Red Hat 3scale API Management 2.12

3scale のインストール

3scale API Management のインストールおよび設定

Last Updated: 2023-07-20

Red Hat 3scale API Management 2.12 3scale のインストール

3scale API Management のインストールおよび設定

法律上の通知

概要

本ガイドは、3scale API Management のインストールおよび設定に関する情報を提供します。

目次

はじめに

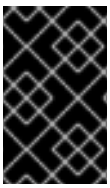
本ガイドは、3scale のインストールおよび設定に役立ちます。

多様性を受け入れるオープンソースの強化

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

第1章 3SCALE 用レジストリーサービスアカウント

3scale 2.12 と共に共有環境で **registry.redhat.io** からのコンテナーイメージを使用するには、個々のユーザーの **カスタマーポータル** のクレデンシャルではなく、**レジストリーサービスアカウント** を使用する必要があります。



重要

どちらのオプションもレジストリーの認証を使用するため、本章で概説している手順に従い、テンプレートまたは Operator を使用して OpenShift にデプロイすることが 3sclae のデプロイメントの要件です。

レジストリーサービスアカウントを作成および変更するには、以下のセクションに概略を示す手順を実施します。

- レジストリーサービスアカウントの作成
- コンテナーレジストリー認証の設定
- レジストリーサービスアカウントの変更

1.1. レジストリーサービスアカウントの作成

レジストリーサービスアカウントを作成するには、以下の手順に従います。

手順

1. Registry Service Accounts のページに移動し、ログインします。
2. **New Service Account** をクリックします。
3. **Create a New Registry Service Account**のページに表示されるフォームに入力します。
   a. **サービスアカウント** の名前を追加します。
   注記:フォームのフィールドの前に、決められた桁数のランダムに生成された数字の文字列が表示されます。
   b. **Description** を入力します。
   c. **Create** をクリックします。
4. **Registry Service Accounts**のページに戻ります。
5. 作成した **サービスアカウント** をクリックします。
6. 接頭辞の文字列を含めたユーザー名 (例: **12345678|username**) およびパスワードを書き留めます。このユーザー名およびパスワードは、**registry.redhat.io** へのログインに使用されます。

注記

Token Information のページには、認証トークンの使用方法を説明したタブがあります。たとえば、**Token Information** タブには、**12345678|username** フォーマットのユーザー名およびその下にパスワードの文字列が表示されます。

1.2. コンテナーレジストリー認証の設定

3scale 管理者は、3scale を OpenShift にデプロイする前に、**registry.redhat.io** との認証を設定します。

前提条件

- 管理者クレデンシャルのある Red Hat OpenShift Container Platform (OCP) アカウント。
- OpenShift **oc** クライアントツールがインストール済みである。詳細は、OpenShift CLI のドキュメント を参照してください。

手順

1. 管理者として OpenShift クラスターにログインします。

```
$ oc login -u system:admin
```

2. 3scale をデプロイするプロジェクトを開きます。

```
oc project your-openshift-project
```

3. Red Hat カスタマーポータルアカウントを使用して **docker-registry** シークレットを作成します。**threescale-registry-auth** は作成するシークレットに置き換えます。

```
$ oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username="customer_portal_username" \
  --docker-password="customer_portal_password" \
  --docker-email="email_address"
```

以下の出力が表示されるはずです。

```
secret/threescale-registry-auth created
```

4. シークレットをサービスアカウントにリンクして、シークレットをイメージをプルするために使用します。サービスアカウント名は、OpenShift Pod が使用する名前と一致する必要があります。以下は、**default** サービスアカウントを使用する例になります。

```
$ oc secrets link default threescale-registry-auth --for=pull
```

5. シークレットを **builder** サービスアカウントにリンクし、ビルドイメージをプッシュおよびプルするためにシークレットを使用します。

```
$ oc secrets link builder threescale-registry-auth
```

関連情報

コンテナーイメージに対する Red Hat の認証に関する詳細は、以下を参照してください。

- Red Hat コンテナーレジストリーの認証
- Red Hat registry service accounts

1.3. レジストリーサービスアカウントの変更

Registry Service Accountページからサービスアカウントを編集または削除することができます。そのためには、表中の各認証トークン右側のポップアップメニューを使用します。

警告

トークンを再生成したり **サービスアカウント** を削除したりすると、そのトークンを用いて認証して、**registry.redhat.io** からコンテンツを取得しているシステムに影響を及ぼします。

各機能の説明は以下のとおりです。

- **トークンの生成:** 許可されたユーザーは、**サービスアカウント** に関連付けられたパスワードをリセットすることができます。
  注記:サービスアカウント のユーザー名を変更することはできません。
- **説明の更新:** 許可されたユーザーは、**サービスアカウント** の説明を更新することができます。
- **アカウントの削除:** 許可されたユーザーは、**サービスアカウント** を削除することができます。

1.4. 関連情報

- Red Hat コンテナーレジストリーの認証
- Authentication enabled Red Hat registry

第2章 OPENSHIFT への 3SCALE のインストール

本セクションでは、OpenShift に Red Hat 3scale API Management 2.12 をデプロイする一連の手順を説明します。

オンプレミスデプロイメントの 3scale ソリューションは、以下の要素で設定されています。

- 2 つの API ゲートウェイ: Embedded APIcast
- 永続ストレージが含まれる 3scale 管理ポータルおよびデベロッパーポータル 1 つ

注記

- 3scale のデプロイに operator を使用する場合、まず Red Hat コンテナーレジストリーへのレジストリー認証を設定する必要があります。コンテナーレジストリー認証の設定 を参照してください。
- 3scale Istio アダプターはオプションのアダプターとして利用可能で、これを使用すると、Red Hat OpenShift Service Mesh 内で実行中のサービスにラベルを付け、そのサービスを 3scale と統合することができます。詳細は、3scale アダプター に関するドキュメントを参照してください。

前提条件

- 3scale サーバーは UTC (協定世界時) に設定しておく。
- レジストリーサービスアカウントの作成 の手順を使用して、ユーザークレデンシャルを作成します。

OpenShift に 3scale をインストールするには、以下のセクションに概略を示す手順を実施します。

- OpenShift に 3scale をインストールするためのシステム要件
- ノードおよびエンタイトルメントの設定
- テンプレートを使用した OpenShift への 3scale のデプロイ
- 3scale テンプレートのパラメーター
- operator を使用した 3scale のデプロイ
- operator を使用した 3scale での高可用性
- operator を使用した OpenShift への 3scale のデプロイメント設定オプション
- システムデータベースに Oracle を使用する 3scale の operator によるインストール
- 典型的な 3scale インストールの問題のトラブルシューティング

2.1. OPENSHIFT に 3SCALE をインストールするためのシステム要件

本セクションでは、3scale - OpenShift テンプレートの要件を示します。

2.1.1. 環境要件

Red Hat 3scale API Management には、Red Hat 3scale API Management のサポート対象設定 に指定されている環境が必要です。

ローカルファイルシステムのストレージを使用している場合は、以下のコマンドを実行します。

永続ボリューム

- Redis および MySQL の永続用の 3 つの RWO (ReadWriteOnce) 永続ボリューム
- デベロッパーポータルコンテンツおよび System-app Assets 用の 1 つの RWX (ReadWriteMany) 永続ボリューム

RWX 永続ボリュームは、グループによる書き込みができるように設定します。必要なアクセスモードをサポートする永続ボリュームタイプのリストは、OpenShift のドキュメント を参照してください。

注記

ネットワークファイルシステム (NFS) は、RWX ボリュームのみ 3scale でサポートされます。

IBM Power (ppc64le) および IBM Z (s390x) の場合は、以下のコマンドを使用してローカルストレージをプロビジョニングします。

ストレージ

- NFS

コンテンツ管理システム (CMS) ストレージに Amazon Simple Storage Service (Amazon S3) バケットを使用している場合は、以下を実行します。

永続ボリューム

- Redis および MySQL の永続用の 3 つの RWO (ReadWriteOnce) 永続ボリューム

ストレージ

- 1 x Amazon S3 バケット
- NFS

2.1.2. ハードウェア要件

ハードウェア要件は、用途のニーズによって異なります。Red Hat は、テストを行い個々の要件を満たすように環境を設定することを推奨します。OpenShift 上に 3scale の環境を設定する場合、以下が推奨されます。

- クラウド環境へのデプロイメントには、コンピュートタスクに最適化したノードを使用する (AWS c4.2xlarge または Azure Standard_F8)。
- メモリーの要件が現在のノードで使用できる RAM よりも大きい場合、非常に大きなインストールでは、Redis に別のノードが必要になることがある (AWS M4 シリーズまたは Azure Av2 シリーズ)。
- ルーティングタスクとコンピュートタスクには別のノードを使用する。

- 3scale 固有のタスクには専用のコンピュートノードを使用する。
- バックエンドリスナーの **PUMA_WORKERS** 変数をコンピュートノードのコア数に設定する。

関連情報

- 永続ストレージについて

2.2. ノードおよびエンタイトルメントの設定

3scale を OpenShift にデプロイする前に、環境が Red Hat Ecosystem Catalog からイメージを取得するのに必要なノードおよびエンタイトルメントを設定する必要があります。ノードとエンタイトルメントを設定するには、以下の手順を実施します。

手順

1. 各ノードに Red Hat Enterprise Linux (RHEL) をインストールします 。
2. インターフェイス または コマンドライン で Red Hat Subscription Manager (RHSM) を使用し、Red Hat にノードを登録します。
3. RHSM を使用して ノードを 3scale サブスクリプションに割り当てます。
4. 以下の要件に準拠して、ノードに OpenShift をインストールします。
   - サポート対象バージョンの OpenShift を使用する。
   - 複数書き込みをサポートするファイルシステムで 永続ストレージ を設定する。
5. OpenShift コマンドラインインターフェイス をインストールします。
6. Subscription Manager を使用して、**rhel-7-server-3scale-amp-2-rpms** リポジトリーへのアクセスを有効にします。

```
sudo subscription-manager repos --enable=rhel-7-server-3scale-amp-2-rpms
```

7. 3scale テンプレート **3scale-amp-template** をインストールします。このテンプレートは **/opt/amp/templates** に保存されます。

```
sudo yum install 3scale-amp-template
```

2.2.1. Amazon Simple Storage Service の設定

重要

ローカルファイルシステムストレージで 3scale をデプロイする場合は、本セクションを飛ばして次に進んでください。

Amazon Simple Storage Service (Amazon S3) バケットをストレージとして使用する場合には、3scale を OpenShift にデプロイする前にバケットを設定する必要があります。

3scale 用の Amazon S3 バケットを設定するには、以下の手順を実施します。

1. 以下の最低限のパーミッションで Identity and Access Management (IAM) ポリシーを作成します。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "s3:ListAllMyBuckets",
         "Resource": "arn:aws:s3:::*"
      },
      {
         "Effect": "Allow",
         "Action": "s3:*",
         "Resource": [
            "arn:aws:s3:::targetBucketName",
            "arn:aws:s3:::targetBucketName/*"
         ]
      }
   ]
}
```

2. 以下のルールで CORS 設定を作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
   <AllowedOrigin>https://*</AllowedOrigin>
   <AllowedMethod>GET</AllowedMethod>
</CORSRule>
</CORSConfiguration>
```

2.3. テンプレートを使用した OPENSHIFT への 3SCALE のデプロイ

注記

OpenShift Container Platform (OCP) 4.x 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す。operator を使用した 3scale のデプロイ を参照してください。

前提条件

- ノードおよびエンタイトルメントの設定 セクションで指定されたとおりに設定された OpenShift クラスター
- OpenShift クラスターに対して解決する ドメイン
- Red Hat Ecosystem Catalog へのアクセス
- コンテンツ管理システム (CMS) ストレージに Amazon Simple Storage Service (Amazon S3) バケットを使用している場合は、以下を実行します。
- (オプション) PostgreSQL を使用したデプロイメント。
    - これは Openshift のデフォルトデプロイメントと同じですが、PostgreSQL を内部システム

データベースとして使用します。

- (オプション) 稼働中の電子メール機能用 SMTP サーバー

注記

テンプレートを使用した OpenShift への 3scale のデプロイは、OpenShift Container Platform 3.11 がベースとなります。

以下の手順に従い、**.yml** テンプレートを使用して 3scale を OpenShift にインストールします。

- コンテナーレジストリー認証の設定
- レジストリーサービスアカウントの作成
- レジストリーサービスアカウントの変更

2.4. 3SCALE テンプレートのパラメーター

テンプレートパラメーターにより、デプロイメント中およびデプロイメント後の 3scale **amp.yml** テンプレートの環境変数を設定します。

表2.1テンプレートパラメーター

名前	説明	デフォルト値	必須/任意
APP_LABEL	オブジェクトアプリのラベルに使用されます。	**3scale-api-management**	必須
ZYNC_DATABASE_PASSWORD	PostgreSQL 接続ユーザーのパスワード。指定のない場合は無作為に生成されます。	該当なし	必須
ZYNC_SECRET_KEY_BASE	Zync の秘密鍵ベース。指定のない場合は無作為に生成されます。	該当なし	必須
ZYNC_AUTHENTICATION_TOKEN	Zync の認証トークン。指定のない場合は無作為に生成されます。	該当なし	必須
AMP_RELEASE	3scale リリースタグ	**2.12.0**	はい
ADMIN_PASSWORD	無作為に生成される 3scale 管理者アカウントのパスワード	該当なし	必須
ADMIN_USERNAME	3scale 管理者アカウントのユーザー名	**admin**	必須

名前	説明	デフォルト値	必須/任意
APICAST_ACCESS_TOKEN	APIcast が設定のダウンロードに使用する読み取り専用アクセストークン	該当なし	必須
ADMIN_ACCESS_TOKEN	すべての API をスコープとし、書き込みアクセス権限が設定された管理者アクセストークン	該当なし	任意
WILDCARD_DOMAIN	ワイルドカードルートのルートドメイン。たとえば、ルートドメイン **example.com** は **3scale-admin.example.com** を生成します。	該当なし	必須
TENANT_NAME	ルート下のテナント名。-admin 接尾辞を付けて管理ポータルにアクセスすることができます。	**3scale**	必須
MYSQL_USER	データベースのアクセスに使用される MySQL ユーザーのユーザー名	**mysql**	必須
MYSQL_PASSWORD	MySQL ユーザーのパスワード	該当なし	必須
MYSQL_DATABASE	アクセスされた MySQL データベースの名前	**system**	必須
MYSQL_ROOT_PASSWORD	Root ユーザーのパスワード	該当なし	必須
SYSTEM_BACKEND_USERNAME	内部 3scale api auth の内部 3scale API ユーザー名	**3scale_api_user**	必須
SYSTEM_BACKEND_PASSWORD	内部 3scale api auth の内部 3scale API パスワード	該当なし	必須
REDIS_IMAGE	使用する Redis イメージ	**registry.redhat.io/rhscl/redis-5-rhel7:5**	はい

名前	説明	デフォルト値	必須/任意
MYSQL_IMAGE	使用する Mysql イメージ	**registry.redhat.io/rhel8/mysql-80:1**	はい
MEMCACHE_SERVERS	memcache サーバーのコンマ区切りの文字列。**system-*** Pod によって使用される memcache サーバーのリングを作成します。	**system-memcache:11211**	はい
	以下に例を示します。**MEMCACHE_SERVERS="cache-1.us-east.domain.com:11211,cache-3.us-east.domain.com:11211,cache-2.us-east.domain.com:11211"**		
MEMCACHED_IMAGE	使用する Memcached イメージ	**registry.redhat.io/3scale-amp2/memcached-rhel7:3scale2.12**	はい
POSTGRESQL_IMAGE	使用する Postgresql イメージ	**registry.redhat.io/rhscl/postgresql-13-rhel7**	はい
AMP_SYSTEM_IMAGE	使用する 3scale システムイメージ	**registry.redhat.io/3scale-amp2/system-rhel7:3scale2.12**	はい
AMP_BACKEND_IMAGE	使用する 3scale バックエンドイメージ	**registry.redhat.io/3scale-amp2/backend-rhel8:3scale2.12**	はい
AMP_APICAST_IMAGE	使用する 3scale APIcast イメージ	**registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.12**	はい
AMP_ZYNC_IMAGE	使用する 3scale Zync イメージ	**registry.redhat.io/3scale-amp2/zync-rhel8:3scale2.12**	はい
SYSTEM_BACKEND_SHARED_SECRET	バックエンドからシステムにイベントをインポートするための共有シークレット	該当なし	必須

名前	説明	デフォルト値	必須/任意
SYSTEM_APP_SECRET_KEY_BASE	システムアプリケーションの秘密鍵ベース	該当なし	必須
APICAST_MANAGEMENT_API	APIcast Management API のスコープ。disable、status、または debug を設定できます。ヘルスチェックには最低でも status が必要です。	**status**	任意
APICAST_OPENSSL_VERIFY	設定のダウンロード時に OpenSSL ピア検証を有効または無効にします。true または false を設定できます。	**false**	任意
APICAST_RESPONSE_CODES	APIcast のロギングレスポンスコードを有効にします。	true	任意
APICAST_REGISTRY_URL	APIcast ポリシーの場所に解決する URL	**http://apicast-staging:8090/policies**	必須
MASTER_USER	マスター管理者アカウントのユーザー名	**master**	必須
MASTER_NAME	マスター管理ポータルのサブドメイン値。**-master** 接尾辞が付けられます。	**master**	必須
MASTER_PASSWORD	無作為に生成されるマスター管理者のパスワード	該当なし	必須
MASTER_ACCESS_TOKEN	API 呼び出しのマスターレベル権限が設定されたトークン	該当なし	必須
IMAGESTREAM_TAG_IMPORT_INSECURE	イメージのインポート中にサーバーが証明書の検証を回避できる、または HTTP 経由で直接接続できる場合は、true を設定します。	**false**	はい

2.5. OPERATOR を使用した 3SCALE のデプロイ

本セクションでは、**APIManager** カスタムリソースを使用して、3scale operator 経由で 3scale ソリューションをインストールおよびデプロイする方法を説明します。

注記

- ワイルドカードルートは、3scale 2.6 以降 廃止されています。
  - この機能は、バックグラウンドで Zync により処理されます。
- API プロバイダーが作成、更新、または削除されると、これらの変更が自動的にルートに反映されます。

前提条件

- コンテナーレジストリー認証の設定
- 3scale operator で自動承認機能を有効にして 3scale のマイクロリリースの自動更新が確実に受信されるようにする。**Automatic** はデフォルトの承認設定です。これを特定のニーズに合わせて変更するには、マイクロリリースの自動アプリケーションの設定 の手順を使用します。
- 先に OpenShift への 3scale operator のインストール の記載の手順に従って Operator を使用して 3scale をデプロイする。
- 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 のみを使用した 3scale のデプロイメントをサポートしています。
  - 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。

以下の手順に従って、operator を使用して 3scale をデプロイします。

- APIManager カスタムリソースのデプロイ
- APIManager 管理ポータルとマスター管理ポータルの認証情報を取得する
- 管理ポータルの URL の取得
- マイクロリリースの自動アプリケーションの設定
- operator を使用した 3scale での高可用性

2.5.1. APIManager カスタムリソースのデプロイ

APIManager カスタムリソースをデプロイすると、operator がプロセスを開始し、そこから 3scale ソリューションがデプロイされます。

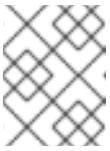
手順

1. **Operators > Installed Operators**の順にクリックします。
   a. **Installed Operators** のリストで、**3scale Operator** をクリックします。
2. **API Manager** タブをクリックします。

3. **Create APIManager** 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。

   - 3scale 2.8 より前のバージョンでは、**highAvailability** フィールドを **true** に設定してレプリカの自動追加を設定できるようになりました。3scale 2.8 以降、レプリカの追加は以下の例のように APIManager CR の replicas フィールドによって制御されます。

     注記

     wildcardDomain パラメーターには、有効な DNS ドメインである、IP アドレスに対して解決する任意の名前を指定できます。

   - 最小要件のある APIManager CR:

     ```
     apiVersion: apps.3scale.net/v1alpha1
     kind: APIManager
     metadata:
       name: apimanager-sample
     spec:
       wildcardDomain: example.com
     ```

   - レプリカが設定された APIManager CR:

     ```
     apiVersion: apps.3scale.net/v1alpha1
     kind: APIManager
     metadata:
       name: apimanager-sample
     spec:
       system:
         appSpec:
           replicas: 1
         sidekiqSpec:
           replicas: 1
       zync:
         appSpec:
           replicas: 1
         queSpec:
           replicas: 1
       backend:
         cronSpec:
           replicas: 1
         listenerSpec:
           replicas: 1
         workerSpec:
           replicas: 1
       apicast:
         productionSpec:
           replicas: 1
         stagingSpec:
           replicas: 1
       wildcardDomain: example.com
     ```

2.5.2. APIManager 管理ポータルとマスター管理ポータルの認証情報を取得する

Operator ベースのデプロイ後に 3scale 管理ポータルまたはマスター管理ポータルのいずれかにログインするには、個別のポータルごとに認証情報が必要です。これらの認証情報を取得するには:

1. 次のコマンドを実行して、管理ポータルの認証情報を取得します。

```
oc get secret system-seed -o json | jq -r .data.ADMIN_USER | base64 -d
oc get secret system-seed -o json | jq -r .data.ADMIN_PASSWORD | base64 -d
```

   a. Admin Portal 管理者としてログインして、これらの認証情報が機能していることを確認します。

2. 次のコマンドを実行して、マスター管理ポータルの認証情報を取得します。

```
oc get secret system-seed -o json | jq -r .data.MASTER_USER | base64 -d
oc get secret system-seed -o json | jq -r .data.MASTER_PASSWORD | base64 -d
```

   a. マスター管理ポータル管理者としてログインして、これらの認証情報が機能していることを確認します。

関連情報

APIManager フィールドに関する詳細は、参考のドキュメント を参照してください。

2.5.3. 管理ポータルの URL の取得

operator を使用して 3scale をデプロイすると、固定 URL のデフォルトテナントが作成されます。**3scale-admin.${wildcardDomain}**

3scale の Dashboard には、テナントの新しいポータル URL が表示されます。たとえば、**<wildCardDomain>** が **3scale-project.example.com** の場合、管理ポータル URL は **https://3scale-admin.3scale-project.example.com** となります。

wildcardDomain は、インストール中に指定した **<wildCardDomain>** パラメーターです。以下のコマンドを使用し、ブラウザーでこの一意の URL を開きます。

```
xdg-open https://3scale-admin.3scale-project.example.com
```

オプションとして、**マスターポータル URL** (**master.${wildcardDomain}**) に新しいテナントを作成できます。

2.5.4. マイクロリリースの自動アプリケーションの設定

マイクロリリースの更新を取得し、これらを自動的に適用するには、3scale operator の承認ストラテジーを **Automatic** に設定する必要があります。ここでは、**Automatic** と **Manual** の設定の違いを説明し、もう 1 つから別の設定に変更する手順を説明します。

自動および手動:

- インストール時に、デフォルトで **Automatic** 設定が選択されたオプションになります。新規更新のインストールは、更新が利用可能になると行われます。これは、インストール時または後にいつでも変更できます。

- インストール時に **手動** オプションを選択するか、またはその後のいつでも手動オプションを選択すると、更新が利用可能になった時点で受信されます。次に、**インストール計画** を承認し、独自に適用する必要があります。

手順

1. **Operators > Installed Operators**の順にクリックします。
2. **Installed Operators** の一覧から **3scale API Management** をクリックします。
3. **Subscription** タブをクリックします。**Subscription Details** の見出しの下に、小見出しの **Approval** が表示されます。
4. **Approval** の下のリンクをクリックします。リンクはデフォルトで **Automatic** に設定されます。小見出しのモーダル (**Change Update Approval Strategy**) が表示されます。
5. 任意のオプションを選択します。**Automatic (デフォルト)** または **Manual** をクリックし、**Save** をクリックします。

関連情報

- OperatorHub を使用した Operator インストール の **Approval Strategy** を参照してください。

2.5.5. operator を使用した 3scale での高可用性

operator を使用した 3scale での高可用性 (HA) は、たとえば 1 つ以上のデータベースに障害が発生した場合に、中断なしのアップタイムを提供することを目的としています。

3scale の operator ベースのデプロイメントで HA が必要な場合は、以下の点に注意してください。

- 3scale の重要なデータベースを外部に設定し、デプロイします。重要なデータベースには、システムデータベース、システム redis およびバックエンド redis コンポーネントが含まれます。これらのコンポーネントが高可用性となるようにデプロイおよび設定するようにしてください。
- 3scale をデプロイする前に対応する Kubernetes シークレットを作成して、3scale のこれらのコンポーネントへの接続エンドポイントを指定します。
  - 詳細は、外部データベースモードでのインストール を参照してください。
  - データベース以外のデプロイメント設定についての詳細は、Enabling Pod Disruption Budgets を参照してください。
- **APIManager** カスタムリソース (CR) で、**.spec.externalComponents** 属性を設定して、システムデータベース、システム redis およびバックエンド redis が外部であることを指定します。

```
externalComponents:
  backend:
    redis: true
system:
  database: true
  redis: true
zync:
  database: true
```

さらに zync データベースを高可用性にして、再起動時のキュージョブデータを失う可能性をなくす場合は、以下の点に注意してください。

- zync データベースを外部でデプロイおよび設定します。このデータベースを高可用性の設定でデプロイおよび設定するようにしてください。
- 3scale をデプロイする前に対応する Kubernetes シークレットを作成して、3scale の zync データベースへの接続エンドポイントを指定します。
  - Zync データベースシークレット を参照してください。
- 3scale を設定するには、**APIManager** CR の **.spec.externalComponents.zync.database** 属性を **true** に設定し、zync データベースが外部データベースであることを指定します。

2.6. OPERATOR を使用した OPENSHIFT への 3SCALE のデプロイメント設定オプション

本セクションでは、operator を使用した OpenShift への Red Hat 3scale API Management のデプロイメント設定オプションについて説明します。

前提条件

- コンテナーレジストリー認証の設定
- 先に OpenShift への 3scale operator のインストール の記載の手順に従って Operator を使用して 3scale をデプロイする。
- 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。

2.6.1. Embedded APIcast のプロキシーパラメーターの設定

3scale の管理者は、Embedded APIcast ステージングおよび実稼働環境用のプロキシーパラメーターを設定することができます。本セクションでは、**APIManager** カスタムリソースでプロキシーパラメーターを指定するための参照情報を提供します。つまり、3scale Operator (**APIManager** カスタムリソース) を使用して OpenShift に 3scale をデプロイします。

これらのパラメーターは、**APIManager** CR を初めてデプロイするときに指定できます。または、デプロイされた **APIManager** CR を更新すると、Operator が更新を調整します。APIManager カスタムリソースのデプロイ を参照してください。

Embedded APIcast には、プロキシー関連の 4 つの設定パラメーターがあります。

- **allProxy**
- **httpProxy**
- **httpsProxy**
- **noProxy**

allProxy

allProxyパラメーターは、要求でプロトコル固有のプロキシーが指定されていない場合にサービスに接続するために使用される HTTP または HTTPS プロキシーを指定します。

プロキシーを設定したら、**allProxy** パラメーターをプロキシーのアドレスに設定して APIcast を設定します。プロキシーでは認証機能はサポートされていません。つまり、APIcast では認証された要求はプロキシーには送信されません。

allProxy パラメーターの値は文字列で、デフォルトはなく、パラメーターは必須ではありません。この形式を使用して、**spec.apicast.productionSpec.allProxy** パラメーターまたは **spec.apicast.stagingSpec.allProxy** パラメーターを設定します。

<scheme>://<host>:<port>

以下に例を示します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      allProxy: http://forward-proxy:80
    stagingSpec:
      allProxy: http://forward-proxy:81
```

httpProxy

httpProxy パラメーターは、HTTP サービスへの接続に使用される HTTP プロキシーを指定します。

プロキシーを設定したら、**httpProxy** パラメーターをプロキシーのアドレスに設定して APIcast を設定します。プロキシーでは認証機能はサポートされていません。つまり、APIcast では認証された要求はプロキシーには送信されません。

httpProxy パラメーターの値は文字列で、デフォルトはなく、パラメーターは必須ではありません。この形式を使用して、**spec.apicast.productionSpec.httpProxy** パラメーターまたは **spec.apicast.stagingSpec.httpProxy** パラメーターを設定します。

http://<host>:<port>

以下に例を示します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      httpProxy: http://forward-proxy:80
    stagingSpec:
      httpProxy: http://forward-proxy:81
```

httpsProxy

httpsProxy パラメーターは、サービスへの接続に使用される HTTPS プロキシーを指定します。

プロキシーを設定したら、**httpsProxy** パラメーターをプロキシーのアドレスに設定して APIcast を設定します。プロキシーでは認証機能はサポートされていません。つまり、APIcast では認証された要求はプロキシーには送信されません。

httpsProxy パラメーターの値は文字列で、デフォルトはなく、パラメーターは必須ではありません。この形式を使用して、**spec.apicast.productionSpec.httpsProxy** パラメーターまたは **spec.apicast.stagingSpec.httpsProxy** パラメーターを設定します。

https://<host>:<port>

以下に例を示します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      httpsProxy: https://forward-proxy:80
    stagingSpec:
      httpsProxy: https://forward-proxy:81
```

noProxy

noProxy パラメーターは、ホスト名とドメイン名のコンマ区切りリストを指定します。要求にこれらの名前のいずれかが含まれる場合、APIcast は要求をプロキシーしません。

たとえば、メンテナーンス操作中にプロキシーへのアクセスを停止する必要がある場合は、**noProxy** パラメーターをアスタリスク (*) に設定します。これは、すべての要求で指定されたすべてのホストに一致し、プロキシーを実質的に無効にします。

noProxy パラメーターの値は文字列で、デフォルトはなく、パラメーターは必須ではありません。**spec.apicast.productionSpec.noProxy** パラメーターまたは **spec.apicast.stagingSpec.noProxy** パラメーターを設定するには、コンマ区切りの文字列を指定します。以下に例を示します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      noProxy: theStore,company.com,big.red.com
    stagingSpec:
      noProxy: foo,bar.com,.extra.dot.com
```

関連情報

- **APIManager** カスタムリソースの APIcast ステージング設定のカスタムリソース定義
- **APIManager** カスタムリソースの APIcast 実稼働設定用のカスタムリソース定義

2.6.2. 3scale Operator を使用したカスタム環境の注入

Embedded APIcast を使用する 3scale インストールでは、3scale Operator を使用してカスタム環境を注入できます。Embedded APIcast は、Managed APIcast または Hosted APIcast とも呼ばれます。カスタム環境は、ゲートウェイが提供するすべてのアップストリーム API に APIcast が適用する動作を定義します。カスタム環境を作成するには、Lua コードでグローバル設定を定義します。

3scale のインストールの前または後にカスタム環境を注入できます。カスタム環境を注入した後、および 3scale をインストールした後、カスタム環境を削除できます。3scale Operator は変更を調整します。

前提条件

- 3scale Operator がインストールされている。

手順

1. 注入するカスタム環境を定義する Lua コードを記述します。たとえば、次の **env1.lua** ファイルは、3scale Operator がすべてのサービスに対してロードするカスタムログポリシーを示しています。

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

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

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

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

2. カスタム環境を定義する Lua ファイルからシークレットを作成します。以下に例を示します。

```
oc create secret generic custom-env-1 --from-file=./env1.lua
```

シークレットには複数のカスタム環境を含めることができます。カスタム環境を定義する各ファイルの **'-from-file** オプションを指定します。Operator は各カスタム環境をロードします。

3. 作成したシークレットを参照する **APIManager** カスタムリソースを定義します。以下の例は、カスタム環境を定義するシークレットの参照に関連するコンテンツのみを示しています。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-apicast-custom-environment
spec:
  wildcardDomain: <desired-domain>
  apicast:
    productionSpec:
```

```
    customEnvironments:
      - secretRef:
          name: custom-env-1
  stagingSpec:
    customEnvironments:
      - secretRef:
          name: custom-env-1
```

APIManager カスタムリソースは、カスタム環境を定義する複数のシークレットを参照できます。Operator は各カスタム環境をロードします。

4. カスタム環境を追加する **APIManager** カスタムリソースを作成します。以下に例を示します。

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

次のステップ

カスタム環境を定義するシークレットのコンテンツを更新することはできません。カスタム環境を更新する必要がある場合は、以下のいずれかを実行できます。

- 推奨されるオプションは、別の名前でシークレットを作成し、**APIManager** カスタムリソースフィールド **customEnvironments[].secretRef.name** を更新することです。Operator はローリング更新をトリガーし、更新されたカスタム環境をロードします。
- あるいは、**spec.apicast.productionSpec.replicas** または **spec.apicast.stagingSpec.replicas** を 0 に設定して既存のシークレットを更新してから、**spec.apicast.productionSpec.replicas** または **spec.apicast.stagingSpec.replicas** を以前の値に設定して APIcast をも再デプロイし直します。

2.6.3. 3scale operator によるカスタムポリシーの注入

Embedded APIcast を使用する 3scale インストールでは、3scale operator を使用してカスタムポリシーを注入できます。Embedded APIcast は、Managed APIcast または Hosted APIcast とも呼ばれます。カスタムポリシーを注入すると、ポリシーコードが APIcast に追加されます。次に、以下のいずれかを使用して、カスタムポリシーを API 製品のポリシーチェーンに追加できます。

- 3scale API
- **Product** カスタムリソース

3scale 管理ポータルを使用してカスタムポリシーを製品のポリシーチェーンに追加するには、カスタムポリシーのスキーマを **CustomPolicyDefinition** カスタムリソースに登録する必要もあります。カスタムポリシー登録は、管理ポータルを使用して製品のポリシーチェーンを設定する場合にのみ必要です。

3scale インストールの一部として、またはインストール後にカスタムポリシーを挿入できます。カスタムポリシーを注入し、3scale をインストールした後、**APIManager** CR から指定内容を削除することにより、カスタムポリシーを削除できます。3scale Operator は変更を調整します。

前提条件

- 3scale operator をインストールしているか、以前にインストールしている。
- Write your own policy で説明されているように、カスタムポリシーを定義している。つまり、カスタムポリシーを定義する **my-policy.lua**、**apicast-policy.json**、および **init.lua** ファイルなどをすでに作成している。

手順

1. 1 つのカスタムポリシーを定義するファイルからシークレットを作成します。以下に例を示します。

```
oc create secret generic my-first-custom-policy-secret \
 --from-file=./apicast-policy.json \
 --from-file=./init.lua \
 --from-file=./my-first-custom-policy.lua
```

   複数のカスタムポリシーがある場合は、カスタムポリシーごとにシークレットを作成します。シークレットには、カスタムポリシーを 1 つだけ含めることができます。

2. カスタムポリシーを含む各シークレットを参照する **APIManager** カスタムリソースを定義します。APIcast ステージングと APIcast 実稼働環境に同じシークレットを指定できます。次の例は、カスタムポリシーを含む参照シークレットに関連するコンテンツのみを示しています。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-apicast-custom-policy
spec:
  apicast:
    stagingSpec:
      customPolicies:
        - name: my-first-custom-policy
          version: "0.1"
          secretRef:
            name: my-first-custom-policy-secret
        - name: my-second-custom-policy
          version: "0.1"
          secretRef:
            name: my-second-custom-policy-secret
    productionSpec:
      customPolicies:
        - name: my-first-custom-policy
          version: "0.1"
          secretRef:
            name: my-first-custom-policy-secret
        - name: my-second-custom-policy
          version: "0.1"
          secretRef:
            name: my-second-custom-policy-secret
```

   APIManager カスタムリソースは、さまざまなカスタムポリシーを定義する複数のシークレットを参照できます。Operator は各カスタムポリシーをロードします。

3. カスタムポリシーを含むシークレットを参照する **APIManager** カスタムリソースを作成します。以下に例を示します。

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

次のステップ

カスタムポリシーを定義するシークレットのコンテンツを更新することはできません。カスタムポリシーを更新する必要がある場合は、次のいずれかを実行できます。

- 推奨されるオプションは、別の名前でシークレットを作成し、**APIManager** カスタムリソース **customPolicies** セクションを更新して新しいシークレットを参照することです。Operator はローリング更新をトリガーし、更新されたカスタムポリシーをロードします。

- あるいは、**spec.apicast.productionSpec.replicas** または **spec.apicast.stagingSpec.replicas** を 0 に設定して既存のシークレットを更新してから、**spec.apicast.productionSpec.replicas** または **spec.apicast.stagingSpec.replicas** を以前の値に設定して APIcast をも再デプロイし直します。

2.6.4. 3scale operator を使用した OpenTracing の設定

Embedded APIcast を使用する 3scale インストールでは、3scale operator を使用して OpenTracing を設定できます。OpenTracing は、ステージング環境または実稼働環境用または両方の環境で設定することができます。OpenTracing を有効にすると、APIcast インスタンスに関してより多くの洞察を得て、可観測性を向上できます。

前提条件

- 3scale operator がインストールされているか、またはインストール中である。
- OpenTracing を使用するための APIcast の設定 に記載の前提条件。
- Jaeger がインストールされている。

手順

1. **stringData.config** に OpenTracing 設定の詳細を含めて、シークレットを定義します。これは、OpenTracing 設定の詳細が含まれる属性の唯一有効な値です。その他の仕様では、APIcast が OpenTracing 設定の詳細を受け取れないようにします。以下の例は、有効なシークレット定義を示しています。

```
apiVersion: v1
kind: Secret
metadata:
  name: myjaeger
stringData:
  config: |-
     {
     "service_name": "apicast",
     "disabled": false,
     "sampler": {
       "type": "const",
       "param": 1
     },
     "reporter": {
       "queueSize": 100,
       "bufferFlushInterval": 10,
       "logSpans": false,
       "localAgentHostPort": "jaeger-all-in-one-inmemory-agent:6831"
     },
     "headers": {
       "jaegerDebugHeader": "debug-id",
       "jaegerBaggageHeader": "baggage",
       "TraceContextHeaderName": "uber-trace-id",
       "traceBaggageHeaderPrefix": "testctx-"
```

```
    },
    "baggage_restrictions": {
      "denyBaggageOnInitializationFailure": false,
      "hostPort": "127.0.0.1:5778",
      "refreshInterval": 60
    }
    }
type: Opaque
```

2. シークレットを作成します。たとえば、以前のシークレット定義を **myjaeger.yaml** ファイルに保存した場合は、以下のコマンドを実行します。

```
oc create secret generic myjaeger --from-file myjaeger.yaml
```

3. **OpenTracing** 属性を指定する **APIManager** カスタムリソースを定義します。CR 定義で、**openTracing.tracingConfigSecretRef.name** 属性を OpenTracing 設定の詳細が含まれるシークレットの名前に設定します。以下の例は、OpenTracing の設定に関するコンテンツのみを示しています。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager1
spec:
  apicast:
    stagingSpec:
      ...
      openTracing:
        enabled: true
        tracingLibrary: jaeger
        tracingConfigSecretRef:
          name: myjaeger
    productionSpec:
      ...
        openTracing:
          enabled: true
          tracingLibrary: jaeger
          tracingConfigSecretRef:
            name: myjaeger
```

4. OpenTracing を設定する **APIManager** カスタムリソースを作成します。たとえば、**APIManager** カスタムリソースを **apimanager1.yaml** ファイルに保存した場合は、以下のコマンドを実行します。

```
oc apply -f apimanager1.yaml
```

次のステップ

OpenTracing のインストール方法に応じて、Jaeger サービスユーザーインターフェイスでトレースが表示されるはずです。

関連情報

- APIManager カスタムリソース定義

2.6.5. 3scale operator を使用した Pod レベルでの TLS の有効化

3scale では、実稼働環境用とステージング環境用の 2 つの APIcast インスタンスをデプロイします。TLS は、実稼働用またはステージングのみ、または両方のインスタンスに対して有効にできます。

前提条件

- TLS を有効にするための有効な証明書。

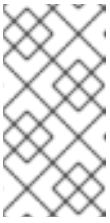
手順

1. 以下のように、有効な証明書からシークレットを作成します。

```
oc create secret tls mycertsecret --cert=server.crt --key=server.key
```

この設定は、**APIManager** CRD のシークレット参照を公開します。シークレットを作成してから、以下のように **APIManager** カスタムリソースでシークレットの名前を参照します。

- 実稼働:**APIManager** CR は **.spec.apicast.productionSpec.httpsCertificateSecretRef** フィールドの証明書を公開します。
- ステージング環境:**APIManager** CR は **.spec.apicast.stagingSpec.httpsCertificateSecretRef** フィールドの証明書を公開します。
必要に応じて、以下を設定できます。
- **httpsPort** は、APIcast が HTTPS 接続に対してリッスンを開始するポートを示します。これが HTTP ポートと競合する場合には、APIcast はこのポートを HTTPS にのみ使用します。
- **httpsVerifyDepth** は、クライアント証明書チェーンの最大長を定義します。



注記

APImanager CR から有効な証明書および参照を指定します。設定で **httpsPort** にアクセスでき、**httpsCertificateSecretRef** ではない場合、APIcast は組み込まれた自己署名証明書を使用します。これは、推奨されません。

2. **Operators > Installed Operators**の順にクリックします。
3. **Installed Operators** のリストで、**3scale Operator** をクリックします。
4. **API Manager** タブをクリックします。
5. **Create APIManager** をクリックします。
6. 以下の YAML 定義をエディターに追加します。
    a. **production** で有効にする場合は、以下の YAML 定義を設定します。

```
spec:
  apicast:
    productionSpec:
      httpsPort: 8443
```

```
    httpsVerifyDepth: 1
    httpsCertificateSecretRef:
      name: mycertsecret
```

b. **staging** で有効にする場合は、以下の YAML 定義を設定します。

```
spec:
  apicast:
    stagingSpec:
      httpsPort: 8443
      httpsVerifyDepth: 1
      httpsCertificateSecretRef:
        name: mycertsecret
```

7. **Create** をクリックします。

2.6.6. 評価用デプロイメントの概念実証

以降のセクションで、3scale の評価用デプロイメントの概念実証に適用される設定オプションを説明します。このデプロイメントでは、デフォルトとして内部データベースが使用されます。

重要

外部データベースの設定は、実稼働環境向けの標準デプロイメントオプションです。

2.6.6.1. デフォルトのデプロイメント設定

- コンテナーには、Kubernetes によるリソースの制限およびリクエスト が適用されます。
  - これにより、最低限のパフォーマンスレベルが確保されます。
  - また、外部サービスおよびソリューションの割り当てを可能にするために、リソースを制限します。
- 内部データベースのデプロイメント
- ファイルストレージは、永続ボリューム (PV) がベースになります。
  - ボリュームの 1 つには、読み取り、書き込み、実行 (RWX) アクセスモードが必要です。
  - OpenShift は、リクエストに応じてこれらを提供するように設定されている必要があります。
- MySQL を内部リレーショナルデータベースとしてデプロイします。

デフォルトの設定オプションは、お客様による概念実証 (PoC) または評価用途に適しています。

1 つ、複数、またはすべてのデフォルト設定オプションを、**APIManager** カスタムリソースの特定フィールドの値で上書きすることができます。3scale operator では可能なすべての組み合わせが許可されますが、テンプレートでは固定のデプロイメントプロファイルが許可されます。たとえば、3scale operator を使用すると、評価モードおよび外部データベースモードで 3scale をデプロイすることができます。テンプレートでは、この特定のデプロイメント設定は許可されません。テンプレートは、最も一般的な設定オプションでしか利用することができません。

2.6.6.2. 評価モードでのインストール

評価モードでのインストールの場合、コンテナーには Kubernetes によるリソースの制限およびリクエスト が適用されません。以下に例を示します。

- メモリーのフットプリントが小さい。
- 起動が高速である。
- ノートパソコンで実行可能である。
- プリセールス/セールスでのデモに適する。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  resourceRequirementsEnabled: false
```

関連情報

- 詳細は、APIManager カスタムリソースを参照してください。

2.6.7. 外部データベースモードでのインストール

外部データベースモードでのインストールは、高可用性 (HA) が必須な場合や専用のデータベースを再利用する場合の実稼働環境での使用に適しています。

重要

3scale の外部データベースインストールモードを有効にすると、以下のデータベースの 1 つまたな複数を 3scale の外部として設定できます。

- **backend-redis**
- **system-redis**
- **system-database** (**mysql**、**postgresql**、または **oracle**)
- **zync-database**

3scale 2.8 以降は、以下のデータベースバージョンとの組み合わせでテストを行いサポートが提供されます。

データベース	バージョン
Redis	5.0
MySQL	8.0
PostgreSQL	13

3scale をデプロイするために **APIManager カスタムリソース** を作成する前に、OpenShift シークレットを使用して以下に示す外部データベースの接続設定を提供する必要があります。

2.6.7.1. バックエンド Redis シークレット

2 つの外部 Redis インスタンスをデプロイし、以下の例に示すように接続設定を入力します。

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-redis
stringData:
  REDIS_STORAGE_URL: "redis://backend-redis-storage"
  REDIS_STORAGE_SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-1.example.com:26379, redis://sentinel-2.example.com:26379"
  REDIS_STORAGE_SENTINEL_ROLE: "master"
  REDIS_QUEUES_URL: "redis://backend-redis-queues"
  REDIS_QUEUES_SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-1.example.com:26379, redis://sentinel-2.example.com:26379"
  REDIS_QUEUES_SENTINEL_ROLE: "master"
type: Opaque
```

シークレット 名は **backend-redis** にする必要があります。

2.6.7.2. システム Redis シークレット

2 つの外部 Redis インスタンスをデプロイし、以下の例に示すように接続設定を入力します。

```
apiVersion: v1
kind: Secret
metadata:
  name: system-redis
stringData:
  URL: "redis://system-redis"
  SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-1.example.com:26379, redis://sentinel-2.example.com:26379"
  SENTINEL_ROLE: "master"
  NAMESPACE: ""
type: Opaque
```

シークレット 名は **system-redis** にする必要があります。

2.6.7.3. システムデータベースシークレット

注記

- **シークレット** 名は **system-database** にする必要があります。

3scale をデプロイする場合には、システムデータベースに 3 つの代替手段があります。代替手段に関連のシークレットごとに、異なる属性と値を設定します。

- MySQL
- PostgreSQL

- Oracle データベース

MySQL、PostgreSQL、または Oracle Database のシステムデータベースシークレットをデプロイするには、以下の例のように接続設定を入力します。

MySQL システムデータベースシークレット

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "mysql2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
type: Opaque
```

重要

3scale 2.12 で MySQL 8.0 を使用する場合は、認証プラグインを **mysql_native_password** に設定する必要があります。MySQL 設定ファイルに以下を追加します。

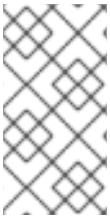
```
[mysqld]
default_authentication_plugin=mysql_native_password
```

PostgreSQL システムデータベースシークレット

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "postgresql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
type: Opaque
```

Oracle システムデータベースシークレット

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "oracle-enhanced://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
  ORACLE_SYSTEM_PASSWORD: "{SYSTEM_PASSWORD}"
type: Opaque
```



注記

- **{DB_USER}** および **{DB_PASSWORD}** は、通常の非システムユーザーのユーザー名およびパスワードです。
- **{DB_NAME}** は Oracle Database の サービス名 です。

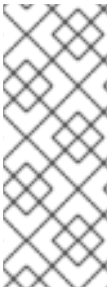
2.6.7.4. Zync データベースシークレット

zync データベースの設定では、**spec.externalComponents.zync.database** フィールドが **true** に設定されている場合、3scale をデプロイする前に **zync** という名前のシークレットを作成する必要があります。このシークレットでは、**DATABASE_URL** および **DATABASE_PASSWORD** フィールドを外部の zync データベースを参照する値に設定します。以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: zync
stringData:
  DATABASE_URL: postgresql://<zync-db-user>:<zync-db-password>@<zync-db-host>:<zync-db-port>/zync_production
  ZYNC_DATABASE_PASSWORD: <zync-db-password>
type: Opaque
```

zync データベースは高可用性モードである必要があります。

2.6.7.5. 3scale をデプロイするための APIManager カスタムリソース



注記

- 外部コンポーネントを有効にする場合は、3scale をデプロイする前に、外部コンポーネント (**backend-redis**、**system-redis**、**system-database**、**zync**) ごとにシークレットを作成する必要があります。
- 外部の **system-database** の場合は、外部化するデータベースのタイプを 1 つだけ選択します。

APIManager カスタムリソースの設定は、選択したデータベースが 3scale デプロイメントの外部にあるかどうかによって異なります。

backend-redis、**system-redis**、または **system-database** が 3scale の外部になる場合、**APIManager** カスタムリソースは対応する **externalComponents** オブジェクトを反映させる必要があります。以下に例を示します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  externalComponents:
    system:
      database: true
```

関連情報

- Backend redis secret
- システムデータベースシークレット
- APIManager ExternalComponentsSpec

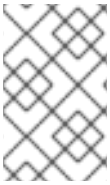
- Zync secret

2.6.8. Amazon Simple Storage Service を使用した 3scale ファイルストレージ のインストール

以下の例で、永続ボリューム要求 (PVC) の代わりに Amazon Simple Storage Service (Amazon S3) を使用した 3scale **ファイルストレージ** について説明します。

3scale をデプロイするために **APIManager** カスタムリソースを作成する前に、OpenShift シークレットを使用して S3 サービスの接続設定を提供する必要があります。

2.6.8.1. Amazon S3 シークレット



注記

AWS S3 互換プロバイダーは、**AWS_HOSTNAME**、**AWS_PATH_STYLE**、および **AWS_PROTOCOL** オプションキーを使用して S3 シークレットで設定できます。詳細は S3 secret リファレンス を参照してください。

以下の例では、任意の **シークレット** 名を指定することができます。シークレット名が **APIManager** カスタムリソースで参照されるためです。

```
kind: Secret
metadata:
  creationTimestamp: null
  name: aws-auth
stringData:
  AWS_ACCESS_KEY_ID: 123456
  AWS_SECRET_ACCESS_KEY: 98765544
  AWS_BUCKET: mybucket.example.com
  AWS_REGION: eu-west-1
type: Opaque
```

注記

Amazon S3 リージョンおよび Amazon S3 バケット設定は、**APIManager** カスタムリソースに直接提供されます。Amazon S3 シークレット名は、**APIManager** カスタムリソースに直接提供されます。

最後に、3scale をデプロイするための **APIManager** カスタムリソースを作成します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  system:
    fileStorage:
      simpleStorageService:
        configurationSecretRef:
          name: aws-auth
```

詳細は、APIManager SystemS3Spec を参照してください。

2.6.9. PostgreSQL のインストール

MySQL 内部リレーショナルデータベースがデフォルトのデプロイメントです。このデプロイメント設定を上書きして、代わりに PostgreSQL を使用することができます。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  system:
    database:
      postgresql: {}
```

関連情報

- 詳細は、APIManager DatabaseSpec を参照してください。

2.6.10. コンポーネントレベルでのコンピュートリソース要件のカスタマイズ

APIManager カスタムリソース属性を使用して、3scale ソリューションの Kubernetes コンピュートリソース要件 をカスタマイズします。この操作により、特定の **APIManager** コンポーネントに割り当てられるコンピュートリソース (CPU およびメモリー) の要件をカスタマイズします。

以下の例で、**backend-listener** および **zync-database** の system-master の **system-provider** コンテナーに対するコンピュートリソース要件をカスタマイズする方法の概要を説明します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      resources:
        requests:
          memory: "150Mi"
          cpu: "300m"
        limits:
          memory: "500Mi"
          cpu: "1000m"
  system:
    appSpec:
      providerContainerResources:
        requests:
          memory: "111Mi"
          cpu: "222m"
        limits:
          memory: "333Mi"
          cpu: "444m"
  zync:
```

```
databaseResources:
  requests:
    memory: "111Mi"
    cpu: "222m"
  limits:
    memory: "333Mi"
    cpu: "444m"
```

関連情報

コンポーネントレベルのカスタムリソース要件の指定方法についての詳細は、APIManager CRD reference を参照してください。

2.6.10.1. APIManager コンポーネントのデフォルトコンピュートリソース

APIManager の **spec.resourceRequirementsEnabled** 属性を **true** に設定すると、デフォルトのコンピュートリソースが **APIManager** コンポーネントに設定されます。

以下の表に、**APIManager** コンポーネントに設定された特定のコンピュートリソースのデフォルト値をまとめます。

2.6.10.1.1. CPU およびメモリーの単位

コンピュートリソースのデフォルト値の表に使用される単位について、以下のリストにまとめます。CPU およびメモリーの単位の詳細は、Managing Resources for Containers を参照してください。

リソースの単位について

- m: ミリ CPU またはミリコア
- Mi: メビバイト
- Gi: ギビバイト
- G: ギガバイト

表2.2 コンピュートリソースのデフォルト値

コンポーネント	CPU 要求	CPU 上限	メモリー要求	メモリー上限
system-app の system-master	50 m	1000 m	600 Mi	800 Mi
system-app の system-provider	50 m	1000 m	600 Mi	800 Mi
system-app の system-developer	50 m	1000 m	600 Mi	800 Mi
system-sidekiq	100 m	1000 m	500 Mi	2 Gi
system-sphinx	80 m	1000 m	250 Mi	512 Mi

コンポーネント	CPU 要求	CPU 上限	メモリー要求	メモリー上限
system-redis	150 m	500 m	256 Mi	32 Gi
system-mysql	250 m	制限なし	512 Mi	2 Gi
system-postgresql	250 m	制限なし	512 Mi	2 Gi
backend-listener	500 m	1000 m	550 Mi	700 Mi
backend-worker	150 m	1000 m	50 Mi	300 Mi
backend-cron	50 m	150 m	40 Mi	80 Mi
backend-redis	1000 m	2000 m	1024 Mi	32 Gi
apicast-production	500 m	1000 m	64 Mi	128 Mi
apicast-staging	50 m	100 m	64 Mi	128 Mi
zync	150 m	1	250 M	512 Mi
zync-que	250 m	1	250 M	512 Mi
zync-database	50 m	250 m	250 M	2 G

2.6.11. コンポーネントレベルでのノードのアフィニティーおよび容認のカスタマイズ

APIManager カスタムリソース属性を使用して Red Hat 3scale API Management ソリューションの Kubernetes の アフィニティー および 容認 をカスタマイズし、インストールのさまざまな 3scale コンポーネントが Kubernetes ノードにスケジュールされる場所および方法をカスタマイズします。

以下の例では、バックエンドのカスタムノードのアフィニティーを設定します。また、**system-memcached** のリスナーおよびカスタム容認も設定します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: "kubernetes.io/hostname"
                operator: In
```

```
            values:
            - ip-10-96-1-105
          - key: "beta.kubernetes.io/arch"
            operator: In
            values:
            - amd64
system:
  memcachedTolerations:
  - key: key1
    value: value1
    operator: Equal
    effect: NoSchedule
  - key: key2
    value: value2
    operator: Equal
    effect: NoSchedule
```

関連情報

アフィニティーおよび容認に関連する属性の完全リストは、APIManager CDR を参照してください。

2.6.12. 調整

3scale をインストールしたら、3scale operator により、カスタムリソースからの特定パラメーターセットを更新してシステム設定オプションを変更することができます。変更は **ホットスワップ** により行われます。つまり、システムの停止やシャットダウンは発生しません。

APIManager カスタムリソース定義 (CRD) のパラメーターがすべて調整可能な訳ではありません。

調整可能なパラメーターのリストを以下に示します。

- 「リソース」
- 「バックエンドレプリカ」
- 「APIcast レプリカ」
- 「システムレプリカ」

2.6.12.1. リソース

すべての 3scale コンポーネントに対するリソースの制限およびリクエスト

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  ResourceRequirementsEnabled: true/false
```

2.6.12.2. バックエンドレプリカ

バックエンド コンポーネントの Pod 数

```
apiVersion: apps.3scale.net/v1alpha1
```

```
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      replicas: X
    workerSpec:
      replicas: Y
    cronSpec:
      replicas: Z
```

2.6.12.3. APIcast レプリカ

APIcast ステージングおよび実稼働環境コンポーネントの Pod 数

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

2.6.12.4. システムレプリカ

システム アプリケーションおよびシステム **sidekiq** コンポーネントの Pod 数

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  system:
    appSpec:
      replicas: X
    sidekiqSpec:
      replicas: Z
```

2.6.12.5. Zync レプリカ

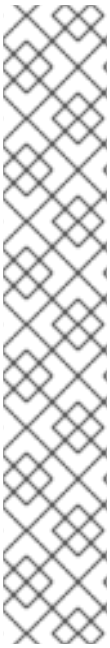
Zync アプリケーションと que コンポーネントの Pod 数

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  zync:
    appSpec:
```

```
    replicas: X
  queSpec:
    replicas: Z
```

2.7. システムデータベースに ORACLE を使用する 3SCALE の OPERATOR によるインストール

Red Hat 3scale API Management 管理者は、Oracle Database を使用する 3scale を operator によりインストールすることができます。デフォルトでは、3scale 2.12 には設定データを MySQL データベースに保管する **system** というコンポーネントが含まれています。このデフォルトのデータベースをオーバーライドし、情報を外部の Oracle Database に保管することができます。



注記

- operator のみで 3scale のインストールを実行する場合には、Oracle Database は OpenShift Container Platform (OCP) のバージョン 4.2 および 4.3 ではサポートされません。詳細は、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。
- このドキュメントでは、レジストリー URL の例として **myregistry.example.com** が使用されています。これは、お使いのレジストリー URL に置き換えてください。
- 免責条項:**以下に示す外部の Web サイトへのリンクは、お客様の利便性のみを目的として提供しています。Red Hat はリンクの内容を確認しておらず、コンテンツまたは可用性について責任を負わないものとします。外部の Web サイトへのリンクを含めることは、Web サイトまたはそれらの法的主体、製品またはサービスについて Red Hat が承認したことを意味するものではありません。お客様は、外部サイトまたはコンテンツの使用 (または信頼) によって生じる損失または費用について、Red Hat が責任を負わないことに同意するものとします。**

前提条件

- 3scale がインストールされる OCP クラスターからアクセスすることのできる、コンテナーイメージをプッシュするためのコンテナーレジストリー
- 3scale operator のインストール
  - **APIManager** カスタムリソースは以降の手順で作成されるので、インストールしないでください。
- 3scale 用レジストリーサービスアカウント
- OpenShift クラスターからアクセスできる Oracle Database のサポート対象バージョン
- インストール手順に必要な Oracle Database の system ユーザーへのアクセス

システムデータベースに Oracle を使用する 3scale を operator によりインストールするには、以下の手順を使用します。

- Oracle Database の準備
- カスタムシステムコンテナーイメージの作成
- operator を使用した Oracle での 3scale のインストール

2.7.1. Oracle Database の準備

3scale 管理者は、Oracle Database をデフォルトとして使用することを決定した場合、3scale インストール用に Oracle Database を完全に準備する必要があります。

手順

1. 新しいデータベースを作成します。
2. 次の設定を適用します。

```
ALTER SYSTEM SET max_string_size=extended SCOPE=SPFILE;
```

3. 3scale を使用して Oracle データベースをインストールする前に、通常の非システムユーザーを作成します。
   a. 通常の Oracle Database の非システムユーザーのパスワードは一意で、**system** パスワードとは一致しないようにする必要があります。
   b. Oracle データベースの初期化スクリプトは、次の SQL コマンドを実行します。

```
ALTER USER {DB_USER} IDENTIFIED BY {DB_PASSWORD}
```

   - 3scale のインストールを開始する前に、この SQL コマンドが正しく実行することを確認してください。
   - **{DB_USER}** および **{DB_PASSWORD}** は、通常の非システムユーザーのユーザー名およびパスワードです。
   - パラメーター **PASSWORD_REUSE_TIME** および **PASSWORD_REUSE_MAX** が同じパスワードの再利用を制限するように設定されている場合、データベース設定によっては、このコマンドが正常に完了しない場合があります。
   - Oracle Database の **system** ユーザーは、システム権限でコマンドを実行します。
     - 一部は、こちらの GitHub リポジトリー に詳細があります。
     - テーブルがデータベースで初期化されると、Oracle Database のイニシャライザー で最新のものを実行できます。これらのリンクにリストされていない他のコマンドが存在する可能性があります。
   - スキーマ移行がある場合に、アップグレードするには **system** ユーザーが必要であるため、直前のリンクに含まれていない他のコマンドを実行することもできます。

関連情報

- 新しいデータベースの作成については、Oracle Database 19c ドキュメントを参照してください。

2.7.2. カスタムシステムコンテナーイメージの作成

手順

1. GitHub リポジトリーから 3scale OpenShift テンプレートをダウンロードし、アーカイブを展開します。

```
tar -xzf 3scale-amp-openshift-templates-3scale-2.12.0-GA.tar.gz
```

2. **Instant Client Downloads** ページから、以下をダウンロードします。
   - クライアント:**basic-lite** または **basic** のいずれかを使用できます。
   - **ODBC ドライバー**
   - Oracle Database 19c の **SDK**
     - 3scale の場合は、Instant Client Downloads for Linux x86-64(64-bit) を使用します。
     - ppc64le および 3scale の場合は、Power Little Endian (64 ビット) 上の Linux の Oracle Instant Client Downloads for Linux を使用します。
3. 以下の Oracle ソフトウェアコンポーネントについては、Red Hat 3scale API Management のサポート対象設定 を確認してください。
   - Oracle Instant Client パッケージ:Basic または Basic Light
   - Oracle Instant Client パッケージ:SDK
   - Oracle Instant Client パッケージ:ODBC

表2.3 3scale 向けの Oracle 19c パッケージの例

Oracle 19c パッケージ名	圧縮ファイル名
Basic	instantclient-basic-linux.x64-19.8.0.0.0dbru.zip
Basic Light	instantclient-basiclite-linux.x64-19.8.0.0.0dbru.zip
SDK	instantclient-sdk-linux.x64-19.8.0.0.0dbru.zip
ODBC	instantclient-odbc-linux.x64-19.8.0.0.0dbru.zip

表2.4 ppc64le および 3scale 用 Oracle 19c パッケージの例

Oracle 19c パッケージ名	圧縮ファイル名
Basic	instantclient-basic-linux.leppc64.c64-19.3.0.0.0dbru.zip
Basic Light	instantclient-basiclite-linux.leppc64.c64-19.3.0.0.0dbru.zip
SDK	instantclient-sdk-linux.leppc64.c64-19.3.0.0.0dbru.zip

Oracle 19c パッケージ名	圧縮ファイル名
ODBC	instantclient-odbc-linux.leppc64.c64-19.3.0.0.0dbru.zip

注記

ローカルにダウンロードされて保存されたクライアントパッケージバージョンが、3scale が想定するバージョンと一致しない場合には、以下の手順で 3scale は適切なバージョンを自動的にダウンロードして使用します。

4. Oracle Database の Instant Client パッケージファイルを **3scale-amp-openshift-templates-3scale-2.12-GA/amp/system-oracle/oracle-client-files** ディレクトリーに配置します。

5. レジストリーサービスアカウントの作成 で作成したクレデンシャルを使用して、**registry.redhat.io** アカウントにログインします。

```
$ docker login registry.redhat.io
```

6. システムの Oracle ベースのカスタムイメージをビルドします。以下の例に示すように、固定のイメージタグを設定する必要があります。

```
$ docker build . --tag myregistry.example.com/system-oracle:2.12.0-1
```

7. システムの Oracle ベースのイメージを、OCP クラスターからアクセス可能なコンテナーレジストリーにプッシュします。このコンテナーレジストリーに、この後 3scale ソリューションがインストールされます。

```
$ docker push myregistry.example.com/system-oracle:2.12.0-1
```

2.7.3. operator を使用した Oracle での 3scale のインストール

手順

1. 該当するフィールドを使用して **system-database** シークレットを作成し、Oracle Database URL の接続文字列および Oracle Database のシステムパスワードを設定します。Oracle Database については、外部データベースモードでのインストール を参照してください。

2. **APIManager** カスタムリソースを作成して、3scale ソリューションをインストールします。operator を使用した 3scale のデプロイ に記載の手順に従います。

   - **APIManager** カスタムリソースでは、**.spec.system.image** フィールドを前のステップでビルドしたシステムの Oracle ベースのイメージに設定する必要があります。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  imagePullSecrets:
```

```
- name: threescale-registry-auth
- name: custom-registry-auth
system:
  image: "myregistry.example.com/system-oracle:2.12.0-1"
externalComponents:
  system:
    database: true
```

2.8. 典型的な 3SCALE インストールの問題のトラブルシューティング

本セクションでは、典型的なインストールの問題と、その問題を解決するためのアドバイスについて説明します。

- 以前のデプロイメントがダーティーな永続ボリューム要求を残す
- 認証されたイメージレジストリーの認証情報が間違っているか、欠落している
- 誤って Docker レジストリーからプルされる
- 永続ボリュームがローカルでマウントされている場合の MySQL の権限の問題
- ロゴまたはイメージをアップロードできない
- OpenShift でテストコールが動作しない
- 3scale 以外のプロジェクトでの APIcast デプロイに失敗する

2.8.1. 以前のデプロイメントがダーティーな永続ボリューム要求を残す

問題

以前のデプロイメントがダーティーな永続ボリューム要求 (PVC) を残そうとするため、MySQL コンテナーの起動に失敗する。

原因

OpenShift のプロジェクトを削除しても、それに関連する PVC は消去されない。

解決方法

手順

1. **oc get pvc** コマンドを使用してエラーのある MySQL データが含まれる PVC を探します。

```
# oc get pvc
NAME                     STATUS    VOLUME    CAPACITY   ACCESSMODES   AGE
backend-redis-storage    Bound     vol003    100Gi      RWO,RWX       4d
mysql-storage            Bound     vol006    100Gi      RWO,RWX       4d
system-redis-storage     Bound     vol008    100Gi      RWO,RWX       4d
system-storage           Bound     vol004    100Gi      RWO,RWX       4d
```

2. OpenShift UI の **cancel deployment** をクリックして、system-mysql Pod のデプロイメントを停止します。
3. MySQL パス以下にあるものすべてを削除し、ボリュームをクリーンアップします。

4. 新たに **system-mysql** のデプロイメントを開始します。

2.8.2. 認証されたイメージレジストリーの認証情報が間違っているか、欠落している

問題

Pod が起動していません。ImageStreams に次のエラーが表示されます。

```
! error: Import failed (InternalError): ...unauthorized: Please login to the Red Hat Registry
```

原因

OpenShift 4.x に 3scale をインストールすると、ImageStreams が参照するイメージをプルできないため、OpenShift は Pod の起動に失敗します。これは、Pod が指しているレジストリーに対して認証できないために発生します。

解決方法

手順

1. 次のコマンドを入力して、コンテナーレジストリー認証の設定を確認します。

```
$ oc get secret
```

- シークレットが存在する場合は、ターミナルに次の出力が表示されます。

```
threescale-registry-auth        kubernetes.io/dockerconfigjson        1        4m9s
```

- ただし、出力が表示されない場合は、次の操作を行う必要があります。

2. レジストリーサービスアカウントの作成 中に以前に設定した認証情報を使用して、シークレットを作成します。

3. 提供されている **oc create secret** コマンドの **<your-registry-service-account-username>** および **<your-registry-service-account-password>** を置き換えて、OpenShift でのレジストリー認証の設定 の手順を使用します。

4. **APIManager** リソースと同じ名前空間で **threescale-registry-auth** シークレットを生成します。**<project-name>** 内で次を実行する必要があります。

```
oc project <project-name>
oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username="<your-registry-service-account-username>" \
  --docker-password="<your-registry-service-account-password>"
  --docker-email="<email-address>"
```

5. **APIManager** リソースを削除して再作成します。

```
$ oc delete -f apimanager.yaml
apimanager.apps.3scale.net "example-apimanager" deleted

$ oc create -f apimanager.yaml
apimanager.apps.3scale.net/example-apimanager created
```

検証

1. 次のコマンドを入力して、デプロイのステータスが **Starting** または **Ready** であることを確認します。その後、Pod が以下を生成し始めます。

```
$ oc describe apimanager
(...)
Status:
  Deployments:
    Ready:
      apicast-staging
      system-memcache
      system-mysql
      system-redis
      zync
      zync-database
      zync-que
    Starting:
      apicast-production
      backend-cron
      backend-worker
      system-sidekiq
      system-sphinx
    Stopped:
      backend-listener
      backend-redis
      system-app
```

2. 以下のコマンドを実行して、各 Pod のステータスを確認します。

```
$ oc get pods
NAME                              READY   STATUS            RESTARTS   AGE
3scale-operator-66cc6d857b-sxhgm  1/1     Running           0          17h
apicast-production-1-deploy       1/1     Running           0          17m
apicast-production-1-pxkqm        0/1     Pending           0          17m
apicast-staging-1-dbwcw           1/1     Running           0          17m
apicast-staging-1-deploy          0/1     Completed         0          17m
backend-cron-1-deploy             1/1     Running           0          17m
```

2.8.3. 誤って Docker レジストリーからプルされる

問題

インストール中に以下のエラーが発生する。

```
svc/system-redis - 1EX.AMP.LE.IP:6379
  dc/system-redis deploys docker.io/rhscl/redis-32-rhel7:3.2-5.3
    deployment #1 failed 13 minutes ago: config change
```

原因

OpenShift は **docker** コマンドを実行し、コンテナーイメージを検索およびプルします。このコマンドは、**registry.redhat.io** Red Hat Ecosystem Catalog ではなく、 **docker.io** Docker レジストリーを参照します。

これは、システムに予期せぬバージョンの Docker コンテナー環境が含まれる場合に発生します。

解決方法

手順

適切なバージョン の Docker コンテナー環境を使用します。

2.8.4. 永続ボリュームがローカルでマウントされている場合の MySQL の権限の問題

問題

system-msql Pod がクラッシュし、デプロイされないため、それに依存する他のシステムのデプロイメントに失敗する。Pod ログに以下のエラーが記録される。

```
[ERROR] Cannot start server : on unix socket: Permission denied
[ERROR] Do you already have another mysqld server running on socket: /var/lib/mysql/mysql.sock ?
[ERROR] Aborting
```

原因

MySQL プロセスが不適切なユーザー権限で起動されている。

解決方法

手順

1. 永続ボリュームに使用されるディレクトリーには、root グループの書き込み権限が必要です。MySQL サービスは root グループの別のユーザーとして実行されるため、root ユーザーの読み取り/書き込み権限では不十分です。root ユーザーとして以下のコマンドを実行します。

```
chmod -R g+w /path/for/pvs
```

2. 以下のコマンドを実行して、SELinux がアクセスをブロックしないようにします。

```
chcon -Rt svirt_sandbox_file_t /path/for/pvs
```

2.8.5. ロゴまたはイメージをアップロードできない

問題

ロゴをアップロードできず、**system-app** ログに以下のエラーが表示される。

```
Errno::EACCES (Permission denied @ dir_s_mkdir - /opt/system/public//system/provider-name/2
```

原因

OpenShift が永続ボリュームに書き込みを行うことができない。

解決方法

手順

OpenShift が永続ボリュームに書き込みを行えるようにします。永続ボリュームのグループ所有者を root グループにし、またグループによる書き込みを可能にしなければなりません。

2.8.6. OpenShift でテストコールが動作しない

問題

OpenShift で新しいサービスとルートを作成した後に、テストコールが動作しない。curl 経由のダイレクトコールも失敗し、**service not available** が出力される。

原因

3scale はデフォルトで HTTPS ルートが必要で、OpenShift ルートはセキュアではない。

解決方法

手順

OpenShift のルーター設定で **secure route** チェックボックスが選択されていることを確認します。

2.8.7. 3scale 以外のプロジェクトでの APIcast デプロイに失敗する

問題

APIcast のデプロイに失敗する (Pod が青にならない)。以下のエラーがログに表示される。

```
update acceptor rejected apicast-3: pods for deployment "apicast-3" took longer than 600 seconds to become ready
```

以下のエラーが Pod に表示される。

```
Error synching pod, skipping: failed to "StartContainer" for "apicast" with RunContainerError: "GenerateRunContainerOptions: secrets \"apicast-configuration-url-secret\" not found"
```

原因

シークレットが適切に設定されていない。

解決方法

手順

APIcast v3 でシークレットを作成する時に **apicast-configuration-url-secret** を指定します。

```
oc create secret generic apicast-configuration-url-secret --from-literal=password=https://<ACCESS_TOKEN>@<TENANT_NAME>-admin.<WILDCARD_DOMAIN>
```

2.9. 関連情報

- 高可用性についての詳細は、APIManager CRD リファレンス に関するドキュメントを参照してください。
- **system-database** シークレットおよびフィールドについての詳細は、APIManager CRD リファレンス に関するドキュメントを参照してください。

第3章 APICAST のインストール

APIcast は、内部および外部の API サービスを Red Hat 3scale API Management Platform と統合するのに使用される NGINX ベースの API ゲートウェイです。APIcast は、ラウンドロビン形式で負荷分散を行います。

本章では、デプロイメントオプション、提供される環境、および使用を開始する方法について説明します。

前提条件

APIcast がスタンドアロンの API ゲートウェイではない。また、3scale API Manager への接続が必要である。

- 稼働中の オンプレミス型 3scale インスタンス

APIcast をインストールするには、以下のセクションに概略を示す手順を実施します。

- APIcast デプロイメントのオプション
- APIcast の環境
- インテグレーション設定
- サービスの設定
- Docker コンテナー環境への APIcast のデプロイ
- OpenShift テンプレートを使用した APIcast のデプロイ
- operator を使用した Self-managed APIcast ゲートウェイソリューションのデプロイ

3.1. APICAST デプロイメントのオプション

Hosted APIcast (ホスト型 APIcast) または Self-managed APIcast (自己管理型 APIcast) を使用できます。どちらの場合にも、APIcast は残りの 3scale API Management プラットフォームに接続している必要があります。

- **Embedded APIcast**3scale API Management インストールにはデフォルトで 2 つの APIcast ゲートウェイ (ステージングと実稼働) が含まれています。これらのゲートウェイは事前設定されているため、そのまま使用することができます。
- **Self-managed APIcast**:APIcast を任意の場所にデプロイできます。APIcast のデプロイメントにおける推奨オプションの一部は以下のとおりです。
  - Docker コンテナー環境への APIcast のデプロイ :そのまま使用できる Docker 形式のコンテナーイメージをダウンロードします。これには、Docker 形式のコンテナーで APIcast を実行するための依存関係がすべて含まれています。
  - Red Hat OpenShift 上での APIcast の実行: APIcast を サポート対象バージョン の OpenShift で実行します。Self-managed APIcast は、オンプレミス型 3scale インストール環境またはホスト型 3scale (SaaS) アカウントに接続できます。このために、Openshift テンプレートを使用して APIcast をデプロイ するか、Operator を使用して APIcast ゲートウェイのセルフマネージドソリューションをデプロイ するオプションがあります。

3.2. APICAST の環境

デフォルトでは、3scale アカウントを作成すると、2 つの異なる環境の Embedded APIcast が提供されます。

- **ステージング環境**:API インテグレーションの設定中またはテスト中にのみ使用することが目的です。設定が想定どおりに動作していることが確認されたら、実稼働環境にデプロイすることができます。OpenShift テンプレートは、設定が各 API 呼び出し (**APICAST_CONFIGURATION_LOADER: lazy、APICAST_CONFIGURATION_CACHE:) で再読み込みされるよう、ステージング APIcast のパラメーターを設定します。0**)。APIcast の設定変更を即座にテストするのに便利です。

- **実稼働**:実稼働向けの環境です。以下のパラメーターは、OpenShift テンプレートの実稼働 APIcast のために設定されています。**APICAST_CONFIGURATION_LOADER: boot、APICAST_CONFIGURATION_CACHE:300**.そのため、APIcast の開始時に設定が完全に読み込まれ、300 秒 (5 分) 間キャッシュに保存されます。設定は 5 分後に再読み込みされます。これにより、設定を実稼働環境にプロモートすると、APIcast の新しいデプロイメントを実行しない限り、設定の適用に 5 分程度かかる場合があります。

3.3. インテグレーション設定

3scale 管理者は、3scale を実行する必要のある環境のインテグレーション設定を行います。

前提条件

管理者権限が設定された 3scale アカウント

手順

1. **[Your_API_name] > Integration > Settings**の順に移動します。
2. **Deployment** セクションでは、デフォルトのオプションは以下のとおりです。
   - デプロイメントオプション:管理された APIcast 3scale
   - 認証モード:API キー
3. 希望するオプションに変更します。
4. 変更を保存するには、**Update Product** をクリックします。

3.4. サービスの設定

Private Base URL フィールドに API バックエンドのエンドポイントホストを指定して、API バックエンドを宣言する必要があります。すべての認証、承認、流量制御、および統計が処理された後、APIcast はすべてのトラフィックを API バックエンドにリダイレクトします。

本セクションでは、サービスの設定について各手順を説明します。

- API バックエンドの宣言
- 認証の設定
- API テストコールの設定

3.4.1. API バックエンドの宣言

通常、API のプライベートベース URL は、管理するドメイン (**yourdomain.com**) 上で **https://api-backend.yourdomain.com:443** のようになります。たとえば、Twitter API と統合する場合、プライベートベース URL は **https://api.twitter.com/** になります。

この例では、3scale がホストする **Echo API** を使用します。これは、任意のパスを受け入れ、リクエストに関する情報 (パス、リクエストパラメーター、ヘッダーなど) を返すシンプルな API です。このプライベートベース URL は **https://echo-api.3scale.net:443** になります。

手順

- プライベート (アンマネージド) API が動作することをテストします。たとえば、Echo API の場合には **curl** コマンドを使用して以下の呼び出しを行うことができます。

```
curl "https://echo-api.3scale.net:443"
```

以下のレスポンスが返されます。

```
{
    "method": "GET",
    "path": "/",
    "args": "",
    "body": "",
    "headers": {
      "HTTP_VERSION": "HTTP/1.1",
      "HTTP_HOST": "echo-api.3scale.net",
      "HTTP_ACCEPT": "*/*",
      "HTTP_USER_AGENT": "curl/7.51.0",
      "HTTP_X_FORWARDED_FOR": "2.139.235.79, 10.0.103.58",
      "HTTP_X_FORWARDED_HOST": "echo-api.3scale.net",
      "HTTP_X_FORWARDED_PORT": "443",
      "HTTP_X_FORWARDED_PROTO": "https",
      "HTTP_FORWARDED": "for=10.0.103.58;host=echo-api.3scale.net;proto=https"
    },
    "uuid": "ee626b70-e928-4cb1-a1a4-348b8e361733"
  }
```

3.4.2. 認証の設定

API の認証設定は **[Your_product_name] > Integration > Settings** の **AUTHENTICATION** セクションで行うことができます。

表3.1 任意の認証フィールド

フィールド	説明
Auth user key	Credentials location に関連付けられたキーを設定します。
Credentials location	クレデンシャルが HTTP ヘッダー、クエリーパラメーター、または HTTP Basic 認証として渡されるかどうかを定義します。

フィールド	説明
Host Header	カスタムの Host リクエストヘッダーを定義します。これは、API バックエンドが特定のホストからのトラフィックのみを許可する場合に必要です。
Secret Token	API バックエンドに直接送られる開発者リクエストをブロックするために使用します。ここにヘッダーの値を設定し、バックエンドがこのシークレットヘッダーを持つ呼び出しのみを許可するようにします。

さらに **[Your_product_name] > Integration > Settings**の順に移動し、**GATEWAY RESPONSE** エラーコードを設定できます。エラーの **レスポンスコード**、**コンテンツタイプ**、 および **レスポンスボディー** を定義します。Authentication failed、Authentication missing、および No match。

表3.2 レスポンスコードとデフォルトのレスポンスボディー

レスポンスコード	レスポンスのボディー
403	Authentication failed
403	Authentication parameters missing
404	No Mapping Rule matched
429	Usage limit exceeded

3.4.3. API テストコールの設定

API の設定では、リクエストコールを元にテストを行うために、プロダクトを含めたバックエンドのテストを行い、APIcast の設定をステージング環境および実稼働環境にプロモートする必要があります。

それぞれのプロダクトについて、リクエストはパスに従って対応するバックエンドにリダイレクトされます。このパスは、バックエンドをプロダクトに追加する際に設定されます。たとえば、プロダクトに 2 つのバックエンドを追加している場合、それぞれのバックエンドは固有のパスを持ちます。

前提条件

- プロダクトに追加された 1 つまたは複数の バックエンド
- プロダクトに追加された各バックエンドの マッピングルール
- アクセスポリシーを定義するための アプリケーションプラン
- アプリケーションプランを参照する アプリケーション

手順

1. **[Your_product_name] > Integration > Configuration**の順に移動して、APIcast 設定をステージング環境にプロモートします。

2. **APIcast Configuration** セクションに、プロダクトに追加された各バックエンドのマッピングルールが表示されます。**Promote v.[n] to Staging APIcast**をクリックします。

   - **v.[n]** は、プロモート先のバージョン番号を表します。

3. ステージング環境にプロモートしたら、実稼働環境にプロモートすることができます。**Staging APIcast** セクションで、**Promote v.[n] to Production APIcast**をクリックします。

   - **v.[n]** は、プロモート先のバージョン番号を表します。

4. コマンドラインで API へのリクエストをテストするには、**Example curl for testing**で提供されるコマンドを使用します。

   - curl コマンドの例は、プロダクトの最初のマッピングルールに基づいています。

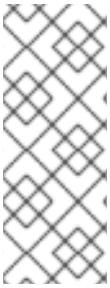
API へのリクエストをテストする際に、メソッドおよびメトリクスを追加して マッピングルールを変更することができます。

設定を変更したら、API への呼び出しを行う前に、必ずステージング環境および実稼働環境にプロモートするようにしてください。ステージング環境にプロモートする保留中の変更がある場合には、管理ポータルの **Integration** メニュー項目の横に感嘆符が表示されます。

3scale Hosted APIcast ゲートウェイはクレデンシャルを検証し、API のアプリケーションプランで定義した流量制御を適用します。クレデンシャルがない、あるいは無効なクレデンシャルで呼び出しを行うと、エラーメッセージ**Authentication failed**が表示されます。

3.5. DOCKER コンテナー環境への APICAST のデプロイ

ここでは、Docker コンテナーエンジン内部に Red Hat 3scale API Management API ゲートウェイとして使用する準備が整っている APIcast をデプロイする方法を、手順をおって説明します。



注記

Docker コンテナー環境に APIcast をデプロイする場合、サポートされる Red Hat Enterprise Linux (RHEL) および Docker のバージョンは以下のとおりです。

- RHEL 7.7
- Docker 1.13.1

前提条件

- 3章*APIcast のインストール* に従って、3scale 管理ポータルで APIcast を設定している。
- Red Hat Ecosystem Catalog へのアクセス
  - レジストリーサービスアカウントを作成するには、レジストリーサービスアカウントの作成 を参照してください。

Docker コンテナー環境に APIcast をデプロイするには、以下のセクションに概略を示す手順を実施します。

- 「Docker コンテナー環境のインストール」

- 「Docker コンテナー環境ゲートウェイの実行」

3.5.1. Docker コンテナー環境のインストール

本セクションでは、RHEL 7 に Docker コンテナー環境を設定する手順を説明します。

Red Hat が提供する Docker コンテナーエンジンは、RHEL の Extras チャンネルの一部としてリリースされています。追加のリポジトリーを有効にするには、Subscription Manager または **yum-config-manager** オプションを使用できます。詳細は、RHEL の製品ドキュメント を参照してください。

Amazon Web Services (AWS)、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスに RHEL 7 をデプロイするには、以下の手順を実施します。

手順

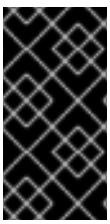
1. **sudo yum repolist all** ですべてのリポジトリーを一覧表示します。
2. ***-extras** リポジトリーを探します。
3. **sudo yum-config-manager --enable rhui-REGION-rhel-server-extras** を実行し、**extras** リポジトリーを有効にします。
4. **sudo yum install docker** を実行し、Docker コンテナー環境のパッケージをインストールします。

関連情報

他のオペレーティングシステムをお使いの場合は、以下の Docker ドキュメントを参照してください。

- Installing the Docker containerized environment on Linux distributions
- Installing the Docker containerized environment on Mac
- Installing the Docker containerized environment on Windows

3.5.2. Docker コンテナー環境ゲートウェイの実行



重要

3scale 2.11 では、RHEL7 および Docker のコンテナーとして実行されている APIcast デプロイメントのサポートは非推奨になりました。今後のリリースでは、3scale は RHEL8 および Podman のみをサポートします。Self-managed APIcast をコンテナーとして実行している場合は、インストールをサポート対象の設定にアップグレードしてください。

Docker コンテナー環境ゲートウェイを実行するには、以下の手順を実施します。

手順

1. Docker デーモンを開始します。

```
sudo systemctl start docker.service
```

2. Docker デーモンが実行されているか確認します。

```
sudo systemctl status docker.service
```

3. Red Hat レジストリーから、そのまま使用できる Docker コンテナーエンジンのイメージをダウンロードします。

```
sudo docker pull registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.12
```

4. Docker コンテナーエンジンで APIcast を実行します。

```
sudo docker run --name apicast --rm -p 8080:8080 -e
THREESCALE_PORTAL_ENDPOINT=https://<access_token>@<domain>-admin.3scale.net
registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.12
```

ここで、**<access_token>** は 3scale Account Management API のアクセストークンに置き換えます。アクセストークンの代わりにプロバイダーキーを使用することもできます。**<domain>-admin.3scale.net** は 3scale 管理ポータルの URL です。

このコマンドは、**apicast**という Docker コンテナーエンジンをポート **8080** で実行し、3scale 管理ポータルから JSON 設定ファイルを取得します。その他の設定オプションについては、APIcast のインストール を参照してください。

3.5.2.1. docker コマンドのオプション

docker run コマンドでは、以下のオプションを使用できます。

- **--rm**:終了時にコンテナーを自動的に削除します。
- **-d** または **--detach**:コンテナーをバックグラウンドで実行し、コンテナー ID を出力します。このオプションを指定しないと、コンテナーはフォアグラウンドモードで実行され、**CTRL+c** を使用して停止することができます。デタッチモードで起動された場合、**docker attach** コマンド (例: **docker attach apicast**) を使用するとコンテナーに再アタッチすることができます。
- **-p** または **--publish**:コンテナーのポートをホストに公開します。値の書式は **<host port="">:<container port="">** とする必要があります。したがって、**-p 80:8080** の場合は、コンテナーのポート **8080** をホストマシンのポート **80** にバインドします。たとえば、Management API はポート **8090** を使用するため、**-p 8090:8090** を **docker run** コマンドに追加してこのポートを公開します。
- **-e** または **--env**:環境変数を設定します。
- **-v** または **--volume**:ボリュームをマウントします。値は通常 **<host path="">:<container path="">[:<options>]** で表されます。**<options>** は任意の属性で、ボリュームを読み取り専用に指定するには、**:ro** に設定します (デフォルトでは読み取り/書き込みモードでマウントされます)。たとえば、**-v /host/path:/container/path:ro** と設定します。

3.5.2.2. APIcast のテスト

以下の手順は、Docker コンテナーエンジンが独自の設定ファイルと、3scale レジストリーからの Docker コンテナーイメージで実行されるようにします。呼び出しは APIcast を介してポート **8080** でテストでき、3scale アカウントから取得できる正しい認証クレデンシャルを提供できます。

テストコールは、APIcast が適切に実行されていることを確認するだけでなく、認証とレポートが正常に処理されたことも確認します。

注記

呼び出しに使用するホストが **Integration** ページの **Public Base URL** フィールドに設定されたホストと同じであるようにしてください。

関連情報

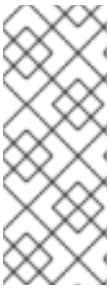
- 使用できるオプションの詳細については、Docker run reference を参照してください。

3.5.3. 関連情報

- テスト済みの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 を参照してください。

3.5.4. Podman への APIcast のデプロイ

ここでは、Pod Manager (Podman) コンテナー環境に Red Hat 3scale API Management API ゲートウェイとして使用される APIcast をデプロイする方法を、手順を追って説明します。



注記

Podman コンテナー環境に APIcast をデプロイする場合、サポートされる Red Hat Enterprise Linux (RHEL) および Podman のバージョンは以下のとおりです。

- RHEL 8
- Podman 1.4.2

前提条件

- 3章*APIcast のインストール* に従って、3scale 管理ポータルで APIcast を設定している。
- Red Hat Ecosystem Catalog へのアクセス
  - レジストリーサービスアカウントを作成するには、レジストリーサービスアカウントの作成 を参照してください。

Podman コンテナー環境に APIcast をデプロイするには、以下のセクションに概略を示す手順を実施します。

- 「Podman コンテナー環境のインストール」
- 「Podman 環境の実行」

3.5.4.1. Podman コンテナー環境のインストール

本セクションでは、RHEL 8 に Podman コンテナー環境を設定する手順を説明します。Docker は RHEL 8 に含まれていないため、コンテナーの操作には Podman を使用します。

Podman と RHEL 8 の使用については、コンテナーのコマンドに関するリファレンスドキュメント を参照してください。

手順

- Podman コンテナー環境パッケージをインストールします。

```
sudo dnf install podman
```

関連情報

他のオペレーティングシステムをお使いの場合は、以下の Podman のドキュメントを参照してください。

- Podman Installation Instructions

3.5.4.2. Podman 環境の実行

Podman コンテナー環境を実行するには、以下の手順に従います。

手順

1. Red Hat レジストリーから、そのまま使用できる Podman コンテナーのイメージをダウンロードします。

```
podman pull registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.12
```

2. Podman で APIcast を実行します。

```
podman run --name apicast --rm -p 8080:8080 -e
THREESCALE_PORTAL_ENDPOINT=https://<access_token>@<domain>-admin.3scale.net
registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.12
```

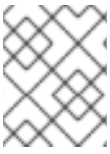
ここで、**<access_token>** は 3scale Account Management API のアクセストークンに置き換えます。アクセストークンの代わりにプロバイダーキーを使用することもできます。**<domain>-admin.3scale.net** は 3scale 管理ポータルの URL です。

このコマンドは、**apicast**という Podman コンテナーエンジンをポート **8080** で実行し、3scale 管理ポータルから JSON 設定ファイルを取得します。その他の設定オプションについては、APIcast のインストール を参照してください。

3.5.4.2.1. Podman による APIcast のテスト

以下の手順は、Podman コンテナーエンジンが独自の設定ファイルと、3scale レジストリーからの Podman コンテナーイメージで実行されるようにします。呼び出しは APIcast を介してポート **8080** でテストでき、3scale アカウントから取得できる正しい認証クレデンシャルを提供できます。

テストコールは、APIcast が適切に実行されていることを確認するだけでなく、認証とレポートが正常に処理されたことも確認します。



注記

呼び出しに使用するホストが **Integration** ページの **Public Base URL** フィールドに設定されたホストと同じであるようにしてください。

3.5.4.3. podman コマンドのオプション

podman コマンドでは、以下に例を示すオプションを使用することができます。

- **-d**:**デタッチモード** でコンテナーを実行し、コンテナー ID を出力します。このオプションを指定しないと、コンテナーはフォアグラウンドモードで実行され、**CTRL+c** を使用して停止することができます。デタッチモードで起動された場合、**podman attach** コマンド (例: **podman**

attach apicast) を使用するとコンテナーに再アタッチすることができます。

- **ps** および **-a**:podman **ps** を使用して、作成中および実行中のコンテナーを一覧表示します。**ps** コマンドに **-a** を追加すると (例: **podman ps -a**)、すべてのコンテナー (実行中および停止中の両方) が表示されます。

- **inspect** および **-l**:実行中のコンテナーを検査します。たとえば、**inspect** を使用して、コンテナーに割り当てられた ID を表示します。**-l** を使用すると、最新のコンテナーの詳細を取得できます (たとえば **podman inspect -l | grep Id\":**)。

3.5.4.4. 関連情報

- テスト済みの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 を参照してください。

- Podman を初めて使用する場合は、Basic Setup and Use of Podman を参照してください。

3.6. OPENSHIFT テンプレートを使用した APICAST のデプロイ

OpenShift テンプレートを使用して APIcast API ゲートウェイをデプロイすることができます。APIcast API ゲートウェイをデプロイすると、API の保護に役立ちます。また、その API に対するトラフィックを分析して監視することができます。

前提条件

- APIcast のインストール に従って、Red Hat 3scale API Management 管理ポータルで APIcast を設定する必要があります。

- インテグレーション設定でデプロイメントオプションに **Self-managed Gateway** が選択されていることを確認してください。

- 手順を進めるには、ステージング環境と実稼働環境の両方を設定している必要があります。

手順

1. デフォルトでは、**developer** としてログインしていて、次のステップに進むことができます。そうでなければ、前の手順でダウンロードおよびインストールした OpenShift クライアントツールから **oc login** コマンドを使用して OpenShift にログインします。デフォルトのログインクレデンシャルは **username = "developer"** と **password = "developer"** です。

```
oc login https://OPENSHIFT-SERVER-IP:8443
```

出力に **Login successful.** が表示されるはずです。

2. プロジェクトを作成します。この例では表示名を **gateway** と設定します。

```
oc new-project "3scalegateway" --display-name="gateway" --description="3scale gateway
demo"
```

応答は以下のようになります。

```
Now using project "3scalegateway" on server "https://172.30.0.112:8443"
```

コマンドプロンプトのテキスト出力で提案される次のステップを無視し、以下に示す次のステップに進みます。

3. プロジェクトを参照する新しいシークレットを作成します。**<access_token>** および **<domain>** はご自分のクレデンシャルに置き換えます。**<access_token>** および **<domain>** の詳細は、以下を参照してください。

```
oc create secret generic apicast-configuration-url-secret --from-
literal=password=https://<access_token>@<admin_portal_domain>  --
type=kubernetes.io/basic-auth
```

ここでは、**<access_token>** は 3scale アカウントの アクセストークン で、**<domain>-admin.3scale.net** は 3scale 管理ポータルの URL になります。

応答は以下のようになります。

```
secret/apicast-configuration-url-secret
```

4. テンプレートから APIcast ゲートウェイのアプリケーションを作成し、デプロイメントを開始します。

```
oc new-app -f https://raw.githubusercontent.com/3scale/3scale-amp-openshift-
templates/2.12.0.GA/apicast-gateway/apicast.yml
```

出力の最後に以下のメッセージが表示されるはずです。

```
--> Creating resources with label app=3scale-gateway ...
    deploymentconfig "apicast" created
    service "apicast" created
--> Success
    Run 'oc status' to view your app.
```

3.7. OPERATOR を使用した SELF-MANAGED APICAST ゲートウェイソリューションのデプロイ

本セクションでは、Openshift Container Platform コンソールから APIcast operator を使用して Self-managed APIcast ゲートウェイソリューションをデプロイする手順について説明します。

前提条件

- OpenShift Container Platform (OCP) 4 以降およびその管理者権限
- OpenShift への APIcast Operator のインストール の手順に従っている。

手順

1. 管理者権限を持つアカウントを使用して OCP コンソールにログインします。
2. **Operators > Installed Operators**の順にクリックします。
3. **Installed Operators** のリストで **APIcast Operator** をクリックします。
4. **APIcast > Create APIcast**の順にクリックします。

3.7.1. APIcast のデプロイメントおよび設定オプション

Self-managed APIcast ゲートウェイソリューションは、以下に示す 2 とおりの方法を使用してデプロイおよび設定することができます。

- 3scale システムエンドポイントの指定
- 設定シークレットの指定

以下も参照してください。

- APIcast Operator を使用したカスタム環境の注入
- APIcast operator を使用したカスタムポリシーの注入
- APIcast operator を使用した OpenTracing の設定

3.7.1.1. 3scale システムエンドポイントの指定

手順

1. 3scale システム管理ポータルのエンドポイント情報が含まれる OpenShift シークレットを作成します。

```
oc create secret generic ${SOME_SECRET_NAME} --from-literal=AdminPortalURL=${MY_3SCALE_URL}
```

   - **${SOME_SECRET_NAME}** はシークレットの名前で、既存のシークレットと競合しない限り、任意の名前を付けることができます。
   - **${MY_3SCALE_URL}** は、3scale アクセストークンおよび 3scale システム管理ポータルのエンドポイントが含まれる URI です。詳細は、**THREESCALE_PORTAL_ENDPOINT** を参照してください。

     例

```
oc create secret generic 3scaleportal --from-literal=AdminPortalURL=https://access-token@account-admin.3scale.net
```

     シークレットの内容についての詳細は、APIcast Custom Resource reference のEmbeddedConfSecret を参照してください。

2. APIcast の OpenShift オブジェクトを作成します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
spec:
  adminPortalCredentialsRef:
    name: SOME_SECRET_NAME
```

   spec.adminPortalCredentialsRef.name は、3scale システム管理ポータルのエンドポイント情報が含まれる既存の OpenShift シークレットの名前でなければなりません。

3. APIcast オブジェクトに関連付けられた OpenShift デプロイメントの **readyReplicas** フィールドが **1** であることを確認し、APIcast Pod が動作状態にあり準備が整っていることを確認します。そうでなければ、以下のコマンドを使用してフィールドが設定されるまで待ちます。

```
$ echo $(oc get deployment apicast-example-apicast -o jsonpath='{.status.readyReplicas}')
1
```

3.7.1.1.1. APIcast ゲートウェイが動作中で利用可能であることの確認

手順

1. ローカルマシンから OpenShift Service APIcast にアクセス可能であることを確認し、テストリクエストを実行します。そのために、APIcast OpenShift Service を **localhost:8080** にポート転送します。

```
oc port-forward svc/apicast-example-apicast 8080
```

2. 設定した 3scale サービスに対してリクエストを行い、HTTP 応答が正常であることを確認します。サービスの **Staging Public Base URL** または **Production Public Base URL** 設定で指定したドメイン名を使用します。以下に例を示します。

```
$ curl 127.0.0.1:8080/test -H "Host: myhost.com"
```

3.7.1.1.2. Kubernetes Ingress 経由での APIcast の外部公開

Kubernetes Ingress 経由で APIcast を外部に公開するには、**exposedHost** セクションを設定します。**ExposedHost** セクションの **host** フィールドを設定すると、Kubernetes Ingress オブジェクトが作成されます。事前にインストールした既存の Kubernetes Ingress Controller はこの Kubernetes Ingress オブジェクトを使用し、APIcast を外部からアクセス可能にします。

APIcast を外部からアクセス可能にするのに使用できる Ingress Controllers およびその設定方法について詳しく知るには、Kubernetes Ingress Controllers のドキュメント を参照してください。

ホスト名 **myhostname.com** で APIcast を公開する例を以下に示します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
spec:
  ...
  exposedHost:
    host: "myhostname.com"
  ...
```

この例では、HTTP 用ポート 80 に Kubernetes Ingress オブジェクトを作成します。APIcast デプロイメントが OpenShift 環境にある場合、OpenShift のデフォルト Ingress Controller は APIcast が作成する Ingress オブジェクト使用して Route オブジェクトを作成し、APIcast インストールへの外部アクセスが可能になります。

exposedHost セクションに TLS を設定することもできます。利用可能なフィールドの詳細を以下の表に示します。

表3.3 APIcastExposedHost の参照テーブル

json/yaml フィールド	タイプ	必須/任意	デフォルト値	説明
host	string	はい	該当なし	ゲートウェイにルーティングされているドメイン名
tls	[]extensions.IngressTLS	いいえ	該当なし	受信 TLS オブジェクトの配列。詳細は、TLS を参照してください。

3.7.1.2. 設定シークレットの指定

手順

1. 設定ファイルを使用してシークレットを作成します。

```
$ curl
https://raw.githubusercontent.com/3scale/APIcast/master/examples/configuration/echo.json -
o $PWD/config.json

oc create secret generic apicast-echo-api-conf-secret --from-file=$PWD/config.json
```

設定ファイルは **config.json** という名前にする必要があります。これは APIcast CRD の要件です。

シークレットの内容についての詳細は、APIcast Custom Resource reference のEmbeddedConfSecret を参照してください。

2. APIcast カスタムリソース を作成します。

```
$ cat my-echo-apicast.yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: my-echo-apicast
spec:
  exposedHost:
    host: YOUR DOMAIN
  embeddedConfigurationSecretRef:
    name: apicast-echo-api-conf-secret

$ oc apply -f my-echo-apicast.yaml
```

a. 埋め込み設定シークレットの例を以下に示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: SOME_SECRET_NAME
type: Opaque
stringData:
```

```
config.json: |
 {
  "services": [
   {
    "proxy": {
     "policy_chain": [
      { "name": "apicast.policy.upstream",
       "configuration": {
        "rules": [{
         "regex": "/",
         "url": "http://echo-api.3scale.net"
        }]
       }
      }
     ]
    }
   }
  ]
 }
```

3. APIcast オブジェクトの作成時に以下の内容を設定します。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
spec:
  embeddedConfigurationSecretRef:
    name: SOME_SECRET_NAME
```

spec.embeddedConfigurationSecretRef.name は、ゲートウェイの設定が含まれる既存の OpenShift シークレットの名前でなければなりません。

4. APIcast オブジェクトに関連付けられた OpenShift デプロイメントの **readyReplicas** フィールドが **1** であることを確認し、APIcast Pod が動作状態にあり準備が整っていることを確認します。そうでなければ、以下のコマンドを使用してフィールドが設定されるまで待ちます。

```
$ echo $(oc get deployment apicast-example-apicast -o jsonpath='{.status.readyReplicas}')
1
```

3.7.1.2.1. APIcast ゲートウェイが動作中で利用可能であることの確認

手順

1. ローカルマシンから OpenShift Service APIcast にアクセス可能であることを確認し、テストリクエストを実行します。そのために、APIcast OpenShift Service を **localhost:8080** にポート転送します。

```
oc port-forward svc/apicast-example-apicast 8080
```

2. 設定した 3scale サービスに対してリクエストを行い、HTTP 応答が正常であることを確認します。サービスの **Staging Public Base URL** または **Production Public Base URL** 設定で指定したドメイン名を使用します。以下に例を示します。

```
$  curl 127.0.0.1:8080/test -H "Host: localhost"
{
  "method": "GET",
  "path": "/test",
  "args": "",
  "body": "",
  "headers": {
    "HTTP_VERSION": "HTTP/1.1",
    "HTTP_HOST": "echo-api.3scale.net",
    "HTTP_ACCEPT": "*/*",
    "HTTP_USER_AGENT": "curl/7.65.3",
    "HTTP_X_REAL_IP": "127.0.0.1",
    "HTTP_X_FORWARDED_FOR": ...
    "HTTP_X_FORWARDED_HOST": "echo-api.3scale.net",
    "HTTP_X_FORWARDED_PORT": "80",
    "HTTP_X_FORWARDED_PROTO": "http",
    "HTTP_FORWARDED": "for=10.0.101.216;host=echo-api.3scale.net;proto=http"
  },
  "uuid": "603ba118-8f2e-4991-98c0-a9edd061f0f0"
```

3.7.1.3. APIcast Operator を使用したカスタム環境の注入

Self-managed APIcast を使用する 3scale インストールでは、**3scale** Operator を使用してカスタム環境を注入できます。カスタム環境は、ゲートウェイが提供するすべてのアップストリーム API に APIcast が適用する動作を定義します。カスタム環境を作成するには、Lua コードでグローバル設定を定義します。

APIcast のインストールの一部として、またはインストール後にカスタム環境を注入できます。カスタム環境を注入した後、その環境を削除すると、**APIcast** Operator が変更を調整します。

前提条件

- APIcast Operator がインストールされている。

手順

1. 注入するカスタム環境を定義する Lua コードを記述します。たとえば、次の **env1.lua** ファイルは、**APIcast** Operator がすべてのサービスに対してロードするカスタムログポリシーを示しています。

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

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

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

return {
```

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

2. カスタム環境を定義する Lua ファイルからシークレットを作成します。以下に例を示します。

```
oc create secret generic custom-env-1 --from-file=./env1.lua
```

シークレットには複数のカスタム環境を含めることができます。カスタム環境を定義する各ファイルの **-from-file** オプションを指定します。Operator は各カスタム環境をロードします。

3. 作成したシークレットを参照する **APIcast** カスタムリソースを定義します。以下の例は、カスタム環境を定義するシークレットの参照に関連するコンテンツのみを示しています。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  customEnvironments:
    - secretRef:
        name: custom-env-1
```

APIcast カスタムリソースは、カスタム環境を定義する複数のシークレットを参照できます。Operator は各カスタム環境をロードします。

4. カスタム環境を追加する **APIcast** カスタムリソースを作成します。たとえば、**APIcast** カスタムリソースを **apicast.yaml** ファイルに保存した場合は、以下のコマンドを実行します。

```
oc apply -f apicast.yaml
```

次のステップ

カスタム環境を更新する場合は、シークレットに更新が含まれるように、必ずそのシークレットを再作成します。**APIcast** operator は更新の有無を監視し、更新を発見すると自動的に再デプロイします。

3.7.1.4. APIcast operator を使用したカスタムポリシーの注入

Self-managed APIcast を使用する 3scale インストールでは、**APIcast** operator を使用してカスタムポリシーを注入できます。カスタムポリシーを注入すると、ポリシーコードが APIcast に追加されます。次に、以下のいずれかを使用して、カスタムポリシーを API 製品のポリシーチェーンに追加できます。

- 3scale API
- **Product** カスタムリソース

3scale 管理ポータルを使用してカスタムポリシーを製品のポリシーチェーンに追加するには、カスタムポリシーのスキーマを **CustomPolicyDefinition** カスタムリソースに登録する必要もあります。カスタムポリシー登録は、管理ポータルを使用して製品のポリシーチェーンを設定する場合にのみ必要です。

APIcast のインストールの一部として、またはインストール後にカスタムポリシーを注入できます。カスタムポリシーを注入した後、それを削除すると、**APIcast** operator が変更を調整します。

前提条件

- APIcast operator がインストールされているか、インストール中である。
- Write your own policy で説明されているように、カスタムポリシーを定義している。つまり、たとえば、カスタムポリシーを定義する**my-first-custom-policy.lua**、**apicast-policy.json**、および**init.lua**ファイルをすでに作成している。

手順

1. 1 つのカスタムポリシーを定義するファイルからシークレットを作成します。以下に例を示します。

```
oc create secret generic my-first-custom-policy-secret \
 --from-file=./apicast-policy.json \
 --from-file=./init.lua \
 --from-file=./my-first-custom-policy.lua
```

   複数のカスタムポリシーがある場合は、カスタムポリシーごとにシークレットを作成します。シークレットには、カスタムポリシーを 1 つだけ含めることができます。

2. 作成したシークレットを参照する **APIcast** カスタムリソースを定義します。以下の例は、カスタムポリシーを定義するシークレットの参照に関連するコンテンツのみを示しています。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  customPolicies:
    - name: my-first-custom-policy
      version: "0.1"
      secretRef:
        name: my-first-custom-policy-secret
```

   APIcast カスタムリソースは、カスタムポリシーを定義する複数のシークレットを参照できます。Operator は各カスタムポリシーをロードします。

3. カスタムポリシーを追加する**APIcast**カスタムリソースを作成します。たとえば、**APIcast** カスタムリソースを **apicast.yaml** ファイルに保存した場合は、以下のコマンドを実行します。

```
oc apply -f apicast.yaml
```

次のステップ

カスタムポリシーを更新する場合は、シークレットに更新が含まれるように、必ずそのシークレットを再作成します。**APIcast** operator は更新の有無を監視し、更新を発見すると自動的に再デプロイします。

関連情報

- APIcast custom resource definition

3.7.1.5. APIcast operator を使用した OpenTracing の設定

Self-managed APIcast を使用する 3scale のインストールでは、**APIcast** operator を使用して OpenTracing を設定できます。OpenTracing を有効にすると、APIcast インスタンスに関してより多くの洞察を得て、可観測性を向上できます。

前提条件

- **APIcast** operator がインストールされているか、インストール中である。
- OpenTracing を使用するための APIcast の設定 に記載の前提条件。
- Jaeger がインストールされている。

手順

1. **stringData.config** に OpenTracing 設定の詳細を含めて、シークレットを定義します。これは、OpenTracing 設定の詳細が含まれる属性の唯一有効な値です。その他の仕様では、APIcast が OpenTracing 設定の詳細を受け取れないようにします。以下の例は、有効なシークレット定義を示しています。

```
apiVersion: v1
kind: Secret
metadata:
  name: myjaeger
stringData:
  config: |-
      {
      "service_name": "apicast",
      "disabled": false,
      "sampler": {
        "type": "const",
        "param": 1
      },
      "reporter": {
        "queueSize": 100,
        "bufferFlushInterval": 10,
        "logSpans": false,
        "localAgentHostPort": "jaeger-all-in-one-inmemory-agent:6831"
      },
      "headers": {
        "jaegerDebugHeader": "debug-id",
        "jaegerBaggageHeader": "baggage",
        "TraceContextHeaderName": "uber-trace-id",
        "traceBaggageHeaderPrefix": "testctx-"
      },
      "baggage_restrictions": {
          "denyBaggageOnInitializationFailure": false,
          "hostPort": "127.0.0.1:5778",
          "refreshInterval": 60
      }
      }
type: Opaque
```

2. シークレットを作成します。たとえば、以前のシークレット定義を **myjaeger.yaml** ファイルに保存した場合は、以下のコマンドを実行します。

```
oc create secret generic myjaeger --from-file myjaeger.yaml
```

3. **OpenTracing** 属性を指定する **APIcast** カスタムリソースを定義します。CR 定義で、**spec.tracingConfigSecretRef.name** 属性を OpenTracing 設定の詳細が含まれるシークレットの名前に設定します。以下の例は、OpenTracing の設定に関するコンテンツのみを示しています。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  ...
  openTracing:
    enabled: true
    tracingConfigSecretRef:
      name: myjaeger
    tracingLibrary: jaeger
...
```

4. OpenTracing を設定する **APIcast** カスタムリソースを作成します。たとえば、**APIcast** カスタムリソースを **apicast1.yaml** ファイルに保存した場合は、以下のコマンドを実行するはずです。

```
oc apply -f apicast1.yaml
```

次のステップ

OpenTracing のインストール方法に応じて、Jaeger サービスユーザーインターフェイスでトレースが表示されるはずです。

関連情報

- APIcast custom resource definition

3.8. 関連情報

APIcast の最新リリースとサポート対象バージョンについては、以下のアーティクルを参照してください。

- Red Hat 3scale API Management のサポート対象設定
- Red Hat 3scale API Management コンポーネントの詳細

第4章 OPENSHIFT への 3SCALE OPERATOR のインストール

注記

3scale は、直近 2 つの OpenShift Container Platform (OCP) 一般提供 (GA) リリースをサポートします。詳細は、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。

本セクションでは、以下の項目の実施方法について説明します。

- 新しいプロジェクトを作成する。
- Red Hat 3scale API Management インスタンスをデプロイする。
- Operator Lifecycle Manager (OLM) を使用して 3scale operator をインストールする。
- operator をデプロイした後にカスタムリソースをデプロイする。

前提条件

- 管理者権限を持つアカウントを使用して、サポート対象バージョンの OpenShift Container Platform 4 クラスターにアクセスできる。
  - 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。

警告

3scale operator とカスタムリソース定義 (CRD) は、新たに作成した空の **プロジェクト** にデプロイしてください。インフラストラクチャーが含まれる既存のプロジェクトにデプロイすると、既存の要素が変更または削除されることがあります。

OpenShift に 3scale operator をインストールするには、以下のセクションに概略を示す手順を実施します。

- 「新しい OpenShift プロジェクトの作成」
- 「OLM を使用した 3scale operator のインストールと設定」

4.1. 新しい OPENSHIFT プロジェクトの作成

以下の手順で、**3scale-project** という新しい OpenShift プロジェクトを作成する方法について説明します。このプロジェクト名を実際のプロジェクト名に置き換えてください。

手順

新しい OpenShift プロジェクトを作成するには、以下の手順を実施します。

- 英数字とダッシュを使用して、有効な名前を指定します。たとえば、以下のコマンドを実行して **3scale-project** を作成します。

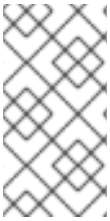
```
oc new-project 3scale-project
```

これにより、operator、**APIManager** カスタムリソース (CR)、および **Capabilities** カスタムリソースがインストールされる新しい **OpenShift プロジェクト** が作成されます。operator は、そのプロジェクトの OLM を通じてカスタムリソースを管理します。

4.2. OLM を使用した 3SCALE OPERATOR のインストールと設定

Operator Lifecycle Manager (OLM) を使用して、OpenShift Container Platform (OCP) 4.6 クラスターに 3scale operator をインストールします。この際に、OCP コンソールの OperatorHub を使用します。以下のインストールモードを使用して、3scale operator をインストールできます。

- クラスター全体。Operator がクラスターのすべての namespace で利用できます。
- クラスター上の特定の namespace。



注記

ネットワークが制限された環境または非接続クラスターで OpenShift Container Platform を使用している場合には、Operator Lifecycle Manager は OperatorHub を使用できなくなります。OLM の設定および使用については、Operator のネットワークが制限された環境での Operator Lifecycle Manager の使用 に記載の手順に従ってください。

前提条件

- 新しい OpenShift プロジェクトの作成 で定義したプロジェクトに 3scale operator をインストールおよびデプロイしている。

手順

1. OpenShift Container Platform コンソールにおいて、管理者権限を持つアカウントを使用してログインします。

注記

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。

2. **Operators > OperatorHub** の順にクリックします。
3. **Filter by keyword** ボックスに **3scale operator** と入力し、3scale operator を検索します。
4. 3scale operator をクリックします。operator に関する情報が表示されます。
5. operator に関する情報を確認し、**Install** をクリックします。**Install Operator** ページが開きます。
6. **Install Operator** ページで、任意のチャネルを選択して、**Update channel** セクションを更新します。
7. **Installation mode** セクションで、Operator のインストール先を選択します。
   a. **All namespaces on the cluster(default)**: Operator はクラスターのすべての namespace で利用可能になります。

b. **A specific namespace on the cluster** operator は、選択したクラスター上の特定の単一 namespace でしか使用することができません。

8. **Subscribe** をクリックします。**3scale operator** の詳細ページが表示され、**Subscription Overview** を確認できます。

9. サブスクリプションの **Upgrade Status** が **Up to date** と表示されていることを確認します。

10. 3scale operator の ClusterServiceVersion (CSV) が表示され、新しい OpenShift プロジェクトの作成 で定義したプロジェクトで operator の **Status** が最終的に **InstallSucceeded** となるのを確認します。

    - **Operators > Installed Operators**の順にクリックします。この場合、インストールに成功すると、**APIManager** CRD および operator の **Capabilities** 機能に関連する CRD が **OpenShift API サーバー** に登録されます。

11. インストールが正常に完了したら、**oc get** を使用して CRD によって定義されたリソースタイプのクエリーを行います。

    a. たとえば、**APIManager** CRD が適切に登録されたことを確認するには、以下のコマンドを実行します。

    ```
    oc get apimanagers
    ```

12. 以下の出力が表示されるはずです。

    ```
    No resources found.
    ```

ネットワークが制限された環境で OCP を使用する場合、ここに示す手順に加えて、3scale デベロッパーポータルで使用する許可されるドメインのリストを作成します。以下の例を参照してください。

- デベロッパーポータルに追加するすべてのリンク
- GitHub などのサードパーティー SSO プロバイダーを使用した SSO インテグレーション
- 請求
- 外部 URL をトリガーする Webhook

4.2.1. ネットワーク接続が得られない環境における制約

3scale 2.12 の非接続環境での現在の制限の概要を以下に示します。

- デベロッパーポータルへの GitHub ログインができない
- サポートのリンクが機能しない
- 外部ドキュメントへのリンクが機能しない
- デベロッパーポータルの OpenAPI Specification (OAS) 検証ツールが機能しない (これにより、外部サービスへのリンクが影響を受けます)
- **ActiveDocs** の製品 **Overview** ページにおいて、OAS へのリンクが機能しない
    - 新たな ActiveDocs 仕様を作成する場合、オプション **Skip swagger validations** を選択する必要もあります。

関連情報

- トラブルシューティングに関する情報は、OpenShift Container Platform のドキュメント を参照してください。
- ネットワークが制限された環境での OLM の使用に関する詳細は、Operator のネットワークが制限された環境での Operator Lifecycle Manager の使用 を参照してください。
- ネットワークが制限された環境でのインストール準備の詳細は、インストール のネットワークが制限された環境でのインストール用のミラーレジストリーの作成 を参照してください。
- 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。

4.3. OLM を使用した 3SCALE OPERATOR のアップグレード

Operator ベースのデプロイメントで、1 つの namespace から全 namespace のクラスター全体のインストールに、3scale Operator をアップグレードするには、その namespace から 3scale Operator を削除してから、クラスター上にその Operator を再インストールする必要があります。

クラスター管理者は Web コンソールを使用して、選択した namespace からインストールされた Operator を削除できます。Operator をアンインストールしても、既存の 3scale インスタンスはアンインストールされません。

3scale Operator を namespace からアンインストールしたら、OLM を使用してクラスター全体のモードで Operator をインストールできます。

前提条件

- namespace の削除パーミッションがある 3scale 管理者権限または OpenShift ロール。

手順

1. OpenShift Container Platform コンソールにおいて、管理者権限を持つアカウントを使用してログインします。

注記

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。

2. **Operators** > **OperatorHub** の順にクリックします。インストール済みの Operator ページが表示されます。
3. **3scale** を **Filter by name** に入力して Operator を見つけ、クリックします。
4. **Operator Details** ページで、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択して、特定の namespace から削除します。
5. **Uninstall Operator?** ダイアログボックスが表示され、以下が通知されます。

> Removing the operator will not remove any of its custom resource definitions or managed resources. If your operator has deployed applications on the cluster or configured off-cluster resources, these will continue to run and need to be cleaned up manually.
> This action removes the operator as well as the Operator deployments and pods, if any. Any operands and resources managed by the operator, including CRDs and CRs, are not

> removed. The web console enables dashboards and navigation items for some operators. To remove these after uninstalling the operator, you might need to manually delete the operator CRDs.

6. **Uninstall** を選択します。この Operator は実行を停止し、更新を受信しなくなります。
7. OpenShift Container Platform コンソールで、**Operators > OperatorHub** をクリックします。
8. **Filter by keyword** ボックスに **3scale operator** と入力し、3scale operator を検索します。
9. 3scale operator をクリックします。operator に関する情報が表示されます。
10. **Install** をクリックします。**Install Operator** ページが開きます。
11. **Install Operator** ページで、任意のチャネルを選択して、**Update channel** セクションを更新します。
12. **Installation mode** セクションで、**All namespaces on the cluster(default)** を選択します。Operator はクラスターのすべての namespace で利用可能になります。
13. **Subscribe** をクリックします。**3scale operator** の詳細ページが表示され、**Subscription Overview** を確認できます。
14. サブスクリプションの **Upgrade Status** が **Up to date** と表示されていることを確認します。
15. 3scale operator の ClusterServiceVersion (CSV) が表示されることを確認します。

関連情報

- 3scale operator のインストールに関する詳細は、3scale API Management のインストールおよび設定 を参照してください。

第5章 APICAST OPERATOR の OPENSHIFT へのインストール

本セクションでは、OpenShift Container Platform (OCP) コンソールから APIcast operator をインストールする手順について説明します。

手順

1. 管理者権限を持つアカウントを使用して OCP コンソールにログインします。
2. **Projects > Create Project**の順に移動し、新規プロジェクト **operator-test** を作成します。
3. **Operators > Installed Operators**の順にクリックします。
4. **Filter by keyword** ボックスに **apicast** と入力し、APIcast operator を検索します。コミュニティーバージョンは使用しないでください。
5. APIcast operator をクリックします。APIcast operator に関する情報が表示されます。
6. **Install** をクリックします。**Create Operator Subscription** ページが表示されます。
7. **Create Operator Subscription** ページで、すべてのデフォルト設定を受け入れ **Subscribe** をクリックします。
    a. サブスクリプションのアップグレードステータスが **Up to date** と表示されます。
8. **Operators > Installed Operators**の順にクリックし、**operator-test** プロジェクトで APIcast operator の **ClusterServiceVersion** (CSV) ステータスが最終的に **InstallSucceeded** と表示されることを確認します。

第6章 3SCALE 高可用性テンプレートおよび評価用テンプレート

ここでは、Red Hat 3scale API Management 2.12 インストール環境で使用される 高可用性 テンプレートおよび 評価用 テンプレートについて説明します。

前提条件

- 高可用性テンプレートおよび評価用テンプレートの要素をデプロイできる OpenShift クラスターが用意されている。

重要

3scale の高可用性テンプレートおよび評価用テンプレートは、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。

高可用性テンプレートおよび評価用テンプレートをデプロイするには、以下のセクションに概略を示す手順を実施します。

- 「高可用性テンプレート」
- 「評価用テンプレート」

6.1. 高可用性テンプレート

高可用性 (HA) テンプレートを使用すると、重要なデータベースの HA を設定できます。

前提条件

- HA テンプレートをデプロイする前に、外部データベースをデプロイおよび設定し、負荷分散されたエンドポイントで HA を設定しておく。

HA テンプレートの使用

高可用性のために **amp-ha-tech-preview.yml** という名前のテンプレートを使用すると、OpenShift 外部に重要なデータベースをデプロイできます。ただし、以下は除外されます。

- Memcached
- Sphinx
- Zync

標準の **amp.yml** テンプレートと **amp-ha-tech-preview.yml** には、以下の違いがあります。

- 以下の要素が削除されています。
  - backend-redis およびその関連コンポーネント
  - system-redis およびその関連コンポーネント

- system-mysql およびその関連コンポーネント
- Redis および MySQL 関連の ConfigMaps
- MYSQL_IMAGE、REDIS_IMAGE、MYSQL_USER、MYSQL_ROOT_PASSWORD パラメーター

- デフォルトで、データベースではない **DeploymentConfig** オブジェクトタイプのレプリカの数が 1 から 2 に増加されます。
- 以下の必須パラメーターが追加されているため、外部データベースの場所を制御できます。
  - BACKEND_REDIS_STORAGE_ENDPOINT
  - BACKEND_REDIS_QUEUES_ENDPOINT
  - SYSTEM_REDIS_URL
  - APICAST_STAGING_REDIS_URL
  - APICAST_PRODUCTION_REDIS_URL
  - SYSTEM_DATABASE_URL

amp-ha-tech-preview.yml を使用する場合、新たに追加された必須パラメーターによりクラスター外のデータベース接続を設定する必要があります (ただし、永続的なデータが含まれない **system-memcache**、**zync-database**、および **system-sphinx** は除外)。エンドポイントには、クレデンシャルを含む、データベースの負荷分散用接続文字列が必要です。また、データベースではないデプロイメントについては、アプリケーションレベルでの冗長性を確保するためにデフォルトで Pod レプリカの数が 2 に増えています。

6.1.1. 高可用性向け RWX_STORAGE_CLASS の設定

ReadWriteMany (RWX) PersistentVolumeClaim (PVC) はストレージクラス RWX_STORAGE_CLASS を使用します。

必須: false

値: **null**

- これを **null** (値なし) に設定した場合、ストレージクラスが自動検出されることが OpenShift に通知されます。
- これを空の文字列 (特定の値または null 以外) に設定すると、文字列ストレージを空にすることが OpenShift に通知されます。これは無効な設定です。

6.2. 評価用テンプレート

評価の目的で、リソースのリクエストや制限のない 3scale 環境をデプロイする **amp-eval-tech-preview.yml** という名前のテンプレートが提供されています。

標準の **amp.yml** テンプレートとの唯一の機能的な違いは、リソースの制限とリクエストが削除されたことです。そのため、このバージョンでは CPU およびメモリーレベルでハードウェアの最低要件が Pod で削除されました。このテンプレートは、指定のハードウェアリソースを使用して、可能な限りコンポーネントをデプロイしようとするため、評価、テスト、および開発のみの使用を目的としています。

第7章 3SCALE の REDIS 高可用性 (HA) サポート

重要

Red Hat は、ゼロダウンタイムのための Redis のセットアップ、3scale のバックエンドコンポーネントの設定、または Redis データベースのレプリケーションおよびシャーディングを正式にはサポートしていません。本章に記載の内容は、参照用途としてのみ提供されています。また、3scale では、**cluster mode** の Redis はサポートされません。

高可用性 (HA) は、OpenShift Container Platform (OCP) によりほとんどのコンポーネントで提供されます。詳細は、OpenShift Container Platform 3.11 30.章、高可用性 を参照してください。

Red Hat 3scale API Management の HA 用データベースコンポーネントは、以下のとおりです。

- **backend-redis**: 統計ストレージおよび一時ジョブストレージに使用されます。
- **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 と **system-redis** は、どちらもサポートされる Redis Sentinel および Redis Enterprise 用 Redis 高可用性バリアントと共に動作します。

Redis Pod が停止した場合や、OpenShift Container Platform によって停止された場合には、新しい Pod が自動作成されます。データは永続ストレージから復元されるので、Pod は動作し続けます。このような場合、新しい Pod が起動するまでの間、短いダウンタイムが発生します。これは、Redis がマルチマスター設定をサポートしないという制限によるものです。Redis をデプロイしたすべてのノードに Redis イメージを事前にインストールすることで、ダウンタイムを削減することができます。これにより、Pod の再起動にかかる時間が短縮されます。

ゼロダウンタイムとなるよう Redis をセットアップし、3scale のバックエンドコンポーネントを設定します。

- ゼロダウンタイムのための Redis 設定
- 3scale 用バックエンドコンポーネントの設定
- Redis データベースのシャーディングおよびレプリケーション

前提条件

- 管理者ロールが設定された 3scale アカウント

7.1. ゼロダウンタイムのための REDIS 設定

ゼロダウンタイムが必要な場合、3scale 管理者は OCP 外部に Redis を設定します。3scale Pod の設定オプションを使用してこの設定を行うには、いくつかの方法があります。

- 独自の自己管理型 Redis を設定する
- Redis Sentinel を使用する (Redis Sentinel ドキュメントの参照)
- サービスとして提供される Redis
  例:
  - Amazon ElastiCache

- Redis Labs

注記

Red Hat は上記のサービスにサポートを提供しません。このようなサービスの言及は、Red Hat による製品やサービスの推奨を意味するものではありません。Red Hat は、Red Hat 外部のコンテンツを使用 (または依存) して発生した損失や費用の責任を負いません。

7.2. 3SCALE 用バックエンドコンポーネントの設定

3scale 管理者は、**backend-cron**、**backend-listener**、および **backend-worker** のデプロイメント設定で、**back-end** コンポーネントの環境変数に Redis HA (フェイルオーバー) を設定します。3scale の Redis HA には、これらの設定が必要です。

注記

Redis と Sentinel を使用するには、**backend-redis**あるいは**system-redis**、またはその両方のシークレットに Sentinel 設定を指定する必要があります。

7.2.1. backend-redis と system-redis シークレットの作成

以下の手順に従い、適宜 **backend-redis** および **system-redis** シークレットを作成します。

- HA 用 3scale の新規インストールのデプロイ
- 3scale の非 HA デプロイメントの HA への移行

7.2.2. HA 用 3scale の新規インストールのデプロイ

手順

1. 以下のフィールドを指定して、**backend-redis** および **system-redis** シークレットを作成します。

backend-redis

```
REDIS_QUEUES_SENTINEL_HOSTS
REDIS_QUEUES_SENTINEL_ROLE
REDIS_QUEUES_URL
REDIS_STORAGE_SENTINEL_HOSTS
REDIS_STORAGE_SENTINEL_ROLE
REDIS_STORAGE_URL
```

system-redis

```
NAMESPACE
SENTINEL_HOSTS
SENTINEL_ROLE
URL
```

- Redis と Sentinel を設定する場合、**backend-redis** および **system-redis** の該当する **URL**

フィールドには、**redis://[:redis-password@]redis-group[/db]** のフォーマットで Redis グループを指定します。ここで、[x] はオプションの要素 **x** を意味し、**redis-password**、**redis-group**、および **db** 変数は適切な値に置き換えてください。

例

```
redis://:redispwd@mymaster/5
```

- **SENTINEL_HOSTS** フィールドは、以下のフォーマットの Sentinel 接続文字列のコンマ区切りリストです。

```
redis://:sentinel-password@sentinel-hostname-or-ip:port
```

  - リスト内の各要素に関して、[x] はオプションの要素 **x** を意味し、**sentinel-password**、**sentinel-hostname-or-ip**、および **port** 変数は適切な値に置き換えてください。

    例

```
:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722
```

- **SENTINEL_ROLE** フィールドの値は、**master** または **slave** のどちらかです。

2. operator を使用した 3scale のデプロイ に記載の手順に従って 3scale を デプロイします。

   a. **backend-redis** および **system-redis** がすでに存在するために表示されるエラーは無視します。

7.2.3. 3scale の非 HA デプロイメントの HA への移行

1. HA 用 3scale の新規インストールのデプロイ に記載のとおり、すべてのフィールドを指定して **backend-redis** および **system-redis** シークレットを編集します。

2. 以下の **backend-redis** 環境変数がバックエンド Pod に対して定義されていることを確認してください。

```
name: BACKEND_REDIS_SENTINEL_HOSTS
  valueFrom:
    secretKeyRef:
      key: REDIS_STORAGE_SENTINEL_HOSTS
      name: backend-redis
name: BACKEND_REDIS_SENTINEL_ROLE
  valueFrom:
    secretKeyRef:
      key: REDIS_STORAGE_SENTINEL_ROLE
      name: backend-redis
```

3. 以下の **system-redis** の環境変数が **system-(app|sidekiq|sphinx)** Pod に対して定義されていることを確認してください。

```
name: REDIS_SENTINEL_HOSTS
  valueFrom:
    secretKeyRef:
      key: SENTINEL_HOSTS
```

```
      name: system-redis
name: REDIS_SENTINEL_ROLE
  valueFrom:
    secretKeyRef:
      key: SENTINEL_ROLE
      name: system-redis
```

4. 指示に従って、テンプレートを使用した 3scale のアップグレード を続行します。

7.2.3.1. Redis Enterprise の使用

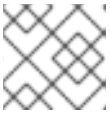
1. 3 つの異なる **redis-enterprise** インスタンスで、OpenShift にデプロイされた Redis Enterprise を使用します。
   a. **system-redis** シークレットを編集します。
      i. **system-redis** のシステム redis データベースを **URL** に設定します。
   b. **backend-redis** のバックエンドデータベースを **REDIS_QUEUES_URL** に設定します。
   c. **backend-redis** の 3 番目のデータベースを **REDIS_STORAGE_URL** に設定します。

7.2.3.2. Redis Sentinel の使用

注記

オプションで、Redis Sentinel をデータベースのいずれかに適用できます。ただし、Red Hat は、HA 用に Redis Sentinel をすべてに適用することを推奨します。

1. 統計のバックエンド redis: **backend-redis** シークレットを更新し、以下の値を指定します。
   - **REDIS_STORAGE_URL**
   - **REDIS_STORAGE_SENTINEL_ROLE**
   - **REDIS_STORAGE_SENTINEL_HOSTS**
     REDIS_STORAGE_SENTINEL_ROLE を Sentinels ホストおよびポートのコンマ区切りリストに設定します (例:**:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722**)。
2. キューのバックエンド redis: **backend-redis** シークレットを更新し、以下の値を指定します。
   - **REDIS_QUEUES_URL**
   - **REDIS_QUEUES_SENTINEL_ROLE**
   - **REDIS_QUEUES_SENTINEL_HOSTS**
     REDIS_QUEUES_SENTINEL_ROLE を Sentinels ホストおよびポートのコンマ区切りリストに設定します (例: **:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722**)。
3. これらの Redis データベースと共に Redis Sentinel を使用します。
4. データ用のシステム redis: **system-redis** シークレットを更新し、以下の値を指定します。



注記

system-redis シークレットを編集します。**URL**

- **SENTINEL_ROLE**
- **NAMESPACE**
- **URL**
- **SENTINEL_HOSTS**
  SENTINEL_HOSTS を Sentinels ホストおよびポートのコンマ区切りリストに設定します (例: **:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722**)。

注記

- system-app および system-sidekiq コンポーネントは、統計を取得するために **back-end** Redis に直接接続します。
  - 3scale 2.7 の時点で、これらのシステムコンポーネントは Sentinel を使用する際にも **back-end** Redis (ストレージ) に接続することができます。
- system-app および system-sidekiq コンポーネントは、**backend-redis** ストレージ **しか使用しません** (**backend-redis** キューは使用しません)。
  - システムコンポーネントに加えた変更は、**backend-redis** ストレージと Sentinel の組み合わせをサポートします。

7.3. REDIS データベースのシャーディングおよびレプリケーション

シャーディング (パーティショニングとも呼ばれる) とは、大規模なデータベースをシャードと呼ばれる小規模なデータベースに分割することを指します。レプリケーションでは、別のマシンでホストされる同じデータベースのコピーでデータベースがセットアップされます。

シャーディング

シャーディングにより、多くのリーダーインスタンスを容易に追加することができます。また、1 つのデータベースには収まらない非常に多くのデータがある場合や、CPU 負荷が 100% に近い場合にも役立ちます。

3scale の Redis HA では、以下の 2 つの理由によりシャーディングが重要となります。

- 大量のデータを分割およびスケーリングし、特定の指標に合わせてシャード数を調整することで、ボトルネックの回避に役立つ。
- 異なるノードに処理を分散させることで、パフォーマンスが向上する (例: 複数のマシンが同じクエリーで動作している場合)。

クラスターモードが無効な Redis データベースシャーディング用の 3 つの主要なソリューションを以下に示します。

- Amazon ElastiCache
- Redis sentinel による標準 Redis
- Redis Enterprise

レプリケーション

Redis データベースのレプリケーションにより、データセットを異なるマシンに複製して冗長性を確保します。レプリケーションを使用すると、リーダーがダウンした場合に Redis の動作を維持することができます。その後、データは 1 つのインスタンス (リーダー) からプルされ、高可用性が確保されます。

3scale の Redis HA を使用すると、データベースのレプリケーションにより、プライマリーシャードの高可用性レプリカが確保されます。基本的な動作は以下のとおりです。

- プライマリーシャードに障害が発生すると、レプリカシャードが自動的に新しいプライマリーシャードにプロモートされる。
- 元のプライマリーシャードが復旧すると、自動的に新しいプライマリーシャードのレプリカシャードになる。

Redis データベースレプリケーション用の 3 つの主要なソリューションを以下に示します。

- Redis Enterprise
- Amazon ElastiCache
- Redis sentinel による標準 Redis

twemproxyを使用したシャーディング

Amazon ElastiCache および標準 Redis のシャーディングでは、キーに基づいてデータを分割します。特定のキーを与えられると探すシャードを認識するプロキシーコンポーネントが必要です (例: **twemproxy**)。nutcracker としても知られる **twemproxy** は Redis プロトコル用の軽量プロキシーソリューションで、割り当てられた特定のキーまたはサーバーのマッピングに基づいてシャードを検索します。**twemproxy** により Amazon ElastiCache または標準 Redis インスタンスにシャーディング機能を追加すると、以下のメリットが得られます。

- データを自動的に複数のサーバーにシャーディングすることができる。
- 複数のハッシュモードをサポートし、一貫性のあるハッシュおよび分散できる。
- 複数のインスタンスで実行することができ、これによりクライアントは利用可能な最初のプロキシーサーバーに接続することができる。
- バックエンドのキャッシュサーバーへの接続数を減らすことができる。

注記

Redis Enterprise は独自のプロキシーを使用するため、**twemproxy** は必要ありません。

関連情報

- Redis Sentinel Documentation
- **twemproxy**

7.4. 関連情報

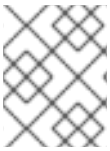
- 3scale と Redis データベースのサポートについては、Red Hat 3scale API Management のサポート対象設定 を参照してください。

- Redis 向け Amazon ElastiCache の詳細は、公式のAmazon ElastiCache Documentation を参照してください。
- Redis Enterprise の詳細は、最新の ドキュメント を参照してください。

第8章 外部 MYSQL データベースの設定

本章では、*6章3scale 高可用性テンプレートおよび評価用テンプレート* の MySQL データベースを外部化する方法について説明します。そのためには、デフォルトの **amp.yml** ファイルを使用します。これは、デフォルトの **system-mysql** Pod を使用するとネットワークやファイルシステムなど複数のインフラストラクチャーの問題が生じる場合に役立ちます。

本章のアプローチと *6章3scale 高可用性テンプレートおよび評価用テンプレート* のアプローチの違いは、本アプローチでは、Red Hat 3scale API Management が最初にデフォルトの **amp.yml** テンプレートを使用していた場合に、MySQL データベースを外部化することができる点です。



注記

Red Hat は外部 MySQL データベースを使用する 3scale の設定をサポートしています。ただし、データベース自体はサポートの範囲外です。

前提条件

- 管理者権限を持つアカウントを使用して OpenShift Container Platform 3.11 クラスターにアクセスできること。
- OpenShift クラスター上にインストールされた 3scale インスタンス。*2章OpenShift への 3scale のインストール* を参照してください。

High Availability (HA) 用に外部 MySQL データベースを設定するには、以下のセクションに概略を示す手順を実施します。

- 「外部 MySQL データベースに関する制約」
- 「MySQL データベースの外部化」
- 「ロールバック」

8.1. 外部 MYSQL データベースに関する制約

MySQL データベースを外部化するプロセスの制約は以下のとおりです。

オンプレミス型 3scale のバージョン

オンプレミス型 3scale のバージョン 2.5 および 2.6 のみテストおよび検証済みです。

MySQL データベースユーザー

URL は以下の形式でなければなりません。

<database_scheme>://<admin_user>:<admin_password>@<database_host>/<database_name>

<admin_user> は、**<database_name>** 論理データベースの完全なパーミッションを持つ外部データベースの既存ユーザーである必要があります。**<database_name>** は、外部データベースの既存の論理データベースである必要があります。

MySQL ホスト

ホスト名 ではなく外部 MySQL データベースの **IP アドレス** を使用します。そうでない場合は、解決されません。たとえば、**mysql.mydomain.com** ではなく **1.1.1.1** を使用します。

8.2. MYSQL データベースの外部化

MySQL データベースを完全に外部化するには、以下の手順を使用します。

警告

この操作により、プロセスの進行中に環境でダウンタイムが発生します。

手順

1. オンプレミス型 3scale インスタンスをホストする OpenShift ノードにログインし、そのプロジェクトに切り替えます。

```
oc login -u <user> <url>
oc project <3scale-project>
```

<user>、**<url>**、および **<3scale-project>** を、実際のクレデンシャルとプロジェクト名に置き換えます。

2. 以下に示す順序で手順実施し、すべての Pod をスケールダウンします。これにより、データの喪失が回避されます。

オンプレミス型 3scale の停止

OpenShift Web コンソールまたはコマンドラインインターフェイス (CLI) から、すべてのデプロイメント設定を以下の順序でゼロレプリカにスケールダウンします。

- 3scale 2.6 より前のバージョンの場合は **apicast-wildcard-router** と **zync**、3scale 2.6 以降の場合は **zync-que** と **zync**
- **apicast-staging** と **apicast-production**
- **system-sidekiq**、**backend-cron**、および **system-sphinx**
  - 3scale 2.3 の場合には **system-resque** を対象に含めます。
- **system-app**
- **backend-listener** と **backend-worker**
- **backend-redis**、**system-memcache**、**system-mysql**、**system-redis**、および **zync-database**
  以下の例は、**apicast-wildcard-router** と **zync** について、CLI でこの操作を実施する方法を示しています。

```
oc scale dc/apicast-wildcard-router --replicas=0
oc scale dc/zync --replicas=0
```

注記

各ステップのデプロイメント設定は同時にスケールダウンできます。たとえば、**apicast-wildcard-router** と **zync** を一緒にスケールダウンできます。ただし、各ステップの Pod が終了するのを待ってから、次の Pod をスケールダウンすることをお勧めします。3scale インスタンスは、完全に再起動されるまで一切アクセスできなくなります。

3. 3scale プロジェクトで実行中の Pod がないことを確認するには、以下のコマンドを使用します。

```
oc get pod
```

このコマンドは、**No resources found** を返すはずです。

4. 以下のコマンドを使用して、データベースレベルの Pod を再度スケールアップします。

```
oc scale dc/{backend-redis,system-memcache,system-mysql,system-redis,zync-database} --replicas=1
```

5. 次のステップに進む前に、**system-mysql** Pod を通じて外部 MySQL データベースにログインできることを確認してください。

```
oc rsh system-mysql-<system_mysql_pod_id>
mysql -u root -p -h <host>
```

- **<system_mysql_pod_id>**:system-mysql Pod の識別子。
- ユーザーには必ず root を使用する。詳しくは、外部 MySQL データベースに関する制約 を参照してください。

    a. CLI に **mysql>** が表示されるようになります。**exit** と入力してから **enter** キーを押します。次のプロンプトで再度 **exit** と入力して、OpenShift ノードのコンソールに戻ります。

6. 以下のコマンドを使用して、MySQL のフルダンプを実行します。

```
oc rsh system-mysql-<system_mysql_pod_id> /bin/bash -c "mysqldump -u root --single-transaction --routines --triggers --all-databases" > system-mysql-dump.sql
```

- **<system_mysql_pod_id>** を一意の **system-mysql** Pod **ID** に置き換えます。
- 次の例のように、ファイル **system-mysql-dump.sql** に有効な MySQL レベルのダンプが含まれていることを確認します。

```
$ head -n 10 system-mysql-dump.sql
-- MySQL dump 10.13  Distrib 8.0, for Linux (x86_64)
--
-- Host: localhost    Database:
-- ------------------------------------------------------
-- Server version   8.0

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
```

```
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
*/;
/*!40101 SET NAMES utf8 */;
```

7. **system-mysql** Pod をスケールダウンし、レプリカが 0 (ゼロ) のままにします。

```
oc scale dc/system-mysql --replicas=0
```

8. **<password>** と **<host>** を適宜置き換え、URL **mysql2://root:<password>@<host>/system** の **base64** 変換値を取得します。

```
echo "mysql2://root:<password>@<host>/system" | base64
```

9. リモート MySQL データベースのデフォルトの **'user'**@'%' を作成します。これには SELECT 権限しか付与する必要はありません。また、その **base64** 変換値を取得します。

```
echo "user" | base64
echo "<password>" | base64
```

- <password> を **'user'**@'%' のパスワードに置き換えます。

10. バックアップを実行し、OpenShift シークレット **system-database** を編集します。

```
oc get secret system-database -o yaml > system-database-orig.bkp.yml
oc edit secret system-database
```

- **URL**:これを [step-8] で取得した値に置き換えます。
- **DB_USER** および **DB_PASSWORD**:いずれの場合も、前の手順の値を使用します。

11. **system-mysql-dump.sql** をリモートデータベースサーバーに送信し、ダンプをインポートします。インポートには、以下のコマンドを使用します。

12. 以下のコマンドを使用して **system-mysql-dump.sql** をリモートデータベースサーバーに送信し、ダンプをサーバーにインポートします。

```
mysql -u root -p < system-mysql-dump.sql
```

13. **system** という新しいデータベースが作成されたことを確認します。

```
mysql -u root -p -se "SHOW DATABASES"
```

14. 以下の手順を使用して、**オンプレミス型 3scale を起動します**。これにより、すべての Pod が正しい順序でスケールアップされます。

オンプレミス型 3scale の起動

- **backend-redis**、**system-memcache**、**system-mysql**、**system-redis**、および **zync-database**
- **backend-listener** と **backend-worker**
- **system-app**

- **system-sidekiq**、**backend-cron**、および **system-sphinx**
  - 3scale 2.3 の場合には **system-resque** を対象に含めます。
- **apicast-staging** と **apicast-production**
- 3scale 2.6 より前のバージョンの場合は **apicast-wildcard-router** と **zync**、3scale 2.6 以降の場合は **zync-que** と **zync**
  以下の例は、**backend-redis**、**system-memcache**、**system-mysql**、**system-redis**、および **zync-database** について、CLI でこの操作を実行する方法を示しています。

```
oc scale dc/backend-redis --replicas=1
oc scale dc/system-memcache --replicas=1
oc scale dc/system-mysql --replicas=1
oc scale dc/system-redis --replicas=1
oc scale dc/zync-database --replicas=1
```

system-app Pod が問題なく起動し、実行されるはずです。

15. 確認後、上記の順序 で他の Pod をスケールアップして元の状態に戻します。
16. **system-mysql** DeploymentConfig オブジェクトのバックアップを作成します。数日後、すべて正常に動作していることが確認できたら、削除してかまいません。**system-mysql** DeploymentConfig を削除することで、この手順を今後再び実行する場合の混乱を防ぐことができます。

8.3. ロールバック

ステップ 14 を実施した後、**system-app** Pod が完全には動作状態に戻らず、その根本的な原因が判断できない、または対処できない場合、ロールバックの手順を実施します。

1. **system-database-orig.bkp.yml** の元の値を使用して、シークレット **system-database** を編集します。[step-10] を参照してください。

```
oc edit secret system-database
```

URL、**DB_USER**、および **DB_PASSWORD** を元の値に置き換えます。

2. **system-mysql** を含め、すべての Pod をスケールダウンしてから、再度スケールアップして元の状態に戻します。**system-app** Pod およびその後に起動されるその他の Pod が、再び起動して実行されるはずです。以下のコマンドを実行して、すべての Pod が元どおりに起動、実行されていることを確認します。

```
oc get pods -n <3scale-project>
```

8.4. 関連情報

- 3scale と MySQL データベースのサポートについては、Red Hat 3scale API Management のサポート対象設定 を参照してください。