



# Red Hat 3scale API Management 2.13

## 3scale の操作

デプロイメントの自動化、環境のスケーリング、および問題のトラブルシューティングを行う方法



## Red Hat 3scale API Management 2.13 3scale の操作

---

デプロイメントの自動化、環境のスケーリング、および問題のトラブルシューティングを行う方法

## 法律上の通知

Copyright © 2024 Red Hat, Inc.

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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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

本書では、Red Hat 3scale API Management 2.13 のデプロイメント操作について説明します。

## 目次

多様性を受け入れるオープンソースの強化 .....	4
第1章 3SCALE の一般的な設定オプション .....	5
1.1. 有効なログインセッションの長さの設定 .....	5
第2章 3SCALE の操作とスケーリング .....	6
2.1. APICAST の再デプロイ .....	6
2.2. オンプレミス型 3SCALE のスケールアップ .....	7
2.3. 操作のトラブルシューティング .....	9
第3章 3SCALE のモニタリング .....	13
3.1. 3SCALE のモニタリングの有効化 .....	14
3.2. 3SCALE を監視するための PROMETHEUS の設定 .....	15
3.3. 3SCALE を監視するための GRAFANA の設定 .....	16
3.4. 3SCALE のメトリックの表示 .....	18
3.5. PROMETHEUS に公開される 3SCALE システムメトリック .....	18
第4章 WEBHOOK を使用した 3SCALE 自動化 .....	20
4.1. WEBHOOK の概要 .....	20
4.2. WEBHOOK の設定 .....	20
4.3. WEBHOOK のトラブルシューティング .....	21
第5章 3SCALE TOOLBOX .....	22
5.1. TOOLBOX のインストール .....	22
5.2. サポートされる TOOLBOX コマンド .....	23
5.3. サービスのインポート .....	24
5.4. サービスのコピー .....	24
5.5. サービス設定のみのコピー .....	25
5.6. OPENAPI 認証 .....	26
5.7. OPENAPI 定義のインポート .....	27
5.8. OPENAPI 定義からの 3SCALE バックエンドのインポート .....	29
5.9. リモートアクセスクレデンシャルの管理 .....	30
5.10. アプリケーションプランの作成 .....	32
5.11. メトリックの作成 .....	38
5.12. メソッドの作成 .....	40
5.13. サービスの作成 .....	43
5.14. ACTIVEDOCS の作成 .....	45
5.15. プロキシ設定のリスト表示 .....	48
5.16. ポリシーレジストリーのコピー .....	51
5.17. アプリケーションのリスト表示 .....	52
5.18. プロダクトのエクスポート .....	55
5.19. プロダクトのインポート .....	59
5.20. プロダクトポリシーチェーンのエクスポートおよびインポート .....	61
5.21. API バックエンドのコピー .....	63
5.22. API プロダクトのコピー .....	64
5.23. SSL および TLS に関する問題のトラブルシューティング .....	65
第6章 3SCALE での API 環境のマッピング .....	67
6.1. 環境ごとのプロダクト .....	67
6.2. オンプレミス型 3SCALE インスタンス .....	68
6.3. 3SCALE の混合アプローチ .....	69
6.4. 3SCALE と APICAST ゲートウェイの組み合わせ .....	69

<b>第7章 AUTOMATING API LIFECYCLE WITH 3SCALE TOOLBOX</b> .....	<b>71</b>
7.1. API ライフサイクルステージの概要	71
7.2. サンプル JENKINS CI/CD パイプラインのデプロイ	73
7.3. 3SCALE JENKINS 共有ライブラリーを使用したパイプラインの作成	81
7.4. JENKINSFILE を使用したパイプラインの作成	83
<b>第8章 3SCALE OPERATOR を使用した 3SCALE の設定とプロビジョニング</b> .....	<b>88</b>
8.1. 一般的な前提条件	88
8.2. 3SCALE OPERATOR を使用したアプリケーション CAPABILITIES	89
8.3. 最初の 3SCALE プロダクトおよびバックエンドのデプロイ	89
8.4. 製品の APICAST 設定のプロモート	91
8.5. 3SCALE OPERATOR が、カスタムリソースのリンク先となるテナントを識別する方法	93
8.6. 3SCALE OPENAPI カスタムリソースのデプロイ	94
8.7. 3SCALE ACTIVEDOC カスタムリソースのデプロイ	99
8.8. CAPABILITIES に関連するバックエンドカスタムリソース	102
8.9. CAPABILITIES に関連するプロダクトカスタムリソース	108
8.10. 3SCALE CUSTOMPOLICYDEFINITION カスタムリソースのデプロイ	120
8.11. テナントカスタムリソースのデプロイ	121
8.12. カスタムリソースのデプロイによる 3SCALE 開発者の管理	123
8.13. 3SCALE OPERATOR の機能の制限	129
8.14. 関連情報	130
<b>第9章 3SCALE のバックアップと復元</b> .....	<b>131</b>
9.1. 前提条件	131
9.2. 永続ボリュームおよび考慮事項	131
9.3. データセットの使用	132
9.4. システムデータベースのバックアップ	133
9.5. システムデータベースの復元	134
<b>第10章 3SCALE 用 RECAPTCHA の設定</b> .....	<b>145</b>
10.1. 3SCALE のスパム保護用 RECAPTCHA の設定	145
<b>第11章 3SCALE WEBASSEMBLY モジュール</b> .....	<b>147</b>
11.1. BOOKINFO アプリケーションを SERVICE MESH にデプロイする	147
11.2. 3SCALE で製品を作成する	148
11.3. 3SCALE と SERVICE MESH の接続	148
11.4. バックエンド URL を SERVICE MESH に追加する	149
11.5. SERVICE MESH と同じクラスターで 3SCALE を使用する	150
11.6. WASMPPLUGIN カスタムリソースの作成	151
11.7. 設定された API のテスト	155
11.8. 3SCALE WEBASSEMBLY モジュール設定	156
11.9. 認証情報ユースケースの 3SCALE WEBASSEMBLY モジュールの例	167
11.10. 3SCALE WEBASSEMBLY モジュールの機能する最低限の設定	171
<b>第12章 API インフラストラクチャーに関するトラブルシューティング</b> .....	<b>174</b>
12.1. インテグレーションに関する典型的な問題	174
12.2. API インフラストラクチャーに関する問題への対応	179
12.3. API へのリクエストに関する問題の特定	182
12.4. ACTIVEDOCS の問題	186
12.5. NGINX でのロギング	187
12.6. 3SCALE のエラーコード	187



## 多様性を受け入れるオープンソースの強化

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



## 第1章 3SCALE の一般的な設定オプション

Red Hat 3scale API Management 管理者は、インストールまたはアカウントで一般的な設定オプションを使用して設定を調整できます。

### 1.1. 有効なログインセッションの長さの設定

Red Hat 3scale API Management 管理者は、管理ポータルとデベロッパーポータルで有効なログインセッションの長さを設定して、最大タイムアウトおよびアクティブではない状態に対して、制限を指定できます。

有効なログインセッションの長さを実装するには、**USER\_SESSION\_TTL** を秒単位で設定する必要があります。たとえば、1,800 秒は 30 分です。値が **null** (設定されていないか、空の文字列に設定されている) の場合には、セッションのデフォルトの長さは 2 週間になります。

#### 前提条件

- 管理者権限が設定された 3scale アカウント

#### 手順

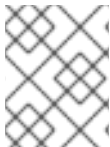
1. **system-app** シークレットの **USER\_SESSION\_TTL** の値を更新します (秒単位)。

```
$ oc patch secret system-app -p '{"stringData": {"USER_SESSION_TTL": "1800"}}'
```

2. **system-app** をロールアウトします。

```
$ oc rollout latest dc/system-app
```

## 第2章 3SCALE の操作とスケーリング



### 注記

本書は、ノートパソコンやこれに類するエンドユーザー機器上のローカルインストールを対象としていません。

本章では、Red Hat 3scale API Management 2.13 インストール環境での操作とスケーリングタスクについて説明します。

### 前提条件

- [サポート対象バージョンの OpenShift](#) にインストールされた初期設定済みのオンプレミス型 3scale インスタンス。

3scale の操作およびスケーリングタスクを行うには、以下のセクションで説明している手順を実行します。

- [APICast の再デプロイ](#)
- [オンプレミス型 3scale のスケールアップ](#)
- [操作のトラブルシューティング](#)

### 2.1. APICAST の再デプロイ

3scale 管理ポータルで、システムの変更をテストしプロモートすることができます。

#### 前提条件

- デプロイされたオンプレミス型 3scale のインスタンス。
- APICast のデプロイメント方法を選択している。

デフォルトでは、OpenShift 上の APICast デプロイメントでは (Embedded 型のデプロイおよび他の OpenShift クラスタ上のデプロイの両方で)、3scale 管理ポータルを介して変更をステージング環境と実稼働環境用のゲートウェイにパブリッシュできるように設定されています。

APICast を OpenShift に再デプロイするには、以下の手順を実施します。

#### 手順

1. システムに変更を加えます。
2. 管理ポータルでステージング環境にデプロイしてテストします。
3. 管理ポータルで実稼働環境にプロモートします。

デフォルトでは、APICast はプロモートされた更新を 5 分ごとに取得し、パブリッシュします。

Docker コンテナ環境またはネイティブインストールで APICast を使用している場合は、ステージング環境用と実稼働環境用のゲートウェイを設定し、パブリッシュした変更をゲートウェイが取得する頻度を指定します。APICast ゲートウェイを設定したら、3scale 管理ポータルで APICast を再デプロイできます。

Docker コンテナ環境またはネイティブインストールに APIcast を再デプロイするには、以下を実行します。

### 手順

1. APIcast ゲートウェイを設定し、オンプレミス型 3scale に接続します。
2. システムに変更を加えます。
3. 管理ポータルでステージング環境にデプロイしてテストします。
4. 管理ポータルで実稼働環境にプロモートします。

APIcast は、設定された頻度でプロモートされた更新を取得してパブリッシュします。

## 2.2. オンプレミス型 3SCALE のスケールアップ

APIcast デプロイメントの規模が大きくなると、利用可能なストレージの量を増やす必要が生じる可能性があります。ストレージをスケールアップする方法は、永続ストレージに使用しているファイルシステムのタイプによって異なります。

ネットワークファイルシステム (NFS) を使用している場合は、以下のコマンドを使用して永続ボリューム (PV) をスケールアップできます。

```
$ oc edit pv <pv_name>
```

他のストレージ手段を使用している場合は、以降のセクションに挙げる方法のいずれかを使用して、永続ボリュームを手動でスケールアップする必要があります。

### 2.2.1. 方法 1: 永続ボリュームをバックアップしてスワップする

#### 手順

1. 既存の永続ボリューム上のデータをバックアップします。
2. 新しいサイズ要件に合わせて、ターゲット永続ボリュームを作成し、アタッチします。
3. 事前バインド型の永続ボリューム要求を作成し、新しい PVC (PersistentVolumeClaim) のサイズと永続ボリュームの名前を指定します。永続ボリューム名には **volumeName** フィールドを使用します。
4. 新しく作成した PV に、バックアップからデータを復元します。
5. 新しい PV の名前でデプロイメント設定を変更します。

```
$ oc edit dc/system-app
```

6. 新しい PV が設定され正常に機能していることを確認します。
7. 以前の PVC を削除して、それが要求していたリソースを解放します。

### 2.2.2. 方法 2: 3scale をバックアップして再デプロイする

## 手順

1. 既存の永続ボリューム上のデータをバックアップします。
2. 3scale Pod をシャットダウンします。
3. 新しいサイズ要件に合わせて、ターゲット永続ボリュームを作成し、アタッチします。
4. 新しく作成した PV に、バックアップからデータを復元します。
5. 事前バインド型の永続ボリューム要求を作成します。以下の項目を指定します。
  - a. 新しい PVC のサイズ
  - b. 永続ボリューム名 (**volumeName** フィールドを使用)
6. **amp.yml** をデプロイします。
7. 新しい PV が設定され正常に機能していることを確認します。
8. 以前の PVC を削除して、それが要求していたリソースを解放します。

### 2.2.3. オンプレミス型 3scale デプロイメントの設定

3scale でスケーリングされる主要なデプロイメント設定項目は以下のとおりです。

- 実稼働環境用 APIcast
- バックエンドリスナー
- バックエンドワーカー

#### 2.2.3.1. OCP によるスケーリング

APIManager CR を使用する OpenShift Container Platform (OCP) を介して、デプロイメント設定をスケールアップまたはスケールダウンできます。

特定のデプロイメント設定をスケーリングするには、以下を使用します。

- 次の APIManager CR を使用して APIcast 実稼働デプロイメント設定をスケールアップします。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      replicas: X
```

- 次の APIManager CR を使用して、デプロイメント設定のバックエンドリスナー、バックエンドワーカー、およびバックエンド cron コンポーネントをスケールアップします。

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

```

name: example-apimanager
spec:
  backend:
    listenerSpec:
      replicas: X
    workerSpec:
      replicas: Y
    cronSpec:
      replicas: Z

```

- 適切な環境変数に、希望する Pod ごとのプロセス数を設定します。

- **backend-listener** Pod の **PUMA\_WORKERS**:

```
$ oc set env dc/backend-listener --overwrite PUMA_WORKERS=
<number_of_processes>
```

- **system-app** Pod の **UNICORN\_WORKERS**:

```
$ oc set env dc/system-app --overwrite UNICORN_WORKERS=
<number_of_processes>
```

### 2.2.3.2. ハードウェアの垂直スケーリングと水平スケーリング

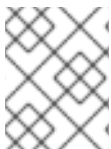
リソースを追加することで、OpenShift 上の 3scale デプロイメントのパフォーマンスを高めることができます。水平スケーリングとして OpenShift クラスタにより多くのコンピュータノードを Pod として追加することや、垂直スケーリングとして既存のコンピュータノードにより多くのリソースを割り当てることができます。

#### 水平スケーリング

コンピュータノードを Pod として OpenShift に追加することができます。追加のコンピュータノードがクラスタ内の既存ノードと一致する場合には、環境変数を再設定する必要はありません。

#### 垂直スケーリング

既存のコンピュータノードに割り当てるリソースを増やすことができます。割り当てるリソースを増やす場合は、追加のプロセスを Pod に追加してパフォーマンスを高める必要があります。



#### 注記

3scale デプロイメントにおいて、仕様や設定の異なるコンピュータノードを使用しないでください。

### 2.2.3.3. ルーターのスケールアップ

トラフィックの増加に応じて、OCP ルーターがリクエストを適切に処理できるようにしてください。ルーターがリクエストのスループットを制限している場合には、ルーターノードをスケールアップする必要があります。

## 2.3. 操作のトラブルシューティング

本セクションでは、OpenShift で表示するために 3scale 監査ロギングを設定する方法と、OpenShift で 3scale ログおよびジョブキューにアクセスする方法を説明します。

### 2.3.1. OpenShift での 3scale 監査ロギングの設定

この設定により、すべてのログが 1箇所に集約され、Elasticsearch、Fluentd、および Kibana (EFK) ロギングツールでクエリーすることができます。これらのツールにより、3scale の設定にいつ誰がどのような変更を加えたかについての可視性が向上します。たとえば、これには、請求、アプリケーションプラン、API 設定などへの変更が含まれます。

#### 前提条件

- 3scale 2.13 デプロイメント

#### 手順

すべてのアプリケーションログを標準の OpenShift Pod ログに転送するように、監査ロギングを **stdout** に設定します。

#### 留意事項

- 3scale がオンプレミスでデプロイされる場合、デフォルトでは **stdout** への監査ログの送付は無効です。この機能が完全に動作するように設定する必要があります。
- ホスト型 3scale では、**stdout** への監査ログの送付を利用することはできません。

### 2.3.2. 監査ロギングの有効化

3scale は、**features.yml** 設定ファイルを使用して、一部のグローバル機能を有効にします。**stdout** への監査ログの送付を有効化するには、このファイルを **ConfigMap** からマウントして、デフォルトのファイルと置き換える必要があります。**features.yml** に依存する OpenShift Pod は、**system-app** および **system-sidekiq** です。

#### 前提条件

- 3scale プロジェクトの管理者アクセス権。

#### 手順

1. 以下のコマンドを入力して、**stdout** への監査ログの送付を有効にします。

```
oc patch configmap system -p '{"data": {"features.yml": "features: &default\n logging:\n audits_to_stdout: true\n\nproduction:\n <<: *default\n"}}'
```

2. 以下の環境変数をエクスポートします。

```
export PATCH_SYSTEM_VOLUMES='{"spec":{"template":{"spec":{"volumes":[{"emptyDir":{"medium":"Memory"},"name":"system-tmp"}],"configMap":{"items":[{"key":"zync.yml","path":"zync.yml"}, {"key":"rolling_updates.yml","path":"rolling_updates.yml"}, {"key":"service_discovery.yml","path":"service_discovery.yml"}, {"key":"features.yml","path":"features.yml"}]}},"name":"system"},"name":"system-config"}}'
```

3. 以下のコマンドを入力して、更新されたデプロイメント設定を関連する OpenShift Pod に適用します。

```
oc patch dc system-app -p $PATCH_SYSTEM_VOLUMES
oc patch dc system-sidekiq -p $PATCH_SYSTEM_VOLUMES
```

### 2.3.3. Red Hat OpenShift のロギングの設定

監査ログの送付を有効にして 3scale アプリケーションログが OpenShift に転送されるようになったら、ロギングツールを使用して 3scale アプリケーションを監視することができます。

Red Hat OpenShift でのログ記録の設定の詳細は、以下を参照してください。

- [Red Hat のロギングサブシステムを理解する](#)

### 2.3.4. ログへのアクセス

各コンポーネントのデプロイメント設定には、アクセスと例外のログが含まれます。デプロイメントで問題が発生した場合には、これらのログで詳細を確認してください。

3scale のログにアクセスするには、以下の手順に従います。

#### 手順

1. ログを必要とする Pod の ID を確認します。

```
oc get pods
```

2. **oc logs** と選択した Pod の ID を入力します。

```
oc logs <pod>
```

システム Pod にはコンテナが 2 つあり、それぞれに別個のログがあります。コンテナのログにアクセスするには、**--container** パラメーターで **system-provider** と **system-developer** Pod を指定します。

```
oc logs <pod> --container=system-provider
oc logs <pod> --container=system-developer
```

### 2.3.5. ジョブキューの確認

ジョブキューには、**system-sidekiq** Pod から送られる情報のログが含まれます。これらのログを使用して、クラスターがデータを処理しているかどうかを確認します。OpenShift CLI を使用してログを照会することができます。

```
oc get jobs
```

```
oc logs <job>
```

### 2.3.6. 単調増加の防止

単調増加を防止するために、3scale はデフォルトで以下のテーブルの自動ページをスケジュールします。

- **user\_sessions**: クリーンアップは週 1 回トリガーされ、2 週間より前のレコードを削除します。

- **audits**: クリーンアップは1日1回トリガーされ、3 カ月より前のレコードを削除します。
- **log\_entries**: クリーンアップは1日1回トリガーされ、6 カ月より前のレコードを削除します。
- **event\_store\_events**: クリーンアップは週1回トリガーされ、1週間より前のレコードを削除します。

以下の表は例外で、データベース管理者が手動でパージする必要があります。

- **alerts**

表2.1 SQL パージコマンド

データベースタイプ	SQL コマンド
MySQL	<code>DELETE FROM alerts WHERE timestamp &lt; NOW() - INTERVAL 14 DAY;</code>
PostgreSQL	<code>DELETE FROM alerts WHERE timestamp &lt; NOW() - INTERVAL '14 day';</code>
Oracle	<code>DELETE FROM alerts WHERE timestamp &lt;= TRUNC(SYSDATE) - 14;</code>

## 関連情報

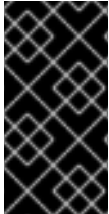
- Openshift Container Platform (OCP) についての詳細は、[OCP のドキュメント](#) を参照してください。
- [Pod の自動スケーリング](#)
- [コンピュータノードの追加](#)
- [ルーティングの最適化](#)



## 第3章 3SCALE のモニタリング

**Prometheus** は、履歴データを保存し、大型でスケーラブルなシステムを監視するために構築されたコンテナネイティブなソフトウェアです。現在実行中のセッションだけでなく、長時間にわたってデータを収集します。Prometheus のアラートルールは、Alertmanager によって管理されます。

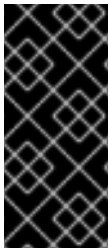
Prometheus および Alertmanager を使用して、3scale データ監視および保存します。これにより、**Grafana** などのグラフィカルツールを使用して、データを視覚化し、クエリーを実行することができます。



### 重要

Prometheus はオープンソースのシステム監視ツールキットで、Grafana はオープンソースのダッシュボードツールキットです。Prometheus および Grafana に対する Red Hat のサポートは、Red Hat の製品ドキュメントに記載されている推奨設定に限定されません。

3scale operator では、既存の Prometheus および Grafana operator のインストールを使用して、3scale の使用状況およびリソースを監視することができます。

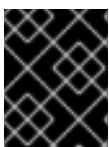


### 重要

- 3scale operator は監視リソースを作成しますが、これらのリソースの変更は妨げません。
- 3scale Operator と Prometheus Operator を同じ namespace にインストールするか、クラスター全体の Operator を使用する必要があります。

### 前提条件

- **3scale operator** がインストールされている。
- **Prometheus operator** がクラスターにインストールされている。Prometheus operator は、Prometheus インスタンスを作成および管理します。3scale の監視に必要な **Prometheus** カスタムリソース定義を提供します。  
以下の Prometheus operator およびイメージバージョンは、3scale でテストされています。
  - Prometheus Operator **v0.37.0**
  - Prometheus イメージ: **quay.io/prometheus/prometheus:v2.16.0**
- **Grafana operator** がクラスターにインストールされている。Grafana operator は、Grafana インスタンスを作成および管理します。これは、3scale の監視に必要な **GrafanaDashboard** カスタムリソース定義を提供します。  
以下の Grafana operator およびイメージバージョンは、3scale でテストされています。
  - Grafana operator **v3.9.0**
  - Grafana イメージ: **registry.hub.docker.com/grafana/grafana:7.1.1**



### 重要

クラスターがインターネット上で公開される場合は、必ず Prometheus サービスおよび Grafana サービスを保護するようにしてください。

本セクションでは、Grafana ダッシュボードを表示できるように、3scale インスタンスのモニタリングを有効にする方法を説明します。

- [3scale のモニタリングの有効化](#)
- [3scale を監視するための Prometheus の設定](#)
- [3scale を監視するための Grafana の設定](#)
- [3scale のメトリクスの表示](#)

### 3.1. 3SCALE のモニタリングの有効化

3scale を監視するには、APIManager カスタムリソースを設定して監視を有効にする必要があります。

#### 手順

1. 3scale を設定し、3scale デプロイメント YAML の **spec.monitoring.enabled** パラメーターを **true** に設定して監視を有効にします。以下に例を示します。
  - a. **3scale-monitoring.yml** という名前の APIManager カスタムリソースを作成し、監視を有効にします。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager1
spec:
  wildcardDomain: example.com
  monitoring:
    enabled: true
    enablePrometheusRules: false 1
```

- 1** オプションで **PrometheusRules** を無効にできます。それ以外の場合は、デフォルトで有効になります。

- b. OpenShift クラスターにログインします。3scale の OpenShift プロジェクトのクラスター **編集** ロールを持つユーザーとしてログインする必要があります (例: **cluster-admin**)。

```
oc login
```

- c. 3scale プロジェクトに切り替えます。

```
oc project <project_name>
```

- d. カスタムリソースをデプロイします。

```
$ oc apply -f 3scale-monitoring.yml
```

#### 関連情報

- [operator を使用した OpenShift への 3scale のデプロイメント設定オプション](#)

- [3scale PrometheusRules](#)

## 3.2. 3SCALE を監視するための PROMETHEUS の設定

3scale のモニタリングを有効にするには、**Prometheus** カスタムリソースを使用して Prometheus をデプロイおよび設定する必要があります。



### 注記

[Prometheus のドキュメント](#) の説明に従ってパーミッションが正しく設定されていることを確認してください。

### 手順

1. クラスター内のすべてのリソースを監視するか、3scale リソースのみを監視するかに応じて、以下のように Prometheus カスタムリソースをデプロイします。
  - クラスターのすべてのリソースを監視するには、**spec.podMonitorSelector** 属性を {} に、**spec.ruleSelector** 属性を {} に設定します。たとえば、以下のカスタムリソースを適用します。

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: example
spec:
  podMonitorSelector: {}
  ruleSelector: {}
```

- 3scale と Prometheus operator を同じ OpenShift プロジェクトにデプロイし、**APP\_LABEL** の値がデフォルトの **3scale-api-management** に設定されている場合は、以下の手順に従って 3scale リソースを監視します。
  - a. **spec.podMonitorSelector** 属性を以下のように設定します。

```
podMonitorSelector:
  matchExpressions:
    - key: app
      operator: In
      values:
        - 3scale-api-management
```

- b. **spec.ruleSelector** 属性を以下のように設定します。

```
matchExpressions:
  - key: app
    operator: In
    values:
      - 3scale-api-management
```

たとえば、以下のカスタムリソースを適用します。

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
```

```

metadata:
  name: example
spec:
  podMonitorSelector:
    matchExpressions:
      - key: app
        operator: In
        values:
          - 3scale-api-management
  ruleSelector:
    matchExpressions:
      - key: app
        operator: In
        values:
          - 3scale-api-management

```

- 3scale と Prometheus operator を別の OpenShift プロジェクトにデプロイした場合には、以下の手順に従って 3scale リソースを監視します。
  - a. 3scale がデプロイされている OpenShift プロジェクトに **MYLABELKEY=MYLABELVALUE** のラベルを付けます。
  - b. **podMonitorNamespaceSelector** フィルターを使用して 3scale Pod を選択します。たとえば、以下のカスタムリソースを適用します。

```

apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: example
spec:
  podMonitorSelector: {}
  ruleSelector: {}
  podMonitorNamespaceSelector:
    matchExpressions:
      - key: MYLABELKEY
        operator: In
        values:
          - MYLABELVALUE

```

2. ダッシュボードおよびアラートが予想通りに機能させるには、以下のいずれかを実行して Kubernetes メトリック ([kube-state-metrics](#)) を取り込む必要があります。
  - Prometheus インスタンスをクラスターのデフォルト Prometheus インスタンスでフェデレーションする。
  - 独自の収集ジョブを設定し、kubelet、etcd、その他からメトリックを取得する。

## 関連情報

- Prometheus の詳細は、[Prometheus のドキュメント](#) を参照してください。

## 3.3. 3SCALE を監視するための GRAFANA の設定

3scale の監視を有効にするには、Grafana を設定する必要があります。

## 手順

1. **app=3scale-api-management** ラベルを上書きして、**GrafanaDashboards** リソースを監視するように Grafana サービスが設定されていることを確認してください。たとえば、以下のカスタムリソースを適用します。

```
apiVersion: integreatly.org/v1alpha1
kind: Grafana
metadata:
  name: grafana
spec:
  dashboardLabelSelector:
  - matchExpressions:
    - key: app
      operator: In
      values:
        - 3scale-api-management
```

3scale operator によって作成された Grafana ダッシュボードには、以下のようにラベルが付けられます。

```
app: 3scale-api-management
monitoring-key: middleware
```

2. Grafana operator が 3scale 以外の namespace にインストールされている場合は、**--namespaces** または **--scan-all** operator フラグを使用して、namespace 外部のリソースを監視するように設定します。Operator フラグについての詳細は、[Grafana のドキュメント](#) を参照してください。
3. タイプが **prometheus** の **GrafanaDataSource** カスタムリソースを作成し、Grafana で Prometheus データを使用できるようにします。以下に例を示します。

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDataSource
metadata:
  name: prometheus
spec:
  name: middleware
  datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus-operated:9090
    isDefault: true
    version: 1
    editable: true
  jsonData:
    timeInterval: "5s"
```

<http://prometheus-operated:9090> は Prometheus のルートです。

4. [Grafana のドキュメント](#) の説明に従ってパーミッションが正しく設定されていることを確認してください。

## 関連情報

- Grafana の詳細は、[Grafana のドキュメント](#) を参照してください。

### 3.4. 3SCALE のメトリックの表示

3scale、Prometheus、および Grafana の設定後に、本セクションで説明するメトリックを表示できます。

#### 手順

1. Grafana コンソールにログインします。
2. 以下についてのメトリックを表示できることを確認します。
  - 3scale がインストールされている Pod および namespace レベルでの Kubernetes リソース
  - ステージング環境用 APIcast
  - 実稼働環境用 APIcast
  - バックエンドワーカー
  - バックエンドリスナー
  - System
  - Zync

### 3.5. PROMETHEUS に公開される 3SCALE システムメトリック

以下のポートを、Prometheus エンドポイントと共に 3scale システム Pod を使用してメトリックを公開するように設定できます。

表3.1 3scale システムポート

system-app	ポート
system-developer	9394
system-master	9395
system-provider	9396

system-sidekiq	ポート
system-sidekiq	9394

エンドポイントは、以下を使用して内部でのみアクセスできます。

```
http://${service}:${port}/metrics
```

以下に例を示します。

http://system-developer:9394/metrics

### 関連情報

- APIcast のモニタリングに関する情報は、[Prometheus への 3scale APIcast メトリクスの公開](#)を参照してください。
- Prometheus のセキュリティー保護に関する詳細は、[Prometheus のセキュリティー](#)に関するドキュメントを参照してください。
- Grafana のセキュリティー保護に関する詳細は、Grafana のドキュメントで [パーミッション](#) および [セキュリティー](#)を確認してください。

## 第4章 WEBHOOK を使用した 3SCALE 自動化

Webhook は自動化を容易にする機能であり、3scale で発生したイベントに基づいて他のシステムを統合するのにも使用されます。3scale システムで指定のイベントが発生すると、Webhook メッセージを使用してアプリケーションに通知が送信されます。たとえば、Webhook を設定して、新規アカウントのサインアップからのデータでデベロッパーポータルに反映することができます。

### 4.1. WEBHOOK の概要

Webhook は、**Webhook** 設定ウィンドウで利用可能なイベントから選択されたイベントによってトリガーされるカスタム HTTP コールバックです。これらのイベントのいずれかが発生すると、3scale システムは、Webhook セクションで指定した URL アドレスに対して HTTP または HTTPS リクエストを行います。Webhook では、リスナーを設定してイベント追跡などの目的の動作を呼び出すことができます。

Webhook のフォーマットは常に同じです。以下の構造の XML ドキュメントでエンドポイントにポストします。

```
<?xml version="1.0" encoding="UTF-8"?>
<event>
  <type>application</type>
  <action>updated</action>
  <object>
    THE APPLICATION OBJECT AS WOULD BE RETURNED BY A GET ON THE ACCOUNT
    MANAGEMENT
    API
  </object>
</event>
```

各要素では、以下の情報を提供します。

- **<type>**: `application`、`account` など、イベントの主体を表します。
- **<action>**: `updated`、`created`、`deleted` などの値を使用してアクションを指定します。
- **<object>**: Account Management API によって返される、同じフォーマットの XML オブジェクト自体です。インタラクティブな ActiveDocs を使用して確認できます。

Webhook が 3scale によって実行されたことを保証する必要がある場合は、HTTPS Webhook URL を公開し、3scale の Webhook 宣言にカスタムパラメーターを追加します。たとえば、<https://your-webhook-endpoint?someSecretParameterName=someSecretParameterValue> となります。パラメーター名と値を指定します。続いて、Webhook エンドポイント内で、このパラメーター値があることを確認します。

### 4.2. WEBHOOK の設定

#### 手順

1. **Dashboard** メニューから **Account Settings** を選択し、**Integrate > Webhooks** に移動します。
2. Webhook の動作を指定します。以下の 2 つのオプションがあります。
  - **Webhooks enabled**: Webhook を有効または無効にするには、このチェックボックスを選択します。



- **Actions in the admin portal also trigger webhooks** イベント発生時に Webhooks をトリガーするには、このチェックボックスを選択します。以下の点を考慮してください。
  - トリガーとなるイベントが設定された内部 3scale API への呼び出しを行う場合は、プロバイダーキーではなくアクセストークンを使用します。
  - このチェックボックスを選択しないままにすると、デベロッパーポータルアクションだけが Webhook のトリガーになります。
- 3. 選択したイベントがトリガーとなった際にイベントを通知するための URL アドレスを指定します。
- 4. 指定した URL アドレスへのコールバックのトリガーとなるイベントを選択します。

設定が完了したら、**Update webhooks settings** をクリックして変更を保存します。

### 4.3. WEBHOOK のトラブルシューティング

リッスンしているエンドポイントに障害が発生した場合、失敗した配信を回復できます。エンドポイントがコード **200** を返す場合に、3scale は Webhook が配信されたとみなします。そうでない場合は、60 秒間隔で 5 回リトライします。障害からの復旧後に、または定期的にチェックを実行し、必要に応じキューをクリーンアップする必要があります。ActiveDocs では、以下のメソッドの詳細情報を確認できます。

- 配信に失敗した Webhook のリスト
- 配信に失敗した Webhook の削除

## 第5章 3SCALE TOOLBOX

3scale toolbox は、コマンドラインから 3scale 製品を管理することのできる Ruby クライアントです。

3scale のドキュメントには、3scale toolbox のインストール、サポートされる toolbox コマンド、サービス、プラン、SSL および TLS に関する問題のトラブルシューティングなどについての情報が掲載されています。詳細は、以下のいずれかのセクションを参照してください。

- [toolbox のインストール](#)
- [サポートされる toolbox コマンド](#)
- [サービスのインポート](#)
- [サービスのコピー](#)
- [サービス設定のみのコピー](#)
- [OpenAPI 認証](#)
- [OpenAPI 定義のインポート](#)
- [OpenAPI 定義からの 3scale バックエンドのインポート](#)
- [リモートアクセスクレデンシャルの管理](#)
- [アプリケーションプランの作成](#)
- [メトリクスの作成](#)
- [メソッドの作成](#)
- [サービスの作成](#)
- [ActiveDocs の作成](#)
- [プロキシ設定の一覧表示](#)
- [ポリシーレジストリーのコピー](#)
- [アプリケーションの一覧表示](#)
- [プロダクトのエクスポート](#)
- [プロダクトのインポート](#)
- [プロダクトポリシーチェーンのエクスポートおよびインポート](#)
- [API バックエンドのコピー](#)
- [SSL および TLS に関する問題のトラブルシューティング](#)

### 5.1. TOOLBOX のインストール

公式にサポートされている 3scale toolbox のインストール方法は、3scale toolbox のコンテナイメージを使用するものです。

### 5.1.1. toolbox コンテナイメージのインストール

本セクションでは、toolbox コンテナイメージをインストールする方法について説明します。

#### 前提条件

- [Red Hat Ecosystem Catalog の 3scale toolbox イメージ](#) を参照する。
- Red Hat レジストリーサービスアカウントがある。
- このトピックの例では、Podman がインストールされていることを前提としている。

#### 手順

1. Red Hat Ecosystem Catalog にログインします。

```
$ podman login registry.redhat.io
Username: ${REGISTRY-SERVICE-ACCOUNT-USERNAME}
Password: ${REGISTRY-SERVICE-ACCOUNT-PASSWORD}
Login Succeeded!
```

2. toolbox のコンテナイメージをプルします。

```
$ podman pull registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13
```

3. インストールを確認します。

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale help
```

#### 関連情報

- OpenShift、Podman、または Docker を使用してツールボックスイメージをインストールする方法の詳細は、[Red Hat Ecosystem Catalog でイメージを取得する手順](#) を参照してください。
- [Kubernetes での 3scale toolbox インストール手順](#) も参照してください。OpenShift では、**kubectrl** ではなく正しいイメージ名と **oc** コマンドを使用する必要があります。

## 5.2. サポートされる TOOLBOX コマンド

3scale toolbox を使用して、コマンドラインツール (CLI) から API を管理します。



#### 注記

**update** コマンドは削除され、**copy** コマンドに置き換えられました。

サポートされるコマンドは以下のとおりです。

#### COMMANDS

account	account super command
activedocs	activedocs super command
application	application super command
application-plan	application-plan super command
backend	backend super command

copy	copy super command
help	print help
import	import super command
method	method super command
metric	metric super command
policy-registry	policy-registry super command
product	product super command
proxy-config	proxy-config super command
remote	remotes super command
service	services super command

#### OPTIONS

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

### 5.3. サービスのインポート

以下のフィールドをこの順序で指定して、CSV ファイルからサービスをインポートします。以下のヘッダーを CSV ファイルに追加します。

```
service_name,endpoint_name,endpoint_http_method,endpoint_path,auth_mode,endpoint_system_name,type
```

以下の情報が必要です。

- 3scale 管理アカウント: **{3SCALE\_ADMIN}**
- 3scale インスタンスが実行されているドメイン: **{DOMAIN\_NAME}**
  - Hosted APICast を使用している場合、ドメインは 3scale.net です。
- アカウントのアクセスキー: **{ACCESS\_KEY}**
- サービスの CSV ファイル (例: **example/import\_example.csv**)

以下のコマンドを実行してサービスをインポートします。

#### 例

```
$ podman run -v $PWD/examples/import_example.csv:/tmp/import_example.csv
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale import csv --
destination=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
file=/tmp/import_example.csv
```

この例では、Podman ボリュームを使用して、リソースファイルをコンテナにマウントします。これは、このファイルが現在の **\$PWD** フォルダにあることを前提としています。

### 5.4. サービスのコピー

同じアカウントまたは別のアカウントから、既存のサービスをベースにして新しいサービスを作成します。サービスをコピーすると、関連する ActiveDocs もコピーされます。

以下の情報が必要です。

- コピーするサービスの ID: **{SERVICE\_ID}**
- 3scale 管理アカウント: **{3SCALE\_ADMIN}**
- 3scale インスタンスが実行されているドメイン: **{DOMAIN\_NAME}**
  - Hosted APICast を使用している場合、ドメインは 3scale.net です。
- アカウントのアクセスキー: **{ACCESS\_KEY}**
- 別のアカウントにコピーする場合は、コピー先アカウントのアクセスキー: **{DEST\_KEY}**
- 新しいサービスの名前: **{NEW\_NAME}**

## 例

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale copy service
{SERVICE_ID} --source=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
destination=https://{DEST_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
target_system_name={NEW_NAME}
```



### 注記

コピーするサービスにカスタムポリシーがある場合、それぞれのカスタムポリシー定義がサービスのコピー先にすでに存在することを確認してください。カスタムポリシー定義のコピーについては、[ポリシーレジストリーのコピー](#)を確認してください。

## 5.5. サービス設定のみのコピー

あるサービスから別の既存サービスに、サービスおよびプロキシ設定、メトリック、メソッド、アプリケーションプラン、アプリケーションプランの制限と共にマッピングルールを一括コピーすることができます。

以下の情報が必要です。

- コピーするサービスの ID: **{SERVICE\_ID}**
- コピー先のサービスの ID: **{DEST\_ID}**
- 3scale 管理アカウント: **{3SCALE\_ADMIN}**
- 3scale インスタンスが実行されているドメイン: **{DOMAIN\_NAME}**
  - Hosted APICast を使用している場合、ドメインは 3scale.net です。
- アカウントのアクセスキー: **{ACCESS\_KEY}**
- コピー先アカウントのアクセスキー: **{DEST\_KEY}**

また、オプションのフラグを使用できます。

- **-f** フラグ: コピーする前に既存の対象サービスのマッピングルールを削除します。
- **-r** フラグ: 対象サービスにマッピングルールのみをコピーします。



### 注記

**update** コマンドは削除され、**copy** コマンドに置き換えられました。

以下のコマンド例により、あるサービスから別の既存サービスに一括コピーが行われます。

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale copy [opts] service --
source=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
destination=https://{DEST_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} {SERVICE_ID}
{DEST_ID}
```

## 5.6. OPENAPI 認証

3scale ツールボックスを使用して OpenAPI 認証を実装すると、承認されたユーザーのみが API にアクセスできるようにして、機密データを保護し、API の使用を効率的に管理できます。このアプローチにより、API インフラストラクチャーが強化され、開発者とコンシューマー間の信頼関係が構築されます。



### 注記

最上位のセキュリティー要件は1つだけサポートされています。操作レベルのセキュリティー要件はサポートされていません。

サポートされているセキュリティースキーム: 任意のフロータイプの **apiKey** および **oauth2**。

apiKey セキュリティースキームタイプの場合は、以下のようになります。

- 認証情報の場所は、セキュリティースキームオブジェクトのフィールドで OpenAPI から読み取られます。
- 認証ユーザーキーは、セキュリティースキームオブジェクトの OpenAPI 名フィールドから読み取られます。

**apiKey** セキュリティー要件を備えた OpenAPI 3.0.2 の部分的な例:

```
openapi: "3.0.2"
security:
  - petstore_api_key: []
components:
  securitySchemes:
    petstore_api_key:
      type: apiKey
      name: api_key
      in: header
  ...
```

**oauth2** セキュリティースキームタイプの場合は、以下のようになります。

- 認証情報の場所は **headers** にハードコーディングされています。
- OpenID Connect Issuer Type のデフォルトは **rest** です。 **--oidc-issuer-type=<value>** コマンドオプションを使用してこれをオーバーライドできます。
- OpenID Connect Issuer は OpenAPI から読み取られません。3scale では発行者の URL にクライアントシークレットが含まれている必要があるため、この **--oidc-issuer-endpoint=<value>** コマンドオプションを使用して発行を設定する必要があります。
- OIDC AUTHORIZATION FLOW は、セキュリティスキームオブジェクトの flows フィールドから読み取られます。

### oauth2 セキュリティー要件を備えた OpenAPI 3.0.2 の部分的な例:

```
openapi: "3.0.2"
security:
  - petstore_oauth:
    - write:pets
    - read:pets
components:
  securitySchemes:
    petstore_oauth:
      type: oauth2
      flows:
        clientCredentials:
          tokenUrl: http://example.org/api/oauth/dialog
          scopes:
            write:pets: modify pets in your account
            read:pets: read your pets
      ...
```

OpenAPI がセキュリティ要件を指定していない場合:

- この製品は **Open API** とみなされます。
- **default\_credentials** 3scale ポリシーが追加されました。注記: これは **anonymous\_policy** とも呼ばれます。
- **--default-credentials-userkey** コマンドが必要です。注記: 指定されていない場合、コマンドは失敗します。

### 関連情報

- [Security Scheme Object](#)

## 5.7. OPENAPI 定義のインポート

新しいサービスを作成する場合、または既存のサービスを更新する場合は、ローカルファイルまたは URL から OpenAPI 定義をインポートすることができます。インポートのデフォルトのサービス名は、OpenAPI 定義の **info.title** で指定されます。ただし、**--target\_system\_name=<NEW NAME>** を使用して、このサービス名を上書きできます。この場合、そのサービス名がすでに存在する場合は更新され、存在しない場合は新しいサービス名が作成されます。

**import openapi** コマンドのフォーマットは以下のとおりです。

```
$ 3scale import openapi [opts] -d=<destination> <specification>
```

OpenAPI <specification> は、以下のいずれかです。

- `/path/to/your/definition/file.[json|yaml|yml]`
- `http[s]://domain/resource/path.[json|yaml|yml]`

## 例

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale import openapi [opts] -d=https://{DEST_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} my-test-api.json
```

## コマンドオプション

`import openapi` コマンドのオプションは以下のとおりです。

**-d --destination=<value>**

3scale のターゲットインスタンス (`http[s]://<authentication>@3scale_domain` 形式)

**-t --target\_system\_name=<value>**

3scale のターゲットシステム名

**--backend-api-secret-token=<value>**

API ゲートウェイによってバックエンド API に送信されるカスタムシークレットトークン

**--backend-api-host-header=<value>**

API ゲートウェイによってバックエンド API に送信されるカスタムホストヘッダー

その他のオプションについては、`3scale import openapi --help` コマンドを参照してください。

## OpenAPI のインポートルール

サポートされるセキュリティスキームは、**apiKey** および **oauth2** (任意の OAuth フロータイプに対応) です。

OpenAPI 仕様は、以下の値のいずれかでなければなりません。

- 利用可能なパスのファイル名
- `toolbox` がコンテンツをダウンロードすることのできる URL。サポートされるスキームは **http** および **https** です。
- **stdin** 標準入力ストリームから読み込む。これは、値に `-` を設定することで制御されます。

OpenAPI 定義をインポートする場合、以下の追加ルールが適用されます。

- 定義は OpenAPI 2.0 または OpenAPI 3.0 として検証される。
- OpenAPI 定義のすべてのマッピングルールがインポートされる。これらについては [API > Integration](#) で確認できます。
- 3scale プロダクトのすべてのマッピングルールは置き換えられる。
- OpenAPI 定義に含まれるメソッドのみが変更される。



- OpenAPI 定義にしか存在していなかったすべてのメソッドが、**Hits** メトリクスにアタッチされる。
- メソッドを置き換えるには、パターンの完全一致の使用により、メソッドの名前が OpenAPI 定義 **operation.operationId** で定義されるメソッドと同一でなければならない。



### 注記

ポリシーチェーンに **default\_credentials** ポリシー (**anonymous\_policy** と呼ばれる) がまだない場合、toolbox はこのポリシーを追加します。**default\_credentials** ポリシーは、オプションのパラメーター **--default-credentials-userkey** で提供される **ユーザーキー** で設定されます。

OpenAPI 3.0 は、セキュリティスキームとセキュリティ要件機能を使用して、API のセキュリティを指定する方法を提供します。詳細は、[Swagger Authentication and Authorization](#) の公式ドキュメントを参照してください。

## OpenAPI 3.0 の制約

OpenAPI 3.0 定義をインポートする場合、以下の制約が適用されます。

- **servers** リストの最初の **server.url** 要素だけがプライベート URL として処理される。**server.url** 要素の **path** コンポーネントが、OpenAPI の **basePath** プロパティとして使用されます。
- toolbox は、パス項目および操作オブジェクトのサーバーを処理しない。
- セキュリティスキームオブジェクトでは、複数のフローはサポートされない。

## 5.8. OPENAPI 定義からの 3SCALE バックエンドのインポート

toolbox **import** コマンドを使用して OpenAPI 定義をインポートし、3scale バックエンド API を作成できます。コマンドラインオプション **--backend** は、この機能を有効にします。3scale は OpenAPI 定義を使用して、バックエンドとそのプライベートベース URL と、そのマッピングルールおよびメソッドを作成し、保存します。

### 前提条件

- オンプレミス型 3scale 2.13 インスタンスの管理者権限を持つユーザーアカウント
- API を定義する OAS ドキュメント

### 手順

- 以下の形式を使用し、**import** コマンドを実行してバックエンドを作成します。

```
$ 3scale import openapi -d <remote> --backend <OAS>
```

- **<remote>** は、バックエンドを作成する 3scale インスタンスの URL に置き換えます。 **http[s]://<authentication>@3scale\_domain** の形式を使用します。
- **<OAS>** は **/path/to/your/oasdoc.yaml** に置き換えます。

表5.1 追加の OpenAPI 定義オプション

オプション	説明
<b>-o --output=&lt;value&gt;</b>	出力フォーマット。JSON または YAML のいずれかを使用できます。
<b>--override-private-base-url=&lt;value&gt;</b>	3scale は OpenAPI 定義の <b>servers[0].url</b> フィールドからバックエンドのプライベートエンドポイントを読み取ります。このフィールドの設定をオーバーライドするには、このオプションを指定し、 <b>&lt;value&gt;</b> は選択したプライベートベース URL に置き換えます。OpenAPI 定義が <b>servers[0].url</b> フィールドに値を指定しておらず、 <b>import</b> コマンドでこのオプションを指定しないと、実行に失敗します。
<b>--prefix-matching</b>	OpenAPI 操作から派生するマッピングルールに厳密なマッチングではなく、接頭辞のマッチングを使用します。
<b>--skip-openapi-validation</b>	OpenAPI スキーマ検証を省略します。
<b>-t --target_system_name=&lt;value&gt;</b>	ターゲットシステム名はテナントの一意のキーです。システム名は OpenAPI 定義から推測できますが、このパラメーターを使用して独自の名前に上書きできます。

## 5.9. リモートアクセスクレデンシャルの管理

リモートの 3scale インスタンスと容易に連携するため、3scale toolbox を使用して、設定ファイルでリモート URL アドレスおよびこれらのリモートインスタンスにアクセスするための認証情報を定義することができます。その後、toolbox コマンドでは短縮名を使用してこれらのリモートを参照することができます。

設定ファイルのデフォルトの場所は **\$HOME/.3scalerc.yaml** です。ただ

し、**THREESCALE\_CLI\_CONFIG** 環境変数または **--config-file <config\_file>** toolbox オプションを使用して、別の場所を指定することができます。

リモートアクセスクレデンシャルを追加する場合、**access\_token** または **provider\_key** を指定することができます。

- **http[s]://<access\_token>@<3scale-instance-domain>**
- **http[s]://<provider\_key>@<3scale-instance-domain>**

### 5.9.1. リモートアクセスクレデンシャルの追加

以下のコマンド例により、短縮名 **<name>** のリモート 3scale インスタンスが **<url>** に追加されます。

```
3scale remote add [--config-file <config_file>] <name> <url>
```

例

■

```
$ podman run --name toolbox-container registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13
3scale remote add instance_a https://123456789@example_a.net

$ podman commit toolbox-container toolbox
```

上記の例では、リモートインスタンスを作成し、コンテナをコミットして新しいイメージを作成します。次に、含まれるリモート情報を使用して新しいイメージを実行することができます。たとえば、以下のコマンドでは、新しいイメージを使用して新たに追加されたリモートを表示します。

```
$ podman run toolbox 3scale remote list
instance_a https://example_a.net 123456789
```

続いて、他の toolbox コマンドは、新たに作成されたイメージを使用して、追加されたリモートにアクセスすることができます。以下の例では、**registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13**ではなく **toolbox** という名前のイメージが使用されています。



### 警告

toolbox のシークレットをコンテナに保存することは、シークレットと共にコンテナを他のユーザーに提供する場合や、自動化のためにコンテナを使用する場合など、セキュリティ上のリスクとなる可能性があります。Podman のセキュアなボリューム、または OpenShift のシークレットを使用してください。

## 関連情報

Podman の使用についての詳細は、以下のドキュメントを参照してください。

- [Red Hat Enterprise Linux 8 コンテナの構築、実行、および管理](#)

### 5.9.2. リモートアクセスクレデンシャルのリスト表示

以下のコマンド例は、リモートアクセスクレデンシャルを一覧表示する方法を示しています。

```
3scale remote list [--config-file <config_file>]
```

このコマンドにより、追加されたリモート 3scale インスタンスのリストが **<name> <URL> <authentication-key>** のフォーマットで表示されます。

#### 例

```
$ podman run <toolbox_image_with_remotes_added> 3scale remote list
instance_a https://example_a.net 123456789
instance_b https://example_b.net 987654321
```

### 5.9.3. リモートアクセスクレデンシャルの削除

以下のコマンド例は、リモートアクセスクレデンシャルを削除する方法を示しています。

```
3scale remote remove [--config-file <config_file>] <name>
```

このコマンドにより、短縮名 **<name>** のリモート 3scale インスタンスが削除されます。

#### 例

```
$ podman run <toolbox_image_with_remote_added> 3scale remote remove instance_a
```

### 5.9.4. リモートアクセスクレデンシャルの名前変更

以下のコマンド例は、リモートアクセスクレデンシャルの名前を変更する方法を示しています。

```
3scale remote rename [--config-file <config_file>] <old_name> <new_name>
```

このコマンドにより、短縮名 **<old\_name>** のリモート 3scale インスタンスの名前が **<new\_name>** に変更されます。

#### 例

```
$ podman run <toolbox_image_with_remote_added> 3scale remote rename instance_a instance_b
```

## 5.10. アプリケーションプランの作成

3scale toolbox を使用して、デベロッパーポータルアプリケーションプランの作成、更新、リスト表示、削除、表示、またはエクスポート/インポートを行います。

### 5.10.1. 新しいアプリケーションプランの作成

新しいアプリケーションプランを作成するには、以下の手順に従います。

- アプリケーションプラン名を指定する必要があります。
- **system-name** を上書きするには、オプションのパラメーターを使用します。
- 同じ名前のアプリケーションプランがすでに存在する場合、エラーメッセージが表示されません。
- **--default** フラグを使用して、アプリケーションプランを **デフォルト** として設定します。
- **--publish** フラグを使用して、**公開済み** アプリケーションプランを作成します。
  - デフォルトでは、**非表示** になります。
- **--disabled** フラグを使用して、**無効な** アプリケーションプランを作成します。
  - デフォルトでは、**有効** になります。



#### 注記

- **service** 位置引数はサービスの参照で、サービスの **id** またはサービスの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。

以下のコマンドにより、新しいアプリケーションプランが作成されます。

```
3scale application-plan create [opts] <remote> <service> <plan-name>
```

アプリケーションプランの作成時に、以下のオプションを使用します。

#### Options

```
--approval-required=<value>  The application requires approval:
                             true or false
--cost-per-month=<value>      Cost per month
--default                     Make the default application plan
--disabled                    Disable all methods and metrics in
                             the application plan
-o --output=<value>           Output format on stdout:
                             one of json|yaml
-p --published                Publish the application plan
--setup-fee=<value>           Set-up fee
-t --system-name=<value>      Set application plan system name
--trial-period-days=<value>   The trial period in days
```

#### Options for application-plan

```
-c --config-file=<value>      3scale toolbox configuration file:
                             defaults to $HOME/.3scalerc.yaml
-h --help                     Print help for this command
-k --insecure                 Proceed and operate even for server
                             connections otherwise considered
                             insecure
-v --version                  Print the version of this command
--verbose                     Verbose mode
```

### 5.10.2. アプリケーションプランの作成または更新

アプリケーションプランが存在しない場合に新しく作成する、または既存のアプリケーションプランを更新するには、以下の手順に従います。

- **--default** フラグを使用して、**デフォルト** アプリケーションプランを更新します。
- **--publish** フラグを使用して、**公開済み** アプリケーションプランを更新します。
- **--hide** フラグを使用して、**非表示の** アプリケーションを更新します。
- **--disabled** フラグを使用して、**無効な** アプリケーションプランを更新します。
- **--enabled** フラグを使用して、**有効な** アプリケーションプランを更新します。



## 注記

- **service** 位置引数はサービスの参照で、サービスの **id** またはサービスの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。
- **plan** 位置引数はプランの参照で、プランの **id** またはプランの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。

以下のコマンドにより、アプリケーションプランが更新されます。

```
3scale application-plan create [opts] <remote> <service> <plan>
```

アプリケーションプランの更新時に、以下のオプションを使用します。

### Options

```
--approval-required=<value>  The application requires approval:
                             true or false
--cost-per-month=<value>      Cost per month
--default                      Make the default application plan
--disabled                     Disable all methods and metrics in
                             the application plan
--enabled                      Enable the application plan
--hide                         Hide the application plan
-n --name=<value>             Set the plan name
-o --output=<value>           Output format on stdout:
                             one of json|yaml
-p --publish                  Publish the application plan
--setup-fee=<value>           Set-up fee
--trial-period-days=<value>   The trial period in days
```

### Options for application-plan

```
-c --config-file=<value>      3scale toolbox configuration file:
                             defaults to $HOME/.3scalerc.yaml
-h --help                    Print help for this command
-k --insecure                 Proceed and operate even for server
                             connections otherwise considered
                             insecure
-v --version                  Print the version of this command
--verbose                     Verbose mode
```

## 5.10.3. アプリケーションプランのリスト表示

以下のコマンドにより、アプリケーションプランがリスト表示されます。

```
3scale application-plan list [opts] <remote> <service>
```

アプリケーションプランのリスト表示時に、以下のオプションを使用します。

### Options

```
-o --output=<value>          Output format on stdout:
```

one of json|yaml

#### Options for application-plan

```
-c --config-file=<value> 3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                          connections otherwise considered insecure
-v --version             Print the version of this command
--verbose                Verbose mode
```

### 5.10.4. アプリケーションプランの表示

以下のコマンドにより、アプリケーションプランが表示されます。

```
3scale application-plan show [opts] <remote> <service> <plan>
```

アプリケーションプランの表示時に、以下のオプションを使用します。

#### Options

```
-o --output=<value>      Output format on stdout:
                          one of json|yaml
```

#### Options for application-plan

```
-c --config-file=<value> 3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                          connections otherwise considered insecure
-v --version             Print the version of this command
--verbose                Verbose mode
```

### 5.10.5. アプリケーションプランの削除

以下のコマンドにより、アプリケーションプランが削除されます。

```
3scale application-plan delete [opts] <remote> <service> <plan>
```

アプリケーションプランの削除時に、以下のオプションを使用します。

#### Options for application-plan

```
-c --config-file=<value> 3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                          connections otherwise considered insecure
-v --version             Print the version of this command
--verbose                Verbose mode
```

### 5.10.6. アプリケーションプランのエクスポート/インポート

単一のアプリケーションプランを **yaml** コンテンツにエクスポートすることや、コンテンツからインポートすることができます。

次の点に注意してください。

- アプリケーションプランで定義される制限が含まれます。
- アプリケーションプランで定義される課金ルールが含まれます。
- 制限および課金ルールで参照されるメトリック/メソッドが含まれます。
- アプリケーションプランで定義される機能が含まれます。
- サービスは **id** または **system\_name** で参照できます。
- アプリケーションプランは **id** または **system\_name** で参照できます。

### 5.10.6.1. ファイルへのアプリケーションプランのエクスポート

以下のコマンドにより、アプリケーションプランがエクスポートされます。

```
3scale application-plan export [opts] <remote> <service_system_name> <plan_system_name>
```

#### 例

```
$ podman run -u root -v $PWD:/tmp registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale application-plan export --file=/tmp/plan.yaml remote_name service_name plan_name
```

この例では、Podman ボリュームを使用して、エクスポートされたファイルをコンテナにマウントし、現在の **\$PWD** フォルダに出力します。

#### 注記

##### export コマンドに固有の事項

- リモートサービスおよびアプリケーションプランでは、読み取り専用操作になります。
- コマンド出力は、**stdout** またはファイルのどちらかです。
  - **-f** オプションで指定しない場合、デフォルトでは、**yaml** コンテンツは **stdout** に書き出されます。

アプリケーションプランのエクスポート時に、以下のオプションを使用します。

#### Options

**-f --file=<value>** Write to file instead of stdout

#### Options for application-plan

**-c --config-file=<value>** 3scale toolbox configuration file:  
defaults to \$HOME/.3scalerc.yaml

**-h --help** Print help for this command

**-k --insecure** Proceed and operate even for server  
connections otherwise considered insecure

**-v --version** Print the version of this command

**--verbose** Verbose mode



### 5.10.6.2. ファイルからのアプリケーションプランのインポート

以下のコマンドにより、アプリケーションプランがインポートされます。

```
3scale application-plan import [opts] <remote> <service_system_name>
```

#### 例

```
$ podman run -v $PWD/plan.yaml:/tmp/plan.yaml registry.redhat.io/3scale-amp2/toolbox-
rhel8:3scale2.13 3scale application-plan import --file=/tmp/plan.yaml remote_name service_name
```

この例では、Podman ボリュームを使用して、現在の **\$PWD** フォルダからインポートされたファイルをコンテナにマウントします。

### 5.10.6.3. URL からのアプリケーションプランのインポート

```
3scale application-plan import -f http[s]://domain/resource/path.yaml remote_name service_name
```

#### 注記

##### import コマンドに固有の事項

- コマンド入力コンテンツは、**stdin**、ファイル、または URL 形式のいずれかです。
  - **-f** オプションで指定しない場合、デフォルトでは、**yaml** コンテンツは **stdin** から読み込まれます。
- アプリケーションプランがリモートサービスで見つからない場合は、アプリケーションプランが作成されます。
- オプションのパラメーター **-p**、**--plan** を使用すると、リモートターゲットのアプリケーションプランの **id** または **system\_name** が上書きされます。
  - **-p** オプションで指定されていない場合、デフォルトでは、**yaml** コンテンツからのプラン属性 **system\_name** によってアプリケーションプランが参照されます。
- **yaml** コンテンツからのメトリックまたはメソッドがリモートサービスで見つからない場合は、メトリックまたはメソッドが作成されます。

アプリケーションプランのインポート時に、以下のオプションを使用します。

#### Options

```
-f --file=<value>          Read from file or URL instead of
                           stdin
-p --plan=<value>         Override application plan reference
```

#### Options for application-plan

```
-c --config-file=<value>  3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                 Print help for this command
-k --insecure             Proceed and operate even for server
                           connections otherwise considered
```

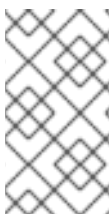
	<code>insecure</code>	
<code>-v --version</code>		Print the version of this command
<code>--verbose</code>		Verbose mode

## 5.11. メトリックの作成

3scale toolbox を使用して、デベロッパーポータルでのメトリックの作成、更新、リスト表示、および削除を行います。

メトリックを作成するには、以下の手順に従います。

- メトリック名を指定する必要があります。
- **system-name** を上書きするには、オプションのパラメーターを使用します。
- 同じ名前のメトリックがすでに存在する場合、エラーメッセージが表示されます。
- **--disabled** フラグを使用して、**無効な** メトリックを作成します。
  - デフォルトでは、**有効** になります。



### 注記

- **service** 位置引数はサービスの参照で、サービスの **id** またはサービスの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。

以下のコマンドにより、メトリックが作成されます。

```
3scale metric create [opts] <remote> <service> <metric-name>
```

メトリックの作成時に、以下のオプションを使用します。

#### Options

<code>--description=&lt;value&gt;</code>	Set a metric description
<code>--disabled</code>	Disable this metric in all application plans
<code>-o --output=&lt;value&gt;</code>	Output format on stdout: one of json yaml
<code>-t --system-name=&lt;value&gt;</code>	Set the application plan system name
<code>--unit=&lt;value&gt;</code>	Metric unit: default hit

#### Options for metric

<code>-c --config-file=&lt;value&gt;</code>	3scale toolbox configuration file: defaults to \$HOME/.3scalerc.yaml
<code>-h --help</code>	Print help for this command
<code>-k --insecure</code>	Proceed and operate even for server connections otherwise considered insecure
<code>-v --version</code>	Print the version of this command
<code>--verbose</code>	Verbose mode

### 5.11.1. メトリックの作成または更新

メトリックが存在しない場合に新しく作成する、または既存のメトリックを更新するには、以下の手順に従います。

- 同じ名前のメトリックがすでに存在する場合、エラーメッセージが表示されます。
- **--disabled** フラグを使用して、**無効な** メトリックを更新します。
- **--enabled** フラグを使用して、**有効な** メトリックに更新します。



#### 注記

- **service** 位置引数はサービスの参照で、サービスの **id** またはサービスの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。
- **metric** 位置引数はメトリック参照で、メトリックの **id** またはメトリックの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。

以下のコマンドにより、メトリックが更新されます。

```
3scale metric apply [opts] <remote> <service> <metric>
```

メトリックの更新時に、以下のオプションを使用します。

#### Options

```
--description=<value>  Set a metric description
--disabled             Disable this metric in all application
                       plans
--enabled              Enable this metric in all application
                       plans
-n --name=<value>      This will set the metric name
--unit=<value>         Metric unit: default hit
-o --output=<value>    Output format on stdout:
                       one of json|yaml
```

#### Options for metric

```
-c --config-file=<value> 3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure             Proceed and operate even for server
                           connections otherwise considered insecure
-v --version              Print the version of this command
--verbose                 Verbose mode
```

### 5.11.2. メトリックのリスト表示

以下のコマンドにより、メトリックがリスト表示されます。

```
3scale metric list [opts] <remote> <service>
```

メトリックのリスト表示時に、以下のオプションを使用します。





### 注記

- **service** 位置引数はサービスの参照で、サービスの **id** またはサービスの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。

以下のコマンドにより、メソッドが作成されます。

```
3scale method create [opts] <remote> <service> <method-name>
```

メソッドの作成時に、以下のオプションを使用します。

#### Options

```
--description=<value>  Set a method description
--disabled             Disable this method in all
                       application plans
-o --output=<value>    Output format on stdout:
                       one of json|yaml
-t --system-name=<value> Set the method system name
```

#### Options for method

```
-c --config-file=<value> 3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure             Proceed and operate even for server
                           connections otherwise considered insecure
-v --version              Print the version of this command
--verbose                 Verbose mode
```

### 5.12.2. メソッドの作成または更新

メソッドが存在しない場合に新しく作成する、または既存のメソッドを更新するには、以下の手順に従います。

- 同じ名前のメソッドがすでに存在する場合、コマンドはエラーメッセージを返します。
- **--disabled** フラグを使用して、**無効な** メソッドに更新します。
- **--enabled** フラグを使用して、**有効な** メソッドに更新します。



### 注記

- **service** 位置引数はサービスの参照で、サービスの **id** またはサービスの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。
- **method** 位置引数はメソッド参照で、メソッドの **id** またはメソッドの **system\_name** のどちらかです。
  - toolbox は、どちらか一方を使用します。

以下のコマンドにより、メソッドが更新されます。

-

```
3scale method apply [opts] <remote> <service> <method>
```

メソッドの更新時に、以下のオプションを使用します。

#### Options

```
--description=<value>  Set a method description
--disabled             Disable this method in all
                       application plans
--enabled              Enable this method in all
                       application plans
-n --name=<value>      Set the method name
-o --output=<value>    Output format on stdout:
                       one of json|yaml
```

#### Options for method

```
-c --config-file=<value> 3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                           connections otherwise considered insecure
-v --version             Print the version of this command
--verbose                Verbose mode
```

### 5.12.3. メソッドのリスト表示

以下のコマンドにより、メソッドがリスト表示されます。

```
3scale method list [opts] <remote> <service>
```

メソッドのリスト表示時に、以下のオプションを使用します。

#### Options

```
-o --output=<value>    Output format on stdout:
                       one of json|yaml
```

#### Options for method

```
-c --config-file=<value> 3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                           connections otherwise considered insecure
-v --version             Print the version of this command
--verbose                Verbose mode
```

### 5.12.4. メソッドの削除

以下のコマンドにより、メソッドが削除されます。

```
3scale method delete [opts] <remote> <service> <metric>
```

メソッドの削除時に、以下のオプションを使用します。

#### Options for method

```

-c --config-file=<value> 3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                        connections otherwise considered insecure
-v --version             Print the version of this command
--verbose               Verbose mode

```

## 5.13. サービスの作成

3scale toolbox を使用して、デベロッパーポータルサービスの作成、適用、リスト表示、表示、または削除を行います。

### 5.13.1. 新しいサービスの作成

以下のコマンドにより、新しいサービスが作成されます。

```
3scale service create [options] <remote> <service-name>
```

サービスの作成時に、以下のオプションを使用します。

#### Options

```

-a --authentication-mode=<value> Specify authentication mode of
                                the service:
                                - '1' for API key
                                - '2' for App Id/App Key
                                - 'oauth' for OAuth mode
                                - 'oidc' for OpenID Connect
-d --deployment-mode=<value>   Specify the deployment mode of
                                the service
--description=<value>          Specify the description of the
                                service
-o --output=<value>            Output format on stdout:
                                one of json|yaml
-s --system-name=<value>       Specify the system-name of the
                                service
--support-email=<value>       Specify the support email of the
                                service

```

#### Options for service

```

-c --config-file=<value>       3scale toolbox configuration file:
                                defaults to $HOME/.3scalerc.yaml
-h --help                      Print help for this command
-k --insecure                  Proceed and operate even for
                                server connections otherwise
                                considered insecure
-v --version                   Print the version of this command
--verbose                      Verbose mode

```

### 5.13.2. サービスの作成または更新

サービスが存在しない場合に新しく作成する、または既存のサービスを更新するには、以下の手順に従います。



## 注記

- **service-id\_or\_system-name** 位置引数は、サービス参照です。
  - サービスの **id**、またはサービスの **system\_name** のどちらかです。
  - toolbox は、これを自動的に判別します。
- このコマンドは **べきとう性** を持ちます。

以下のコマンドにより、サービスが更新されます。

```
3scale service apply <remote> <service-id_or_system-name>
```

サービスの更新時に、以下のオプションを使用します。

### Options

```
-a --authentication-mode=<value>  Specify authentication mode of
                                the service:
                                - '1' for API key
                                - '2' for App Id/App Key
                                - 'oauth' for OAuth mode
                                - 'oidc' for OpenID Connect
-d --deployment-mode=<value>      Specify the deployment mode of
                                the service
--description=<value>             Specify the description of the
                                service
-n --name=<value>                 Specify the name of the metric
--support-email=<value>           Specify the support email of the
                                service
-o --output=<value>              Output format on stdout:
                                one of json|yaml
```

### Options for services

```
-c --config-file=<value>         3scale toolbox configuration file:
                                defaults to $HOME/.3scalerc.yaml
-h --help                        Print help for this command
-k --insecure                    Proceed and operate even for
                                server connections otherwise
                                considered insecure
-v --version                     Print the version of this command
--verbose                        Verbose mode
```

### 5.13.3. サービスのリスト表示

以下のコマンドにより、サービスがリスト表示されます。

```
3scale service list <remote>
```

サービスのリスト表示時に、以下のオプションを使用します。

### Options

```
-o --output=<value>             Output format on stdout:
                                one of json|yaml
```



## Options for services

```

-c --config-file=<value>    3scale toolbox configuration file:
                             defaults to $HOME/.3scalerc.yaml
-h --help                    Print help for this command
-k --insecure                Proceed and operate even for server
                             connections otherwise considered insecure
-v --version                 Print the version of this command
--verbose                    Verbose mode

```

#### 5.13.4. サービスの表示

以下のコマンドにより、サービスが表示されます。

```
3scale service show <remote> <service-id_or_system-name>
```

サービスの表示時に、以下のオプションを使用します。

## Options

```

-o --output=<value>        Output format on stdout:
                             one of json|yaml

```

## Options for services

```

-c --config-file=<value>    3scale toolbox configuration file:
                             defaults to $HOME/.3scalerc.yaml
-h --help                    Print help for this command
-k --insecure                Proceed and operate even for server
                             connections otherwise considered insecure
-v --version                 Print the version of this command
--verbose                    Verbose mode

```

#### 5.13.5. サービスの削除

以下のコマンドにより、サービスが削除されます。

```
3scale service delete <remote> <service-id_or_system-name>
```

サービスの削除時に、以下のオプションを使用します。

## Options for services

```

-c --config-file=<value>    3scale toolbox configuration file:
                             defaults to $HOME/.3scalerc.yaml
-h --help                    Print help for this command
-k --insecure                Proceed and operate even for server
                             connections otherwise considered insecure
-v --version                 Print the version of this command
--verbose                    Verbose mode

```

### 5.14. ACTIVEDOCS の作成

3scale toolbox を使用して、デベロッパーポータル *ActiveDocs* の作成、更新、リスト表示、または削除を行います。

### 5.14.1. 新しい ActiveDocs の作成

OpenAPI 使用に準拠した API 定義から新しい ActiveDocs を作成するには、以下の手順を実施します。

1. API 定義を 3scale に追加し、オプションで名前を付けます。

```
3scale activedocs create <remote> <activedocs-name> <specification>
```

ActiveDocs の OpenAPI 仕様は必須で、以下の値のいずれかでなければなりません。

- 利用可能なパスのファイル名
- toolbox がコンテンツをダウンロードすることのできる URL。サポートされるスキームは **http** および **https** です。
- **stdin** 標準入力ストリームから読み込む。これは、値に **-** を設定することで制御されます。ActiveDocs の作成時に、以下のオプションを使用します。

#### Options

```
-d --description=<value>    Specify the description of
                             the ActiveDocs
-i --service-id=<value>    Specify the Service ID
                             associated to the ActiveDocs
-o --output=<value>        Output format on stdout: one
                             of json|yaml
-p --published              Specify to publish the
                             ActiveDocs on the Developer
                             Portal. Otherwise it is hidden.
-s --system-name=<value>   Specify the system-name of
                             the ActiveDocs
--skip-swagger-validations Specify to skip validation
                             of the Swagger specification
```

#### Options for ActiveDocs

```
-c --config-file=<value>   toolbox configuration file.
                             Defaults to $HOME/.3scalerc.yaml
-h --help                  Print help for this command
-k --insecure              Proceed and operate even for
                             server connections otherwise
                             considered insecure
-v --version               Print the version of this command
--verbose                  Verbose mode
```

2. デベロッパーポータルに定義を [公開](#) します。

### 5.14.2. ActiveDocs の作成または更新

ActiveDoc が存在しない場合に新しく作成する、または新しい API 定義で既存の ActiveDocs を更新するには、以下のコマンドを使用します。

```
3scale activedocs apply <remote> <activedocs_id_or_system_name>
```

ActiveDocs の更新時に、以下のオプションを使用します。

#### Options

```

-d --description=<value>      Specify the description of the
                               ActiveDocs
  --hide                       Specify to hide the ActiveDocs
                               on the Developer Portal
-i --service-id=<value>      Specify the Service ID associated
                               to the ActiveDocs
-o --output=<value>          Output format on stdout:
                               one of json|yaml
  --openapi-spec=<value>      Specify the Swagger specification.
                               Can be a file, a URL or '-' to read
                               from stdin. This is a mandatory
                               option when applying the ActiveDoc
                               for the first time.
-p --publish                  Specify to publish the ActiveDocs
                               on the Developer Portal. Otherwise
                               it is hidden
-s --name=<value>            Specify the name of the ActiveDocs
  --skip-swagger-validations=<value> Specify whether to skip validation
                               of the Swagger specification: true
                               or false. Defaults to true.

```

#### Options for ActiveDocs

```

-c --config-file=<value>      3scale toolbox configuration file:
                               defaults to $HOME/.3scalerc.yaml
-h --help                    Print help for this command
-k --insecure                 Proceed and operate even for server
                               connections otherwise considered
                               insecure
-v --version                  Print the version of this command
  --verbose                   Verbose mode

```



#### 注記

**activedocs apply --skip-swagger-validations** の動作が、3scale 2.8 で変更されました。**activedocs apply** を使用する既存のスクリプトを更新しなければならない場合があります。従来は、各 **activedocs apply** コマンドでこのオプションを指定しない場合、検証はスキップされませんでした。今回、**--skip-swagger-validations** はデフォルトで **true** になりました。

### 5.14.3. ActiveDocs のリスト表示

以下の項目を含め、デベロッパーポータルすべての ActiveDocs に関する情報を取得するには、以下のコマンドを使用します。

- ID
- 名前
- システム名
- 説明
- 公開済み (つまり、デベロッパーポータルに表示可能) かどうか
- 作成日

- 最終更新日

以下のコマンドにより、定義済みの ActiveDocs がすべてリスト表示されます。

```
3scale activedocs list <remote>
```

ActiveDocs のリスト表示時に、以下のオプションを使用します。

#### Options

```
-o --output=<value>      Output format on stdout:
                          one of json|yaml
-s --service-ref=<value>  Filter the ActiveDocs by service
                          reference
```

#### Options for ActiveDocs

```
-c --config-file=<value>  3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
-h --help                 Print help for this command
-k --insecure             Proceed and operate even for server
                          connections otherwise considered insecure
-v --version              Print the version of this command
--verbose                 Verbose mode
```

### 5.14.4. ActiveDocs の削除

以下のコマンドにより、ActiveDocs が削除されます。

```
3scale activedocs delete <remote> <activedocs-id_or-system-name>
```

ActiveDocs の削除時に、以下のオプションを使用します。

#### Options for ActiveDocs

```
-c --config-file=<value>  3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
-h --help                 Print help for this command
-k --insecure             Proceed and operate even for server
                          connections otherwise considered insecure
-v --version              Print the version of this command
--verbose                 Verbose mode
```

### 5.15. プロキシ設定のリスト表示

3scale toolbox を使用して、デベロッパーポータルすべての定義済みプロキシ設定のリスト表示、表示、およびプロモートを行います。

以下のコマンドにより、プロキシ設定がリスト表示されます。

```
3scale proxy-config list <remote> <service> <environment>
```

プロキシ設定のリスト表示時に、以下のオプションを使用します。

#### Options

```
-o --output=<value>      Output format on stdout:
```

one of json|yaml

#### Options for proxy-config

```
-c --config-file=<value>  3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                  Print help for this command
-k --insecure              Proceed and operate even for server
                           connections otherwise considered insecure
-v --version               Print the version of this command
--verbose                  Verbose mode
```

### 5.15.1. プロキシ設定の表示

以下のコマンドにより、プロキシ設定が表示されます。

```
3scale proxy-config show <remote> <service> <environment>
```

プロキシ設定の表示時に、以下のオプションを使用します。

#### Options

```
--config-version=<value> Specify the proxy configuration version.
                           If not specified, defaults to latest
-o --output=<value>       Output format on stdout:
                           one of json|yaml
```

#### Options for proxy-config

```
-c --config-file=<value>  3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                  Print help for this command
-k --insecure              Proceed and operate even for server
                           connections otherwise considered
                           insecure
-v --version               Print the version of this command
--verbose                  Verbose mode
```

### 5.15.2. プロキシ設定のプロモート

以下のコマンドにより、最新のステージング環境用プロキシ設定が実稼働環境にプロモートされます。

```
3scale proxy-config promote <remote> <service>
```

最新のステージング環境用プロキシ設定を実稼働環境にプロモートする際に、以下のオプションを使用します。

#### Options for proxy-config

```
-c --config-file=<value>  3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
-h --help                  Print help for this command
-k --insecure              Proceed and operate even for server
                           connections otherwise considered insecure
-v --version               Print the version of this command
--verbose                  Verbose mode
```

### 5.15.3. プロキシ設定のエクスポート

Self-managed APIcast ゲートウェイが 3scale インスタンスに接続されていない場合などに、**proxy-config export** コマンドを使用します。このシナリオでは、3scale 設定を手動で注入するか、[APICast デプロイメントおよび設定オプション](#) を使用して注入します。いずれの場合も、3scale の設定を指定する必要があります。

以下のコマンドは、APIcast ゲートウェイに注入することのできる設定をエクスポートします。

```
3scale proxy-config export <remote>
```

3scale [設定ファイル](#) として使用されるプロバイダーアカウントのプロキシ設定をエクスポートする際に、以下のオプションを指定することができます。

```
Options for proxy-config
--environment=<value>    Gateway environment. Must be 'sandbox' or
                        'production' (default: sandbox)
-o --output=<value>      Output format. One of: json|yaml
```

### 5.15.4. プロキシ設定のデプロイ

以下の **deploy** コマンドは、Service Mesh を使用している場合に、APIcast 設定を 3scale のステージング環境または実稼働環境にプロモートします。

```
3scale proxy deploy <remote> <service>
```

**deploy** コマンドを使用して APIcast 設定をステージング環境にプロモートする時に、以下のオプションを指定できます。

```
-o --output=<value>      Output format. One of: json|yaml
```

### 5.15.5. プロキシ設定の更新

以下の **update** コマンドにより、APIcast の設定が更新されます。

```
3scale proxy update <remote> <service>
```

update コマンドを使用して APIcast 設定を **更新** する場合に、以下のオプションを指定することができます。

```
-o --output=<value>      Output format. One of: json|yaml
-p --param=<value>       APIcast configuration parameters. Format:
                        [--param key=value]. Multiple options allowed.
```

### 5.15.6. プロキシ設定の表示

以下の **show** コマンドにより、アンデプロイされた APIcast 設定が取得されます。

```
3scale proxy show <remote> <service>
```

**show** コマンドを使用して、アンデプロイされた APIcast 設定を取得する場合は、以下のオプションを指定できます。

`-o --output=<value>`      Output format. One of: json|yaml

### 5.15.7. プロキシ設定のデプロイ (非推奨)



#### 注記

3scale 2.12 では、**proxy-config deploy** コマンドのサポートは非推奨になりました。

以下のコマンドを使用します。

- **proxy deploy**
- **proxy update**
- **proxy show**

詳細は、[プロキシ設定のデプロイ](#) を参照してください。

以下の **deploy** コマンドは、Service Mesh を使用している場合に、APIcast 設定を 3scale のステージング環境または実稼働環境にプロモートします。

`3scale proxy-config deploy <remote> <service>`

**deploy** コマンドを使用して APIcast 設定をステージング環境にプロモートする時に、以下のオプションを指定できます。

`-o --output=<value>`      Output format. One of: json|yaml

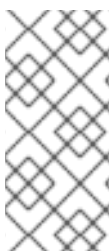
#### 関連情報

- [リモート](#)

### 5.16. ポリシーレジストリーのコピー

以下に該当する場合、toolbox コマンドを使用して、3scale のソースアカウントからターゲットアカウントにポリシーレジストリーをコピーします。

- 存在しないカスタムポリシーがターゲットアカウントに作成されている。
- 一致するカスタムポリシーがターゲットアカウントで更新されている。
- この copy コマンドがべきとう性を持つ。



#### 注記

- 存在しないカスタムポリシーとは、ソースアカウントには存在するが、アカウントのテナントには存在しないカスタムポリシーと定義されます。
- 一致するカスタムポリシーとは、ソースアカウントとターゲットアカウントの両方に存在するカスタムポリシーと定義されます。

以下のコマンドにより、ポリシーレジストリーがコピーされます。

```
3scale policy-registry copy [opts] <source_remote> <target_remote>
```

Option for policy-registry

```
-c --config-file=<value> 3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                        connections otherwise considered insecure
-v --version             Print the version of this command
--verbose               Verbose mode
```

## 5.17. アプリケーションのリスト表示

3scale toolbox を使用して、デベロッパーポータルでのアプリケーションのリスト表示、作成、表示、適用、または削除を行います。

以下のコマンドにより、アプリケーションがリスト表示されます。

```
3scale application list [opts] <remote>
```

アプリケーションのリスト表示時に、以下のオプションを使用します。

### OPTIONS

```
--account=<value>      Filter by account
-o --output=<value>    Output format on stdout:
                        one of json|yaml
--plan=<value>         Filter by application plan. Service
                        option required.
--service=<value>     Filter by service
```

### OPTIONS FOR APPLICATION

```
-c --config-file=<value> 3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                        connections otherwise considered insecure
-v --version             Print the version of this command
--verbose               Verbose mode
```

### 5.17.1. アプリケーションの作成

特定の 3scale アカウントおよびアプリケーションプランにリンクされたアプリケーションを 1 つ作成するには、create コマンドを使用します。

必要な位置パラメーターは以下のとおりです。

- **<service>** 参照。サービスの **id** またはサービスの **system\_name** のどちらかです。
- **<account>** 参照。次のいずれかです。
  - アカウント **id**
  - アカウントの管理ユーザーの **username**、**email**、または **user\_id**



- **provider\_key**
- **<application plan>** 参照。プランの **id** またはプランの **system\_name** のどちらかです。
- **<name>** アプリケーション名。

以下のコマンドにより、アプリケーションが作成されます。

```
3scale application create [opts] <remote> <account> <service> <application-plan> <name>
```

アプリケーションの作成時に、以下のオプションを使用します。

#### OPTIONS

- application-id=<value> App ID or Client ID (for OAuth and OpenID Connect authentication modes) of the application to be created.
- application-key=<value> App Key(s) or Client Secret (for OAuth and OpenID Connect authentication modes) of the application created.
- description=<value> Application description
- o --output=<value> Output format on stdout:  
one of json|yaml
- redirect-url=<value> OpenID Connect redirect url
- user-key=<value> User Key (API Key) of the application to be created.

#### OPTIONS FOR APPLICATION

- c --config-file=<value> 3scale toolbox configuration file:  
defaults to \$HOME/.3scalerc.yaml
- h --help Print help for this command
- k --insecure Proceed and operate even for server connections otherwise considered insecure
- v --version Print the version of this command
- verbose Verbose mode

### 5.17.2. アプリケーションの表示

以下のコマンドにより、アプリケーションが表示されます。

```
3scale application show [opts] <remote> <application>
```

アプリケーションパラメーターは以下のいずれかです。

- **user\_key**: API キー
- **App\_id**: app\_id/app\_key ペアから、または **OAuth** および **OpenID Connect (OIDC)** 認証モードの **Client ID**
- アプリケーションの内部 **id**

#### OPTIONS

- o --output=<value> Output format on stdout:  
one of json|yaml

## OPTIONS FOR APPLICATION

```

-c --config-file=<value> 3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
-h --help                Print help for this command
-k --insecure            Proceed and operate even for server
                        connections otherwise considered insecure
-v --version             Print the version of this command
--verbose               Verbose mode

```

## 5.17.3. アプリケーションの作成または更新

アプリケーションが存在しない場合に新しく作成する、または既存のアプリケーションを更新するには、以下のコマンドを使用します。

```
3scale application apply [opts] <remote> <application>
```

アプリケーションパラメーターは以下のいずれかです。

- **user\_key**: API キー
- **App\_id**: app\_id/app\_key ペアから、または OAuth および OIDC 認証モードの Client ID
- アプリケーションの内部 **id**
- アプリケーションが見つからず作成する必要がある場合は、オプションの **account** 引数が必要です。次のいずれかです。
  - アカウント **id**
  - 3scale アカウントの管理ユーザーの **username**、**email**、または **user\_id**
  - **provider\_key**
- 3scale ではアプリケーション名が一意ではないため、**name** は固有の識別子として使用できません。
- **--resume** フラグで、一時停止されていたアプリケーションを再開します。
- アプリケーションの一時停止: **--suspend** フラグで状態を一時停止中に変更します。

アプリケーションの更新時に、以下のオプションを使用します。

## OPTIONS

```

--account=<value>      Application's account. Required when
                        creating
--application-key=<value> App Key(s) or Client Secret (for OAuth
                        and OpenID Connect authentication
                        modes) of the application to be
                        created. Only used when application
                        does not exist.
--description=<value>  Application description
--name=<value>         Application name
-o --output=<value>    Output format on stdout:
                        one of json|yaml
--plan=<value>         Application's plan. Required when

```

```

        creating.
--redirect-url=<value>   OpenID Connect redirect url
--resume                Resume a suspended application
--service=<value>       Application's service. Required when
                        creating.
--suspend               Suspends an application (changes the
                        state to suspended)
--user-key=<value>      User Key (API Key) of the application
                        to be created.

```

#### OPTIONS FOR APPLICATION

```

-c --config-file=<value> 3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
-h --help                Show help for this command
-k --insecure            Proceed and operate even for server
                        connections otherwise considered insecure
-v --version             Print the version of this command
--verbose                Verbose mode.

```

#### 5.17.4. アプリケーションの削除

以下のコマンドにより、アプリケーションが削除されます。

```
3scale application delete [opts] <remote> <application>
```

アプリケーションパラメーターは以下のいずれかです。

- **user\_key**: API キー
- **App\_id**: app\_id/app\_key ペアから、または OAuth および OIDC 認証モードの Client ID
- アプリケーションの内部 id

#### 5.18. プロダクトのエクスポート

3scale プロダクト定義を **.yaml** 形式でエクスポートすると、そのプロダクトをソース 3scale インスタンスとの接続がない 3scale インスタンスにインポートすることができます。3scale プロダクトを設定してから、そのプロダクトをエクスポートする必要があります。[Creating new products to test API calls](#) を参照してください。

2つの 3scale インスタンスにネットワーク接続がある場合、両方の 3scale インスタンスで同じ 3scale 製品を使用するには、[toolbox 3scale copy コマンド](#) を使用します。

#### 説明

3scale 製品をエクスポートすると、ツールボックスは、[製品](#) および [バックエンド](#) のカスタムリソース定義 (CRD) に準拠する **.yaml** 形式で製品定義をシリアル化します。詳細は、[3scale Operator を使用した 3scale の設定とプロビジョニング](#) を参照してください。**.yaml** 出力には、プロダクトの基本情報の他に、以下が含まれます。

- プロダクトにリンクされたバックエンド。
- リンクされたバックエンドのメトリック、メソッド、およびマッピングルール。
- アプリケーションプランで定義される制限および課金ルール。

- 制限および課金ルールで参照されるメトリックおよびメソッド。

プロダクトのエクスポートは、読み取り専用の操作です。つまり、プロダクトを繰り返しエクスポートしても安全性に問題はありません。toolbox は、エクスポートされるプロダクトを変更しません。必要であれば、別の 3scale インスタンスにインポートする前に **.yaml** 出力を変更することができます。

3scale プロダクトのエクスポートは、以下の状況を対象としています。

- 移行元および宛先 3scale インスタンス間の接続がない。たとえば、ネットワークに重大な制限があり、複数の 3scale インスタンスで同じプロダクトを使用するときに toolbox の **3scale copy** コマンドを実行できない場合などです。
- Git またはその他のソースコントロールシステムを使用して、**.yaml** 形式で 3scale プロダクト定義を維持する。

3scale toolbox の **export** および **import** コマンドは、プロダクト定義のバックアップおよび復元にも役立つことがあります。

## 形式

**export** コマンドを実行するには、以下の形式を使用します。

```
$ 3scale product export [-f output-file] <remote> <product>
```

**export** コマンドは、出力を **stdout** またはファイルに送信できます。デフォルトは **stdout** です。出力をファイルに送信するには、**.yaml** ファイルの名前を指定して **-f** または **--file** オプションを指定します。

**<remote>** を、プロダクトのエクスポート元の 3scale インスタンスに関連付けられた 3scale プロバイダーアカウントエイリアスまたは URL に置き換えます。これを指定する方法の詳細については、[リモートアクセスクレデンシャルの管理](#) を参照してください。

**<product>** を、エクスポートするプロダクトのシステム名または 3scale ID に置き換えます。このプロダクトは、指定した 3scale プロバイダーアカウントに関連付けられている必要があります。プロダクトの [概要](#) ページの 3scale GUI では、プロダクトのシステム名を確認できます。プロダクトの 3scale ID を取得するには、[toolbox 3scale services show コマンド](#) を実行します。

## 例

以下のコマンドは、**my-3scale-1** プロバイダーアカウントに関連付けられた 3scale インスタンスから **petstore** プロダクトをエクスポートし、それを **petstore-product.yaml** ファイルに出力します。

```
$ 3scale product export -f petstore-product.yaml my-3scale-1 petstore
```

以下は、**Default API** プロダクトのシリアルライズの例です。

```
apiVersion: v1
kind: List
items:
- apiVersion: capabilities.3scale.net/v1beta1
  kind: Product
  metadata:
    annotations:
      3scale_toolbox_created_at: '2021-02-17T10:59:23Z'
      3scale_toolbox_version: 0.17.1
  name: api.xysnalcj
```

```
spec:
  name: Default API
  systemName: api
  description: "
  mappingRules:
  - httpMethod: GET
    pattern: "/v2"
    metricMethodRef: hits
    increment: 1
    last: false
  metrics:
  hits:
    friendlyName: Hits
    unit: hit
    description: Number of API hits
  methods:
  servicemethod01:
    friendlyName: servicemethod01
    description: "
  policies:
  - name: apicast
    version: builtin
    configuration: {}
    enabled: true
  applicationPlans:
  basic:
    name: Basic
    appsRequireApproval: false
    trialPeriod: 0
    setupFee: 0.0
    custom: false
    state: published
    costMonth: 0.0
    pricingRules:
    - from: 1
      to: 1000
      pricePerUnit: 1.0
      metricMethodRef:
        systemName: hits
    limits:
    - period: hour
      value: 1222222
      metricMethodRef:
        systemName: hits
        backend: backend_01
  backendUsages:
  backend_01:
    path: "/v1/pets"
  backend_02:
    path: "/v1/cats"
  deployment:
  apicastSelfManaged:
  authentication:
  oidc:
    issuerType: rest
    issuerEndpoint: https://hello:test@example.com/auth/realms/3scale-api-consumers
```

```
jwtClaimWithClientID: azp
jwtClaimWithClientIDType: plain
authenticationFlow:
  standardFlowEnabled: false
  implicitFlowEnabled: true
  serviceAccountsEnabled: false
  directAccessGrantsEnabled: true
credentials: query
security:
  hostHeader: "
  secretToken: some_secret
gatewayResponse:
  errorStatusAuthFailed: 403
  errorHeadersAuthFailed: text/plain; charset=us-ascii
  errorAuthFailed: Authentication failed
  errorStatusAuthMissing: 403
  errorHeadersAuthMissing: text/plain; charset=us-ascii
  errorAuthMissing: Authentication parameters missing
  errorStatusNoMatch: 404
  errorHeadersNoMatch: text/plain; charset=us-ascii
  errorNoMatch: No Mapping Rule matched
  errorStatusLimitsExceeded: 429
  errorHeadersLimitsExceeded: text/plain; charset=us-ascii
  errorLimitsExceeded: Usage limit exceeded
stagingPublicBaseURL: http://staging.example.com:80
productionPublicBaseURL: http://example.com:80
- apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  annotations:
    3scale_toolbox_created_at: '2021-02-17T10:59:34Z'
    3scale_toolbox_version: 0.17.1
  name: backend.01.pcjwxbdu
spec:
  name: Backend 01
  systemName: backend_01
  privateBaseURL: https://b1.example.com:443
  description: new desc
  mappingRules:
  - httpMethod: GET
    pattern: "/v1/pets"
    metricMethodRef: hits
    increment: 1
    last: false
  metrics:
    hits:
      friendlyName: Hits
      unit: hit
      description: Number of API hits
  methods:
    mybackendmethod01:
      friendlyName: mybackendmethod01
      description: "
- apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
```

```

annotations:
  3scale_toolbox_created_at: '2021-02-17T10:59:34Z'
  3scale_toolbox_version: 0.17.1
name: backend.02.tiedgjsk
spec:
  name: Backend 02
  systemName: backend_02
  privateBaseURL: https://b2.example.com:443
  description: "
  mappingRules:
  - httpMethod: GET
    pattern: "/v1/cats"
    metricMethodRef: hits
    increment: 1
    last: false
  metrics:
    hits:
      friendlyName: Hits
      unit: hit
      description: Number of API hits
  methods:
    backend02_method01:
      friendlyName: backend02_method01
      description: "

```

## Product CR へのエクスポートおよびパイプ

**export** コマンドを実行すると、出力をパイプして **プロダクトカスタムリソース (CR)** を作成できます。この CR が含まれる 3scale インスタンスは以下により異なります。

- **threescale-provider-account** シークレットが定義されている場合、3scale operator はそのシークレットによって識別される 3scale インスタンスにプロダクト CR を作成します。
- **threescale-provider-account** シークレットが定義されていない場合は、新規プロダクト CR が置かれている namespace に 3scale インスタンスがインストールされていると、3scale operator はその namespace にプロダクト CR を作成します。
- **threescale-provider-account** シークレットが定義されておらず、新規プロダクト CR が置かれている namespace に 3scale インスタンスが含まれていない場合は、3scale operator はプロダクト CR を failed 状態とマークします。

**threescale-provider-account** シークレットが含まれる namespace で以下のコマンドを実行するとします。toolbox は、**threescale-provider-account** シークレットで識別された 3scale インスタンスに **petstore** CR をパイプ処理します。

```
$ 3scale product export my-3scale-1 petstore | oc apply -f -
```

## 関連情報

- [3scale Operator を使用した 3scale の設定とプロビジョニング](#)
- [3scale Operator が、カスタムリソースのリンク先となるテナントを識別する方法](#)

## 5.19. プロダクトのインポート

移行元および宛先の 3scale インスタンスにネットワーク接続がない場合に、同じ 3scale プロダクトを複数の 3scale インスタンスで使用するには、ある 3scale インスタンスから 3scale プロダクトをエクスポートし、別の 3scale インスタンスにインポートします。プロダクトをインポートするには、toolbox **3scale product import** コマンドを実行します。

2 つの 3scale インスタンスにネットワーク接続がある場合、両方の 3scale インスタンスで同じ 3scale 製品を使用するには、toolbox **3scale copy** コマンドを使用します。

## 説明

3scale プロダクトをインポートすると、toolbox は、**Product** および **Backend** カスタムリソース定義 (CRD) に準拠する、**.yaml** 形式のシリアライズされたプロダクト定義を想定します。toolbox **3scale product export** コマンドを実行するか、**.yaml** 形式のプロダクト定義を手動で作成して、この **.yaml** コンテンツを取得できます。

プロダクトをエクスポートした場合、インポートされた定義にはエクスポートされた内容が含まれません。これには以下が含まれます。

- プロダクトにリンクされたバックエンド。
- リンクされたバックエンドのメトリック、メソッド、およびマッピングルール。
- アプリケーションプランで定義される制限および課金ルール。
- 制限および課金ルールで参照されるメトリックおよびメソッド。

必要であれば、別の 3scale インスタンスにインポートする前に、エクスポートされた **.yaml** 出力を変更することができます。

**import** コマンドはべきとう性を持ちます。これを何回でも実行して同じプロダクトをインポートしても、作成される 3scale 設定は同じままとなります。インポートプロセス中にエラーが発生した場合は、コマンドを再実行しても安全性に問題はありません。**import** プロセスが 3scale インスタンスでプロダクトを見つけられない場合、プロダクトが作成されます。また、**.yaml** 定義で定義され、3scale インスタンスで見つけられないメトリック、メソッド、またはバックエンドも作成されます。

3scale プロダクトのインポートは、以下の状況を対象としています。

- 移行元および宛先 3scale インスタンス間の接続がない。たとえば、ネットワークに重大な制限があり、複数の 3scale インスタンスで同じプロダクトを使用するときに toolbox の **3scale copy** コマンドを実行できない場合などです。
- Git またはその他のソースコントロールシステムを使用して、**.yaml** 形式で 3scale プロダクト定義を維持する。

3scale toolbox の **export** および **import** コマンドは、プロダクト定義のバックアップおよび復元にも役立つことがあります。

## 形式

**import** コマンドを実行するには、この形式を使用します。

```
3scale product import [<options>] <remote>
```

**import** コマンドは、**.yaml** の入力を **stdin** またはファイルから取得します。デフォルトは **stdin** です。

以下のオプションを指定することができます。



- **-f** または **--file** の後にファイル名を指定すると、指定した **.yaml** ファイルから入力取得されます。このファイルには、3scale の **Product** および **Backend** CRD に準拠する 3scale プロダクト定義が含まれる必要があります。
- **-o** または **--output** の後に **json** または **yaml** を指定すると、指定した形式でインポートされたものがリストされたレポートが出力されます。デフォルトの出力形式は **json** です。

<remote> を、プロダクトのインポート先の 3scale インスタンスに関連付けられた 3scale プロバイダーアカウントエイリアスまたは URL に置き換えます。これを指定する方法の詳細については、[リモートアクセスクレデンシャルの管理](#) を参照してください。

## 例

以下のコマンドは、**petstore-product.yaml** で定義されたプロダクトを、**my-3scale-2** プロバイダーアカウントに関連付けられた 3scale インスタンスにインポートします。デフォルトでは、インポートされた内容のレポートは **.json** 形式になります。

```
3scale product import -f petstore-product.yaml my-3scale-2
```

**import** コマンドは、インポートされたアイテムをリスト表示するレポートを出力します。以下に例を示します。

```
api:
  product_id: 2555417888846
  backends:
    backend_01:
      backend_id: 73310
      missing_metrics_created: 1
      missing_methods_created: 1
      missing_mapping_rules_created: 1
    backend_02:
      backend_id: 73311
      missing_metrics_created: 0
      missing_methods_created: 2
      missing_mapping_rules_created: 1
  missing_methods_created: 1
  missing_metrics_created: 1
  missing_mapping_rules_created: 2
  missing_application_plans_created: 2
  application_plans:
    basic:
      application_plan_id: 2357356246461
      missing_limits_created: 7
      missing_pricing_rules_created: 7
    unlimited:
      application_plan_id: 2357356246462
      missing_limits_created: 1
      missing_pricing_rules_created: 0
```

シリアライズされたプロダクト定義の例は、[プロダクトのエクスポート](#) の最後にあります。

## 5.20. プロダクトポリシーチェーンのエクスポートおよびインポート

プロダクトのポリシーチェーンを **yaml** または **json** コンテンツにエクスポートまたはインポートすることができます。コマンドラインで、**id** または **system** の値でプロダクトを参照します。プロダクトのポ

リシーチェーンをエクスポートまたはインポートする前に、3scale プロダクトを設定する必要があります。[API コールをテストするための新規プロダクトの作成](#) を参照してください。

### export コマンドの機能

- このコマンドは、リモートプロダクトの読み取り専用操作になります。
- このコマンドは、デフォルトで出力を標準出力 **stdout** に書き込みます。 **-f** フラグは、コマンドの出力をファイルに書き込むために使用できます。
- コマンド出力形式は、 **json** または **yaml** のどちらかです。デフォルトの形式は **yaml** であることに注意してください。

### エクスポートプロダクトポリシーチェーンのヘルプオプション

```
NAME
  export - export product policy chain
USAGE
  3scale policies export [opts] <remote>
  <product>
DESCRIPTION
  export product policy chain
OPTIONS
  -f --file=<value>      Write to file instead of stdout
  -o --output=<value>    Output format. One of: json|yaml
```

### コマンドの形式

- ポリシーチェーンを **yaml** のファイルにエクスポートするコマンドの形式を以下に示します。

```
$ 3scale policies export -f policies.yaml -o yaml remote_name product_name
```

### import コマンドの機能:

- コマンドは、標準入力または **stdin** から入力を読み取ります。 **-f FILE** フラグが設定されている場合、入力はファイルから読み取られます。 **-u URL** フラグが設定されている場合、入力は URL から読み取られます。
- インポートされたコンテンツは、 **yaml** または **json** のいずれかになります。 **toolbox** が自動的に検出するため、形式を指定する必要はありません。
- 既存のポリシーチェーンは、新しくインポートされたポリシーチェーンで上書きされません。 **SET** セマンティクスが実装されます。
- すべてのコンテンツの検証は、3scale API に委任されます。

### インポートプロダクトポリシーチェーンのヘルプオプション

```
NAME
  import - import product policy chain
USAGE
  3scale policies import [opts] <remote>
  <product>
DESCRIPTION
```

```
import product policy chain
OPTIONS
-f --file=<value>      Read from file
-u --url=<value>      Read from url
```

## コマンドの形式

- 以下は、ファイルからポリシーチェーンをインポートするコマンドの形式です。

```
$ 3scale policies import -f plan.yaml remote_name product_name
```

- 以下は、URI からポリシーチェーンをインポートするコマンドの形式です。

```
$ 3scale policies import -f http[s]://domain/resource/path.yaml remote_name product_name
```

## 5.21. API バックエンドのコピー

指定した 3scale システムに、特定のソース API バックエンドのコピーを作成します。ターゲットのシステムは、デフォルトでは、まずソースのバックエンドシステム名で検索されます。

- 選択したシステム名のバックエンドが見つからない場合は、そのバックエンドが作成されません。
- 選択したシステム名のバックエンドが見つかった場合は、置き換えられます。不足しているメトリックとメソッドのみが作成され、マッピングルールは完全に新しいものに置き換えられます。

**--target\_system\_name** オプションを使用して、システム名を上書きすることができます。

### コピーされるコンポーネント

以下の API バックエンドコンポーネントがコピーされます。

- メトリック
- メソッド
- マッピングルール: これらはコピーされ、置き換えられます。

### 手順

- 以下のコマンドを入力して API バックエンドをコピーします。

```
3scale backend copy [opts] -s <source_remote> -d <target_remote> <source_backend>
```

3scale インスタンスには、リモート名または URL を指定することができます。



### 注記

1つのコマンドにつき1つの API バックエンドしかコピーすることはできません。複数のコマンドを使用して、複数のバックエンドをコピーすることができます。**--target\_system\_name** で異なる名前を指定して、同じバックエンドを複数回コピーすることができます。

API バックエンドのコピー時に、以下のオプションを使用します。

#### Options

- d --destination=<value> 3scale target instance: URL or remote name (required).
- s --source=<value> 3scale source instance: URL or remote name (required).
- t --target\_system\_name=<value> Target system name: defaults to source system name.

以下のコマンド例は、**--target\_system\_name** で異なる値を指定して、API バックエンドを複数回コピーする方法を示しています。

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale backend copy [-t target_system_name] -s 3scale1 -d 3scale2 api_backend_01
```

## 5.22. API プロダクトのコピー

ターゲットの 3scale システムに、特定のソース API プロダクトのコピーを作成します。ターゲットのシステムは、デフォルトでは、まずソースの API プロダクトのシステム名で検索されます。

- 選択した **system-name** のプロダクトが見つからない場合は、そのプロダクトが作成されず。
- 選択した **system-name** のプロダクトが存在する場合は、そのプロダクトが更新されます。不足しているメトリックとメソッドのみが作成され、マッピングルールは完全に新しいものに置き換えられます。

**--target\_system\_name** オプションを使用して、システム名を上書きすることができます。

### コピーされるコンポーネント

以下の API プロダクトコンポーネントがコピーされます。

- 定義および設定
- メトリックおよびメソッド
- マッピングルール: これらはコピーされ、置き換えられます。
- アプリケーションプラン、課金ルール、および制限
- アプリケーションの使用に関するルール
- ポリシー
- バックエンド
- ActiveDocs

### 手順

- 以下のコマンドを入力して API プロダクトをコピーします。

```
3scale product copy [opts] -s <source_remote> -d <target_remote> <source_product>
```

3scale インスタンスには、リモート名または URL を指定することができます。



### 注記

1つのコマンドにつき1つの API プロダクトしかコピーすることはできません。複数のコマンドを使用して、複数のプロダクトをコピーすることができます。--**target\_system\_name** で異なる名前を指定して、同じプロダクトを複数回コピーすることができます。

API プロダクトのコピー時に、以下のオプションを使用します。

#### Options

- d --destination=<value>      3scale target instance: URL or remote name (required).
- s --source=<value>            3scale source instance: URL or remote name (required).
- t --target\_system\_name=<value>    Target system name: defaults to source system name.

以下のコマンド例は、--**target\_system\_name** で異なる値を指定して、API プロダクトを複数回コピーする方法を示しています。

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.13 3scale product copy [-t target_system_name] -s 3scale1 -d 3scale2 my_api_product_01
```

## 5.23. SSL および TLS に関する問題のトラブルシューティング

本セクションでは、Secure Sockets Layer/Transport Layer Security (SSL/TLS) に関する問題の解決方法について説明します。

自己署名 SSL 証明書に関連する問題が発生している場合、本セクションで説明されているようにリモートホスト証明書をダウンロードして使用することができます。典型的なエラーの例としては、**SSL certificate problem: self signed certificate** または **self signed certificate in certificate chain** があります。

### 手順

1. **openssl** を使用して、リモートホストの証明書をダウンロードします。以下に例を示します。

```
$ echo | openssl s_client -showcerts -servername self-signed.badssl.com -connect self-signed.badssl.com:443 2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > self-signed-cert.pem
```

2. **curl** を使用して、証明書が正常に機能していることを確認します。以下に例を示します。

```
$ SSL_CERT_FILE=self-signed-cert.pem curl -v https://self-signed.badssl.com
```

証明書が正しく機能している場合は、SSL エラーが表示されることはなくなります。証明書が正常に機能していない場合は、**-k** オプション (または long 形式の **--insecure**) を使用して **curl** コマンドの実行を試行します。これは、サーバーコネクションがセキュアでなくても、続行することを示しています。

3. **3scale** コマンドに **SSL\_CERT\_FILE** 環境変数を追加します。以下に例を示します。

```
$ podman run --env "SSL_CERT_FILE=/tmp/self-signed-cert.pem" -v $PWD/self-signed-cert.pem:/tmp/self-signed-cert.pem registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.13 3scale service list https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME}
```

この例では、Podman ボリュームを使用して、証明書ファイルをコンテナにマウントします。これは、このファイルが現在の **\$PWD** フォルダにあることを前提としています。

これ以外に、ベースイメージとして 3scale toolbox イメージを使用して専用の toolbox イメージを作成し、独自の信頼済み証明書ストアをインストールするアプローチもあります。

## 関連情報

- SSL 証明書の詳細については、[Red Hat Certificate System のドキュメント](#) を参照してください。
- Podman の詳細については、[Red Hat Enterprise Linux 8 コンテナの構築、実行、および管理](#) を参照してください。

## 第6章 3SCALE での API 環境のマッピング

API プロバイダーは、3scale 管理ポータルを通じて管理される API へのアクセスを提供します。続いて、API バックエンドを多くの環境にデプロイします。API バックエンド環境には、以下が含まれます。

- 開発、品質保証 (QA)、ステージング、および実稼働環境に使用されるさまざまな環境。
- API バックエンドの独自のセットを管理するチームまたは部門に使用されるさまざまな環境。

Red Hat 3scale API Management プロダクトは、単一の API または API のサブセットを表しますが、さまざまな API バックエンド環境のマッピングおよび管理にも使用されます。

3scale プロダクトの API 環境のマッピング方法に関しては、以下のセクションを参照してください。

- [環境ごとのプロダクト](#)
- [オンプレミス型 3scale インスタンス](#)
- [3scale の混合アプローチ](#)
- [3scale と APIcast ゲートウェイの組み合わせ](#)

### 6.1. 環境ごとのプロダクト

この方法では、API バックエンド環境ごとに個別の 3scale プロダクトを使用します。それぞれのプロダクトで、実稼働環境のゲートウェイとステージングゲートウェイを設定します。これにより、ゲートウェイ設定の変更を安全にテストし、API バックエンドと同様に実稼働設定にプロモートできます。

```
Production Product => Production Product APIcast gateway => Production Product API upstream
Staging Product => Staging Product APIcast gateway => Staging Product API upstream
```

API バックエンド環境のプロダクトを以下のように設定します。

- 環境用の API バックエンドのベース URL を使用して [バックエンドを作成](#)します。
- バックエンドパス / を使用して、環境のプロダクトに [バックエンドを追加](#)します。

#### 開発環境

- 開発バックエンドの作成
  - **名前:** Dev
  - **プライベートベース URL:** API バックエンドの URL
- Dev プロダクトの作成
  - **本番パブリックベース URL:** <https://dev-api-backend.yourdomain.com>
  - **ステージングパブリックベース URL:** <https://dev-api-backend.yourdomain.com>
  - バックエンドパス / を使用した [開発バックエンドの追加](#)

#### QA 環境

- QA バックエンドの作成
  - 名前: QA
  - プライベートベース URL: API バックエンドの URL
- QA プロダクトの作成
  - 本番パブリックベース URL: <https://qa-api-backend.yourdomain.com>
  - ステージングパブリックベース URL: <https://qa-api-backend.yourdomain.com>
  - バックエンドパス / を使用した [QA バックエンドの追加](#)

### 実稼働環境

- 実稼働環境用のバックエンドの作成
  - 名前: Prod
  - プライベートベース URL: API バックエンドの URL
- Prod プロダクトの作成
  - 本番パブリックベース URL: <https://prod-api-backend.yourdomain.com>
  - ステージング環境用の公開ベース URL: <https://prod-api-backend.yourdomain.com>
  - バックエンドパス / を使用した [実稼働バックエンドの追加](#)

### 関連情報

- 3scale プロダクトの詳細は、[First steps with 3scale](#) を参照してください。

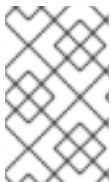
## 6.2. オンプレミス型 3SCALE インスタンス

オンプレミス型 3scale インスタンスの場合、API バックエンド環境を管理するために 3scale を設定する方法は複数あります。

- API バックエンド環境ごとに個別の 3scale インスタンス
- [マルチテナンシー](#) 機能を使用する単一の 3scale インスタンス

### 6.2.1. 環境ごとの 3scale インスタンスの分離

このアプローチでは、API バックエンド環境ごとに個別の 3scale インスタンスがデプロイされます。このアーキテクチャーの利点は、各環境が互いに分離されるため、共有するデータベースやその他のリソースがないことです。たとえば、ある環境で行われる負荷テストは、他の環境のリソースには影響しません。



#### 注記

このインストールの分離は上記のような利点がありますが、より多くの運用リソースとメンテナンスが必要になります。これらの追加リソースは、OpenShift 管理レイヤーで必要になりますが、3scale レイヤーで必要になるとは限りません。



## 6.2.2. 環境ごとの 3scale テナントの分離

この方法では、単一の 3scale インスタンスが使用されますが、マルチテナンシー機能は複数の API バックエンドをサポートするために使用されます。

以下の 2 つのオプションがあります。

- 単一のテナント内で、環境と 3scale プロダクト間の 1 対 1 のマッピングを作成します。
- 必要に応じて、テナントごとに 1 つ以上のプロダクトを使用して、環境とテナントの間に 1 対 1 のマッピングを作成します。
  - API バックエンド環境に対応するテナントが 3 つあります (dev-tenant、qa-tenant、prod-tenant)。このアプローチの利点は、環境を論理的に分離可能にしますが、共有物理リソースを使用できることです。



### 注記

API 環境を複数のテナントを持つ単一のインストールにマッピングするための最適なストラテジーを分析する場合、共有物理リソースを最終的に考慮する必要があります。

## 6.3. 3SCALE の混合アプローチ

[オンプレミス型 3scale インスタンス](#) で説明されている方法を組み合わせることができます。以下に例を示します。

- 実稼働用の別の 3scale インスタンス
- dev および qa の非実稼働環境用の独立したテナントを使用する別の 3scale インスタンス

## 6.4. 3SCALE と APICAST ゲートウェイの組み合わせ

オンプレミス型 3scale インスタンスの場合、API バックエンド環境を管理するために 3scale を設定する選択肢が 2 つあります。

- 3scale インストールごとに、ステージングおよび実稼働用の 2 つの組み込み APICast ゲートウェイがあります。
- 3scale が実行されている OpenShift クラスターの外部に [追加の APICast](#) ゲートウェイをデプロイします。

### 6.4.1. APICast の組み込みデフォルトゲートウェイ

APICast 組み込みゲートウェイを使用する場合、[3scale と APICast ゲートウェイの組み合わせ](#) で説明されている上記のアプローチを使用して設定された API バックエンドは自動的に処理されます。3scale マスター管理によりテナントが追加されると、実稼働環境およびステージングの組み込み APICast ゲートウェイでテナントのルートが作成されます。[マルチテナント対応サブドメインについて](#) を参照してください。

- `<API_NAME>-<TENANT_NAME>-apicast.staging.<WILDCARD_DOMAIN>`
- `<API_NAME>-<TENANT_NAME>-apicast.production.<WILDCARD_DOMAIN>`

したがって、異なるテナントにマッピングされた各 API バックエンド環境は、独自のルートを取得します。以下に例を示します。

- Dev <API\_NAME>-dev-apicast.staging.<WILDCARD\_DOMAIN>
- QA <API\_NAME>-qa-apicast.staging.<WILDCARD\_DOMAIN>
- Prod <API\_NAME>-prod-apicast.staging.<WILDCARD\_DOMAIN>

## 6.4.2. 追加の APIcast ゲートウェイ

追加の APIcast ゲートウェイは、3scale インスタンスが実行されているものとは異なる [OpenShift クラスタ](#) にデプロイされたものです。追加の APIcast ゲートウェイを設定して使用方法は複数あります。APIcast の起動時に使用される環境変数の値 **THREESCALE\_PORTAL\_ENDPOINT** は、追加の APIcast ゲートウェイの設定方法によって異なります。

API バックエンド環境ごとに個別の APIcast ゲートウェイを使用することができます。以下に例を示します。

```
DEV_APICAST -> DEV_TENANT ; DEV_APICAST started with  
THREESCALE_PORTAL_ENDPOINT = admin portal for DEV_TENANT  
QA_APICAST -> QA_TENANT ; QA_APICAST started with THREESCALE_PORTAL_ENDPOINT =  
admin portal for QA_APICAST  
PROD_APICAST -> PROD_TENANT ; PROD_APICAST started with  
THREESCALE_PORTAL_ENDPOINT = admin portal for PROD_APICAST
```

**THREESCALE\_PORTAL\_ENDPOINT** は、設定をダウンロードするために APIcast によって使用されます。API バックエンド環境にマッピングする各テナントは、個別の APIcast ゲートウェイを使用します。**THREESCALE\_PORTAL\_ENDPOINT** は、その API バックエンド環境に固有のすべてのプロダクト設定が含まれるテナントの管理ポータルに設定されます。

単一の APIcast ゲートウェイは、複数の API バックエンド環境と共に使用することができます。この場合、**THREESCALE\_PORTAL\_ENDPOINT** は [マスター管理ポータル](#) に設定されます。

### 関連情報

- [API プロバイダー](#) の詳細については、用語集を参照してください。
- 3scale [プロダクト](#) の詳細については、用語集を参照してください。

## 第7章 AUTOMATING API LIFECYCLE WITH 3SCALE TOOLBOX

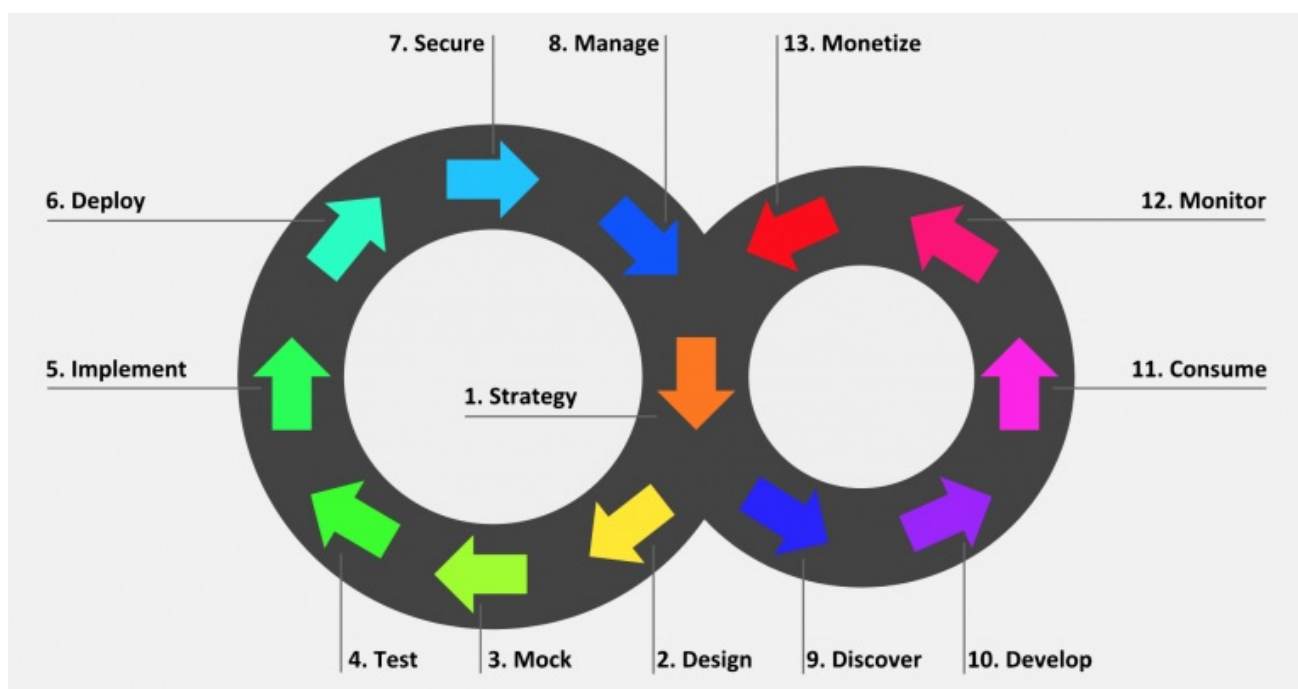
本トピックでは、Red Hat 3scale API Management での API ライフサイクルの概念について説明し、3scale toolbox コマンドにより API プロバイダーが Jenkins Continuous Integration/Continuous Deployment (CI/CD) パイプラインを使用してデプロイメントステージを自動化する方法を紹介いたします。ここでは、サンプルの Jenkins CI/CD パイプラインのデプロイ方法、3scale 共有ライブラリーを使用してカスタムの Jenkins パイプラインを作成する方法、およびカスタムパイプラインをゼロから作成する方法について説明します。

- 「API ライフサイクルステージの概要」
- 「サンプル Jenkins CI/CD パイプラインのデプロイ」
- 「3scale Jenkins 共有ライブラリーを使用したパイプラインの作成」
- 「Jenkinsfile を使用したパイプラインの作成」

### 7.1. API ライフサイクルステージの概要

API ライフサイクルは、API が作成されてから非推奨になるまでに必要なすべてのアクティビティについて説明するものです。3scale を使用すると、API プロバイダーはあらゆる API ライフサイクル管理を実施できるようになります。本セクションでは、API ライフサイクルの各ステージ、ならびにその目的および予想される結果について説明します。

以下の図は、左側に API プロバイダーベースのステージを、右側に API 利用者ベースのステージを示しています。



#### 注記

Red Hat は、現在 API プロバイダーサイクルの設計、実装、デプロイ、保護、および管理のフェーズ、ならびに API 利用者サイクルのすべてのフェーズをサポートしていません。

#### 7.1.1. API プロバイダーサイクル

API プロバイダーサイクルのステージは、API の詳細規定、開発、およびデプロイをベースとしています。以下に、各ステージの目的と成果を説明します。

表7.1 API プロバイダーライフサイクルのステージ

ステージ	目的	成果
1.ストラテジー	目的、リソース、ターゲットマーケット、タイムフレームを含む API の企業ストラテジーを決定し、計画を立てる。	目的を達成するための明確な計画と共に、企業ストラテジーが定義される。
2.設計	API 契約を早期に作成し、プロジェクト間の依存関係を解消し、フィードバックを収集して、リスクを下げ、市場に出すまでの時間を短縮する (たとえば、Apicurio Studio を使用)。	利用者向けの API 契約により、API で交換できるメッセージが定義される。API 利用者がフィードバックを提供している。
3.モック	実際の例と負荷を想定してさらに API 契約を規定し、API 利用者がこれを使用して実装を開始できるようにする。	モック API が稼働中で、実際の例を返す。例を想定した API 契約が完成する。
4.テスト	ビジネスを想定してさらに API 契約を規定し、開発した API のテストに使用できるようにする。	受け入れテストのセットが作成される。ビジネスを想定した API ドキュメントが完成する。
5.実装	Red Hat Fuse や希望の開発言語などのインテグレーションフレームワークを使用して、API を実装する。実装と API 契約を一致させる。	API が実装される。カスタム API 管理機能が必要な場合は、3scale APIcast ポリシーも開発される。
6.デプロイ	CI/CD パイプラインを 3scale toolbox で使用して、API インテグレーション、テスト、デプロイメント、および管理を自動化する。	CI/CD パイプラインにより、API が自動化された方法で実稼働環境に統合、テスト、デプロイ、および管理される。
7.保護	API が保護されるようにする (たとえば、セキュアな開発プラクティスと自動化されたセキュリティテストを使用)。	セキュリティガイドライン、プロセス、およびゲートが準備される。
8.管理	環境間の API プロモーション、バージョン管理、非推奨化、および廃止をまとめて管理する。	API をまとめて管理するためのプロセスとツールが準備される (たとえば、セマンティックバージョン管理により API の変更の違反を防止)。

## 7.1.2. API 利用者サイクル

API 利用者サイクルのステージは、API を利用するためのプロモーション、配布、および調整をベースとしています。以下に、各ステージの目的と成果を説明します。

表7.2 API 利用者ライフサイクルのステージ

ステージ	目的	成果
9.検出	API をサードパーティーの開発者、パートナー、および内部ユーザーにプロモーションする。	デベロッパーポータルが稼働中で、最新版のドキュメントがこのデベロッパーポータルに継続的にプッシュされる (たとえば、3scale ActiveDocs を使用)。
10.開発	サードパーティーの開発者、パートナー、および内部ユーザーが API をベースにアプリケーションを開発できるよう支援する。	デベロッパーポータルに、ベストプラクティス、ガイド、および推奨事項が含まれる。API 開発者がモックエンドポイントおよびテストエンドポイントにアクセスし、ソフトウェアを開発する。
11.利用	API 利用の増加を処理し、多数の API 利用者を管理する。	ステージングされたアプリケーションプランが利用でき、最新の価格と制限が継続的にプッシュされる。API 利用者が CI/CD パイプラインから API キーまたはクライアント ID/シークレットの生成を統合できる。
12.監視	API の健全性、品質、および開発者の関与について、実際の定量化されたフィードバックを収集する (たとえば、最初の Hello World! の時間のメトリックなど)。	監視システムが準備される。ダッシュボードに API の KPI (たとえば、稼働時間、分ごとのリクエスト数、レイテンシーなど) が表示される。
13.収益化	新しい収益を大規模に獲得する (このステージはオプション)。	たとえば、小規模な API 利用者を多数獲得することをターゲットにする場合、収益化が有効化され、利用者が使用量に基づいて自動的に課金される。

## 7.2. サンプル JENKINS CI/CD パイプラインのデプロイ

3scale toolbox による API ライフサイクルの自動化は、API ライフサイクルのデプロイメントステージが対象で、CI/CD パイプラインを使用して API 管理ソリューションを自動化することができます。本トピックでは、3scale toolbox を呼び出すサンプル Jenkins パイプラインをデプロイする方法を説明します。

- [「サンプル Jenkins CI/CD パイプライン」](#)
- [「ホスト型 3scale 環境の設定」](#)
- [「オンプレミス型 3scale 環境の設定」](#)

- 「OpenID Connect 向け Red Hat Single Sign-On のデプロイ」
- 「3scale toolbox のインストールおよびアクセスの有効化」
- 「API バックエンドのデプロイ」
- 「Self-managed APIcast インスタンスのデプロイ」
- 「サンプルパイプラインのインストールとデプロイ」
- 「3scale toolbox を使用した API ライフサイクル自動化の制約」

### 7.2.1. サンプル Jenkins CI/CD パイプライン

API ライフサイクルの自動化用に Jenkins パイプラインを作成してデプロイする方法の例として、以下のサンプルが Red Hat Integration リポジトリで提供されています。

表7.3 サンプル Jenkins 共有ライブラリーパイプライン

サンプルパイプライン	ターゲット環境	セキュリティー
<b>SaaS - API key</b>	ホスト型 3scale	API キー
<b>Hybrid - open</b>	Self-managed APIcast を使用するホスト型 3scale およびオンプレミス型 3scale	なし
<b>Hybrid - OpenID Connect</b>	Self-managed APIcast を使用するホスト型 3scale およびオンプレミス型 3scale	OpenID Connect (OIDC)
<b>Multi-environment</b>	Self-managed APIcast を使用する、開発、テスト、および実稼働環境のホスト型 3scale	API キー
<b>Semantic versioning</b>	Self-managed APIcast を使用する、開発、テスト、および実稼働環境のホスト型 3scale	API キー、なし、OIDC

これらのサンプルは、3scale toolbox を呼び出す 3scale Jenkins 共有ライブラリーを使用して、主要な API 管理機能を実証します。本トピックの設定手順を実施したら、各 [Red Hat Integration リポジトリのユースケース例](#) で提供される OpenShift テンプレートを使用してパイプラインをインストールすることができます。



#### 重要

サンプルのパイプラインおよびアプリケーションは、例としてのみ提供されています。ベースとなる API、CLI、およびサンプルパイプラインが活用するその他のインターフェイスは、Red Hat により完全にサポートされています。パイプラインに対して行った変更については、Red Hat による直接のサポートはありません。

### 7.2.2. ホスト型 3scale 環境の設定

ホスト型 3scale 環境の設定は、すべてのサンプル Jenkins CI/CD パイプラインで必要です。



### 注記

**SaaS - API key**、**Multi-environment**、および **Semantic versioning** のサンプルパイプラインは、ホスト型 3scale しか使用しません。**Hybrid - open** および **Hybrid - OIDC** のパイプラインは、オンプレミス型 3scale も使用します。[オンプレミス型 3scale 環境の設定](#) も参照してください。

### 前提条件

- Linux ワークステーションがある。
- ホスト型 3scale 環境が用意されている。
- OpenShift 3.11 クラスタがある。現在、OpenShift 4 はサポートされていません。
  - サポート対象設定の情報は、[Red Hat 3scale API Management のサポート対象構成](#) を参照してください。
- [OpenShift のドキュメント](#) で説明されているように、OpenShift ルーターでワイルドカードルートを有効にしておく。

### 手順

1. ホスト型 3scale 管理ポータルコンソールにログインします。
2. Account Management API への書き込みアクセス権限を設定して、新しいアクセストークンを生成します。
3. 後で使用できるように、生成されたアクセストークンを保存します。以下に例を示します。

```
export SAAS_ACCESS_TOKEN=123...456
```

4. 後で使用できるように、3scale テナントの名前を保存します。これは、管理ポータル URL の **admin.3scale.net** の前にある文字列です。以下に例を示します。

```
export SAAS_TENANT=my_username
```

5. 管理ポータルで **Audience > Accounts > Listing** の順に移動します。
6. **Developer** をクリックします。
7. **Developer Account ID** を保存します。これは、**/buyers/accounts/** に続く URL の最後の部分です。以下に例を示します。

```
export SAAS_DEVELOPER_ACCOUNT_ID=123...456
```

### 7.2.3. オンプレミス型 3scale 環境の設定

オンプレミス型 3scale 環境の設定は、**Hybrid - open** と **Hybrid - OIDC** のサンプル Jenkins CI/CD パイプラインでのみ必要です。





## 注記

これらの **Hybrid** サンプルパイプラインを使用する場合は、オンプレミス型 3scale 環境とホスト型 3scale 環境を設定する必要があります。[ホスト型 3scale 環境の設定](#) も参照してください。

## 前提条件

- Linux ワークステーションがある。
- オンプレミス型 3scale 環境を用意する。テンプレートを使用して OpenShift 上にオンプレミス型 3scale をインストールする方法については、[3scale のインストールに関するドキュメント](#) を参照してください。
- OpenShift 3.11 クラスタがある。現在、OpenShift 4 はサポートされていません。
  - サポート対象設定の情報は、[Red Hat 3scale API Management のサポート対象構成](#) を参照してください。
- [OpenShift のドキュメント](#) で説明されているように、OpenShift ルーターでワイルドカードルートを有効にしておく。

## 手順

1. オンプレミス型 3scale 管理ポータルコンソールにログインします。
2. Account Management API への書き込みアクセス権限を設定して、新しいアクセストークンを生成します。
3. 後で使用できるように、生成されたアクセストークンを保存します。以下に例を示します。

```
export SAAS_ACCESS_TOKEN=123...456
```

4. 後で使用できるように、3scale テナントの名前を保存します。

```
export ONPREM_ADMIN_PORTAL_HOSTNAME="$(oc get route system-provider-admin -o jsonpath='{.spec.host}')
```

5. ワイルドカードルートを定義します。

```
export OPENSIFT_ROUTER_SUFFIX=app.openshift.test # Replace me!
```

```
export APICAST_ONPREM_STAGING_WILDCARD_DOMAIN=onprem-staging.$OPENSIFT_ROUTER_SUFFIX
```

```
export APICAST_ONPREM_PRODUCTION_WILDCARD_DOMAIN=onprem-production.$OPENSIFT_ROUTER_SUFFIX
```



## 注記

**OPENSIFT\_ROUTER\_SUFFIX** の値を OpenShift ルーターの接尾辞に設定する必要があります (たとえば、**app.openshift.test**)。

6. ワイルドカードルートを既存のオンプレミス型 3scale インスタンスに追加します。



```
oc create route edge apicast-wildcard-staging --service=apicast-staging --
hostname="wildcard.$APICAST_ONPREM_STAGING_WILDCARD_DOMAIN" --insecure-
policy=Allow --wildcard-policy=Subdomain
```

```
oc create route edge apicast-wildcard-production --service=apicast-production --
hostname="wildcard.$APICAST_ONPREM_PRODUCTION_WILDCARD_DOMAIN" --
insecure-policy=Allow --wildcard-policy=Subdomain
```

7. 管理ポータルで **Audience > Accounts > Listing** の順に移動します。
8. **Developer** をクリックします。
9. **Developer Account ID** を保存します。これは、**/buyers/accounts/** に続く URL の最後の部分です。

```
export ONPREM_DEVELOPER_ACCOUNT_ID=5
```

#### 7.2.4. OpenID Connect 向け Red Hat Single Sign-On のデプロイ

**Hybrid - OpenID Connect (OIDC)** または **Semantic versioning** のサンプルパイプラインを使用している場合、本セクションの手順を実施して 3scale で Red Hat Single Sign-On (RH-SSO) をデプロイします。これは OIDC 認証に必要であり、両方のサンプルで使用されます。

##### 手順

1. [RH-SSO のドキュメント](#) で説明されているように、RH-SSO 7.3 をデプロイします。以下のコマンド例は、簡単なサマリーを提供します。

```
oc replace -n openshift --force -f https://raw.githubusercontent.com/jboss-container-
images/redhat-ssso-7-openshift-image/sso73-dev/templates/sso73-image-stream.json
```

```
oc replace -n openshift --force -f https://raw.githubusercontent.com/jboss-container-
images/redhat-ssso-7-openshift-image/sso73-dev/templates/sso73-x509-postgresql-
persistent.json
```

```
oc -n openshift import-image redhat-ssso73-openshift:1.0
```

```
oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default
```

```
oc new-app --template=sso73-x509-postgresql-persistent --name=sso -p
DB_USERNAME=sso -p SSO_ADMIN_USERNAME=admin -p DB_DATABASE=sso
```

2. 後で使用できるように、RH-SSO インストールのホスト名を保存します。

```
export SSO_HOSTNAME="$(oc get route sso -o jsonpath='{.spec.host}')
```

3. [3scale デベロッパーポータルのドキュメント](#) で説明されているように 3scale 向けに RH-SSO を設定します。
4. 後で使用できるように、レルム名、クライアント ID、およびクライアントシークレットを保存します。

```
export REALM=3scale
```

```
export CLIENT_ID=3scale-admin
export CLIENT_SECRET=123...456
```

### 7.2.5. 3scale toolbox のインストールおよびアクセスの有効化

本セクションでは、toolbox のインストール、リモート 3scale インスタンスの作成、および管理ポータルへのアクセスに使用されるシークレットのプロビジョニングを行う方法について説明します。

#### 手順

1. [3scale toolbox](#) で説明されているように、ローカルに 3scale toolbox をインストールします。
2. 適切な toolbox コマンドを実行して、3scale のリモートインスタンスを作成します。

#### ホスト型 3scale

```
3scale remote add 3scale-saas "https://$SAAS_ACCESS_TOKEN@$SAAS_TENANT-admin.3scale.net/"
```

#### オンプレミス型 3scale

```
3scale remote add 3scale-onprem
"https://$ONPREM_ACCESS_TOKEN@$ONPREM_ADMIN_PORTAL_HOSTNAME/"
```

3. 以下の OpenShift コマンドを実行して、3scale 管理ポータルとアクセストークンが含まれるシークレットをプロビジョニングします。

```
oc create secret generic 3scale-toolbox -n "$TOOLBOX_NAMESPACE" --from-file="$HOME/.3scalerc.yaml"
```

### 7.2.6. API バックエンドのデプロイ

本セクションでは、サンプルパイプラインで提供される API バックエンドの例をデプロイする方法について説明します。独自のパイプラインを作成してデプロイする場合、必要に応じて独自の API バックエンドを代わりに使用できます。

#### 手順

1. 以下のサンプルで使用するために、Beer Catalog API バックエンドの例をデプロイします。

- **SaaS - API key**
- **Hybrid - open**
- **Hybrid - OIDC**

```
oc new-app -n "$TOOLBOX_NAMESPACE" -i openshift/redhat-openjdk18-openshift:1.4
https://github.com/microcks/api-lifecycle.git --context-dir=/beer-catalog-demo/api-implementation --name=beer-catalog
```

```
oc expose -n "$TOOLBOX_NAMESPACE" svc/beer-catalog
```

- 後で使用できるように、Beer Catalog API のホスト名を保存します。

```
export BEER_CATALOG_HOSTNAME="$(oc get route -n "$TOOLBOX_NAMESPACE"
beer-catalog -o jsonpath='{.spec.host}')
```

- 以下のサンプルで使用するために、Red Hat Event API バックエンドの例をデプロイします。

- **Multi-environment**
- **Semantic versioning**

```
oc new-app -n "$TOOLBOX_NAMESPACE" -i openshift/nodejs:10
'https://github.com/nmasse-itix/rhte-api.git#085b015' --name=event-api

oc expose -n "$TOOLBOX_NAMESPACE" svc/event-api
```

- 後で使用できるように、Event API のホスト名を保存します。

```
export EVENT_API_HOSTNAME="$(oc get route -n "$TOOLBOX_NAMESPACE" event-api
-o jsonpath='{.spec.host}')
```

### 7.2.7. Self-managed APIcast インスタンスのデプロイ

本セクションは、ホスト型 3scale 環境で Self-managed APIcast インスタンスで使用するためのものです。本セクションの説明は、**SaaS - API key** 以外のすべてのサンプルパイプラインに該当します。

#### 手順

- ワイルドカードルートを定義します。

```
export APICAST_SELF_MANAGED_STAGING_WILDCARD_DOMAIN=saas-
staging.$OPENSIFT_ROUTER_SUFFIX

export APICAST_SELF_MANAGED_PRODUCTION_WILDCARD_DOMAIN=saas-
production.$OPENSIFT_ROUTER_SUFFIX
```

- Self-managed APIcast インスタンスをプロジェクトにデプロイします。

```
oc create secret generic 3scale-tenant --from-
literal=password=https://$SAAS_ACCESS_TOKEN@$SAAS_TENANT-admin.3scale.net

oc create -f https://raw.githubusercontent.com/3scale/apicast/v3.5.0/openshift/apicast-
template.yml

oc new-app --template=3scale-gateway --name=apicast-staging -p
CONFIGURATION_URL_SECRET=3scale-tenant -p CONFIGURATION_CACHE=0 -p
RESPONSE_CODES=true -p LOG_LEVEL=info -p CONFIGURATION_LOADER=lazy -p
APICAST_NAME=apicast-staging -p DEPLOYMENT_ENVIRONMENT=sandbox -p
IMAGE_NAME=registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.13

oc new-app --template=3scale-gateway --name=apicast-production -p
CONFIGURATION_URL_SECRET=3scale-tenant -p CONFIGURATION_CACHE=60 -p
RESPONSE_CODES=true -p LOG_LEVEL=info -p CONFIGURATION_LOADER=boot -p
APICAST_NAME=apicast-production -p DEPLOYMENT_ENVIRONMENT=production -p
```

```

IMAGE_NAME=registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.13

oc scale dc/apicast-staging --replicas=1

oc scale dc/apicast-production --replicas=1

oc create route edge apicast-staging --service=apicast-staging --
hostname="wildcard.$APICAST_SELF_MANAGED_STAGING_WILDCARD_DOMAIN" --
insecure-policy=Allow --wildcard-policy=Subdomain

oc create route edge apicast-production --service=apicast-production --
hostname="wildcard.$APICAST_SELF_MANAGED_PRODUCTION_WILDCARD_DOMAIN"
--insecure-policy=Allow --wildcard-policy=Subdomain

```

## 7.2.8. サンプルパイプラインのインストールとデプロイ

必要な環境を設定したら、各 [Red Hat Integration リポジトリのサンプルユースケース](#) 用に提供される OpenShift テンプレートを使用して、サンプルパイプラインをインストールしてデプロイすることができます。本セクションでは、**SaaS - API Key** のサンプルについてのみ説明します。

### 手順

1. 提供される OpenShift テンプレートを使用して、Jenkins パイプラインをインストールします。

```

oc process -f saas-usecase-apikey/setup.yaml \
  -p DEVELOPER_ACCOUNT_ID="$SAAS_DEVELOPER_ACCOUNT_ID" \
  -p PRIVATE_BASE_URL="http://$BEER_CATALOG_HOSTNAME" \
  -p NAMESPACE="$TOOLBOX_NAMESPACE" |oc create -f -

```

2. サンプルを以下のようにデプロイします。

```
oc start-build saas-usecase-apikey
```

### 関連情報

- [Red Hat Integration リポジトリのサンプルユースケース](#)

## 7.2.9. 3scale toolbox を使用した API ライフサイクル自動化の制約

本リリースでは、以下の制約が適用されます。

### OpenShift のサポート

サンプルパイプラインは OpenShift 3.11 でのみサポートされます。現在、OpenShift 4 はサポートされていません。サポート対象設定の情報は、[Red Hat 3scale API Management のサポート対象構成](#) を参照してください。

### アプリケーションの更新

- アプリケーション用 **3scale application apply** toolbox コマンドを使用して、アプリケーションの作成と更新の両方を行うことができます。作成コマンドは、アカウント、プラン、サービス、およびアプリケーションキーをサポートします。

- 更新コマンドは、アカウント、プラン、またはサービスに対する変更をサポートしません。変更が渡されると、パイプラインがトリガーされエラーは表示されませんが、これらのフィールドは更新されません。

## サービスのコピー

**3scale copy service** toolbox コマンドを使用してカスタムポリシーが設定されたサービスをコピーする場合、先に個別にカスタムポリシーをコピーする必要があります。

## 7.3. 3SCALE JENKINS 共有ライブラリーを使用したパイプラインの作成

本セクションでは、3scale toolbox を使用するカスタム Jenkins パイプラインを作成するためのベストプラクティスについて説明します。ここでは、アプリケーションの例をベースに、3scale Jenkins 共有ライブラリーを使用して toolbox を呼び出す Jenkins パイプラインを Groovy で記述する方法を説明します。詳細は、[Jenkins 共有ライブラリー](#) についてのドキュメントを参照してください。



### 重要

Red Hat では、Red Hat Integration リポジトリで提供される [サンプル Jenkins パイプライン](#) をサポートしています。

このパイプラインに対して行った変更については、Red Hat による直接のサポートはありません。独自の環境用に作成したカスタムのパイプラインはサポート対象外です。

## 前提条件

- [サンプル Jenkins CI/CD パイプライン](#) をデプロイする。
- API の OpenAPI 仕様ファイルを用意する。たとえば、[Apicurio Studio](#) を使用してこのファイルを生成できます。

## 手順

1. Jenkins パイプラインの先頭に以下の設定を追加して、パイプラインから 3scale 共有ライブラリーを参照します。

```
#!groovy

library identifier: '3scale-toolbox-jenkins@master',
  retriever: modernSCM([class: 'GitSCMSource',
    remote: 'https://github.com/rh-integration/3scale-toolbox-jenkins.git'])
```

2. **ThreescaleService** オブジェクトを保持するグローバル変数を宣言し、パイプラインの別ステージからそれを使用できるようにします。

```
def service = null
```

3. 関連情報がすべて含まれる **ThreescaleService** を作成します。

```
stage("Prepare") {
  service = toolbox.prepareThreescaleService(
    openapi: [ filename: "swagger.json" ],
    environment: [ baseSystemName: "my_service" ],
    toolbox: [ openshiftProject: "toolbox",
      destination: "3scale-tenant",
```

```

        secretName: "3scale-toolbox" ],
    service: [:],
    applications: [
      [ name: "my-test-app", description: "This is used for tests", plan: "test", account: "
<CHANGE_ME>" ]
    ],
    applicationPlans: [
      [ systemName: "test", name: "Test", defaultPlan: true, published: true ],
      [ systemName: "silver", name: "Silver" ],
      [ artefactFile: "https://raw.githubusercontent.com/my_username/API-Lifecycle-
Mockup/master/testcase-01/plan.yaml"],
    ]
  )

  echo "toolbox version = " + service.toolbox.getToolboxVersion()
}

```

- **openapi.filename** は、OpenAPI 仕様が含まれるファイルへのパスです。
  - **environment.baseSystemName** は、**environment.environmentName** と OpenAPI 仕様 **info.version** からの API メジャーバージョンをベースにした、最終的な **system\_name** の算出に使用されます。
  - **toolbox.openshiftProject** は、そこで Kubernetes ジョブが作成される OpenShift プロジェクトです。
  - **toolbox.secretName** は、[3scale toolbox のインストールおよびアクセスの有効化](#) に示すように、3scale toolbox 設定ファイルが含まれる Kubernetes シークレットの名前です。
  - **toolbox.destination** は、3scale toolbox リモートインスタンスの名前です。
  - **applicationPlans** は、**.yaml** ファイルを使用して、またはアプリケーションプランのプロパティ詳細を提示することで作成するアプリケーションプランのリストです。
4. 3scale でサービスをプロビジョニングするパイプラインステージを追加します。

```

stage("Import OpenAPI") {
  service.importOpenAPI()
  echo "Service with system_name ${service.environment.targetSystemName} created !"
}

```

5. アプリケーションプランを作成するステージを追加します。

```

stage("Create an Application Plan") {
  service.applyApplicationPlans()
}

```

6. テストアプリケーションを作成するグローバル変数とステージを追加します。

```

stage("Create an Application") {
  service.applyApplication()
}

```

7. インテグレーションテストを実行するステージを追加します。Hosted APIcast インスタンスを使用する場合、ステージング環境用の公開 URL を抽出するためにプロキシ定義を取得する必要があります。

```
stage("Run integration tests") {
    def proxy = service.readProxy("sandbox")
    sh """set -e +x
    curl -f -w "ListBeers: %{http_code}\n" -o /dev/null -s ${proxy.sandbox_endpoint}/api/beer -H
'api-key: ${service.applications[0].userkey}'
    curl -f -w "GetBeer: %{http_code}\n" -o /dev/null -s
${proxy.sandbox_endpoint}/api/beer/Weissbier -H 'api-key: ${service.applications[0].userkey}'
    curl -f -w "FindBeersByStatus: %{http_code}\n" -o /dev/null -s
${proxy.sandbox_endpoint}/api/beer/findByStatus/ available -H 'api-key:
${service.applications[0].userkey}'
    """
}
```

8. API を実稼働環境にプロモートするステージを追加します。

```
stage("Promote to production") {
    service.promoteToProduction()
}
```

#### 関連情報

- [Jenkinsfile を使用したパイプラインの作成](#)
- [3scale toolbox](#)

## 7.4. JENKINSFILE を使用したパイプラインの作成

本セクションでは、3scale toolbox を使用するカスタム **Jenkinsfile** を新規に Groovy で記述するためのベストプラクティスについて説明します。



### 重要

Red Hat では、Red Hat Integration リポジトリで提供される [サンプル Jenkins パイプライン](#) をサポートしています。

このパイプラインに対して行った変更については、Red Hat による直接のサポートはありません。独自の環境用に作成したカスタムのパイプラインはサポート対象外です。本セクションは参照用途としてのみ提供されています。

#### 前提条件

- [サンプル Jenkins CI/CD パイプラインをデプロイする](#)。
- API の OpenAPI 仕様ファイルを用意する。たとえば、[Apicurio Studio](#) を使用してこのファイルを生成できます。

#### 手順

1. 3scale toolbox を呼び出すためのユーティリティ関数を記述します。以下の例は、3scale toolbox を実行する Kubernetes ジョブを作成します。

```

#!groovy

def runToolbox(args) {
  def kubernetesJob = [
    "apiVersion": "batch/v1",
    "kind": "Job",
    "metadata": [
      "name": "toolbox"
    ],
    "spec": [
      "backoffLimit": 0,
      "activeDeadlineSeconds": 300,
      "template": [
        "spec": [
          "restartPolicy": "Never",
          "containers": [
            [
              "name": "job",
              "image": "registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.13",
              "imagePullPolicy": "Always",
              "args": [ "3scale", "version" ],
              "env": [
                [ "name": "HOME", "value": "/config" ]
              ],
              "volumeMounts": [
                [ "mountPath": "/config", "name": "toolbox-config" ],
                [ "mountPath": "/artifacts", "name": "artifacts" ]
              ]
            ]
          ],
          "volumes": [
            [ "name": "toolbox-config", "secret": [ "secretName": "3scale-toolbox" ] ],
            [ "name": "artifacts", "configMap": [ "name": "openapi" ] ]
          ]
        ]
      ]
    ]
  ]

  kubernetesJob.spec.template.spec.containers[0].args = args

  sh "rm -f -- job.yaml"
  writeYaml file: "job.yaml", data: kubernetesJob

  sh """set -e
oc delete job toolbox --ignore-not-found
sleep 2
oc create -f job.yaml
sleep 20 # Adjust the sleep duration to your server velocity
"""

  def logs = sh(script: "set -e; oc logs -f job/toolbox", returnStdout: true)
  echo logs
  return logs
}

```



## Kubernetes オブジェクトテンプレート

この関数は、Kubernetes オブジェクトテンプレートを使用して 3scale toolbox を実行するもので、必要に応じて調整できます。3scale toolbox CLI 引数を設定し、結果の Kubernetes ジョブ定義を YAML ファイルに記述し、toolbox の以前の実行をクリーンアップし、Kubernetes ジョブを作成して、待機します。

- 待機時間は、Pod が **Created** から **Running** 状態に移行するのに要する時間に一致するように、サーバー速度に合わせて調整することができます。このステップは、ポーリングループを使用して調整できます。
  - OpenAPI 仕様ファイルは、**openapi** という **ConfigMap** から取得されます。
  - 3scale 管理ポータルホスト名とアクセストークンは、[3scale toolbox のインストールおよびアクセスの有効化](#) に示すように、**3scale-toolbox** というシークレットから取得されます。
  - **ConfigMap** は、ステップ 3 でパイプラインによって作成されます。ただし、シークレットはすでにパイプライン外にプロビジョニングされており、セキュリティを強化するロールベースのアクセス制御 (RBAC) の対象です。
2. Jenkins パイプラインステージで 3scale toolbox で使用するグローバル環境変数を定義します。以下に例を示します。

### ホスト型 3scale

```
def targetSystemName = "saas-apikey-usecase"
def targetInstance = "3scale-saas"
def privateBaseURL = "http://echo-api.3scale.net"
def testUserKey = "abcdef1234567890"
def developerAccountId = "john"
```

### オンプレミス型 3scale

Self-managed APIcast またはオンプレミス型 3scale のインストールを使用する場合、さらに 2 つの変数を宣言する必要があります。

```
def publicStagingBaseURL = "http://my-staging-api.example.test"
def publicProductionBaseURL = "http://my-production-api.example.test"
```

変数の説明は、以下のとおりです。

- **targetSystemName**: 作成されるサービスの名前
- **targetInstance**: この変数は、[3scale toolbox のインストールおよびアクセスの有効化](#) で作成された 3scale リモートインスタンスの名前と一致します。
- **privateBaseURL**: API バックエンドのエンドポイントホスト
- **testUserKey**: インテグレーションテストの実行に使用されるユーザー API キー。これは、例のようにハードコーディングされる場合と、HMAC 機能から生成される場合があります。
- **developerAccountId**: テストアプリケーションが作成されるターゲットアカウントの ID
- **publicStagingBaseURL**: 作成されるサービスのステージング環境用公開ベース URL

- **publicProductionBaseURL**: 作成されるサービスの実稼働環境用公開ベース URL
3. 以下のように、OpenAPI 仕様ファイルを取得して OpenShift で **ConfigMap** としてプロビジョニングするパイプラインステージを追加します。

```
node() {
  stage("Fetch OpenAPI") {
    sh """set -e
    curl -sfk -o swagger.json https://raw.githubusercontent.com/microcks/api-
lifecycle/master/beer-catalog-demo/api-contracts/beer-catalog-api-swagger.json
    oc delete configmap openapi --ignore-not-found
    oc create configmap openapi --from-file="swagger.json"
    """
  }
}
```

4. 3scale toolbox を使用して API を 3scale にインポートするパイプラインステージを追加します。

### ホスト型 3scale

```
stage("Import OpenAPI") {
  runToolbox([ "3scale", "import", "openapi", "-d", targetInstance, "/artifacts/swagger.json", "--
override-private-base-url=${privateBaseURL}", "-t", targetSystemName ])
}
```

### オンプレミス型 3scale

Self-managed APIcast またはオンプレミス型 3scale のインストールを使用する場合、ステージング環境と実稼働環境の公開ベース URL のオプションも指定する必要があります。

```
stage("Import OpenAPI") {
  runToolbox([ "3scale", "import", "openapi", "-d", targetInstance, "/artifacts/swagger.json", "--
override-private-base-url=${privateBaseURL}", "-t", targetSystemName, "--production-public-
base-url=${publicProductionBaseURL}", "--staging-public-base-
url=${publicStagingBaseURL}" ])
}
```

5. toolbox を使用して 3scale のアプリケーションプランとアプリケーションを作成するパイプラインステージを追加します。

```
stage("Create an Application Plan") {
  runToolbox([ "3scale", "application-plan", "apply", targetInstance, targetSystemName, "test",
"-n", "Test Plan", "--default" ])
}
```

```
stage("Create an Application") {
  runToolbox([ "3scale", "application", "apply", targetInstance, testUserKey, "--
account=${developerAccountId}", "--name=Test Application", "--description=Created by
Jenkins", "--plan=test", "--service=${targetSystemName}" ])
}
```

```
stage("Run integration tests") {
  def proxyDefinition = runToolbox([ "3scale", "proxy", "show", targetInstance,
targetSystemName, "sandbox" ])
}
```

```
def proxy = readJSON text: proxyDefinition
proxy = proxy.content.proxy

sh """set -e
echo "Public Staging Base URL is ${proxy.sandbox_endpoint}"
echo "userkey is ${testUserKey}"
curl -vfk ${proxy.sandbox_endpoint}/beer -H 'api-key: ${testUserKey}'
curl -vfk ${proxy.sandbox_endpoint}/beer/Weissbier -H 'api-key: ${testUserKey}'
curl -vfk ${proxy.sandbox_endpoint}/beer/findByStatus/available -H 'api-key: ${testUserKey}'
"""
}
```

6. toolbox を使用して API を実稼働環境にプロモートするステージを追加します。

```
stage("Promote to production") {
    runToolbox([ "3scale", "proxy", "promote", targetInstance, targetSystemName ])
}
```

## 関連情報

- [Jenkinsfile を使用したパイプラインの作成](#)
- [3scale toolbox](#)

## 第8章 3SCALE OPERATOR を使用した 3SCALE の設定とプロビジョニング

3scale の管理者は、3scale Operator を使用して 3scale サービスを設定し、3scale リソースをプロビジョニングすることができます。OpenShift Container Platform(OCP) ユーザーインターフェイスで Operator を使用します。Operator の使用は、管理ポータルで、または 3scale 内部 API を使用して 3scale を設定およびプロビジョニングする代わりとなります。

3scale Operator を使用してサービスを設定するか、リソースをプロビジョニングする場合、そのサービスまたはリソースを更新する唯一の方法は、そのカスタムリソース (CR) を更新することです。新しいサービスおよびリソースが管理ポータルに表示される間は、管理ポータルでサービスまたはリソースを更新したり、内部 3scale API を使用して更新したりすることはできません。更新しようとする、Operator は更新を元に戻し、CR はそのままになります。



### 重要

3scale operator の機能は、テクノロジープレビューの機能としてのみ提供されます。テクノロジープレビューの機能は、Red Hat の実稼働環境のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

本章では、Operator アプリケーションの機能の仕組みおよび Operator を使用したカスタムリソースのデプロイ方法について説明します。

- [最初の 3scale プロダクトおよびバックエンド](#)
- [製品の APIcast 設定のプロモート](#)
- [capabilities に関連するバックエンドカスタムリソース](#)
- [capabilities に関連するプロダクトカスタムリソース](#)
- [テナントカスタムリソース](#)
- [開発者アカウントのカスタムリソース](#)

さらに、3scale Operator を使用する場合の機能制限に関する情報があります。

### 8.1. 一般的な前提条件

3scale Operator を使用して 3scale を設定およびプロビジョニングするには、以下の要素が必要です。

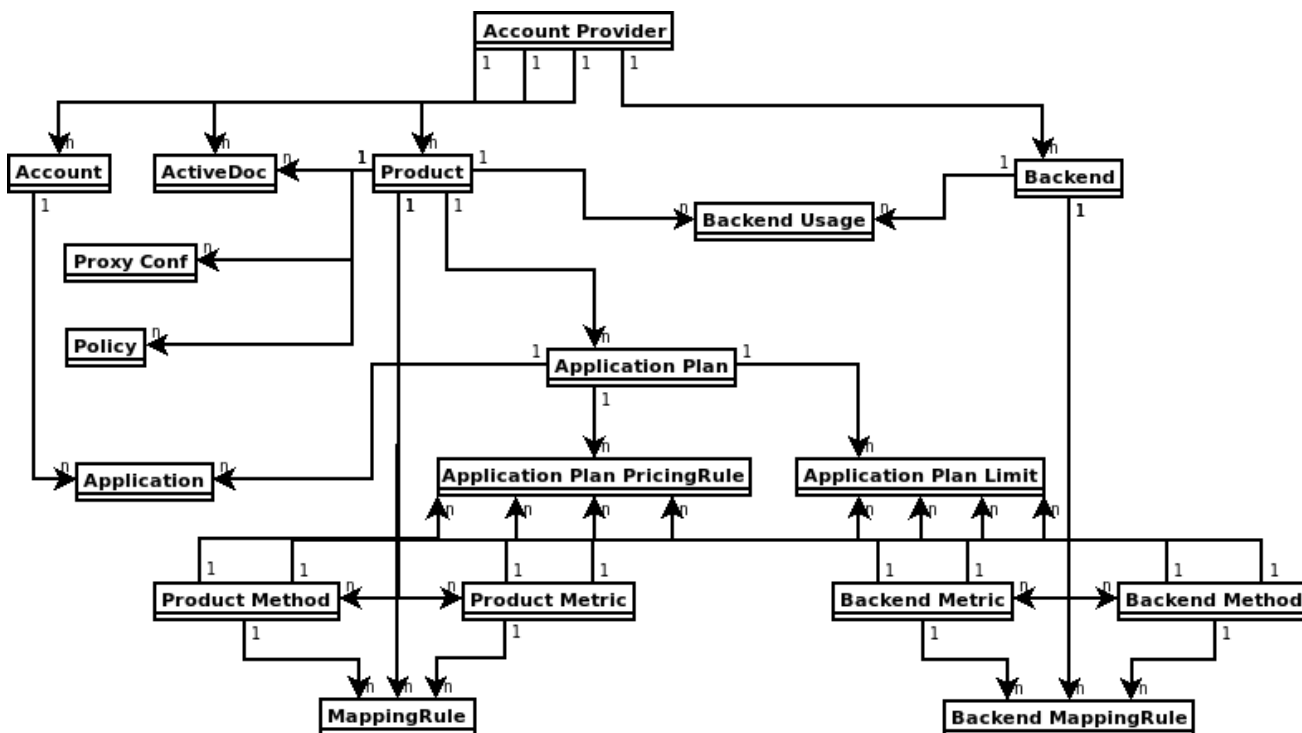
- [オンプレミス型 3scale 2.13 インスタンスの管理者権限を持つユーザーアカウント](#)
- [3scale operator がインストールされている](#)
- OpenShift Container Platform 4 および OpenShift クラスターの管理者権限を持つユーザーアカウント
  - サポート対象設定の情報は、[Red Hat 3scale API Management のサポート対象構成](#) を参照してください。

## 8.2. 3SCALE OPERATOR を使用したアプリケーション CAPABILITIES

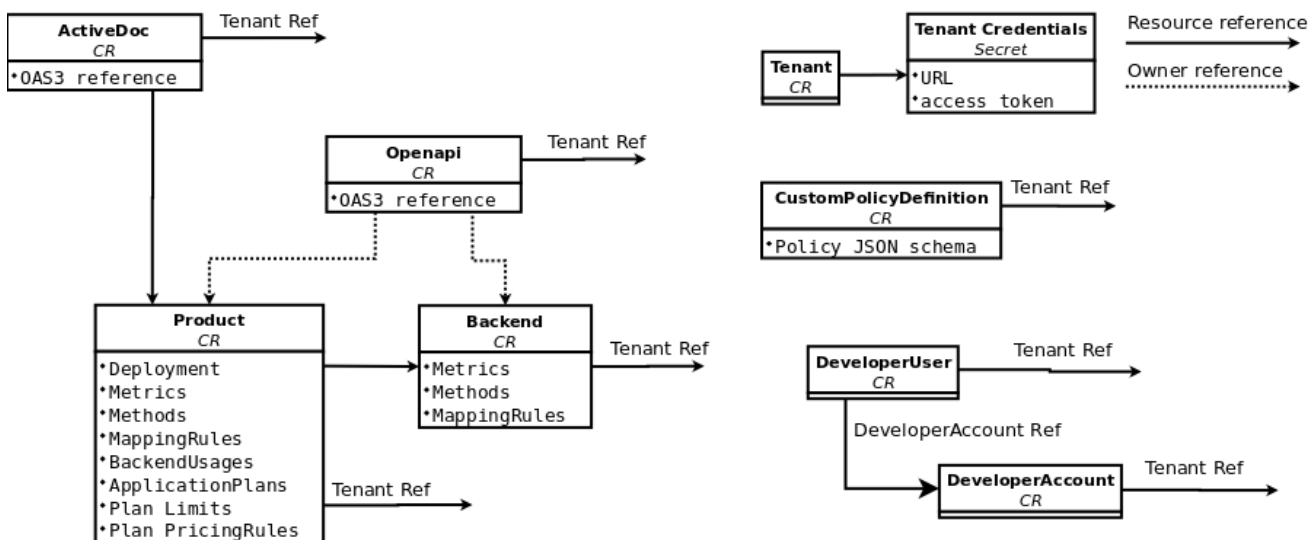
3scale operator には、以下の機能が含まれています。

- ベースとなる Red Hat 3scale API Management ソリューションとの対話を可能にする。
- OpenShift からのカスタムリソースを宣言的に使用して 3scale アプリケーションを管理する。

以下の図は、OpenShift カスタムリソースを宣言的に使用して管理することのできる 3scale エンティティおよび関係を示しています。プロダクトには1つまたは複数のバックエンドが含まれます。プロダクトレベルでは、アプリケーション、アプリケーションプラン、およびマッピングルールを設定できます。バックエンドレベルでは、各バックエンドのメトリクス、メソッド、およびマッピングルールを設定できます。



以下の図に示すように、3scale operator は、カスタムリソース定義とその関係を提供します。



## 8.3. 最初の 3SCALE プロダクトおよびバックエンドのデプロイ

新しく作成したテナントで OpenShift Container Platform を使用して、最初の 3scale プロダクトおよびバックエンドを最低限必要な設定でデプロイします。

## 前提条件

[一般的な前提条件](#) に記載の条件と同じインストール要件が適用されます。ただし、以下の考慮事項に注意してください。

- 3scale アカウントは、稼働用の OpenShift namespace のローカルとするか、リモートインストールにすることができます。
- このアカウントから必要なパラメーターは、3scale 管理 URL アドレスとアクセストークンです。

## 手順

1. 3scale 管理ポータルからのクレデンシャルを使用して、3scale プロバイダーアカウントのシークレットを作成します。例: **adminURL=https://3scale-admin.example.com** および **token=123456**。

```
oc create secret generic threescale-provider-account --from-literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
```

2. アップストリーム API URL を使用して 3scale バックエンドを設定します。
  - a. 以下の内容を含む YAML ファイルを作成します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend1
spec:
  name: "Operated Backend 1"
  systemName: "backend1"
  privateBaseURL: "https://api.example.com"
```

- ファイルを作成すると、operator はステップが成功したかどうかを確認します。
- Backend カスタムリソースのフィールドおよび設定可能な値の詳細は、[Backend CRD field Reference](#) を参照してください。

- b. カスタムリソースを作成します。

```
oc create -f backend1.yaml
```

3. 3scale プロダクトを設定します。

- a. 前のステップで作成したバックエンドに適用したすべてのデフォルト設定でプロダクトを作成します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
```

```
systemName: "operatedproduct1"
backendUsages:
  backend1:
    path: /
```

- ファイルを作成すると、operator はステップが成功したかどうかを確認します。
- Product カスタムリソースのフィールドおよび設定可能な値の詳細は、[Product CRD field Reference](#) を参照してください。

b. カスタムリソースを作成します。

```
oc create -f product1.yaml
```

4. 作成したカスタムリソースが 3scale インスタンスに反映されるのに数秒かかります。リソースが同期されるタイミングを確認するには、以下のいずれかの方法を選択できます。

- オブジェクトのステータス フィールドを確認する。
- **oc wait** コマンドを使用する。

```
oc wait --for=condition=Synced --timeout=-1s backend/backend1
oc wait --for=condition=Synced --timeout=-1s product/product1
```

## 8.4. 製品の APICAST 設定のプロモート

3scale operator を使用して、製品の APICAST 設定をステージングまたは実稼働に昇格させることができます。**ProxyConfigPromote** カスタムリソース (CR) は、最新の APICAST 設定をステージング環境にプロモートします。必要に応じて、**ProxyConfigPromote** CR を設定して、運用環境にも昇格させることができます。



### 注記

**ProxyConfigPromote** オブジェクトは、作成されたときにのみ有効になります。作成後、それらの更新は調整されません。

### 前提条件

以下を含む、[一般的な前提条件](#) に記載されているものと同じインストール要件:

- [製品 CR](#) を作成済みである。

### 手順

1. 次の内容で YAML ファイルを作成して保存します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
```

APIcast 設定を本番環境にプロモートするには、オプションのフィールド **spec.production** を **true** に設定します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
  production: true
```

昇格が成功した後に **ProxyConfigPromote object** を削除するには、オプションのフィールド **spec.deleteCR** を **true** に設定します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
  deleteCR: true
```

2. ファイルのステータス条件を確認するには、次のコマンドを入力します。

```
oc get proxyconfigpromote proxyconfigpromote-sample -o yaml
```

- a. 出力には、ステータスが **Ready** であることが示されます。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
status:
  conditions:
  - lastTransitionTime: "2022-10-28T11:35:19Z"
    status: "True"
    type: Ready
```

3. カスタムリソースを作成します。

```
oc create -f proxyconfigpromote-sample.yaml
```

- この例では、出力は以下のようになります。

```
proxyconfigpromote.capabilities.3scale.net/proxyconfigpromote-sample created
```

## 関連情報

- [ProxyConfigPromote CRD リファレンス](#)



## 8.5. 3SCALE OPERATOR が、カスタムリソースのリンク先となるテナントを識別する方法

3scale カスタムリソース (CR) をデプロイして、さまざまな 3scale オブジェクトを管理できます。3scale CR は、1つのテナントにのみリンクします。

3scale operator が 3scale と同じ namespace にインストールされている場合には、3scale CR がその 3scale インスタンスのデフォルトテナントにリンクされるのがデフォルトの動作です。3scale CR を異なるテナントにリンクするには、以下のいずれかを行います。

- 3scale CR が含まれる namespace に **threescale-provider-account** シークレットを作成します。3scale CR のデプロイ時に、Operator はこのシークレットを読み取り、CR がリンクするテナントを特定します。Operator がこのシークレットを使用するには、以下のいずれかが true である必要があります。
  - 3scale CR は **spec.providerAccountRef** フィールドを null として指定します。
  - 3scale CR は **spec.providerAccountRef** フィールドを省略します。**threescale-provider-account** シークレットは、CR のリンク先のテナントを識別します。シークレットには、URL の形式で 3scale インスタンスへの参照が含まれ、トークン形式でその 3scale インスタンス内のテナントにアクセスするためのクレデンシャルが含まれている必要があります。以下に例を示します。

```
oc create secret generic threescale-provider-account --from-literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
```

**threescale-provider-account** シークレットは、HTTP 接続が利用可能であれば、任意の 3scale インスタンスのすべてのテナントを特定できます。つまり、3scale CR および CR がリンクするテナントを含む 3scale インスタンスは、異なる namespace にあるか、異なる OpenShift クラスタにある可能性があります。

- 3scale CR で、**spec.providerAccountRef** を指定し、テナントを識別する OpenShift シークレットへのローカル参照の名前に設定します。以下の 3scale **DeveloperAccount** CR の例では、**mytenant** がシークレットです。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: DeveloperAccount
metadata:
  name: developeraccount-simple-sample
spec:
  orgName: Ecorp
  providerAccountRef:
    name: mytenant
```

シークレットでは、以下が行われます。

- **AdminURL** は、任意の namespace にある 3scale インスタンスの URL を指定します。
- **token** は、その 3scale インスタンス内の1つのテナントにアクセスするためのクレデンシャルを指定します。このテナントは、デフォルトのテナントまたはそのインスタンス内の他のテナントに指定できます。通常、テナントカスタムリソースをデプロイするときに、このシークレットを作成します。以下に例を示します。

```
apiVersion: v1
```

```

kind: Secret
metadata:
  name: mytenant
type: Opaque
stringData:
  adminURL: https://my3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

```

3scale Operator が、CR がリンクするテナントを特定できない場合、Operator はエラーメッセージを生成します。

## 8.6. 3SCALE OPENAPI カスタムリソースのデプロイ

**OpenAPI** カスタムリソース (CR) は、開発者ポータルでの ActiveDocs に使用できる OpenAPI Specification (OAS) ドキュメントをインポートする 1 つの方法です。OAS は、API に対して使用できるプログラミング言語が特定のものだけに限定されないようにする規格です。人間とコンピューターは、ソースコードのアクセス、ドキュメント、またはネットワークトラフィックの検査なしに API プロダクトの機能をより簡単に理解することができます。

### 前提条件

- オンプレミス型 3scale 2.13 インスタンスの管理者権限を持つユーザーアカウント
- API を定義する OAS ドキュメント
- **OpenAPI** CR がテナントにリンクする方法に関する理解

### 8.6.1. シークレットから OAS ドキュメントをインポートする 3scale OpenAPI カスタムリソースのデプロイ

3scale [バックエンド](#) および [プロダクト](#) を作成できるように、**OpenAPI** カスタムリソース (CR) をデプロイします。



#### 注記

Operator はシークレットのコンテンツのみを読み取ります。Operator はシークレットのフィールド名を読み取りません。

### 前提条件

3scale Operator が [カスタムリソースのリンク先となるテナントを識別する方法](#) を理解している。

### 手順

1. OAS ドキュメントが含まれるシークレットを定義します。たとえば、以下の内容で **myoasdoc1.yaml** を作成することができます。

```

openapi: "3.0.2"
info:
  title: "some title"
  description: "some description"
  version: "1.0.0"
paths:
  /pet:

```

```

get:
  operationId: "getPet"
  responses:
    405:
      description: "invalid input"

```

- シークレットを作成します。以下に例を示します。

```

$ oc create secret generic myoasdoc1 --from-file myoasdoc1.yaml

secret/myoasdoc1 created

```

- OpenAPI** CR を定義します。OAS ドキュメントが含まれるシークレットへの参照を指定するようにしてください。たとえば、**myopenapicr1.yaml** ファイルを作成することができます。

```

apiVersion: capabilities.3scale.net/v1beta1
kind: OpenAPI
metadata:
  name: myopenapicr1
spec:
  openapiRef:
    secretRef:
      name: myoasdoc1

```

- 定義したばかりのリソースを作成します。以下に例を示します。

```

$ oc create -f myopenapicr1.yaml

```

この例では、出力は以下のようになります。

```

openapi.capabilities.3scale.net/myopenapicr1 created

```

### 8.6.2. 3scale OpenAPI カスタムリソース定義の機能

**OpenAPI** カスタムリソース定義 (CRD) のデプロイメント機能に関する知識は、3scale プロダクトの設定、バックエンド、およびその後の開発者ポータル用の ActiveDocs の作成に役立ちます。

- OAS ドキュメントは、以下から読み込むことができます。
  - Kubernetes シークレット
  - http および https 形式の両方の URL
- OAS ドキュメントでは、**info.title** 設定は 215 文字を超えることができません。Operator はこの設定を使用して、長さの制限がある OpenShift オブジェクト名を作成します。
- サーバリストの最初の **servers[0].url** 要素のみがプライベート URL として解析されます。OpenAPI Specification(OAS) は、**servers[0].url** 要素の **basePath** コンポーネントを使用しません。
- **OpenAPI** CRD は単一の最上位のセキュリティー要件をサポートしますが、運用レベルのセキュリティーはサポートしません。
- **OpenAPI** CRD は **apiKey** セキュリティースキームをサポートします。

## 関連情報

- [capabilities](#) に関連するプロダクトカスタムリソース
- [OpenAPI CRD Reference](#)
- [Object Names and IDs](#)

### 8.6.3. OpenAPI カスタムリソースを定義する際のインポートルール

インポートルールは、3scale デプロイメントの OpenAPI ドキュメントを設定する際に、OpenAPI Specification(OAS) が 3scale とどのように機能するかを指定します。

#### プロダクト名

デフォルトのプロダクトシステム名は、OpenAPI ドキュメントの **info.title** フィールドから取得されます。OpenAPI ドキュメントのプロダクト名を上書きするには、**OpenAPI** カスタムリソース (CR) の **spec.productSystemName** フィールドを指定します。

#### プライベートベース URL

プライベートベース URL は **OpenAPI CR servers[0].url** フィールドから読み込まれます。**OpenAPI CR** の **spec.privateBaseURL** フィールドを使用して、これを上書きできます。

#### 3scale メソッド

インポートされた OpenAPI ドキュメントで定義される各操作は、プロダクトレベルで1つの 3scale メソッドに変換されます。メソッド名は、操作オブジェクトの **operationId** フィールドから読み取られます。

#### 3scale のマッピングルール

インポートされた OpenAPI ドキュメントで定義される各操作は、プロダクトレベルで1つの 3scale マッピングルールに変換されます。以前の既存のマッピングルールは **OpenAPI CR** でインポートされたルールに置き換えられました。

OpenAPI ドキュメントでは、**paths** オブジェクトは動詞およびパターンプロパティのマッピングルールを提供します。3scale メソッドは **operationId** に応じて関連付けられます。

delta の値は **1** にハードコーディングされます。

デフォルトでは、**Strict マッチング** ポリシーが設定されます。マッチングポリシーは、**OpenAPI CRD** の **spec.PrefixMatching** フィールドを使用して、**接頭辞マッチング** に切り替えることができます。

#### 認証

1つのトップレベルのセキュリティ要件のみがサポートされます。操作レベルのセキュリティ要件はサポートされていません。

サポートされるセキュリティスキームは **apiKey** です。

**apiKey** セキュリティスキームタイプ:

- **クレデンシャルの場所** は、セキュリティスキームオブジェクトの OpenAPI ドキュメント **in** フィールドから読み込まれます。
- **認証ユーザー キー** は、セキュリティスキームオブジェクトの OpenAPI ドキュメント **name** フィールドから読み込まれます。

以下は、**apiKey** セキュリティー要件のある OAS 3.0.2 の例の一部です。

```
openapi: "3.0.2"
security:
  - petstore_api_key: []
components:
  securitySchemes:
    petstore_api_key:
      type: apiKey
      name: api_key
      in: header
```

OpenAPI ドキュメントがセキュリティ要件を指定しない場合、以下が適用されます。

- プロダクト認証が **apiKey** に設定されます。
- クレデンシャルの場所 はデフォルトで 3scale の値 **As query parameters (GET) or body parameters (POST/PUT/DELETE)** に設定されます。
- Auth ユーザー キーはデフォルトで 3scale の値 **user\_key** に設定されます。

3scale 認証セキュリティ は、**OpenAPI CRD** の **spec.privateAPIHostHeader** フィールドおよび **spec.privateAPISecretToken** フィールドを使用して設定できます。

### ActiveDocs

3scale ActiveDoc は作成されていません。

### 3scale プロダクトポリシーチェーン

3scale ポリシーチェーンは、3scale が作成するデフォルトのポリシーチェーンです。

### 3scale デプロイメントモード

デフォルトでは、設定した 3scale デプロイメントモードは APIcast 3scale により管理されます。しかし、**spec.productionPublicBaseURL** または **spec.stagingPublicBaseURL**、あるいは両方のフィールドが **OpenAPI CR** にある場合、プロダクトのデプロイメントモードは APIcast で自己管理されます。

カスタム公開ベース URL を持つ **OpenAPI CR** の例:

```
apiVersion: capabilities.3scale.net/v1beta1
kind: OpenAPI
metadata:
  name: openapi1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-
Specification/master/examples/v3.0/petstore.yaml"
  productionPublicBaseURL: "https://production.my-gateway.example.com"
  stagingPublicBaseURL: "https://staging.my-gateway.example.com"
```

## 8.6.4. URL から OAS ドキュメントをインポートする 3scale OpenAPI カスタムリソースのデプロイ

指定した URL から OAS ドキュメントをインポートする **OpenAPI** カスタムリソースをデプロイできます。その後、この OAS ドキュメントを、デベロッパーポータルで API の ActiveDocs の基礎として使用できます。

## 前提条件

- 同じ namespace にある 3scale インスタンスのデフォルトのテナントにリンクしない **OpenAPI** カスタムリソースを作成する場合、**OpenAPI** CR が含まれる namespace には、**OpenAPI** CR がリンクするテナントを特定するシークレットが含まれます。シークレットの名前は以下のいずれかになります。
  - **threescale-provider-account**
  - ユーザー定義

このシークレットには、3scale インスタンスの URL と、3scale インスタンスの1つのテナントにアクセスするためのクレデンシャルが含まれるトークンが含まれます。

## 手順

1. OpenShift アカウントで、**Operators > Installed operators** に移動します。
2. 3scale operator をクリックします。
3. **YAML** タブを選択します。
4. **OpenAPI** カスタムリソースを作成します。以下に例を示します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: OpenAPI
metadata:
  name: openapi1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-
Specification/master/examples/v3.0/petstore.yaml"
  providerAccountRef:
    name: mytenant
```

5. **Save** をクリックします。3scale operator が **OpenAPI** CR を作成するまでに数秒かかります。

## 検証

1. OpenShift の **3scale Product Overview** ページで、**Synced** 状態が **True** とマークされていることを確認します。
2. 3scale アカウントに移動します。
3. OAS ドキュメントが存在することを確認します。上記の例では、**openapi1** という名前の新しい OAS ドキュメントが表示されます。

### 8.6.5. 関連情報

- [OpenAPI Specification \(OAS\)](#)
- [OpenAPI CRD Reference](#)

- [オプションテナントカスタムリソースのデプロイ](#)

## 8.7. 3SCALE ACTIVEDOC カスタムリソースのデプロイ

Red Hat 3scale API Management ActiveDocs は、[OpenAPI Specification](#) に準拠する RESTful Web サービスを定義する API 定義ドキュメントをベースとしています。**ActiveDoc** カスタムリソース (CR) は、開発者ポータル ActiveDocs に使用できる OpenAPI Specification (OAS) ドキュメントをインポートする 1 つの方法です。OAS は、API に対して使用できるプログラミング言語が特定のものだけに限定されないようにする規格です。人間とコンピューターは、ソースコードのアクセス、ドキュメント、またはネットワークトラフィックの検査なしに API プロダクトの機能をより簡単に理解することができます。

### 前提条件

- オンプレミス型 3scale 2.13 インスタンスの管理者権限を持つユーザーアカウント
- API を定義する OAS ドキュメント
- **ActiveDoc** CR がテナントにリンクする方法を理解している。

### 8.7.1. シークレットから OAS ドキュメントをインポートする 3scale ActiveDoc カスタムリソースのデプロイ

3scale [バックエンド](#) および [プロダクト](#) を作成できるように、**ActiveDoc** カスタムリソース (CR) をデプロイします。



#### 注記

Operator はシークレットのコンテンツのみを読み取ります。Operator はシークレットのフィールド名を読み取りません。たとえば、データは **key: value** のペアで設定され、**value** はファイルの内容を表し、**key** はファイル名になります。ファイル名は、ActiveDoc CRD のこのコンテキストで Operator によって無視されます。Operator はファイルの内容のみを読み取ります。

### 前提条件

- [3scale Operator がカスタムリソースのリンク先となるテナントを識別する方法](#) を理解している。
- OAS (OpenAPI Specification) ドキュメントが含まれるシークレットを定義します。たとえば、以下の内容で **myoasdoc1.yaml** を作成することができます。

```
openapi: "3.0.2"
info:
  title: "some title"
  description: "some description"
  version: "1.0.0"
paths:
  /pet:
    get:
      operationId: "getPet"
      responses:
        405:
          description: "invalid input"
```

## 手順

1. シークレットを作成します。以下に例を示します。

```
$ oc create secret generic myoasdoc1 --from-file myoasdoc1.yaml
secret/myoasdoc1 created
```

2. **ActiveDoc** CR を定義します。OAS ドキュメントが含まれるシークレットへの参照を指定するようにしてください。たとえば、**myactivedoccr1.yaml** ファイルを作成できます。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ActiveDoc
metadata:
  name: myactivedoccr1
spec:
  name: "Operated ActiveDoc From secret"
  activeDocOpenAPIRef:
    secretRef:
      name: myoasdoc1
```

3. 定義したばかりのリソースを作成します。以下に例を示します。

```
$ oc create -f myactivedoccr1.yaml
```

この例では、出力は以下のようになります。

```
activedoc.capabilities.3scale.net/myactivedoccr1 created
```

## 検証

1. Red Hat OpenShift Container Platform (OCP) 管理者アカウントにログインします。
2. **Operators > Installed Operators** に移動します。
3. **Red Hat Integration - 3scale** をクリックします。
4. **Active Doc** タブをクリックします。
5. OAS ドキュメントが存在することを確認します。上記の例では、**myactivedoccr1** という名前の新しい OAS ドキュメントが表示されます。

### 8.7.2. 3scale ActiveDoc カスタムリソース定義の機能

**ActiveDoc** カスタムリソース定義 (CRD) は、開発者の **OpenAPI** ドキュメント形式の製品ドキュメントを考慮します。**ActiveDoc** CRD デプロイメント機能に関する知識は、[デベロッパーポータル](#)の **ActiveDocs** の作成に役立ちます。

- **ActiveDoc** CR は、以下のいずれかから OpenAPI ドキュメントを読み取りできます。
  - Secret
  - **http** または **https** 形式の URL



- オプションで、**productSystemName** フィールドを使用して、**ActiveDoc** CR を 3scale 製品にリンクできます。値は、3scale プ製品の CR の **system\_name** である必要があります。
- **published** フィールドを使用して、3scale の **ActiveDoc** ドキュメントを公開または非表示にすることができます。デフォルトでは、これは **hidden** に設定されています。
- **skipSwaggerValidations** フィールドを使用して、OpenAPI 3.0 検証を省略できます。デフォルトでは、**ActiveDoc** CR は検証されます。

## 関連情報

- [capabilities に関連するプロダクトカスタムリソース](#)
- [ActiveDoc CRD リファレンス](#)

### 8.7.3. URL から OAS ドキュメントをインポートする 3scale ActiveDoc カスタムリソースのデプロイ

指定した URL から OAS(OpenAPI Specification) ドキュメントをインポートする **ActiveDoc** カスタムリソース (CR) をデプロイできます。その後、この OAS ドキュメントを、デベロッパーポータルで API の ActiveDocs の基礎として使用できます。

## 前提条件

- [3scale Operator がカスタムリソースのリンク先となるテナントを識別する方法](#) を理解している。

## 手順

1. OpenShift アカウントで、**Operators > Installed operators** に移動します。
2. 3scale operator をクリックします。
3. **Active Doc** タブをクリックします。
4. **ActiveDoc** CR を作成します。以下に例を示します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ActiveDoc
metadata:
  name: myactivedoccr1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-Specification/master/examples/v3.0/petstore.yaml"
  providerAccountRef:
    name: mytenant
```

5. オプション。Self-managed APIcast の場合、**ActiveDoc** CR で **productionPublicBaseURL** および **stagingPublicBaseURL** フィールドをデプロイメントの URL に設定します。以下に例を示します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ActiveDoc
metadata:
```

```
name: myactivedoccr1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-
Specification/master/examples/v3.0/petstore.yaml"
  productionPublicBaseURL: "https://production.my-gateway.example.com"
  stagingPublicBaseURL: "https://staging.my-gateway.example.com"
```

6. **Save** をクリックします。3scale operator が **ActiveDoc** CR を作成するまでに数秒かかります。

## 検証

1. Red Hat OpenShift Container Platform (OCP) 管理者アカウントにログインします。
2. **Operators** > **Installed Operators** に移動します。
3. **Red Hat Integration 3scale** をクリックします。
4. **Active Doc** タブをクリックします。
5. OAS ドキュメントが存在することを確認します。上記の例では、**myactivedoccr1** という名前の新しい OAS ドキュメントが表示されます。

### 8.7.4. 関連情報

- [OpenAPI Specification \(OAS\)](#)
- [ActiveDoc CRD リファレンス](#)
- [オプションテナントカスタムリソースのデプロイ](#)

## 8.8. CAPABILITIES に関連するバックエンドカスタムリソース

新しく作成したテナントで OpenShift Container Platform を使用し、バックエンド、それらに対応するメトリック、メソッド、およびマッピングルールを設定します。バックエンドのカスタムリソースのステータスや、バックエンドがどのようにテナントアカウントにリンクされているかについても説明します。

### 前提条件

[一般的な前提条件](#) に記載の条件と同じインストール要件が適用されます。ただし、以下の考慮事項に注意してください。

- 3scale アカウントから最低限必要なパラメーターは、管理ポータル URL アドレスとアクセストークンです。

### 8.8.1. capabilities に関連するバックエンドカスタムリソースのデプロイ

新しく作成したテナントで OpenShift Container Platform を使用し、新規バックエンドを設定します。

### 手順

1. OpenShift アカウントで、**Installed operators** に移動します。

2. 3scale operator をクリックします。
3. **3scale Backend** で、**Create Instance** をクリックします。
4. YAML View を選択します。
5. 特定の 3scale 管理 URL アドレスをポイントする 3scale バックエンドを作成します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: <your_backend_OpenShift_name>
spec:
  name: "<your_backend_name>"
  privateBaseURL: "<your_admin_portal_URL>"
```

以下に例を示します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
```

6. 変更を保存するには、**Create** をクリックします。
7. OpenShift と 3scale アカウントの両方でバックエンドが作成されるまで数秒待機します。その後、以下の検証を実行できます。
  - a. **3scale Backend Overview** ページで、**Synced** 状態が **True** とマークされていることを確認して、バックエンドが OpenShift で作成されたことを確認します。
  - b. 3scale アカウントに移動し、バックエンドが作成されたことを確認します。上記の例では、**My Backend Name** という新しいバックエンドが表示されます。

### 8.8.2. バックエンドメトリクスの定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、バックエンドのカスタムリソースに必要なバックエンドメトリックを定義します。

以下の点について考慮してください。

- **metrics** マップキー名は **system\_name** として使用されます。下の例では **metric01**、**metric02**、および **hits** です。
- **metrics** マップキー名は、すべてのメトリックおよびメソッド間で一意である必要があります。
- **unit** および **friendlyName** は必須フィールドです。
- **Hits** メトリックを追加しない場合、このメトリックは operator によって作成されます。

手順

- 以下の例に示すように、新しい 3scale バックエンドにバックエンドメトリクスを追加します。

```

apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  metrics:
    metric01:
      friendlyName: Metric01
      unit: "1"
    metric02:
      friendlyName: Metric02
      unit: "1"
  hits:
    description: Number of API hits
    friendlyName: Hits
    unit: "hit"

```

### 8.8.3. バックエンドメソッドの定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、バックエンドのカスタムリソースに必要なバックエンドメソッドを定義します。

以下の点について考慮してください。

- **methods** マップキー名は **system\_name** として使用されます。以下の例では、**Method01** と **Method02** です。
- **methods** マップキー名は、すべてのメトリックおよびメソッド間で一意である必要があります。
- **friendlyName** は必須フィールドです。

#### 手順

- 以下の例に示すように、新しい 3scale バックエンドにバックエンドメソッドを追加します。

```

apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  methods:
    method01:
      friendlyName: Method01
    method02:
      friendlyName: Method02

```

### 8.8.4. バックエンドマッピングルールの定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、バックエンドのカスタムリソースに必要なバックエンドマッピングルールを定義します。

以下の点について考慮してください。

- **httpMethod**、**pattern**、**increment**、および **metricMethodRef** は必須フィールドです。
- **metricMethodRef** は、既存のメトリックまたはメソッドマップキー名 **system\_name** への参照を保持します。以下の例では、**hits** です。

#### 手順

- 以下の例に示すように、新しい 3scale バックエンドにバックエンドマッピングルールを追加します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  mappingRules:
    - httpMethod: GET
      pattern: "/pets"
      increment: 1
      metricMethodRef: hits
    - httpMethod: GET
      pattern: "/pets/id"
      increment: 1
      metricMethodRef: hits
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
```

#### 8.8.5. バックエンドカスタムリソースのステータス

**status** フィールドには、エンドユーザーに役立つリソース情報が表示されます。手動で更新することは意図されず、リソースの変更ごとに同期されます。

**status** フィールドの属性は以下のとおりです。

- **backendId**: 3scale バックエンドの内部 ID。
- **conditions: status.Conditions** Kubernetes 共通パターンを表します。これには、以下のタイプまたは状態があります。
  - **Invalid: BackendSpec** の設定の組み合わせはサポート対象外です。これは一時的なエラーではなく、進捗するには修正する必要がある状態を示します。
  - **Synced**: バックエンドは正常に同期されています。
  - **Failed**: 同期中にエラーが発生しています。

- **observedGeneration**: ステータス情報が最新のリソース仕様で更新されていることを確認するヘルパーフィールドです。

同期されたリソースの例:

```
status:
  backendId: 59978
  conditions:
  - lastTransitionTime: "2020-06-22T10:50:33Z"
    status: "False"
    type: Failed
  - lastTransitionTime: "2020-06-22T10:50:33Z"
    status: "False"
    type: Invalid
  - lastTransitionTime: "2020-06-22T10:50:33Z"
    status: "True"
    type: Synced
  observedGeneration: 2
```

### 8.8.6. テナントアカウントにリンクされたバックエンドカスタムリソース

3scale operator が新しい 3scale リソースを見つけると、**LookupProviderAccount** プロセスは、リソースを所有するテナントを識別するために開始されます。

プロセスは、テナントのクレデンシャルソースを確認します。何も見つからない場合は、エラーが発生します。

以下の手順では、プロセスがテナントのクレデンシャルソースを検証する方法を説明します。

1. **providerAccountRef** リソース属性からのクレデンシャルを確認します。これはシークレットのローカル参照です (例: **mytenant**)。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  providerAccountRef:
    name: mytenant
```

**mytenant** シークレットの **adminURL** および **token** フィールドには、テナントクレデンシャルが設定されている必要があります。以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: mytenant
type: Opaque
stringData:
  adminURL: https://my3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

2. デフォルトの **threescale-provider-account** シークレットを確認します。例:  
**adminURL=https://3scale-admin.example.com** および **token=123456**:

```
oc create secret generic threescale-provider-account --from-literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
```

3. 3scale デプロイメントの同じ namespace にあるデフォルトのプロバイダーアカウントを確認します。3scale インストール環境がカスタムリソースと同じ namespace にある場合、operator はデフォルトの 3scale テナント (プロバイダーアカウント) に必要なクレデンシャルを自動的に収集します。

### 8.8.7. バックエンドカスタムリソースの削除

バックエンドエンティティは、これを管理する **Backend** カスタムリソース (CR) を削除して削除できます。バックエンド CR を削除する **場合**、3scale operator は削除されたバックエンドを参照するデプロイ済みのプロダクト CR を更新します。更新は次の属性に含まれます。

- **backendUsages**
- **applicationPlans**

これらの属性は、削除されたバックエンドを参照しなくなりました。



#### 重要

バックエンド CR で定義される API バックエンドを削除する唯一の方法は、ここで説明されている手順に従います。管理ポータルまたは 3scale API を使用して、カスタムリソースとしてデプロイされたバックエンドを削除しないでください。

#### 前提条件

- 削除する **Backend** CR が含まれる namespace の削除パーミッションを持つ 3scale 管理者権限または OpenShift ロール。特定の **Backend** カスタムリソースを削除できるユーザーを特定するには、**oc policy who-can delete** コマンドを実行します。たとえば、CR の名前が **mybackend** の場合、以下のコマンドを実行します。

```
oc policy who-can delete product.capabilities.3scale.net/mybackend
```

- 有効なテナントへのリンクを削除する **Backend** CR。

#### 手順

- **oc delete** コマンドを実行して、**Backend** カスタムリソースを削除します。たとえば、**mybackend.yaml** ファイルで定義された **Backend** をデプロイした場合は、以下のコマンドを実行します。

```
oc delete -f mybackend.yaml
```

または、**oc delete** コマンドを実行し、その定義で指定したバックエンドの名前を指定します。以下に例を示します。

```
oc delete backend.capabilities.3scale.net/mybackend
```

## 8.9. CAPABILITIES に関連するプロダクトカスタムリソース

新しく作成したテナントで OpenShift Container Platform を使用し、プロダクト、それらに対応するメトリック、メソッド、アプリケーションプラン、およびマッピングルールを設定し、プロダクトバックエンドの使用状況を定義すると共にプロダクトをテナントアカウントにリンクします。

### 前提条件

[一般的な前提条件](#) に記載の条件と同じインストール要件が適用されます。ただし、以下の考慮事項に注意してください。

- 3scale アカウントから最低限必要なパラメーターは、プロダクト名です。

### 8.9.1. capabilities に関連するプロダクトカスタムリソースのデプロイ

新しく作成したテナントで OpenShift Container Platform を使用し、新規プロダクトを設定します。

#### 8.9.1.1. 基本的なプロダクトカスタムリソースのデプロイ

##### 手順

1. OpenShift アカウントで、**Installed operators** に移動します。
2. 3scale operator をクリックします。
3. **3scale Product** セクションで、**Create Instance** をクリックします。
4. YAML View を選択します。
5. 3scale プロダクトを作成します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: <your_product_OpenShift_name>
spec:
  name: "<your_product_name>"
```

以下に例を示します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
```

6. 変更を保存するには、**Create** をクリックします。
7. OpenShift と 3scale アカウントの両方でプロダクトが作成されるまで数秒待機します。その後、以下の検証を実行できます。
  - a. **3scale Product Overview** ページで、**Synced** 状態が **True** とマークされていることを確認して、プロダクトが OpenShift で作成されたことを確認します。



- b. 3scale アカウントに移動し、プロダクトが作成されたことを確認します。上記の例では、**OperatedProduct 1** という新しいプロダクトが表示されます。

また、作成する各プロダクトに APIcast のデプロイメントモードを指定することもできます。以下の2つの方法があります。

- [Hosted APIcast](#)
- [Self-managed APIcast](#)

### 8.9.1.2. Hosted APIcast を使用するプロダクトのデプロイ

Hosted APIcast と共にプロダクトを設定します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  deployment:
    apicastHosted: {}
```

### 8.9.1.3. Self-managed APIcast を使用するプロダクトのデプロイ

Self-managed APIcast と共にプロダクトを設定します。この場合、**stagingPublicBaseURL** および **productionPublicBaseURL** を指定します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  deployment:
    apicastSelfManaged:
      stagingPublicBaseURL: "https://staging.api.example.com"
      productionPublicBaseURL: "https://production.api.example.com"
```

## 8.9.2. プロダクトのアプリケーションプランの定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、**applicationPlans** オブジェクトを使用してプロダクトカスタムリソースに必要なアプリケーションプランを定義します。

以下の点について考慮してください。

- **applicationPlans** マップキー名は **system\_name** として使用されます。以下の例では、**plan01** および **plan02** です。



#### 注記

**setupFee** および **costMonth** は、一般的な 3scale の概念です。3scale ユーザーインターフェイスでアプリケーションプランを作成する際に、これらの詳細情報を入力する必要があります。[アプリケーションプランへの課金ルールの設定](#) を参照してください。

## 手順

- 以下の例に示すように、新しい 3scale プロダクトにアプリケーションプランを追加します。

```

apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  applicationPlans:
    plan01:
      name: "My Plan 01"
      setupFee: "14.56"
    plan02:
      name: "My Plan 02"
      trialPeriod: 3
      costMonth: 3

```

### 8.9.3. プロダクトアプリケーションプランの制限の定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、**applicationPlans.limits** リストを使用してプロダクトアプリケーションプランに必要な制限を定義します。

以下の点について考慮してください。

- **period**、**value**、および **metricMethodRef** は必須フィールドです。
- **metricMethodRef** 参照は、プロダクトまたはバックエンド参照のいずれかです。オプションの **backend** フィールドを使用して、バックエンドメトリックの所有者を参照します。

## 手順

- 以下の例に示すように、3scale プロダクトのアプリケーションプランに制限を定義します。

```

apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
  applicationPlans:
    plan01:
      name: "My Plan 01"
      limits:
        - period: month
          value: 300
          metricMethodRef:
            systemName: hits
            backend: backendA

```

```
- period: week
  value: 100
  metricMethodRef:
    systemName: hits
```

#### 8.9.4. プロダクトアプリケーションプランの課金ルールの定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、**applicationPlans.pricingRules** リストを使用してプロダクトアプリケーションプランに必要な課金ルールを定義します。

以下の点について考慮してください。

- **from**、**to**、**pricePerUnit**、および **metricMethodRef** は必須フィールドです。
- **from** および **to** は検証されます。すべてのルールで、**from** の値が **to** より小さい設定、および同じメトリックで範囲が重複する設定は許されません。
- **metricMethodRef** 参照は、プロダクトまたはバックエンド参照のいずれかです。オプションの **backend** フィールドを使用して、バックエンドメトリックの所有者を参照します。

#### 手順

- 以下の例に示すように、3scale プロダクトのアプリケーションプランに課金ルールを定義します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
  applicationPlans:
    plan01:
      name: "My Plan 01"
      pricingRules:
        - from: 1
          to: 100
          pricePerUnit: "15.45"
          metricMethodRef:
            systemName: hits
        - from: 1
          to: 300
          pricePerUnit: "15.45"
          metricMethodRef:
            systemName: hits
            backend: backendA
```

#### 8.9.5. OpenID Connect を使用したプロダクト認証の定義

任意の OAuth 2.0 フローの認証に [OpenID Connect \(OIDC\)](#) を使用する 3scale プロダクトの **Product** カスタムリソースをデプロイできます。3scale は、OpenID Connect などのサードパーティーアイデンティティプロバイダー (IdP) と統合して、API リクエストの認証を行います。OpenID Connect についての詳しい情報は、[OpenID Connect integration](#) を参照してください。サードパーティーの IdP との統合後、プロダクトカスタムリソースに 2 種類のデータを含めます。

- **issuerType**: Red Hat Single Sign-On (RH-SSO) 使用時の **keycloak** の値およびサードパーティー IdP との統合時の **rest** の値。
- **issuerEndpoint**: 必要なクレデンシャルを持つ URL。

### 前提条件

- RH-SSO を設定しておく。[Red Hat Single Sign-On の設定](#) を参照してください。
- [サードパーティーアイデンティティプロバイダーとの HTTP インテグレーション](#)を設定している。



### 注記

**issuerEndpoint** で提供されるクレデンシャル **CLIENT\_ID** および **CLIENT\_CREDENTIALS** には、レルムで他のクライアントを管理するために十分なパーミッションが必要です。

### 手順

1. OpenID プロバイダーの場所を定義するエンドポイント **issuerEndpoint** を決定し、プロダクトカスタムリソースでこの形式を使用します。

```
https://<client_id>:<client_secret>@<host>:<port_number>/auth/realms/<realm_name>
```

2. OAuth 2.0 フローの OpenID Connect (OIDC) 認証を指定する 3scale **Product** CR を定義または更新します。以下に例を示します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  deployment:
    <any>:
      authentication:
        oidc:
          issuerType: "keycloak"
          issuerEndpoint:
            "https://myclientid:myclientsecret@mykeycloak.example.com/auth/realms/myrealm"
          authenticationFlow:
            standardFlowEnabled: false
            implicitFlowEnabled: true
            serviceAccountsEnabled: true
            directAccessGrantsEnabled: true
            jwtClaimWithClientID: "azp"
            jwtClaimWithClientIDType: "plain"
```

3. 定義したばかりのリソースを作成します。以下に例を示します。

```
oc create -f product1.yaml
```

この例では、出力は以下のようになります。

```
product.capabilities.3scale.net/product1 created
```

## 関連情報

- [Product CRD Reference](#)

### 8.9.6. プロダクトメトリクスの定義

新しく作成した 3scale テナントで OpenShift Container Platform を使用し、**metrics** オブジェクトを使用してプロダクトカスタムリソースに必要なメトリックを定義します。

以下の点について考慮してください。

- **metrics** マップキー名は **system\_name** として使用されます。以下の例では、**metric01** および **hits** です。
- **metrics** マップキー名は、すべてのメトリックおよびメソッド間で一意である必要があります。
- **unit** および **friendlyName** は必須フィールドです。
- **hits** メトリックを追加しない場合は、operator によって作成されます。

## 手順

- 以下の例に示すように、新しい 3scale バックエンドにプロダクトメトリクスを追加します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    metric01:
      friendlyName: Metric01
      unit: "1"
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
```

### 8.9.7. プロダクトメソッドの定義

新しく作成した 3scale テナントで OpenShift Container Platform を使用し、**methods** オブジェクトを使用してプロダクトカスタムリソースに必要なメソッドを定義します。

以下の点について考慮してください。

- **methods** マップキー名は **system\_name** として使用されます。以下の例では、**Method01** と **Method02** です。
- **methods** マップキー名は、すべてのメトリックおよびメソッド間で一意である必要があります。
- **friendlyName** は必須フィールドです。

#### 手順

- 以下の例のように、新しい 3scale プロダクトにメソッドを追加します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  methods:
    method01:
      friendlyName: Method01
    method02:
      friendlyName: Method02
```

#### 8.9.8. プロダクトマッピングルールの定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、**mappingRules** オブジェクトを使用してプロダクトカスタムリソースに必要なマッピングルールを定義します。

以下の点について考慮してください。

- **httpMethod**、**pattern**、**increment**、および **metricMethodRef** は必須フィールドです。
- **metricMethodRef** は、既存のメトリックまたはメソッドマップキー名 **system\_name** への参照を保持します。以下の例では、**hits** です。

#### 手順

- 以下の例に示すように、新しい 3scale バックエンドにプロダクトマッピングルールを追加します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
  methods:
    method01:
      friendlyName: Method01
```

```
mappingRules:
  - httpMethod: GET
    pattern: "/pets"
    increment: 1
    metricMethodRef: hits
  - httpMethod: GET
    pattern: "/cars"
    increment: 1
    metricMethodRef: method01
```

### 8.9.9. プロダクトバックエンドの使用状況の定義

新しく作成した 3scale テナントで Openshift Container Platform を使用し、**backendUsages** オブジェクトを適用して宣言的にプロダクトに追加される必要なバックエンドを定義します。

以下の点について考慮してください。

- **path** は必須フィールドです。
- **backendUsages** マップキー名は、バックエンドの **system\_name** への参照です。以下の例では、**backendA** および **backendB** です。

#### 手順

- 以下の例に示すように、プロダクトにバックエンドを追加して、その使用状況を宣言的に定義します。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  backendUsages:
    backendA:
      path: /A
    backendB:
      path: /B
```

### 8.9.10. 3scale プロダクトカスタムリソースでのゲートウェイ応答の設定

3scale 管理者は、**Product** カスタムリソースを設定して、その API プロダクトの公開された API へのリクエストに対するゲートウェイレスポンスを指定できます。CR のデプロイ後に、3scale は指定した応答およびエラーメッセージをゲートウェイが返すようにします。

**Product** CR の **gatewayResponse** オブジェクトには、ゲートウェイが返すレスポンスが含まれます。

#### 手順

1. 新規またはデプロイされた **Product** CR で、**gatewayResponse** オブジェクトに1つ以上の応答を設定します。次の例は、**userKey** と呼ばれる認証モードの Apacast ホスト型デプロイメントのレスポンス設定を示しています。

```
apiVersion: capabilities.3scale.net/v1beta1
```

```

kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  deployment:
    apicastHosted:
      authentication:
        userkey:
          gatewayResponse:
            errorStatusAuthFailed: 500
            errorHeadersAuthFailed: "text/plain; charset=mycharset"
            errorAuthFailed: "My custom reponse body"
            errorStatusAuthMissing: 500
            errorHeadersAuthMissing: "text/plain; charset=mycharset"
            errorAuthMissing: "My custom reponse body"
            errorStatusNoMatch: 501
            errorHeadersNoMatch: "text/plain; charset=mycharset"
            errorNoMatch: "My custom reponse body"
            errorStatusLimitsExceeded: 502
            errorHeadersLimitsExceeded: "text/plain; charset=mycharset"
            errorLimitsExceeded: "My custom reponse body"

```

2. ゲートウェイのレスポンスが含まれる **Product** CR をデプロイします。たとえば、**product1.yaml** ファイルを更新した場合は、以下のコマンドを実行します。

```
oc create -f product1.yaml
```

この例では、出力は以下のようになります。

```
product.capabilities.3scale.net/product1 created
```

### 8.9.11. 3scale プロダクトカスタムリソースでのポリシーチェーンの設定

3scale の管理者は、**Product** カスタムリソースを設定して、その API プロダクトに適用するポリシーチェーンを指定できます。CR のデプロイ後に、3scale は設定済みのポリシーをプロダクトのアップストリームの公開された API へのリクエストに適用します。

#### 手順

1. 新規またはデプロイされた **Product** CR で、**policies** オブジェクトに1つ以上のポリシーを設定します。以下に例を示します。

```

apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  policies:
    - configuration:
        http_proxy: http://example.com
        https_proxy: https://example.com
      enabled: true

```



```

name: camel
version: builtin
- configuration: {}
enabled: true
name: apicast
version: builtin

```

それぞれのポリシーについて、以下のフィールドを指定します。

- ポリシーにパラメーターがない場合、**configuration** は空白の中かっこを指定します。ポリシーにパラメーターがある場合は、ここで指定します。指定する必要があるパラメーターの名前については、[API ゲートウェイ](#)、[APIcast 標準ポリシーの管理](#) の関連するポリシーのドキュメントを参照してください。
- **enabled** は、ポリシーをオンまたはオフにするブール値のスイッチです。
- **name** はポリシーを識別します。これは、**Product** カスタムリソースがリンクするテナントの範囲にある一意の名前です。ポリシー名を特定するには、[API ゲートウェイ](#)、[APIcast 標準ポリシーの管理](#) の関連するポリシーのドキュメントを参照してください。
- **version** は、標準ポリシーの場合は **builtin** で、カスタムポリシーの場合はユーザー定義の文字列です。たとえば、カスタムポリシーのバージョンを **1.0** に設定できます。**Product** CR に **apicast** ポリシーを指定しないと、operator がこれを追加します。

ポリシーチェーンがすでに管理ポータルに定義されている場合は、3scale toolbox の **export** コマンドを実行してポリシーチェーンを **.yaml** 形式でエクスポートできます。**export** 出力を **Product** CR に貼り付けできます。たとえば、**api-provider-account-one** は 3scale プロバイダーアカウントの名前で、**my-api-product-one** はポリシーチェーンをエクスポートするプロダクトの名前である場合、以下のコマンドを実行します。

```
3scale policies export api-provider-account-one my-api-product-one
```

2. ポリシーチェーンが含まれる **Product** CR をデプロイします。たとえば、**product1.yaml** ファイルを更新した場合は、以下のコマンドを実行します。

```
oc create -f product1.yaml
```

この例では、出力は以下のようになります。

```
product.capabilities.3scale.net/product1 created
```

### 8.9.12. プロダクトカスタムリソースのステータス

**status** フィールドには、エンドユーザーに役立つリソース情報が表示されます。手動で更新することは意図されず、リソースの変更ごとに同期されます。

**status** フィールドの属性は以下のとおりです。

- **productid**: 3scale プロダクトの内部 ID
- **conditions: status.Conditions** Kubernetes 共通パターンを表します。これには、以下のタイプまたは状態があります。
  - **Failed**: 同期中にエラーが発生しています。操作はリトライされます。

- **Synced:** プロダクトは正常に同期されています。
- **Invalid:** オブジェクトが無効です。これは一時的なエラーではなく、無効な使用が報告され、それを変更する必要があります。操作はリトライされません。
- **Orphan:** 仕様は存在しないリソースを参照しています。operator はリトライします。
- **observedGeneration:** ステータス情報が最新のリソース仕様で更新されていることを確認します。
- **state:** 3scale API から読み取られた 3scale プロダクトの内部状態。
- **providerAccountHost:** バックエンドが同期される 3scale プロバイダーアカウントの URL。

同期されたリソースの例:

```
status:
  conditions:
  - lastTransitionTime: "2020-10-21T18:07:01Z"
    status: "False"
    type: Failed
  - lastTransitionTime: "2020-10-21T18:06:54Z"
    status: "False"
    type: Invalid
  - lastTransitionTime: "2020-10-21T18:07:01Z"
    status: "False"
    type: Orphan
  - lastTransitionTime: "2020-10-21T18:07:01Z"
    status: "True"
    type: Synced
observedGeneration: 1
productId: 2555417872138
providerAccountHost: https://3scale-admin.example.com
state: incomplete
```

### 8.9.13. テナントアカウントにリンクされたプロダクトカスタムリソース

3scale operator が新しい 3scale リソースを見つけると、**LookupProviderAccount** プロセスは、リソースを所有するテナントを識別するために開始されます。

プロセスは、テナントのクレデンシャルソースを確認します。何も見つからない場合は、エラーが発生します。

以下の手順では、プロセスがテナントのクレデンシャルソースを検証する方法を説明します。

1. **providerAccountRef** リソース属性からのクレデンシャルを確認します。これはシークレットのローカル参照です (例: **mytenant**)。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  providerAccountRef:
    name: mytenant
```

mytenant シークレットの `adminURL` および `token` フィールドには、テナントクレデンシャルが設定されている必要があります。以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: mytenant
type: Opaque
stringData:
  adminURL: https://my3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

2. デフォルトの `threescale-provider-account` シークレットを確認します。例:  
`adminURL=https://3scale-admin.example.com` および `token=123456`:

```
oc create secret generic threescale-provider-account --from-literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
```

3. 3scale デプロイメントの同じ namespace にあるデフォルトのプロバイダーアカウントを確認します。3scale インストール環境がカスタムリソースと同じ namespace にある場合、operator はデフォルトの 3scale テナント (プロバイダーアカウント) に必要なクレデンシャルを自動的に収集します。

#### 8.9.14. Product カスタムリソースの削除

3scale プロダクトエンティティを削除するには、管理するカスタムリソースを削除します。**Product** カスタムリソースを削除する場合、3scale operator は削除したプロダクトを参照するオブジェクトを更新しません。



#### 重要

**Product** カスタムリソースで定義される API プロダクトを削除する唯一の方法として、ここで説明されている手順に従います。管理ポータルまたは 3scale API を使用して、カスタムリソースとしてデプロイされたプロダクトを削除しないでください。

#### 前提条件

- 削除するカスタムリソースが含まれる namespace の削除パーミッションが指定された 3scale 管理者権限または OpenShift ロール。特定の **Product** カスタムリソースを削除できるユーザーを特定するには、`oc policy who-can delete` コマンドを実行します。たとえば、CR の名前が `myproduct` の場合には、以下のコマンドを実行します。

```
oc policy who-can delete product.capabilities.3scale.net/myproduct
```

- 有効なテナントへのリンクを削除する **Product** CR。

#### 手順

- `oc delete` コマンドを実行して、**Product** カスタムリソースを削除します。たとえば、`myproduct.yaml` ファイルで定義した **Product** をデプロイした場合は、以下のコマンドを実行します。

```
oc delete -f myproduct.yaml
```

または、**oc delete** コマンドを実行して、定義で指定した製品の名前を指定します。以下に例を示します。

```
oc delete product.capabilities.3scale.net/myproduct
```

## 8.10. 3SCALE CUSTOMPOLICYDEFINITION カスタムリソースのデプロイ

**CustomPolicyDefinition** CRD を使用して、管理ポータルから 3scale プロダクトのカスタムポリシーを設定できます。

3scale operator が新規の **CustomPolicyDefinition** CR を見つけると、operator は、[How the 3scale operator identifies the tenant that a custom resource links to](#) で説明されているように CR を所有するテナントを識別します。

### 前提条件

- 3scale Operator がインストールされている。
- カスタムポリシーファイルをデプロイする準備が完了している。
- [ゲートウェイにカスタムポリシーを挿入](#) している。

### 手順

1. **CustomPolicyDefinition** カスタムリソースを定義して保存します (例: **my-apicast-custom-policy-definition.yaml** ファイル)。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: CustomPolicyDefinition
metadata:
  name: custompolicydefinition-sample
spec:
  version: "0.1"
  name: "APIcast Example Policy"
  schema:
    name: "APIcast Example Policy"
    version: "0.1"
    $schema: "http://apicast.io/policy-v1/schema#manifest#"
    summary: "This is just an example."
  configuration:
    type: object
    properties: {}
```

2. **CustomPolicyDefinition** CR をデプロイします。

```
oc create -f my-apicast-custom-policy-definition.yaml
```

### 関連情報

- [CustomPolicyDefinition CRD Reference](#)

## 8.11. テナントカスタムリソースのデプロイ

テナントカスタムリソースは、**プロバイダーアカウント**とも呼ばれます。

**APIManager** カスタムリソース (CR) をデプロイする場合には、3scale operator を使用して 3scale をデプロイします。デフォルトの 3scale インストール環境には、使用可能なデフォルトのテナントが含まれます。オプションで、**Tenant** カスタムリソースをデプロイして他のテナントを作成できます。

### 前提条件

- [一般的な前提条件](#)
- 新規 **Tenant** CR の namespace に対して **create** パーミッションが指定された OpenShift ロール。

### 手順

1. 3scale がインストールされている OpenShift プロジェクトに移動します。たとえば、プロジェクトの名前が **my-3scale-project** の場合は、以下のコマンドを実行します。

```
oc project my-3scale-project
```

2. 新規テナントの 3scale 管理アカウントのパスワードが含まれるシークレットを作成します。**Tenant** CR の定義で、**passwordCredentialsRef** 属性をこのシークレットの名前に設定します。ステップ 4 の **Tenant** CR 定義の例では、**ADMIN\_SECRET** はこのシークレットのプレースホルダーになります。以下のコマンドは、シークレットの作成例を示しています。

```
oc create secret generic ecorp-admin-secret --from-literal=admin_password=<admin
password value>
```

3. 3scale マスターアカウントのホスト名を取得します。operator を使用して 3scale をデプロイする場合、マスターアカウントには、**master.\${wildcardDomain}** のパターンの固定 URL があります。3scale がインストールされている namespace にアクセスできる場合は、以下のコマンドでマスターアカウントのホスト名を取得できます。

```
oc get routes --field-selector=spec.to.name==system-master -o jsonpath="{.items[].spec.host}"
```

ステップ 4 の **Tenant** CR 定義の例では、**MASTER\_HOSTNAME** はこの名前のプレースホルダーです。

4. 新しい **Tenant** カスタムリソースを定義するファイルを作成します。**Tenant** CR の定義で、**masterCredentialsRef.name** 属性を **system-seed** に設定します。テナント管理タスクは、3scale マスターアカウントのクレデンシャルを使用してのみ実行できます (アクセストークンの使用が推奨されます)。**APIManager** CR のデプロイメント時に、Operator はマスターアカウントの認証情報が含まれるシークレットを作成します。シークレットの名前は **system-seed** です。

3scale がクラスター全体のモードでインストールされている場合には、3scale が含まれる namespace とは異なる namespace に新規テナントをデプロイできます。これには、**masterCredentialsRef.namespace** を 3scale インストールが含まれる namespace に設定します。

以下の例では、3scale がクラスター全体のモードにインストールされていることを前提とします。

```
apiVersion: capabilities.3scale.net/v1alpha1
kind: Tenant
metadata:
  name: ecorp-tenant
  namespace: <namespace-in-which-to-create-Tenant-CR>
spec:
  username: admin
  systemMasterUrl: https://<MASTER_HOSTNAME>
  email: admin@ecorp.com
  organizationName: ECorp
  masterCredentialsRef:
    name: system-seed
    namespace: <namespace-where-3scale-is-deployed>
  passwordCredentialsRef:
    name: <ADMIN_SECRET>
  tenantSecretRef:
    name: tenant-secret
```

5. **Tenant** カスタムリソースを作成します。たとえば、直前のサンプル CR を **mytenant.yaml** ファイルに保存した場合は、以下を実行します。

```
oc create -f mytenant.yaml
```

このコマンドを実行すると、以下のようになります。

- operator は、**spec.systemMasterUrl** 属性を設定して、3scale インストールのテナントをデプロイします。
- 3scale operator は、新規テナントの認証情報が含まれるシークレットを作成します。シークレットの名前は、**tenantSecretRef.name** 属性に指定した値です。このシークレットには、新規テナントの管理者 URL およびアクセストークンが含まれます。参考として、Operator が作成するシークレットの例を以下に示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: tenant-secret
type: Opaque
stringData:
  adminURL: https://my3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

- 新規テナントにリンクする **Product**、**Backend**、**OpenAPI**、**DeveloperAccount**、および **DeveloperUser** CR をデプロイできるようになりました。

## Tenant カスタムリソースのデプロイ

デプロイされた **Tenant** カスタムリソースを削除するには、リソース定義が含まれるファイルの名前を指定します。以下に例を示します。

```
oc delete -f mytenant.yaml
```

または、**oc delete** コマンドを実行して **Tenant** CR に名前を指定し、テナントの namespace を指定することもできます。以下に例を示します。

```
oc delete tenant.capabilities.3scale.net mytenant -n mynamespace
```

テナントを削除すると、3scale operator は以下を行います。

- 3scale インストール環境からテナントを非表示にする。
- テナントが所有するデプロイ済みのカスタムリソースを削除する。
- 15 日後に削除するテナントをマークする。

テナントの削除後に復元することはできません。15 日の間に、テナントが所有するリソースをすべてバックアップしてください。この操作は、管理ポータルでのみ実行できます。3scale は 15 日後にテナントを削除します。データ保護法に準拠するために、今後の参照用に一部のデータが保持されます。



### 重要

**Tenant** CR を使用してテナントをデプロイした場合は、管理ポータルでテナントを削除しないでください。これを実行する場合に、Operator は削除しようとしているテナントの CR を使用して別のテナントを作成します。

### 関連情報

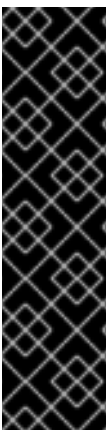
- [Tenant CRD field reference](#)

## 8.12. カスタムリソースのデプロイによる 3SCALE 開発者の管理

3scale の管理者は、カスタムリソース (CR) を使用して、個別の開発者ユーザーをグループ化する開発者アカウントをデプロイすることができます。これらのアカウントにより、開発者ポータルで 3scale が管理する API への開発者アクセスを整理および管理できます。

テナントには、任意の数の開発者アカウントを含めることができ、各開発者アカウントは必ず 1 つのテナントにリンクされます。開発者アカウントには、任意の数の開発者ユーザーを含めることができ、各開発者ユーザーは 1 つの開発者アカウントにリンクします。テナントプランは、作成できる開発者アカウントの数と、各開発者アカウントにグループ化できる開発者ユーザーの数の制限を決定します。

開発者カスタムリソースを使用するには、3scale が 3scale operator によってインストールされている必要があります。開発者カスタムリソースは、3scale Operator が含まれる namespace にのみデプロイできます。開発者カスタムリソースのデプロイは、別の開発者の管理方法で、3scale 管理ポータルまたは 3scale 内部 API を使用します。



### 重要

カスタムリソースをデプロイして開発者アカウントまたは開発者ユーザーを作成する場合、管理ポータルまたは内部 3scale API を使用してそれらの開発者アカウントまたは開発者ユーザーを更新することはできません。開発者 CR をデプロイした後、管理ポータルの **アカウント** ページに新しい開発者アカウントまたは新しい開発者ユーザーが表示されるため、注意することが重要となります。管理ポータルまたは API を使用して CR でデプロイされた開発者アカウントまたは開発者ユーザーを更新しようとする、3scale Operator はデプロイされた CR を反映するために変更を元に戻します。これは、今後のリリースで削除される予定の制限です。ただし、管理ポータルまたは API を使用して、CR をデプロイして作成した開発者アカウントまたは開発者ユーザーを削除することができます。

### 8.12.1. 前提条件

- 3scale が 3scale operator によってインストールされている。
- **Account Management** API スコープの読み取りおよび書き込み権限を持つアクセストークン (3scale の管理者権限が提供される)

### 8.12.2. DeveloperAccount カスタムリソースのデプロイによる 3scale の開発者アカウントの管理

3scale operator を使用して 3scale をインストールする場合は、**DeveloperAccount** および **DeveloperUser** カスタムリソース (CR) をデプロイできます。これらの CR により、開発者ポータルで 3scale が管理する API に開発者がアクセスするためのアカウントを作成および更新できます。

新規の **DeveloperAccount** CR をデプロイするには、**admin** ロールを持つユーザーの **DeveloperUser** CR もデプロイする必要があります。ここで提供される手順は、新規 **DeveloperAccount** CR をデプロイするためのものです。**DeveloperAccount** CR のデプロイ後、更新または削除する手順は他の CR と同じです。

CR は 3scale Operator が含まれる namespace にのみデプロイできます。

#### 前提条件

- [3scale Operator がカスタムリソースのリンク先となるテナントを識別する方法](#) を理解している。
- 同じ namespace にある 3scale インスタンスのデフォルトのテナントにリンクしない **DeveloperAccount** カスタムリソースを作成する場合には、**DeveloperAccount** CR が含まれる namespace には、**DeveloperAccount** CR がリンクするテナントを特定するシークレットを追加する。シークレットの名前は以下のいずれかになります。
  - **threescale-provider-account**
  - ユーザー定義

このシークレットには、3scale インスタンスの URL と、3scale インスタンスの1つのテナントにアクセスするためのクレデンシャルが含まれるトークンが含まれます。

- 新規 **DeveloperAccount** CR の **admin** ロールを持つ少なくとも1人の開発者ユーザーのユーザー名、パスワード、およびメールアドレスを把握している。

#### 手順

1. 3scale operator が含まれる namespace で、新しい開発者アカウントリソースで **admin** ロールを持つ開発者ユーザーのユーザー名とパスワードが含まれるシークレットを定義するリソースファイルを作成して保存します。たとえば、**myusername01.yaml** ファイルには以下が含まれます。

```
apiVersion: v1
kind: Secret
metadata:
  name: myusername01
stringData:
  password: "123456"
```

2. シークレットを作成します。以下に例を示します。



```
oc create -f myusername01.yaml
```

この例では、出力は以下のようになります。

```
secret/myusername01 created
```

3. **admin** ロールを持つ開発者の **DeveloperUser** CR を定義する a **.yaml** ファイルを作成し、保存します。3scale operator が新規 **DeveloperAccount** CR をデプロイするには、この **DeveloperUser** CR が必要です。たとえば、**developeruser01.yaml** ファイルには以下が含まれる場合があります。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: DeveloperUser
metadata:
  name: developeruser01
spec:
  username: myusername01
  email: myusername01@example.com
  passwordCredentialsRef:
    name: myusername01
  role: admin
  developerAccountRef:
    name: developeraccount1
  providerAccountRef:
    name: mytenant
```

**DeveloperUser** CR では:

- 開発者ユーザーのアカウント名、ユーザー名、および電子メールは、含んでいる **DeveloperAccount** のリンク先のテナントで一意である必要があります。
  - ここで指定する開発者アカウント名は、この手順でデプロイする **DeveloperAccount** CR の名前に一致する必要があります。 **DeveloperAccount** CR の作成は、この **DeveloperUser** CR を作成する前でも後でもかまいません。
  - **DeveloperUser** CR がリンクするテナントは、指定された **DeveloperAccount** CR リンク先と同じテナントである必要があります。
4. 定義したばかりのリソースを作成します。以下に例を示します。

```
oc create -f developeruser01.yaml
```

この例では、出力は以下のようになります。

```
developeruser.capabilities.3scale.net/developeruser01 created
```

5. **DeveloperAccount** CR を定義する **.yaml** ファイルを作成および保存します。この **.yaml** ファイルでは、**spec.OrgName** フィールドは組織名を指定する必要があります。たとえば、**developeraccount01.yaml** ファイルには以下が含まれる場合があります。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: DeveloperAccount
metadata:
  name: developeraccount01
```

```
spec:
  orgName: Ecorp
  providerAccountRef:
    name: mytenant
```

6. 定義したばかりのリソースを作成します。以下に例を示します。

```
oc create -f developeraccount01.yaml
```

この例では、出力は以下のようになります。

```
developeraccount.capabilities.3scale.net/developeraccount01 created
```

## 次のステップ

3scale Operator が 3scale 設定を更新して、新規または更新されたカスタムリソースを反映するには、数秒かかります。Operator がカスタムリソース情報を正常に伝搬しているかどうかを確認するには、**DeveloperAccount** カスタムリソースの **status** フィールドを確認するか、以下のように **oc wait** コマンドを実行します。

```
oc wait --for=condition=Ready --timeout=30s developeraccount/developeraccount1
```

失敗した場合、カスタムリソースの **status** フィールドは、エラーが一時的または永続的であるかどうかを示し、問題の修正に役立つエラーメッセージを提供します。

新しい開発者ユーザーに、開発者ポータルにログインできることを通知します。また、ログインクレデンシャルを伝える必要がある場合もあります。

他のカスタムリソースを更新するのと同じ方法で、デプロイされた **DeveloperAccount** カスタムリソースを更新できます。たとえば、更新する **DeveloperAccount** CR を所有するテナントを含む OpenShift プロジェクトでは、次のコマンドを実行して **devaccount1** CR を更新します。

```
oc edit developeraccount devaccount1
```

## 関連情報

- [DeveloperAccount または DeveloperUser カスタムリソースの削除](#)
- [DeveloperAccount CRD Reference](#)
- [DeveloperUser CRD Reference](#)

### 8.12.3. DeveloperUser カスタムリソースのデプロイによる 3scale 開発者ユーザーの管理

3scale operator を使用して 3scale をインストールする場合、Developer Portal で 3scale 管理の API への開発者アクセスを管理するために、**DeveloperUser** カスタムリソース (CR) をデプロイできます。ここで提供される手順は、新規 **DeveloperUser** CR をデプロイするためのものです。**DeveloperUser** CR のデプロイ後、更新または削除する手順は他の CR と同じです。

CR は 3scale Operator が含まれる namespace にのみデプロイできます。

## 前提条件

- [3scale Operator がカスタムリソースのリンク先となるテナントを識別する方法](#) を理解している。
- **admin** ロールを持つユーザー用に、デプロイされた **DeveloperUser** CR が少なくとも1つ含まれる **DeveloperAccount** カスタムリソースが少なくとも1つデプロイされている。同じ namespace にある 3scale インスタンスのデフォルトのテナントにリンクしない **DeveloperUser** カスタムリソースを作成する場合、**DeveloperUser** CR が含まれる namespace には、**DeveloperUser** CR がリンクするテナントを特定するシークレットを含める。シークレットの名前は以下のいずれかになります。
  - **threescale-provider-account**
  - ユーザー定義

このシークレットには、3scale インスタンスの URL と、3scale インスタンスの1つのテナントにアクセスするためのクレデンシャルが含まれるトークンが含まれます。
- 新しい **DeveloperUser** カスタムリソース用に、その開発者のユーザー名、パスワード、およびメールアドレスを把握している。

## 手順

1. 3scale Operator が含まれる namespace で、開発者ユーザーのユーザー名とパスワードが含まれるシークレットを定義するリソースファイルを作成し、保存します。たとえば、**myusername02.yaml** ファイルには以下が含まれます。

```
apiVersion: v1
kind: Secret
metadata:
  name: myusername02
stringData:
  password: "987654321"
```

2. シークレットを作成します。以下に例を示します。

```
oc create -f myusername02.yaml
```

この例では、出力は以下のようになります。

```
secret/myusername02 created
```

3. **DeveloperUser** CR を定義する **.yaml** ファイルを作成および保存します。**spec.role** フィールドには **admin** または **member** を指定します。たとえば、**developeruser02.yaml** ファイルには以下が含まれる場合があります。

```
apiVersion: capabilities.3scale.net/v1beta1
kind: DeveloperUser
metadata:
  name: developeruser02
spec:
  username: myusername02
  email: myusername02@example.com
  passwordCredentialsRef:
    name: myusername02
  role: member
```

```
developerAccountRef:
  name: developeraccount1
providerAccountRef:
  name: mytenant
```

**DeveloperUser** CR では:

- 開発者のユーザー名 (**metadata.name** フィールドで指定)、ユーザー名、および電子メールは、含んでいる **DeveloperAccount** のリンク先のテナントで一意である必要があります。
  - **developerAccountRef** フィールドには、デプロイされた **DeveloperAccount** CR の名前を指定する必要があります。
  - **DeveloperUser** CR がリンクするテナントは、指定された **DeveloperAccount** CR リンク先と同じテナントである必要があります。
4. 定義したばかりのリソースを作成します。以下に例を示します。

```
oc create -f developefuser02.yaml
```

この例では、出力は以下のようになります。

```
developeruser.capabilities.3scale.net/developeruser02 created
```

## 次のステップ

3scale Operator が 3scale 設定を更新して、新規または更新されたカスタムリソースを反映するには、数秒かかります。Operator がカスタムリソース情報を正常に伝搬しているかどうかを確認するには、**DeveloperUser** カスタムリソースの **status** フィールドを確認するか、以下のように **oc wait** コマンドを実行します。

```
oc wait --for=condition=Ready --timeout=30s developeruser/developeruser02
```

失敗した場合、カスタムリソースの **status** フィールドは、エラーが一時的または永続的であるかどうかを示し、問題の修正に役立つエラーメッセージを提供します。

新しい開発者ユーザーに、開発者ポータルにログインできることを通知します。また、ログインクレデンシャルを伝える必要がある場合もあります。

デプロイされた **DeveloperUser** カスタムリソースは、他のカスタムリソースを更新するのと同じ方法で更新できます。

## 関連情報

- [DeveloperAccount または DeveloperUser カスタムリソースの削除](#)
- [DeveloperUser CRD Reference](#)

### 8.12.4. DeveloperAccount または DeveloperUser カスタムリソースの削除

3scale 開発者エンティティを削除するには、管理するカスタムリソースを削除します。**DeveloperAccount** カスタムリソースを削除すると、3scale Operator は、削除された **DeveloperAccount** CR にリンクするすべての **DeveloperUser** CR も削除します。



## 重要

カスタムリソースによって定義された開発者アカウントまたは開発者ユーザーを削除する唯一の方法は、ここで説明する手順に従うことです。管理ポータルまたは 3scale API を使用して、カスタムリソースとしてデプロイされた開発者エンティティを削除しないでください。

### 前提条件

- 削除するカスタムリソースが含まれる namespace の削除パーミッションが指定された 3scale 管理者権限または OpenShift ロール。特定のカスタムリソースを削除できることを確認するには、**oc auth can-i delete** コマンドを実行します。たとえば、**DeveloperAccount** CR の名前が **devaccount1** の場合は、次のコマンドを実行します。

```
oc auth can-i delete developeraccount.capabilities.3scale.net/devaccount1 -n my-namespace
```

- 削除する **DeveloperAccount** または **DeveloperUser** CR は、有効なテナントにリンクしていません。

### 手順

- CR がリンクするテナントを含む OpenShift プロジェクトで、**oc delete** コマンドを実行して、**DeveloperAccount** または **DeveloperUser** カスタムリソースを削除します。たとえば、**devaccount1.yaml** ファイルで定義された **DeveloperAccount** CR をデプロイした場合は、次のコマンドを実行します。

```
oc delete -f devaccount1.yaml
```

または、**oc delete** コマンドを実行して、定義で指定した CR の名前を指定します。以下に例を示します。

```
oc delete developeraccount.capabilities.3scale.net/devaccount1
```

## 8.13. 3SCALE OPERATOR の機能の制限

Red Hat 3scale API Management 2.13 では、3scale operator には機能に関する以下の制限があります。

- バックエンドカスタムリソース定義 (CRD) の削除は調整されません。3scale の既存のバックエンドは削除されません。
- プロダクト CRD の削除は調整されません。3scale の既存プロダクトは削除されません。
- DeveloperAccount** または **DeveloperUser** カスタムリソースの削除は調整されません。Operator は削除イベントを受け取りますが、Operator はイベントに対応しません。開発者アカウントまたは開発者ユーザーはそのままとなります。削除したカスタムリソースと同じアカウント名、ユーザー名、または電子メールアドレスで新しい開発者アカウントまたは開発者ユーザーを作成しようとすると、アカウントがすでに存在するというエラーが表示されます。アカウントを作成するには、別のパラメーターを指定する必要があります。
- 管理ポータルの Single Sign-On (SSO) 認証。
- デベロッパーポータルの SSO 認証。
- ActiveDocs CRD は現在利用できません。

- ゲートウェイポリシー CRD は現在利用できません。
- プロダクト CRD ゲートウェイはレスポンスのカスタムコードおよびエラーをサポートしません。
- OAS3 を保持する 3scale operator CRD は、3scale プロダクト設定の信頼できるソースとして参照しません。

## 8.14. 関連情報

詳細は、以下のガイドを参照してください。

- [Backend CRD field Reference](#)
- [Product CRD field Reference](#)
- [CustomPolicyDefinition CRD Reference](#)
- [Tenant CRD field reference](#)

## 第9章 3SCALE のバックアップと復元

本セクションでは、Red Hat 3scale API Management インストール環境の管理者が以下の操作を行うのに必要な情報を提供します。

- 永続データのバックアップ手順を設定する
- 永続データのバックアップから復元を行う

MySQL データベースの1つまたは複数に問題が発生した場合に、3scale を以前の稼働状態に正しく復元することができます。

### 9.1. 前提条件

- 3scale 2.13 インスタンス。3scale のインストール方法については、[OpenShift への 3scale のインストール](#) を参照してください。
- OpenShift クラスターの以下のいずれかのロールを持つ OpenShift Container Platform 4 ユーザーアカウント
  - cluster-admin
  - admin
  - edit



#### 注記

3scale インストールの namespace にローカルでバインドされた **edit** クラスターロールを持つユーザーは、バックアップと復元の手順を実行できます。

以下には、永続データのバックアップ手順を設定し、永続データのバックアップから復元を実行する方法に関する情報が含まれています。MySQL データベースの1つまたは複数に障害が発生した場合に、3scale を以前の稼働状態に正しく復元することができます。

- [永続ボリュームおよび考慮事項](#)
- [データセットの使用](#)
- [システムデータベースのバックアップ](#)
- [システムデータベースの復元](#)

### 9.2. 永続ボリュームおよび考慮事項

#### 永続ボリューム

OpenShift 上の 3scale [デプロイメント](#) の場合:

- ベースとなるインフラストラクチャーによってクラスターに提供される永続ボリューム (PV)
- クラスター外のストレージサービス。これは、同じデータセンターまたは別のデータセンターに配置することができます。

#### 留意事項

永続データのバックアップおよび復元手順は、使用するストレージタイプによって異なります。バックアップと復元とでデータの一貫性を維持するためには、データベースのベースとなる PV をバックアップするだけでは不十分です。部分書き込みや部分トランザクションだけを取得するべきではないからです。PV をバックアップするのではなく、データベースのバックアップメカニズムを使用してください。

データの一部は、異なるコンポーネント間で同期されます。あるコピーは、データセットの **信頼できるソース** とみなされます。他は、ローカルでは変更されないが **信頼できるソース** から同期されるコピーです。このような場合は、復元が完了したら、**信頼できるソース** を復元し、その他のコンポーネントのコピーをそこから同期する必要があります。

## 9.3. データセットの使用

本セクションでは、さまざまな永続ストアのさまざまなデータセット、その目的、使用されるストレージタイプ、およびそれが **信頼できるソース** であるかどうかについて、さらに詳しく説明します。

3scale デプロイメントの状態は、すべて以下の **DeploymentConfig** オブジェクトとその PV のいずれかに保存されます。

名前	説明
<code>system-mysql</code>	MySQL データベース ( <b>mysql-storage</b> )
<code>system-storage</code>	ファイル用のボリューム
<code>backend-redis</code>	Redis データベース ( <b>backend-redis-storage</b> )
<code>system-redis</code>	Redis データベース ( <b>system-redis-storage</b> )

### 9.3.1. system-mysql の定義

**system-mysql** はリレーショナルデータベースで、3scale 管理コンソールのユーザー、アカウント、API、プランなどについての情報を保存します。

サービスに関連するこの情報のサブセットは、**Backend** コンポーネントと同期され、**backend-redis** に保存されます。**system-mysql** は、この情報の **信頼できるソース** です。

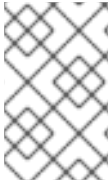
### 9.3.2. system-storage の定義

**system-storage** は、システム コンポーネントにより読み取り/書き込みされるファイルを保存します。

これは 2 つのカテゴリに分類されます。

- 実行時に **システム** コンポーネントが読み込む設定ファイル
- デベロッパーポータルを作成する目的で、CMS 機能によってシステムにアップロードされる静的ファイル (たとえば、HTML、CSS、JS など)





## 注記

**System** は、上記の静的ファイルをアップロードおよび読み込む複数の Pod により水平スケーリングすることができます。したがって、ReadWriteMany (RWX) **PersistentVolume** が必要です。

### 9.3.3. backend-redis の定義

**backend-redis** には、バックエンド コンポーネントにより使用される複数のデータセットが含まれます。

- **Usages:** これは **Backend** により集約された API 使用量についての情報です。これは、**Backend** による流量制限の決定と、**System** による分析情報の表示 (UI または API 経由) に使用されます。
- **Config:** これはサービス、流量制御などに関する設定情報で、内部 API 経由で **System** から同期されます。これは、この情報の **信頼できるソース** ではありませんが、**System** と **system-mysql** は信頼できるソースです。
- **Queues:** これは、ワーカープロセスで実行されるバックグラウンドジョブのキューです。これらは一時的なものであり、処理後に削除されます。

### 9.3.4. system-redis の定義

**system-redis** には、バックグラウンドで処理されるジョブのキューが含まれます。これらは一時的なものであり、処理後に削除されます。

## 9.4. システムデータベースのバックアップ

以下のコマンドを実行する順序に指定はありませんが、システムデータベースをバックアップしてアーカイブする場合に、必要に応じて使用することができます。

### 9.4.1. system-mysql のバックアップ

MySQL バックアップコマンドを実行します。

```
oc rsh $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq -r '.items[0].metadata.name')
bash -c 'export MYSQL_PWD=${MYSQL_ROOT_PASSWORD}; mysqldump --single-transaction -
hsystem-mysql -uroot system' | gzip > system-mysql-backup.gz
```

### 9.4.2. system-storage のバックアップ

**system-storage** ファイルを別のストレージにアーカイブします。

```
oc rsync $(oc get pods -l 'deploymentConfig=system-app' -o json | jq '.items[0].metadata.name' -
r):/opt/system/public/system ./local/dir
```

### 9.4.3. backend-redis のバックアップ

redis からの **dump.rdb** ファイルをバックアップします。

```
oc cp $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq '.items[0].metadata.name' -
r):/var/lib/redis/data/dump.rdb ./backend-redis-dump.rdb
```

#### 9.4.4. system-redis のバックアップ

redis からの **dump.rdb** ファイルをバックアップします。

```
oc cp $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq '.items[0].metadata.name' -r):/var/lib/redis/data/dump.rdb ./system-redis-dump.rdb
```

#### 9.4.5. zync-database のバックアップ

**zync\_production** データベースをバックアップします。

```
oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq -r '.items[0].metadata.name') bash -c 'pg_dump zync_production' | gzip > zync-database-backup.gz
```

#### 9.4.6. OpenShift シークレットおよび ConfigMap のバックアップ

OpenShift シークレットおよび ConfigMap のコマンドリストを以下に示します。

##### 9.4.6.1. OpenShift シークレット

```
oc get secrets system-smtp -o json > system-smtp.json
oc get secrets system-seed -o json > system-seed.json
oc get secrets system-database -o json > system-database.json
oc get secrets backend-internal-api -o json > backend-internal-api.json
oc get secrets system-events-hook -o json > system-events-hook.json
oc get secrets system-app -o json > system-app.json
oc get secrets system-recaptcha -o json > system-recaptcha.json
oc get secrets system-redis -o json > system-redis.json
oc get secrets zync -o json > zync.json
oc get secrets system-master-apicast -o json > system-master-apicast.json
```

##### 9.4.6.2. ConfigMaps

```
oc get configmaps system-environment -o json > system-environment.json
oc get configmaps apicast-environment -o json > apicast-environment.json
```

### 9.5. システムデータベースの復元



#### 重要

**system-app** のような Pod をスケールダウンしたり、ルートを無効にしたりして、レコードが作成されないようにします。

以下のコマンドとスニペットの例では、**`\${DEPLOYMENT\_NAME}`** を 3scale デプロイメントの作成時に定義した名前に置き換えます。



#### 注記

出力に少なくとも中括弧 **{ }** のペアが含まれており、空でないことを確認してください。

## 手順

1. 後でスケールアップできるように、現在のレプリカ数を保存します。

```
SYSTEM_SPEC=`oc get APIManager/${DEPLOYMENT_NAME} -o
jsonpath='{.spec.system.appSpec}'`
```

2. 前のコマンドの結果を確認し、**\$SYSTEM\_SPEC** の内容を確認します。

```
echo $SYSTEM_SPEC
```

3. レプリカ数を **0** にスケールする次のコマンドを使用して、APIManager CR にパッチを適用します。

```
$ oc patch APIManager/${DEPLOYMENT_NAME} --type merge -p '{"spec": {"system":
{"appSpec": {"replicas": 0}}}'
```

あるいは、**system-app** をスケールダウンするには、次の例に示すように、既存の **APIManager/\${DEPLOYMENT\_NAME}** を編集し、システムレプリカ数を 0 に設定します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: <DEPLOYMENT_NAME>
spec:
  system:
    appSpec:
      replicas: 0
```

OpenShift シークレットおよびシステムデータベースを復元するには、以下の手順を使用します。

- [operator ベースのデプロイメントの復元](#)
- [system-mysql の復元](#)
- [system-storage の復元](#)
- [zync-database の復元](#)
- [バックエンドとシステム間における情報の一貫性の確保](#)

### 9.5.1. operator ベースのデプロイメントの復元

operator ベースのデプロイメントを復元するには、以下の手順に従います。

## 手順

1. [3scale Operator](#) を OpenShift にインストールします。
2. APIManager リソースを作成する前に、シークレットを復元します。

```
$ oc apply -f system-smtp.json
$ oc apply -f system-seed.json
$ oc apply -f system-database.json
```

```
$ oc apply -f backend-internal-api.json
$ oc apply -f system-events-hook.json
$ oc apply -f system-app.json
$ oc apply -f system-recaptcha.json
$ oc apply -f system-redis.json
$ oc apply -f zync.json
$ oc apply -f system-master-apicast.json
```

3. APIManager リソースを作成する前に、ConfigMaps を復元します。

```
$ oc apply -f system-environment.json
$ oc apply -f apicast-environment.json
```

4. APIManager CR を使用して、[Operator](#) で [3scale](#) をデプロイします。

## 9.5.2. system-mysql の復元

### 手順

1. MySQL ダンプを system-mysql Pod にコピーします。

```
$ oc cp ./system-mysql-backup.gz $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq '.items[0].metadata.name' -r):/var/lib/mysql
```

2. バックアップファイルをデプロイメントします。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq -r '.items[0].metadata.name') bash -c 'gzip -d ${HOME}/system-mysql-backup.gz'
```

3. MySQL DB バックアップファイルを復元します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq -r '.items[0].metadata.name') bash -c 'export MYSQL_PWD=${MYSQL_ROOT_PASSWORD}; mysql -hsystem-mysql -uroot system < ${HOME}/system-mysql-backup'
```

## 9.5.3. system-storage の復元

バックアップファイルを system-storage に復元します。

```
$ oc rsync ./local/dir/system/ $(oc get pods -l 'deploymentConfig=system-app' -o json | jq '.items[0].metadata.name' -r):/opt/system/public/system
```

## 9.5.4. zync-database の復元

3scale operator デプロイメントの **zync-database** を復元する手順。

### 9.5.4.1. operator ベースのデプロイメント



## 注記

**Operator を使用した 3scale のデプロイ (特に APIManager CR のデプロイ)** に記載の手順にしたがって、3scale インスタンスを再デプロイします。

## 手順

1. **{DEPLOYMENT\_NAME}** を 3scale デプロイメントの作成時に定義した名前に置き換えて、レプリカ数を保存します。

```
ZYNC_SPEC=`oc get APIManager/{DEPLOYMENT_NAME} -o json | jq -r '.spec.zync`
```

2. zync DeploymentConfig を 0 Pod にスケールダウンします。

```
$ oc patch APIManager/{DEPLOYMENT_NAME} --type merge -p '{"spec": {"zync": {"appSpec": {"replicas": 0}, "queSpec": {"replicas": 0}}}'
```

3. zync データベースダンプを **zync-database** Pod にコピーします。

```
$ oc cp ./zync-database-backup.gz $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq '.items[0].metadata.name' -r):/var/lib/pgsql/
```

4. バックアップファイルをデプロイメントします。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq -r '.items[0].metadata.name') bash -c 'gzip -d ${HOME}/zync-database-backup.gz'
```

5. zync データベースのバックアップファイルを復元します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq -r '.items[0].metadata.name') bash -c 'psql zync_production -f ${HOME}/zync-database-backup'
```

6. 元のレプリカ数に復元します。

```
$ oc patch APIManager/{DEPLOYMENT_NAME} --type json -p '[{"op": "replace", "path": "/spec/zync", "value": "{$ZYNC_SPEC}"]'
```

- 以下のコマンドの出力に **replicas** キーが含まれていない場合は、

```
$ echo $ZYNC_SPEC
```

- 以下の追加コマンドを実行して **zync** をスケールアップします。

```
$ oc patch dc/zync -p '{"spec": {"replicas": 1}}'
```

### 9.5.4.2. backend-redis と system-redis での 3scale オプションの復元

3scale を復元することで、**backend-redis** と **system-redis** が復元されます。これらのコンポーネントには、以下の機能があります。

\***backend-redis**: 3scale のアプリケーション認証とレート制限をサポートするデータベース。統計ストレージおよび一時ジョブストレージにも使用されます。\***system-redis**: 3scale のバックグラウンドジョ

ブの一時ストレージを提供し、**system-app Pod** の Ruby プロセスのメッセージバスとしても使用されます。

## backend-redis コンポーネント

**backend-redis** コンポーネントには、**data** と **queue** の2つのデータベースがあります。デフォルトの 3scale デプロイメントでは、**data** および **queues** は、異なる論理データベースインデックス /0 および /1 で、Redis データベースにデプロイされます。**data** データベースの復元は問題なく実行されますが、**queues** データベースを復元すると、ジョブが重複してしまう可能性があります。

ジョブの重複に関して、3scale ではバックエンドワーカーがバックグラウンドジョブを処理します (ミリ秒単位)。最後のデータベーススナップショットの 30 秒後に **backend-redis** が失敗し、そのスナップショットを復元しようとする、バックエンドには重複を防ぐためのシステムがないため、30 秒の間に発生したバックグラウンドジョブは 2 回実行されます。

このシナリオでは、/0 データベースインデックスにその他の場所に保存されないデータが含まれているため、バックアップを復元する必要があります。/0 データベースインデックスを復元すると、/1 データベースインデックスを復元する必要もあります。どちらか1つだけを復元することはできません。異なるインデックスの1つのデータベースではなく、異なるサーバー上のデータベースを分離することを選択した場合、キューのサイズはほぼゼロになるため、バックアップを復元せず、いくつかのバックグラウンドジョブを失うことが望ましくなります。これは、3scale のホスト型設定の場合であるため、両方に異なるバックアップおよび復元ストラテジーを適用する必要があります。

## `system-redis` コンポーネント

3scale システムのバックグラウンドジョブの大半はベジ等です。つまり、実行する回数に関係なく、同じリクエストが同じ結果を返します。

以下は、システムのバックグラウンドジョブによって処理されるイベントの例のリストです。

- プランの試用期間の有効期限がまもなく切れる、クレジットカードの有効期限がまもなく切れる、アクティベーションのリマインダー、プランの変更、請求書の状態の変更、PDF レポートなどの通知ジョブ。
- インボイスや課金などの請求。
- 複雑なオブジェクトの削除。
- バックエンド同期ジョブ。
- たとえば sphinx を使用したインデックス作成ジョブ。
- 請求書 ID などのサニタイゼーションジョブ。
- 監査、ユーザーセッション、期限切れのトークン、ログエントリーのページ、非アクティブなアカウントを一時停止するなどの管理タスク。
- トラフィックの更新。
- プロキシの設定変更の監視およびプロキシのデプロイメント。
- バックグラウンドのサインアップジョブ。
- シングルサインオン (SSO) の同期、ルート作成などの Zync ジョブ。

上記のバックグラウンドジョブのリストを復元する場合には、3scale のシステムは復元された各ジョブの状態を維持します。復元の完了後にシステムの整合性を確認することが重要です。

### 9.5.5. バックエンドとシステム間における情報の一貫性の確保

**backend-redis** の復元後、**system** からの Config 情報を強制的に同期させ、**backend** の情報と **信頼** できるソースである **system** の情報の一貫性を確保する必要があります。

#### 9.5.5.1. backend-redis のデプロイメント設定の管理

以下の手順は、動作中の **backend-redis** インスタンス用です。

##### 手順

1. **redis-config** configmap を編集します。

```
$ oc edit configmap redis-config
```

2. **redis-config** configmap の **SAVE** コマンドをコメント化します。

```
#save 900 1  
#save 300 10  
#save 60 10000
```

3. **redis-config** configmap の **appendonly** を **no** に設定します。

```
appendonly no
```

4. **backend-redis** を再デプロイして、新しい設定を読み込みます。

```
$ oc rollout latest dc/backend-redis
```

5. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/backend-redis
```

6. **dump.rdb** ファイルの名前を変更します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/dump.rdb ${HOME}/data/dump.rdb-  
old'
```

7. **appendonly.aof** ファイルの名前を変更します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/appendonly.aof  
${HOME}/data/appendonly.aof-old'
```

8. バックアップファイルを POD に移動します。

```
$ oc cp ./backend-redis-dump.rdb $(oc get pods -l 'deploymentConfig=backend-redis' -o json  
| jq '.items[0].metadata.name' -r):/var/lib/redis/data/dump.rdb
```

9. **backend-redis** を再デプロイして、バックアップを読み込みます。

```
$ oc rollout latest dc/backend-redis
```

10. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/backend-redis
```

11. **appendonly** ファイルを作成します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'redis-cli BGREWRITEAOF'
```

12. しばらくしてから、AOF の書き換えが完了していることを確認します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'redis-cli info' | grep aof_rewrite_in_progress
```

- **aof\_rewrite\_in\_progress = 1** の間は、実行は進行中です。
- **aof\_rewrite\_in\_progress = 0** となるまで、定期的に確認します。ゼロは実行が完了したことを示します。

13. **redis-config** configmap を編集します。

```
$ oc edit configmap redis-config
```

14. **redis-config** configmap の **SAVE** コマンドをコメント解除します。

```
save 900 1  
save 300 10  
save 60 10000
```

15. **redis-config** configmap の **appendonly** を **yes** に設定します。

```
appendonly yes
```

16. **backend-redis** を再デプロイして、デフォルト設定を再読み込みします。

```
$ oc rollout latest dc/backend-redis
```

17. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/backend-redis
```

### 9.5.5.2. system-redis のデプロイメント設定の管理

以下の手順は、動作中の **system-redis** インスタンス用です。

#### 手順

1. **redis-config** configmap を編集します。

```
$ oc edit configmap redis-config
```



2. **redis-config** configmap の **SAVE** コマンドをコメント化します。

```
#save 900 1  
#save 300 10  
#save 60 10000
```

3. **redis-config** configmap の **appendonly** を **no** に設定します。

```
appendonly no
```

4. **system-redis** を再デプロイして、新しい設定を読み込みます。

```
$ oc rollout latest dc/system-redis
```

5. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/system-redis
```

6. **dump.rdb** ファイルの名前を変更します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/dump.rdb ${HOME}/data/dump.rdb-  
old'
```

7. **appendonly.aof** ファイルの名前を変更します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/appendonly.aof  
${HOME}/data/appendonly.aof-old'
```

8. **バックアップ** ファイルを POD に移動します。

```
$ oc cp ./system-redis-dump.rdb $(oc get pods -l 'deploymentConfig=system-redis' -o json |  
jq '.items[0].metadata.name' -r):/var/lib/redis/data/dump.rdb
```

9. **system-redis** を再デプロイして、バックアップを読み込みます。

```
$ oc rollout latest dc/system-redis
```

10. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/system-redis
```

11. **appendonly** ファイルを作成します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'redis-cli BGREWRITEAOF'
```

12. しばらくしてから、AOF の書き換えが完了していることを確認します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'redis-cli info' | grep aof_rewrite_in_progress
```

- **aof\_rewrite\_in\_progress = 1**の間は、実行は進行中です。
- **aof\_rewrite\_in\_progress = 0**となるまで、定期的を確認します。ゼロは実行が完了したことを示します。

13. **redis-config** configmap を編集します。

```
$ oc edit configmap redis-config
```

14. **redis-config** configmap の **SAVE** コマンドをコメント解除します。

```
save 900 1
save 300 10
save 60 10000
```

15. **redis-config** configmap の **appendonly** を **yes** に設定します。

```
appendonly yes
```

16. **system-redis** を再デプロイして、デフォルト設定を再読み込みします。

```
$ oc rollout latest dc/system-redis
```

17. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/system-redis
```

### 9.5.6. backend-worker の復元

これらの手順は、**backend-worker** を復元することを目的としています。

#### 手順

1. 最新バージョンの **backend-worker** に復元します。

```
$ oc rollout latest dc/backend-worker
```

2. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/backend-worker
```

### 9.5.7. system-app の復元

これらの手順は、**system-app** を復元することを目的としています。

#### 手順

1. **system-app** をスケールアップするには、既存の **APIManager/\${DEPLOYMENT\_NAME}** を編集して、**.spec.system.appSpec.replicas** を元のレプリカ数に戻すか、次のコマンドを実行して、以前に保存された仕様を適用します。

```
$ oc patch APIManager/${DEPLOYMENT_NAME} --type json -p '{"op": "replace", "path": "/spec/system/appSpec", "value": "${SYSTEM_SPEC}"}'
```

- 以下のコマンドの出力に **replicas** キーが含まれていない場合は、

```
$ echo $SYSTEM_SPEC
```

- 以下の追加コマンドを実行して **system-app** をスケールアップします。

```
$ oc patch dc/system-app -p '{"spec": {"replicas": 1}}'
```

2. 最新バージョンの **system-app** に復元します。

```
$ oc rollout latest dc/system-app
```

3. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/system-app
```

### 9.5.8. system-sidekiq の復元

これらの手順は、**system-sidekiq** を復元することを目的としています。

#### 手順

1. 最新バージョンの **system-sidekiq** に復元します。

```
$ oc rollout latest dc/system-sidekiq
```

2. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/system-sidekiq
```

#### 9.5.8.1. system-sphinx の復元

これらの手順は、**system-sphinx** を復元することを目的としています。

#### 手順

1. 最新バージョンの **system-sphinx** に復元します。

```
$ oc rollout latest dc/system-sphinx
```

2. ロールアウトのステータスを表示し、読み込みが完了したことを確認します。

```
$ oc rollout status dc/system-sphinx
```

### 9.5.8.2. zync が管理する OpenShift ルートの復元

- 不足している OpenShift ルートを再作成するように zync に強制します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-sidekiq' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'bundle exec rake zync:resync:domains'
```

## 第10章 3SCALE 用 RECAPTCHA の設定

本章では、オンプレミス型 Red Hat 3scale API Management に reCAPTCHA を設定して、スパムから保護する方法を説明します。

### 前提条件

- [サポート対象バージョンの OpenShift](#) にインストールされた設定済みのオンプレミス型 3scale インスタンス。
- reCAPTCHA v2 のサイトキーと秘密鍵を取得している。[新しいサイトを登録する](#) の Web ページを参照してください。
- ドメイン名の検証を使用する場合は、デベロッパーポータルドメインを許可リストに追加する。

3scale に reCAPTCHA を設定するには、以下の手順に概略を示すステップを実施します。

- [「3scale のスパム保護用 reCAPTCHA の設定」](#)

### 10.1. 3SCALE のスパム保護用 RECAPTCHA の設定

スパム保護に reCAPTCHA を設定するには、reCAPTCHA が含まれるシークレットファイルにパッチを適用する方法が 2 つあります。これらのオプションには、OpenShift Container Platform (OCP) ユーザーインターフェイスまたはコマンドラインインターフェイス (CLI) を使用します。

#### 手順

1. OCP 4.x: Project: [Your\_project\_name] > Workloads > Secrets に移動します。
2. **system-recaptcha** シークレットファイルを編集します。  
reCAPTCHA サービスからの **PRIVATE\_KEY** および **PUBLIC\_KEY** は、base64 フォーマットでエンコーディングされている必要があります。鍵を base64 エンコーディングに手動で変換します。



#### 注記

CLI の reCAPTCHA オプションでは、base64 フォーマットのエンコードは必要ありません。

- CLI: 以下のコマンドを入力します。

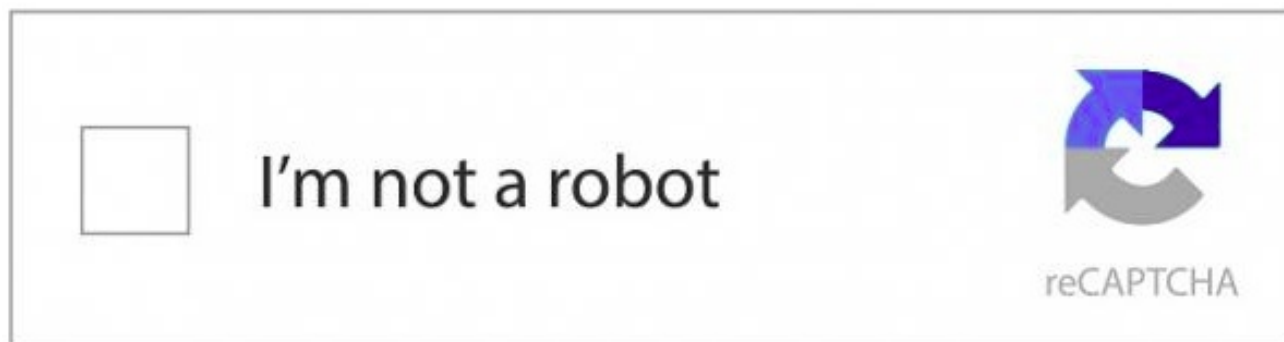
```
oc patch secret/system-recaptcha -p '{"stringData": {"PUBLIC_KEY": "public-key-from-service", "PRIVATE_KEY": "private-key-from-service"}}'
```

#### 手順後のステップ

- 上記のいずれかのオプションが完了したら、システム Pod を再デプロイします。
- 3scale 管理ポータルでは、署名されていないユーザーに対するスパム保護を有効にします。
  1. **Audience > Developer Portal > Spam Protection** の順に移動します。
  2. 以下のオプションのいずれかを選択します。

- **Always:** ログインしていないユーザーにフォームが表示されると、常に reCAPTCHA が表示されます。
- **Suspicious only:** 自動チェックでスパムの可能性が検出された場合にのみ、reCAPTCHA が表示されます。
- **Never:** Spam 保護をオフにします。

**system-app** が再デプロイされると、デベロッパーポータルスパム保護が使用されるページには、reCAPTCHA の I'm not a robot チェックボックスが表示されます。



#### 関連情報

- 詳細、ガイド、およびサポートについては、[ReCAPTCHA](#) のホームページを参照してください。

## 第11章 3SCALE WEBASSEMBLY モジュール

**threescale-wasm-auth** モジュールは、Service Mesh にプラグインされ、Red Hat 3scale API Management を使用して受信リクエストを承認できるようにする **WebAssembly** モジュールです。Service Mesh の機能を拡張し、マイクロサービスの認証、分析、課金などの完全な API 管理機能を提供します。

Service Mesh は、トラフィック管理、サービス検出、負荷分散、セキュリティーなどの機能を備えたインフラストラクチャー層に焦点を当てています。API 管理は、API の作成、公開、管理に重点を置いています。

Service Mesh と 3scale を組み合わせることで、マイクロサービスと API の信頼性、スケーラビリティ、セキュリティー、パフォーマンスを向上させることができます。



### 注記

**threescale-wasm-auth** モジュールは、3scale 2.11 以降と Red Hat OpenShift Service Mesh 2.1.0 以降のインテグレーションで実行されます。

### 前提条件

- 管理者権限が設定された 3scale アカウント
- Service Mesh 2.4 以降のインストール。
  - Service Mesh 2.3 は現在、[OSSM-3647](#) のため動作しません。
  - Service Mesh 2.1 および 2.2 については、[3scale WebAssembly モジュールの使用](#) を参照してください。
- Service Mesh 内で実行されるアプリケーション。
  - [Bookinfo サンプルアプリケーション](#) を使用します。

OpenShift Container Platform (OCP) のクラスター管理者は、WasmPlugin カスタムリソースを介して 3scale への HTTP リクエストを承認するように **threescale-wasm-auth** モジュールを設定できます。次に、Service Mesh はモジュールをサイドカーに注入してホストサービスを公開し、モジュールを使用してプロキシリクエストを処理できるようにします。

3scale の観点から見ると、**threescale-wasm-auth** モジュールはゲートウェイとして機能し、Service Mesh と統合するときに APIcast を置き換えます。これは、一部の APIcast 機能、特にポリシー、ステージング環境、実稼働環境が使用できないことを意味します。

## 11.1. BOOKINFO アプリケーションを SERVICE MESH にデプロイする

Service Mesh のサンプル Bookinfo アプリケーションを使用して、3scale で Service Mesh を設定する手順をデモンストレーションできます。

### 手順

1. Bookinfo アプリケーションをデプロイします。
  - [Bookinfo サンプルアプリケーション](#) を参照してください。
2. アプリケーションが利用可能であることを確認します。

■

```
$ export GATEWAY_URL=$(oc -n istio-system get route istio-ingressgateway -o
jsonpath='{.spec.host}')
```

```
$ curl -I "http://$GATEWAY_URL/productpage"
HTTP/1.1 200 OK
```

## 11.2. 3SCALE で製品を作成する

製品とは、バックエンドと呼ばれる複数の内部 API をリダイレクトまたは使用できる顧客向け API のことです。Service Mesh で 3scale を使用する場合、バックエンドは使用されません。製品とプライベートベース URL 間のリンクは、メッシュ内に作成されます。このため、必要なのは製品のみです。

### 手順

1. 新しい製品、アプリケーション計画、およびアプリケーションを作成します。[Creating new products to test API calls](#) を参照してください。
2. デプロイメントを Istio に変更します。
  - [Your\_product\_name] > Integration > Settings の順に移動します。
  - デプロイメントを Istio に変更します。
  - **Update Product** をクリックして設定を更新します。
3. 設定をプロモートします。
  - [Your\_product\_name] > Integration > Configuration の順に移動します。
  - **設定の更新** をクリックします。

## 11.3. 3SCALE と SERVICE MESH の接続



### 重要

**ServiceEntry** カスタムリソース (CR) と **DestinationRule** CR を、service-mesh/istio-system namespace または info namespace に作成します。これは、ServiceMeshControlPlane を含む namespace に存在する必要があります。

Service Mesh から 3scale にアクセスするには、**ServiceEntry** および **DestinationRule** (CR) を通じて、テナント URL とバックエンド URL の両方を外部サービスとして設定する必要があります。これにより、**threescale-wasm-auth** モジュールは、リクエストの承認を処理するバックエンドと、製品設定が取得されるシステムの両方にアクセスできるようになります。

### 11.3.1. 3scale URL を Service Mesh に追加する

**ServiceEntry** は Service Mesh 内からサービスへのリクエストを許可するために必要であり、**DestinationRule** は 3scale サービスの安全な接続を設定するためにあります。

#### 11.3.1.1. テナント URL を Service Mesh に追加する

### 手順



1. システムテナント URL を収集します。
  - これは、製品の作成に使用した 3scale 管理ポータル URL です。
2. システムの **ServiceEntry** を作成します。

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: <service_entry_threescale_system>
spec:
  hosts:
  - <system_hostname>
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
EOF
```

3. システムの **DestinationRule** を作成します。

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: <destination_rule_threescale_system>
spec:
  host: <system_hostname>
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: <system_hostname>
EOF
```

#### 関連情報

- [サービスエントリーについて](#)
- [宛先ルールについて](#)

## 11.4. バックエンド URL を SERVICE MESH に追加する

3scale バックエンド URL を Service Mesh セットアップに組み込むことで、マイクロサービスと 3scale バックエンドの間に安全な通信チャネルを確立できます。このインテグレーションにより、Service Mesh 環境で API を管理するための認証、分析、課金機能の実装が可能になります。バックエンドには、外部からは公開されたルートを使用してアクセスでき、内部的には OpenShift サービスを使用してアクセスできます。

### 11.4.1. Service Mesh とは別のクラスターで 3scale を使用する

#### 手順

## 1. バックエンド URL を収集します。

- 3scale Hosted の場合、バックエンド URL は **su1.3scale.net** です。
- オンプレミス型 3scale の場合は、次のコマンドを使用して URL を取得します。

```
$ oc get -n <3scale_namespace> route backend --template="{{.spec.host}}"
```

2. バックエンドの **ServiceEntry** を作成します。

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: <service_entry_threescale_backend>
spec:
  hosts:
  - <backend_hostname>
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
EOF
```

3. バックエンドの **DestinationRule** を作成します。

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: <destination_rule_threescale_backend>
spec:
  host: <backend_hostname>
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: <backend_hostname>
EOF
```

## 11.5. SERVICE MESH と同じクラスターで 3SCALE を使用する



### 注記

次の手順は、[Service Mesh へのバックエンド URL の追加](#) の代替手順です。

**threescale-wasm-auth module** に 3scale に対するリクエストを承認させるには、モジュールが 3scale サービスにアクセスできる必要があります。これを Red Hat OpenShift Service Mesh 内で行うには、外部 **ServiceEntry** オブジェクトと、**HTTPS** プロトコルを使用する **TLS** 設定の対応する **DestinationRule** オブジェクトを適用します。

カスタムリソース (CR) は、Service Management API および Account Management API のバックエンドおよびシステムコンポーネントに対して、Service Mesh 内から 3scale への安全なアクセスのための

サービスエントリーと宛先ルールをセットアップします。Service Management API は、各リクエストの承認ステータスのクエリーを受信します。Account Management API は、サービスの API 管理設定を提供します。

## 手順

1. バックエンドの **ServiceEntry** を作成します。

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: <service_entry_threescale_backend>
spec:
  hosts:
  - backend-listener.<3scale_namespace>.svc.cluster.local
  ports:
  - number: 80
    name: http
    protocol: HTTP
  location: MESH_EXTERNAL
  resolution: DNS
EOF
```

2. バックエンドの **DestinationRule** を作成します。

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: <destination_rule_threescale_backend>
spec:
  host: backend-listener.<3scale_namespace>.svc.cluster.local
EOF
```

## 11.6. WASMPLUGIN カスタムリソースの作成

Service Mesh は、**WasmPlugin** として知られる Proxy-WASM エクステンションを指定してサイドカープロキシに適用するためのカスタムリソース定義 (CRD) を提供します。Service Mesh は、3scale による HTTP API 管理を必要とする一連のワークロードにカスタムリソース (CR) を適用します。

## 手順

1. このモジュールを適用する Service Mesh デプロイメント上の OpenShift Container Platform (OCP) namespace (info プロジェクトなど) を特定します。
2. [registry.redhat.io](https://registry.redhat.io) 認証情報を使用してプルシークレットを取得します。
  - **WasmPlugin** と同じ namespace に新しいプルシークレットリソースを作成します。
3. モジュールが適用されるアプリケーションのセットを識別するセレクターとともに、**threescale-wasm-auth** モジュールがデプロイされる namespace を宣言する必要があります。次の例は、**threescale-wasm-auth** モジュールの CR の YAML 形式です。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <namespace>
spec:
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  imagePullSecret: <pull_secret_resource>
  phase: AUTHZ
  priority: 100
  match:
    - mode: CLIENT
  selector:
    matchLabels:
      app: <selector>
  pluginConfig:
    api: v1
    system:
      name: system
      upstream:
        name: outbound|443||<system_host>
        url: <system_url>
        timeout: 5000
        token: <access_token>

    backend:
      name: backend
      upstream:
        name: outbound|<backend_port>||<backend_host>
        url: <backend_url>
        timeout: 5000
      extensions:
        - no_body
      services:
        - id: '<product_id>'
      authorities:
        - "*"
      credentials:
        user_key:
          - query_string:
              keys:
                - user_key
          - header:
              keys:
                - user_key

```

- **spec.pluginConfig** フィールドはアプリケーションによって異なります。その他のフィールドはすべて、このカスタムリソースの複数のインスタンス間で永続します。
- この特定の **WasmPlugin spec.pluginConfig** は、クエリー文字列で提供される **user\_key** 認証を使用して設定されています。
- 説明:
  - **name**: 3scale 内の **WasmPlugin** の一意の名前または識別子を指定します。

- **namespace:** ワークロードの namespace。
- **imagePullSecret:** 手順 2 で作成したプルシークレットの名前。
- **selector:** ワークロードラベルセレクター。info プロジェクトの製品ページをご利用ください。
- **backend-port:** 使用している 3scale によって異なります。[3scale URL を Service Mesh に追加する](#) を参照してください。たとえば、内部 3scale はポート 80 を使用し、外部 3scale はポート 443 を使用します。
- **backend-host、system-host:** 3scale URL を Service Mesh に追加するで使用したホストと同じホストを使用します。
- **system-url、backend-url:** それぞれのホストを使用し、プロトコルを追加します。たとえば、[https://<system-host>](#) などです。
- **access-token:** システムテナントへのアクセストークン。
- **product\_id:** 使用する製品の ID。複数の製品が必要な場合は、サービスセクションで複数の製品を定義します。
- **spec.pluginConfig** および残りのカスタムリソースにモジュール設定を追加したら、**oc apply** コマンドでこれを適用します。

```
$ oc apply -f threescale-wasm-auth-info.yaml
```

### 11.6.1. 3scale WasmPlugin 認証オプション

これらは、3scale User key (App IDd/App key) 認証の設定例です。

#### ユーザーキー

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  ...
  pluginConfig:
    ...
    services:
      - id: '<service_id>'
        authorities:
          - ""
        credentials:
          user_key:
            - query_string:
                keys:
                  - user_key
            - header:
                keys:
                  - user_key
```

#### App ID と App key

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  ...
  pluginConfig:
    ...
  services:
  - id: '<service_id>'
    authorities:
      - "*"
    credentials:
      app_id:
        - query_string:
            keys:
              - app_id
        - header:
            keys:
              - app_id
      app_key:
        - query_string:
            keys:
              - app_key
        - header:
            keys:
              - app_key

```

## OIDC

**WasmPlugin** 自体とは別に、OpenID Connect (OIDC) が機能するには、**RequestAuthentication** と呼ばれる追加のカスタムリソースも必要です。**RequestAuthentication** を適用すると、JWT トークンを検証するネイティブプラグインを使用して **Envoy** が設定されます。プロキシは、モジュールを実行する前にすべてを検証します。したがって、失敗したリクエストが 3scale WebAssembly モジュールに実行されません。

```

apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: <info>
spec:
  selector:
    matchLabels:
      app: <productpage>
  jwtRules:
  - issuer: >-
    "<url>/auth/realms/<realm_name>"
    jwksUri: >-
    "<url>/auth/realms/<realm_name>/protocol/openid-connect/certs"

```

## 説明

- **<url>**: OIDC インスタンスの URL は、keycloak で設定されている場合、認証設定用の keycloak OIDC プロバイダーのメタデータエンドポイントを指定するために使用されます。

- `<realm_name>`: OIDC で使用されるレルムの名前。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  ...
  pluginConfig:
    ...
    services:
      - id: '<service_id>'
        authorities:
          - "*"
        credentials:
  app_id:
    - filter:
      path:
        - envoy.filters.http.jwt_authn
        - "0"
      keys:
        - azp
        - aud
      ops:
        - take:
            head: 1

```

## 関連情報

- [JSON Web トークンを使用したアクセスの制限](#)
- [Wasm Plugin](#)

## 11.7. 設定された API のテスト

アプリケーションを呼び出すときに認証チェックを実行することで、API 設定の有効性を検証できます。認証メカニズムを徹底的にテストすることで、承認されたリクエストのみが処理されることを確実にし、アプリケーションのセキュリティとインテグリティを維持できます。

### 手順

1. **WasmPlugin** を適用した Bookinfo アプリケーションの呼び出しを試してみます。認証が含まれていないため、拒否されるはずです。

```
$ export GATEWAY_URL=$(oc -n istio-system get route istio-ingressgateway -o jsonpath='{.spec.host}')
```

```
$ curl -I "http://$GATEWAY_URL/productpage"
HTTP/1.1 403
```

2. 認証用のユーザーキーを取得します。
  - `[Your_product_name]` > Applications > Listings に移動します。

- アプリケーションを選択します。
  - **Authentication > User Key** を探します。
3. ユーザーキーがある状態で呼び出しを再試行してください。

```
$ curl -I "http://$GATEWAY_URL/productpage?user_key=$USER_KEY"
HTTP/1.1 200 OK
```

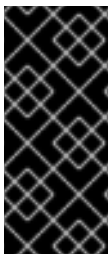
4. 呼び出しがメトリクスに登録されたことを確認します。
- **[Your\_product\_name] > Analytics > Traffic** に移動します。
  - 呼び出しが登録されていることが表示されるはずです。

## 11.8. 3SCALE WEBASSEMBLY モジュール設定

**WasmPlugin** カスタムリソース仕様は、**Proxy-WASM** モジュールが読み取る設定を提供します。

仕様はホストに組み込まれ、**Proxy-WASM** モジュールによって読み取られます。通常は、モジュールが解析するように設定は JSON ファイル形式です。ただし、**WasmPlugin** リソースは仕様値を YAML として解釈し、モジュールで使用できるように JSON に変換できます。

スタンドアロンモードで **Proxy-WASM** モジュールを使用する場合は、JSON 形式を使用して設定を作成する必要があります。JSON 形式を使用する場合は、**host** 設定ファイル内の必要な場所で、エスケープと引用を使用できます (例:**Envoy**)。 **WasmPlugin** リソースで WebAssembly モジュールを使用する場合、設定は YAML 形式になります。この場合は、無効な設定により、JSON 表現に基づいて診断がモジュールによって強制的にサイドカーコンテナのログングストリームに表示されます。



### 重要

**EnvoyFilter** カスタムリソースは一部の 3scale Istio アダプターまたは Service Mesh リリースで使用できますが、このカスタムリソースはサポートされる API ではありません。**EnvoyFilter** カスタムリソースの使用は推奨されていません。**EnvoyFilter** カスタムリソースの代わりに **WasmPlugin** API を使用します。**EnvoyFilter** カスタムリソースを使用する必要がある場合は、仕様を JSON 形式で指定する必要があります。

### 11.8.1. 3scale WebAssembly モジュールの設定

3scale の WebAssembly モジュール設定のアーキテクチャーは、3scale アカウントおよび承認サービスや処理するサービスのリストによって異なります。

#### 前提条件

前提条件は、すべてのケースで最小の必須フィールドのセットです。

- 3scale アカウントおよび承認サービス:**backend-listener** URL。
- 処理するサービスリスト: サービス ID と少なくとも 1 つの認証情報の検索方法、およびその検索場所。
- **userkey**、**appid**、**appkey**、および OpenID Connect (OIDC) パターンを処理する例がありません。
- WebAssembly モジュールは、静的設定で指定した設定を使用します。たとえば、モジュールにマッピングルール設定を追加する場合は、3scale 管理ポータルにこのようなマッピングルール



が設定されていない場合でも、常に適用されます。残りの **WasmPlugin** リソースは **spec.pluginConfig** YAML エントリーに存在します。

### 11.8.2. 3scale WebAssembly モジュール **api** オブジェクト

3scale WebAssembly モジュールからの **api** 最上位文字列は、モジュールが使用する設定のバージョンを定義します。



#### 注記

存在しないバージョンまたはサポート対象外のバージョンの **api** オブジェクトの場合、レンダリングされた 3scale WebAssembly モジュールは操作不能になります。

#### api 最上位文字列の例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info>
spec:
  pluginConfig:
    api: v1
...
```

**api** エントリーは、設定の残りの値を定義します。許可される値は **v1** のみです。現在の設定との互換性が失われるか、**v1** を使用するモジュールが処理できない追加のロジックが必要な新しい設定には、別の値が必要になります。

### 11.8.3. 3scale WebAssembly モジュール **system** オブジェクト

**system** 最上位オブジェクトは、特定のアカウントの 3scale Account Management API にアクセスする方法を指定します。**upstream** フィールドは、オブジェクトの最も重要な部分です。**system** オブジェクトはオプションですが、3scale WebAssembly モジュールに完全に静的な設定を提供する場合を除き、推奨されます。後者は、3scale のシステム コンポーネントへの接続を提供したくない場合のオプションです。

**system** オブジェクトに加えて静的設定オブジェクトを指定する場合は、静的な設定オブジェクトが優先されます。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    system:
      name: <saas_porta>
      upstream: <object>
      token: <my_account_token>
      ttl: 300
...
```

表11.1 system オブジェクトフィールド

名前	説明	必須
<b>name</b>	3scale サービスの識別子 (現在、参照されていません)。	任意
<b>upstream</b>	問い合わせるネットワークホストの詳細。 <b>upstream</b> は、system として知られる 3scale Account Management API ホストを参照します。	はい
<b>token</b>	読み取り権限を持つ 3scale の個人アクセストークン。	はい
<b>ttl</b>	新規の変更を取得する前に、このホストから取得した設定を有効なものを見なす最小時間 (秒数)。デフォルトは 600 秒 (10 分) です。 <b>注記:</b> 最大の期間はありますが、モジュールは通常、この TTL が経過した後に妥当な時間内に設定を取得します。	任意

#### 11.8.4. 3scale WebAssembly モジュール upstream オブジェクト

**upstream** オブジェクトは、プロキシが呼び出しを実行できる外部ホストを説明しています。

```
apiVersion: maistra.io/v1
upstream:
  name: outbound|443|multitenant.3scale.net
  url: "https://myaccount-admin.3scale.net/"
  timeout: 5000
...
```

表11.2 upstream オブジェクトフィールド

名前	説明	必須
----	----	----

名前	説明	必須
<b>name</b>	<b>name</b> は自由形式の識別子ではありません。これは、プロキシ設定で定義される外部ホストの識別子です。スタンドアロン <b>Envoy</b> 設定の場合は、これは他のプロキシの <b>upstream</b> と呼ばれる <b>クラスター</b> 名にマッピングします。 <b>注記:</b> Service Mesh および 3scale Istio アダプターコントロールプレーンは、複数のフィールドの区切り文字として垂直バー ( ) を使用する形式に従って名前を設定します。この統合の目的上、常に <b>outbound &lt;port&gt; &lt;hostname&gt;</b> の形式を使用します。	はい
<b>url</b>	記述されたサービスにアクセスするための完全な URL。スキームによって暗示されていない限り、TCP ポートが含まれている必要があります。	はい
<b>Timeout</b>	応答にかかる時間がこの設定を超えたこのサービスへの接続がエラーとみなされるためのタイムアウト (ミリ秒単位)。デフォルトは 1000 秒です。	任意

### 11.8.5. 3scale WebAssembly モジュール backend オブジェクト

**backend** 最上位オブジェクトは、HTTP リクエストの承認および報告のために 3scale Service Management API にアクセスする方法を指定します。このサービスは、3scale の **バックエンド** コンポーネントによって提供されます。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    backend:
      name: backend
      upstream: <object>
    ...

```

表11.3 backend オブジェクトフィールド

名前	説明	必須
<b>name</b>	3scale バックエンドの識別子 (現在、参照されていません)。	任意
<b>upstream</b>	問い合わせるネットワークホストの詳細。これは、system として知られる 3scale Account Management API ホストを参照する必要があります。	有効。最も重要な必須フィールドです。

### 11.8.6. 3scale WebAssembly モジュール services オブジェクト

**services** の最上位オブジェクトは、**module** のこの特定のインスタンスで処理されるサービス識別子を指定します。

アカウントには複数のサービスがあるため、どのサービスが処理されているかを指定する必要があります。残りの設定は、サービスの設定方法に関するものです。

**services** フィールドは必須です。有用とするサービスを少なくとも1つ含める必要がある配列です。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
  services:
    - id: "2555417834789"
      token: service_token
      authorities:
        - "*.app"
        - 0.0.0.0
        - "0.0.0.0:8443"
      credentials: <object>
      mapping_rules: <object>
    ...

```

**services** 配列の各要素は、3scale サービスを表します。

表11.4 services オブジェクトフィールド

名前	説明	必須
<b>id</b>	この 3scale サービスの識別子 (現在、参照されていません)。	はい

名前	説明	必須
<b>token</b>	この <b>token</b> は、System 内のサービスのプロキシ設定にあるか、以下の <b>curl</b> コマンドを使用して System から取得できます。  <b>curl</b> " <b>https://&lt;system_host&gt;/admin/api/services/&lt;service_id&gt;/proxy/configs/production/latest.json?access_token=&lt;access_token&gt;</b> "   jq <b>'proxy_config.content.backend_authentication_value'</b>	任意
<b>authorities</b>	文字列の配列。それぞれが一致する URL の <b>認証局</b> を表します。これらの文字列は、アスタリスク (*)、正符号 (+)、および疑問符 (?) マッチャーに対応する glob パターンを受け入れます。	はい
<b>credentials</b>	検索する認証情報の種類と場所を定義するオブジェクト。	はい
<b>mapping_rules</b>	ヒットするマッピングルールおよび 3scale メソッドを表すオブジェクトの配列。	任意

### 11.8.7. 3scale WebAssembly モジュール credentials オブジェクト

**credentials** オブジェクトは **service** オブジェクトのコンポーネントです。 **credentials** は、検索する認証情報の種類と、このアクションを実行する手順を指定します。

すべてのフィールドはオプションですが、少なくとも1つの **user\_key** または **app\_id** を指定する必要があります。各認証情報を指定する順番は、モジュールによって事前確立されているために無関係です。各認証情報の1つのインスタンスのみを指定します。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
  services:
    - credentials:
      user_key: <array_of_lookup_queries>
      app_id: <array_of_lookup_queries>
      app_key: <array_of_lookup_queries>
    ...
```

表11.5 credentials オブジェクトフィールド

名前	説明	必須
<b>user_key</b>	これは、3scale ユーザーキーを定義する検索クエリーの配列です。ユーザーキーは、一般に API キーと呼ばれます。	任意
<b>app_id</b>	これは、3scale のアプリケーション識別子を定義する検索クエリーの配列です。アプリケーションの識別子は、3scale または <a href="#">Red Hat Single Sign-On (RH-SSO)</a> や OpenID Connect (OIDC) などのアイデンティティプロバイダーを使用して提供されます。成功して2つの値に解決するたびに、ここで指定された検索クエリーの解決で、 <b>app_id</b> と <b>app_key</b> が設定されます。	任意
<b>app_key</b>	これは、3scale のアプリケーションキーを定義する検索クエリーの配列です。解決される <b>app_id</b> のないアプリケーションキーは無意味なため、 <b>app_id</b> が指定されている場合のみこのフィールドを指定します。	任意

### 11.8.8. 3scale WebAssembly モジュール検索クエリー

**lookup query** オブジェクトは、**credentials** オブジェクトのフィールドの一部になります。特定の認証情報フィールドが検出され、処理される方法を指定します。評価されると、解決に成功すると、1つ以上の値が見つかったことを意味します。解決に失敗したことは、値が見つからなかったことを意味します。

**lookup queries** の配列は、ショートサーキットまたは関係を定義しています。いずれかのクエリーの正常な解決により、残りのクエリーの評価が停止され、値を指定の credential-type に割り当てます。アレイの各クエリーは、互いに独立しています。

**lookup query** は、1つのフィールド(ソースオブジェクト)で設定されています。これは、複数のソースタイプの1つになります。以下の例を参照してください。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
  services:
  - credentials:
    user_key:
```

```

- <source_type>: <object>
- <source_type>: <object>
...
app_id:
- <source_type>: <object>
...
app_key:
- <source_type>: <object>
...
...

```

**source** オブジェクトは、任意の **credentials** オブジェクトフィールド内のソースの配列の一部として存在します。**source-type** として参照されるオブジェクトフィールド名は、以下のいずれかになります。

- **header**: 検索クエリーは、HTTP リクエストヘッダーを入力として受け取ります。
- **query\_string**: **lookup query** は、URL クエリー文字列パラメーターを入力として受け取ります。
- **filter**: **lookup query** は、フィルターメタデータをインプットとして受け取ります。

すべての **source-type** オブジェクトには、少なくとも以下の2つのフィールドがあります。

表11.6 **source-type** オブジェクトフィールド

名前	説明	必須
<b>keys</b>	文字列の配列。それぞれが <b>key</b> で、入力データで検索されたエントリーを参照します。	はい
<b>ops</b>	<b>key</b> エントリーの照合を行う <b>operations</b> の配列。配列は、操作が入力を受け取り、次の操作の出力を生成するパイプラインです。出力に失敗した <b>operation</b> は、 <b>lookup query</b> を失敗として解決します。操作のパイプラインの順序により、評価の順序が決定されます。	任意
<b>path</b>	データの検索に使用されるメタデータ内のパスを表示します。ただし、 <b>header</b> または <b>query_string</b> ソースタイプを使用する場合は必要ありませんが、 <b>filter</b> ソースタイプを使用する場合は必須です。	任意

キーが入力データと一致する場合は、残りの鍵は評価されず、ソース解決アルゴリズムは、指定した **operations (ops)** の実行にジャンプします。**ops** を指定しないと、一致する **key** の結果値 (ある場合) が返されます。

**Operations** は、最初のフェーズが **key** を検索した後に、入力に対する特定の条件および変換を指定する方法を提供します。プロパティを変換、デコード、および要求する必要があるときに、**operations** を使用しますが、すべてのニーズに対応する成熟した言語は提供されず、**Turing-completeness** はありません。

スタックは **operations** の出力を保存します。評価されると、認証情報が消費する値の数に応じて、スタックの下部に値を割り当てて、**lookup query** は終了します。

### 11.8.9. 3scale WebAssembly モジュール operations オブジェクト

特定の **source type** に属する **ops** 配列の各要素は、値に変換を適用するか、テストを実行する **operation** オブジェクトです。このようなオブジェクトに使用するフィールド名は **operation** 自体の名前で、値は **operation** に対するパラメーターです。これは、フィールドと値のマップ、リスト、または文字列など、構造化オブジェクトになります。

ほとんどの **operations** は、1つ以上の入力を処理し、1つ以上の出力を生成します。入力を使用したり、出力を生成したりする場合、それらは値のスタックで作業します。操作によって消費される各値は、値のスタックからポップアップされ、**source** マッチと共に初期入力されます。出力される値はスタックにプッシュされます。他の **operations** は、特定のプロパティを要求する以外、出力を使用または生成しませんが、値のスタックを検査します。



#### 注記

解決が完了すると、次の手順 (値を **app\_id**、**app\_key**、または **user\_key** に割り当てるなど) でピックアップされる値はスタックの下部の値から取得されます。

**operations** カテゴリーはいくつかあります。

- **decode**: 別の形式を取得するために、入力値をデコードして変換します。
- **string**: 文字列値を入力として取り、変換を実行し、確認します。
- **stack**: 入力の値のセットを取得し、複数のスタック変換とスタック内の特定の位置の選択を実行します。
- **check**: 影響を及ぼさない方法で、操作セットに関するプロパティを要求します。
- **control**: 評価フローを変更できる操作を実施します。
- **format**: 入力値の形式固有の構造を解析し、その値を検索します。

すべての操作は、name 識別子で文字列として指定されます。

#### 関連情報

- [利用可能な操作](#)

### 11.8.10. 3scale WebAssembly モジュール mapping\_rules オブジェクト

**mapping\_rules** オブジェクトは **service** オブジェクトの一部です。これは、REST パスパターンのセットならびに関連する 3scale メトリックおよびパターンが一致する時に使用するカウント増分を指定します。



**system** 最上位オブジェクトに動的設定が提供されていない場合は、値が必要です。**system** 最上位エントリーに加えてオブジェクトが提供されると、**mapping\_rules** オブジェクトが最初に評価されます。

**mapping\_rules** は配列オブジェクトです。そのアレイの各要素は **mapping\_rule** オブジェクトです。受信したリクエストの評価されたマッチするマッピングルールにより、承認およびAPIManager へのレポート用の 3scale **methods** のセットが提供されます。複数のマッチングルールが同じ **methods** を参照する場合は、3scale への呼び出し時に **deltas** の合算があります。たとえば、2つのルールが、1と3の **deltas** で **Hits** メソッドを2回増やすと、3scale にレポートする Hits の単一のメソッドエントリーの **delta** は4になります。

### 11.8.11. 3scale WebAssembly モジュール mapping\_rule オブジェクト

**mapping\_rule** オブジェクトは **mapping\_rules** オブジェクトの配列の一部です。

**mapping\_rule** オブジェクトフィールドは、以下の情報を指定します。

- 照合する HTTP 要求メソッド。
- パスに一致するパターン。
- 報告する量と共にレポートする 3scale メソッド。フィールドを指定する順番により、評価順序が決定されます。

表11.7 mapping\_rule オブジェクトフィールド

名前	説明	必須
メソッド	HTTP リクエストメソッド (動詞) を表す文字列を指定します。許可される値は、許可される HTTP メソッド名の1つと一致し、大文字と小文字を区別しません。すべてのマッチのすべてのメソッドの特殊な値。	はい
pattern	HTTP リクエストの URI パスコンポーネントに一致するパターン。このパターンは、3scale で説明されている構文に従います。ワイルドカード、アスタリスク (*) 文字の使用、中括弧間の任意の文字シーケンスの使用 ( <b>{this}</b> など) が許可されます。	はい

名前	説明	必須
<b>usages</b>	<p><b>usage</b> オブジェクトのリスト。ルールがマッチすると、<b>deltas</b> を持つすべてのメソッドが、承認およびレポートのために 3scale に送信されるメソッドのリストに追加されます。</p> <p>以下の必須フィールドに <b>usages</b> オブジェクトを埋め込みます。</p> <ul style="list-style-type: none"> <li>● <b>name</b>: レポートする <b>method</b> のシステム名。注記:<b>name</b> は大文字と小文字を区別します。</li> <li>● <b>delta</b>: その <b>method</b> の増分。</li> </ul>	はい
<b>last</b>	このルールが正常にマッチした場合に、それ以外のマッピングルールの評価を停止する必要があるかどうか。	任意のブール値。デフォルトは <b>false</b> です。

以下の例は、3scale のメソッド間の既存の階層とは独立しています。つまり、3scale 側で実行されたすべての内容はこれには影響しません。たとえば、**Hits** メトリックは、それらすべての親となる可能性があるため、承認されたリクエストで報告されたすべてのメソッドの合計により 4 ヒットを保管し、3scale **Authrep** API エンドポイントを呼び出します。

以下の例では、すべてのルールに一致する、パス **/products/1/sold** への **GET** リクエストを使用します。

### mapping\_rules GET リクエストの例

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
  mapping_rules:
    - method: GET
      pattern: /
      usages:
        - name: hits
          delta: 1
    - method: GET
      pattern: /products/
      usages:
        - name: products

```

```

    delta: 1
  - method: ANY
  pattern: /products/{id}/sold
  usages:
    - name: sales
      delta: 1
    - name: products
      delta: 1
  ...

```

すべての **usages** は、モジュールが使用状況データを使用して 3scale に実施するリクエストに追加されます。

- Hits: 1
- products: 2
- sales: 1

## 11.9. 認証情報ユースケースの 3SCALE WEBASSEMBLY モジュールの例

ほとんどの時間を費やして、設定手順を適用してサービスへのリクエストの認証情報を取得します。

以下は **credentials** の例です。これは、特定のユースケースに合わせて変更できます。

複数のソースオブジェクトと独自の **lookup queries** を指定する場合、これらはすべて組み合わせることができ、いずれか1つが正しく解決されるまで、それらは順番に評価されます。

### 11.9.1. クエリー文字列パラメーターの API キー (user\_key)

以下の例では、クエリー文字列パラメーターまたは同じ名前のヘッダーで **user\_key** を検索します。

```

credentials:
  user_key:
    - query_string:
      keys:
        - user_key
    - header:
      keys:
        - user_key

```

### 11.9.2. アプリケーション ID およびキー

以下の例では、クエリーまたはヘッダーの **app\_key** および **app\_id** 認証情報を検索します。

```

credentials:
  app_id:
    - header:
      keys:
        - app_id
    - query_string:
      keys:
        - app_id
  app_key:

```

```
- header:
  keys:
    - app_key
- query_string:
  keys:
    - app_key
```

### 11.9.3. 認証ヘッダー

リクエストには、**authorization** ヘッダーに **app\_id** および **app\_key** が含まれます。最後に出力される値が1つまたは2つある場合は、**app\_key** を割り当てることができます。

ここでの解決は、最後に出力された1つまたは2つの出力がある場合は **app\_key** を割り当てます。

**authorization** ヘッダーは承認の種類で値を指定し、その値は **Base64** としてエンコードされます。つまり、値を空白文字で分割し、2番目の出力を取得して、コロン (:) をセパレーターとして使用して再度分割できます。たとえば、**app\_id:app\_key** という形式を使用する場合、ヘッダーは以下の **credential** の例のようになります。

```
aladdin:opensesame: Authorization: Basic YWxhZGRpbjpvGVuc2VzYW11
```

次の例に示すように、ヘッダーフィールド名は小文字を使用する必要があります。

```
credentials:
  app_id:
    - header:
      keys:
        - authorization
  ops:
    - split:
      separator: " "
      max: 2
    - length:
      min: 2
    - drop:
      head: 1
    - base64_urlsafesafe
    - split:
      max: 2
  app_key:
    - header:
      keys:
        - app_key
```

前述のユースケースの例は、**authorization** のヘッダーを確認します。

1. これは文字列の値を取り、スペースで分割し、**credential**-type および **credential** 自体の少なくとも2つの値を生成することを確認してから、**credential**-type をドロップします。
2. 次に、必要なデータが含まれる2番目の値をデコードし、最初の **app\_id** の後にもしあれば **app\_key** が含まれる操作スタックとなるように、コロン (:) 文字を使用して分割します。
  - a. **app\_key** が認証ヘッダーに存在しない場合は、その特定のソースがチェックされます。たとえば、この場合はヘッダーにキー **app\_key** が付いています。

3. **credentials** に追加の条件を追加するには、**Basic** 認証を許可します。ここで、**app\_id** は **aladdin** もしくは **admin**、または長さが 8 文字以上の任意の **app\_id** になります。
4. **app\_key** には値が含まれ、以下の例のように最小で 64 文字を指定する必要があります。

```
credentials:
  app_id:
    - header:
      keys:
        - authorization
      ops:
        - split:
            separator: " "
            max: 2
        - length:
            min: 2
        - reverse
        - glob:
            - Basic
        - drop:
            tail: 1
        - base64_urlsafe
        - split:
            max: 2
        - test:
            if:
              length:
                min: 2
            then:
              - strlen:
                  max: 63
              - or:
                  - strlen:
                      min: 1
                  - drop:
                      tail: 1
            - assert:
                - and:
                    - reverse
                - or:
                    - strlen:
                        min: 8
                    - glob:
                        - aladdin
                        - admin
```

5. **authorization** ヘッダーの値を選択したら、タイプが上部に配置されるようにスタックを逆にして **Basic credential**-type を取得します。
6. glob マッチを実行します。検証し、認証情報がデコードされ、分割されると、スタックの下部に **app\_id** を取得し、上部に **app\_key** を取得する可能性があります。
7. **test:** を実行します。スタックに 2 つの値がある場合は、**app\_key** が取得されたことになりません。

- a. **app\_id** および **app\_key** を含め、文字列の長さが1から 63 文字になるようにします。キーの長さがゼロの場合は破棄し、キーが存在しないものとして続行します。**app\_id** のみがあり、**app\_key** がない場合、不明なブランチは、テストに成功し、評価が続行されます。

最後の操作は **assert** で、スタックに副作用がないことを示します。その後、スタックを変更できません。

1. **app\_id** が最上部になるように、スタックを逆にします。
  - a. **app\_key** が存在するかどうかで、スタックを逆にすると、**app\_id** が上部になります。
2. **and** を使用して、テスト間でスタックの内容を保持します。次に、以下のいずれかの方法を使用します。
  - **app\_id** に 8 文字以上の文字列が設定されていることを確認してください。
  - **app\_id** が **aladdin** または **admin** と一致していることを確認します。

#### 11.9.4. OpenID Connect (OIDC) のユースケース

Service Mesh および 3scale Istio アダプターの場合は、以下の例のように **RequestAuthentication** をデプロイし、独自のワークロードデータおよび **jwtRules** を入力する必要があります。

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: <info>
spec:
  selector:
    matchLabels:
      app: <productpage>
  jwtRules:
    - issuer: >-
      "<url>/auth/realms/<realm_name>"
      jwksUri: >-
        "<url>/auth/realms/<realm_name>/protocol/openid-connect/certs"
```

**RequestAuthentication** を適用するとき、**JWT** トークンを検証するために **ネイティブプラグイン** で **Envoy** を設定します。プロキシは、モジュールを実行する前にすべてを検証します。したがって、失敗したリクエストが 3scale WebAssembly モジュールに実行されません。

**JWT** トークンが検証されると、プロキシはそのコンテンツを内部メタデータオブジェクトに格納します。エントリーのキーは、プラグインの特定の設定に依存します。このユースケースでは、不明なキー名が含まれる単一のエントリーを持つ構造化オブジェクトを検索できます。

OIDC の 3scale **app\_id** は、OAuth **client\_id** と一致します。これは **JWT** トークンの **azp** フィールドまたは **aud** フィールドにあります。

Envoy のネイティブ **JWT** 認証フィルターから **app\_id** フィールドを取得するには、以下の例を参照してください。

```
credentials:
  app_id:
    - filter:
      path:
```

```

- envoy.filters.http.jwt_authn
- "0"
keys:
- azp
- aud
ops:
- take:
  head: 1

```

この例では、モジュールに対し、**filter** ソースタイプを使用して **Envoy** 固有の **JWT** 認証ネイティブプラグインからオブジェクトのフィルターメタデータを検索するよう指示します。このプラグインには、1つのエントリーと事前に設定された名前を持つ構造化オブジェクトの一部として **JWT** トークンが含まれます。**0** を使用して、単一のエントリーのみアクセスするように指定します。

結果の値は、以下の2つのフィールドを解決する構造です。

- **azp: app\_id** が見つけられる値。
- **aud**: この情報も見つけられる値。

この操作により、割り当て用に1つの値のみが保持されます。

### 11.9.5. ヘッダーからの JWT トークンの取得

一部のセットアップには、**JWT** トークンの検証プロセスがあり、検証されたトークンが JSON 形式のヘッダーを介してこのモジュールに到達する場合があります。

**app\_id** を取得するには、以下の例を参照してください。

```

credentials:
  app_id:
    - header:
      keys:
        - x-jwt-payload
      ops:
        - base64_urlsafe
        - json:
            - keys:
                - azp
                - aud
            - take:
                head: 1

```

## 11.10. 3SCALE WEBASSEMBLY モジュールの機能する最低限の設定

以下は、3scale WebAssembly モジュールの機能する最低限の設定の例です。これをコピーアンドペーストし、これを独自の設定で機能するように編集できます。

```

apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  imagePullSecret: <pull_secret_resource>

```

```
phase: AUTHZ
match:
  - mode: SERVER
priority: 100
selector:
  matchLabels:
    app: <productpage>
pluginConfig:
  api: v1
  system:
    name: <system_name>
  upstream:
    name: outbound|443|multitenant.3scale.net
    url: https://istiodevel-admin.3scale.net/
    timeout: 5000
  token: <token>
backend:
  name: <backend_name>
  upstream:
    name: outbound|443|su1.3scale.net
    url: https://su1.3scale.net/
    timeout: 5000
  extensions:
    - no_body
services:
  - id: '2555417834780'
  authorities:
    - "*"
  credentials:
    user_key:
      - query_string:
          keys:
            - <user_key>
      - header:
          keys:
            - <user_key>
    app_id:
      - query_string:
          keys:
            - <app_id>
      - header:
          keys:
            - <app_id>
    app_key:
      - query_string:
          keys:
            - <app_key>
      - header:
          keys:
            - <app_key>
```

## 関連情報

- [ServiceMeshExtension から WasmPlugin リソースへの移行](#)
- [Kubernetes カスタムリソース](#)



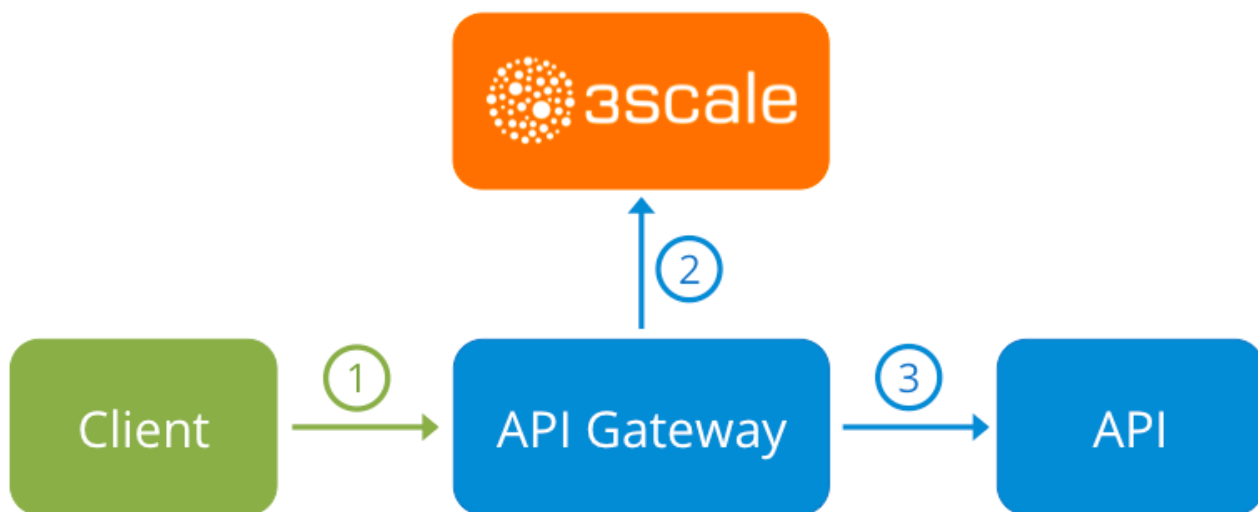
- [Wasm Plugin](#)

## 第12章 API インフラストラクチャーに関するトラブルシューティング

本章の目的は、ユーザーが API インフラストラクチャーに関連する問題の原因を特定して修正できるように支援することです。

API インフラストラクチャーは、長く複雑なトピックです。ただし、少なくとも、インフラストラクチャーには3つの可動部分があります。

1. API ゲートウェイ
2. 3scale
3. API



これらの3つの要素のいずれかでエラーが起これると、APIの利用者はAPIにアクセスできなくなります。ただし、障害の原因となったコンポーネントを特定することは困難です。本章では、インフラストラクチャーのトラブルシューティングを行って問題を特定するためのヒントを紹介します。

以下のセクションを参照して、発生する可能性のある典型的な問題の特定および修正を行います。

- [「インテグレーションに関する典型的な問題」](#)
- [「API インフラストラクチャーに関する問題への対応」](#)
- [「API へのリクエストに関する問題の特定」](#)
- [「ActiveDocs の問題」](#)
- [「NGINX でのロギング」](#)
- [「3scale のエラーコード」](#)

### 12.1. インテグレーションに関する典型的な問題

3scale とのインテグレーションに関する非常に典型的な問題を示す症状がいくつかあります。これらは、API プロジェクトの最初の段階であるのか、インフラストラクチャーをセットアップしているのか、すでに実稼働環境に移行しているのかによって異なります。

### 12.1.1. インテグレーションの問題

以降のセクションで、3scale とのインテグレーションにおける初期段階 (Hosted APIcast 使用の初期段階および実稼働環境への移行前、ならびに Self-managed APIcast の稼働中) で、APIcast エラーログでよく見られる問題のいくつかについて概要を説明します。

#### 12.1.1.1. Hosted APIcast

Service Integration 画面で API と Hosted APIcast を初めて統合する場合、以下のエラーのいずれかがページに表示されたり、インテグレーションの成功を確認するためのテストコールでエラーが返されたりする可能性があります。

- **Test request failed: execution expired**  
API が一般のインターネットからアクセス可能であることを確認します。Hosted APIcast は、プライベート API を扱うことができません。Hosted APIcast と統合する際に API を一般に公開したくない場合は、Hosted APIcast と API との間にプライベートシークレットを設定し、API ゲートウェイ以外からの呼び出しを拒否することができます。
- 設定可能なフォーマットは **protocol://address(:port)** です。  
API のプライベートベース URL の最後にあるパスを削除します。これらのパスは、マッピングルールのパターン、または API テスト GET リクエストの最初に追加できます。
- **テストリクエストが失敗し HTTP コード XXX が返される**
  - **405:** エンドポイントが GET リクエストを受け入れることを確認します。APIcast は、インテグレーションをテストするための GET リクエストのみをサポートしています。
  - **403: Authentication parameters missing:** API にすでに何らかの認証が設定されている場合は、APIcast はテストリクエストを送信することができません。
  - **403: Authentication failed:** 3scale でこれ以前にサービスを作成したことがある場合は、テストリクエストを行うためのクレデンシャルが設定されたサービスでアプリケーションを作成していることを確認します。これが統合する最初のサービスである場合は、サインアップ時に作成したテストアカウントまたはアプリケーションを削除していないことを確認します。

#### 12.1.1.2. Self-managed APIcast

Self-managed APIcast とのインテグレーションのテストが正常に終了したら、API ゲートウェイを独自にホストすることが望ましい場合があります。以下は、自己管理型ゲートウェイを初めてインストールし、これを介して API を呼び出す際に生じる可能性のあるエラーです。

- **upstream timed out (110: Connection timed out) while connecting to upstream**  
API ゲートウェイと一般のインターネットの間に、Self-managed APIcast ゲートウェイが 3scale に到達するのを妨げるファイアウォールまたはプロキシがないことを確認します。
- **failed to get list of services: invalid status: 403 (Forbidden)**

```
2018/06/04 08:04:49 [emerg] 14#14: [lua] configuration_loader.lua:134: init(): failed to load configuration, exiting (code 1)
```

```
2018/06/04 08:04:49 [warn] 22#22: *2 [lua] remote_v2.lua:163: call(): failed to get list of services: invalid status: 403 (Forbidden) url: https://example-admin.3scale.net/admin/api/services.json , context: ngx.timer
```

```
ERROR: /opt/app-root/src/src/apicast/configuration_loader.lua:57: missing configuration
```

**THREESCALE\_PORTAL\_ENDPOINT** の値に使用するアクセストークンが正しいこと、またス

コープに Account Management API が含まれていることを確認します。そのためには、**curl** コマンドを使用して確認します (**curl -v "https://example-admin.3scale.net/admin/api/services.json?access\_token=<YOUR\_ACCESS\_TOKEN>"**)。

JSON ボディーでレスポンス 200 が返されるはずですが、エラーステータスコードを返す場合は、レスポンスのボディーで詳細を確認します。

- **service not found for host apicast.example.com**

```
2018/06/04 11:06:15 [warn] 23#23: *495 [lua] find_service.lua:24: find_service(): service not found for host apicast.example.com, client: 172.17.0.1, server: _, request: "GET / HTTP/1.1", host: "apicast.example.com"
```

このエラーは、公開ベース URL が正しく設定されていないことを示しています。設定された公開ベース URL は、Self-managed APIcast へのリクエストに使用するものと同じにする必要があります。正しい公開ベース URL を設定した後、以下を実行します。

- APIcast が実稼働用に設定されていることを確認します (**THREESCALE\_DEPLOYMENT\_ENV** 変数で上書きされていない場合のスタンドアロン APIcast のデフォルト設定)。必ず設定を実稼働環境にプロモートしてください。
- 環境変数 **APICAST\_CONFIGURATION\_CACHE** と **APICAST\_CONFIGURATION\_LOADER** を使用して自動的に設定を再読み込みするように設定していなかった場合は、APIcast を再起動します。

以下は、Self-managed APIcast の誤ったインテグレーションを示すその他の症状の例です。

- **マッピングルールの不一致/APIコールの二重カウント**: メソッドと API の実際の URL エンドポイント間のマッピングをどのように定義したかによって、場合により、メソッドが一致しない、またはリクエストごとに複数回カウントが増加することがあります。この問題のトラブルシューティングを行うには、[3scale デバッグヘッダー](#) を使用して API にテストコールを行います。これにより、API コールで一致したすべてのメソッドのリストが返されます。
- **認証パラメーターが見つからない**: パラメーターを Service Integration 画面で指定した正しい場所へ送信していることを確認します。クレデンシャルをヘッダーとして送信しない場合、GET リクエストについてはクエリーパラメーターとして、その他の HTTP メソッドについてはボディーパラメーターとして送信する必要があります。3scale デバッグヘッダーを使用して、API ゲートウェイによりリクエストから読み取られるクレデンシャルを再確認します。

## 12.1.2. 実稼働環境の問題

セットアップを完全にテストし、しばらくの間実際に API を運用した後に、API ゲートウェイに関連して問題が発生することはほとんどありません。ただし、実際の実稼働環境で発生しうる問題の一部をここに挙げます。

### 12.1.2.1. 可用性の問題

可用性の問題は、通常、nginx error.log に **upstream timed out** エラーが表示されることが特徴です。以下に例を示します。

```
upstream timed out (110: Connection timed out) while connecting to upstream, client: X.X.X.X, server: api.example.com, request: "GET /RESOURCE?CREDENTIALS HTTP/1.1", upstream: "http://Y.Y.Y.Y:80/RESOURCE?CREDENTIALS", host: "api.example.com"
```

断続的に 3scale の可用性の問題が発生する場合、以下が原因の可能性がります。

- 使用されていない古い 3scale IP に解決しようとしている。  
最新バージョンの API ゲートウェイ設定ファイルは、毎回強制的に IP を解決するために、変数として 3scale を定義します。応急処置として、NGINX インスタンスを再読み込みします。長期的な修正としては、アップストリームブロックで 3scale バックエンドを定義するのではなく、たとえば以下のように、各サーバーブロック内の変数として定義します。

```
server {
    # Enabling the Lua code cache is strongly encouraged for production use. Here it is enabled
    .
    .
    .
    set $threescale_backend "https://su1.3scale.net:443";
```

これを参照する場合は、以下のとおりです。

```
location = /threescale_authrep {
    internal;
    set $provider_key "YOUR_PROVIDER_KEY";

    proxy_pass $threescale_backend/transactions/authrep.xml?
    provider_key=$provider_key&service_id=$service_id&$usage&$credentials&log%5Bcode%5
    D=$arg_code&log%5Brequest%5D=$arg_req&log%5Bresponse%5D=$arg_resp;
}
```

- すべての 3scale IP がホワイトリスト上に記載されていない。3scale が解決する IP の現在のリストを以下に示します。
  - 75.101.142.93
  - 174.129.235.69
  - 184.73.197.122
  - 50.16.225.117
  - 54.83.62.94
  - 54.83.62.186
  - 54.83.63.187
  - 54.235.143.255

上記の問題は、3scale の可用性の問題と考えられます。ただし、API が AWS ELB の背後に置かれている場合、API ゲートウェイからの API 可用性に同様の問題が発生する可能性があります。これは、デフォルトでは NGINX が起動時に DNS 解決を行ってから IP アドレスをキャッシュするためです。ただし、ELB は静的 IP アドレスを確保せず、頻繁に変わる可能性があります。ELB が別の IP に変わると、NGINX はその IP に到達できません。

この問題の解決方法は、強制的にランタイム DNS 解決を行う上述の修正と類似していません。

1. **http** セクションの最上部に **resolver 8.8.8.8 8.8.4.4;** という行を追加して、Google DNS などの特定の DNS リゾルバーを設定します。
2. **server** セクションの最上部近くの任意の場所に、API のベース URL を変数として設定します **set \$api\_base "http://api.example.com:80";**

3. **location** / セクション内で **proxy\_pass** の行を探し、それを **proxy\_pass \$api\_base;** に置き換えます。

### 12.1.3. デプロイ後の問題

新しいエンドポイントを追加するなど、API に変更を加える場合、API ゲートウェイの新しい設定ファイルのセットをダウンロードする前に、必ず新しいメソッドおよび URL マッピングを追加する必要があります。

3scale からダウンロードした設定を変更した場合の最も典型的な問題は、Lua のコードエラーです。これにより、以下のような **500 - Internal server error** が発生します。

```
curl -v -X GET "http://localhost/"
* About to connect() to localhost port 80 (#0)
* Trying 127.0.0.1... connected
> GET / HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
librtmp/2.3
> Host: localhost
> Accept: */*
>
< HTTP/1.1 500 Internal Server Error
< Server: openresty/1.5.12.1
< Date: Thu, 04 Feb 2016 10:22:25 GMT
< Content-Type: text/html
< Content-Length: 199
< Connection: close
<

<head><title>500 Internal Server Error</title></head>

<center><h1>500 Internal Server Error</h1></center>
<hr/><center>openresty/1.5.12.1</center>

* Closing connection #0
```

nginx error.log を見て、原因を確認することができます。以下に例を示します。

```
2016/02/04 11:22:25 [error] 8980#0: *1 lua entry thread aborted: runtime error:
/home/pili/NGINX/troubleshooting/nginx.lua:66: bad argument #3 to '_newindex' (number expected,
got nil)
stack traceback:
coroutine 0:
  [C]: in function '_newindex'
  /home/pili/NGINX/troubleshooting/nginx.lua:66: in function 'error_authorization_failed'
  /home/pili/NGINX/troubleshooting/nginx.lua:330: in function 'authrep'
  /home/pili/NGINX/troubleshooting/nginx.lua:283: in function 'authorize'
  /home/pili/NGINX/troubleshooting/nginx.lua:392: in function while sending to client, client:
127.0.0.1, server: api-2445581381726.staging.apicast.io, request: "GET / HTTP/1.1", host: "localhost"
```

access.log では、以下のようになります。

```
127.0.0.1 - - [04/Feb/2016:11:22:25 +0100] "GET / HTTP/1.1" 500 199 "-" "curl/7.22.0 (x86_64-pc-
linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
```

上記のセクションでは、3scale 運用のいずれかのステージで発生する可能性のある最も典型的でよく知られた問題の概要を示します。

これらをすべて確認してもなお問題の原因と解決策が見つからない場合は、[API へのリクエストに関する問題の特定](#)で説明する、より詳細なトラブルシューティングに進む必要があります。問題のある箇所の特定制を試みるため、API から始めてクライアントまで遡って作業します。

## 12.2. API インフラストラクチャーに関する問題への対応

サーバーへの接続時にエラーが発生する場合、それが API ゲートウェイでも、3scale でも、またはご自分の API でも、まずは以下のトラブルシューティング手順から作業を始めてください。

### 12.2.1. 接続の可否確認

telnet を使用して、基本的な TCP/IP 接続 (**telnet api.example.com 443**) を確認します。

- 正常に接続できる場合

```
telnet echo-api.3scale.net 80
Trying 52.21.167.109...
Connected to tf-lb-i2t5pgt2cfdnbdfh2c6qqoartm-829217110.us-east-1.elb.amazonaws.com.
Escape character is '^]'.
Connection closed by foreign host.
```

- 接続できない場合

```
telnet su1.3scale.net 443
Trying 174.129.235.69...
telnet: Unable to connect to remote host: Connection timed out
```

### 12.2.2. サーバー接続の問題

さまざまなネットワークの場所、デバイス、および宛先から、同じサーバーへの接続を試みます。たとえば、クライアントが API に到達できない場合は、API ゲートウェイなど、アクセスできるはずのマシンから API への接続を試みます。

接続試行のいずれかが成功した場合、実際のサーバーに関する問題を除外して、両者間のネットワークのトラブルシューティングに集中できます。問題がここにある可能性が最も高いからです。

### 12.2.3. DNS での問題の有無確認

ホスト名の代わりに IP アドレスを使用してサーバーへの接続を試みます。たとえば、**telnet apis.io 80** の代わりに **telnet 94.125.104.17 80** を使用します。

これにより、DNS に関する問題はすべて排除されます。

3scale の例では、**dig su1.3scale.net** または **dig any su1.3scale.net** (ホストが解決する IP が複数あると思われる場合) のように、**dig** を使用してサーバーの IP アドレスを取得することができます。

**注記:** 一部のホストは **dig any** をブロックします

### 12.2.4. SSL に問題がないか調べる



OpenSSL を使用して、以下の項目をテストすることができます。

- ホストまたは IP へのセキュアな接続 (たとえば、シェルプロンプトから **openssl s\_client -connect su1.3scale.net:443** を実行)

出力:

```
CONNECTED(00000003)
depth=1 C = US, O = GeoTrust Inc., CN = GeoTrust SSL CA - G3
verify error:num=20:unable to get local issuer certificate
---
Certificate chain
 0 s:/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks, S.L./OU=IT/CN=*.3scale.net
  i:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
 1 s:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
  i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIE8zCCA9ugAwIBAgIQcz2Y9JNxxH7f2zpOT0DajUjANBgkqhkiG9w0BAQsFADBE
...
TRUNCATED
...
3FZigX+OpWLVrjYsr0kZzX+HCerYMwc=
-----END CERTIFICATE-----
subject=/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks,
S.L./OU=IT/CN=*.3scale.net
issuer=/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
---
Acceptable client certificate CA names
/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks, S.L./OU=IT/CN=*.3scale.net
/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
Client Certificate Types: RSA sign, DSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384
:RSA+SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA22
4:RSA+SHA1:DSA+SHA1:ECDSA+SHA1:RSA+MD5
Shared Requested Signature Algorithms:
RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384
:RSA+SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA22
4:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA512
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 3281 bytes and written 499 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
  Protocol : TLSv1.2
  Cipher  : ECDHE-RSA-AES256-GCM-SHA384
  Session-ID:
A85EFD61D3BFD6C27A979E95E66DA3EC8F2E7B3007C0166A9BCBDA5DCA5477B8
```



```

Session-ID-ctx:
Master-Key:
F7E898F1D996B91D13090AE9D5624FF19DFE645D5DEEE2D595D1B6F79B1875CF935B3
A4F6ECCA7A6D5EF852AE3D4108B
Key-Arg : None
PSK identity: None
PSK identity hint: None
SRP username: None
TLS session ticket lifetime hint: 300 (seconds)
TLS session ticket:
0000 - a8 8b 6c ac 9c 3c 60 78-2c 5c 8a de 22 88 06 15  ..!..<`x,\.."...
0010 - eb be 26 6c e6 7b 43 cc-ae 9b c0 27 6c b7 d9 13  ..&!.{C....'!...
0020 - 84 e4 0d d5 f1 ff 4c 08-7a 09 10 17 f3 00 45 2c  .....L.z.....E,
0030 - 1b e7 47 0c de dc 32 eb-ca d7 e9 26 33 26 8b 8e  ..G...2....&3&..
0040 - 0a 86 ee f0 a9 f7 ad 8a-f7 b8 7b bc 8c c2 77 7b  .....{...w{
0050 - ae b7 57 a8 40 1b 75 c8-25 4f eb df b0 2b f6 b7  ..W.@.u.%O...+..
0060 - 8b 8e fc 93 e4 be d6 60-0f 0f 20 f1 0a f2 cf 46  .....`.. ....F
0070 - b0 e6 a1 e5 31 73 c2 f5-d4 2f 57 d1 b0 8e 51 cc  ....1s.../W...Q.
0080 - ff dd 6e 4f 35 e4 2c 12-6c a2 34 26 84 b3 0c 19  ..nO5.,l.4&....
0090 - 8a eb 80 e0 4d 45 f8 4a-75 8e a2 06 70 84 de 10  ....ME.Ju...p...

Start Time: 1454932598
Timeout : 300 (sec)
Verify return code: 20 (unable to get local issuer certificate)
---
```

- SSLv3 のサポート (3scale ではサポートされません)  
**openssl s\_client -ssl3 -connect su.3scale.net:443**

出力

```

CONNECTED(00000003)
140735196860496:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert handshake
failure:s3_pkt.c:1456:SSL alert number 40
140735196860496:error:1409E0E5:SSL routines:ssl3_write_bytes:ssl handshake
failure:s3_pkt.c:644:
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 0 bytes
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
  Protocol : SSLv3
  Cipher   : 0000
  Session-ID:
  Session-ID-ctx:
  Master-Key:
  Key-Arg : None
  PSK identity: None
```

```
PSK identity hint: None
SRP username: None
Start Time: 1454932872
Timeout : 7200 (sec)
Verify return code: 0 (ok)
---
```

詳細は、[OpenSSL の man ページ](#) を参照してください。

## 12.3. API へのリクエストに関する問題の特定

API に対するリクエストのどこに問題があるのかを特定するには、以下のリストに従って確認を行います。

### 12.3.1. API

API が動作状態にあり、リクエストに応答していることを確認するため、同じリクエストを API に対し直接、API ゲートウェイを経由せずに実行します。API ゲートウェイを経由する場合のリクエストと同じパラメーターおよびヘッダーを送信していることを確認する必要があります。失敗したリクエストが正確にわからない場合は、API ゲートウェイと API 間のトラフィックを取得します。

呼び出しに成功する場合、API に関する問題を除外できますが、失敗した場合には、さらに API のトラブルシューティングを行う必要があります。

### 12.3.2. API ゲートウェイ > API

API ゲートウェイと API 間のネットワークの問題を除外するには、前と同じ呼び出しを、API に直接、ゲートウェイサーバーから実行します。

呼び出しに成功する場合、API ゲートウェイ自体のトラブルシューティングに進むことができます。

### 12.3.3. API ゲートウェイ

API ゲートウェイが正常に機能していることを確認するためには、多くのステップを順に実施します。

#### 12.3.3.1. API ゲートウェイの起動および稼働確認

ゲートウェイが稼働しているマシンにログインします。これに失敗する場合、ゲートウェイサーバーがダウンしている可能性があります。

ログインしたら、NGINX プロセスが実行中であることを確認します。そのためには、`ps ax | grep nginx` または `htop` を実行します。

リストに `nginx master process` と `nginx worker process` が表示されている場合、NGINX は稼働中です。

#### 12.3.3.2. ゲートウェイログでのエラーの有無確認

以下は、`error.log` のゲートウェイログで表示される可能性のある典型的なエラーの例です。

- API ゲートウェイが API に接続できない

```
upstream timed out (110: Connection timed out) while connecting to upstream, client:
X.X.X.X, server: api.example.com, request: "GET /RESOURCE?CREDENTIALS HTTP/1.1",
upstream: "http://Y.Y.Y.Y:80/RESOURCE?CREDENTIALS", host: "api.example.com"
```

- API ゲートウェイが 3scale に接続できない

```
2015/11/20 11:33:51 [error] 3578#0: *1 upstream timed out (110: Connection timed out) while
connecting to upstream, client: 127.0.0.1, server: , request: "GET /api/activities.json?
user_key=USER_KEY HTTP/1.1", subrequest: "/threescale_authrep", upstream:
"https://54.83.62.186:443/transactions/authrep.xml?
provider_key=YOUR_PROVIDER_KEY&service_id=SERVICE_ID&usage[hits]=1&user_key=U
SER_KEY&log%5Bcode%5D=", host: "localhost"
```

### 12.3.4. API ゲートウェイ > 3scale

API ゲートウェイが正常に動作していることを確認したら、次のステップは API ゲートウェイと 3scale 間の接続についてのトラブルシューティングです。

#### 12.3.4.1. API ゲートウェイでの 3scale へのアクセスの可否確認

API ゲートウェイに NGINX を使用している場合、ゲートウェイが 3scale と通信できないときは、以下のメッセージが nginx エラーログに表示されます。

```
2015/11/20 11:33:51 [error] 3578#0: *1 upstream timed out (110: Connection timed out) while
connecting to upstream, client: 127.0.0.1, server: , request: "GET /api/activities.json?
user_key=USER_KEY HTTP/1.1", subrequest: "/threescale_authrep", upstream:
"https://54.83.62.186:443/transactions/authrep.xml?
provider_key=YOUR_PROVIDER_KEY&service_id=SERVICE_ID&usage[hits]=1&user_key=USER_KE
Y&log%5Bcode%5D=", host: "localhost"
```

ここでは、upstream の値に注意してください。この IP は、3scale プロダクトが解決する IP の 1 つに対応します。これは、3scale へのアクセスに問題があることを意味します。逆引き DNS ルックアップを実行して、**nslookup** を呼び出すことで、IP のドメインを確認することができます。

たとえば、API ゲートウェイが 3scale にアクセスできないからといって、3scale がダウンしているとは限りません。この問題の最も典型的な理由の 1 つは、API ゲートウェイが 3scale に接続することを妨げるファイアウォールルールです。

ゲートウェイと 3scale の間に、接続のタイムアウトを引き起こすネットワークの問題が存在する可能性があります。この場合、[一般的な接続の問題のトラブルシューティング](#) に関する手順を実施して、問題がどこにあるのかを特定する必要があります。

ネットワークの問題を除外するため、traceroute または MTR を使用して、ルーティングおよびパケット送信を確認します。3scale と API ゲートウェイに接続できるマシンから同じコマンドを実行し、出力を比較することもできます。

さらに、API ゲートウェイと 3scale の間で送信されているトラフィックを確認するには、一時的に 3scale プロダクトの HTTP エンドポイント (**su1.3scale.net**) を使用するように切り替えている限りは、tcpdump を使用できます。

#### 12.3.4.2. API ゲートウェイが 3scale のアドレスを正しく解決していることの確認

nginx.conf にリゾルバーディレクティブが追加されていることを確認します。

nginx.conf の設定例を以下に示します。

```
http {
    lua_shared_dict api_keys 10m;
    server_names_hash_bucket_size 128;
    lua_package_path "::$prefix/?.lua;";
    init_by_lua 'math.randomseed(ngx.time()) ; cJSON = require("cjson");

    resolver 8.8.8.8 8.8.4.4;
```

Google DNS (8.8.8.8 および 1377 8.8.4.4) は希望する DNS と置き換え可能です。

API ゲートウェイから DNS 解決を確認するには、以下のように指定したリゾルバー IP で nslookup を呼び出します。

```
nslookup su1.3scale.net 8.8.8.8
;; connection timed out; no servers could be reached
```

上記の例は、Google DNS に到達できない場合に返されるレスポンスを示しています。この場合、リゾルバー IP を更新する必要があります。nginx の error.log に、以下のアラートが表示される場合もあります。

```
2016/05/09 14:15:15 [alert] 9391#0: send() failed (1: Operation not permitted) while resolving,
resolver: 8.8.8.8:53
```

最後に、**dig any su1.3scale.net** を実行して、現在 3scale Service Management API について動作中の IP アドレスを確認します。これは、3scale によって使用される可能性のある IP アドレスの範囲全体ではないことに注意してください。容量の理由から、一部の IP アドレスがスワップインまたはスワップアウトされる場合があります。さらに、3scale サービスのドメイン名を今後追加することもできます。このため、該当する場合は、インテグレーション中に指定された特定のアドレスに対して必ずテストを行う必要があります。

### 12.3.4.3. API ゲートウェイが 3scale を正しく呼び出していることの確認

API ゲートウェイが 3scale に送信しているリクエストを確認する場合は、トラブルシューティング用途に限り、**nginx.conf** の 3scale authrep の場所 (API キーおよび App\_id 認証モードの場合は **/threescale\_authrep**) に、以下のスニペットを追加することができます。

```
body_filter_by_lua_block{
    if ngx.req.get_headers()["X-3scale-debug"] == ngx.var.provider_key then
        local resp = ""
        ngx.ctx.buffered = (ngx.ctx.buffered or "") .. string.sub(ngx.arg[1], 1, 1000)
        if ngx.arg[2] then
            resp = ngx.ctx.buffered
        end

        ngx.log(0, ngx.req.raw_header())
        ngx.log(0, resp)
    end
}
```

**X-3scale-debug header** が送信されると (例: **curl -v -H 'X-3scale-debug: YOUR\_PROVIDER\_KEY' -X GET "https://726e3b99.ngrok.com/api/contacts.json?access\_token=7c6f24f5"**)、このスニペットにより以下の追加ロギングが nginx error.log に追加されます。

これにより、以下のログエントリが生成されます。

```
2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:7: GET /api/contacts.json?
access_token=7c6f24f5 HTTP/1.1
Host: 726e3b99.ngrok.io
User-Agent: curl/7.43.0
Accept: */*
X-Forwarded-Proto: https
X-Forwarded-For: 2.139.235.79

while sending to client, client: 127.0.0.1, server: pili-virtualbox, request: "GET /api/contacts.json?
access_token=7c6f24f5 HTTP/1.1", subrequest: "/threescale_authrep", upstream:
"https://54.83.62.94:443/transactions/oauth_authrep.xml?
provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f24f5", host:
"726e3b99.ngrok.io"
2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:8: <?xml version="1.0" encoding="UTF-
8"?><error code="access_token_invalid">access_token "7c6f24f5" is invalid: expired or never
defined</error> while sending to client, client: 127.0.0.1, server: pili-virtualbox, request: "GET
/api/contacts.json?access_token=7c6f24f5 HTTP/1.1", subrequest: "/threescale_authrep", upstream:
"https://54.83.62.94:443/transactions/oauth_authrep.xml?
provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f24f5", host:
"726e3b99.ngrok.io"
```

最初のエントリ (**2016/05/05 14:24:33 [] 7238#0: \*57 [lua] body\_filter\_by\_lua:7:**) は、3scale に送信されたリクエストヘッダーを出力します。この例では、Host、User-Agent、Accept、X-Forwarded-Proto、および X-Forwarded-For です。

2 番目のエントリ (**2016/05/05 14:24:33 [] 7238#0: \*57 [lua] body\_filter\_by\_lua:8:**) は、3scale からのレスポンスを出力します。この例では、**<error code="access\_token\_invalid">access\_token "7c6f24f5" is invalid: expired or never defined</error>** となります。

両方がオリジナルのリクエスト (**GET /api/contacts.json?access\_token=7c6f24f5**) とサブリクエストの位置 (**/threescale\_authrep**)、ならびにアップストリームリクエスト (**upstream: "https://54.83.62.94:443/transactions/threescale\_authrep.xml?provider\_key=REDACTED&service\_id=REDACTED&usage[hits]=1&access\_token=7c6f24f5"**) を出力します。この最後の値で、どの 3scale IP が解決されているかと、3scale に行った実際のリクエストも確認できます。

## 12.3.5. 3scale

### 12.3.5.1. 3scale がエラーを返しているかの確認

3scale は利用可能だが、呼び出しが API に通ることを妨げるエラーを API ゲートウェイに返している可能性もあります。承認呼び出しを 3scale で直接実行して、レスポンスを確認します。エラーが発生した場合は、3scale のエラーコードセクションで何が問題かを確認します。

### 12.3.5.2. 3scale デバッグヘッダーの使用

たとえば、以下のように **X-3scale-debug** ヘッダーを設定して API を呼び出すことで、3scale デバッグヘッダーを有効にすることもできます。

```
curl -v -X GET "https://api.example.com/endpoint?user_key" X-3scale-debug:
YOUR_SERVICE_TOKEN
```

これにより、API レスポンスで以下のヘッダーが返されます。

```
X-3scale-matched-rules: /, /api/contacts.json
< X-3scale-credentials: access_token=TOKEN_VALUE
< X-3scale-usage: usage[hits]=2
< X-3scale-hostname: HOSTNAME_VALUE
```

### 12.3.5.3. インテグレーションエラーの確認

管理ポータルでインテグレーションエラーを確認し、3scale へのトラフィックに関する問題がないか確認することもできます。 [https://YOUR\\_DOMAIN-admin.3scale.net/apiconfig/errors](https://YOUR_DOMAIN-admin.3scale.net/apiconfig/errors) を参照してください。

インテグレーションエラーの理由の1つは、サーバブロックでは無効な **underscores\_in\_headers** ディレクティブによりヘッダーでクレデンシャルを送信していることです。

### 12.3.6. クライアント API ゲートウェイ

#### 12.3.6.1. 一般のインターネットから API ゲートウェイにアクセスできるか調べる

ブラウザをゲートウェイサーバーの IP アドレス (またはドメイン名) に転送するよう試みます。これに失敗する場合、該当するポートのファイアウォールが開いていることを確認してください。

#### 12.3.6.2. クライアントから API ゲートウェイにアクセスできるかの確認

可能な場合は、前述の方法 (telnet、curl など) のいずれかを使用して、クライアントから API ゲートウェイへの接続を試みます。接続に失敗する場合、クライアントと API ゲートウェイ間のネットワークに問題が発生しています。

そうでない場合は、API への呼び出しを行うクライアントのトラブルシューティングに進む必要があります。

### 12.3.7. クライアント

#### 12.3.7.1. 別のクライアントを使用した同じ呼び出しのテスト

リクエストが想定される結果を返さない場合は、別の HTTP クライアントでテストします。たとえば、Java HTTP クライアントで API を呼び出している時に何らかの問題が生じる場合、cURL を使用して結果を照合します。

クライアントとゲートウェイ間のプロキシ経由で API を呼び出し、クライアントが送信している正確なパラメーターとヘッダーを取得することもできます。

#### 12.3.7.2. クライアントから送信されたトラフィックの確認

Wireshark などのツールを使用して、クライアントが送信しているリクエストを調べます。これにより、クライアントが API を呼び出しているかどうか、およびリクエストの詳細を確認することができます。

## 12.4. ACTIVEDOCS の問題

コマンドラインから API を呼び出す場合には機能するが、ActiveDocs を経由する場合には失敗することがあります。



ActiveDocs 呼び出しを機能させるために、これらの呼び出しを 3scale 側のプロキシ経由で送信します。このプロキシが追加する特定のヘッダーが API にとって想定外だった場合に、問題を引き起こす可能性があります。これを確認するには、以下の手順を試みます。

### 12.4.1. petstore.swagger.io の使用

Swagger では、petstore.swagger.io にホスト型の swagger-ui が用意されています。これを使用して、最新バージョンの swagger-ui により Swagger 仕様と API をテストすることができます。swagger-ui と ActiveDocs の両方が同じように失敗する場合、ActiveDocs や ActiveDocs プロキシの問題は除外して、ご自分の仕様のトラブルシューティングに集中できます。あるいは、swagger-ui GitHub リポジトリで、現在の swagger-ui のバージョンに関する既知の問題を確認できます。

### 12.4.2. ファイアウォールが ActiveDocs プロキシからの接続を許可していることの確認

API を使用するクライアントの IP アドレスをホワイトリスト化しないよう推奨しています。ActiveDocs プロキシは、高可用性を実現するためにフローティング IP アドレスを使用していますが、現在これらの IP の変更を通知する仕組みはありません。

### 12.4.3. 無効なクレデンシャルを使用した API の呼び出し

ActiveDocs プロキシが正しく機能しているかどうかを確認する方法の1つは、無効なクレデンシャルを使用して API を呼び出すことです。これにより、ActiveDocs プロキシと API ゲートウェイの両方について、問題の有無を確認することができます。

API 呼び出しから 403 コード (または不正なクレデンシャルに対してゲートウェイで設定しているコード) が返される場合、呼び出しはゲートウェイに到達しているので、問題は API にあります。

### 12.4.4. 呼び出しの比較

ActiveDocs から行った呼び出しと ActiveDocs 外からの呼び出し間でヘッダーおよびパラメーターの相違点を特定するには、オンプレミスの APItools や Runscope などのサービスを介して呼び出しを行います。これにより、API に送信する前に HTTP 呼び出しを検査し、比較することができます。この操作により、問題を引き起こす可能性のあるリクエスト内のヘッダーやパラメーターを特定することができます。

## 12.5. NGINX でのロギング

これについての包括的なガイドは、[NGINX のロギングとモニタリング](#)に関するドキュメントを参照してください。

### 12.5.1. デバッグログの有効化

デバッグログの有効化の詳細については、[NGINX のデバッグログに関するドキュメント](#)を参照してください。

## 12.6. 3SCALE のエラーコード

3scale Service Management API エンドポイントによって返されるエラーコードを確認するには、以下の手順に従って [3scale API Documentation](#) のページを参照します。

1. 管理ポータルの上隅にある疑問符 (?) アイコンをクリックします。

## 2. 3scale API Docs を選択します。

以下は、3scale によって返される HTTP レスポンスコードと、そのコードが返される条件のリストです。

- 400: 不正なリクエスト。原因は以下のとおりです。
  - エンコーディングが無効である。
  - 負荷が大きすぎる。
  - コンテンツタイプが無効 (POST 呼び出しの場合) である。**Content-Type** ヘッダーの有効な値は、**application/x-www-form-urlencoded**、**multipart/form-data**、または空のヘッダーです。
- 403:
  - クレデンシャルが有効ではない。
  - 3scale に GET リクエスト用のボディーデータを送信している
- 404: アプリケーションやメトリックなど、存在しないエンティティが参照されている
- 409:
  - 使用制限の超過。
  - アプリケーションがアクティブではない。
  - アプリケーションキーが無効、または提供されない (**app\_id/app\_key** 認証メソッドの場合)。
  - 参照元が許可されていない、または提供されない (参照元フィルターが有効で必要な場合)
- 422: 必要なパラメーターが提供されない

これらのエラーレスポンスのほとんどには、マシンリーダブルなエラーカテゴリと人が判読できる説明が含まれる XML ボディーも含まれています。

標準の API ゲートウェイ設定を使用する場合、3scale から 200 以外のコードが返されると、クライアントには以下のどちらかのコードと共にレスポンスが返されます。

- 403
- 404