- **replace**:マッチした際に置き換えられる文字列
- **options**:これは任意です。正規表現との照合がどのように行われるかを定義するオプション。設定可能なオプションに関する情報は、OpenResty Lua モジュールプロジェクトのドキュメントの ngx.re.match セクションを参照してください。
- **break**:これは任意です。チェックボックスを選択して true に設定すると、コマンドが URL を書き換えた場合、それが適用される最後のコマンドになり、リスト内の後続コマンドはすべて破棄されます。

クエリー文字列を書き換えるためのコマンド

query_args_commands リストの各コマンドは、以下の設定パラメーターで設定されます。

- **op**:クエリー引数に適用される操作。以下のオプションを設定できます。
  - **add**:既存の引数に値を追加します。
  - **set**:引数が設定されていなければ作成し、設定されていればその値を置き換えます。
  - **push**:引数が設定されていなければ作成し、設定されていれば値を追加します。
  - **delete**:引数を削除します。
- **arg**:操作が適用されるクエリー引数の名前
- **value**:クエリー引数に使用される値を指定します。値のタイプが liquid の場合には、値は **{{ variable_from_context }}** の形式にする必要があります。**delete** 操作の場合には、値を考慮する必要はありません。
- **value_type**:これは任意です。クエリー引数の値がどのように評価されるかを定義します。プレーンテキストの場合の **plain** または Liquid テンプレートとして評価する場合の **liquid** いずれ

かに設定します。詳細は、「ポリシーでの変数およびフィルターの使用」 を参照してください。指定しなければ、デフォルトではタイプ plain が使用されます。

例

URL Rewriting ポリシーは、以下のように設定します。

```
{
  "name": "url_rewriting",
  "version": "builtin",
  "configuration": {
    "query_args_commands": [
      {
        "op": "add",
        "arg": "addarg",
        "value_type": "plain",
        "value": "addvalue"
      },
      {
        "op": "delete",
        "arg": "user_key",
        "value_type": "plain",
        "value": "any"
      },
      {
        "op": "push",
        "arg": "pusharg",
        "value_type": "plain",
        "value": "pushvalue"
      },
      {
        "op": "set",
        "arg": "setarg",
        "value_type": "plain",
        "value": "setvalue"
      }
    ],
    "commands": [
      {
        "op": "sub",
        "regex": "^/api/v\\d+/",
        "replace": "/internal/",
        "options": "i"
      }
    ]
  }
```

APIcast に送信される元のリクエスト URI:

```
https://api.example.com/api/v1/products/123/details?
user_key=abc123secret&pusharg=first&setarg=original
```

URL の書き換えを適用した後に APIcast が API バックエンドに送信する URI:

```
https://api-backend.example.com/internal/products/123/details?
pusharg=first&pusharg=pushvalue&setarg=setvalue
```

以下の変換が適用されます。

1. サブストリング **/api/v1/** はパス書き換えコマンドだけにマッチし、**/internal/** に置き換えられます。
2. **user_key** クエリー引数は削除されます。
3. 値 **pushvalue** は追加の値として **pusharg** クエリー引数に追加されます。
4. クエリー引数 **setarg** の値 **original** は、設定した値 **setvalue** に置き換えられます。
5. クエリー引数 **addarg** は元の URL に存在しないため、コマンド **add** は適用されませんでした。

ポリシーの設定方法に関する情報は、このドキュメントの 3scale でのポリシーチェーンの作成 セクションを参照してください。

4.1.38. URL Rewriting with Captures

URL Rewriting with Captures ポリシーは URL Rewriting ポリシーの代替で、API リクエストの URL を API バックエンドに渡す前に書き換えることができます。

URL Rewriting with Captures ポリシーは URL の引数を取得し、その値を書き換えられる URL で使用します。

このポリシーでは **transformations** 設定パラメーターがサポートされます。これは、リクエスト URL に適用される変換を定義するオブジェクトのリストです。それぞれの変換オブジェクトは、以下の 2 つのプロパティーで設定されます。

- **match_rule**:このルールは受信したリクエストの URL と照合されます。このルールには **{nameOfArgument}** 形式の名前付き引数を含めることができ、これらの引数を書き換えられる URL で使用することができます。URL は正規表現として **match_rule** と比較されます。名前付き引数と照合する値には、**[\w-.~%!$&'()*+,;=@:]+** の文字しか使用することができません (PCRE 正規表現表記)。他の正規表現トークンを **match_rule** の表記で使用することができます。たとえば、文字列先頭の **^** および文字列末尾の **$** 等です。
- **template**: URL のテンプレートで、これにより元の URL が書き換えられます。 **match_rule** からの名前付き引数を使用することができます。

元の URL のクエリーパラメーターは、**template** で指定したクエリーパラメーターとマージされます。

例

URL Rewriting with Captures ポリシーは、以下のように設定します。

```
{
  "name": "rewrite_url_captures",
  "version": "builtin",
  "configuration": {
    "transformations": [
      {
        "match_rule": "/api/v1/products/{productId}/details",
        "template": "/internal/products/details?id={productId}&extraparam=anyvalue"
```

```
      }
    ]
  }
}
```

APIcast に送信される元のリクエスト URI:

```
https://api.example.com/api/v1/products/123/details?user_key=abc123secret
```

URL の書き換えを適用した後に APIcast が API バックエンドに送信する URI:

```
https://api-backend.example.com/internal/products/details?
user_key=abc123secret&extraparam=anyvalue&id=123
```

4.1.39. Websocket

Websocket ポリシーは、アップストリーム API への WebSocket プロトコル接続を有効にします。WebSocket プロトコルを有効にする予定の場合は、以下を考慮してください。

- WebSocket プロトコルは JSON Web Token をサポートしません。
- WebSocket プロトコルは追加のヘッダーを許可しません。
- WebSocket プロトコルは HTTP/2 標準に含まれない。

WebSocket 接続を有効にするアップストリーム API については、そのバックエンドを **http[s]** または **ws[s]** として定義できます。

Websocket ポリシーをポリシーチェーンに追加する場合は、3scale APIcast ポリシーの前に Websocket ポリシーが設定されていることを確認してください。

4.2. 3SCALE 標準ポリシーからのポリシーチェーン

API プロダクトごとにポリシーチェーンを指定することができます。ポリシーチェーンは以下を行います。

- APIcast がリクエストに適用するポリシーを指定する。
- ポリシーの設定情報を提供する。
- APIcast がポリシーを適用する順番を決定する。

チェーン内でポリシーを正しい順番で設定するには、APIcast がポリシーを API 利用者のリクエストに適用する方法を理解することが重要です。

4.2.1. APIcast NGINX フェーズが 3scale ポリシーを処理する仕組み

3scale API ゲートウェイ (APIcast) は、NGINX プロキシー Web サーバーを使用してポリシーを適用します。APIcast が API 利用者からリクエストを受け取ると、APIcast は要求を一連の NGINX フェーズの順番どおりに処理します。それぞれの NGINX フェーズでは、APIcast は以下のポリシーを適用して元のリクエストを変更することができます。

- アップストリーム API ポリシーチェーンのポリシー。ポリシーチェーンは、順番が設定されたポリシーのリストです。デフォルトでは、アップストリーム API のポリシーチェーンには

3scale APIcast ポリシーが含まれます。API プロバイダーは、3scale プロダクトのポリシーチェーンにポリシーを追加することができます。APIcast は、アップストリーム API ポリシーチェーンのポリシーを、そのアップストリーム API に送信された API 利用者のリクエストだけに適用します。

- グローバル 3scale ポリシーチェーンのポリシー。API プロバイダーは、3scale の環境変数を設定して、グローバルポリシーチェーンを更新することができます。APIcast は、グローバルポリシーチェーンのポリシーをすべての API 利用者のリクエストに適用します。

同じポリシーがアップストリーム API のポリシーチェーンおよびグローバルポリシーチェーンにある場合は、アップストリーム API のポリシーチェーンのポリシー設定が優先されます。

APIcast がすべての NGINX フェーズの必要な処理を行った後に、APIcast はリクエストの結果をアップストリーム API に送信します。そのため、処理によって API 利用者のリクエストを変更できるため、希望の動作を実現するには、NGINX フェーズがポリシーを処理する順序を理解することが重要です。

NGINX フェーズの順序と説明

APIcast が API 利用者からリクエストを受け取ると、APIcast はアップストリーム API のポリシーチェーンおよびグローバルポリシーチェーンのポリシーを適用してリクエストを処理します。各 3scale ポリシーは 1 つ以上の関数を定義します。APIcast は、一連の NGINX フェーズの順番どおりにポリシーの関数を実行します。各フェーズでは、NGINX は適用されるポリシーで定義され、そのフェーズでの実行を指定する関数を実行します。以下の表は、ポリシー関数を実行する NGINX のフェーズのリストを示しています。この表に記載されていない追加の NGINX フェーズは、ポリシーチェーン内のポリシーの順序によって影響を受けない処理を実行します。

NGINX フェーズのリスト	このフェーズでの処理の説明
rewrite	要求のターゲット URI を変更する関数を実行します。
access	リクエストを行うためのクライアントの承認を検証する関数を実行します。
content	アップストリーム API に送信されるリクエストのコンテンツを生成します。 NGINX は、**content** フェーズ内の 1 つのポリシーのみを適用します。ポリシーチェーン内の複数のポリシーがリクエストのコンテンツで動作する場合、NGINX はチェーン内で最も早いポリシーのみを適用します。組み込みの **3scale APIcast** ポリシーは常にポリシーチェーンにあり、**content** フェーズで NGINX の処理が必要になるため、これを理解することが重要です。 たとえば、**3scale APIcast** ポリシーと **Upstream** ポリシーの両方が、リクエストを更新してアップストリーム API のパスを指定します。NGINX は、これらの関数を **content** フェーズで処理します。**3scale APIcast** ポリシーが **Upstream** ポリシーの前にある場合、NGINX はアップストリーム API の設定を使用して、そのパスを修正したリクエストに追加します。**Upstream** ポリシーが **3scale APIcast** ポリシーの前にある場合には、NGINX は **Upstream** ポリシーの式を評価します。一致がある場合、NGINX は修正したリクエストで適宜アップストリーム API のパスを変更します。
balancer	負荷分散関数を実行します。
header_filter	リクエストヘッダーを処理する関数を実行します。

NGINX フェーズのリスト	このフェーズでの処理の説明
body_filter	リクエストボディーを処理する関数を実行します。
post_action	NGINX がヘッダーとボディーで関数を実行した後にリクエストを処理する関数を実行します。
log	リクエストに関するログ情報を生成します。
metrics	Prometheus エンドポイントから受信されるデータで機能します。

ポリシーの順序によって影響を受けない処理を実行する NGINX フェーズの例:

- APIcast が起動すると、NGINX は **init** フェーズに関連付けられたタスクを実行します。
- APIcast ワーカーが起動すると、NGINX は **init_worker** フェーズに関連付けられたタスクを実行します。
- APIcast が HTTPS 接続を終了すると、NGINX は **ssl_certificate** フェーズに関連付けられたタスクを実行します。

NGINX がポリシー関数を実行する順序

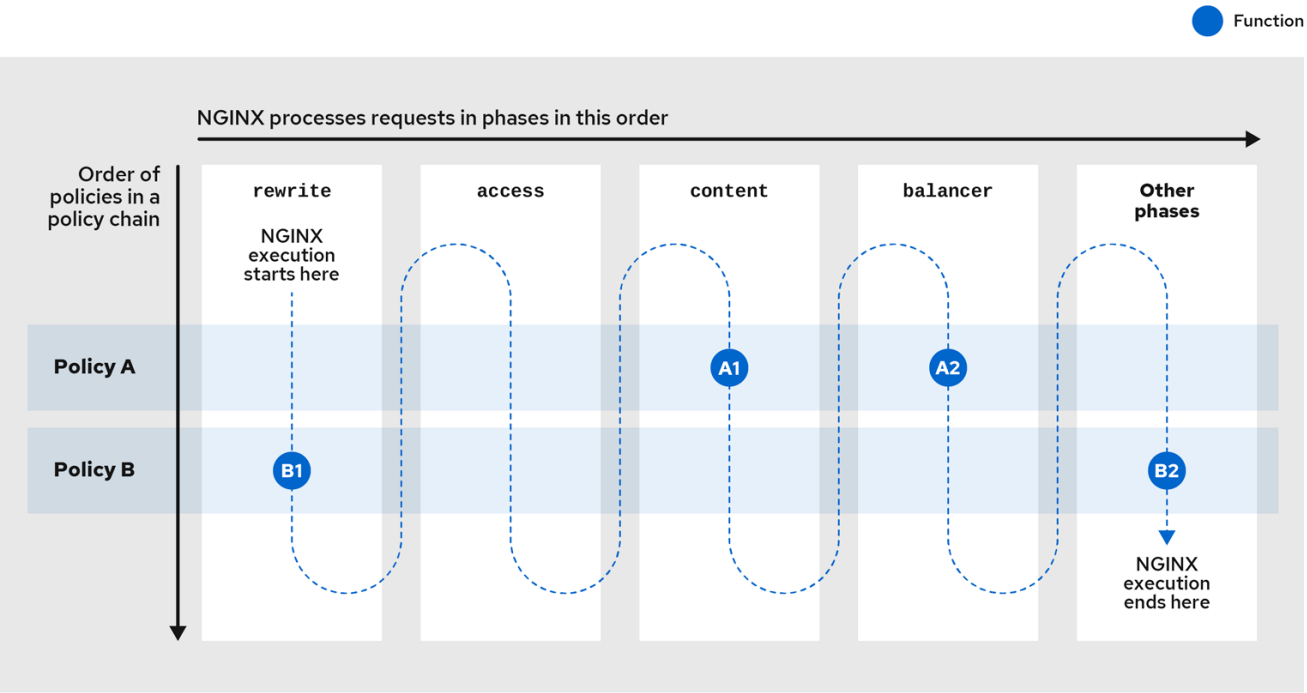
API プロバイダーは、1 つ以上のポリシーを 3scale プロダクトに追加してポリシーチェーンを形成できます。各フェーズでは、NGINX はそのフェーズでの実行を指定するポリシー関数のみを処理します。各ポリシー関数は、1 つの NGINX フェーズでの処理中に APIcast がそのデフォルトの動作をどのように変更するかを指定します。たとえば、**header_filter** フェーズでは、NGINX は **header_filter** を指定し、リクエストヘッダーで操作する関数を処理します。各フェーズでは、NGINX は、ポリシーチェーンにある順序で該当する関数を処理します。

ポリシーは、**context** オブジェクトを使用してデータを共有できます。ポリシーは、各フェーズで **context** オブジェクトを読み取ったり、変更したりできます。

NGINX がポリシー関数を実行する順序は、以下の要素によって異なります。

- ポリシーチェーンでのポリシーの位置
- 特定のポリシー関数を処理する NGINX フェーズ

必要な動作を得るには、ポリシーチェーンの順序を正しく指定する必要があります。ポリシー適用の結果は、ポリシーチェーン内の位置により異なる可能性があるためです。以下の図は、NGINX がポリシーを適用する順序の例を示しています。

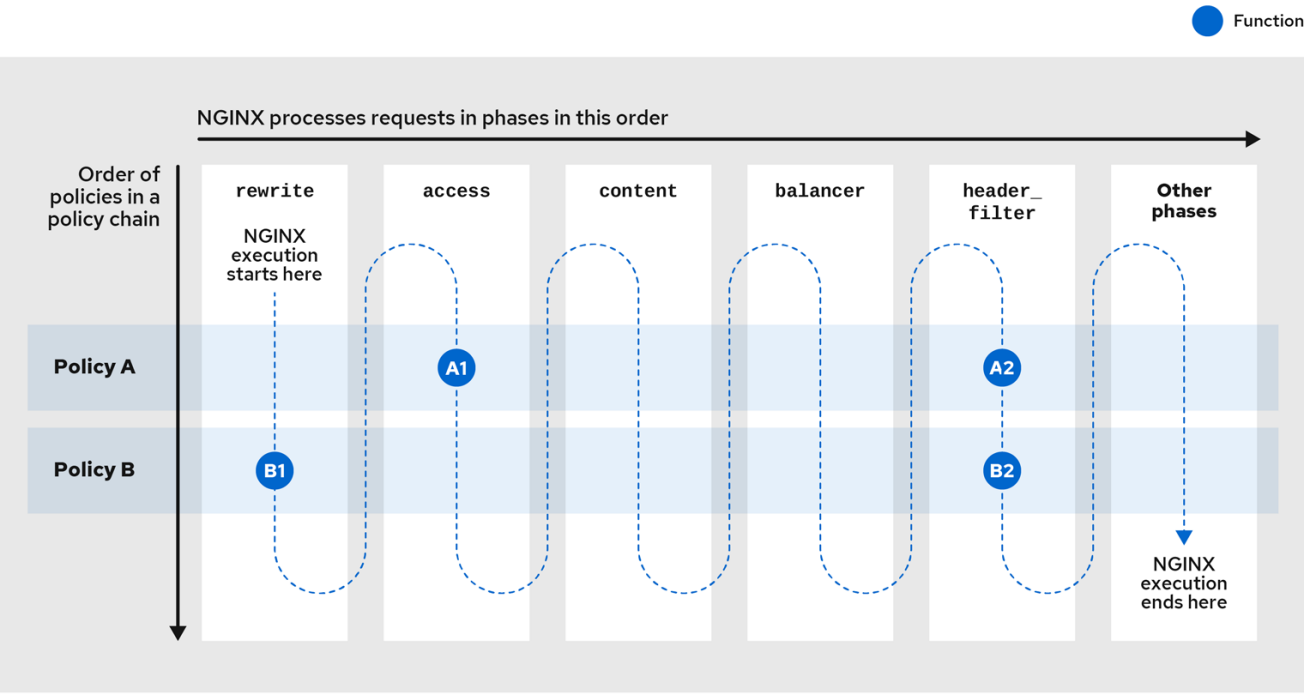


上の図では、ポリシー A がポリシーチェーンの最初のものです。しかし、NGINX は、まずポリシー B の関数を処理します。その関数が NGINX の最初のフェーズである **rewrite** フェーズに関連するためです。

ここで、以下の関数を持つポリシー A およびポリシー B が含まれるプロダクトのポリシーチェーンについて検討します。

- ポリシー A は以下を指定します。
  - **access** フェーズで実行する NGINX の関数 **A1**
  - **header_filter** フェーズで実行する NGINX の関数 **A2**
- ポリシー B は以下を指定します。
  - **rewrite** フェーズで実行する NGINX の関数 **B1**
  - **header_filter** フェーズで実行する NGINX の関数 **B2**

以下の図は、NGINX がプロダクトのポリシー関数を実行する順序を示しています。



APIcast がこのプロダクトにより公開されるアップストリーム API へのアクセスリクエストを受け取ると、APIcast はプロダクトのポリシーチェーンを確認し、以下の表で説明するように関数を実行します。

NGINX フェーズのリスト	NGINX が各フェーズで実行する機能
rewrite	ポリシー B が **rewrite** フェーズに指定する関数 **B1** を実行します。
access	ポリシー A が **access** フェーズに指定する関数 **A1** を実行します。
content	ポリシー A とポリシー B のいずれも、**content** フェーズで実行する関数を指定しません。
balancer	ポリシー A とポリシー B のいずれも、**balancer** フェーズで実行する関数を指定しません。
header_filter	ポリシーチェーンはポリシー A を指定し、次にポリシー B を指定します。したがって、このフェーズは、ポリシー A が **header_filter** フェーズに指定する関数 **A2** を実行し、続いてポリシー B が **header_filter** フェーズに指定する関数 **B2** を実行します。
body_filter	ポリシー A とポリシー B のいずれも、このフェーズで実行する関数を指定しません。
post_action	ポリシー A とポリシー B のいずれも、このフェーズで実行する関数を指定しません。
log	ポリシー A とポリシー B のいずれも、このフェーズで実行する関数を指定しません。

この例では、ポリシー **A** がポリシーチェーンの最初のものですが、ポリシー **B** の関数が NGINX が実行する最初の関数です。これは、ポリシー **B** が NGINX が (他のフェーズよりも前にある) **rewrite** フェーズで処理する関数 **B1** を指定するためです。

別の例として、以下のポリシーチェーンについて考えてみましょう。

1. **URL Rewriting**
2. **3scale APIcast** (すべてのプロダクトに割り当てられたデフォルトポリシー)

URL Rewriting ポリシーは、リクエストのターゲットパスを変更します。APIcast は **rewrite** フェーズで **URL Rewriting** 関数を実行します。**3scale APIcast** ポリシーは、APIcast が **rewrite** フェーズで実行する関数に加えて、APIcast が他の 3 つのフェーズで実行する関数を定義します。**URL Rewriting** ポリシーが最初になると、**3scale APIcast** ポリシーにより、書き換えられたパスにマッピングルールが適用されます。**3scale APIcast** ポリシーが最初にあり、**URL Rewriting** ポリシーが 2 番目の場合には、**3scale APIcast** ポリシーにより、元のパスにマッピングルールが適用されます。

関連情報

- 3scale 標準ポリシー関数を実行する NGINX フェーズ
- 3scale 標準ポリシーおよびそれらのポリシーを処理する NGINX フェーズ

4.2.2. 3scale 管理ポータルでのポリシーチェーンの変更

APIcast ゲートウェイ設定の一部として、3scale 管理ポータルでプロダクトのポリシーチェーンを変更します。

手順

1. 3scale にログインします。
2. ポリシーチェーンを設定する API プロダクトに移動します。
3. **[your_product_name] > Integration > Policies**の順に移動し、**Add policy** をクリックします。
4. **Policy Chain** セクションで、矢印アイコンを使用してポリシーチェーン内のポリシーの順番を変更します。
5. **Update Policy Chain** をクリックし、ポリシーチェーンを保存します。

次のステップ

管理ポータルの左側のナビゲーションパネルに、APIcast にプロモートしていない **設定** 変更があることを示す警告が表示されるようになりました。ポリシーチェーンの更新をステージング APIcast にプロモートし、必要に応じて更新をテストします。必要な動作を確認したら、更新を実稼働 APIcast にプロモートします。**APICAST_CONFIGURATION_CACHE** 環境変数をゼロより大きい数 (デフォルト) に設定すると、APIcast が更新された設定を使用するまでにその秒数かかります。

4.2.3. JSON 設定ファイルでの 3scale ポリシーチェーンの作成

APIcast のネイティブデプロイメントを使用している場合には、JSON 設定ファイルを作成して、外部でポリシーチェーンを制御することができます。

ポリシーチェーン JSON 設定ファイルには、以下の情報で設定される JSON 配列が含まれます。

- **services** オブジェクト (**id** の値 (ポリシーチェーンが適用されるサービスを指定する数字) が含まれる)
- **proxy** オブジェクト (**policy_chain** オブジェクトおよび下位オブジェクトが含まれる)
- **policy_chain** オブジェクト (ポリシーチェーンを定義する値が含まれる)
- 個々の **policy** オブジェクト (ポリシーを識別しポリシーの動作を設定するのに必要な **name** および **configuration** データの両方を指定する)

カスタムポリシー **sample_policy_1** および標準の API イントロスペクションポリシー **token_introspection** で設定されるポリシーチェーンの例を、以下に示します。

```
{
  "services":[
    {
      "id":1,
      "proxy":{
        "policy_chain":[
          {
            "name":"sample_policy_1", "version": "1.0",
            "configuration":{
              "sample_config_param_1":["value_1"],
              "sample_config_param_2":["value_2"]
            }
          },
          {
            "name": "token_introspection", "version": "builtin",
            "configuration": {
              introspection_url:["https://tokenauthorityexample.com"],
              client_id:["exampleName"],
              client_secret:["secretexamplekey123"]
          },
          {
             "name": "apicast", "version": "builtin",
          }
        ]
      }
    }
  ]
}
```

すべてのポリシーチェーンには、内蔵のポリシー **apicast** を含める必要があります。**apicast** ポリシーをポリシーチェーンのどこに設定したかによって、ポリシーの動作が変わります。

4.2.4. 3scale 標準ポリシー関数を実行する NGINX フェーズ

以下の表は、メインの NGINX フェーズおよび NGINX がそのフェーズで実行する関数を定義する標準ポリシーのリストを示しています。この表には、NGINX が処理する順序でフェーズが記載されています。

ポリシーチェーンには、NGINX が特定のフェーズで処理するポリシーを複数含めることができます。このような場合には、チェーン内のポリシーの順序が、API リクエストを処理して希望の結果を取得するための正しい順序になるようにしてください。この表では、ポリシーをアルファベット順で一覧表示します。

NGINX フェーズのリスト	このフェーズで処理される関数を定義する標準ポリシー
rewrite	3scale APIcast 3scale Referrer Anonymous Access Echo Header Modification NGINX Filter SOAP Upstream URL Rewriting URL Rewriting with Captures Websocket
access	3scale APIcast 3scale Batcher Camel Proxy Content Caching Edge Limiting IP Check JWT Claim Check RH-SSO/Keycloak Role Check Maintenance Mode OAuth 2.0 Mutual TLS Client Authentication OAuth 2.0 Token Introspection Rate Limit Headers Response/Request Content Limits Routing TLS Client Certificate Validation Upstream
content	3scale APIcast Liquid Context Debug Rate Limit Headers Routing Upstream
balancer	Upstream Mutual TLS
header_filter	CORS Request Handling Header Modification Response/Request Content Limits HTTP Response Code Overwrite
body_filter	Response/Request Content Limits
post_action	3scale APIcast Custom metrics
log	Edge Limiting Logging

関連情報

- APIcast NGINX フェーズが 3scale ポリシーを処理する仕組み
- 3scale 標準ポリシーおよびそれらのポリシーを処理する NGINX フェーズ

4.2.5. 3scale 標準ポリシーおよびそれらのポリシーを処理する NGINX フェーズ

以下の表は、標準ポリシーと、そのポリシーの関数を実行する NGINX フェーズまたはフェーズのリストを示しています。この表を使用してポリシーチェーンでポリシーを正しい順序で設定し、アップストリーム API の正しいリクエストを生成します。

標準ポリシー	ポリシー関数を実行する NGINX フェーズ
3scale APIcast	**init** **rewrite** **access** **content** **post_action** APIcast は、**3scale APIcast** ポリシーをすべてのリクエストに適用します。
Anonymous Access	**rewrite**
3scale Auth Caching	ポリシーチェーンでは、このポリシーの位置は問題ではありません。
3scale Batcher	**access**
3scale Referrer	**rewrite**
Camel Service	**access**
Conditional **ポリシー**	ポリシーチェーンでは、このポリシーの位置は問題ではありません。
Content Caching	**access**
CORS Request Handling	**header_filter**
Custom metrics	**post_action**
Echo	**rewrite**
Edge Limiting	**access** **log**
Header Modification	**rewrite** **header_filter**

標準ポリシー	ポリシー関数を実行する NGINX フェーズ
HTTP Response Code Overwrite	**header_filter**
IP Check	**access**
JWT Claim Check	**access**
Liquid Context Debug	**content**
Logging	**log**
Maintenance Mode	**access**
NGINX Filter	**rewrite**
OAuth 2.0 Mutual TLS Client Authentication	**access**
OAuth 2.0 Token Introspection	**access**
Proxy Service	ポリシーチェーンでは、このポリシーの位置は問題ではありません。
Rate Limit Headers	**access** +**content**
Response/Request Content Limits	**access** **header_filter** **body_filter**
Retry	ポリシーチェーンでは、このポリシーの位置は問題ではありません。
RH-SSO/Keycloak Role Check	**access**
Routing	**access** **content**
SOAP	**rewrite**
TLS Client Certificate Validation	**access**
TLS Termination	**ssl_certificate**
Upstream	**rewrite** **access** **content**

標準ポリシー	ポリシー関数を実行する NGINX フェーズ
Upstream Connection	ポリシーチェーンでは、このポリシーの位置は問題ではありません。
Upstream Mutual TLS	**balancer**
URL Rewriting	**rewrite**
URL Rewriting with Captures	**rewrite**
Websocket	**rewrite**

関連情報

- APIcast NGINX フェーズが 3scale ポリシーを処理する仕組み
- 3scale 標準ポリシー関数を実行する NGINX フェーズ

4.3. カスタム 3SCALE APICAST ポリシー

APIcast の動作を変更するためのカスタムポリシーを設定します。まず、カスタムポリシーを含む APIcast ポリシーを設定するポリシーチェーンを定義し、続いてポリシーチェーンを APIcast に追加します。

注記

Red Hat 3scale でカスタムポリシーを追加することは可能ですが、そのカスタムポリシーはサポートの対象ではありません。

APIcast のカスタムポリシーは、3scale デプロイメントの設定によって異なります。

- 次の Self-managed APIcast デプロイメントに、カスタムポリシーを追加します。OpenShift 上の APIcast、およびインストールしたコンテナー化された環境の APIcast。
- カスタムポリシーを Hosted APIcast に追加することはできません。

警告

決して、実稼働環境のゲートウェイで直接ポリシーを変更しないでください。変更を必ずテストしてください。

4.3.1. 3scale APIcast デプロイメントのカスタムポリシーについて

カスタム APIcast ポリシーを新規に作成することや、標準ポリシーを変更することができます。

カスタムポリシーを作成するには、以下の点を理解している必要があります。

- ポリシーは Lua で記述される。
- ポリシーは適切なファイルディレクトリーに保管しなければならない。
- ポリシーチェーン内での設定場所により、ポリシーの動作が異なる。
- カスタムポリシーを追加するインターフェイスは完全にサポートされているが、カスタムポリシー自体はサポートされていない。

4.3.2. 3scale Embedded APIcast へのカスタムポリシーの追加

オンプレミスデプロイメントにカスタム APIcast ポリシーを追加するには、カスタムポリシーが含まれる OpenShift イメージをビルドし、それをデプロイメントに追加する必要があります。3scale では、サンプルリポジトリーを提供しています。このリポジトリーを、カスタムポリシーを作成してオンプレミスデプロイメントに追加するためのフレームワークとして使用することができます。

このサンプルリポジトリーには、カスタムポリシー用の正しいディレクトリー構造に加えて、イメージストリームを作成するテンプレート、および作成するカスタムポリシーが含まれる新しい APIcast OpenShift イメージをビルドするための BuildConfigs が含まれています。

警告

apicast-custom-policies をビルドすると、ビルドプロセスは新しいイメージを **amp-apicast:latest** タグにプッシュします。このイメージストリームでイメージが変更されると、デフォルトでは **apicast-staging** および **apicast-production** タグの両方が自動的に新しいデプロイメントを開始するように設定されています。ステージング環境または実稼働環境のサービスが中断されるのを避けるためには、**Automatically start a new deployment when the image changes** チェックボックスの選択を解除して、自動デプロイメントを無効にします。あるいは、実稼働環境用に別のイメージストリームタグ (例: **amp-apicast:production**) を設定します。

手順

1. Creating a registry service account で作成したクレデンシャルを使用して、**docker-registry** シークレットを作成します。その際に、以下の点に留意してください。
   - **your-registry-service-account-username** を 12345678|username のフォーマットで作成したユーザー名に置き換えてください。
   - **your-registry-service-account-password** を Token Information タブでユーザー名の下に表示されるパスワードの文字列に置き換えてください。
   - イメージストリームが存在し **registry.redhat.io** を使用するすべての新規 **namespace** について、**docker-registry** シークレットを作成します。
     以下のコマンドを実行して **docker-registry** シークレットを作成します。

```
oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
```

```
--docker-username="your-registry-service-account-username" \
--docker-password="your-registry-service-account-password"
```

2. ポリシーの例が含まれる公開リポジトリー をフォークするか、そのコンテンツが含まれるプライベートリポジトリーを作成します。OpenShift がイメージをビルドするには、Git リポジトリーでカスタムポリシーのコードが必要です。プライベート Git リポジトリーを使用するには、OpenShift でシークレットを設定する必要がある点に注意してください。

3. リポジトリーをローカルにクローンし、ポリシーの実装を追加し、変更をご自分の Git リポジトリーにプッシュします。

4. **openshift.yml** テンプレートを更新します。具体的には、以下のパラメーターを変更します。

   a. **spec.source.git.uri: https://github.com/3scale/apicast-example-policy.git** (ポリシーの BuildConfig): ご自分の Git リポジトリーの場所に変更します。

   b. **spec.source.images[0].paths.sourcePath: /opt/app-root/policies/example** (カスタムポリシーの BuildConfig): **example** をリポジトリーの **policies** ディレクトリーに追加したカスタムポリシーの名前に変更します。

   c. オプションで、OpenShift オブジェクト名およびイメージタグを更新します。ただし、変更の一貫性が維持されるようにする必要があります。例: **apicast-example-policy** BuildConfig が **apicast-policy:example** イメージをビルドおよびプッシュし、それを **apicast-custom-policies** BuildConfig がソースとして使用する。これによりダグの一貫性が維持されます。

5. 以下のコマンドを実行して OpenShift オブジェクトを作成します。

```
oc new-app -f openshift.yml --param AMP_RELEASE=2.12
```

6. ビルドが自動的に開始されない場合には、以下の 2 つのコマンドを実行します。**apicast-example-policy** を変更している場合には、ご自分の BuildConfig 名に置き換えます (例: **apicast-<name>-policy**)。最初のコマンドが完了するのを待ってから、2 番目のコマンドを実行してください。

```
oc start-build apicast-example-policy
oc start-build apicast-custom-policies
```

   Embedded APIcast のイメージに **amp-apicast:latest** イメージストリームの変更を追跡するトリガーがある場合には、APIcast の新しいデプロイメントが開始されます。**apicast-staging** が再開されたら **Integration > Policies** の順に移動し、**Add Policy** ボタンをクリックしてご自分のカスタムポリシーがリストに表示されるのを確認します。カスタムポリシーを選択して設定したら、**Update Policy Chain** をクリックし、カスタムポリシーをステージング APIcast で動作状態にします。

4.3.3. 別の OpenShift Container Platform 上の 3scale へのカスタムポリシーの追加

カスタムポリシーを OpenShift Container Platform (OCP) 上の APIcast に追加することができます。そのためには、ご自分のカスタムポリシーが含まれる APIcast イメージを 統合 OpenShift Container Platform レジストリー から取得します。

手順

1. Embedded APIcast にポリシーを追加します。

2. APIcast ゲートウェイをメインの OpenShift クラスターにデプロイしていない場合には、メインの OpenShift クラスター上の内部レジストリーへの アクセスを確立します。

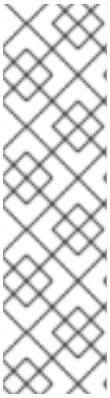
3. 3scale 2.12 APIcast OpenShift テンプレートを ダウンロードします。

4. テンプレートを変更するには、デフォルトの **image** ディレクトリーを内部レジストリーの完全なイメージ名に置き換えます。

```
image: <registry>/<project>/amp-apicast:latest
```

5. カスタマイズしたイメージを指定し、OpenShift テンプレートを使用して APIcast をデプロイします。

```
oc new-app -f customizedApicast.yml
```



注記

カスタムポリシーが APIcast に追加されて新しいイメージがビルドされ、そのイメージを使用して APIcast がデプロイされると、管理ポータルではそれらのポリシーが利用可能なポリシーとして自動的に表示されます。既存のサービスはこの新しいポリシーを利用可能なポリシーリストで認識できるので、任意のポリシーチェーンで使用することができます。

カスタムポリシーがイメージから削除され、APIcast が再起動されると、そのポリシーはリスト上で利用可能なポリシーとは表示されなくなるので、ポリシーチェーンに追加することができなくなります。

4.3.4. 3scale カスタムポリシーへの外部 Lua 依存関係の追加

外部の Lua 依存関係をカスタムポリシーに追加して、APIcast がまだ 3scale イメージにない Lua ライブラリーを使用できるようにすることができます。

以下の手順では、レスポンスボディーを JSON から XML に変換するカスタムポリシーの例 を使用して、この作業を行う方法を説明します。カスタムポリシーの例には、Lua で書かれた **xml2lua** XML パーサーが必要です。完全な例は、ビルドおよびテストを簡潔に示していますが、サンプル手順のみに従ってカスタムポリシーをデプロイすることはできません。外部の Lua 依存関係を持つカスタムポリシーをデプロイするには、この手順と共に 別の OpenShift Container Platform 上の 3scale へのカスタムポリシーの追加 の手順を実施する必要があります。

JSON to XML カスタムポリシーは、例としてのみ提示しています。実稼働環境で使用するためのものではありません。

前提条件

- 3scale カスタムポリシー
- 外部 Lua ライブラリーへのアクセス

手順

1. カスタムポリシーが含まれるディレクトリーに、外部 Lua ライブラリーを識別するファイルを追加します。
ファイルの名前は **Roverfile** である必要があります。**JSON to XML** カスタムポリシーの例では、**Roverfile** の内容は以下のようになります。

```
luarocks {
 group 'production' {
  module { 'xml2lua' },
 }
}
```

lua-rover は LuaRocks のラッパーです。**lua-rover** は依存関係の推移ロックを提供します。LuaRocks は、Lua モジュールのパッケージマネージャーです。

2. カスタムポリシーが含まれるディレクトリーに、**lua-rover** ロックファイルを追加します。ファイルの名前は **Roverfile.lock** である必要があります。**JSON to XML** カスタムポリシーの例では、**Roverfile.lock** の内容は以下のようになります。

```
xml2lua 1.5-2||productionbash-4.4
```

Roverfile および **Roverfile.lock** を併用することで、APIcast または 3scale operator が依存関係ライブラリーをフェッチできます。

3. カスタムポリシーを定義するファイルで、Lua 依存関係を指定する行を追加します。**JSON to XML** カスタムポリシーの例では、以下の行を指定します。

```
local xml2lua = require("xml2lua")
```

4. カスタムポリシーの構築に使用する Dockerfile で、**Roverfile** および **Roverfile.lock** をコピーし、**rover install** を実行します。**JSON to XML** カスタムポリシーの例では、以下の行を Dockerfile に追加します。

```
COPY Roverfile .
COPY Roverfile.lock .

RUN rover install --roverfile=/opt/app-root/src/Roverfile
```

Dockerfile は、APIcast または 3scale operator を使用してポリシーをビルドすることができます。

5. カスタムポリシーの **Makefile** で、他のカスタムポリシーと同様に **build** ターゲットを指定します。
たとえば、**build** ターゲットは以下のようになります。

```
TARGET_IMAGE="apicast/json_to_xml:latest"
# IP="http://localhost:8080"

build:
 docker build . --build-arg IMAGE=registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.12 -t $(TARGET_IMAGE)
```

次のステップ

外部 Lua 依存関係を持つカスタムポリシーをデプロイする残りの手順は、他のカスタムポリシーのデプロイと同じです。つまり、イメージをリポジトリーにプッシュし、APIcast イメージをビルドしたばかりのイメージに置き換える必要があります。

関連情報

- 別の OpenShift Container Platform 上の 3scale へのカスタムポリシーの追加
- APIManager CRD Reference, APIcastSpec **image** パラメーター
- APIcast Custom Resource reference **image** パラメーター

第5章 ポリシーチェーンと APICAST ネイティブデプロイメントのインテグレーション

ネイティブ APIcast デプロイメントの場合、**THREESCALE_CONFIG_FILE** 環境変数を使用して設定ファイルを指定することにより、カスタムポリシーチェーン を統合することができます。以下の例では、設定ファイル **example.json** を指定しています。

```
THREESCALE_CONFIG_FILE=example.json bin/apicast
```

5.1. ポリシーでの変数およびフィルターの使用

一部の標準ポリシーでは Liquid テンプレートがサポートされます。Liquid テンプレートにより、通常の文字列値だけでなくリクエストのコンテキストに存在する変数も使用することができます。

コンテキスト変数を使用するには、その名前を **{{** および **}}** で囲みます (例: **{{ uri }}**)。変数がオブジェクトの場合には、その属性にもアクセスすることができます (例: **{{ somevar.attr }}**)。

すべてのポリシーで使用することのできる標準の変数を以下に示します。

- **uri**:クエリーパラメーターを除外したリクエストのパス。組み込まれた NGINX 変数 **$uri** の値
- **host**:リクエストのホスト (組み込まれた NGINX 変数 **$host** の値)
- **remote_addr**:クライアントの IP アドレスト (組み込まれた NGINX 変数 **$remote_addr** の値)
- **headers**:リクエストヘッダーが含まれるオブジェクト。特定のヘッダーの値を取得するには、**{{headers['Some-Header']}}** を使用します。
- **http_method**:リクエストのメソッド (GET、POST 等)

これらの標準変数はリクエストのコンテキストで使用されますが、ポリシーではコンテキストにさらに変数を追加することができます。なお、ここで使われるフェーズとは、APIcast のすべての実行ステップを指します。以下に示す状況では、ポリシーチェーンのすべてのポリシーで変数を使用することができます。

- 同一フェーズ内 (ポリシーで変数が追加され、追加後に次のポリシーで使用される)。
- 次フェースで (あるフェーズで変数が追加され、その変数が次のフェーズで使用される)。

標準の 3scale APIcast ポリシーでコンテキストに追加される変数の例を以下に示します。

- **jwt**:解析された JWT トークンの JSON ペイロード (OpenID Connect 認証用)
- **credentials**:アプリケーションのクレデンシャルを保持するオブジェクト。例: **"app_id":"972f7b4f"**、**"user_key":"13b668c4d1e10eaebaa5144b4749713f"**
- **service**:現在のリクエストが処理されるサービスの設定を保持するオブジェクト。たとえば、サービス ID は {{ service.id }} として利用可能です。

コンテキストで使用することのできるオブジェクトおよび値の完全なリストは、Liquid コンテキストのデバッグ を参照してください。

変数は Liquid テンプレートの機能を活用して使用されます。たとえば **{{ remote_addr }}**、**{{ headers['Some-Header'] }}**、**{{ jwt.aud }}** 等。値に変数をサポートするポリシーは、**_type** 接尾辞が付く特殊なパラメーターを持ちます (例: **value_type**、**name_type** 等)。このパラメーターには、2 つの

値を設定することができます (プレーンテキストの場合の plain および liquid テンプレートの場合の liquid)。

APIcast では、変数の値に適用することのできる Liquid フィルターもサポートされます。フィルターは Liquid 変数の値に NGINX 関数を適用します。

フィルターは変数出力タグ **{{ }}** で囲み、変数の名前または実際の値、パイプ記号 **|**、およびフィルター名の順に定義します。以下に例を示します。

- **{{ 'username:password' | encode_base64 }}** (ここで **username:password** が変数)
- **{{ uri | escape_uri }}**

パラメーターを必要としないフィルターもあります。この場合には、変数の代わりに空の文字列を使用することができます。たとえば、**{{ '' | utctime }}** は現在の時刻を TUC タイムゾーンで返します。

フィルターは、**{{ variable | function1 | function2 }}** のようにつなげることができます。たとえば **{{ '' | utctime | escape_uri }}**。

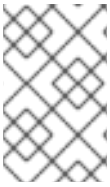
利用可能な関数のリストを以下に示します。

- escape_uri
- unescape_uri
- encode_base64
- decode_base64
- crc32_short
- crc32_long
- hmac_sha1
- md5
- md5_bin
- sha1_bin
- quote_sql_str
- today
- time
- now
- localtime
- utctime
- cookie_time
- http_time
- parse_http_time

第6章 FUSE のポリシーエクステンションを使用した 3SCALE メッセージコンテンツの変換

Red Hat Fuse を使用して、Red Hat 3scale API Management 用の非常に柔軟なポリシーエクステンションを作成することができます。そのためには、Fuse on OpenShift でポリシーエクステンションを作成し、それらを 3scale 管理ポータルでポリシーとして設定します。APIcast Camel プロキシーポリシーを使用して、リクエストおよびレスポンスのメッセージコンテンツの複雑な変換を実施することができます (例: XML から JSON への変換)。この変換は、Apache Camel インテグレーションのフレームワークに実装されます。

さらに、静的な APIcast コンテナーイメージを再ビルドして再デプロイする代わりに、Camel でカスタムポリシーエクステンションを動的に追加または変更することができます。Camel Domain Specific Language (DSL) で書かれた任意の Camel Enterprise Integration Pattern (EIP) を使用して、APIcast ポリシーエクステンションを実装することができます。これにより、Java や XML 等の使い慣れたプログラミング言語を使用して、ポリシーエクステンションを作成することができます。本トピックの例では、Camel Netty4 HTTP コンポーネントを使用して、HTTP プロキシーを Java で実装しています。



注記

3scale API バックエンドですでに Fuse Camel アプリケーションを使用している場合は、この機能は必要ありません。この場合は、既存の Fuse Camel アプリケーションを使用して変換を実施することができます。

必要なソフトウェアコンポーネント

以下の Red Hat Integration コンポーネントを同じ OpenShift クラスターにデプロイしている必要があります。

- Fuse on OpenShift 7.10
- オンプレミス型 3scale 2.12
- Embedded APIcast (ステージングおよび実稼働環境用のデフォルトゲートウェイ)、または Self-managed APIcast

カスタム Fuse ポリシーを 3scale とは別の OpenShift プロジェクトにデプロイすることができますが、これは必須の要件ではありません。ただし、両プロジェクト間で通信ができるようにしてください。詳細については、Networking の Configuring network policy with OpenShift SDN を参照してください。

関連情報

- Fuse on OpenShift ガイド

6.1. APICAST と FUSE の APACHE CAMEL による変換のインテグレーション

APIcast と Fuse on OpenShift で Apache Camel アプリケーションとして記述された変換を統合することができます。ポリシーエクステンションによる変換が 3scale に設定およびデプロイされると、3scale トラフィックは Camel ポリシーエクステンションを通過し、メッセージのコンテンツが変換されます。この場合、Camel はリバース HTTP プロキシーとして機能し、APIcast は Camel に 3scale トラフィックを送信し、Camel がそのトラフィックを API バックエンドに送信します。

本トピックの例では、Camel Netty4 HTTP コンポーネントを使用して HTTP プロキシーを作成しています。

- HTTP プロキシープロトコルを通じて受け取ったリクエストは、HTTP ボディーが大文字に変換されて目的のサービスに転送されます。
- 目的のサービスからのレスポンスは大文字への変換処理を受け、続いてクライアントに戻されます。
- 以下の例で、HTTP および HTTPS のユースケースに必要な設定を説明します。

前提条件

- Fuse on OpenShift 7.10 および 3scale 2.12 を、同じ OpenShift クラスターにデプロイしている必要があります。インストールの詳細については、以下のドキュメントを参照してください。
  - Fuse on OpenShift ガイド
  - 3scale のインストール
- Fuse on OpenShift および 3scale をインストールしてプロジェクトを作成するには、クラスターの管理者権限が必要です。ただし、プロジェクトごとの編集アクセス権限で、デプロイ設定の作成、Pod のデプロイ、およびサービスの作成が可能です。

手順

1. Camel **netty4-http** コンポーネントを使用して Java で Apache Camel アプリケーションを作成し、HTTP プロキシーを実装します。これにより、任意の Camel コンポーネントを使用して、メッセージを変換することができます。
   以下に示す簡単な例では、リクエストおよびサービスからのレスポンスを大文字に変換します。

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Locale;

import org.apache.camel.Exchange;
import org.apache.camel.Message;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.RouteDefinition;

public class ProxyRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {
        final RouteDefinition from;
        if (Files.exists(keystorePath())) {
            from = from("netty4-http:proxy://0.0.0.0:8443?
ssl=true&keyStoreFile=/tls/keystore.jks&passphrase=changeit&trustStoreFile=/tls/keystore.jks"
); (1)
        } else {
            from = from("netty4-http:proxy://0.0.0.0:8080");
        }

        from
            .process(ProxyRoute::uppercase)
            .toD("netty4-http:"
                + "${headers." + Exchange.HTTP_SCHEME + "}://" (2)
                + "${headers." + Exchange.HTTP_HOST + "}:"
```

```
                + "${headers." + Exchange.HTTP_PORT + "}"
                + "${headers." + Exchange.HTTP_PATH + "}")
            .process(ProxyRoute::uppercase);
    }

    Path keystorePath() {
        return Path.of("/tls", "keystore.jks");
    }

    public static void uppercase(final Exchange exchange) { 3
        final Message message = exchange.getIn();
        final String body = message.getBody(String.class);
        message.setBody(body.toUpperCase(Locale.US));
    }

}
```

1. この簡単な例では、Java キーストアファイルが **/tls/keystore.jks** にマウントされている場合、リスニングポートは **8443** に設定されます。
2. 3scale により Camel プロキシーポリシーが呼び出されると、**HTTP_SCHEME**、**HTTP_HOST**、**HTTP_PORT**、および **HTTP_PATH** ヘッダーの値は、3scale のバックエンド API に設定された値を元に自動的に設定されます。
3. この簡単な例では、メッセージの内容が大文字に変換されます。Camel エンタープライズ統合パターンを使用して、リクエストおよびレスポンスメッセージの内容のより複雑な変換を実施することができます (例: XML から JSON への変換)。

2. Camel アプリケーションを OpenShift にデプロイし、それをサービスとして公開します。詳細は、Fuse on OpenShift でのアプリケーションの作成およびデプロイ を参照してください。

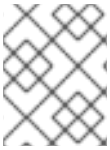
関連情報

- Apache Camel Component Reference の Netty4 HTTP Component

6.2. FUSE ON OPENSHIFT の APACHE CAMEL を使用して作成された APICAST ポリシーエクステンションの設定

Fuse on OpenShift を使用した Apache Camel による変換を実装したら、3scale 管理ポータルを使用して APIcast ポリシーチェーンのポリシーエクステンションとして設定することができます。

ポリシーエクステンションにより、Camel HTTP プロキシーを使用するように 3scale プロダクトを設定することができます。このサービスを使用して HTTP プロキシー経由で 3scale トラフィックを送信し、サードパーティープロキシーでリクエスト/レスポンスの変更を実施します。ここでは、サードパーティープロキシーは Fuse on OpenShift を使用して実装された Apache Camel です。TLS を使用して Camel HTTP プロキシーサービスにセキュアに接続するように、APIcast を設定することもできます。



注記

ポリシーエクステンションコードは Fuse on OpenShift の Apache Camel アプリケーションで実装されており、3scale から変更したり削除したりすることはできません。

前提条件

- Fuse on OpenShift 7.10 および 3scale 2.12 を、同じ OpenShift クラスターにデプロイしている必要があります。インストールの詳細については、以下のドキュメントを参照してください。
  - Fuse on OpenShift ガイド
  - 3scale のインストール
- Fuse on OpenShift の Apache Camel アプリケーションを使用して、APIcast ポリシーエクステンションを実装している必要があります。「APIcast と Fuse の Apache Camel による変換のインテグレーション」を参照してください。
- OpenShift Pod に Apache Camel アプリケーションをデプロイし、それをサービスとして公開している必要があります。詳細は、Creating and Deploying Applications on Fuse on OpenShift を参照してください。

手順

1. 3scale 管理ポータルで **Integration** > **Policies** の順に選択します。
2. **POLICIES** > **Add policy** > **Camel Service** の順に選択します。
3. 適切なフィールドに、Camel HTTP プロキシーサービスに接続するための OpenShift ルートを入力します。
   - **https_proxy**: **http** プロトコルおよび TLS ポートを使用して、Camel HTTPS プロキシーに接続します。以下に例を示します。

     ```
     http://camel-proxy.my-3scale-management-project.svc:8443
     ```

   - **http_proxy**: **http** プロトコルおよびポートを使用して、Camel HTTP プロキシーに接続します。以下に例を示します。

     ```
     http://camel-proxy.my-3scale-management-project.svc:8080
     ```

   - **all_proxy**:プロトコルが指定されない場合に、**http** プロトコルおよびポートを使用して、Camel HTTP プロキシーに接続します。以下に例を示します。

     ```
     http://camel-proxy.my-3scale-management-project.svc:8080
     ```

4. 更新されたポリシー設定をステージング環境または実稼働環境にプロモートします。たとえば、**Promote v.3 to Staging APIcast**をクリックします。
5. 3scale **curl** コマンドを使用して、APIcast ポリシーの設定をテストします。以下に例を示します。

   ```
   curl "https://testapi-3scale-apicast-staging.myuser.app.dev.3sca.net:443/?
   user_key=MY_USER_KEY" -k
   ```

   APIcast は、Camel HTTP プロキシーへの接続に新しい TLS セッションを確立します。
6. メッセージのコンテンツが変換されていることを確認します。この例では大文字に変換されます。

7. APIcast をバイパスし、TLS を使用して直接 Camel HTTP プロキシーをテストする場合は、カスタム HTTP クライアントを使用する必要があります。たとえば、**netcat** コマンドを使用することができます。

```
$ print "GET https://mybackend.example.com HTTP/1.1\nHost:
mybackend.example.com\nAccept: */*\n\n" | ncat --no-shutdown --ssl my-camel-proxy 8443
```

この例では、**GET** の後に完全な URL を使用して HTTP プロキシーリクエストを作成し、 **ncat --ssl** パラメーターを使用してポート **8443** の **my-camel-proxy** ホストへの TLS 接続を指定しています。

注記

プロキシーは **CONNECT** メソッドを使用した HTTP トンネリングをサポートしないため、**curl** またはその他の一般的な HTTP クライアントを使用して Camel HTTP プロキシーを直接テストすることはできません。**CONNECT** で HTTP トンネリングを使用する場合、トランスポートはエンドツーエンドで暗号化され、Camel HTTP プロキシーはペイロードを仲介することはできません。

関連情報

- 「Camel Service」

第7章 APICAST の環境変数

APIcast の環境変数を使用すると、APIcast の動作を変更することができます。サポートされている環境変数の値を以下に示します。

注記

- サポートされていない環境変数および非推奨の環境変数は記載されていません
- 一部の環境変数の機能は、APIcast ポリシーに移されています

- all_proxy、ALL_PROXY
- APICAST_ACCESS_LOG_BUFFER
- APICAST_ACCESS_LOG_FILE
- APICAST_BACKEND_CACHE_HANDLER
- APICAST_CACHE_MAX_TIME
- APICAST_CACHE_STATUS_CODES
- APICAST_CONFIGURATION_CACHE
- APICAST_CONFIGURATION_LOADER
- APICAST_CUSTOM_CONFIG
- APICAST_ENVIRONMENT
- APICAST_EXTENDED_METRICS
- APICAST_HTTPS_CERTIFICATE
- APICAST_HTTPS_CERTIFICATE_KEY
- APICAST_HTTPS_PORT
- APICAST_HTTPS_VERIFY_DEPTH
- APICAST_LOAD_SERVICES_WHEN_NEEDED
- APICAST_LOG_FILE
- APICAST_LOG_LEVEL
- APICAST_MANAGEMENT_API
- APICAST_MODULE
- APICAST_OIDC_LOG_LEVEL
- APICAST_PATH_ROUTING
- APICAST_PATH_ROUTING_ONLY

- APICAST_POLICY_LOAD_PATH
- APICAST_PROXY_HTTPS_CERTIFICATE
- APICAST_PROXY_HTTPS_CERTIFICATE_KEY
- APICAST_PROXY_HTTPS_PASSWORD_FILE
- APICAST_PROXY_HTTPS_SESSION_REUSE
- APICAST_REPORTING_THREADS
- APICAST_RESPONSE_CODES
- APICAST_SERVICE_CACHE_SIZE
- APICAST_SERVICE_${ID}_CONFIGURATION_VERSION
- APICAST_SERVICES_LIST
- APICAST_SERVICES_FILTER_BY_URL
- APICAST_UPSTREAM_RETRY_CASES
- APICAST_WORKERS
- BACKEND_ENDPOINT_OVERRIDE
- HTTP_KEEPALIVE_TIMEOUT
- http_proxy HTTP_PROXY
- https_proxy HTTPS_PROXY
- no_proxy NO_PROXY
- OPENSSL_VERIFY
- OPENTRACING_CONFIG
- OPENTRACING_HEADER_FORWARD
- OPENTRACING_TRACER
- RESOLVER
- THREESCALE_CONFIG_FILE
- THREESCALE_DEPLOYMENT_ENV
- THREESCALE_PORTAL_ENDPOINT

all_proxy、ALL_PROXY

デフォルト: 値なし **値**: 文字列 **例**: **http://forward-proxy:80**

プロトコル固有のプロキシーが指定されていない場合に、サービスへの接続に使用される HTTP プロキシーを定義します。認証機能はサポートされていません。

APICAST_ACCESS_LOG_BUFFER
デフォルト: 値なし

値: 正の整数

アクセスログ書き込みをバイトのチャンクに含めることを許可します。その結果、システムコールが少なくなり、ゲートウェイのパフォーマンスが向上します。

APICAST_ACCESS_LOG_FILE
デフォルト: stdout

アクセスログを保存するファイルを定義します。

APICAST_BACKEND_CACHE_HANDLER
値: strict | resilient

デフォルト: strict

非推奨:この環境変数の代わりに、Caching ポリシーを使用してください。

バックエンドにアクセスすることができない場合に、承認キャッシュがどのように動作するかを定義します。strict に設定すると、バックエンドにアクセスすることができない場合にキャッシュされたアプリケーションを削除します。resilient に設定すると、バックエンドから承認が拒否された場合にのみキャッシュされたアプリケーションを削除します。

APICAST_CACHE_MAX_TIME
デフォルト: 1m

値: 文字列

レスポンスがシステムにキャッシュされる設定の場合、この変数の値はキャッシュされる最大の時間を示します。cache-control ヘッダーが設定されていない場合、キャッシュされる時間はここで定義される時間になります。

この値のフォーマットは、**proxy_cache_valid** NGINX ディレクティブ で定義されます。

このパラメーターは、コンテンツキャッシュポリシーを使用している API によってのみ使用され、リクエストのキャッシュが可能です。

APICAST_CACHE_STATUS_CODES
デフォルト: 200、302

値: 文字列

アップストリームからのレスポンスコードがこの環境変数で定義されるステータスコードのいずれかと一致する場合、レスポンスの内容が NGINX にキャッシュされます。キャッシュされる時間は、ヘッダーキャッシュ時間の値または **APICAST_CACHE_MAX_TIME** 環境変数で定義される最大時間のいずれかに依存します。

このパラメーターは、コンテンツキャッシュポリシーを使用している API によってのみ使用され、リクエストのキャッシュが可能です。

APICAST_CONFIGURATION_CACHE
値: 数字

デフォルト: 0

設定を保存する間隔 (秒単位) を指定します。0 (ブート値の **APICAST_CONFIGURATION_LOADER** との互換性はない) または 60 より大きい値を設定する必要があります。たとえば、**APICAST_CONFIGURATION_CACHE** を 120 に設定すると、ゲートウェイは 2 分 (120 秒) ごとに設定を API Manager から読み込み直します。負の値を設定すると、再読み込みは無効になります。

APICAST_CONFIGURATION_LOADER
値: boot | lazy

デフォルト: lazy

設定の読み込み方法を定義します。boot に設定すると、ゲートウェイの起動時に API Manager に設定を要求します。lazy に設定すると、リクエストを受信するたびに設定を読み込みます (リクエストのたびに完全にリフレッシュするには、**APICAST_CONFIGURATION_CACHE** を 0 に設定する必要があります)。

APICAST_CUSTOM_CONFIG
非推奨:この環境変数の代わりに、policies を使用してください。

既存の APIcast ロジックをオーバーライドするカスタムロジックを実装する Lua モジュールの名前を定義します。

APICAST_ENVIRONMENT
デフォルト:

値: 文字列[:]

例: production:cloud-hosted

APIcast は、環境 (またはパス) のリストをコンマ (**:**) 区切りで読み込む必要があります。このリストは、CLI の **-e** または **--environment** パラメーターの代わりに使用することができ、たとえばデフォルト環境としてコンテナーイメージに保存されます。CLI で渡される値は、常にこの変数に優先します。

APICAST_EXTENDED_METRICS
デフォルト: false

値: ブール値

例:true

Prometheus メトリクスに関する追加情報を有効にします。以下のメトリクスでは **service_id** および **service_system_name** ラベルを使用することができ、APIcast をより詳細に設定することができます。

- **total_response_time_seconds**
- **upstream_response_time_seconds**
- **upstream_status**

APICAST_HTTPS_CERTIFICATE
デフォルト: 値なし

HTTPS 接続用 X.509 証明書が含まれる PEM 形式ファイルへのパス

APICAST_HTTPS_CERTIFICATE_KEY
デフォルト: 値なし

X.509 証明書の秘密鍵が含まれる PEM 形式ファイルへのパス

APICAST_HTTPS_PORT
デフォルト: 値なし

HTTPS 接続用に APIcast がリッスンを開始するポートを制御します。この設定が HTTP 用ポートと競合する場合には、HTTPS 用にだけ使用されます。

APICAST_HTTPS_VERIFY_DEPTH
デフォルト: 1

値: 正の整数

クライアント証明書チェーンの最大長を定義します。このパラメーターの値が **1** の場合は、クライアント証明書チェーンに追加の証明書を含めることができます。たとえば、ルート認証局などです。

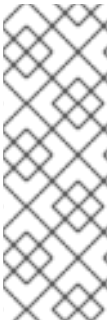
APICAST_LOAD_SERVICES_WHEN_NEEDED
値:

- **真** の場合には **true** または **1**
- **偽** の場合には **false**、**0**、または空欄

デフォルト: **false**

多くのサービスが設定されている場合に、このオプションを使用することができます。ただし、そのパフォーマンスは、サービスの数、APIcast と 3scale 管理ポータル間のレイテンシー、設定の残存期間 (TTL) など、他の要素に依存します。

デフォルトでは、APIcast が管理ポータルから設定をダウンロードするたびに、すべてのサービスが読み込みます。このオプションを有効にすると、設定にレイジーローディングが使用されます。APIcast は、リクエストの Host ヘッダーで指定したホストに設定されたサービスだけを読み込みます。



注記

- **APICAST_CONFIGURATION_CACHE** で定義されたキャッシュ動作が適用されます。
- **APICAST_CONFIGURATION_LOADER** が **boot** に設定されている場合には、このオプションは無効になります。
- **APICAST_PATH_ROUTING** との互換性はありません。

APICAST_LOG_FILE
デフォルト: stderr

OpenResty エラーログが含まれるファイルを定義します。このファイルは、**bin/apicast** の **error_log** ディレクティブで使用されます。ファイルパスは、絶対パスまたは APIcast プリフィックスディレクトリーへの相対パスのいずれかで指定します。デフォルトの接頭辞ディレクトリーは APIcast である点に注意してください。詳細は、NGINX のドキュメント を参照してください。

APICAST_LOG_LEVEL
値: debug | info | notice | warn | error | crit | alert | emerg

デフォルト: warn

OpenResty ログのログレベルを指定します。

APICAST_MANAGEMENT_API
値:

- **disabled**: 完全に無効で、ポートをリッスンするだけの状態
- **status**: ヘルスチェック用に、/status/ エンドポイントだけが有効な状態
- **debug**: API 全体がオープンな状態

Management API の機能は強力で、APIcast の設定を制御することができます。debug レベルは、デバッグ用途にのみ有効にしてください。

APICAST_MODULE
デフォルト: apicast

非推奨:この環境変数の代わりに、policies を使用してください。

API ゲートウェイロジックを実装するメインの Lua モジュール名を指定します。カスタムモジュールは、デフォルトの **apicast.lua** モジュールの機能に優先します。モジュールの使用方法の 例 を参照してください。

APICAST_OIDC_LOG_LEVEL
値: debug | info | notice | warn | error | crit | alert | emerg

デフォルト: err

OpenID Connect インテグレーションに関するログのログレベルを設定することができます。

APICAST_PATH_ROUTING
値:

- 真の場合には **true** または **1**
- 偽の場合には **false**、**0**、または空欄

このパラメーターを **true** に設定すると、ゲートウェイはデフォルトのホストベースのルーティングに加えて、パスベースのルーティングを使用します。API リクエストは、リクエストの **Host** ヘッダーの値が **Public Base URL** にマッチするサービスの中で、マッピングルールが最初にマッチするサービスにルーティングされます。

APICAST_PATH_ROUTING_ONLY
値:

- 真の場合には **true** または **1**
- 偽の場合には **false**、**0**、または空欄

このパラメーターを **true** に設定すると、ゲートウェイはパスベースのルーティングを使用し、デフォルトのホストベースのルーティングにはフォールバックしません。API リクエストは、リクエストの **Host** ヘッダーの値が **公開ベース URL** にマッチするサービスの中で、マッピングルールが最初にマッチするサービスにルーティングされます。

このパラメーターは、APICAST_PATH_ROUTING に優先します。**APICAST_PATH_ROUTING_ONLY** が有効な場合は、APIcast は **APICAST_PATH_ROUTING** の値に関わらずパスベースのルーティングだけを実施します。

APICAST_POLICY_LOAD_PATH
デフォルト: **APICAST_DIR/policies**

値: 文字列[:]

例: **~/apicast/policies:$PWD/policies**

APIcast がポリシーを探すパスのコロン (**:**) 区切りリスト。開発用ディレクトリーから最初にポリシーを読み込む場合や、例を読み込む場合に使用することができます。

APICAST_PROXY_HTTPS_CERTIFICATE
デフォルト:

値: 文字列

例: /home/apicast/my_certificate.crt

APIcast がアップストリームと接続する際に使用するクライアント SSL 証明書へのパス。この証明書が設定内のすべてのサービスに使用される点に注意してください。

APICAST_PROXY_HTTPS_CERTIFICATE_KEY
デフォルト:

値: 文字列

例: /home/apicast/my_certificate.key

クライアント SSL 証明書の鍵へのパス

APICAST_PROXY_HTTPS_PASSWORD_FILE
デフォルト:

値: 文字列

例: /home/apicast/passwords.txt

APICAST_PROXY_HTTPS_CERTIFICATE_KEY で指定する SSL 証明書の鍵のパスフレーズが含まれるファイルへのパス

APICAST_PROXY_HTTPS_SESSION_REUSE
デフォルト: on

値:

- **on**: SSL セッションを再利用します。
- **off**: SSL セッションを再利用しません。

APICAST_REPORTING_THREADS
デフォルト: 0

値: 0 または正の整数

実験的機能:負荷が極端に大きい場合のパフォーマンスは予測が不可能で、レポートが失われる可能性があります。

0 より大きい値を設定すると、バックエンドへの非同期のレポートが有効になります。これは、パ

フォーマンスを向上させるための新たな **実験的** 機能です。クライアントにはバックエンドのレイテンシーは適用されず、すべてが非同期状態で処理されます。この値により、同時に実行することのできる非同期レポートの数を決定します。レポート数がこの値を超えると、クライアントにスロットリングが適用され、レイテンシーが追加されます。

APICAST_RESPONSE_CODES
値:

- 真の場合には **true** または **1**
- 偽の場合には **false**、**0**、または空欄

デフォルト: 空欄 (**false**)

true に設定すると、APIcast は API バックエンドから返されたレスポンスのレスポンスコードを 3scale に記録します。詳細は、Setting up and evaluating the 3scale response codes log for your API を参照してください。

APICAST_SERVICE_CACHE_SIZE
値: 0 または正の整数

デフォルト: 1000

APIcast が内部キャッシュに保存することのできるサービスの数を指定します。Lua の **lru** キャッシュによりすべてのエントリーが初期化されるので、大きな値を設定するとパフォーマンスに影響が出ます。

APICAST_SERVICE_${ID}_CONFIGURATION_VERSION
${ID} を実際のサービス ID に置き換えてください。値は、管理ポータルの設定履歴に表示される設定バージョンにする必要があります。特定のバージョンに設定すると自動更新されなくなり、常にそのバージョンが使用されます。

APICAST_SERVICES_LIST
値: サービス ID のコンマ区切りリスト

APICAST_SERVICES_LIST 環境変数を使用して、3scale API Manager で設定されているサービスにフィルターを適用します。この環境変数は、ゲートウェイの特定サービスの設定にだけ適用され、リストで指定されていないサービス ID は無視されます。プロダクトのサービス識別子は、管理ポータルの **Products > [Your_product_name] > Overview**に移動し、続いて、**Configuration, Methods and Settings** および **ID for API calls**を参照して確認することができます。

APICAST_SERVICES_FILTER_BY_URL
値: **.*.example.com** 等の PCRE (Perl 互換正規表現)

3scale API Manager で設定されているサービスにフィルターを適用します。

このフィルターは、ステージングまたは実稼働環境いずれかの **公開ベース URL** を照合します。フィルターにマッチしないサービスは、無視されます。正規表現をコンパイルすることができない場合には、サービスは読み込まれません。

注記

フィルターにはマッチしないが「APICAST_SERVICES_LIST」に **含まれる** サービスは、無視されません。

例7.1 バックエンドのエンドポイントに適用される正規表現フィルター

正規表現フィルター **http://.*.foo.dev** が以下のバックエンドのエンドポイントに適用される。

1. http://staging.foo.dev
2. http://staging.bar.dev
3. http://prod.foo.dev
4. http://prod.bar.dev

この場合、**1** および **3** は APIcast に設定され、 **2** および **4** は無視されます。

APICAST_UPSTREAM_RETRY_CASES

値: error | timeout | invalid_header | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | non_idempotent | off

注記

この環境変数は Retry ポリシーが設定されている場合にのみ使用され、いつアップストリーム API へのリクエストを再試行するかを指定します。Nginx の PROXY_NEXT_UPSTREAM モジュール と同じ値を設定することができます。

APICAST_WORKERS

デフォルト: auto

値: **数字** | auto

この値は、nginx の **worker_processes** ディレクティブ で使用されます。**1** が使用される開発環境を除き、デフォルトの APIcast では **auto** が使用されます。

BACKEND_ENDPOINT_OVERRIDE

設定からのバックエンドエンドポイントをオーバーライドする URI。OpenShift がデプロイした AMP の外部にデプロイする際に役立ちます。**例**: **https://backend.example.com**。

HTTP_KEEPALIVE_TIMEOUT

デフォルト: 75 **値**: 正の整数 **例:1**

このパラメーターにより、キープアライブ用のクライアント接続がタイムアウトする期間を設定します。この間、サーバー側では接続が開き続けます。値にゼロを設定すると、キープアライブ用のクライアント接続は無効になります。

デフォルトでは、ゲートウェイは HTTP_KEEPALIVE_TIMEOUT を無効にした状態に維持します。この設定により、NGINX からのキープアライブのタイムアウト (デフォルト値は 75 秒) を使用することができます。

http_proxy、HTTP_PROXY

デフォルト: 値なし **値**: 文字列 **例**: **http://forward-proxy:80**

HTTP サービスへの接続に使用される HTTP プロキシーを定義します。認証機能はサポートされていません。

https_proxy、HTTPS_PROXY

デフォルト: 値なし **値**: 文字列 **例**: **https://forward-proxy:443**

HTTPS サービスへの接続に使用される HTTP プロキシーを定義します。認証機能はサポートされていません。

no_proxy、NO_PROXY
デフォルト: 値なし **値**: string\[,<string>\]; * **例**: **foo,bar.com,.extra.dot.com**

リクエストをプロキシーすべきではないホスト名およびドメイン名のコンマ区切りリストを定義します。*1 文字 (すべてのホストとマッチする) を設定すると、実質的にプロキシーは無効になります。

OPENSSL_VERIFY
値:

- **0、false**: ピア検証を無効にします
- **1、true**: ピア検証を有効にします

OpenSSL ピア検証を制御します。OpenSSL はシステム証明書ストアを使用することができないため、デフォルトではオフになっています。カスタム証明書バンドルを信頼済み証明書に追加する必要があります。

lua_ssl_trusted_certificate を使用して、 export-builtin-trusted-certs により生成される証明書バンドルをポイントすることを推奨します。

OPENTRACING_CONFIG
この環境変数は、OpenTracing トレーサーの設定ファイルを定義するのに使用されます。**OPENTRACING_TRACER** が設定されていない場合には、この変数は無視されます。

それぞれのトレーサーには、デフォルトの設定ファイル * **jaeger: conf.d/opentracing/jaeger.example.json** があります。

この変数を使用してファイルパスを設定することにより、デフォルトで提供されるものとは異なる設定をマウントすることができます。

例: **/tmp/jaeger/jaeger.json**

OPENTRACING_HEADER_FORWARD
デフォルト: **uber-trace-id**

この環境変数は、OpenTracing の情報を転送するのに使用される HTTP ヘッダーを制御します。この HTTP ヘッダーは、アップストリームサーバーに転送されます。

OPENTRACING_TRACER
例: **jaeger**

この環境変数は、読み込むトレースライブラリーを制御します。現時点では、OpenTracing のトレーサーとして利用可能なのは **jaeger** だけです。

空欄の場合には、OpenTracing のサポートは無効です。

RESOLVER
OpenResty で使用されるカスタム DNS リゾルバーを指定することができます。**RESOLVER** パラメーターが空欄の場合には、DNS リゾルバーは自動検出されます。

THREESCALE_CONFIG_FILE

ゲートウェイの設定が含まれる JSON ファイルへのパス。ゲートウェイが正常に動作するには、THREESCALE_PORTAL_ENDPOINT または **THREESCALE_CONFIG_FILE** のどちらかを指定する必要があります。これら 2 つの環境変数のうち、**THREESCALE_CONFIG_FILE** が優先されます。

ゲートウェイの設定が含まれるファイルをビルドする方法は、サービスの数により 2 とおりあります。

- **オプション 1:** URL を使用して管理ポータルから設定をダウンロードする。

  ```
  <schema>://<admin-portal-domain>/admin/api/nginx/spec.json
  ```

  以下の点を考慮してください。

  - エンドポイントのサービス数が 20 を超えると、制限が適用されます。
  - **https://account-admin.3scale.net/admin/api/nginx/spec.json** の例を確認してください。

- **オプション 2:** 利用可能な 3scale API エンドポイントを使用する。
  - **Proxy Config Show** または **Proxy Config Show Latest** のどちらかです。
  - 続いて、すべてのサービスで操作を繰り返し、設定ファイルをビルドします。このステップでは、利用可能な 3scale API エンドポイントまたは等価な 3scale toolbox コマンド を使用します。

コンテナーイメージを使用してゲートウェイをデプロイする場合:

1. イメージへのファイルを読み取り専用ボリュームとして設定します。
2. ボリュームをマウントした場所を示すパスを指定します。

設定ファイルの例については、examples フォルダーを参照してください。

THREESCALE_DEPLOYMENT_ENV

値: staging | production

デフォルト: production

この環境変数の値を使用して、新しい APIcast を使用する際に設定をダウンロードする環境 (3scale ステージングまたは実稼働環境) を定義します。

この値は、3scale Service Management API への承認/レポートリクエストのヘッダー **X-3scale-User-Agent** でも使用されます。この値は、3scale では統計のためだけに使用されます。

THREESCALE_PORTAL_ENDPOINT

パスワードおよびポータルエンドポイントが含まれる、以下の形式の URI。

<schema>://<password>@<admin-portal-domain>

ここで、

- **<password>** は、 プロバイダーキーまたは 3scale Account Management API の アクセストークンのいずれかです。
- **<admin-portal-domain>** は、3scale 管理ポータルにログインするための URL アドレスです。

例: **https://access-token@account-admin.3scale.net**

THREESCALE_PORTAL_ENDPOINT 環境変数を指定すると、ゲートウェイは初期化時に 3scale から設定をダウンロードします。この設定には、API の Integration ページで指定したすべての設定が含まれます。

この環境変数を使用して、マスター管理ポータルを使用して単一のゲートウェイを作成する こともできます。

ゲートウェイが正常に動作するには、**THREESCALE_PORTAL_ENDPOINT** または **THREESCALE_CONFIG_FILE** (優先) を指定する **必要があります**。

第8章 パフォーマンスを向上させるための APICAST 設定

本セクションでは、APIcast のパフォーマンスの問題をデバッグする際の全般的なガイドラインについて説明します。また、使用可能なキャッシュモードを紹介しパフォーマンスの向上にどのように役立つかを説明します、さらに、同期モードの詳細についても言及します。コンテンツは、以下のセクションで設定されています。

- 「全般的なガイドライン」
- 「デフォルトのキャッシング」
- 「非同期レポートスレッド」
- 「3scale Batcher ポリシー」

8.1. 全般的なガイドライン

典型的な APIcast デプロイメントで考慮すべき 3 つのコンポーネントを以下に示します。

- APIcast
- リクエストを承認し使用状況を追跡する 3scale バックエンドサーバー
- アップストリーム API

APIcast でパフォーマンスの問題が発生している場合には、以下の手順に従います。

- 問題の原因となっているコンポーネントを特定します。
- アップストリーム API のレイテンシーを測定し、APIcast と 3scale バックエンドサーバーで生じるレイテンシーを把握します。
- ベンチマーク試験を実施するのと同じツールを使用して、新たな計測を実施します。ただし、直接アップストリーム API をポイントするのではなく、APIcast をポイントします。

これらの結果を比較することで、APIcast と 3scale バックエンドサーバーで生じるレイテンシーを把握することができます。

ホスト型 (SaaS) システムと Self-managed APIcast の組み合わせにおいて、APIcast と 3scale バックエンドサーバーで生じるレイテンシーが高い場合には、以下の手順に従います。

1. APIcast がデプロイされているマシンから 3scale バックエンドサーバーにリクエストを送信します。
2. レイテンシーを測定します。

3scale バックエンドサーバーは、バージョンを返すエンドポイント https://su1.3scale.net/status を公開します。それに比べて、承認呼び出しは鍵、制限、およびキューのバックグラウンドジョブを検証するため、より多くのリソースを必要とします。3scale バックエンドサーバーはこれらのタスクを数ミリ秒で実行しますが、これには **/status** エンドポイントが処理するようなバージョン確認よりも多くの作業が必要です。たとえば、ご自分の APIcast 環境から **/status** へのリクエストに約 300 ミリ秒かかるとすると、キャッシュされないすべてのリクエストについて、承認にはより長い時間がかかります。

8.2. デフォルトのキャッシング

キャッシュされていないリクエストの場合には、フローは以下のようになります。

1. APIcast は、マッチするマッピングルールから使用状況のメトリクスを抽出します。
2. APIcast は、メトリクスおよびアプリケーションのクレデンシャルを 3scale バックエンドサーバーに送信します。
3. 3scale バックエンドサーバーは、以下の処理を実施します。
    a. アプリケーションキーおよび報告されたメトリクスの使用状況が定義された制限内であることを確認する。
    b. 報告されたメトリクスの使用状況を増やすバックグラウンドジョブをキューに入れる。
    c. リクエストを承認すべきかどうかを APIcast に応答する。
4. リクエストが承認されると、リクエストがアップストリームに送信されます。

この場合、3scale バックエンドサーバーが応答するまで、リクエストはアップストリームに到着しません。

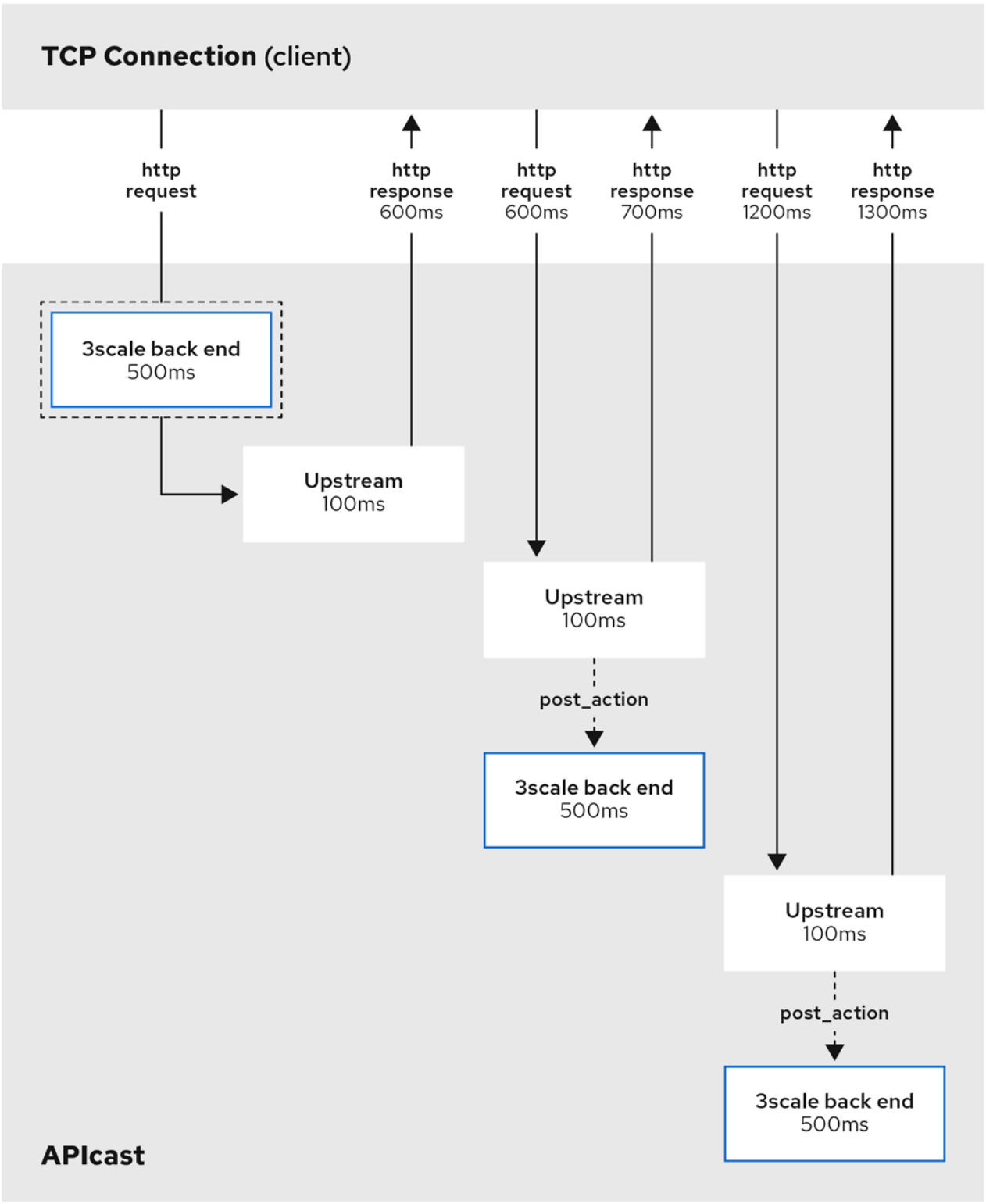
一方、デフォルトで有効になっているキャッシングメカニズムを使用した場合には、フローは以下のようになります。

- 3scale バックエンドサーバーへの承認呼び出しが承認された場合、APIcast はその結果をキャッシュに保存します。
- 同じクレデンシャルおよびメトリクスが使用される次のリクエストは、3scale バックエンドサーバーに照会する代わりに、キャッシュされたその承認を使用します。
- リクエストが承認されなかった場合、または APIcast がそのクレデンシャルを初めて受け取る場合には、APIcast は上述のように同期した状態で 3scale バックエンドサーバーに呼び出しを行います。

認証がキャッシュされている場合には、APIcast はまずアップストリームに呼び出しを行い、続いて **ポストアクション** と呼ばれるフェーズで 3scale バックエンドサーバーに呼び出しを行い、次のリクエスト用に承認をキャッシュに保存します。3scale バックエンドサーバーへの呼び出しはリクエスト時に行われる訳ではないので、レイテンシーが生じない点に注意してください。ただし、同じ接続で送信されたリクエストは、**ポストアクション** フェーズが終了するまで待つ必要があります。

クライアントが **キープアライブ** を使用していて、1 秒ごとにリクエストを送信するシナリオを考えてみます。アップストリームの応答時間が 100 ミリ秒で 3scale バックエンドサーバーへのレイテンシーが 500 ミリ秒の場合、クライアントは毎回 100 ミリ秒でレスポンスを得ます。アップストリームのレスポンスとレポートの合計は 600 ミリ秒かかります。これにより、次のリクエストを受け取るまでにさらに 400 ミリ秒かかります。

以下の図で、上述のデフォルトのキャッシュ動作を説明します。キャッシュメカニズムの動作は、Caching ポリシー を使用して変更することができます。



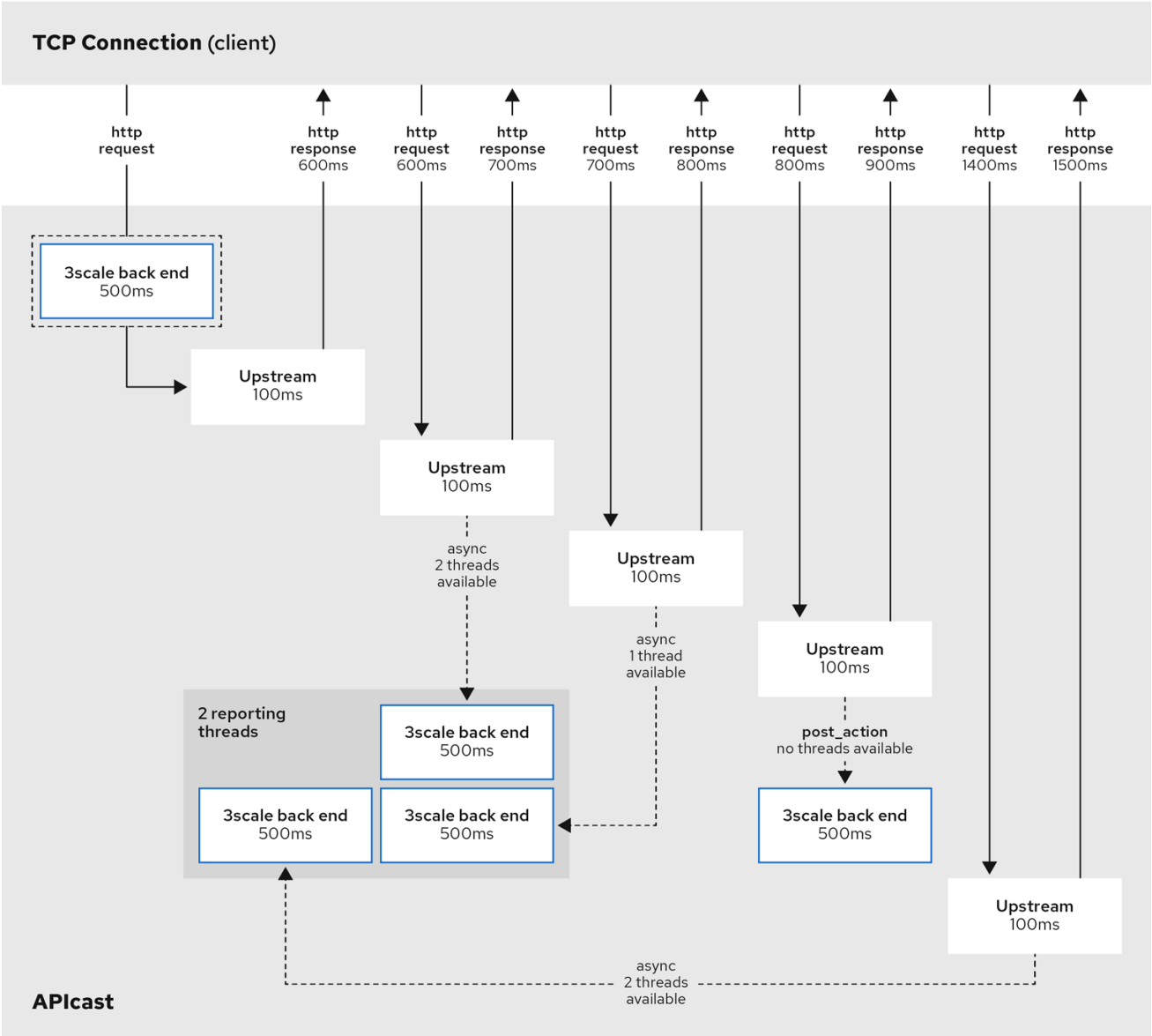
8.3. 非同期レポートスレッド

APIcast には、3scale バックエンドサーバーに対して承認するスレッドのプールを有効にする機能があります。この機能を有効にすると、APIcast はまず同期した状態で 3scale バックエンドサーバーに呼び出しを行い、マッピングルールがマッチするアプリケーションおよびメトリクスを検証します。この動作は、デフォルトで有効になっているキャッシュメカニズムを使用する場合と類似しています。違いは、プール内に空のレポートスレッドがある限り、それ以降の 3scale バックエンドサーバーへの呼び出しが完全に非同期状態で報告されることです。

レポートスレッドはゲートウェイ全体で共通的に使用され、すべてのサービス間で共有されます。2 番目の TCP 接続が確立されると、承認がすでにキャッシュされている限り、完全に非同期になります。空のレポートスレッドがない場合には、同期モードは標準の非同期モードにフォールバックし、ポストアクションフェーズでレポートを行います。

APICAST_REPORTING_THREADS 環境変数を使用して、この機能を有効にすることができます。

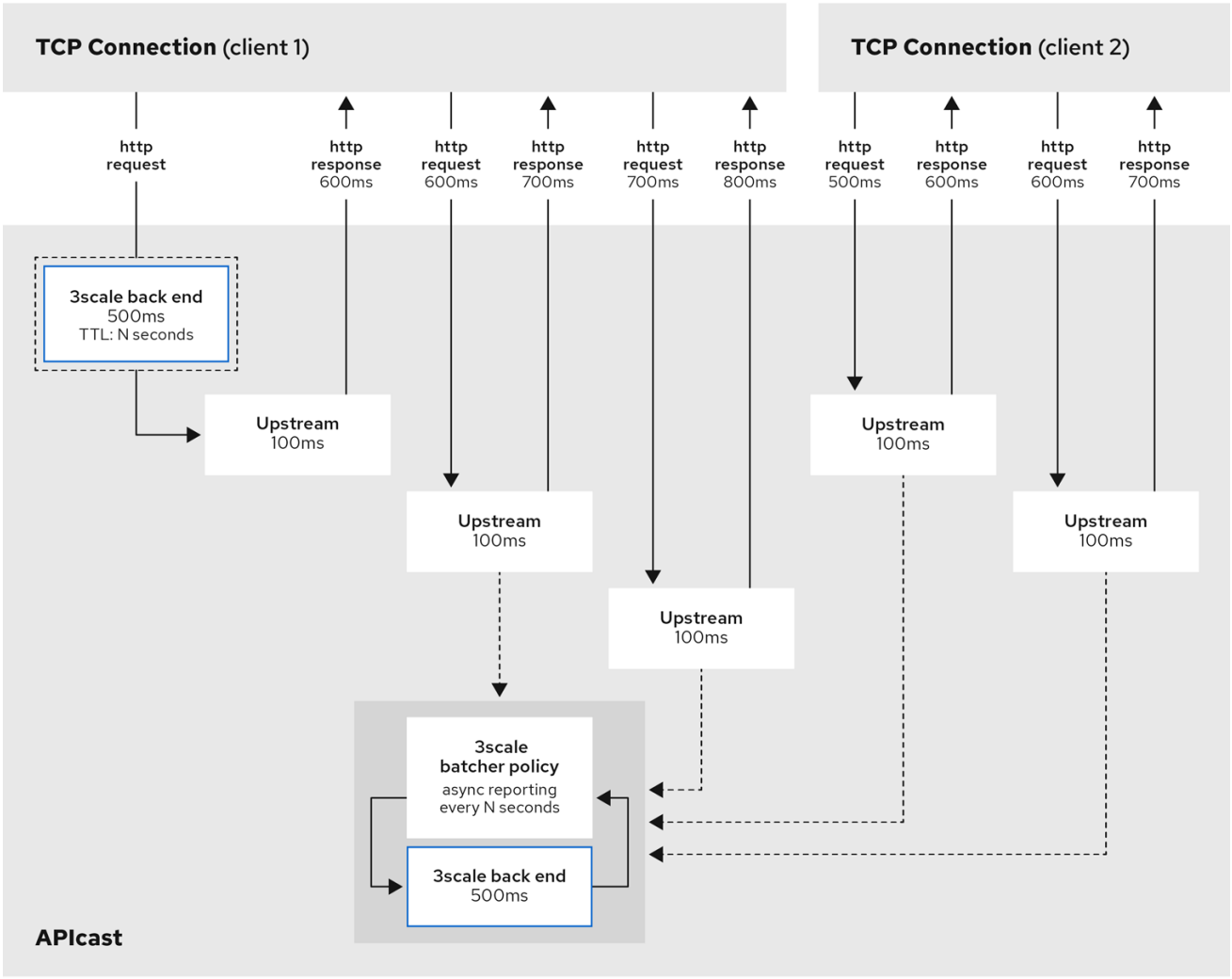
下記の図で、非同期レポートスレッドプールが機能する仕組みを説明します。



8.4. 3SCALE BATCHER ポリシー

デフォルトでは、APIcast はリクエストを受け取るたびに 1 回 3scale バックエンドサーバーに呼び出しを行います。3scale Batcher ポリシーの目的は、3scale バックエンドサーバーに対して行われるリクエストの数を大幅に減らすことにより、レイテンシーを低減しスループットを向上させることです。そのために、このポリシーは承認ステータスをキャッシュし、レポートをバッチ処理します。

詳細は、3scale Batcher ポリシーを参照してください。以下の図で、このポリシーが機能する仕組みを説明します。


TCP Connection (client 1)
TCP Connection (client 2)
http request
http response 600ms
http request 600ms
http response 700ms
http request 700ms
http response 800ms
http request 500ms
http response 600ms
http request 600ms
http response 700ms
3scale back end
500ms
TTL: N seconds
Upstream
100ms
Upstream
100ms
Upstream
100ms
Upstream
100ms
Upstream
100ms
3scale
batcher policy
async reporting
every N seconds
3scale back end
500ms
APIcast

第9章 PROMETHEUS への 3SCALE APICAST メトリクスの公開

重要

3scale の本リリースでは、Prometheus のインストールおよび設定はサポートされていません。オプションとして、コミュニティーバージョンの Prometheus を使用して、APIcast 管理下の API サービスのメトリクスおよび警告を視覚化することができます。

9.1. PROMETHEUS の概要

Prometheus はオープンソースのシステム監視用ツールキットです。Prometheus を使用して、Red Hat OpenShift 環境にデプロイされた 3scale APIcast のサービスを監視することができます。

Prometheus を使用してサービスを監視するには、サービスは Prometheus エンドポイントを公開する必要があります。このエンドポイントは、メトリクスのリストおよびメトリクスの現在の値を公開する HTTP インターフェイスです。Prometheus は定期的にこれらのターゲット定義のエンドポイントを収集し、収集したデータをそのデータベースに書き込みます。

9.1.1. Prometheus クエリー

Prometheus UI において、Prometheus Query Language (PromQL) でクエリーを作成してメトリクス情報を抽出することができます。PromQL を使用すると、リアルタイムに時系列データを選択して集計することができます。

たとえば、以下のクエリーを使用すると、Prometheus が直近の 5 分間に記録したすべて時系列データから、メトリクス名が **http_requests_total** の値をすべて選択することができます。

```
http_requests_total[5m]
```

メトリックの **ラベル** (キー/値のペア) を指定して、クエリーの結果をさらに定義または絞り込むことができます。たとえば、以下のクエリーを使用すると、Prometheus が直近の 5 分間に記録したすべて時系列データから、メトリクス名が **http_requests_total** でかつ **job** ラベルが **integration** に設定されている値をすべて選択することができます。

```
http_requests_total{job="integration"}[5m]
```

クエリーの結果は、Prometheus の表示ブラウザーでグラフや表として表示したり、Prometheus HTTP API を使用して外部のシステムで処理したりすることができます。Prometheus はデータのグラフィカルビューを提供します。Prometheus メトリクスを表示するより安定したグラフィカルダッシュボードとしては、Grafana を選択するのが一般的です。

PromQL 言語を使用して、Prometheus alertmanager ツールで警告を設定することもできます。

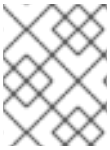
注記

Grafana はコミュニティーがサポートする機能です。Grafana をデプロイして Red Hat 3scale プロダクトを監視する設定は、Red Hat の実稼働環境のサービスレベルアグリーメント (SLA) の対象外です。

9.2. APICAST と PROMETHEUS のインテグレーション

APIcast と Prometheus のインテグレーションは、以下のデプロイメントオプションで利用することができます。

- Self-managed APIcast: ホスト型 3scale およびオンプレミス型 API Management 両方との組み合わせに対応
- オンプレミス型 3scale の Embedded APIcast



注記

APIcast と Prometheus のインテグレーションは、ホスト型 API Management と Hosted APIcast の組み合わせでは利用することができません。

デフォルトでは、Prometheus は 表9.2「3scale APIcast 用 Prometheus のデフォルトメトリクス」 に記載された APIcast メトリクスを監視することができます。

9.2.1. 追加オプション

オプションとして、OpenShift クラスターのクラスター管理者アクセス権限があれば、**total_response_time_seconds**、**upstream_response_time_seconds**、および **upstream_status** メトリクスを拡張して **service_id** および **service_system_name** ラベルを含めることができます。これらのメトリクスを拡張するには、以下のコマンドを使用して **APICAST_EXTENDED_METRICS** OpenShift 環境変数を **true** に設定します。

```
oc set env dc/apicast APICAST_EXTENDED_METRICS=true
```

3scale Batcher ポリシー (「3scale Batcher」 に詳細を記載) を使用している場合、Prometheus は 表9.3「3scale APIcast Batch ポリシー用 Prometheus メトリクス」 に記載されたメトリクスも監視することができます。

注記

メトリクスに値がない場合、Prometheus はメトリクスを非表示にします。たとえば、**nginx_error_log** に報告するエラーがない場合、Prometheus は **nginx_error_log** メトリクスを表示しません。**Nginx_error_log** メトリクスは、値がある場合にのみ表示されます。

関連情報

Prometheus についての情報は、PROMETHEUS: スタートガイド を参照してください。

9.3. 3SCALE APICAST 用 OPENSHIFT 環境変数

Prometheus インスタンスを設定するには、表9.1「3scale APIcast 用 Prometheus 環境変数」 に記載の OpenShift 環境変数を設定します。

表9.1 3scale APIcast 用 Prometheus 環境変数

環境変数	説明	デフォルト

環境変数	説明	デフォルト
APICAST_EXTENDED_METRICS	Prometheus メトリクスに関する追加情報を有効にするブール値。以下のメトリクスでは **service_id** および **service_system_name** ラベルを使用することができ、APIcast をより詳細に設定することができます。 • **total_response_time_seconds** • **upstream_response_time_seconds** • **upstream_status**	**false**

関連情報

環境変数の設定については、該当する OpenShift のガイドを参照してください。

- OpenShift 4: アプリケーション
- OpenShift 3.11:開発者ガイド

サポート対象設定の情報については、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。

9.4. PROMETHEUS に公開される 3SCALE APICAST メトリクス

3scale APIcast の監視用に Prometheus を設定すると、デフォルトでは 表9.2「3scale APIcast 用 Prometheus のデフォルトメトリクス」に記載されたメトリクスを監視することができます。

表9.3「3scale APIcast Batch ポリシー用 Prometheus メトリクス」 に記載されたメトリクスは、3scale Batcher ポリシーを使用する場合にのみ利用することができます。

表9.2 3scale APIcast 用 Prometheus のデフォルトメトリクス

メトリクス	説明	タイプ	ラベル
nginx_http_connections	HTTP 接続の数	ゲージ	state (accepted、active、handled、reading、total、waiting、writing)
nginx_error_log	APIcast エラー	カウンター	**level (debug、info、notice、warn、error、crit、alert、emerg)**

メトリクス	説明	タイプ	ラベル
openresty_shdict_capacity	ワーカー間で共有されるディクショナリーの容量	ゲージ	**dict** (すべてのディクショナリーで共通)
openresty_shdict_free_space	ワーカー間で共有されるディクショナリーの空き容量	ゲージ	**dict** (すべてのディクショナリーで共通)
nginx_metric_errors_total	メトリクスを管理する Lua ライブラリーのエラーの数	カウンター	none
total_response_time_seconds	クライアントにレスポンスを送信するのに必要な時間 (秒単位) **注記:service_id** および **service_system_name** ラベルにアクセスするためには、「APIcast と Prometheus のインテグレーション」に記載のとおり **APICAST_EXTENDED_METRICS** 環境変数を **true** に設定する必要があります。	ヒストグラム	**service_id**、**service_system_name**
upstream_response_time_seconds	アップストリームサーバーからの応答時間 (秒単位) **注記:service_id** および **service_system_name** ラベルにアクセスするためには、「APIcast と Prometheus のインテグレーション」に記載のとおり **APICAST_EXTENDED_METRICS** 環境変数を **true** に設定する必要があります。	ヒストグラム	**service_id**、**service_system_name**

メトリクス	説明	タイプ	ラベル
upstream_status	アップストリームサーバーからの HTTP ステータス **注記:service_id** および **service_system_name** ラベルにアクセスするためには、「APIcast と Prometheus のインテグレーション」に記載のとおり **APICAST_EXTENDED_METRICS** 環境変数を **true** に設定する必要があります。	カウンター	**status**、**service_id**、**service_system_name**
threescale_backend_calls	3scale バックエンド (Apisonator) に対する承認および報告リクエスト	カウンター	**endpoint** (**authrep**、**auth**、**report**)、**status** (**2xx**、**4xx**、**5xx**)

表9.3 3scale APIcast Batch ポリシー用 Prometheus メトリクス

メトリクス	説明	タイプ	ラベル
batching_policy_auths_cache_hits	3scale Batcher ポリシーの承認キャッシュのヒット数	カウンター	none
batching_policy_auths_cache_misses	3scale Batcher ポリシーの承認キャッシュのミス数	カウンター	none

パート II. API のバージョン管理

第10章 API のバージョン管理

Red Hat 3scale API Management 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。

10.1. 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 とおり説明します。

- 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

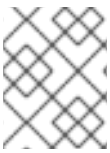
10.2. 前提条件

- 以下に示すクイックスタートガイドを使用する前に、First steps with 3scale の手順を完了してください。

10.3. 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



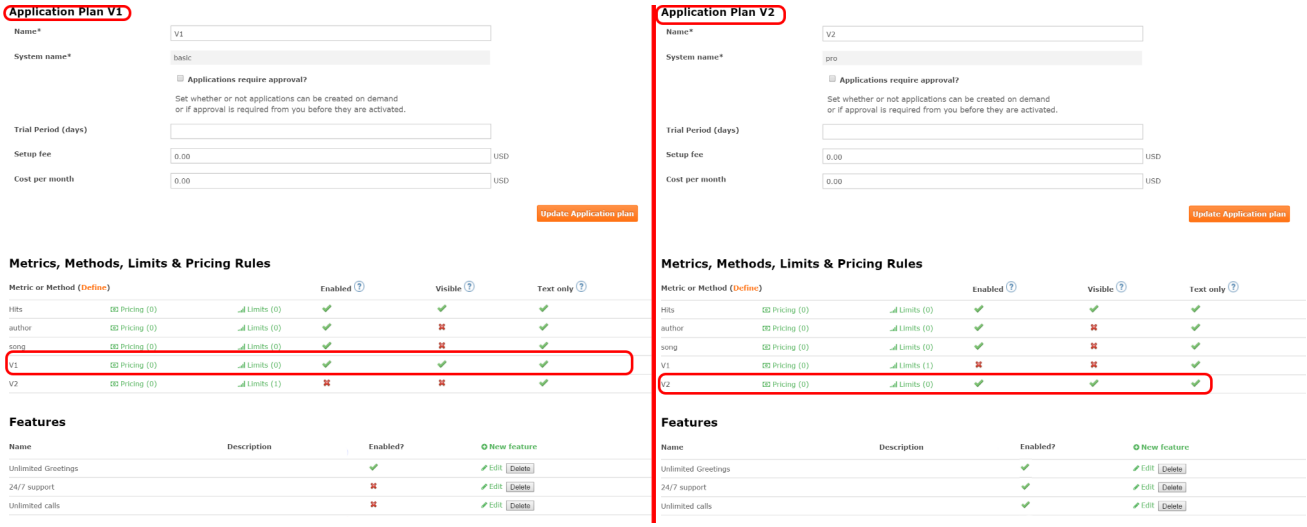
注記

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。

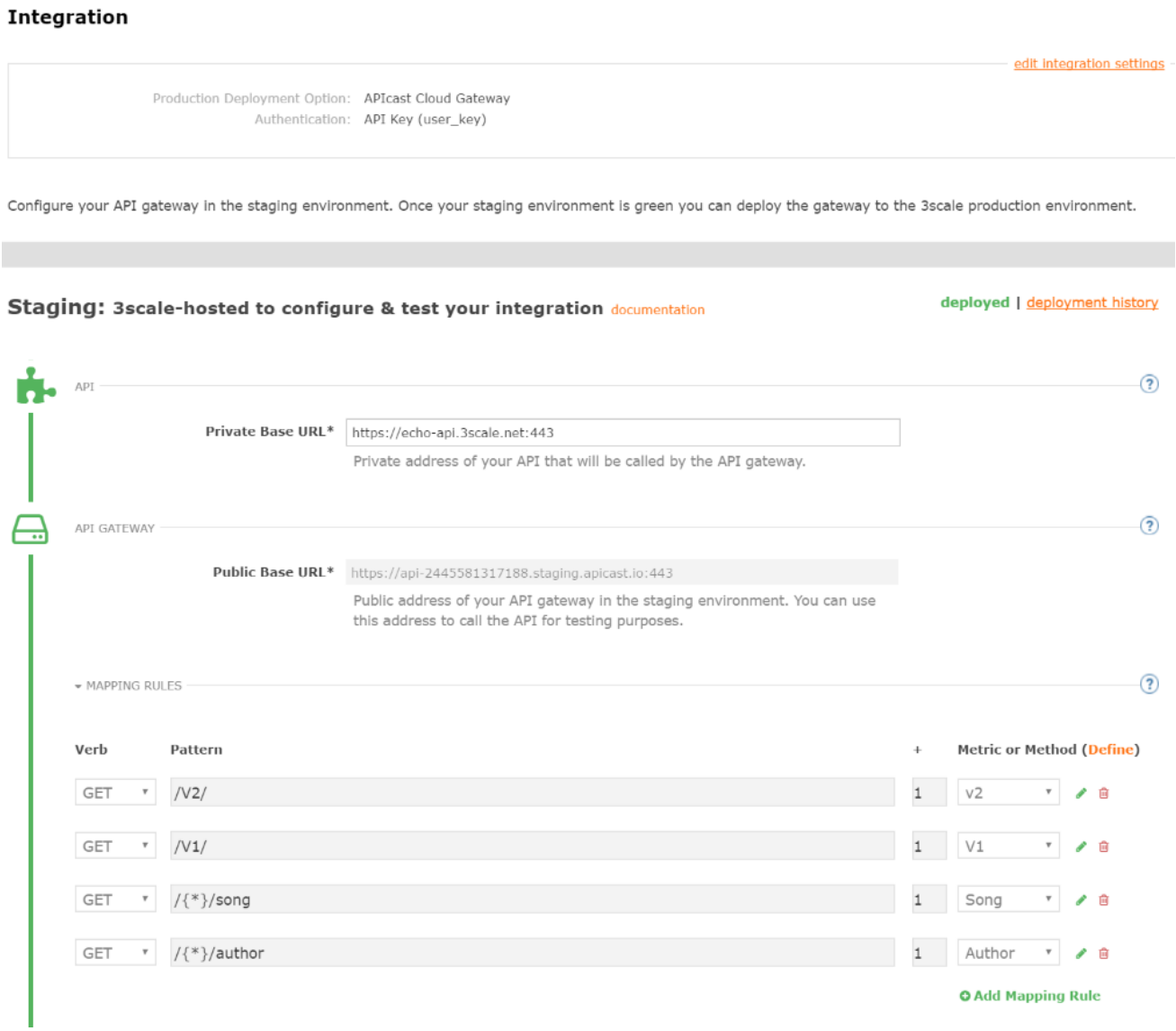
3scale 機能の柔軟性を以下のスクリーンショットに示します。

図10.1 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。

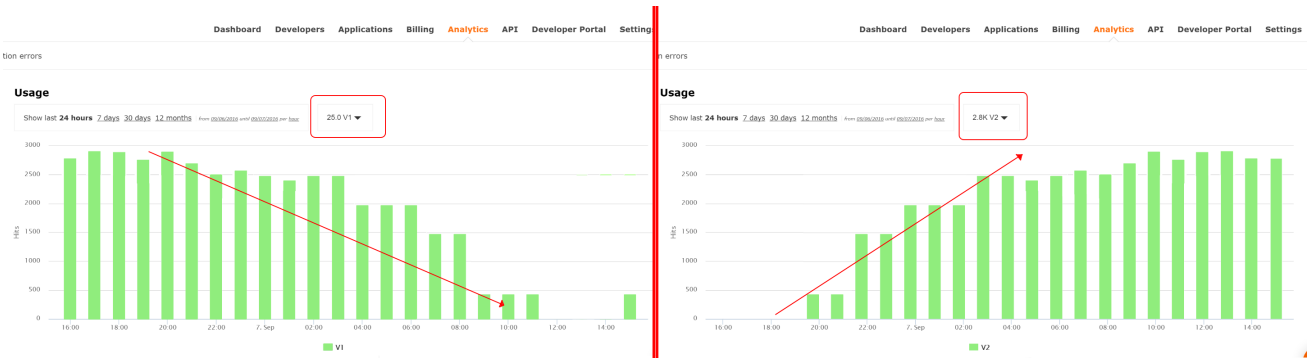
図10.2 メトリクスへの URI のマッピング



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10.3 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。

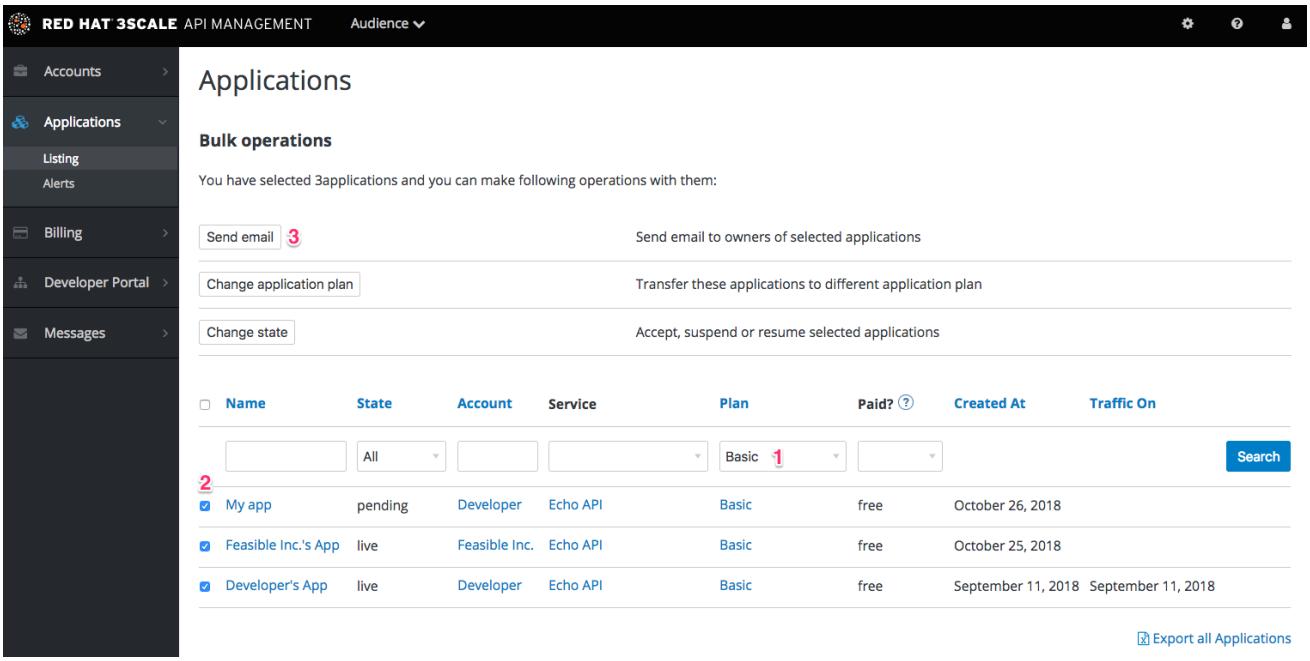
1. **Audience** > **Applications** > **Listing**の順に移動し、廃止の通知を送信するアプリケーションプランを選択して **Search** をクリックし、リストを絞り込みます。

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アプリケーション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、手順に従って選択したアプリケーションのオーナー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10.4 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。

10.4. エンドポイントによるバージョン管理

エンドポイントによるバージョン管理では、API のバージョンごとに異なるエンドポイントを設定することができます (例: **api.cons.com/author_v1**)。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。

10.5. 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。

- この手法の最も重要なメリットは、開発者の指定する URL およびエンドポイントが変更されないことです。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11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。

11.1. サポートされる認証パターン

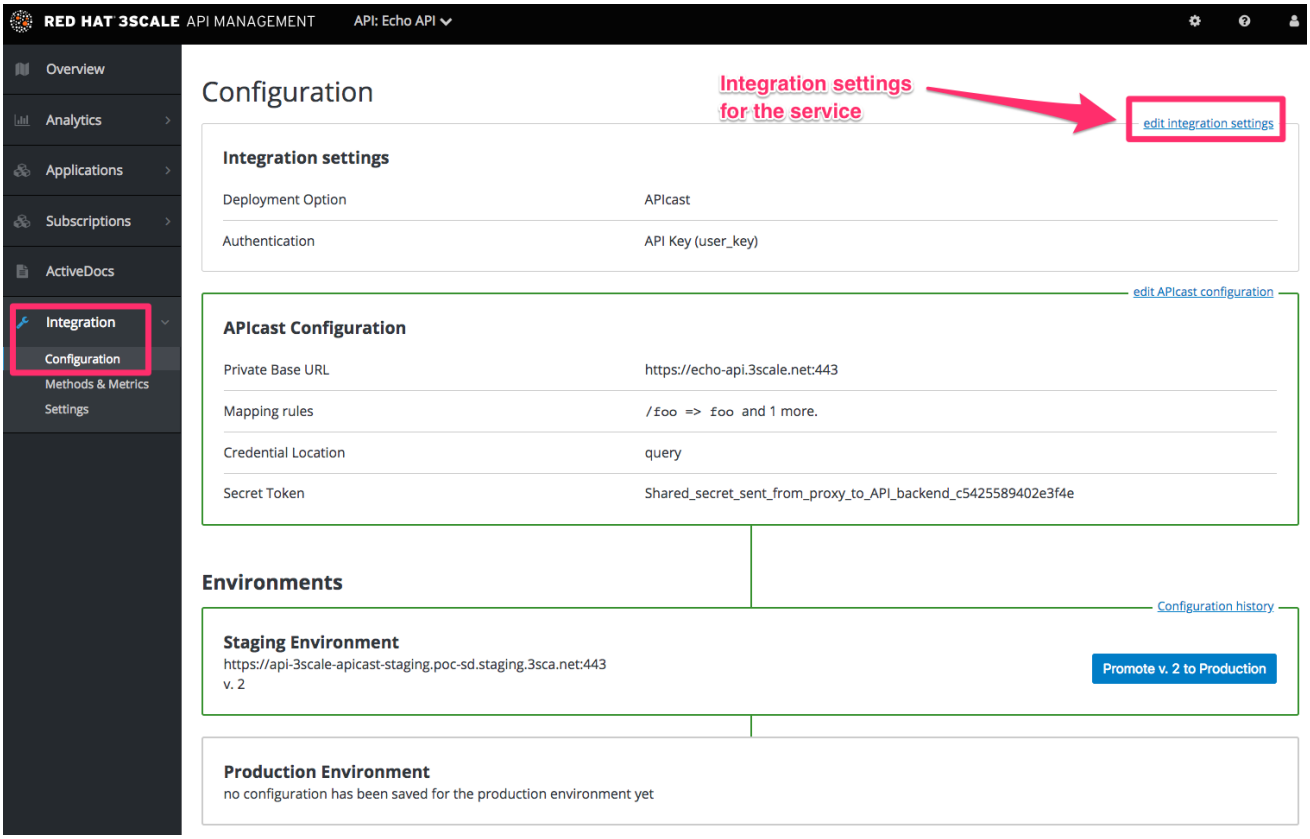
3scale では、以下の認証パターンが標準でサポートされます。

- **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、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**

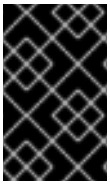
11.2. 認証パターンのセットアップ

11.2.1. 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。

11.2.2. 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

11.2.3. 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。

11.2.4. 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。

11.3. 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。

11.3.1. 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。

11.3.2. 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。
- また、各アプリケーション は 1 つから 5 つまでの数の **アプリケーションキー** (App_Key) を持ちます。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。

11.3.3. OpenID Connect

OpenID Connect による認証に関する詳細は、OpenID Connect インテグレーションの章を参照してください。

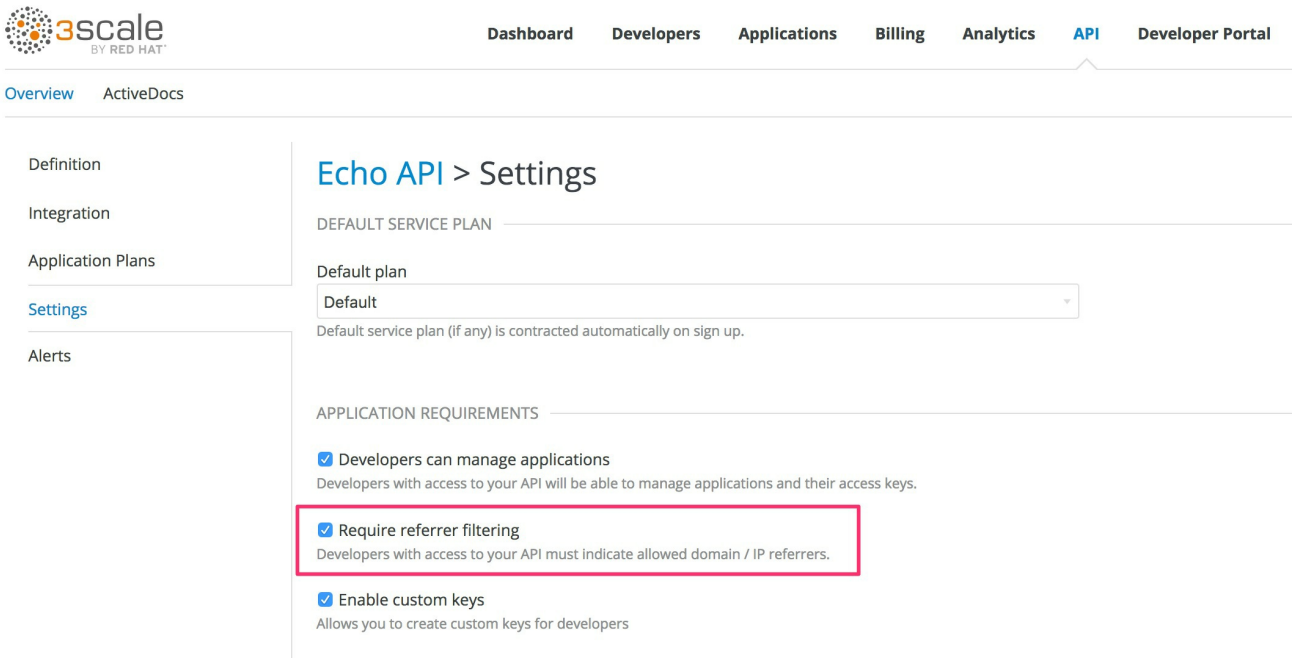
11.4. 参照元フィルター機能

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

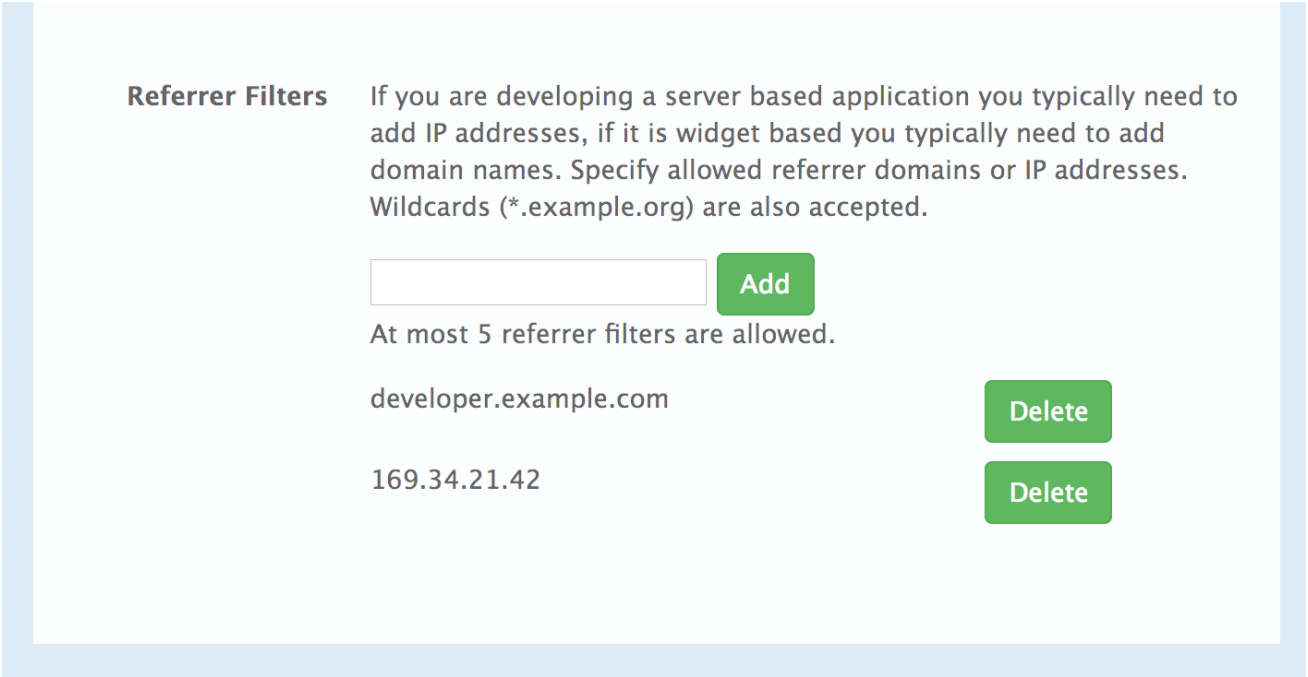
は、RFC 7231 のセクション 5.5.2Referer に説明があります。

参照元フィルター機能が動作するためには、サービスポリシーチェーンで APIcast Referrer ポリシー を有効にする必要があります。

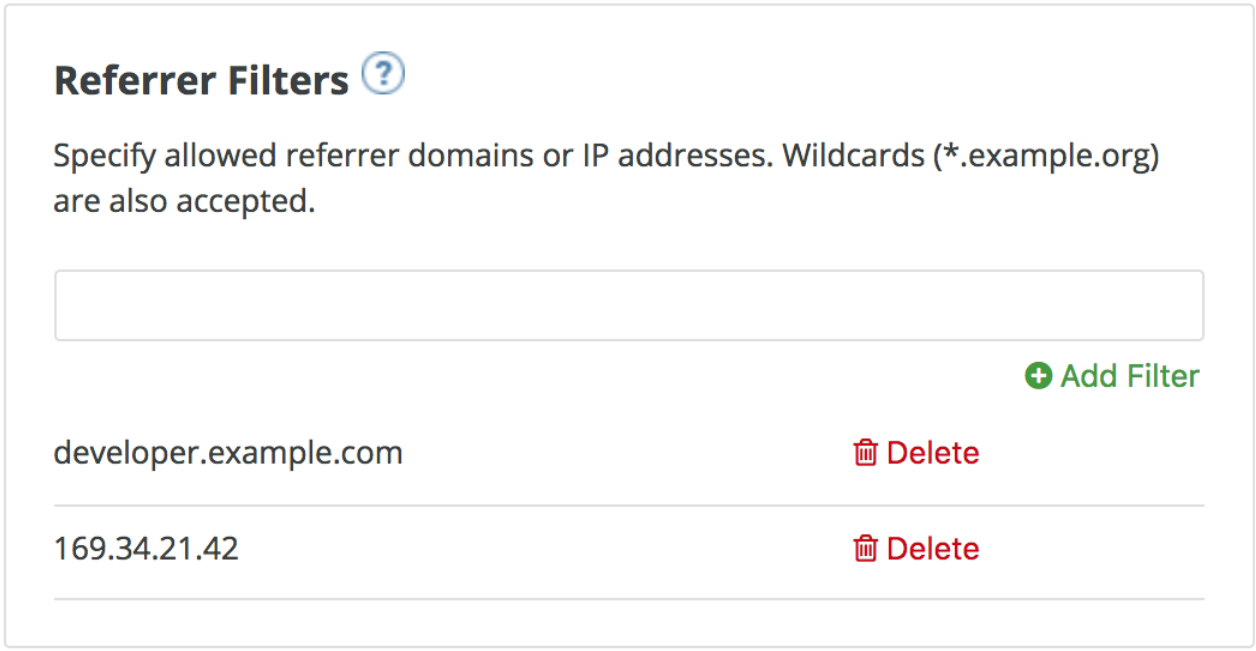
参照元フィルター機能を有効にするには、**[your_API_name] > Applications > Settings > Usage Rules** の順に移動します。**Require referrer filtering** を選択し、**Update Product** をクリックします。



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。

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。

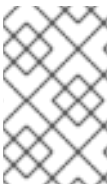
referrer パラメーターが渡されるか	アプリケーションに参照元フィルターが設定されているか	referrer パラメーターの値	HTTP レスポンス	レスポンスのボディー
はい	はい	参照元フィルターと一致する	200 OK	**<status><authorized>true</authorized></status>**
はい	いいえ	参照元フィルターと一致する	200 OK	**<status><authorized>true</authorized></status>**

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12章 3SCALE と OPENID CONNECT アイデンティティープロバイダーの統合

API 要求を認証するために、3scale は OpenID Connect 仕様 に準拠するアイデンティティープロバイダーと統合できます。3scale との互換性を確保するために、アイデンティティープロバイダーは Red Hat Single Sign-On(RH-SSO) または デフォルトの Keycloak クライアント登録 を実装するサードパーティーのアイデンティティープロバイダーに指定できます。3scale API ゲートウェイ (APIcast) との互換性のために、OpenID Connect を実装するアイデンティティープロバイダーを使用できます。



注記

3scale では、RFC 7591 Dynamic Client Registration Mechanism は使用されません。3scale と OpenID Connect アイデンティティープロバイダー間で互換性を完全に確保するため、デフォルトの Keycloak クライアント登録の依存関係があります。

OpenID Connect のベースは OAuth 2.0 Authorization Framework (RFC 6749) です。OpenID Connect は API 要求で JSON Web Token(JWT)(RFC 7519) を使用してその要求を認証します。3scale を OpenID Connect アイデンティティープロバイダーと統合する場合に、プロセスには 主要な部分が 2 つあります。

- APIcast は要求内の JWT を解析して検証します。成功すると、APIcast は API コンシューマークライアントアプリケーションの ID を認証します。
- 3scale Zync コンポーネントは、3scale アプリケーションの詳細を OpenID Connect アイデンティティープロバイダーと同期します。

RH-SSO が OpenID Connect アイデンティティープロバイダーの場合には、3scale ではこれらのインテグレーションポイントの両方がサポートされます。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。ただし、RH-SSO は必須ではありません。OpenID Connect 仕様およびデフォルトの Keycloak クライアント登録をサポートするアイデンティティープロバイダーを使用できます。APIcast のインテグレーションは、RH-SSO および ForgeRock でテストされています。

以下のセクションでは、OpenID Connect アイデンティティープロバイダーを使用するように 3scale を設定する方法について説明します。

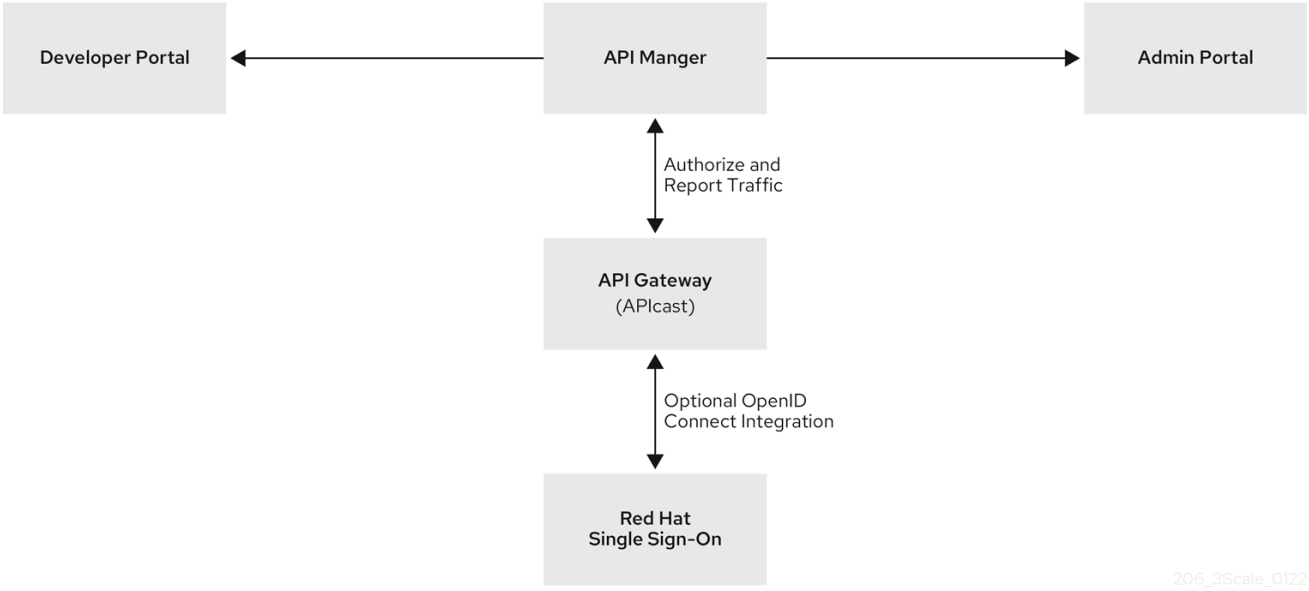
- 3scale と OpenID Connect アイデンティティープロバイダーのインテグレーションの概要
- APIcast が JSON Web トークンを処理する方法
- 3scale Zync がアプリケーションの情報を OpenID Connect アイデンティティープロバイダーと同期する方法
- OpenID Connect アイデンティティープロバイダーとしての 3scale と Red Hat Single Sign-On のインテグレーション
- 3scale とサードパーティー OpenID Connect アイデンティティープロバイダーの統合
- OpenID Connect アイデンティティープロバイダーを使用した 3scale のテスト
- OpenID Connect アイデンティティープロバイダーおよび OpenID Connect アイデンティティープロバイダーとの 3scale インテグレーションの例

12.1. 3SCALE と OPENID CONNECT アイデンティティープロバイダーのインテグレーションの概要

主要な 3scale コンポーネントはそれぞれ、以下のように認証に参加します。

- APIcast は、API コンシューマーアプリケーションが提供する認証トークンの信頼性を検証します。デフォルトの 3scale デプロイメントでは、APIcast は API プロダクトの OpenID Connect 設定の自動検出を実装するので、このような検証が可能です。
- API プロバイダーは管理ポータルを使用して認証フローを設定します。
- 3scale 管理対象の API が標準の API キーまたはアプリケーション ID とキーペアで要求を認証しない場合には、API プロバイダーは 3scale を OpenID Connect アイデンティティープロバイダーと統合する必要があります。以下の図は、OpenID Connect アイデンティティープロバイダーは Red Hat Single Sign-On (RH-SSO) です。
- 認証設定とライブの開発者ポータルにより、API コンシューマーは開発者ポータルを使用して特定の 3scale API プロダクトにアクセスできるようにするアプリケーションプランにサブスクライブします。
- OpenID Connect が 3scale と統合されると、サブスクリプションは API コンシューマーアプリケーションに設定されたフローをトリガーして、OpenID Connect アイデンティティープロバイダーから JSON Web Tokens (JWT) を取得します。API プロバイダーは OpenID Connect を使用するように API プロダクトを設定するタイミングで、このフローを指定します。

図12.1 OpenID Connect アイデンティティープロバイダーと主要な 3scale コンポーネントを示しています。



アプリケーションプランにサブスクライブした後に、API コンシューマーは統合された OpenID Connect アイデンティティープロバイダーから認証情報を取得します。これらの認証情報により、API コンシューマー アプリケーションがアップストリーム API (API コンシューマーがアクセスできる 3scale API プロダクトによって提供される API) に送信する要求の認証が可能になります。

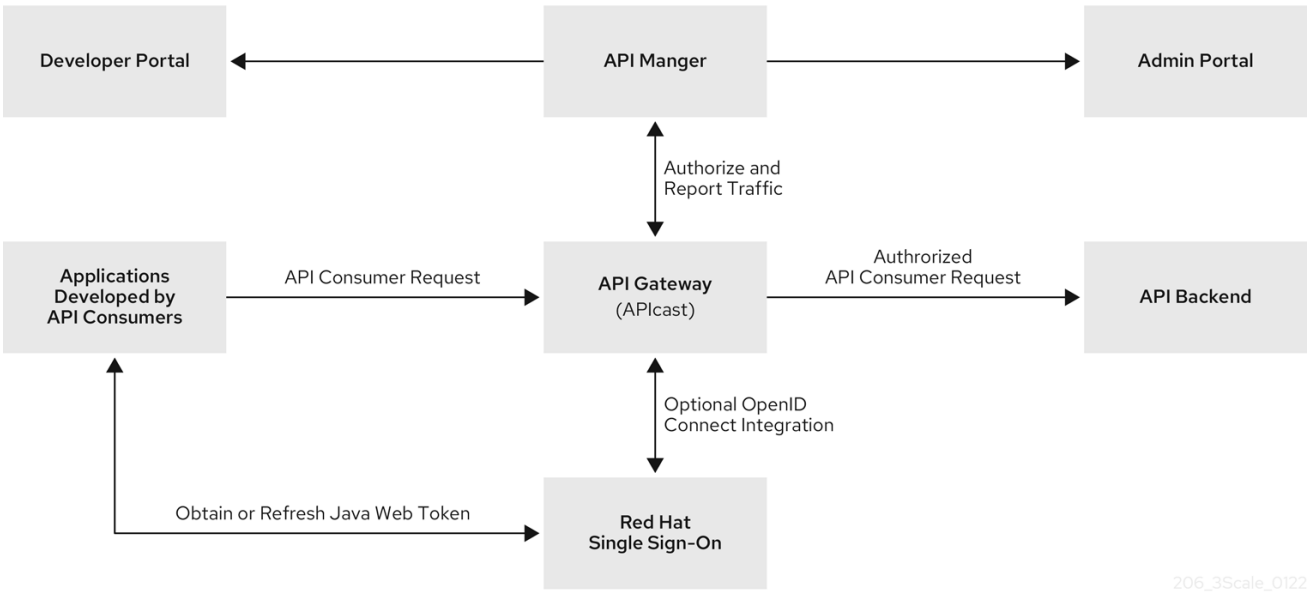
認証情報には、クライアント ID とクライアントシークレットが含まれます。API コンシューマーによって作成されたアプリケーションは、これらの資格情報を使用して、OpenID Connect ID プロバイダーから JSON Web Token (JWT) を取得します。OpenID Connect で 3scale インテグレーションを設定する

場合には、API コンシューマーアプリケーションが JWT を取得する方法のフローを選択します。RH-SSO でデフォルトの **Authorization Code** フローを使用する API コンシューマーアプリケーションでは、アプリケーションが以下を実行する必要があります。

1. アップストリーム API バックエンドへの最初の要求の前に、OpenID Connect アイデンティティープロバイダーで OAuth 承認フローを開始します。承認コードフローはエンドユーザーを RH-SSO にリダイレクトします。エンドユーザーはログインして認可コードを取得します。
2. JWT の認可コードを交換します。
3. 認証時に RH-SSO から JWT を受信します。
4. JWT が含まれる API 要求をアップストリーム API バックエンドに送信します。
5. 有効期限が切れるまで、同じ JWT で後続の API 要求を送信します。
6. JWT を更新するか、OpenID Connect アイデンティティープロバイダーに新しい要求を送信し、新しい JWT を取得します。必要なアクションは OpenID Connect アイデンティティープロバイダーによって異なります。

APIcast は API コンシューマーから要求を受け取り、その要求内の JWT を確認します。APIcast が JWT を検証する場合には、APIcast は JWT などの要求をアップストリーム API バックエンドに送信します。

図12.2 OpenID Connect アイデンティティープロバイダーは RH-SSO ですが、他の OpenID Connect アイデンティティープロバイダーの設定が可能です。



12.2. APICAST が JSON WEB トークンを処理する方法

APIcast は、API コンシューマーアプリケーションの認証時に OpenID Connect アイデンティティープロバイダーが返す JSON Web Token (JWT) を確認して各要求を処理します。要求には、統合された OpenID Connect アイデンティティープロバイダーが発行した形式の JWT が含まれます。JWT は **Authorization** ヘッダーに配置し、**Bearer** スキーマを使用する必要があります。たとえば、ヘッダーは以下のようになります。

```
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2lkcC5leGFtcGxlLmNvbSIsInN1YiI
```

```
6ImFiYzEyMyIsIm5iZiI6MTUzNzg5MjQ5NCwiZXhwIjoxNTM3ODk2MDk0LCJpYXQiOjE1Mzc4OTI0OTQsImp0aSI6ImlkMTIzNDU2IiwidHlwIjoiQmVhcmVyIn0.LM2PSmQ0k8mR7eDS_Z8iRdGta-Ea-pJRrf4C6bAiKz-Nzhxpm7fF7oV3BOipFmimwkQ_-mw3kN--oOc3vU1RE4FTCQGbzO1SAWHOZqG5ZUx5ugaASY-hUHIohy6PC7dQl0e2NlAeqqg4MuZtEwrpESJW-VnGdljrAS0HsXzd6nENM0Z_ofo4ZdTKvIKsk2KrdyVBOcjgVjYongtppR0cw30FwnpqfeCkuATeINN5OKHXOibRA24pQyIF1s81nnmxLnjnVbu24SFE34aMGRXYzs4icMI8sK65eKxbvwV3PIG3mM0C4ilZPO26doP0YrLfVwFcqEirmENUAcHXz7NuvA
```

JWT を安全にデコードするためのオープンソースツールが多数あります。公開 Web ツールで JWT をデコードしないように注意してください。デコードされた JWT では、トークンに次の 3 つの部分があることを確認できます。

- ヘッダーでは、トークンの形成方法と、トークンの署名に使用されたアルゴリズムに関する情報を指定します。
- ペイロードは、要求を送信した API コンシューマーを識別します。この詳細には、この API コンシューマーが実行できる読み取りおよび書き込みアクション、API コンシューマーのメールアドレス、API コンシューマーに関するその他の情報を含めることができます。
- 署名は、トークンが変更されていないことを示す暗号署名です。

APIcast は、次の特性があるか JWT をチェックします。

- **整合性**:JWT が悪意のあるユーザーによって変更されているか ?署名は有効か ?
  JWT には、既知の発行者がトークンに署名したことを確認するために、トークンの受信者が検証できる署名が含まれています。また、この検証により、コンテンツが作成済みであることを確認できます。3scale では、公開鍵と秘密鍵のペアに基づいて RSA 署名がサポートされます。発行者は秘密鍵を使用して JWT に署名します。APIcast は公開鍵を使用してトークンを検証します。APIcast は OpenID Connect Discovery を使用して JSON Web キー (JWK) を取得し、JWT 署名を検証するために使用できます。
- **タイミング**:現在の時刻は、トークンが処理できるようになるタイミングより後か ?JWT の有効期限が切れているか ?つまり、APIcast は JWT **nbf** (時間以降) および **exp** (有効期限) クレームを確認します。
- **発行者**:JWT は APIcast として知られる OpenID Connect アイデンティティープロバイダーによって発行されたか ?つまり、APIcast は JWT で指定された発行者が **OpenID Connect Issuer** フィールドで設定された API プロバイダーと同じ発行者であることを確認します。発行者の指定は、3scale と OpenID Connect アイデンティティープロバイダーを統合する手順の一部です。これは、JWT **iss** クレームです。
- **クライアント ID**:トークンに APIcast として知られる 3scale クライアントアプリケーション ID が含まれているか ?このクライアント ID は、3scale を OpenID Connect アイデンティティープロバイダーと統合する手順に指定されている API プロバイダーが **ClientID Token Claim** と一致する必要があります。これは、JWT の **azp** (JWT の発行先の承認済みのパーティー) クレームと **aud** (オーディエンス) クレームです。

JWT の検証または承認の確認に失敗すると、APIcast は **Authentication failed** エラーを返します。それ以外の場合は、APIcast は要求を 3scale アップストリーム API バックエンドに送信します。**Authorization** ヘッダーは要求にそのまま残るので、API バックエンドは JWT 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。

12.3. 3SCALE ZYNC がアプリケーションの情報を OPENID CONNECT アイデンティティープロバイダーと同期する方法

Zync は、3scale アプリケーションに関するデータを OpenID Connect アイデンティティープロバイダーに確実にプッシュする 3scale のコンポーネントです。この対話では、3scale アプリケーションは OpenID Connect アイデンティティープロバイダークライアントに対応します。つまり、Zync は OpenID Connect アイデンティティープロバイダーと通信して OpenID Connect クライアントの作成、更新、および削除を行います。

Zync は Keycloak のデフォルトのクライアント登録 を実装します。この API を使用する場合は、クライアントの表現が Keycloak および RH-SSO 固有であることを意味します。アイデンティティープロバイダーは、3scale アプリケーションの認証クレデンシャルであるクライアント ID およびクライアントシークレットを返します。

3scale がアプリケーションの作成、更新、または削除を行うたびに、Zync は OpenID Connect アイデンティティープロバイダーと通信し、対応するクライアントを適宜更新します。同期を成功させるには、指定の 3scale プロダクトに以下の設定が必要です。

- 認証メカニズムは OpenID Connect です。
- **OpenID Connect Issuer Type** は以下のいずれかになります。
  - **RH-SSO**: Red Hat Single Sign-On が OpenID Connect アイデンティティープロバイダーの場合。この発行者タイプを使用すると、Zync はクライアント登録要求を Keycloak/RH-SSO のデフォルトクライアント登録 API に送信します。
  - **REST API**: 他の OpenID Connect アイデンティティープロバイダーの場合。この発行者タイプを使用すると、Zync は Zync REST API の例 に示すようにクライアント登録要求を送信します。
- 以下の URL は **OpenID Connect Issuer**:**http://id:/api_endpoint** です。

OpenShift クラスターにデプロイする場合には、2 つの Zync プロセスがあります。

- **zync** は、**system-sidekiq** から通知を受け取り、バックグラウンドジョブを **zync-que** にキューに追加する REST API です。通知には、新規、更新、および削除された 3scale アプリケーションに関するものがあります。
- **Zync-que** は、**system-app** と OpenID Connect アイデンティティープロバイダーと通信するこれらのバックグラウンドジョブを処理します。たとえば、RH-SSO が設定された OpenID Connect アイデンティティープロバイダーの場合には、Zync は RH-SSO レルムでクライアントを作成、更新、および削除します。

12.4. OPENID CONNECT アイデンティティープロバイダーとしての 3SCALE と RED HAT SINGLE SIGN-ON のインテグレーション

API プロバイダーは、3scale を OpenID Connect アイデンティティープロバイダーとして Red Hat Single Sign-On (RH-SSO) と統合できます。以下の手順は、API 要求の認証に OpenID Connect を必要とする 3scale API プロダクト向けです。

この手順の一部で、Zync が RH-SSO と通信してトークンを交換するため、3scale Zync と RH-SSO との間で SSL 接続を確立します。Zync と RH-SSO との間に SSL 接続を設定しないと、トークンはリッスンしているすべてに対してオープンになります。

3scale 2.2 以降のバージョンでは、SSL_CERT_FILE 環境変数により、RH-SSO のカスタム CA 証明書がサポートされます。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。OpenID Connect アイデンティティープロバイダーとして 3scale と RH-SSO を統合する際に、以下の要素を次の順序で設定します。

- RH-SSO が信頼できる認証局 (CA) が発行する証明書を **使用しない** 場合は、3scale Zync がカスタム CA 証明書を使用するように設定する必要があります。信頼できる CA が発行する証明書を RH-SSO が使用する場合には、この作業は必要ありません。
- 3scale クライアントを使用するように RH-SSO を設定します。
- RH-SSO と連携するように 3scale を設定します。

前提条件

- RH-SSO サーバーは **HTTPS** 経由で利用でき、**zync-que** によりアクセスできる。これをテストするには、以下のように **zync-que** Pod 内から **curl https://rhsso-fqdn** を実行します。

```
oc rsh -n $THREESCALE_PROJECT $(oc get pods -n $THREESCALE_PROJECT --field-selector=status.phase==Running -o name | grep zync-que) /bin/bash -c "curl -v https://<rhsso-fqdn>/auth/realms/master"
```

- OpenShift クラスター管理者のパーミッション。
- OpenID Connect インテグレーションを RH-SSO と設定する 3scale API プロダクト。

詳細は、以下のセクションを参照してください。

- カスタム認証局証明書を使用する 3scale Zync の設定
- 3scale クライアントを使用する RH-SSO の設定
- RH-SSO と連携する 3scale の設定

12.4.1. カスタム認証局証明書を使用する 3scale Zync の設定

信頼できる認証局 (CA) が発行する証明書を RH-SSO が使用する場合には、この作業は必要ありません。ただし、信頼できる CA が発行する証明書を RH-SSO が **使用しない** 場合は、3scale Zync を設定して 3scale クライアントを使用するように RH-SSO を、RH-SSO と連携するように 3scale を設定する必要があります。

手順

1. **.pem** 形式で CA 証明書チェーンを取得し、各証明書を個別のファイル (例: **customCA1.pem**、**customCA2.pem** など) として保存します。
2. 各証明書ファイルをテストし、これが有効な CA であることを確認します。以下に例を示します。

```
openssl x509 -in customCA1.pem -noout -text | grep "CA:"
```

これは **CA:TRUE** または **CA:FALSE** のいずれかを出力します。各証明書ファイルで、出力を **CA:TRUE** にします。出力が **CA:FALSE** の場合には、証明書は有効な CA ではありません。

3. 以下の **cURL** コマンドを使用して、各証明書ファイルを検証します。以下に例を示します。

```
curl -v https://<secure-sso-host>/auth/realms/master --cacert customCA1.pem
```

<secure-sso-host> は、RH-SSO ホストの完全修飾ドメイン名に置き換えます。

RH-SSO レルムの JSON 設定が返されるはずです。検証に失敗した場合は、証明書が正しくない可能性があります。

4. **zync-que** 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。

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

5. 以下のように、各カスタム CA 証明書ファイルの内容を **zync.pem** に追加します。

```
cat customCA1.pem customCA2.pem ... >> zync.pem
```

6. 新規ファイルを configmap オブジェクトとして **zync-que** Pod に割り当てます。

```
oc create configmap zync-ca-bundle --from-file=./zync.pem
oc set volume dc/zync-que --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"}]}}'
```

これで、証明書バンドルの **zync-que** Pod への追加が完了しました。

7. 証明書が追加され、コンテンツが正しいことを確認します。

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

8. 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-que SSL_CERT_FILE=/etc/pki/tls/zync/zync.pem
```

12.4.2. 3scale クライアントを使用する RH-SSO の設定

OpenShift RH-SSO ダッシュボードで、RH-SSO を 3scale クライアントを使用するように設定します。これは特殊な管理クライアントです。API コンシューマーが開発者ポータルの API をサブスクライブするたびに、3scale はこの手順で作成した RH-SSO 管理クライアントを使用して API コンシューマーアプリケーションのクライアントを作成します。

手順

1. RH-SSO コンソールで、3scale クライアントの レルムを作成 するか、3scale クライアントが含まれる既存のレルムを選択します。
2. 左側のナビゲーションパネルで、新しいレルムまたは選択したレルムの **Clients** をクリックします。
3. **Create** をクリックして新規クライアントを作成します。
4. **Client ID** フィールドに、このクライアントを 3scale クライアントとして識別するのに役立つ名前 (**oidc-issuer-for-3scale** など) を指定します。
5. **Client Protocol** フィールドを **openid-connect** に設定します。
6. 新しいクライアントを保存します。
7. 新しいクライアントの設定で、以下を設定します。

- Access Type を **Confidential** に設定します。
- Standard Flow Enabled を **OFF** に設定します。
- Direct Access Grants Enabled を **OFF** に設定します。
- Service Accounts Enabled を **ON** に設定します。この設定により、このクライアントはサービスアカウントを発行できます。

a. **Save** をクリックします。

8. 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** を選択し、**Add selected >>** をクリックしてロールを割り当てます。

9. 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。

10. 承認フローのテストを容易にするには、レルムにユーザーを追加します。

a. ウィンドウの左側にある **Users** を展開します。

b. **Add user** をクリックします。

c. ユーザー名を入力し、**Email Verified** 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 **Set 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、**Set password** をクリックします。

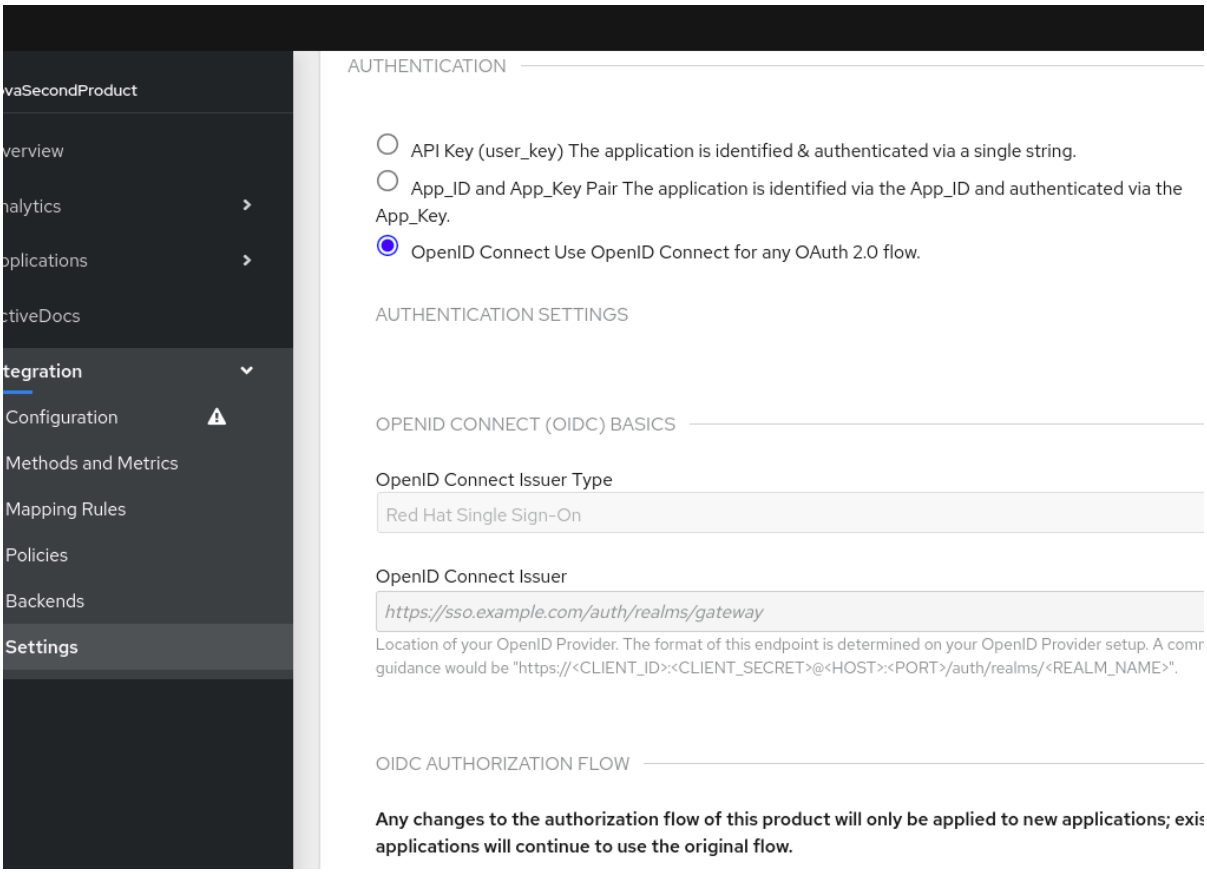
12.4.3. RH-SSO と連携する 3scale の設定

3scale 管理ポータルでインテグレーション設定を指定して、RH-SSO と連携するように 3scale を設定します。

手順

1. 3scale 管理ポータルでの最上位のセレクターで **Products** をクリックし、OpenID Connect 認証を有効にする 3scale API プロダクトを選択します。
2. **[Your_product_name] > Integration > Settings** の順に移動します。
3. **Authentication** で、**OpenID Connect Use OpenID Connect for any OAuth 2.0 flow** を選択します。

図12.3 OPENID CONNECT(OIDC)BASICS セクションを表示します。



4. **OpenID Connect Issuer Type** フィールドで、設定が **Red Hat Single Sign-On** であることを確認します。

5. **OpenID Connect Issuer** フィールドに、設定された OpenID Connect アイデンティティープロバイダーの URL を入力します。この URL の形式は以下のようになります。

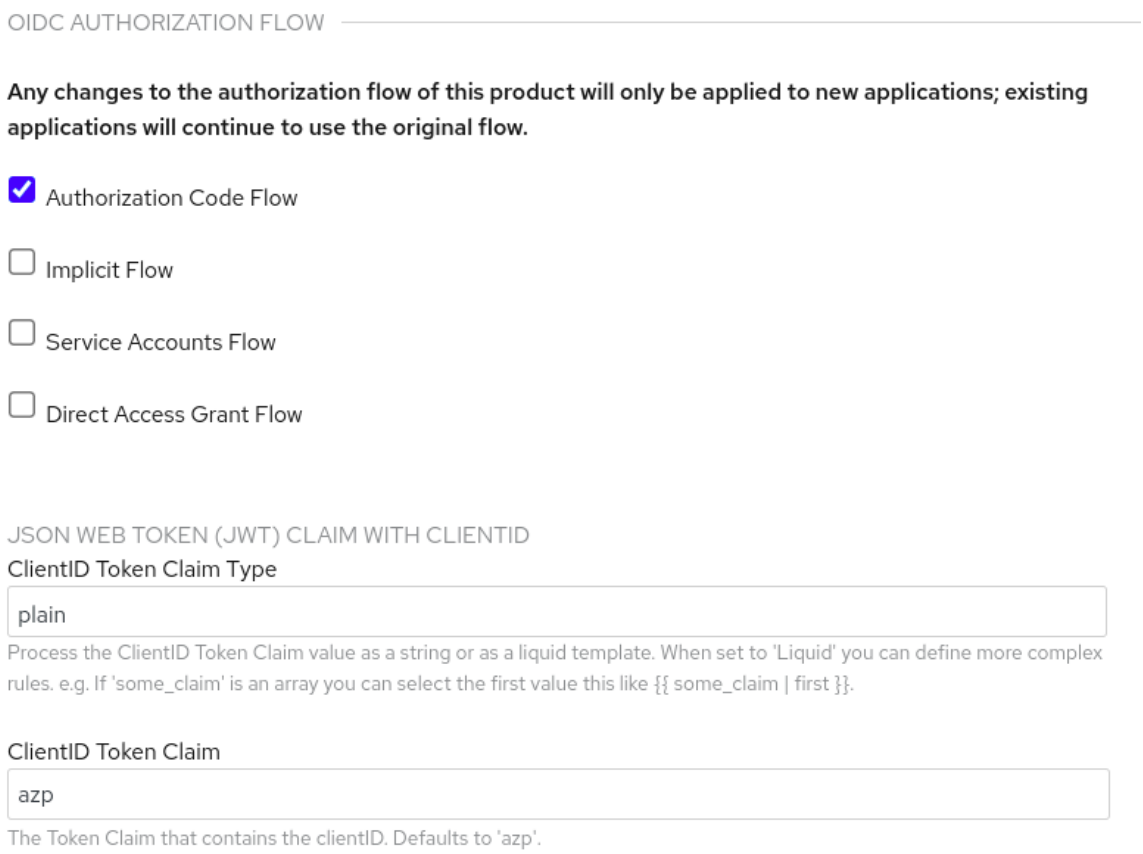
```
https://<client_id>:<client_secret>@<rhsso_host>:<rhsso_port>/auth/realms/<realm_name>
```

6. プレースホルダーは、以前に書き留めた RH-SSO クライアントの認証情報、RH-SSO サーバーのホストおよびポート、および RH-SSO クライアントが含まれるレルムの名前に置き換えます。

7. **OIDC AUTHORIZATION FLOW** で、以下のいずれかを選択します。

- **認可コードフロー**: RH-SSO では標準フローです。
- **インプリシットフロー**
- **サービスアカウントフロー**: クライアントクレデンシャルフローとも呼ばれます。
- **直接アクセスグラントフロー**: リソースオーナーパスワードクレデンシャルとも呼ばれます。

図12.4 承認フローを選択する場所を示します。



これにより、API コンシューマーが OpenID Connect アイデンティティープロバイダーから JSON Web Tokens (JWT) を取得する方法が設定されます。3scale が OpenID Connect アイデンティティープロバイダーとして RH-SSO を統合する際に、Zync は **認可コードフロー** のみが有効になっている RH-SSO クライアントを作成します。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。OpenID Connect アイデンティティープロバイダーがサポートする OAuth 2.0 フロー を必ず選択してください。

8. 下方向にスクロールして **Update Product** をクリックし、設定を保存します。
9. 左側のナビゲーションパネルで **Integration > Configuration** をクリックします。
10. 下にスクロールして **Promote v. x to APIcast Staging** をクリックします。

次のステップ

RH-SSO とのインテグレーションをアイデンティティープロバイダーとしてテストします。すべてがそのまま機能したら、**Integration > Configuration** ページに戻り、下方向にスクロールして APIcast ステージングバージョンを実稼働バージョンにプロモートします。

12.5. 3SCALE とサードパーティー OPENID CONNECT アイデンティティープロバイダーの統合

API プロバイダーとして、3scale とサードパーティーの OpenID Connect アイデンティティープロバイダーとの間で HTTP インテグレーションを設定できます。つまり、Red Hat Single Sign-On 以外の OpenID Connect アイデンティティープロバイダーを設定できます。3scale はこのインテグレーションを使用して API コンシューマーからの要求を認証し、サードパーティーのアイデンティティープロバイダーを最新の 3scale アプリケーション詳細で更新します。

3scale とサードパーティーの OpenID Connect アイデンティティープロバイダーを統合するのに必要な作業にはほぼ、以下のタスクが含まれます。

- 3scale Zync 関連の前提条件を満たす。
- 3scale アプリケーションから要求を承認するように OpenID Connect アイデンティティープロバイダーを設定する。

前提条件

- 3scale Zync がインストールされている。
- 選択したサードパーティー OpenID Connect アイデンティティープロバイダー:
  - 3scale により提供される Zync の OpenAPI 仕様に準拠している 。
  - 要求でパラメーターとして宣言された **<client_id>** と **<client_secret>** でクライアントの登録を許可する。3scale は、常に 3scale とサードパーティーの OpenID Connect アイデンティティープロバイダーとのインテグレーションにおいて、クライアント ID 管理のソースである。
  - 3scale アプリケーションから要求を承認するように設定されている。
- 上記の前提条件に対応できない場合は、Zync 抽象アダプターに基づくカスタムアダプターを実装する必要がある。Zync はこのアダプターを使用して OpenID Connect アイデンティティープロバイダーと対話します。このアダプターを作成するには、3scale Zync REST API の例 に含まれる rest_adapter.rb を変更してください。
  rest_adapter.rb モジュールは、要件に最も適した方法に従って **zync-que** Pod に含めることができます。たとえば、ボリュームを使用して **configMap** をマウントするか、Zync の新規イメージをビルドできます。

手順

1. 3scale 管理ポータルでの最上位のセレクターで **Products** をクリックし、OpenID Connect 認証を有効にする 3scale API プロダクトを選択します。
2. **[Your_product_name] > Integration > Settings**の順に移動します。
3. **Authentication** で、 **OpenID Connect Use OpenID Connect for any OAuth 2.0 flow**を選択します。
   これにより、**OPENID CONNECT(OIDC)BASICS** セクションが表示されます。
4. **OpenID Connect Issuer Type** フィールドで、設定が **REST API** であることを確認します。
5. **OpenID Connect Issuer** フィールドに OpenID Connect アイデンティティープロバイダーの URL を入力します。この URL の形式は以下のようになります。

   ```
   https://<client_id>:<client_secret>@<oidc_host>:<oidc_port>/<endpoint>
   ```

   たとえば、Zync **rest_adapter.rb** の例 では、URL エンドポイントは **{endpoint}/clients** としてハードコーディングされます。エンドポイントは **{endpoint}/register** などです。
6. **OIDC AUTHORIZATION FLOW** で、以下のいずれかを選択します。
   - **認可コードフロー**
   - **インプリシットフロー**

- **サービスアカウントフロー**
- **直接アクセスグラントフロー**

これにより、API コンシューマーアプリケーションが OpenID Connect アイデンティティープロバイダーから JSON Web Tokens (JWT) を受け取る方法が設定されます。**認可コードフロー**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。OpenID Connect アイデンティティープロバイダーがサポートする OAuth 2.0 フロー を必ず選択してください。

7. 下方向にスクロールして **Update Product** をクリックし、設定を保存します。
8. 左側のナビゲーションパネルで **Integration > Configuration** をクリックします。
9. 下にスクロールして **Promote v. x to APIcast Staging** をクリックします。

次のステップ

サードパーティーアイデンティティープロバイダーとのインテグレーションをテストします。すべてがそのまま機能したら、**Integration > Configuration** ページに戻り、下方向にスクロールして APIcast ステージングバージョンを実稼働バージョンにプロモートします。

12.6. OPENID CONNECT アイデンティティープロバイダーとの 3SCALE インテグレーションのテスト

3scale と OpenID Connect アイデンティティープロバイダーを統合した後に、インテグレーションをテストし、以下を確認します。

- API コンシューマーは、3scale 管理の API にサブスクライブすると、アクセスクレデンシャルを受け取ります。
- APIcast は API コンシューマーからのリクエストを認証できます。

前提条件

- 特定の 3scale API プロダクトに対して 3scale と OpenID Connect アイデンティティープロバイダー間のインテグレーションが用意されている。このインテグレーションは **認可コードフロー**を使用します。
- API コンシューマーが開発ポータルでサブスクライブできるアプリケーションプランがある。このアプリケーションプランを使用して、3scale 管理の API (OpenID Connect 認証を設定した 3scale プロダクト) にアクセスできる。
- アップストリーム API に要求を送信するアプリケーション。アップストリーム API は、サブスクリプションを行うことで API コンシューマーアプリケーションがアクセスできる 3scale プロダクトのバックエンドです。または、Postman を使用して要求を送信することもできます。

手順

1. 開発者ポータルで、アプリケーションプランにサブスクライブします。
これにより、デベロッパーポータルでアプリケーションが作成されます。OpenID Connect アイデンティティープロバイダーは、デベロッパーポータルのアプリケーションページに表示されるクライアント ID およびクライアントシークレットを返す必要があります。
2. 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. OpenID Connect アイデンティティープロバイダーが同じクライアント ID とクライアントシークレットを持つクライアントであることを確認します。たとえば、Red Hat Single Sign-On (RH-SSO) が OpenID Connect アイデンティティープロバイダーである場合に、設定された RH-SSO レルムに新しいクライアントが表示されます。

4. 管理ポータルのアプリケーションページの **REDIRECT URL** フィールドに、API 要求をアップストリーム API に送信するアプリケーションの URL を入力します。

5. OpenID Connect アイデンティティープロバイダーに正しいリダイレクト URL があることを確認します。

6. このエンドポイントを使用して、OpenID Connect アイデンティティープロバイダーの認証要求を受信する URL を見つけます。

   ```
   .well-known/openid-configuration
   ```

   以下に例を示します。

   ```
   https://<rhsso_host>:<rhsso_port>/auth/realms/<realm_name>/.well-known/openid-configuration
   ```

7. ベース URL には、API プロバイダーが OpenID Connect Issuer フィールドで設定した値を使用します。

8. アップストリーム API の使用に関するアプリケーションを記述する API コンシューマーは、以下を行います。

   a. OpenConnect アイデンティティープロバイダーで承認フローを開始します。この要求には、3scale 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が含まれている必要があります。エンドユーザーのアイデンティティーも必要になる場合があります。

   b. 認可コードが含まれるアイデンティティープロバイダーの応答を受け取ります。

9. API コンシューマーのアプリケーションは以下を行います。

   a. JWT の認可コードを交換します。

   b. 認証時に RH-SSO から JWT を受け取ります。

   c. JWT が含まれる API 要求をアップストリーム API バックエンドに送信します。

APIcast が要求で JWT を受け入れると、アプリケーションは API バックエンドから応答を受け取ります。

また、API コンシューマーアプリケーションの代わりに、 Postman を使用して、トークンフローが適切に実装されていることをテスト します。

12.7. OPENID CONNECT アイデンティティープロバイダーとの 3SCALE インテグレーションの例

以下の例は、3scale を OpenID Connect アイデンティティープロバイダーとして Red Hat Single Sign-On (RH-SSO) と統合する場合のフローを示しています。この例には、以下の特徴があります。

- 管理ポータルで、API プロバイダーが 3scale API プロダクトを定義し、そのプロダクトが OpenID Connect ID プロバイダーとして RH-SSO を使用するように設定されています。

- このプロダクトの OpenID Connect 設定には以下が含まれます。
  - 公開ベース URL: **https://api.example.com**
  - プライベートベース URL: **https://internal-api.example.com**
  - OpenID Connect 発行者: **https://zync:41dbb98b-e4e9-4a89-84a3-91d1d19c4207@idp.example.com/auth/realms/myrealm**
  - 標準フローである **認可コードフロー** を選択します。
- 3scale 開発者ポータルには、以下の特徴を持つアプリケーションがあります。このアプリケーションは、開発者ポータルの特定のアプリケーションプランで指定されている 3scale API プロダクトにアクセスできるように、API コンシューマーがサブスクライブした結果、作成されたものです。
  - **クライアント ID**: **myclientid**
  - **Client Secret**: **myclientsecret**
  - **URL のリダイレクト**: **https://myapp.example.com**
- RH-SSO のレルム **myrealm** に、以下の特性を持つクライアントがあります。
  - **クライアント ID**: **myclientid**
  - **Client Secret**: **myclientsecret**
  - **URL のリダイレクト**: **https://myapp.example.com**
- **myrealm** レルムには、以下のユーザーが含まれます。
  - ユーザー名: **myuser**
  - パスワード: **mypassword**
- **myrealm** の 3scale Zync クライアントが正しい **サービスアカウント** ロールを持ちます。

フローは以下のようになります。

1. エンドユーザー (API コンシューマー) は、以下の例のエンドポイントで認証サーバーの RH-SSO に承認要求を送信します。

   ```
   https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/auth
   ```

   この要求では、アプリケーションは以下のパラメーターを提供します。

   - クライアント ID: **myclientid**
   - リダイレクト URL: **https://myapp.example.com**
2. アプリケーションはエンドユーザーを RH-SSO ログインウィンドウにリダイレクトします。
3. エンドユーザーは、以下の認証情報を使用して RH-SSO にログインします。
   - ユーザー名: **myuser**
   - パスワード: **mypassword**

4. 設定およびエンド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。

5. RH-SSO は、エンドユーザーに認可コードを発行します。

6. API コンシューマー アプリケーションは、次のエンドポイントを使用して、JWT の認可コードを交換する要求を送信します。

   ```
   https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/token
   ```

   要求には、認可コードとこれらのパラメーターが含まれます。

   - クライアント ID: **myclientid**
   - クライアントシークレット: **myclientsecret**
   - リダイレクト URL: **https://myapp.example.com**

7. RH-SSO は、**eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A** などの access_token フィールドを使用して JSON Web Token (JWT) を返します。

8. API コンシューマーアプリケーションは、以下のようなヘッダーを使用して API 要求を **https://api.example.com** に送信します。

   ```
   Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A.
   ```

9. 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。