



Red Hat 3scale API Management 2.7

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

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 © 2023 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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

本書では、Red Hat 3scale API Management 2.7 のデプロイメント操作について説明します。

目次

第1章 3SCALE の操作とスケーリングのガイド	4
1.1. APICAST の再デプロイ	4
1.2. オンプレミス型 3SCALE のスケールアップ	5
1.3. 操作のトラブルシューティング	7
第2章 3SCALE TOOLBOX の使用	11
2.1. TOOLBOX のインストール	11
2.2. サポートされる TOOLBOX コマンドの使用	12
2.3. サービスのインポート	12
2.4. サービスのコピー	13
2.5. サービス設定のみのコピー	14
2.6. OPENAPI 定義のインポート	14
2.7. リモートアクセスクレデンシャルの管理	15
2.8. アプリケーションプラン	16
2.9. メトリック	22
2.10. メソッド	24
2.11. サービスの作成	26
2.12. ACTIVEDOCS	29
2.13. プロキシ設定	31
2.14. ポリシーレジストリー (カスタムポリシー) のコピー	32
2.15. アプリケーション	33
2.16. SSL の問題のトラブルシューティング	36
第3章 3SCALE TOOLBOX による API ライフサイクルの自動化	37
3.1. API ライフサイクルステージの概要	37
3.2. サンプル JENKINS CI/CD パイプラインのデプロイ	39
3.3. 3SCALE JENKINS 共有ライブラリーを使用したパイプラインの作成	47
3.4. JENKINSFILE を使用したパイプラインの作成	49
第4章 3SCALE での API 環境のマッピング	54
4.1. 環境ごとのプロダクト	54
4.2. オンプレミス型 3SCALE インスタンス	55
4.3. 3SCALE の混合アプローチ	56
4.4. 3SCALE と APICAST ゲートウェイの組み合わせ	56
第5章 機能: OPERATOR を使用した 3SCALE のサービスおよび設定のプロビジョニング	58
5.1. 前提条件	58
5.2. CAPABILITIES に関連するカスタムリソースのデプロイ	58
5.3. オプションテナントカスタムリソースのデプロイ	63
5.4. 作成したカスタムリソースの削除	64
第6章 3SCALE のバックアップと復元	65
6.1. 前提条件	65
6.2. 永続ボリューム	65
6.3. 留意事項	65
6.4. データセットの使用	66
6.5. バックアップの手順	67
6.6. データベースの復元手順	68
第7章 トラブルシューティング	79
7.1. 典型的な問題	79
7.2. トラブルシューティングの基本	83
7.3. トラブルシューティングのチェックリスト	87

7.4. その他の問題	91
7.5. 等価な ZYNC ルートの作成	92
7.6. 付録	93

第1章 3SCALE の操作とスケーリングのガイド

本章では、Red Hat 3scale API Management 2.7 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



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。

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」](#)

関連情報

- [OpenShift のドキュメント](#)

1.1. 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 クラスター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用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。

1.2. 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。

1.2.1. ストレージのスケールアップ

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、**oc edit pv** コマンドを使用して永続ボリューム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。

1.2.1.1. 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 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。

1.2.1.2. 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。

1.2.2. パフォーマンスのスケールアップ

パフォーマンスのスケールアップは、Pod の合計数に応じて行われます。ハードウェアリソースが多いほど、デプロイする Pod 数が増えます。

以下のコマンドを使用して、Pod の数によりパフォーマンスのスケールアップを行います。

```
oc scale dc dc-name --replicas=X
```

1.2.3. 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

1.2.4. OCP コマンドラインインターフェイスを使用したスケーリング

OpenShift Container Platform (OCP) コマンドラインインターフェイス (CLI) を使用して、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。

- 以下のコマンドを使用して、実稼働環境用 APIcast のデプロイメント設定をスケールアップします。

```
oc scale dc apicast-production --replicas=X
```

- 以下のコマンドを使用して、バックエンドリスナーのデプロイメント設定をスケールアップします。

```
oc scale dc backend-listener --replicas=Y
```

- 以下のコマンドを使用して、バックエンドワーカーのデプロイメント設定をスケールアップします。

```
oc scale dc backend-worker --replicas=Z
```

1.2.5. 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 クラスター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ー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。

1.2.6. 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。

1.3. 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。

1.3.1. 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 2.7 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。

1.3.2. 監査ロギングの有効化

3scale は、**features.xml** 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.xml** に依存する OpenShift Pod は、**system-app** と **system-sidekiq** です。

前提条件

- OpenShift でのクラスター管理者アクセス権限がある。

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
```

1.3.3. EFK ロギングの設定

stdout への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、EFK 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。

OpenShift での EFK ロギングの設定方法については、以下のドキュメントを参照してください。

- [OCP 3.11 への EFK のデプロイ](#)
- [OCP 4.1 への EFK のデプロイ](#)

1.3.4. 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
```

1.3.5. 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>
```

1.3.6. 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パ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。

alerts テーブルは例外で、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。

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

本セクションで説明しない他のテーブルについては、データベース管理者は、システムで自動的にパージされないテーブルを手動でクリーンアップする必要があります。

第2章 3SCALE TOOLBOX の使用

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。



重要

3scale toolbox は、すべての API プロダクト (APIaaS) 機能をサポートする訳ではありません。詳細は、3scale のリリースノートの [既知の問題](#) を参照してください。

2.1. 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。

2.1.1. toolbox コンテナイメージのインストール

前提条件

- [Red Hat Container Catalog の 3scale toolbox イメージ](#) を参照してください。
- Red Hat レジストリーサービスアカウントが必要です。
- このトピックの例では、Docker がインストールされ、デーモンが動作していることを前提としています。

手順

1. Red Hat コンテナレジストリーにログインします。

```
$ docker 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。

```
$ docker pull registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7
```

3. インストールを確認します。

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale help
```

関連情報

- OpenShift、Podman、または Docker での toolbox イメージインストールの詳細については、[Red Hat Container Catalog のイメージ取得手順](#) 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、**kubect**l 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。

2.1.2. サポートされないバージョンの toolbox のインストール

手順

- Fedora Linux、Ubuntu Linux、Windows、または macOS に、サポートされないバージョンの toolbox をインストールすることができます。そのためには、最新の [.rpm](#)、[.deb](#)、[.msi](#)、または [.pkg](#) ファイルを [GitHub](#) からダウンロードしてインストールします。

2.2. サポートされる TOOLBOX コマンドの使用

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 コマンドが非推奨になり、**copy** コマンドに置き換えられています。Red Hat は、非推奨になった機能の使用をお勧めしません。

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
copy	copy super command
help	show help
import	import super command
method	method super command
metric	metric super command
policy-registry	policy-registry super command
proxy-config	proxy-config super command
remote	remotes super command
service	services super command
update	[DEPRECTATED] update 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

2.3. 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 (CSV ファイルにこれらのヘッダーを含める必要もあります)。

```
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

```
$ docker run -v $PWD/examples/import_example.csv:/tmp/import_example.csv
registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale import csv --
destination=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
file=/tmp/import_example.csv
```

この例では、Docker 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** フォルダーにあることを前提としています。

2.4. 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

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 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。

2.5. 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、メトリクス、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し更新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 コマンドが非推奨になり、**copy** コマンドに置き換えられています。非推奨のコマンドの使用はサポートされていません。

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更新が行われます。

例

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale update [opts] service --
source=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
destination=https://{DEST_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} {SERVICE_ID}
{DEST_ID}
```

2.6. 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アクセスクレデンシャルからの定義を使用します。そのサービス名がすでに存在する場合は、それが更新されます。また、そのサービス名が存在しない場合は、作成されます。

インポートのデフォルトのサービス名は、OpenAPI 定義の **info.title** から取得されます。 **--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。そのサービス名がすでに存在する場合は、それが更新されます。また、そのサービス名が存在しない場合は、作成されます。

以下のルールはすべてのインポートに適用されます。

- 定義は OpenAPI 2.0 として検証される。
- 3scale 製品のすべてのマッピングルールが削除されます。
- 置き換えるためには、パターンの完全一致の使用により、すべてのメソッドの名前が 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。
- 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。
- 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。



注記

swagger 仕様にセキュリティ要件がない場合、サービスは Open API とみなされます。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。

例

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale import openapi [opts] --
destination=https://{DEST_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME}
```

2.6.1. オプションのフラグ

-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>

ターゲットシステム名

2.7. 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、設定ファイルでリモートの Web アドレス (URL) と共にこれらのインスタンスへのアクセスに使用する認証を定義します。3scale toolbox コマンドでは、これを短縮名で参照します。

設定ファイルのデフォルトの場所は **\$HOME/.3scalerc.yaml** ですが、**THREESCALE_CLI_CONFIG** 環境変数、または **--config-file <config_file>** 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。

access_token または **provider_key** のいずれかを使用して、リモート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>**

2.7.1. リモートアクセスクレデンシャルの一覧表示

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

既存のリモートのリスト (名前、URL、 および認証キー) を表示します。

例

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale remote list
instance_a https://example_a.net 123456789
instance_b https://example_b.net 987654321
```

2.7.2. リモートアクセスクレデンシャルの追加

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

短縮名 **<name>** のリモートを **<url>** に追加します。

例

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale remote add instance_a
https://123456789@example_a.net
```

2.7.3. リモートアクセスクレデンシャルの削除

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

短縮名 **<name>** のリモートを削除します。

例

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale remote remove
instance_a
```

2.7.4. リモートアクセスクレデンシャルの名前変更

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

リモートの短縮名を **<old_name>** から **<new_name>** に変更します。

例

```
$ docker run registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale remote rename
instance_a instance_b
```

2.8. アプリケーションプラン

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、一覧表示、削除、表示、またはエクスポート/インポートを行います。

2.8.1. 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す。
- **--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
-d --default                  This will make the default application plan
--disabled                    This will disable all methods and metrics in
                              the application plan
--end-user-required=<value>    End user required: true or false
-p --published                This will publish the application plan
--setup-fee=<value>            Set-up fee
-t --system-name=<value>       This will set application plan system name
--trial-period-days=<value>    The trial period in days
```

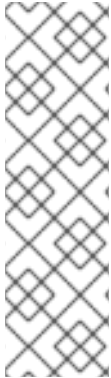
Options for application-plan

```
-c --config-file=<value>      3scale toolbox configuration file
                              (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                   This will print the version of this command
--verbose                     Verbose mode
```

2.8.2. 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                      This will make the default application plan
--disabled                     This will disable all methods and metrics in
                              the application plan
--enabled                      This will enable the application plan
--end-user-required=<value>    End user required: true or false
--hide                         This will hide the application plan
-n --name=<value>              This will set the plan name
-p --publish                   This will 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
                              (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                   This will print the version of this command
--verbose                      Verbose mode
```

2.8.3. アプリケーションプランの一覧表示

以下のコマンドにより、アプリケーションプランが一覧表示されます。

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

アプリケーションプランの一覧表示時に、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
                          (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              This will print the version of this command
--verbose                Verbose mode
```

2.8.4. 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 for application-plan

```
-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              This will print the version of this command
--verbose                Verbose mode
```

2.8.5. 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
                          (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              This will print the version of this command
--verbose                Verbose mode
```

2.8.6. 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クス/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。

2.8.6.1. 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

```
$ docker run -u root -v $PWD:/tmp registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.7 3scale application-plan export --file=/tmp/plan.yaml remote_name service_name plan_name
```

この例では、Docker 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** フォルダーに出力します。



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
(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

2.8.6.2. 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

```
$ docker run -v $PWD/plan.yaml:/tmp/plan.yaml registry.redhat.io/3scale-amp2/toolbox-
rhel7:3scale2.7 3scale application-plan import --file=/tmp/plan.yaml remote_name service_name
```

この例では、Docker 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。

2.8.6.3. 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クス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、メトリクス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 (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

2.9. メトリック

3scale toolbox を使用して、デベロッパーポータルでメトリックスの作成、更新、一覧表示、および削除を行います。

2.9.1. メトリックスの作成

メトリックスを作成するには、以下の手順に従います。

- メトリックス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ス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** メトリックスを作成します。
 - 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、メトリックスが作成されます。

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

メトリックスの作成時に、以下のオプションを使用します。

Options

```
--description=<value>  This will set a metric description
--disabled             This will disable this metric in all application plans
-t --system-name=<value> This will set the application plan system name
--unit=<value>         Metric unit: default hit
```

Option for metric

```
-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              This will print the version of this command
--verbose                 Verbose mode
```

2.9.2. メトリックスの作成または更新

メトリックス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スを更新するには、以下の手順に従います。

- 同じ名前のメトリックス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** メトリクスを更新します。
- **--enabled** フラグを使用して、**有効な** メトリクスに更新します。



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** 位置引数はメトリクス参照で、メトリクスの **id** またはメトリクスの **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、メトリクスが更新されます。

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

メトリクスの更新時に、以下のオプションを使用します。

Options

```
--description=<value>  This will set a metric description
--disabled             This will disable this metric in all application plans
--enabled              This will enable this metric in all application plans
-n --name=<value>      This will set the metric name
--unit=<value>         Metric unit: default hit
```

Options for metric

```
-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              This will print the version of this command
--verbose                 Verbose mode
```

2.9.3. メトリクスの一覧表示

以下のコマンドにより、メトリクスが一覧表示されます。

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

メトリクスの一覧表示時に、以下のオプションを使用します。

Options for metric

```
-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              This will print the version of this command
--verbose                 Verbose mode
```

2.9.4. メトリクスの削除

以下のコマンドにより、メトリクスが削除されます。

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

メトリクスの削除時に、以下のオプションを使用します。

Options for metric

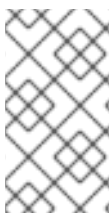
```
-c --config-file=<value>    3scale toolbox configuration file
                             (default: $HOME/.3scalerc.yaml)
-h --help                  show help for this command
-k --insecure              Proceed and operate even for server
                           connections otherwise considered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

2.10. メソッド

3scale toolbox を使用して、デベロッパーポータルでのメソッドの作成、適用、一覧表示、および削除を行います。

2.10.1. メソッドの作成

- メソッド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。
- 同じ名前のメソッド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** メソッドを作成します。
 - 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。

Option

```
--description=<value>    This will set a method description
--disabled                This will disable this method in all
                           application plans
-t --system-name=<value> This will set the method system name
```

Options for method

```
-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                 This will print the version of this command
--verbose                    Verbose mode
```

2.10.2. 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、コマンドは失敗します。
- **--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>    This will set a method description
--disabled                This will disable this method in all
                           application plans
--enabled                 This will enable this method in all
                           application plans
-n --name=<value>         This will set the method name
```

Options for method

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

2.10.3. メソッドの一覧表示

以下のコマンドにより、メソッドが一覧表示されます。

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

メソッドの一覧表示時に、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
                             (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               This will print the version of this command
--verbose                 Verbose mode
```

2.10.4. 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
                             (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               This will print the version of this command
--verbose                 Verbose mode
```

2.11. 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、一覧表示、表示、または削除を行います。

2.11.1. 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
-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
                                (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

```

2.11.2. 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

```

Options for services

- 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

2.11.3. サービスの一覧表示

以下のコマンドにより、サービスが一覧表示されます。

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

サービスの一覧表示時に、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 (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

2.11.4. 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 for services

- 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

2.11.5. 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 (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

2.12. ACTIVEDOCS

3scale toolbox を使用して、デベロッパーポータルの ActiveDocs の作成、更新、一覧表示、または削除を行います。

2.12.1. 新しい ActiveDocs の作成

OpenAPI/Swagger 準拠の 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> <spec>
```

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
- p --published Specify it to publish the ActiveDoc on the Developer Portal. Otherwise it will be hidden
- s --system-name=<value> Specify the system-name of the ActiveDocs
- skip-swagger-validations Specify it to skip validation of the Swagger specification

Options for ActiveDocs

- 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

2. デベロッパーポータルに定義を [公開](#) します。

2.12.2. 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 it to hide the ActiveDocs on the Developer Portal
- i --service-id=<value> Specify the Service ID associated to the ActiveDocs
- openapi-spec=<value> Specify the swagger spec. Can be a file, an URL or '-' to read from stdin. This option is mandatory when applying the ActiveDoc for the first time
- p --publish Specify it to publish the ActiveDocs on the Developer Portal. Otherwise it will be hidden
- s --name=<value> Specify the name of the ActiveDocs
- skip-swagger-validations Specify it to skip validation of the Swagger specification

Options for ActiveDocs

- 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

2.12.3. ActiveDocs の一覧表示

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 がすべて一覧表示されます。

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

ActiveDocs の一覧表示時に、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
                             (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
```

2.12.4. 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
                             (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
```

2.13. プロキシ設定

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一覧表示、表示、およびプロモートを行います。

2.13.1. プロキシ設定の一覧表示

以下のコマンドにより、プロキシ設定が一覧表示されます。

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

プロキシ設定の一覧表示時に、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 (default:
                             /home/msoriano/.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

2.13.2. 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 for proxy-config
-c --config-file=<value>    3scale toolbox configuration file
                             (default: /home/msoriano/.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
```

2.13.3. 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 (default:
                             /home/msoriano/.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
```

2.14. ポリシーレジストリー (カスタムポリシー) 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 (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
```

2.15. アプリケーション

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一覧表示、作成、表示、適用、または削除を行います。

2.15.1. アプリケーションの一覧表示

以下のコマンドにより、アプリケーションが一覧表示されます。

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

アプリケーションの一覧表示時に、以下のオプションを使用します。

OPTIONS

```
--account=<value>          Filter by account
--plan=<value>              Filter by application plan.
                             Service option required.
--service=<value>          Filter by service
```

2.15.2. 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 to be created.
- description=<value> Application description
- redirect-url=<value> OpenID Connect redirect url
- user-key=<value> User Key (API Key) of the application to be created.

2.15.3. 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**

2.15.4. 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
--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.
```

2.15.5. 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

2.16. SSL の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。

2.16.1. 信頼済み証明書のインストール

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。

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。

```
$ docker 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-rhel7:3scale2.7 3scale service list https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME}
```

この例では、Docker 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。
- コンテナ使用の詳細については、[Red Hat Guide to Linux Containers](#) を参照してください。
- Docker 使用の詳細については、[Docker のドキュメント](#) を参照してください。

第3章 3SCALE TOOLBOX による API ライフサイクルの自動化

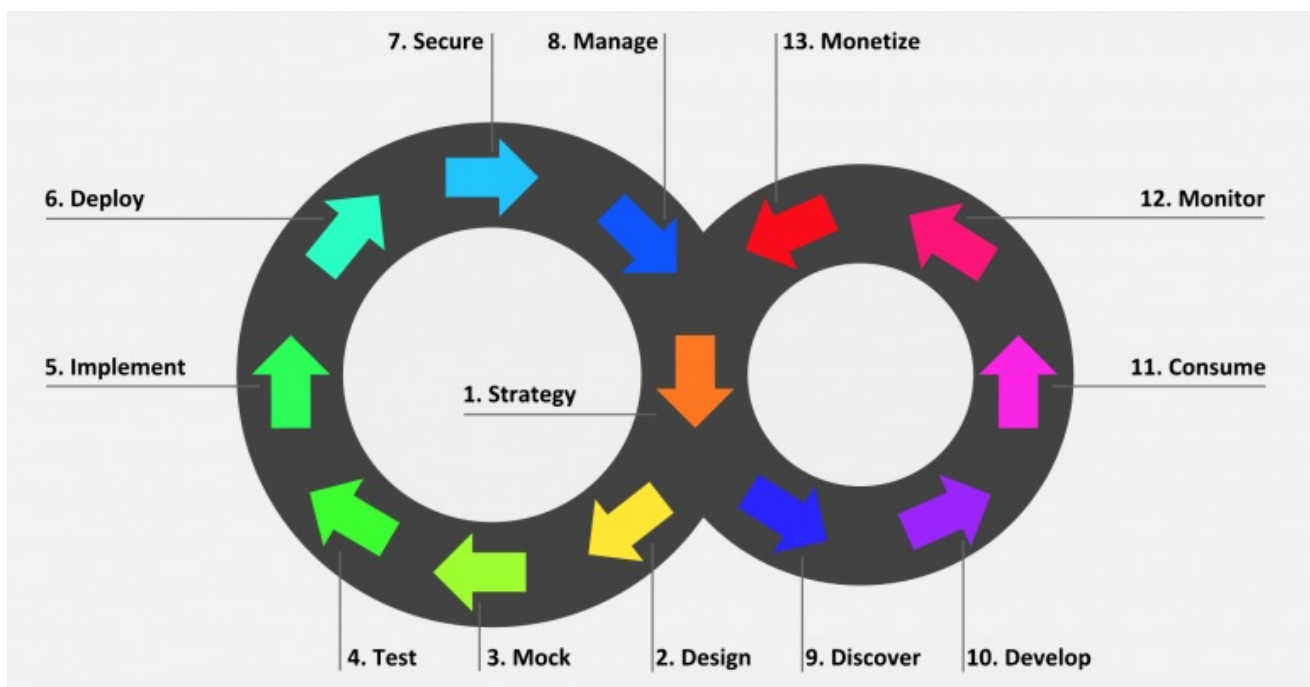
本トピックでは、Red Hat 3scale API Management での API ライフサイクルの概念について説明し、3scale toolbox コマンドにより API プロバイダーが Jenkins Continuous Integration/Continuous Deployment (CI/CD) パイプラインを使用してデプロイメントステージを自動化する方法を紹介します。ここでは、サンプルの Jenkins CI/CD パイプラインのデプロイ方法、3scale 共有ライブラリーを使用してカスタムの Jenkins パイプラインを作成する方法、およびカスタムパイプラインをゼロか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

3.1. 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ー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す。

3.1.1. 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3.1 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)。

3.1.2. 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3.2 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! の時間のメトリクスなど)。	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。

3.2. 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」](#)

3.2.1. 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3.3 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
SaaS - API key	ホスト型 3scale	API キー
Hybrid - open	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
Hybrid - OpenID Connect	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)
Multi-environment	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
Semantic versioning	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。

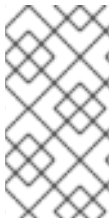


重要

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。

3.2.2. 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」](#) も併せて参照してください。

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については、[Red Hat 3scale API Management のサポート対象設定](#) のアート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の有効にしておく。

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. 後で使えるように、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. 後で使えるように、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
```

3.2.3. 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](#)」も併せて参照してください。

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 環境が必要です。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 クラスターが必要です。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については、[Red Hat 3scale API Management のサポート対象設定](#)のアーティクルを参照してください。
- [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. 後で使えるように、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. 後で使えるように、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 OPENSHIFT_ROUTER_SUFFIX=app.openshift.test # Replace me!
```

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

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



注記

OPENSHIFT_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
```

3.2.4. 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-sso-7-openshift-image/sso73-dev/templates/sso73-image-stream.json
```

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

```
oc -n openshift import-image redhat-sso73-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
```

3.2.5. 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. [2章 3scale toolbox の使用](#) で説明されているように、3scale ツールボックスをローカルに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"
```

3.2.6. 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
```

2. 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}')
```

3. 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
```

4. 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}')
```

3.2.7. 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

1. 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.$OPENSHIFT_ROUTER_SUFFIX

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

2. 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-rhel7:3scale2.7

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-rhel7:3scale2.7

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

```

3.2.8. 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](#)

3.2.9. 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については、[Red Hat 3scale API Management のサポート対象設定](#) のアート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。

API プロダクト

3scale toolbox は、すべての API プロダクト (APIaaS) 機能をサポートする訳ではありません。詳細は、3scale のリリースノートの [既知の問題](#) を参照してください。

3.3. 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 パイプラインのデプロイ」](#)
- 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。

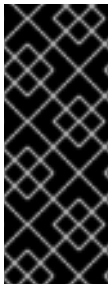
```
stage("Promote to production") {
    service.promoteToProduction()
}
```

関連情報

- [「Jenkinsfile を使用したパイプラインの作成」](#)
- [2章 3scale toolbox の使用](#)

3.4. 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 パイプラインのデプロイ」](#)

- 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ー関数を記述します。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-rhel7:3scale2.7",
              "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" ] ]
          ]
        ]
      ]
    ],
    "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](#)」のように **3scale-toolbox** という名前の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

- [「3scale Jenkins 共有ライブラリーを使用したパイプラインの作成」](#)
- [2章3scale toolbox の使用](#)

第4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」](#)

4.1. 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。

4.2. 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

4.2.1. 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스가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。

4.2.2. 環境ごとの 3scale テナントの分離

このアプローチでは単一の 3scale インスタンスを使用しますが、マルチテナンシー機能を利用して複数の API バックエンドをサポートします。

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 つ以上の製品を使用して、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 3 つのテナント (dev-tenant、qa-tenant、prod-tenant) があります。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。

4.3. 3SCALE の混合アプローチ

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

- 実稼働用の個別の 3scale インスタンス。
- 非本番環境用に開発および QA で個別のテナントを持つ個別の 3scale インスタンス。

4.4. 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。

- 各 3scale インストールには、ステージングおよび実稼働用に 2 つの組み込み APIcast ゲートウェイが付属しています。
- 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。

4.4.1. 組み込みの APIcast デフォルトゲートウェイ

組み込み APIcast ゲートウェイを使用する場合、[APIcast ゲートウェイを使用した 3scale](#) 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 バックエンドが自動的に処理されます。3scale マスター管理者がテナントを追加すると、実稼働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用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プされた各 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。

- Developer: <API_NAME>-dev-apicast.staging.<WILDCARD_DOMAIN>
- QA: <API_NAME>-qa-apicast.staging.<WILDCARD_DOMAIN>
- Production: <API_NAME>-prod-apicast.staging.<WILDCARD_DOMAIN>

4.4.2. 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 の輝度言う時に使用される環境変数 **THREESCALE_PORTAL_ENDPOINT** の値は、追加の APIcast ゲートウェイの設定方法により異なります。

API バックエンド環境ごとに個別の APIcast ゲートウェイを設定できます。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。

複数の API バックエンド環境で単一の APIcast ゲートウェイを使用できます。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5章 機能: OPERATOR を使用した 3SCALE のサービスおよび設定のプロビジョニング

このドキュメントでは、3scale operator の機能について説明します。これは、OpenShift Container Platform ユーザーインターフェイスを介して 3scale operator を使用して 3scale 製品および設定をプロビジョニングするものです。



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の機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

5.1. 前提条件

- [オンプレミス型 3scale 2.7 インスタンス](#)
- [3scale operator がインストールされている](#) 必要があります。
- OpenShift Container Platform 4
 - OpenShift クラスターの管理者権限を持つユーザーアカウント
 - **注記:** OCP 4 は、operator を使用した 3scale のデプロイメントのみをサポートしていません。
 - サポート対象設定の情報については、[Red Hat 3scale API Management Supported Configurations](#) のアートを参照してください。



警告

3scale operator を使用して 3scale で API 設定を更新する場合は、カスタムリソース定義 (CRD) が信頼できるソースです。管理ユーザーインターフェイスに変更が加えられると、それらは永続化されず、最終的に CRD の定義で上書きされます。

5.2. CAPABILITIES に関連する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、**API**、**メトリクス**、および **マッピングルール** を設定します。

5.2.1. API の作成

以下の手順により、**api: api01** というラベルの API を作成します。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **API** タブをクリックします。
3. **Create API** をクリックします。
4. サンプルのコンテンツを消去して以下の **YAML** 定義をエディターに追加し、続いて **Create** をクリックします。

```
apiVersion: capabilities.3scale.net/v1alpha1
kind: API
metadata:
  creationTimestamp: 2019-01-25T13:28:41Z
  generation: 1
  labels:
    environment: testing
  name: api01
spec:
  planSelector:
    matchLabels:
      api: api01
  description: api01
  integrationMethod:
    apicastHosted:
      apiTestGetRequest: /
    authenticationSettings:
      credentials:
        apiKey:
          authParameterName: user-key
          credentialsLocation: headers
      errors:
        authenticationFailed:
          contentType: text/plain; charset=us-ascii
          responseBody: Authentication failed
          responseCode: 403
        authenticationMissing:
          contentType: text/plain; charset=us-ascii
          responseBody: Authentication Missing
          responseCode: 403
      hostHeader: ""
      secretToken: Shared_secret_sent_from_proxy_to_API_backend_9603f637ca51ccfe
    mappingRulesSelector:
      matchLabels:
        api: api01
      privateBaseURL: https://echo-api.3scale.net:443
  metricSelector:
    matchLabels:
      api: api01
```



注記

すべてのセレクトター (metric、plan、mappingrules) で、特定のラベル **api: api01** が使用されます。複雑なシナリオに対応するために、さらにラベルを追加してセレクトターを設定することで、この設定を変更することができます。

5.2.2. プランの追加

以下の手順により、**api: api01** というラベルのプランを追加します。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **Plan** タブをクリックします。
3. **Create Plan** をクリックします。
4. サンプルのコンテンツを消去し、以下の **YAML** 定義をエディターに追加してから、**Create** をクリックします。

```
apiVersion: capabilities.3scale.net/v1alpha1
kind: Plan
metadata:
  labels:
    api: api01
  name: plan01
spec:
  approvalRequired: false
  default: true
  costs:
    costMonth: 0
    setupFee: 0
  limitSelector:
    matchLabels:
      api: api01
  trialPeriod: 0
```

5.2.3. メトリクスの追加

以下の手順により、**metric01** というメトリクスを追加します。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。

- **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **Metric** タブをクリックします。
 3. **Create Metric** をクリックします。
 4. サンプルのコンテンツを消去し、以下の **YAML** 定義をエディターに追加してから、**Create** をクリックします。

```
apiVersion: capabilities.3scale.net/v1alpha1
kind: Metric
metadata:
  labels:
    api: api01
    name: metric01
spec:
  description: metric01
  unit: hit
  incrementHits: false
```

5.2.4. 制限の設定

以下の手順により、メトリクスに1日あたり10 ヒットという制限を設定します。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **Limit** タブをクリックします。
3. **Create Limit** をクリックします。
4. サンプルのコンテンツを消去して以下の **YAML** 定義をエディターに追加し、続いて **Create** をクリックします。

```
apiVersion: capabilities.3scale.net/v1alpha1
kind: Limit
metadata:
  labels:
    api: api01
    name: plan01-metric01-day-10
spec:
  description: Limit for metric01 in plan01
  max_value: 10
  metricRef:
    name: metric01
  period: day
```

5.2.5. マッピングルールの追加

以下の手順により、**metric01** のカウントを増加させる **MappingRule** を追加します。

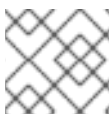
手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **MappingRule** タブをクリックします。
3. **Create Mapping Rule** をクリックします。
4. サンプルのコンテンツを消去し、以下の **YAML** 定義をエディターに追加してから、**Create** をクリックします。

```
apiVersion: capabilities.3scale.net/v1alpha1
kind: MappingRule
metadata:
  labels:
    api: api01
    name: metric01-get-path01
spec:
  increment: 1
  method: GET
  metricRef:
    name: metric01
  path: /path01
```

5.2.6. バインディングの作成

バインディングオブジェクトを使用してバインドするには、以下の手順に従います。



注記

テナントコントローラー で作成されたクレデンシャルを使用します。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **Binding** タブをクリックします。
3. **Create Binding** をクリックします。
4. サンプルのコンテンツを消去して以下の **YAML** 定義をエディターに追加し、続いて **Create** をクリックします。

```

apiVersion: capabilities.3scale.net/v1alpha1
kind: Binding
metadata:
  name: mytestingbinding
spec:
  credentialsRef:
    name: ecorp-tenant-secret
  APISelector:
    matchLabels:
      environment: testing

```

バインディングオブジェクトは、**ecorp-tenant-secret** を参照して、**environment: staging** のラベルが設定された API オブジェクトを作成します。

5. 新しい 3scale テナントに移動し、前のステップで実行したすべての設定が作成されていることを確認します。



注記

詳細は、参考のドキュメント [Capabilities CRD](#) を確認してください。

5.3. オプションテナントカスタムリソースのデプロイ

オプションで、**Tenant** カスタムリソースオブジェクトをデプロイするその他のテナントを作成できます。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **Tenant** タブをクリックします。
3. **Create Tenant** をクリックします。
4. サンプルのコンテンツを消去し、以下の **YAML** 定義をエディターに追加してから、**Create** をクリックします。

```

apiVersion: capabilities.3scale.net/v1alpha1
kind: Tenant
metadata:
  name: ecorp-tenant
spec:
  username: admin
  systemMasterUrl: https://master.<wildcardDomain>
  email: admin@ecorp.com
  organizationName: ECorp
  masterCredentialsRef:
    name: system-seed
  passwordCredentialsRef:

```

```
name: ecorp-admin-secret
tenantSecretRef:
  name: ecorp-tenant-secret
  namespace: operator-test
```

テナントの **provider_key** と **admin domain URL** はシークレットに格納されます。**tenantSecretRef** テナント仕様キーを使用して、シークレットの場所を指定できます。



注記

Tenant Custom Resource フィールドと設定可能な値については、[テナント CRD のリファレンス](#) を参照してください。

5.4. 作成したカスタムリソースの削除

以下の手順で、カスタムリソースを削除する方法を説明します。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. 削除するリソースのタブをクリックします。
 - a. すでにリソースが作成されていれば、リソースのリストが表示されます。
3. 名前をクリックして概要を表示します。
4. **Action > Delete** の順にクリックします。
5. **Delete** をクリックして削除を確認するか、**Cancel** をクリックして前の画面に戻ります。

あるいは、3scale operator、その関連付けられたロール、およびサービスアカウントを削除するには、以下の手順を実施します。

手順

1. メニュー構造は、使用している OpenShift のバージョンによって異なります。
 - OCP 4.1 の場合は、**Catalog > Installed Operators** の順にクリックします。
 - OCP 4.2 の場合は、**Operators > Installed Operators** の順にクリックします。
 - **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **Action > Delete Cluster Service Version** の順にクリックします。
3. **Delete** をクリックして削除を確認するか、**Cancel** をクリックして前の画面に戻ります。

第6章 3SCALE のバックアップと復元

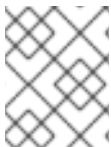
本セクションでは、Red Hat 3scale API Management インストール環境の管理者が以下の操作を行うのに必要な情報を提供します。

- 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。

6.1. 前提条件

- 3scale 2.7 インスタンス。3scale のインストール方法については、3scale のインストールの [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
- サポート対象設定の情報については、[Red Hat 3scale API Management Supported Configurations](#) のアートを参照してください。



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。

6.2. 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。

6.3. 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。

6.4. 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 (mysql-storage)
<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 (backend-redis-storage)
<code>system-redis</code>	Redis データベース (system-redis-storage)

6.4.1. 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。

6.4.2. system-storage の定義



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。

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)

6.4.3. 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 **バックエンド** により集約された API 使用量についての情報です。これは、**バックエンド** による流量制限の決定と、**システム** による解析情報の表示 (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 **システム** 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。

6.4.4. 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。

6.5. バックアップの手順

システムデータベースをバックアップしアーカイブするには、以下のコマンドを使用します。

6.5.1. 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
```

6.5.2. 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
```

6.5.3. 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
```

6.5.4. 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
```

6.5.5. 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
```

6.5.6. 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。

6.5.6.1. 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
```

6.5.6.2. ConfigMap

```
oc get configmaps system-environment -o json > system-environment.json
oc get configmaps apicast-environment -o json > apicast-environment.json
```

6.6. データベースの復元手順

以下のコマンドを使用して、システム障害が発生した後にシステムデータベースを復元できます。



重要

system-app のような Pod をスケールダウンしたり、ルートが無効にしたりして、レコードが作成されないようにします。

6.6.1. テンプレートベースのデプロイメントの復元

テンプレート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. デプロイ用テンプレートを作成する前に、ConfigMaps を復元します。

```
oc apply -f smtp.json
```

2. テンプレートパラメーターは、コピーされたシークレットおよび configmaps から読み込まれます。

```
oc new-app --file /opt/amp/templates/amp.yml \
  --param APP_LABEL=$(cat system-environment.json | jq -r '.metadata.labels.app') \
```



```

--param TENANT_NAME=$(cat system-seed.json | jq -r '.data.TENANT_NAME' | base64 -
d) \
--param SYSTEM_DATABASE_USER=$(cat system-database.json | jq -r
'.data.DB_USER' | base64 -d) \
--param SYSTEM_DATABASE_PASSWORD=$(cat system-database.json | jq -r
'.data.DB_PASSWORD' | base64 -d) \
--param SYSTEM_DATABASE=$(cat system-database.json | jq -r '.data.URL' | base64 -d
| cut -d '/' -f4) \
--param SYSTEM_DATABASE_ROOT_PASSWORD=$(cat system-database.json | jq -r
'.data.URL' | base64 -d | awk -F '[:@]' '{print $3}') \
--param WILDCARD_DOMAIN=$(cat system-environment.json | jq -r
'.data.THREESCALE_SUPERDOMAIN') \
--param SYSTEM_BACKEND_USERNAME=$(cat backend-internal-api.json | jq
'.data.username' -r | base64 -d) \
--param SYSTEM_BACKEND_PASSWORD=$(cat backend-internal-api.json | jq
'.data.password' -r | base64 -d) \
--param SYSTEM_BACKEND_SHARED_SECRET=$(cat system-events-hook.json | jq -r
'.data.PASSWORD' | base64 -d) \
--param SYSTEM_APP_SECRET_KEY_BASE=$(cat system-app.json | jq -r
'.data.SECRET_KEY_BASE' | base64 -d) \
--param ADMIN_PASSWORD=$(cat system-seed.json | jq -r '.data.ADMIN_PASSWORD'
| base64 -d) \
--param ADMIN_USERNAME=$(cat system-seed.json | jq -r '.data.ADMIN_USER' |
base64 -d) \
--param ADMIN_EMAIL=$(cat system-seed.json | jq -r '.data.ADMIN_EMAIL' | base64 -d) \
--param ADMIN_ACCESS_TOKEN=$(cat system-seed.json | jq -r
'.data.ADMIN_ACCESS_TOKEN' | base64 -d) \
--param MASTER_NAME=$(cat system-seed.json | jq -r '.data.MASTER_DOMAIN' |
base64 -d) \
--param MASTER_USER=$(cat system-seed.json | jq -r '.data.MASTER_USER' | base64 -
d) \
--param MASTER_PASSWORD=$(cat system-seed.json | jq -r
'.data.MASTER_PASSWORD' | base64 -d) \
--param MASTER_ACCESS_TOKEN=$(cat system-seed.json | jq -r
'.data.MASTER_ACCESS_TOKEN' | base64 -d) \
--param RECAPTCHA_PUBLIC_KEY=$(cat system-recaptcha.json | jq -r
'.data.PUBLIC_KEY' | base64 -d)" \
--param RECAPTCHA_PRIVATE_KEY=$(cat system-recaptcha.json | jq -r
'.data.PRIVATE_KEY' | base64 -d)" \
--param SYSTEM_REDIS_URL=$(cat system-redis.json | jq -r '.data.URL' | base64 -d) \
--param SYSTEM_MESSAGE_BUS_REDIS_URL=$(cat system-redis.json | jq -r
'.data.MESSAGE_BUS_URL' | base64 -d)" \
--param SYSTEM_REDIS_NAMESPACE=$(cat system-redis.json | jq -r
'.data.NAMESPACE' | base64 -d)" \
--param SYSTEM_MESSAGE_BUS_REDIS_NAMESPACE=$(cat system-redis.json | jq -r
'.data.MESSAGE_BUS_NAMESPACE' | base64 -d)" \
--param ZYNC_DATABASE_PASSWORD=$(cat zync.json | jq -r
'.data.ZYNC_DATABASE_PASSWORD' | base64 -d) \
--param ZYNC_SECRET_KEY_BASE=$(cat zync.json | jq -r '.data.SECRET_KEY_BASE'
| base64 -d) \
--param ZYNC_AUTHENTICATION_TOKEN=$(cat zync.json | jq -r
'.data.ZYNC_AUTHENTICATION_TOKEN' | base64 -d) \
--param APICAST_ACCESS_TOKEN=$(cat system-master-apicast.json | jq -r
'.data.ACCESS_TOKEN' | base64 -d) \
--param APICAST_MANAGEMENT_API=$(cat apicast-environment.json | jq -r
'.data.APICAST_MANAGEMENT_API') \

```

```
--param APICAST_OPENSSL_VERIFY=$(cat apicast-environment.json | jq -r
'.data.OPENSSL_VERIFY') \
--param APICAST_RESPONSE_CODES=$(cat apicast-environment.json | jq -r
'.data.APICAST_RESPONSE_CODES') \
--param APICAST_REGISTRY_URL=$(cat system-environment.json | jq -r
'.data.APICAST_REGISTRY_URL')
```

6.6.2. 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 カスタムリソースを使用して、[Operator](#) で [3scale](#) をデプロイします。

6.6.3. system-mysql の復元

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展開します。

```
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'
```

6.6.4. 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
```

6.6.5. zync-database の復元

zync-database を復元する手順は、3scale に適用したデプロイメントタイプによって異なります。

テンプレートベースのデプロイメント

1. Zync DeploymentConfig を 0 Pod にスケールダウンします。

```
oc scale dc zync --replicas=0
oc scale dc zync-queue --replicas=0
```

2. 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/
```

3. バックアップファイルを展開します。

```
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'
```

4. PostgreSQL DB 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 -f ${HOME}/zync-database-backup'
```

5. 以下のコマンドで **\${ZYNC_REPLICAS}** をレプリカ数に置き換えて、元のレプリカ数に復元します。

```
oc scale dc zync --replicas=${ZYNC_REPLICAS}
oc scale dc zync-queue --replicas=${ZYNC_REPLICAS}
```



OPERATOR ベースのデプロイメント

Operator を使用した 3scale のデプロイ (特に **APIManager カスタムリソースのデプロイ**) 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。

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展開します。

```
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. PostgreSQL DB 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 -f ${HOME}/zync-database-backup'
```

6. 元のレプリカ数に復元します。

```
oc patch APIManager ${DEPLOYMENT_NAME} --type merge -p '{"spec": {"zync": "${ZYNC_SPEC}}'
```

6.6.6. 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べき等です。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一覧です。

- 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。

6.6.7. バックエンド と システム 間の情報の一貫性確保

backend-redis の復元後、**System** からの設定情報と同期させ、**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。

6.6.7.1. 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。

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
```

6.6.7.2. 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。

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. **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
```

6.6.8. backend-worker の復元

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
```

6.6.9. system-app の復元

1. 最新バージョンの **system-app** に復元します。

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

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-app
```

6.6.10. system-sidekiq の復元

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
```

6.6.11. system-sphinx の復元

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
```

6.6.12. Zync が管理する OpenShift ルートの復元

1. 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。

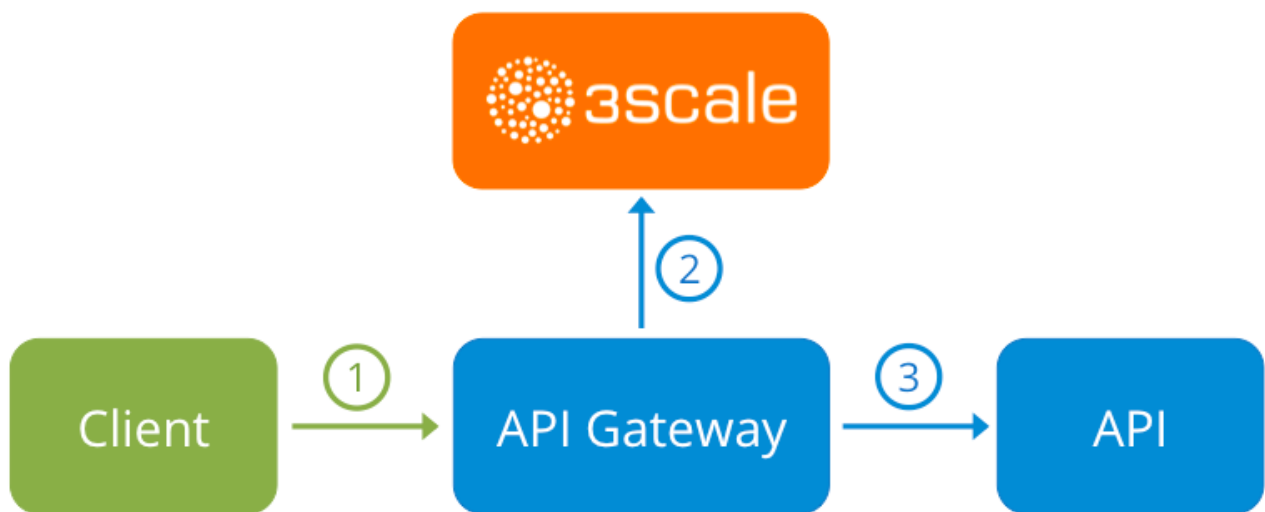
```
oc rsh $(oc get pods -l 'deploymentConfig=system-sidekiq' -o json | jq  
' .items[0].metadata.name' -r) bash -c 'bundle exec rake zync:resync:domains'
```

第7章 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ると、顧客は 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。

7.1. 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、または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。

7.1.1. 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。

7.1.1.1. 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。

- The accepted format is protocol://address(:port)
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。

7.1.1.2. 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。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。

7.1.2. 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。

7.1.2.1. 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あります。

- 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す。

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。

7.1.3. 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、より詳細な操作上のトラブルシューティングに進む必要があります (以下のトラブルシューティングのチェックリストセクションを参照)。問題のある箇所の特定を試みるため、API から始めてクライアントまで遡って作業します。

7.2. トラブルシューティングの基本

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。

7.2.1.1. 接続の可否を確かめる

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-i2t5pgt2cfdnbdhf2c6qqoartm-829217110.us-east-1.elb.amazonaws.com.
Escape character is '^]'.
Connection closed by foreign host.
```

- 失敗

```
telnet su1.3scale.net 443
Trying 174.129.235.69...
telnet: Unable to connect to remote host: Connection timed out
```

7.2.2. 2.問題の発生場所を特定する

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。

7.2.3. 3. DNS に問題がないか調べる

ホスト名の代わりに IP アドレスを使っ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

7.2.4. 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  ..l..<`x,\..."...
0010 - eb be 26 6c e6 7b 43 cc-ae 9b c0 27 6c b7 d9 13  ..&l.{C....'l...
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:ssl3 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。

7.3. トラブルシューティングのチェックリスト

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。

7.3.1. 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。

7.3.2. 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。

7.3.3. 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。

7.3.3.1. 1. API ゲートウェイが起動して稼働しているか調べる

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。

7.3.3.2. 2. ゲートウェイログでエラーの有無を確認する

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"
```

7.3.4. 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。

7.3.4.1. 1. API ゲートウェイが 3scale にアクセスできるか調べる

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、tracert 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。

7.3.4.2. 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か調べる

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。

7.3.4.3. 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ーが生成されます。

```
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"
```

最初のエントリー (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 番目のエントリー (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。

7.3.5. 3scale

7.3.5.1. 1. 3scale が利用可能か確かめる

[3scale のステータスページ](#) または Twitter で [@3scalestatus](#) を確認してください。

7.3.5.2. 2. 3scale がエラーを返しているか調べる

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。

7.3.5.3. 3. 3scale デバッグヘッダーを使用する

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
```



```
< X-3scale-matched-rules: /aos/internal/gate_status/v1/flights/histories
< X-3scale-credentials: user_key=b3ad95279b0600a2595165404fbae70c
< X-3scale-usage: usage%5Bhits%5D=1
< X-3scale-hostname: apicast-staging-3-nkhnn
< X-3scale-service-id: 2
< X-3scale-service-name: api
```

7.3.5.4. 4. インテグレーションエラーを確認する

3scale へのトラフィックに関する問題がないか確認するには、管理ポータルでインテグレーションエラー (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、サーバー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。

7.3.6. クライアント API ゲートウェイ

7.3.6.1. 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。

7.3.6.2. 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調べる

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。

7.3.7. クライアント

7.3.7.1. 1. 別のクライアントを使用して同じ呼び出しをテストする

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。

7.3.7.2. 2. クライアントから送信されたトラフィックを確認する

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。

7.4. その他の問題

7.4.1. 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。

7.4.1.1. petstore.swagger.io を使用する

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。

7.4.1.2. 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を確認する

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。

7.4.1.3. 無効なクレデンシャルを使用して API を呼び出す

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。

7.4.1.4. 呼び出しを比較する

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、API に送信する前に HTTP 呼び出しを検証および比較できる特定のサービス (オンプレミスの APItools、Runscope など) を介して呼び出しを実行することが有益な場合もあります。これにより、API プロバイダー側で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。

7.5. 等価な ZYNC ルートの作成

Zync による等価ルートの作成を行うには、OpenShift が Zync でルーティングするすべての 3scale API およびテナントを強制的に再同期させます。

```
oc exec -t $(oc get pods -l 'deploymentConfig=system-sidekiq' -o json | jq '.items[0].metadata.name' -r) -- bash -c "bundle exec rake zync:resync:domains"
```

システムへの通知についての情報が含まれる以下の出力が表示されます。

```
No valid API key has been set, notifications will not be sent
ActiveMerchant MODE set to 'production'
[Core] Using http://backend-listener:3000/internal/ as URL
OpenIdAuthentication.store is nil. Using in-memory store.
[EventBroker] notifying subscribers of Domains::ProviderDomainsChangedEvent 59a554f6-7b3f-
```



```

4246-9c36-24da988ca800
[EventBroker] notifying subscribers of ZyncEvent caa8e941-b734-4192-acb0-0b12cbaab9ca
Enqueued ZyncWorker#d92db46bdba7a299f3e88f14 with args: ["caa8e941-b734-4192-acb0-0b12cbaab9ca", {type=>"Provider", :id=>1, :parent_event_id=>"59a554f6-7b3f-4246-9c36-24da988ca800", :parent_event_type=>"Domains::ProviderDomainsChangedEvent", :tenant_id=>1}]
[EventBroker] notifying subscribers of Domains::ProviderDomainsChangedEvent 9010a199-2af1-4023-9b8d-297bd618096f
...

```

強制的に Zync による再評価を行った後に、すべての既存テナントおよびサービスに対して新規ルートが作成されます。サービスおよびテナントの数によっては、ルートの作成に数分かかる場合があります。

7.6. 付録

7.6.1. 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ーリング](#)に関するドキュメントを参照してください。

7.6.1.1. 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。

7.6.2. 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一覧です。

- **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タを送信している
- **404:** アプリケーションやメトリクスなど、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