Red Hat 3scale API Management 2.6

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: 2023-02-03

Red Hat 3scale API Management 2.6 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

Copyright © 2023 | You need to change the HOLDER entity in the en-US/Administering_the_API_Gateway.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

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

前書き 5

第1章 APICAST の運用 6
1.1. マッピングルール 6
1.1.1. マッピングルールの照合 6
1.1.2. マッピングルールのワークフロー 7
1.2. HOST ヘッダー 8
1.3. 実稼働環境用のデプロイメント 8
1.4. 公開ベース URL 8
1.5. API バックエンドの保護 9
1.6. プライベート API を扱う APICAST の使用 9
1.7. APICAST への OPENTRACING の設定 9
1.7.1. 前提条件 9
1.7.2. 手順 10
1.7.3. 補足情報 10
1.7.4. OpenShift インスタンスへの Jaeger のインストール 10

第2章 DOCKER コンテナー環境の操作 13
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 13
2.1.1. Cannot connect to the Docker daemon エラー 13
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 13

第3章 高度な APICAST 設定 14
3.1. シークレットトークンの定義 14
3.2. クレデンシャル 14
3.3. エラーメッセージの設定 15
3.4. 設定履歴 16
3.5. デバッグ 16
3.6. パスルーティング 17

第4章 APICAST ポリシー 18
4.1. APICAST 標準ポリシー 18
4.1.1. 3scale Auth Caching ポリシー 19
4.1.2. 3scale Batcher ポリシー 20
4.1.3. Anonymous Access ポリシー 21
4.1.4. CORS Request Handling ポリシー 22
4.1.5. Echo ポリシー 24
4.1.6. Edge Limiting ポリシー 24
4.1.6.1. 制限のタイプ 25
4.1.6.2. 制限の定義 25
4.1.6.3. Liquid テンプレート 26
4.1.6.4. 条件の適用 26
4.1.6.5. ストアの設定 27
4.1.6.6. エラー処理 27
4.1.7. Header Modification ポリシー 28
4.1.8. IP Check ポリシー 29
4.1.9. JWT Claim Check ポリシー 30
4.1.9.1. JWT Claim Check ポリシーの概要 30
4.1.9.2. ポリシーチェーンへの JWT Claim Check ポリシーの設定 31
4.1.9.2.1. 前提条件 31
4.1.9.2.2. ポリシーの設定 31
4.1.10. Liquid Context Debug ポリシー 32

4.1.11. Logging ポリシー 32
4.1.12. OAuth 2.0 Token Introspection ポリシー 33
4.1.13. Prometheus メトリクス 36
4.1.14. Referrer ポリシー 38
4.1.15. Retry ポリシー 38
4.1.16. RH-SSO/Keycloak Role Check ポリシー 39
4.1.17. Routing ポリシー 41
4.1.17.1. ルーティングルール 41
4.1.17.2. リクエストパスルール 42
4.1.17.3. ヘッダールール 42
4.1.17.4. クエリー引数ルール 42
4.1.17.5. JWT クレームルール 43
4.1.17.6. 複数演算ルール 44
4.1.17.7. ルールの結合 45
4.1.17.8. キャッチオールルール 45
4.1.17.9. サポートされる操作 46
4.1.17.10. Liquid テンプレート 47
4.1.17.11. Host ヘッダーで使用されるホストの設定 47
4.1.18. SOAP ポリシー 48
4.1.19. TLS Client Certificate Validation ポリシー 49
4.1.19.1. TLS Client Certificate Validation ポリシーの概要 49
4.1.19.2. TLS Client Certificate Validation ポリシーに対応する APIcast の設定 49
4.1.19.2.1. 前提条件: 49
4.1.19.2.2. ポリシーに対応する APIcast のセットアップ 49
4.1.19.3. ポリシーチェーンへの TLS Client Certificate Validation ポリシーの設定 50
4.1.19.3.1. 前提条件 51
4.1.19.3.2. ポリシーの設定 51
4.1.19.4. TLS Client Certificate Validation ポリシー機能の確認 51
4.1.19.4.1. 前提条件 51
4.1.19.4.2. ポリシー機能の確認 51
4.1.19.5. ホワイトリストからの証明書の削除 52
4.1.19.5.1. 前提条件 52
4.1.19.5.2. 証明書の削除 52
4.1.19.6. リファレンス資料 52
4.1.20. Upstream ポリシー 52
4.1.21. Upstream Connection ポリシー 53
4.1.21.1. Upstream Connection ポリシーの概要 53
4.1.21.2. ポリシーチェーンへの Upstream Connection ポリシーの設定 53
4.1.21.2.1. 前提条件 53
4.1.21.2.2. ポリシーの設定 53
4.1.22. URL Rewriting ポリシー 54
4.1.22.1. パスを書き換えるためのコマンド 54
4.1.22.2. クエリー文字列を書き換えるためのコマンド 55
4.1.23. URL Rewriting with Captures ポリシー 56
4.2. 管理ポータルでのポリシーの有効化 57
4.3. カスタム APICAST ポリシーの作成 57
4.4. APICAST へのカスタムポリシーの追加 58
4.4.1. Embedded APIcast へのカスタムポリシーの追加 58
4.4.2. 別の OpenShift Container Platform 上の APIcast へのカスタムポリシーの追加 60
4.5. 3SCALE でのポリシーチェーンの作成 60
4.6. ポリシーチェーン JSON 設定ファイルの作成 61

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 63

5.1. ポリシーでの変数およびフィルターの使用 63

第6章 APICAST の環境変数 66
APICAST_BACKEND_CACHE_HANDLER 67
APICAST_CONFIGURATION_CACHE 67
APICAST_CONFIGURATION_LOADER 68
APICAST_CUSTOM_CONFIG 68
APICAST_ENVIRONMENT 68
APICAST_EXTENDED_METRICS 68
APICAST_LOG_FILE 68
APICAST_LOG_LEVEL 68
APICAST_ACCESS_LOG_FILE 69
APICAST_OIDC_LOG_LEVEL 69
APICAST_MANAGEMENT_API 69
APICAST_MODULE 69
APICAST_PATH_ROUTING 69
APICAST_PATH_ROUTING_ONLY 69
APICAST_POLICY_LOAD_PATH 70
APICAST_PROXY_HTTPS_CERTIFICATE_KEY 70
APICAST_PROXY_HTTPS_CERTIFICATE 70
APICAST_PROXY_HTTPS_PASSWORD_FILE 70
APICAST_PROXY_HTTPS_SESSION_REUSE 70
APICAST_HTTPS_VERIFY_DEPTH 71
APICAST_REPORTING_THREADS 71
APICAST_RESPONSE_CODES 71
APICAST_SERVICES_FILTER_BY_URL 71
APICAST_SERVICES_LIST 72
APICAST_UPSTREAM_RETRY_CASES 72
APICAST_SERVICE_${ID}_CONFIGURATION_VERSION 72
APICAST_WORKERS 72
BACKEND_ENDPOINT_OVERRIDE 72
OPENSSL_VERIFY 72
RESOLVER 73
THREESCALE_CONFIG_FILE 73
THREESCALE_DEPLOYMENT_ENV 73
THREESCALE_PORTAL_ENDPOINT 73
OPENTRACING_TRACER 74
OPENTRACING_CONFIG 74
OPENTRACING_HEADER_FORWARD 74
APICAST_HTTPS_PORT 74
APICAST_HTTPS_CERTIFICATE 74
APICAST_HTTPS_CERTIFICATE_KEY 74
all_proxy、ALL_PROXY 74
http_proxy、HTTP_PROXY 75
https_proxy、HTTPS_PROXY 75
no_proxy、NO_PROXY 75

第7章 パフォーマンスを向上させるための APICAST 設定 76
7.1. 全般的なガイドライン 76
7.2. デフォルトのキャッシング 76
7.3. 非同期レポートスレッド 78
7.4. 3SCALE BATCHER ポリシー 79

前書き

本ガイドは、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。

第1章 APICAST の運用

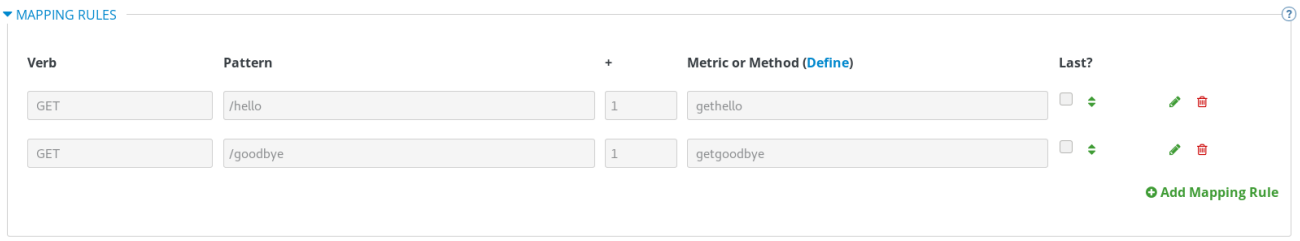
本セクションでは、高度な APIcast 設定を使用する際に考慮すべき概念について説明します。

1.1. マッピングルール

マッピングルールにより、API に対するリクエストに応じて報告するメトリクスまたはメソッドを定義します。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。ただし、このルールは一般的すぎるので、通常は変更されます。

より具体的な Echo API ルールの例を以下に示します。



1.1.1. マッピングルールの照合

マッピングルールは前方一致で照合され、複雑な設定が可能です (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。
- 文字列 (例: **/hello**) を使用してパスの照合を行うことができます。
- マッピングルールでは、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 (例: **/{word}?value={value}**)。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名前付きワイルドカードを含めることができます (例: **/{word}**)。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、指定したソート法に従ってすべてのマッピングルールが順番に評価されます。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。

1.1.2. 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。

- 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追加 を参照)。
- 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。
- **Update & Test Staging Configuration**をクリックすると、すべての変更および削除が保存されます。

マッピングルールの追加

新たなマッピングルールを追加するには、以下の手順を実施します。

1. **Add Mapping Rule** をクリックします。
2. 以下の設定を指定します。
   - **Verb**: HTTP リクエストの動詞 (**GET**、**POST**、**DELETE**、または **PUT**)。
   - **Pattern**: 照合するパターン (例: /**hello**)。
   - **+**: メトリクスのカウントを増やす数 (例: **1**)。
   - **Metric (or method)**: メトリクスまたはメソッドの名前 (例: **gethello**)。
3. **Update & Test Staging Configuration**をクリックして変更を適用します。

他のマッピングルールの停止

他のマッピングルールの処理を停止するには、**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場合には、

```
(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、APIcast は残りのマッピングルールを処理してルールがマッチした際にそのメトリクスのカウントを増やすのを停止します。

マッピングルールの並べ替え

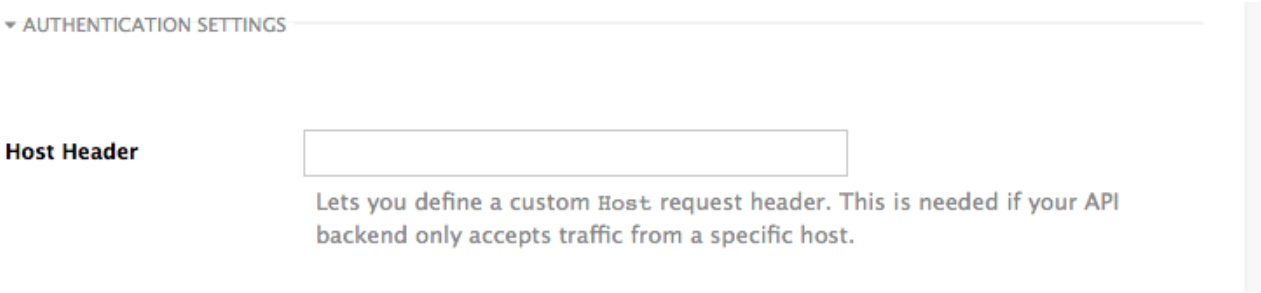
マッピングルールを並べ替えるには、各マッピングルールの **Last?** 設定の横にある緑色の矢印を使用して、ルールをドラッグアンドドロップします。**Update & test in Staging Environment**をクリックすると、指定した並べ替えがデータベースに保存されプロキシー設定に維持されます。

その他の設定オプションは、高度な APIcast 設定 を参照してください。

1.2. 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 バックエンドにのみ必要です。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 バックエンドの前にあると問題が生じます (例: **xxx-yyy.staging.apicast.io**)。

この問題を避けるには、AUTHENTICATION SETTINGS の **Host Header** フィールドで API バックエンドが想定するホストを定義します。これにより、Hosted APIcast インスタンスはホストを書き換えます。



1.3. 実稼働環境用のデプロイメント

API インテグレーションを設定しステージング環境で動作することを確認したら、実稼働環境用 APIcast デプロイメントのいずれかに進むことができます。

Integration ページの最後に **Production** セクションがあります。ここに、2 つのフィールド **Private Base URL** (**Staging** セクションで設定したものと同じ) および **Public Base URL** があります。

1.4. 公開ベース URL

公開ベース URL は、3scale により保護されたご自分の API に開発者がリクエストを行うのに使用する URL です。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 Public Base URL で指定したホスト名に対する呼び出ししか受け付けない点に注意してください。たとえば、上記で使用した Echo API の例で、パブリックベース URL として **https://echo-api.3scale.net:443** を指定すると、正しい呼び出しは次のようになります。

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

まだ API 用の公開ドメインがない場合には、リクエストに APIcast IP を使用することもできますが、それでも Public Base URL フィールドには値を指定する必要があります (実際のドメインではなくても)。その場合には Host ヘッダーでホストを指定する必要があります。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、ドメインに localhost を使用することもできます。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 サービスが複数ある場合には、それぞれのサービスについてこの公開ベース URL を適切に設定する必要があります。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。

1.5. 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ら、クレデンシャルが提供されない API バックエンドへの直接アクセスを制限することが望まれる場合があります。そのための最も簡単な方法は、APIcast が設定するシークレットトークンを使用することです。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。

1.6. 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 バックエンド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、ステージング環境のメリットが得られない点およびテストコールが成功しない点に注意して下さい。ステージング用 APIcast インスタンスは 3scale がホストし、プライベート API のバックエンドにはアクセスすることができないためです。しかし、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。

1.7. APICAST への OPENTRACING 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。

1.7.1. 前提条件

APIcast デプロイメントに分散トレーシング機能を追加するには、以下の前提条件が満たされている必要があります。

- 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 (通常は HTTP ヘッダー経由)。
- 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。

1.7.2. 手順

OpenTracing を設定するには、以下の環境変数を使用します。

- OPENTRACING_TRACER: 使用するトレーサーの実装を定義します。現在、利用することができるのは Jaeger だけです。
- OPENTRACING_CONFIG: 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。ここに例を示します。
- OPENTRACING_HEADER_FORWARD: オプション。実際の OpenTracing 設定に応じて、この環境変数を設定することができます。

これらの変数に関する詳細は、APIcast の環境変数 を参照してください。

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する必要があります。

1.7.3. 補足情報

OpenTracing および Jaeger インテグレーションは、アップストリームプロジェクト (https://github.com/3scale/apicast) で公開されています。

1.7.4. OpenShift インスタンスへの Jaeger のインストール

本セクションでは、実行中の OpenShift インスタンスへの Jaeger のインストールについて説明します。

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。

1. 現在の namespace に Jaeger オールインワン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
   }
}
```

- すべてのリクエストをサンプリングするために、**sampler** 定数を 1 に設定します。
- **reporter** の場所およびキューサイズを設定します。
- リクエストを追跡するのに使用する **TraceContextHeaderName** を含め、**headers** を設定します。

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 volume 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. 最後のステップは、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 に追加することです。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。

Jaeger の設定に関する詳細は、以下のドキュメントを参照してください。

- Jaeger OpenShift Templates の 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。

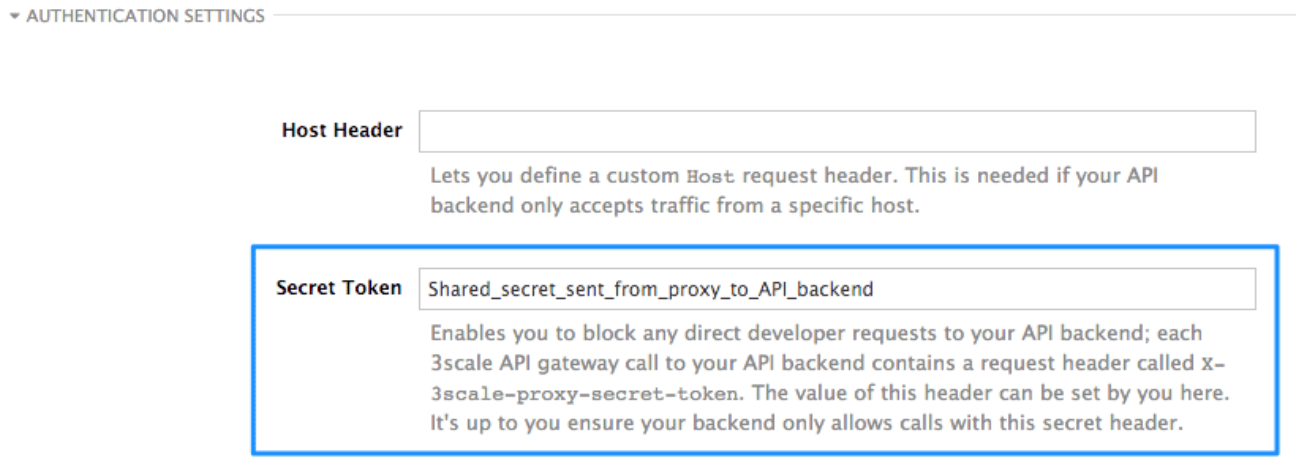
使用できるコマンドの詳細は、Docker コマンドのリファレンス を参照してください。

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。

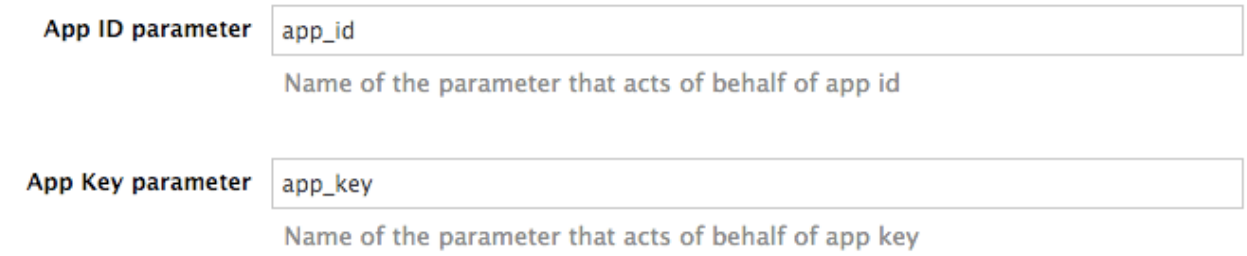
ゲートウェイが機能するためには、API バックエンドには公開されている解決可能なドメインが必要です。したがって、API バックエンドを知っていれば、誰でもクレデンシャルの確認を迂回することができます。ステージング環境の API ゲートウェイは実稼働環境での使用を想定していないので、このことが問題になることはありません。ただし、アクセスを防げるようにしておいた方が望ましいと言えます。

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。

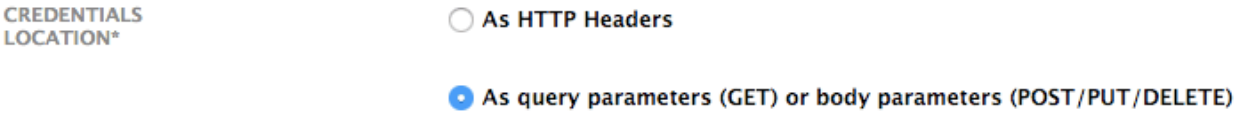
なお、API で異なるクレデンシャル名を使用することが望ましい場合もあります。この場合、API キーモードを使用していれば **user_key** にカスタムな名前を設定する必要があります。

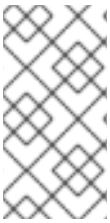
あるいは、以下のように **app_id** および **app_key** を設定します。



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_API_name] > Integration > Configuration > edit APIcast configuration > GATEWAY RESPONSE** の順に移動します。

2. 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. **Update & test in Staging Environment**をクリックして変更を保存します。

3.4. 設定履歴

Update & test in Staging Environmentボタンをクリックするたびに、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。

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、すべての設定バージョンの履歴が、対応する JSON ファイルにより提供されます。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、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て、ヘッダー **X-3scale-debug: {SERVICE_TOKEN}** を API リクエストに追加する必要があります。

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

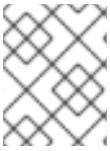
APIcast ポリシーとは、APIcast の動作を変更する機能の単位です。ポリシーを有効、無効、および設定して、APIcast 動作の変更を制御することができます。ポリシーを使用して、デフォルトの APIcast デプロイメントでは利用することのできない機能を追加します。自分専用のポリシーを作成することや、Red Hat 3scale の提供する 標準ポリシー を使用することができます。

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

- APIcast 標準ポリシー
- カスタム APIcast ポリシーの作成
- 3scale でのポリシーチェーンの作成

ポリシーチェーンを使用して、サービスに対するポリシーを制御します。ポリシーチェーンの機能を以下に示します。

- APIcast が使用するポリシーを指定する
- 3scale が使用するポリシーの設定情報を提供する
- 3scale がポリシーを読み込む順番を指定する



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。

カスタムポリシーを使用して APIIcast の動作を変更するには、以下の操作が必要です。

- カスタムポリシーを APIcast に追加する
- APIcast ポリシーを設定するポリシーチェーンを定義する
- ポリシーチェーンを APIcast に追加する

4.1. APICAST 標準ポリシー

3scale では、以下の標準ポリシーが利用可能です。

- 「3scale Auth Caching ポリシー」
- 「3scale Batcher ポリシー」
- 「Anonymous Access ポリシー」
- 「CORS Request Handling ポリシー」
- 「Echo ポリシー」
- 「Edge Limiting ポリシー」
- 「Header Modification ポリシー」

- 「IP Check ポリシー」
- 「JWT Claim Check ポリシー」
- 「Liquid Context Debug ポリシー」
- 「Logging ポリシー」
- 「OAuth 2.0 Token Introspection ポリシー」
- 「Prometheus メトリクス」
- 「Referrer ポリシー」
- 「Retry ポリシー」
- 「RH-SSO/Keycloak Role Check ポリシー」
- 「Routing ポリシー」
- 「SOAP ポリシー」
- 「TLS Client Certificate Validation ポリシー」
- 「Upstream ポリシー」
- 「Upstream Connection ポリシー」
- 「URL Rewriting ポリシー」
- 「URL Rewriting with Captures ポリシー」

3scale API Management で標準ポリシーを 有効化および設定 することができます。

4.1.1. 3scale Auth Caching ポリシー

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。

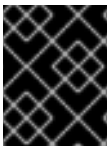
strict モードでは、承認された呼び出しだけがキャッシュされます。ポリシーを strict モードで実行中に呼び出しが失敗または拒否されると、ポリシーはキャッシュエントリーを無効にします。バックエンドにアクセスできなくなると、キャッシュステータスにかかわらず、キャッシュされたすべての呼び出しが拒否されます。

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、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。

4.1.2. 3scale Batcher ポリシー

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。

3scale Batcher ポリシーを使用すると、3scale バックエンドへのリクエスト数が大幅に削減され、レイテンシーが減少しスループットが向上します。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、使用状況レポートをバッチ処理します。

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)。

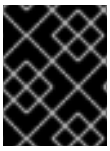
3scale Batcher ポリシーではスループットが向上しますが、精度は低下します。使用上限および現在の使用状況は 3scale に保存され、APIcast は 3scale バックエンドに呼び出しを行う時にのみ正しい承認ステータスを取得することができます。3scale Batcher ポリシーが有効な場合には、APIcast が 3scale に呼び出しを送信しない期間があります。この期間中、呼び出しを行うアプリケーションが定義された限度を超える可能性があります。

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。

- **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。



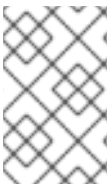
重要

このポリシーを使用するには、ポリシーチェーンで **3scale APIcast** および **3scale Batcher** ポリシーの両方を有効にします。

4.1.3. Anonymous Access ポリシー

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 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を使用するサービスしかサポートしません。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。



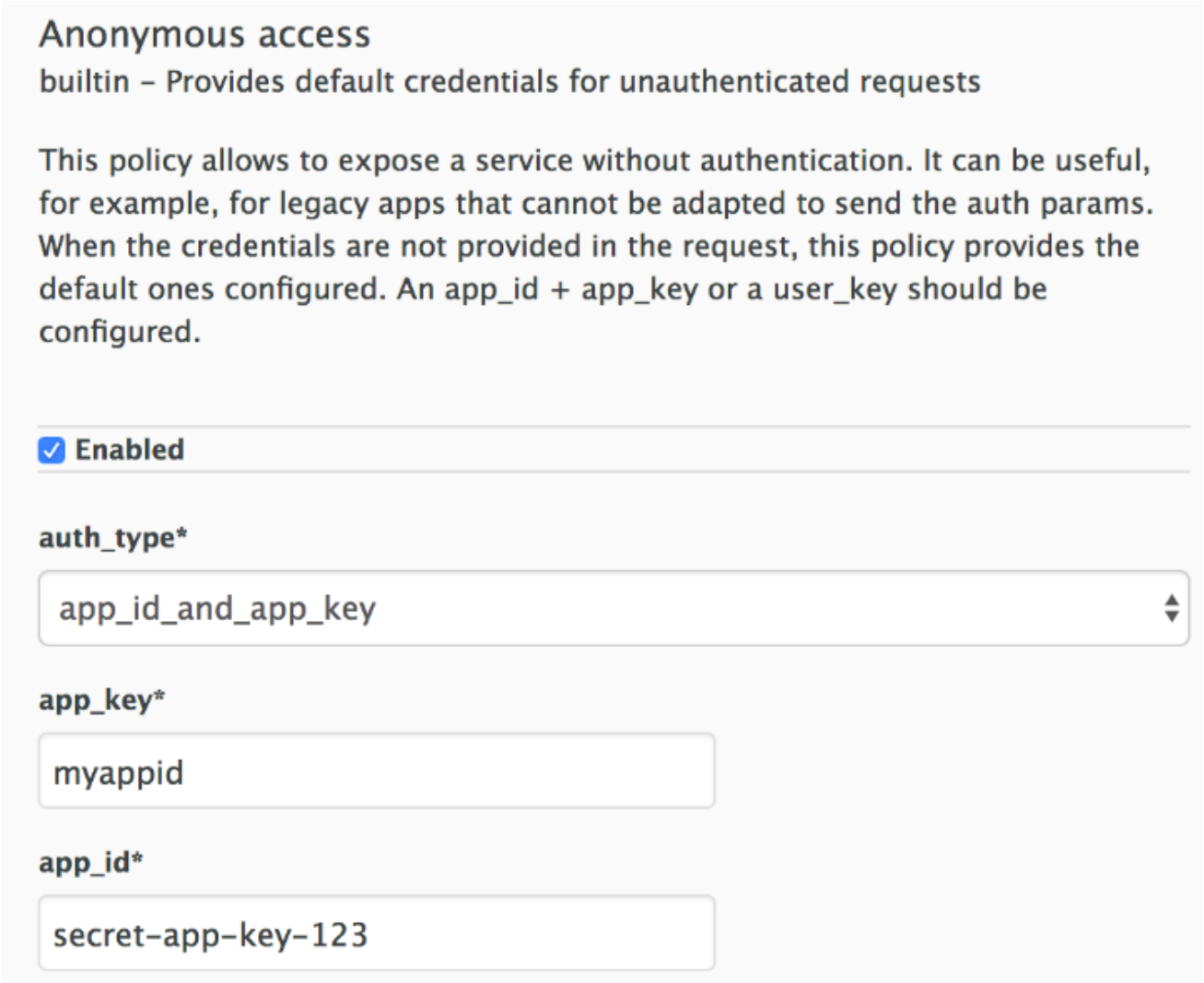
注記

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。

- **auth_type**: 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



4.1.4. CORS Request Handling ポリシー

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

- 許可されるクレデンシャル
- 許可されるオリジンヘッダー

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	任意

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"
  }
}
```

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。

4.1.5. Echo ポリシー

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	いいえ
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。

4.1.6. Edge Limiting ポリシー

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: 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 }}** として設定します)。
- 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。

4.1.6.1. 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にした leaky_bucket アルゴリズムに基づきます。
- **fixed_window_limiters**: 特定の期間 (最後の X 秒) に基づきます。
- **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。

4.1.6.2. 制限の定義

リミッターはキーを持ち、このキーで制限を定義するのに使用するエンティティーをエンコードします (IP、サービス、エンドポイント、ID、特定ヘッダーの値など)。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**: キーの名前です。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。

それぞれの制限には、そのタイプに応じたパラメーターもあります。

- **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。許容レート (**rate** で指定される) 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**、**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**: 制限を超える接続を遅延させる秒数です。

例

1. 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
}
```

2. burst および delay をそれぞれ 10 および 1 に設定して、100 の接続を許可します。

```
{
  "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。

4.1.6.3. 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 パラメーターを使用することができます。例: 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 }}**。

以下に例を示します。

```
{
  "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。

4.1.6.4. 条件の適用

条件は、API ゲートウェイがリミッターをいつ適用するかを定義します。各リミッターの **condition** プロパティーで少なくとも 1 つの操作を指定する必要があります。

以下のプロパティーで **condition** を定義します。

- **combine_op**。演算のリストに適用されるブール演算子です。**or** および **and** の 2 つ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。

以下に例を示します。

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

4.1.6.5. ストアの設定

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 インスタンスが使用されている場合に役立ちます。Redis サーバーは、**redis_url** パラメーターを使用して設定することができます。

4.1.6.6. 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設定することができます。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。デフォルトは **429** です。
  - **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設定することができます。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。デフォルトは **500** です。
  - **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。

4.1.7. Header Modification ポリシー

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。

4.1.8. IP Check ポリシー

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

プロパティー	説明	データタイプ	必須/任意
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	いいえ
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)。	いいえ

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。

4.1.9. JWT Claim Check ポリシー

4.1.9.1. JWT Claim Check ポリシーの概要

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 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"
        }
     ]
  }
}
```

4.1.9.2. ポリシーチェーンへの JWT Claim Check ポリシーの設定

4.1.9.2.1. 前提条件

- 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。

4.1.9.2.2. ポリシーの設定

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 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。

付加手順:

- 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。

変更を保存するには、**Update & test in Staging Environment**をクリックします。

4.1.10. Liquid Context Debug ポリシー

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。

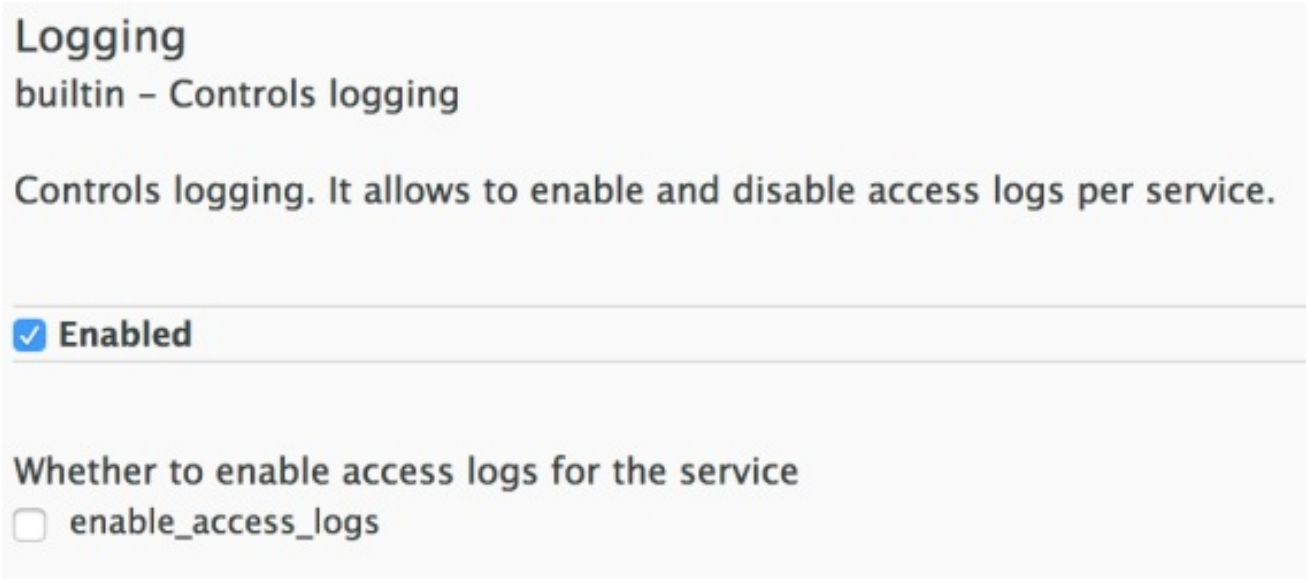
このポリシーが有効な時に APIcast が返す値の例を以下に示します。

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

4.1.11. Logging ポリシー

Logging ポリシーにより、各 API サービスの APIcast (NGINX) アクセスログを個別に有効または無効にすることができます。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。

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。サービスのアクセスロギングを無効にするには、ポリシーを有効にし、**enable_access_logs** パラメーターの選択を解除して、**Submit** ボタンをクリックします。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。



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、???を参照してください。

4.1.12. OAuth 2.0 Token Introspection ポリシー

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。

例4.1 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

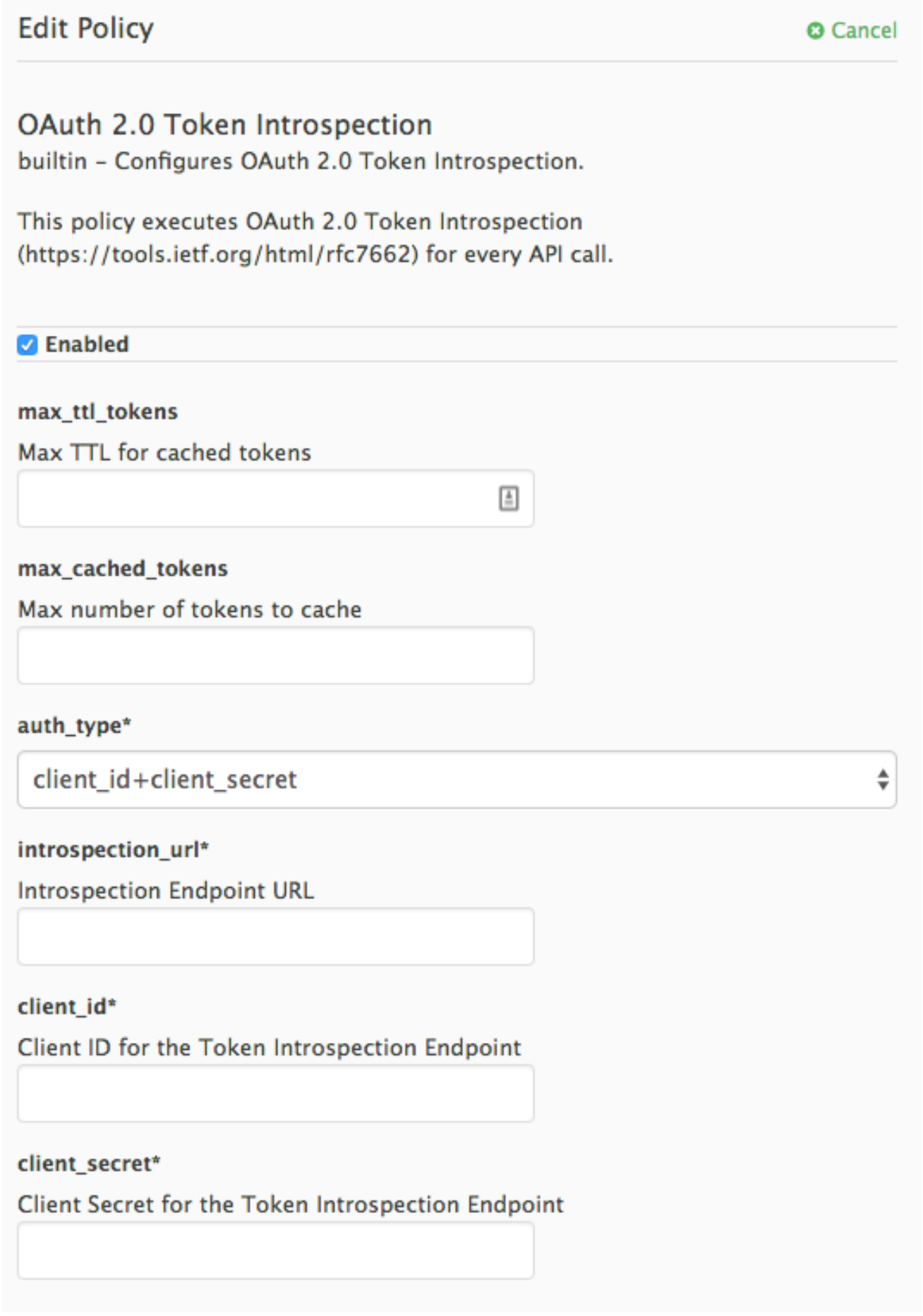
る **クライアント ID** および **クライアントシークレット** と共に、異なるトークンイントロスペクションエンドポイントを指定することができます。このオプションを使用する場合には、以下の設定パラメーターを定義します。

- **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。

例4.2 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。

4.1.13. Prometheus メトリクス

Prometheus はスタンドアロンのオープンソースのシステムモニターリングおよびアラート用ツールキットです。

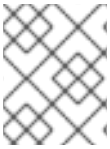
重要

Red Hat 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アラートを視覚化することができます。

Prometheus メトリクスの可用性

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 API Management およびオンプレミス型 API Management 両方との組み合わせに対応)
- Embedded APIcast (オンプレミス型 API Management)



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。

Prometheus メトリクスのリスト

以下のメトリクスは常に使用可能です。

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	-
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_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_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_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_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

メトリクス	説明	タイプ	ラベル
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	エンドポイント (authrep、auth、report)、ステータス (2xx、4xx、5xx)

以下のメトリクスは、3scale Batcher ポリシーを使用する場合にのみ使用可能です。

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	-
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	-

値を持たないメトリクス

メトリクスが値を持たない場合には、メトリクスは非表示になります。たとえば、**nginx_error_log** に報告するエラーがない場合、**nginx_error_log** メトリックは表示されません。値を持った場合にのみ表示されます。

4.1.14. Referrer ポリシー

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 は今後のリクエストの **Referer** ポリシーの値を **referrer** パラメーターで Service Management API (AuthRep 呼び出し) に送信します。参照元フィルター機能の仕組みの詳細については、**Authentication Patterns**の Referrer filtering セクションを参照してください。

4.1.15. Retry ポリシー

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 2.6 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
      }
    }
  }
}
```

4.1.16. RH-SSO/Keycloak Role Check ポリシー

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** (デフォルト): **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場合に使用します (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** のどちらかを指定して、name がどのように評価されるかを定義します (**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** のどちらかを指定して、**name** の値がどのように評価されるかを定義します (**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** のどちらかを指定して、**client** の値がどのように評価されるかを定義します (**resource_type** の場合と同じように機能します)。

4.1.17. Routing ポリシー

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

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は APIcast ポリシーの前に Routing ポリシーを設定する必要があります。2 つのポリシーのうち始めのポリシーがレスポンスにコンテンツを出力するためです。2 番目のポリシーにコンテンツフェーズを実行する変更が加えられても、リクエストはすでにクライアントに送信されるため、レスポンスには何も出力されません。

4.1.17.1. 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。

4.1.17.2. 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.3. 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.4. 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.5. 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.6. 複数演算ルール

ルールに複数の演算を設定し、すべてが true と評価される場合にだけ (and**combine_op** を使用) 指定したアップストリームにルーティングすることや、少なくとも 1 つが true と評価される場合に (or**combine_op** を使用) 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.7. 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.8. 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": []
        }
      }
    ]
  }
}
```

4.1.17.9. 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.10. 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"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.17.11. Host ヘッダー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": "/"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.18. SOAP ポリシー

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、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。

4.1.19. TLS Client Certificate Validation ポリシー

4.1.19.1. TLS Client Certificate Validation ポリシーの概要

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。

4.1.19.2. 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。

4.1.19.2.1. 前提条件:

- 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。

4.1.19.2.2. 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/2.6.0.GA/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
```

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

4.1.19.3.1. 前提条件

- 3scale へ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。

4.1.19.3.2. ポリシーの設定

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 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 & test in Staging Environment**をクリックします。

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

4.1.19.4.1. 前提条件

- 3scale へ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。

4.1.19.4.2. ポリシー機能の確認

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
```

4.1.19.5. ホワイトリストからの証明書の削除

4.1.19.5.1. 前提条件

- 3scale へ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。

4.1.19.5.2. 証明書の削除

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 & test in Staging Environment**をクリックします。

4.1.19.6. リファレンス資料

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。

4.1.20. Upstream ポリシー

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

プロパティー	説明	値	必須/任意
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。

4.1.21. Upstream Connection ポリシー

4.1.21.1. Upstream Connection ポリシーの概要

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**

4.1.21.2. ポリシーチェーンへの Upstream Connection ポリシーの設定

4.1.21.2.1. 前提条件

- 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。

4.1.21.2.2. ポリシーの設定

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 **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 & test in Staging Environment**をクリックします。

4.1.22. URL Rewriting ポリシー

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 3scale 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。

4.1.22.1. 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** (オプション): 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** (オプション): true に設定すると (チェックボックスを選択する)、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ます (リスト内の後続コマンドはすべて破棄されます)。

4.1.22.2. 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することができます。
  - **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** (オプション): 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、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。

4.1.23. URL Rewriting with Captures ポリシー

URL Rewriting with Captures ポリシーは「URL Rewriting ポリシー」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
```

4.2. 管理ポータルでのポリシーの有効化

管理ポータルでポリシーを有効にするには、以下の手順を実施します。

1. 3scale にログインします。
2. **API service** に移動します。
3. **[your_API_name] > Integration > Configuration**の順に移動し、**edit APIcast configuration** を選択します。
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、必須フィールドに入力します。
6. **Update & test in Staging Environment**ボタンをクリックし、ポリシーチェーンを保存します。

4.3. カスタム APICAST ポリシーの作成

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。

4.4. APICAST へのカスタムポリシーの追加

カスタムポリシーを作成したら、それを APIcast に追加する必要があります。その手順は、APIcast がデプロイされている場所により異なります。

以下に示す Self-managed APIcast デプロイメントに、カスタムポリシーを追加することができます。

- Embedded APIcast ゲートウェイ (OpenShift 上の 3scale オンプレミスデプロイメントの一部)
- OpenShift および Docker コンテナー環境上の 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。

4.4.1. 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。Red Hat 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タグ (**:latest**) 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。実稼働環境 (あるいは、必要であればステージング環境) のサービスが中断されるのを避けるためには、自動デプロイメントを無効にするか (**Automatically start a new deployment when the image changes**チェックボックス) か、実稼働環境用に別のイメージストリームタグ (例: **amp-apicast:production**) を設定することをお勧めします。

カスタムポリシーをオンプレミスデプロイメントに追加するには、以下の手順に従います。

1. https://github.com/3scale/apicast-example-policy (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。

2. 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。

3. **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)。

4. 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=2.6.0
```

5. 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** が再開されたら管理ポータルの Integration ページに移動し、**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 & test in Staging Environment** 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。

4.4.2. 別の OpenShift Container Platform 上の APIcast 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。

別の OpenShift Container Platform 上の APIcast にカスタムポリシーを追加します

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 2.6 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

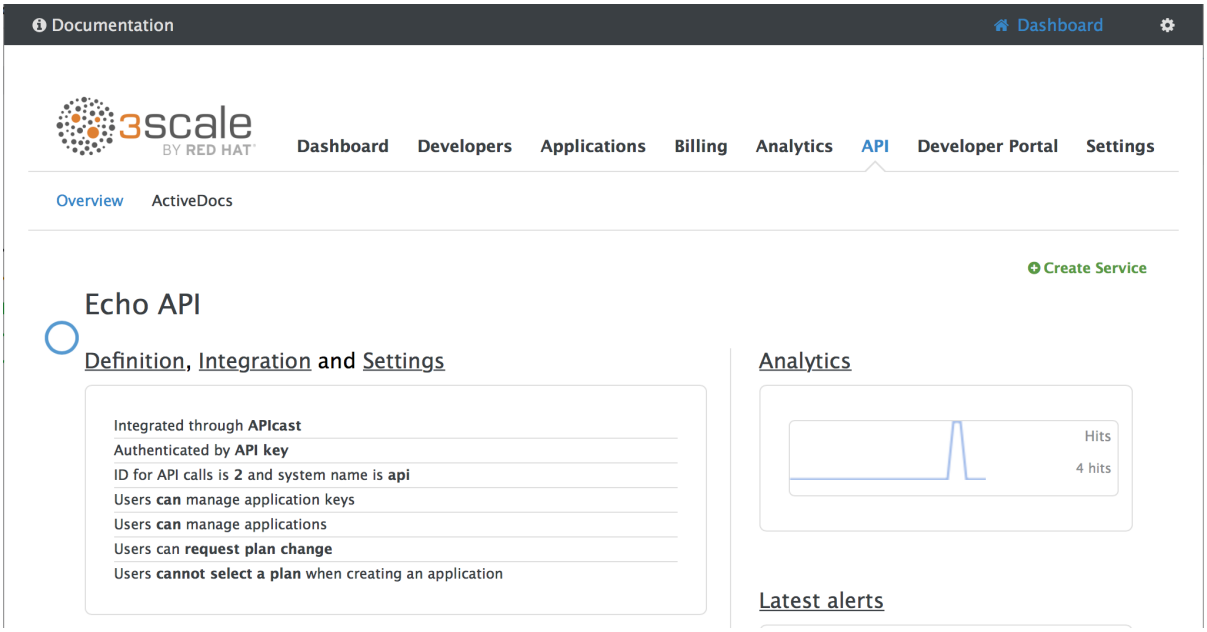
カスタムポリシーが APIcast に追加されて新しいイメージがビルドされ、そのイメージを使用して APIcast がデプロイされると、管理ポータルではそれらのポリシーが利用可能なポリシーとして自動的に表示されます。既存のサービスはこの新しいポリシーを利用可能なポリシーリストで認識できるので、任意のポリシーチェーンで使用することができます。

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。

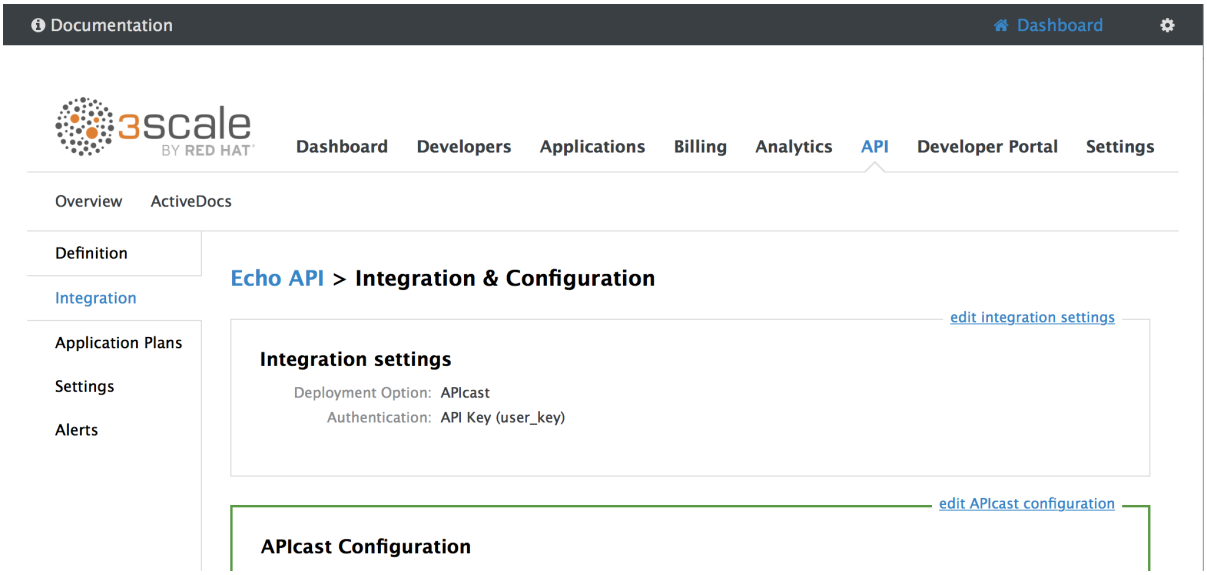
4.5. 3SCALE でのポリシーチェーンの作成

APIcast ゲートウェイ設定の一部として、3scale でポリシーチェーンを作成します。UI でポリシーチェーンを変更するには、以下の手順に従います。

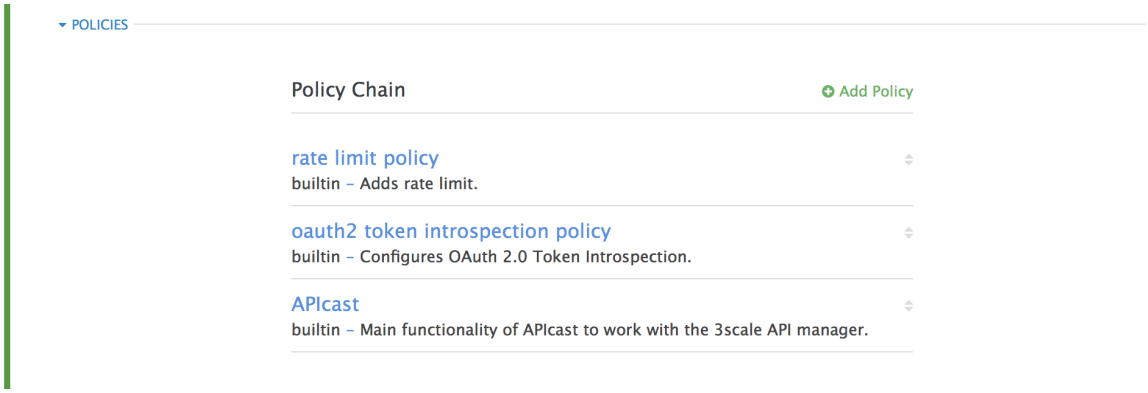
1. AMP にログインします。
2. API サービスに移動します。



3. [your_API_name] > Integration > Configurationの順に移動し、**edit APIcast configuration** を選択します。



4. **POLICIES** 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。必ず **APIcast** ポリシーをポリシーチェーンの最後に設定します。



5. **Update & test in Staging Environment**ボタンをクリックし、ポリシーチェーンを保存します。

4.6. ポリシーチェーン JSON 設定ファイルの作成

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、AMP の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 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。

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「APIcast 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。たとえば **"app_id": "972f7b4f"**、**"user_key": "13b668c4d1e10eaebaa5144b4749713f"** 等。
- **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 Context Debug ポリシー」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 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

- APICAST_BACKEND_CACHE_HANDLER
- APICAST_CONFIGURATION_CACHE
- APICAST_CONFIGURATION_LOADER
- APICAST_CUSTOM_CONFIG
- APICAST_ENVIRONMENT
- APICAST_EXTENDED_METRICS
- APICAST_LOG_FILE
- APICAST_LOG_LEVEL
- APICAST_ACCESS_LOG_FILE
- APICAST_OIDC_LOG_LEVEL
- APICAST_MANAGEMENT_API
- APICAST_MODULE
- APICAST_PATH_ROUTING
- APICAST_PATH_ROUTING_ONLY
- APICAST_POLICY_LOAD_PATH
- APICAST_PROXY_HTTPS_CERTIFICATE_KEY
- APICAST_PROXY_HTTPS_CERTIFICATE
- APICAST_PROXY_HTTPS_PASSWORD_FILE
- APICAST_PROXY_HTTPS_SESSION_REUSE
- APICAST_HTTPS_VERIFY_DEPTH
- APICAST_REPORTING_THREADS
- APICAST_RESPONSE_CODES
- APICAST_SERVICES_FILTER_BY_URL

- APICAST_SERVICES_LIST
- APICAST_UPSTREAM_RETRY_CASES
- APICAST_SERVICE_${ID}_CONFIGURATION_VERSION
- APICAST_WORKERS
- BACKEND_ENDPOINT_OVERRIDE
- OPENSSL_VERIFY
- RESOLVER
- THREESCALE_CONFIG_FILE
- THREESCALE_DEPLOYMENT_ENV
- THREESCALE_PORTAL_ENDPOINT
- OPENTRACING_TRACER
- OPENTRACING_CONFIG
- OPENTRACING_HEADER_FORWARD
- APICAST_HTTPS_PORT
- APICAST_HTTPS_CERTIFICATE
- APICAST_HTTPS_CERTIFICATE_KEY
- all_proxy ALL_PROXY
- http_proxy HTTP_PROXY
- https_proxy HTTPS_PROXY
- no_proxy NO_PROXY

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_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 が読み込む環境 (またはパス) のコロン (**:**) 区切りリスト。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_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。詳細は、NGINX のドキュメント を参照してください。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。デフォルトのプリフィックスディレクトリーは APIcast である点に注意してください。

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_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_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_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_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_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_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_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_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** の場合には、チェーンに 1 つの追加証明書 (たとえば **中間 CA**) が含まれる可能性があることを意味します。

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

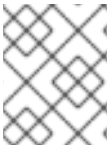
デフォルト: 空欄 (**false**)

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。レスポンスコード機能の詳細は、3scale カスタマーポータル を参照してください。

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、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6.1 example

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

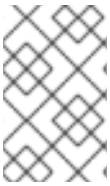
4. http://prod.bar.dev

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

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_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_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_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**。

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。

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。

Proxy Config Show および **Proxy Config Show Latest** エンドポイントは、サービスによって範囲が限定され、**Proxy Configs List** サービスも対象となります。サービスの ID を知っている必要があります。以下のオプションを使用します。

- **Proxy Configs List** プロバイダーエンドポイントを使用します: **<schema>://<admin-portal-domain>/admin/api/account/proxy_configs/<env>.json**
    - エンドポイントは、各サービスの最新のものだけでなく、プロバイダーの保存されているすべてのプロキシー設定を返します。JSON で返された **proxy_configs** の配列を繰り返し処理し、同じ **proxy_config.content.id** (サービスの ID) を持つすべてのプロキシー設定の中で **proxy_config.version** が最も高い **proxy_config.content** を選択します。
- **Service List** エンドポイントの使用: **/admin/api/services.json**
    - エンドポイントには、プロバイダーのすべてのサービスが一覧表示されます。サービスの配列を繰り返し処理し、サービスごとに、サービスによってスコープが設定された **Proxy Config Show Latest** エンドポイントを消費します。

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**。

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。

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。

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_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

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。

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。

第7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」

7.1. 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。

7.2. 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。

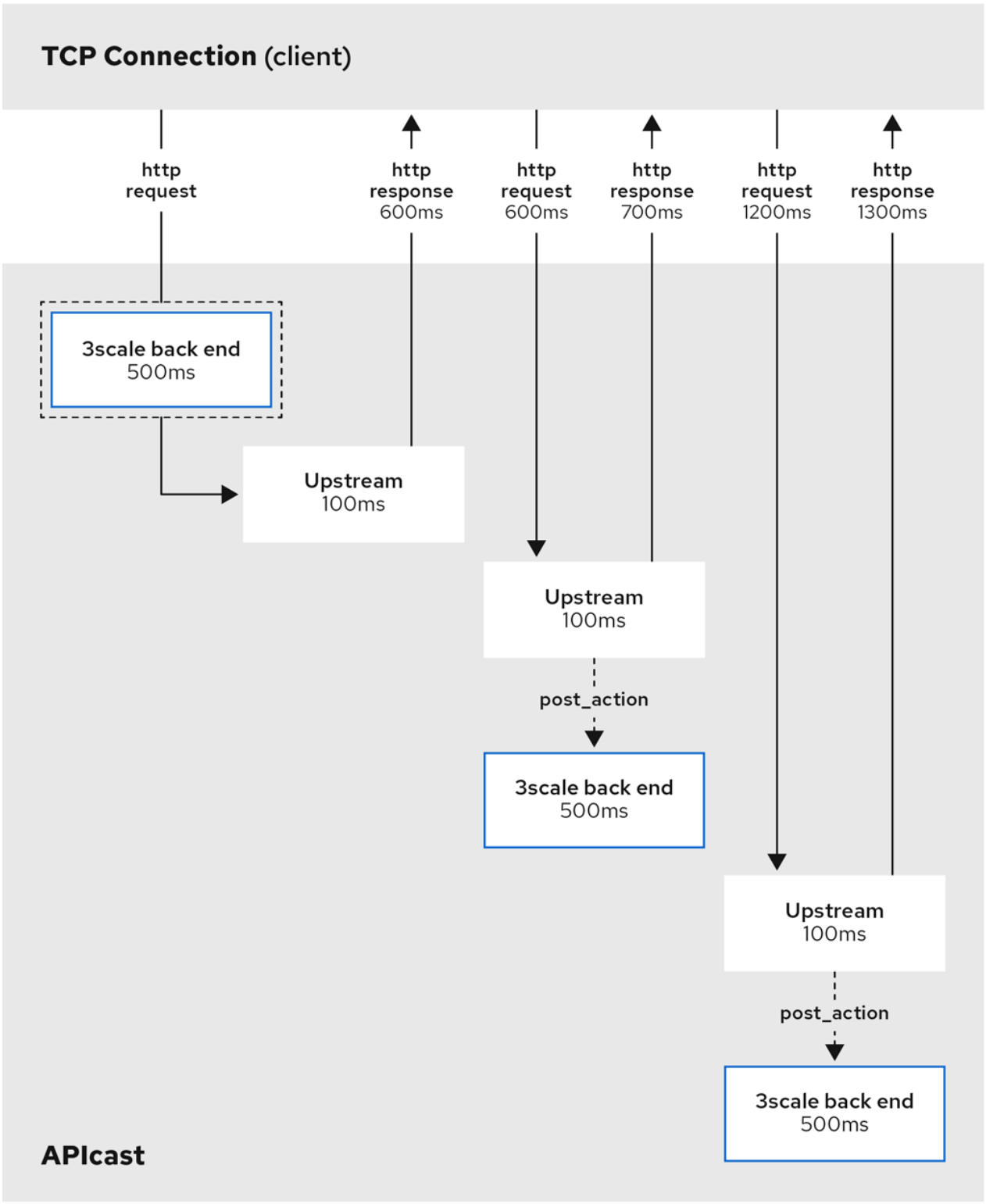
一方、デフォルトで有効になっているキャッシングメカニズムを使用した場合には、フローは以下のようになります。

- 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。



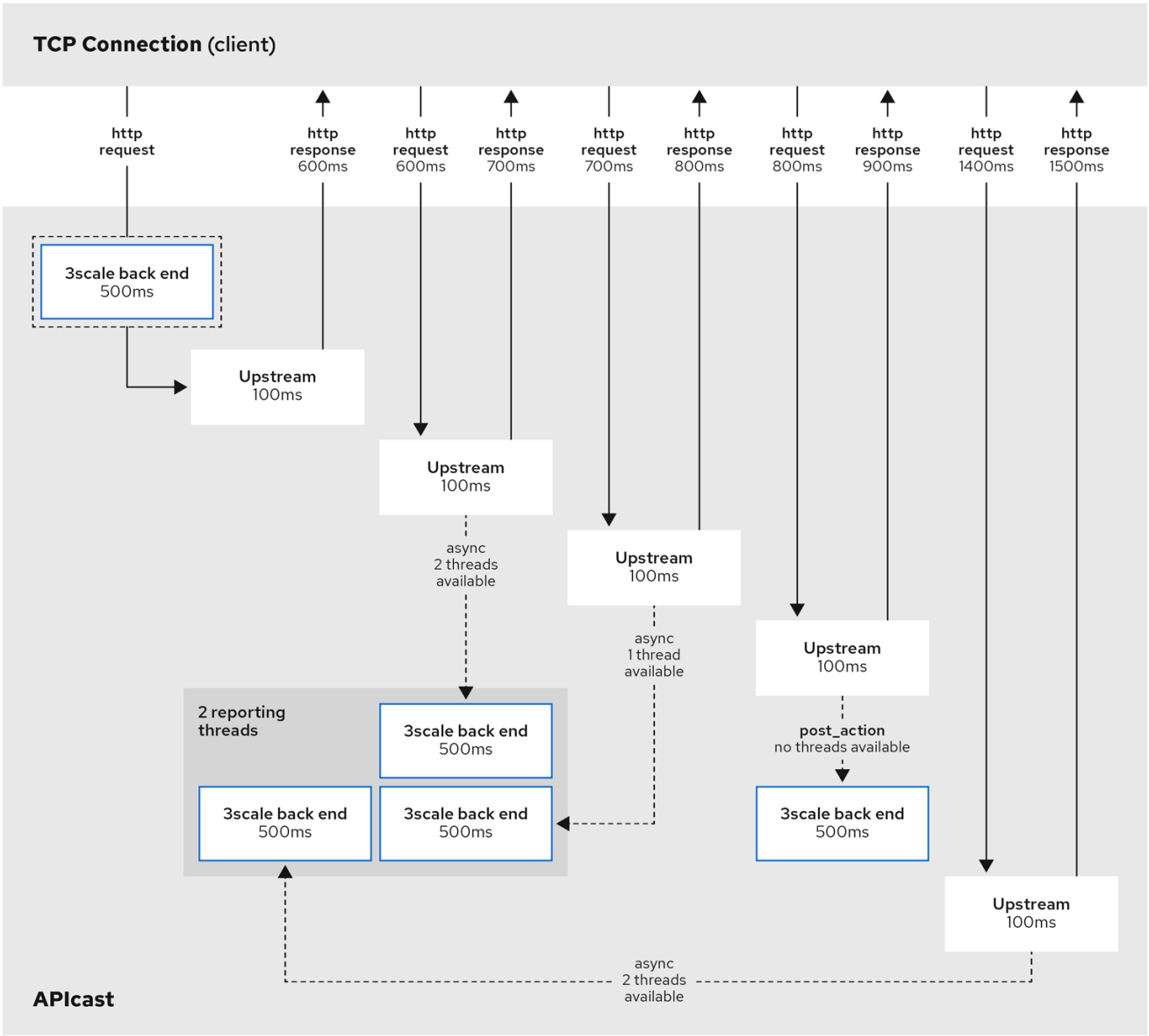
7.3. 非同期レポートスレッド

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。

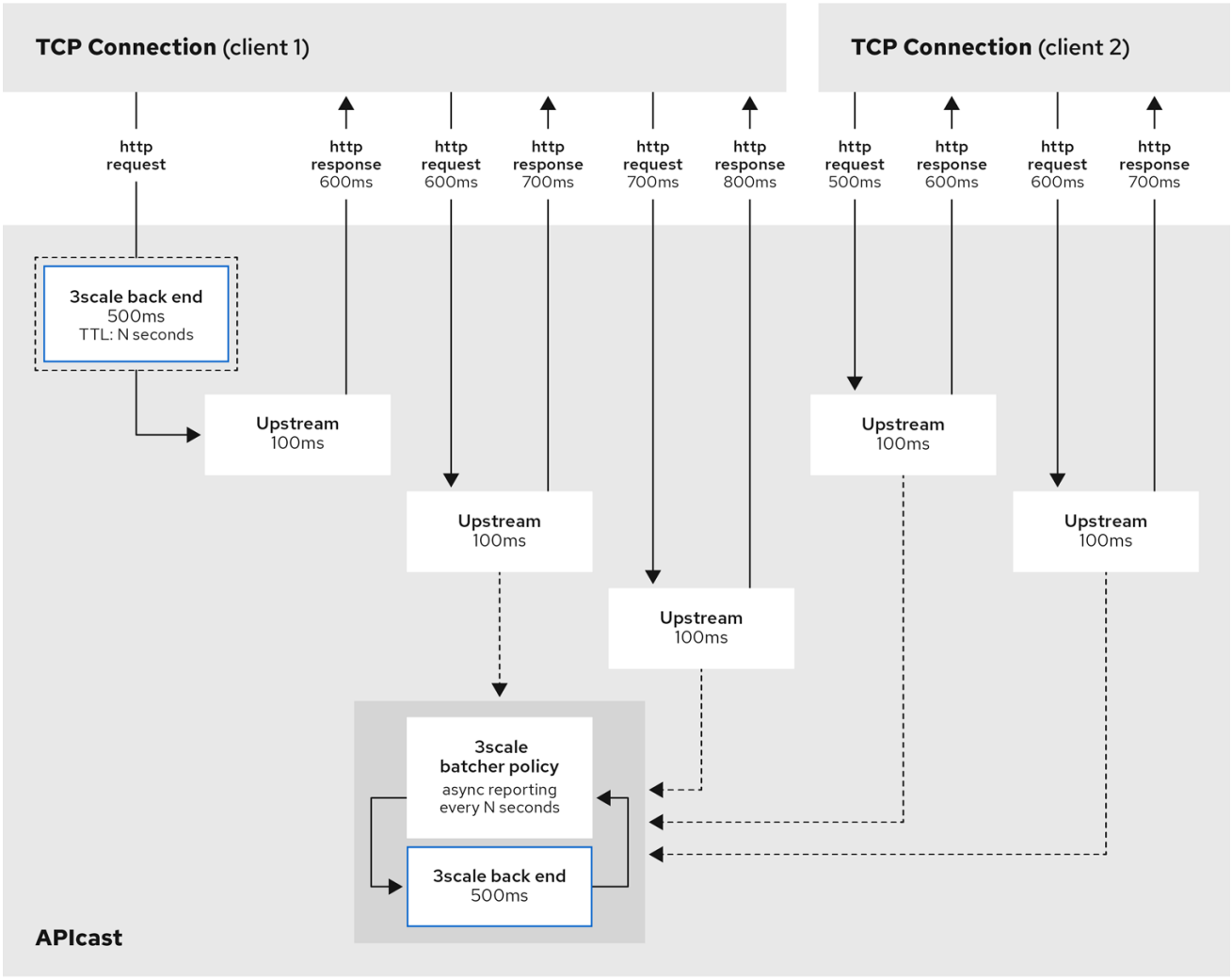


3Scale_38_0819

7.4. 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。

このセクション に、3scale Batcher ポリシーの詳細を記載しています。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
3Scale_38_0819