Red Hat 3scale API Management 2.8

3scale の移行

3scale API Management インストールのアップグレード

Last Updated: 2023-07-06

Red Hat 3scale API Management 2.8 3scale の移行

3scale API Management インストールのアップグレード

法律上の通知

概要

本ガイドでは、3scale API Management インストールを最新バージョンにアップグレードするための情報を提供します。

目次

はじめに

このガイドは、Red Hat 3scale API Management の移行およびアップグレードに役立ちます。

3scale インストールを 2.7 から 2.8 にアップグレードするには、インストールの種類に応じてガイドが 2 つあります。

- テンプレートベースの 3scale のアップグレードガイド
- operator ベースの 3scale アップグレードガイド

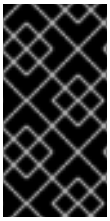
開発者ポータルで API をプロビジョニングするためのアップグレード後の手順

- 3scale のアップグレードが正常に完了したら、開発者ポータルで OpenAPI 仕様 3.0 (OAS 3.0) を設定するには、OAS 3.0 を使用した開発者ポータルの設定 を参照してください。

テンプレートベースのデプロイメントから Operator ベースのデプロイメントに移行するには 3章*3scale API Management の移行ガイド: テンプレートベースから operator ベースのデプロイメントへ* に記載されている手順に従います。

第1章 3SCALE テンプレートベースのアップグレードガイド: 2.7 から 2.8 へ

このセクションでは、テンプレートベースのデプロイメントで Red Hat 3scale API Management をバージョン 2.7 から 2.8 にアップグレードする方法について説明します。



重要

必要な条件および手順を理解するために、記載の手順を適用する前に、アップグレードガイド全体を読んでください。アップグレードプロセスの手順が完了するまで、サービスの提供が中断されます。このサービス中断が生じるため、メンテナンス期間を設けるようにしてください。

1.1. アップグレードの準備

この章では、3scale をアップグレードする前に満たすべき条件について説明します。また、アップグレードに必要な前提条件として必要なタスクとツールも示します。

1.1.1. アップグレードの条件

アップグレードに進む前に、次の点を考慮する必要があります。

- 3scale は、OpenShift 3.11 上のテンプレートを使用して、2.7 から 2.8 へのアップグレードパスをサポートします。
- OpenShift CLI ツールが 3scale をデプロイしたプロジェクトに設定されるようにする。

1.1.2. アップグレードを行うための前提条件

このセクションでは、テンプレートベースのインストールで 3scale を 2.7 から 2.8 にアップグレードするために必要なタスクとツールについて説明します。

主要なタスク

- 3scale で使用しているデータベースのバックアップを行う。バックアップの手順は、データベースのタイプや設定により異なります。

ツール

アップグレードを行うには、以下のツールが必要です。

- OpenShift 3.11 プロジェクトにテンプレートを使用してデプロイされた 3scale 2.7。
- Bash シェル: アップグレード手順で詳細を説明するコマンドを実行します。
- base64: シークレットの情報をエンコードおよびデコードします。
- jq: JSON 変換用です。

1.2. テンプレートベースのインストールでの 2.7 から 2.8 へのアップグレード

このセクションで説明されている手順に従って、テンプレートベースのインストールで 3scale 2.7 を 2.8 にアップグレードします。

アップグレードを開始するには、3scale がデプロイされているプロジェクトに移動します。

```
$ oc project <3scale-project>
```

続いて、以下の順序で手順を実行します。

1. 「3scale プロジェクトのバックアップの作成」
2. 「**smtp** ConfigMap の **system-smtp** secret への移行」
3. 「**system-app** DeploymentConfig の **pre-hook pod** コマンドの更新」
4. 「**system-app** DeploymentConfig の **pre-hook pod** 環境へのパッチ適用」
5. 「**system-app** DeploymentConfig コンテナーの環境へのパッチ適用」
6. 「**system-sidekiq** DeploymentConfig コンテナーの環境へのパッチ適用」
7. 「S3 固有の設定の移行」
8. 「3scale のバージョン番号の更新」
9. 「3scale イメージのアップグレード」
10. 「**smtp** ConfigMap の削除」

1.2.1. 3scale プロジェクトのバックアップの作成

前の手順

なし

現在の手順

3scale プロジェクトのバックアップを作成するのに必要なアクションを以下のリストに示します。

1. 3scale で使用するデータベースに応じて、${SYSTEM_DB} を以下のいずれかの値に設定します。
   - データベースが MySQL の場合: **SYSTEM_DB=system-mysql**
   - データベースが PostgreSQL の場合: **SYSTEM_DB=system-postgresql**
2. 既存の DeploymentConfigs でバックアップファイルを作成します。

```
$ THREESCALE_DC_NAMES="apicast-production apicast-staging backend-cron backend-listener backend-redis backend-worker system-app system-memcache ${SYSTEM_DB} system-redis system-sidekiq system-sphinx zync zync-database zync-que"

for component in ${THREESCALE_DC_NAMES}; do oc get --export -o yaml dc ${component} > ${component}_dc.yml ; done
```

3. **export all** コマンドでエクスポートされるプロジェクト内のすべての既存 OpenShift リソースをバックアップします。

```
$ oc get -o yaml --export all > threescale-project-elements.yaml
```

4. **export all** コマンドでエクスポートされない追加の要素でバックアップファイルを作成します。

```
$ for object in rolebindings serviceaccounts secrets imagestreamtags cm
rolebindingrestrictions limitranges resourcequotas pvc templates cronjobs statefulsets hpa
deployments replicasets poddisruptionbudget endpoints
do
  oc get -o yaml --export $object > $object.yaml
done
```

5. 生成されたすべてのファイルが空ではないこと、およびそれらすべての内容が予想どおりであることを確認します。

次の手順

「**smtp** ConfigMap の **system-smtp** secret への移行」

1.2.2. smtp ConfigMap の system-smtp secret への移行

前の手順

「3scale プロジェクトのバックアップの作成」

現在の手順

このステップの目標は、**システム** の SMTP 設定を ConfigMap から Secret に移行することです。SMTP 設定には機密情報が含まれているため、この移行にはメール関連の部分が含まれます。この情報の保護には、シークレットは ConfigMap よりも安全です。

1. **app** ラベルの現在の値を収集します。

```
$ DEPLOYED_APP_LABEL=$(oc get dc backend-listener -o json | jq
.spec.template.metadata.labels.app -r)
```

- 次のコマンドを実行して、DEPLOYED_APP_LABEL が空でないことを確認できます。

```
$ echo  ${DEPLOYED_APP_LABEL}
```

2. **smtp** ConfigMap の現在の内容を収集します。

```
$ CFGMAP_DATA_CONTENTS=$(oc get configmap smtp -o json | jq -r .data)
```

- 次のコマンドを実行して、CFGMAP_DATA_CONTENTS が空でないことを確認できます。

```
$ echo  ${CFGMAP_DATA_CONTENTS}
```

- 次のコマンドを実行して、CFGMAP_DATA_CONTENTS の値を確認できます。

```
$ oc get configmap smtp -o json | jq -r .data
```

3. **smtp** ConfigMap の内容を使用して **system-smtp** シークレットを作成します。

```
$ cat <<EOF | oc create -f -
{
  "apiVersion": "v1",
  "kind": "Secret",
  "metadata": {
    "creationTimestamp": null,
    "labels": {
      "app": "${DEPLOYED_APP_LABEL}",
      "threescale_component": "system",
      "threescale_component_element": "smtp"
    },
    "name": "system-smtp"
  },
  "stringData": ${CFGMAP_DATA_CONTENTS}
}
EOF
```

4. 次のコマンドを実行して、**system-smtp** シークレットが作成されたことを確認します。

```
$ oc get secret system-smtp -o yaml
```

すべてのデータキーと関連する値が、**system-smtp** シークレットと **smtp** ConfigMap の両方で同じであることを確認します。**system-smtp** シークレットのデータ値は base64 でエンコードされているため、実際の値を確認するにはデコードする必要があります。たとえば、シークレットデータ内のキーの名前が **mykey** の場合に、そのキーに関連付けられている値をコピーし、次のコマンドでデコードして実際の値を確認できます。

```
$ oc get secret system-smtp -o json | jq -r .data.mykey | base64 -d
```

キーに関連付けられた値が空の文字列の場合には、前のコマンドの結果には出力がありません。

次のステップ

「**system-app** DeploymentConfig の **pre-hook pod** コマンドの更新」

1.2.3. system-app DeploymentConfig の pre-hook pod コマンドの更新

前の手順

「**smtp** ConfigMap の **system-smtp** secret への移行」

現在の手順

3scale から最新の機能を取得するために、このステップでは、**system-app** DeploymentConfig で **pre-hook pod** コマンドを更新する方法について説明します。

1. **system-app** DeploymentConfig 内で、pre-hook pod コマンドをこのリリースに必要な新しいコマンドに更新します。

```
oc patch dc/system-app -p '{"spec":{"strategy":{"rollingParams":{"pre":{"execNewPod":
{"command":["bash","-c","bundle exec rake boot openshift:deploy"]}}}}}}'
```

2. **pre-hook pod** コマンドが新しい値に変更されたことを確認します。

```
oc get dc system-app -o json | jq .spec.strategy.rollingParams.pre.execNewPod.command
```

- 上記のコマンドの結果は以下のようになります。

```
[
  "bash",
  "-c",
  "bundle exec rake boot openshift:deploy"
]
```

次のステップ

「**system-app** DeploymentConfig の **pre-hook pod** 環境へのパッチ適用」

1.2.4. system-app DeploymentConfig の pre-hook pod 環境へのパッチ適用

前の手順

「**system-app** DeploymentConfig の **pre-hook pod** コマンドの更新」

現在の手順

このステップでは、**pre-hook pod** 環境の **system-app** DeploymentConfig に環境変数を追加します。このように追加することで、SMTP 関連の環境変数が 新しく作成された **system-smtp** secret を参照するようになります。また、環境変数の追加で **pre-hook pod** コマンドの変更 に関連する変数が正しく設定されるようになります。

1. **system-app** DeploymentConfig で **pre-hook Pod** 環境変数にパッチを適用します。

```
oc get dc system-app -o json | jq 'del(.spec.strategy.rollingParams.pre.execNewPod.env[] |
select(.name == "SMTP_ADDRESS" // .name == "SMTP_USER_NAME" // .name ==
"SMTP_PASSWORD" // .name == "SMTP_DOMAIN" // .name == "SMTP_PORT" // .name
== "SMTP_AUTHENTICATION" // .name == "SMTP_OPENSSL_VERIFY_MODE")) |
.spec.strategy.rollingParams.pre.execNewPod.env +=
[{"name":"SMTP_ADDRESS","valueFrom":{"secretKeyRef":{"key":"address","name":"system-
smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"secretKeyRef":
{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":
{"secretKeyRef":{"key":"password","name":"system-smtp"}}},
{"name":"SMTP_DOMAIN","valueFrom":{"secretKeyRef":{"key":"domain","name":"system-
smtp"}}},{"name":"SMTP_PORT","valueFrom":{"secretKeyRef":{"key":"port","name":"system-
smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"secretKeyRef":
{"key":"authentication","name":"system-smtp"}}},
{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"secretKeyRef":
{"key":"openssl.verify.mode","name":"system-smtp"}}},
{"name":"MASTER_ACCESS_TOKEN","valueFrom":{"secretKeyRef":
{"key":"MASTER_ACCESS_TOKEN","name":"system-seed"}}}]' | oc apply -f -
```

2. 次のアクションポイントに従って、**pre-hook pod** 環境にパッチが適用されていることを確認します。

   a. **system-app** pre-hook Pod で MASTER_ACCESS_TOKEN がシークレット参照として設定されていることを確認します。

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env |
map(select(.name == "MASTER_ACCESS_TOKEN")) | length'
```

想定される出力: **1**

- MASTER_ACCESS_TOKEN が **システムシード** シークレットを指すように正しく設定されていることを確認できます。

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env
| map(select(.name == "MASTER_ACCESS_TOKEN"))'
```

想定される出力:

```
[
  {
    "name": "MASTER_ACCESS_TOKEN",
    "valueFrom": {
        "secretKeyRef": {
            "key": "MASTER_ACCESS_TOKEN",
            "name": "system-seed"
        }
    }
  }
]
```

b. すべての SMTP_* 環境変数が **system-app** pre-hook Pod でシークレット参照として設定されていることを確認します。

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env |
map(select(.name | contains("SMTP")))'
```

- 以下の出力リストの各環境変数は、**system-smtp** シークレットキーへの参照である必要があります。
  - SMTP_ADDRESS
  - SMTP_USER_NAME
  - SMTP_PASSWORD
  - SMTP_DOMAIN
  - SMTP_PORT
  - SMTP_AUTHENTICATION
  - SMTP_OPENSSL_VERIFY_MODE

次のステップ

「**system-app** DeploymentConfig コンテナーの環境へのパッチ適用」

1.2.5. system-app DeploymentConfig コンテナーの環境へのパッチ適用

前の手順

「**system-app** DeploymentConfig の **pre-hook pod** 環境へのパッチ適用」

現在の手順

この手順では、環境変数を **system-app** コンテナー環境に追加および変更します。SMTP 関連の環境変数が、新しく作成された **system-smtp** シークレットを参照するすようにします。

1. **system-app** DeploymentConfig のコンテナー環境変数にパッチを適用します。

```
oc patch dc/system-app -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-master","env":[{"name":"SMTP_ADDRESS","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"authentication","name":"system-smtp"}}},{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-smtp"}}}]},{"name":"system-provider","env":[{"name":"SMTP_ADDRESS","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"authentication","name":"system-smtp"}}},{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-smtp"}}}]},{"name":"system-developer","env":[{"name":"SMTP_ADDRESS","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"authentication","name":"system-smtp"}}},{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-smtp"}}}]}]}}}}'
```

2. すべての SMTP_* 環境変数が、次のリストにある **system-app** コンテナーでシークレット参照として設定されていることを確認します。

- **system-developer**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name == "system-developer"))[].env | map(select(.name | contains("SMTP")))'
```

- **system-provider**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-provider"))[].env | map(select(.name | contains("SMTP")))'
```

- **system-master**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-master"))[].env | map(select(.name | contains("SMTP")))'
```

これらのコンテナーでは、以下の出力リストの環境変数は、**system-smtp** シークレットキーへの参照である必要があります。

- SMTP_ADDRESS
- SMTP_USER_NAME
- SMTP_PASSWORD
- SMTP_DOMAIN
- SMTP_PORT
- SMTP_AUTHENTICATION
- SMTP_OPENSSL_VERIFY_MODE

次のステップ

「**system-sidekiq** DeploymentConfig コンテナーの環境へのパッチ適用」

1.2.6. system-sidekiq DeploymentConfig コンテナーの環境へのパッチ適用

前の手順

「**system-app** DeploymentConfig コンテナーの環境へのパッチ適用」

現在の手順

この手順では、環境変数を **system-sidekiq** Pod 環境に追加および変更します。ここに記載されている手順を実行すると、SMTP 関連の環境変数が新しく作成された **system-smtp** シークレットを参照するようになります。

1. **system-sidekiq** DeploymentConfig の環境変数にパッチを適用します。

```
oc patch dc/system-sidekiq -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-
sidekiq","env":[{"name":"SMTP_ADDRESS","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},
{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},
{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},
{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"authentication","name":"system-smtp"}}},
```

```
{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-
smtp"}}}]}}}}}'
```

2. すべての SMTP_* 環境変数がシークレット参照として設定されていることを確認します。

```
oc get dc system-sidekiq -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-sidekiq"))[].env | map(select(.name | contains("SMTP")))'
```

以下の出力リストの各環境変数は、**system-smtp** シークレットキーへの参照である必要があります。

- SMTP_ADDRESS
- SMTP_USER_NAME
- SMTP_PASSWORD
- SMTP_DOMAIN
- SMTP_PORT
- SMTP_AUTHENTICATION
- SMTP_OPENSSL_VERIFY_MODE

次のステップ

- **amp-s3** テンプレート(「**smtp** ConfigMap の **system-smtp** secret への移行」)を使用して、Amazon Simple Storage Service (Amazon S3)を使用して 3scale 2.7 をデプロイした場合。
- **amp-s3** テンプレートを 3scale 2.7 にインストールした場合: 「3scale のバージョン番号の更新」

1.2.7. S3 固有の設定の移行

注記

amp-s3 テンプレートを 3scale 2.7 にインストールした場合は、この手順の指示に従います。それ以外の場合は、「3scale のバージョン番号の更新」 の手順でアップグレードを続行します。

前の手順

「**system-sidekiq** DeploymentConfig コンテナーの環境へのパッチ適用」

現在の手順

このステップでは、S3 に固有の設定を **システム環境** の ConfigMap から **aws-auth** シークレットに移行するためのタスクを示します。

1. 値を既存の **aws-auth** シークレットに追加します。

```
oc patch secret aws-auth --patch "{\"stringData\": $(oc get configmap system-environment -o
json | jq '.data | {"AWS_BUCKET": .AWS_BUCKET, "AWS_REGION": .AWS_REGION } ')}"
```

- キーとその値が **aws-auth** シークレットに追加されていることを確認します。これらの値は base64 でエンコードされています。

```
oc get secret aws-auth -o yaml
```

2. **system-app** DeploymentConfig から pre-hook Pod 環境変数にパッチを適用します。

```
oc get dc system-app -o json | jq 'del(.spec.strategy.rollingParams.pre.execNewPod.env[] |
select(.name == "AWS_BUCKET" // .name == "AWS_REGION")) |
.spec.strategy.rollingParams.pre.execNewPod.env +=
[{"name":"AWS_BUCKET","valueFrom":{"secretKeyRef":
{"key":"AWS_BUCKET","name":"aws-auth"}}},{"name":"AWS_REGION","valueFrom":
{"secretKeyRef":{"key":"AWS_REGION","name":"aws-auth"}}},
{"name":"AWS_PROTOCOL","valueFrom":{"secretKeyRef":
{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]' | oc apply -f -
```

- すべての AWS_* 環境変数が、**system-app** pre-hook pod でシークレット参照として設定されていることを確認します。

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env |
map(select(.name | contains("AWS")))'
```

- 以下の出力リストの各環境変数は、**aws-auth** シークレットキーへの参照である必要があります。
  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_BUCKET
  - AWS_REGION
  - AWS_PROTOCOL
  - AWS_HOSTNAME
  - AWS_PATH_STYLE

3. **system-app** DeploymentConfig のコンテナー環境変数にパッチを適用します。

```
oc patch dc/system-app -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-
master","env":[{"name":"AWS_BUCKET","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},
{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":
{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]},{"name":"system-
provider","env":[{"name":"AWS_BUCKET","valueFrom":
```

```
{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},
{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":
{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]},{"name":"system-
developer","env":[{"name":"AWS_BUCKET","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},
{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":
{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]}}}}}'
```

system-app の 3 つのコンテナーで、すべての AWS_* 環境変数がシークレット参照として設定されていることを確認します。

- **system-developer**:

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-developer"))[].env | map(select(.name | contains("AWS")))'
```

- **system-master**:

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-master"))[].env | map(select(.name | contains("AWS")))'
```

- **system-provider**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-provider"))[].env | map(select(.name | contains("AWS")))'
```

3 つのコンテナーすべてについて、以下の出力リストの各環境変数は、**aws-auth** シークレットキーへの参照である必要があります。

  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_BUCKET
  - AWS_REGION
  - AWS_PROTOCOL
  - AWS_HOSTNAME
  - AWS_PATH_STYLE

4. **system-sidekiq** DeploymentConfig のコンテナー環境変数にパッチを適用します。

```
oc patch dc/system-sidekiq -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-sidekiq","env":[{"name":"AWS_BUCKET","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]}}}}}'
```

- すべての AWS_* 環境変数がシークレット参照として設定されていることを確認します。

  ```
  oc get dc system-sidekiq -o json | jq '.spec.template.spec.containers | map(select(.name == "system-sidekiq"))[].env | map(select(.name | contains("AWS")))'
  ```

  以下の出力リストの各環境変数は、**aws-auth** シークレットキーへの参照である必要があります。

  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_BUCKET
  - AWS_REGION
  - AWS_PROTOCOL
  - AWS_HOSTNAME
  - AWS_PATH_STYLE

5. 未使用の **system-environment** ConfigMap キーを削除します。

```
oc patch configmap system-environment --patch '{"data": {"AWS_BUCKET": null, "AWS_REGION": null}}'
```

次のステップ

「3scale のバージョン番号の更新」

1.2.8. 3scale のバージョン番号の更新

前の手順

- **amp-s3** テンプレートを 3scale 2.7 にインストールした場合: 「**smtp** ConfigMap の **system-smtp** secret への移行」
- **amp-s3** テンプレートを 3scale 2.7 にインストールした場合: 「**system-sidekiq** DeploymentConfig コンテナーの環境へのパッチ適用」

現在の手順

この手順では、**system-environment** ConfigMap の 3scale のリリースバージョン番号を **2.7** から **2.8** に更新します。AMP_RELEASE は、一部の DeploymentConfig コンテナー環境で参照される ConfigMap のエントリーです。

1. AMP_RELEASE にパッチを適用するには、以下のコマンドを実行します。

```
oc patch cm system-environment --patch '{"data": {"AMP_RELEASE": "2.8"}}'
```

2. system-environment ConfigMap の AMP_RELEASE キーの値が **2.8** であることを確認します。

```
oc get cm system-environment -o json | jq .data.AMP_RELEASE
```

次のステップ

「3scale イメージのアップグレード」

1.2.9. 3scale イメージのアップグレード

前の手順

「3scale のバージョン番号の更新」

現在の手順

この手順では、アップグレードプロセスに必要な 3scale イメージを更新します。

1. **amp-system** イメージストリームにパッチを適用します。
amp-system イメージストリームにパッチを適用するには、3scale デプロイで使用されるデータベースを考慮する必要があります。

   - 3scale が Oracle Database を使用してデプロイされている場合には、ステップ 1、2、4、8 および 9 を実施して Oracle Database でシステムイメージをビルド します。
   - データベースが Oracle DB と異なる場合は、次のコマンドを使用します。

```
oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP system 2.8"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/system-rhel7:3scale2.8"}, "name": "2.8", "referencePolicy": {"type": "Source"}}}]'
oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP system (latest)"}, "from": { "kind": "ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

   これにより、**system-app**、**system-sphinx**、および **system-sidekiq** DeploymentConfig の再デプロイがトリガーされます。再デプロイが完了し、対応する新規 Pod が使用できる状態になり、古い Pod が終了するまで待ちます。

2. **amp-apicast** イメージストリームにパッチを適用します。

```
oc patch imagestream/amp-apicast --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP APIcast 2.8"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.8"}, "name": "2.8", "referencePolicy": {"type": "Source"}}}]'
```

```
oc patch imagestream/amp-apicast --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "AMP APIcast (latest)"}, "from": { "kind":
"ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

これにより、**apicast-production** および **apicast-staging** DeploymentConfig の再デプロイがトリガーされます。再デプロイが完了し、対応する新規 Pod が使用できる状態になり、古い Pod が終了するまで待ちます。

3. **amp-backend** イメージストリームにパッチを適用します。

```
oc patch imagestream/amp-backend --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "AMP Backend 2.8"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/backend-rhel7:3scale2.8"}, "name":
"2.8", "referencePolicy": {"type": "Source"}}}]'
oc patch imagestream/amp-backend --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "AMP Backend (latest)"}, "from": {
"kind": "ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type":
"Source"}}}]'
```

これにより、**backend-listener**、**backend-worker**、および **backend-cron** DeploymentConfig の再デプロイがトリガーされます。再デプロイが完了し、対応する新規 Pod が使用できる状態になり、古い Pod が終了するまで待ちます。

4. **amp-zync** イメージストリームにパッチを適用します。

```
oc patch imagestream/amp-zync --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "AMP Zync 2.8"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/zync-rhel7:3scale2.8"}, "name":
"2.8", "referencePolicy": {"type": "Source"}}}]'
oc patch imagestream/amp-zync --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "AMP Zync (latest)"}, "from": { "kind":
"ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

これにより、**zync** および **zync-que** DeploymentConfig の再デプロイがトリガーされます。再デプロイが完了し、対応する新規 Pod が使用できる状態になり、古い Pod が終了するまで待ちます。

5. **system-memcached** ImageStream にパッチを適用します。

```
oc patch imagestream/system-memcached --type=json -p '[{"op": "add", "path": "/spec/tags/-
", "value": {"annotations": {"openshift.io/display-name": "System 2.8 Memcached"}, "from": {
"kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/memcached-
rhel7:3scale2.8"}, "name": "2.8", "referencePolicy": {"type": "Source"}}}]'
oc patch imagestream/system-memcached --type=json -p '[{"op": "add", "path": "/spec/tags/-
", "value": {"annotations": {"openshift.io/display-name": "System Memcached (latest)"}, "from":
{ "kind": "ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type":
"Source"}}}]'
```

これにより、**system-memcache** DeploymentConfig の再デプロイがトリガーされます。再デプロイが完了し、対応する新規 Pod が使用できる状態になり、古い Pod が終了するまで待ちます。

6. **zync-database-postgresql** イメージストリームにパッチを適用します。

```
oc patch imagestream/zync-database-postgresql --type=json -p '[{"op": "add", "path":
```

```
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "Zync 2.8 PostgreSQL"},
"from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"},
"name": "2.8", "referencePolicy": {"type": "Source"}}}]'
oc patch imagestream/zync-database-postgresql --type=json -p '[{"op": "add", "path":
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "Zync PostgreSQL
(latest)"}, "from": { "kind": "ImageStreamTag", "name": "2.8"}, "name": "latest",
"referencePolicy": {"type": "Source"}}}]'
```

- このパッチコマンドにより **zync-database-postgresql** イメージストリームが更新され、2.8 タグが含まれるようになります。次のコマンドを実行して、2.8 タグが作成されたことを確認できます。

  ```
  oc get is/zync-database-postgresql
  ```

  次に、**tags** 列に **2.8** タグが表示されていることを確認します。

- このパッチは、イメージに新しい更新がある場合に、**zync-database** DeploymentConfig の再デプロイをトリガーすることもあります。その場合は、新規 Pod の再デプロイが完了して使用できる状態になり、古い Pod が終了するまで待ちます。

7. 3scale 2.6 のインストール環境に以下の DeploymentConfig の 1 つまたは複数が存在する場合は、該当する DeploymentConfig のリンクをクリックして詳細な操作手順を確認してください。
   - 「**backend-redis** DeploymentConfig」
   - 「**system-redis** DeploymentConfig」
   - 「**system-mysql** DeploymentConfig」
   - 「**system-postgresql** DeploymentConfig」
8. DeploymentConfig のすべてのイメージ URL に、各 URL アドレスの最後に追加されたハッシュと共に新しいイメージレジストリーの URL が含まれていることを確認します。

```
$ THREESCALE_DC_NAMES="apicast-production apicast-staging backend-cron backend-
listener backend-redis backend-worker system-app system-memcache system-mysql system-
redis system-sidekiq system-sphinx zync zync-database zync-que"

for component in ${THREESCALE_DC_NAMES}; do echo -n "${component} image: " && oc
get dc $component -o json | jq .spec.template.spec.containers[0].image ; done
```

次のステップ

「**smtp** ConfigMap の削除」

1.2.9.1. 既存の DeploymentConfig に関する追加手順

1.2.9.1.1. backend-redis DeploymentConfig

現在の 3scale インストール環境に **backend-redis** DeploymentConfig が存在する場合は、**backend-redis** イメージストリームにパッチを適用します。

```
oc patch imagestream/backend-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "Backend 2.8 Redis"}, "from": { "kind": "DockerImage",
```

```
"name": "registry.redhat.io/rhscl/redis-32-rhel7:3.2"}, "name": "2.8", "referencePolicy": {"type":
"Source"}}}]'
oc patch imagestream/backend-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "Backend Redis (latest)"}, "from": { "kind":
"ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- このパッチにより **backend-redis** イメージストリームが更新され、**2.8** タグが含まれるようになります。以下のコマンドにより **tags** 欄に **2.8** が表示されれば、タグが作成されていることを確認することができます。

```
oc get is/backend-redis
```

- イメージに新しい更新があれば、このパッチがトリガーとなり **backend-redis** DeploymentConfig も再デプロイされます。その場合は、新規 Pod の再デプロイが完了して使用できる状態になり、古い Pod が終了するまで待ちます。

3scale イメージのアップグレード を続行します。

1.2.9.1.2. system-redis DeploymentConfig

system-redis DeploymentConfig が現在の 3scale インストールに存在する場合は、 **system-redis** イメージストリームにパッチを適用します。

```
oc patch imagestream/system-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "System 2.8 Redis"}, "from": { "kind": "DockerImage",
"name": "registry.redhat.io/rhscl/redis-32-rhel7:3.2"}, "name": "2.8", "referencePolicy": {"type":
"Source"}}}]'
oc patch imagestream/system-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "System Redis (latest)"}, "from": { "kind":
"ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- このパッチにより **system-redis** イメージストリームが更新され、**2.8** タグが含まれるようになります。以下のコマンドにより **tags** 欄に **2.8** が表示されれば、タグが作成されていることを確認することができます。

```
oc get is/system-redis
```

- イメージに新しい更新があれば、このパッチがトリガーとなり **system-redis** DeploymentConfig も再デプロイされます。その場合は、新規 Pod の再デプロイが完了して使用できる状態になり、古い Pod が終了するまで待ちます。

3scale イメージのアップグレード を続行します。

1.2.9.1.3. system-mysql DeploymentConfig

現在の 3scale インストールに **system-mysql** DeploymentConfig が存在する場合は、 **system-mysql** イメージストリームにパッチを適用します。

```
oc patch imagestream/system-mysql --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "System 2.8 MySQL"}, "from": { "kind": "DockerImage",
"name": "registry.redhat.io/rhscl/mysql-57-rhel7:5.7"}, "name": "2.8", "referencePolicy": {"type":
"Source"}}}]'
```

```
oc patch imagestream/system-mysql --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System MySQL (latest)"}, "from": { "kind": "ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- このパッチにより **system-mysql** イメージストリームが更新され、**2.8** タグが含まれるようになります。以下のコマンドにより **tags** 欄に **2.8** が表示されれば、タグが作成されていることを確認することができます。

```
oc get is/system-mysql
```

- イメージに新しい更新があれば、このパッチがトリガーとなり **system-mysql** DeploymentConfig も再デプロイされます。その場合は、新規 Pod の再デプロイが完了して使用できる状態になり、古い Pod が終了するまで待ちます。

3scale イメージのアップグレード を続行します。

1.2.9.1.4. system-postgresql DeploymentConfig

現在の 3scale インストール環境に **system-postgresql** DeploymentConfig が存在する場合は、**system-postgresql** イメージストリームにパッチを適用します。

```
oc patch imagestream/system-postgresql --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.8 PostgreSQL"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7
"}, "name": "2.8", "referencePolicy": {"type": "Source"}}}]'
oc patch imagestream/system-postgresql --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System PostgreSQL (latest)"}, "from": { "kind": "ImageStreamTag", "name": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- このパッチにより **system-postgresql** イメージストリームが更新され、**2.8** タグが含まれるようになります。以下のコマンドにより **tags** 欄に **2.8** が表示されれば、タグが作成されていることを確認することができます。

```
oc get is/system-postgresql
```

- イメージに新しい更新があれば、このパッチがトリガーとなり **system-postgresql** DeploymentConfig も再デプロイされます。その場合は、新規 Pod の再デプロイが完了して使用できる状態になり、古い Pod が終了するまで待ちます。

3scale イメージのアップグレード を続行します。

1.2.10. smtp ConfigMap の削除

前の手順

「3scale イメージのアップグレード」

現在の手順

この ConfigMap は **system-smtp** シークレットに移行されているため、この手順では **smtp** ConfigMap が削除されます。

smtp ConfigMap を削除するには、次のコマンドを実行します。

```
$ oc delete cm smtp
```

-

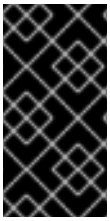
コマンドでエラーが返されない場合には正常に機能しています。

次のステップ

なし。上記の手順をすべて実施すると、テンプレートベースのデプロイメントにおける 3scale 2.7 から 2.8 へのアップグレードが完了します。

第2章 OPERATOR ベースの 3SCALE のアップグレードガイド: 2.7 から 2.8 へ

本セクションでは、operator ベースのデプロイメントにおいて、Red Hat 3scale API Management をバージョン 2.7 から 2.8 にアップグレードする方法について説明します。



重要

必要な条件および手順を理解するために、記載の手順を適用する前に、アップグレードガイド全体を読んでください。アップグレードプロセスの手順が完了するまで、サービスの提供が中断されます。このサービス中断が生じるため、メンテナンス期間を設けるようにしてください。

前提条件

- 3scale operator によりデプロイされている 3scale 2.7
- OpenShift Container Platform (OCP) 4.x クラスターおよびその管理者アクセス

2.1. 3SCALE 2.7 から 2.8 へのアップグレード

operator ベースのデプロイメントにおいて、3scale をバージョン 2.7 から 2.8 にアップグレードするには、以下の手順を使用します。

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. **3scale-operator** がデプロイされているプロジェクトを選択します。
3. **Operators > Installed Operators**の順にクリックします。
4. 3scale operator の **Subscription > Channel**を選択します。
5. **threescale-2.8** を選択してサブスクリプションのチャネルを編集し、変更を保存します。
   - これによりアップグレードプロセスが開始されます。
   - **APIManager** のアップグレードプロセスが完了するまで待ちます。
6. プロジェクトの Pod ステータスのクエリーを行います。

```
oc get pods
```

   - すべての新しいバージョンが動作して使用できる状態になり、エラーが無くなるまで待ちます。
   - アップグレードプロセス中、一時的にエラーが発生する場合があります。

注記

所要時間はおよそ 5 - 10 分間の範囲で幅があります。すべての Pod が動作して使用できる状態になり、エラーが無くなるまで、Pod の状態確認を続けてください。

7. 3scale 管理ポータルにログインして期待通りに動作することを確認し、アップグレードプロセスが正常に完了したことを確認します。

8. **APIManager** オブジェクトのステータスを確認し、以下のコマンドを実行して **YAML** のコンテンツを取得します。

```
oc get apimanager <myapimanager> -o yaml
```

   a. 新しいアノテーションおよび値は以下のようになります。

```
apps.3scale.net/apimanager-threescale-version: "2.8"
apps.3scale.net/threescale-operator-version: "0.5.0"
```

   上記の手順をすべて実施すると、operator ベースのデプロイメントにおける 3scale 2.7 から 2.8 へのアップグレードが完了します。

第3章 3SCALE API MANAGEMENT の移行ガイド: テンプレートベースから OPERATOR ベースのデプロイメントへ

本セクションでは、Red Hat 3scale API Management を Red Hat OpenShift 3.11 を使用するテンプレートベースのデプロイメントから Red Hat OpenShift 4.x を使用する operator ベースのデプロイメントに移行する方法について説明します。

警告

必要な条件および手順を理解するために、記載の手順を適用する前に、移行ガイド全体を読んでください。移行プロセスの手順が完了するまで、サービスの提供が中断されます。このサービス中断が生じるため、メンテナーンス期間を設けるようにしてください。

3.1. 移行の準備

3scale インストールをテンプレートベースから operator ベースのデプロイメントに移行する前に、以下のガイドを参照してデプロイメントがサポートされていることを確認してください。

- テンプレートベースの 3scale デプロイメントのバックアップ
- operator ベースのデプロイメントにおけるバックアップの復元

3.2. 3SCALE のテンプレートベースから OPERATOR ベースのデプロイメントへの移行

前提条件

- 両環境に Red Hat 3scale API Management 2.8 がデプロイされている。
- 各 OpenShift クラスターのドメイン、および 3scale 用の別の WILDCARD_DOMAIN例 :
  - Red Hat OpenShift 3.11 (OCP3): **ocp3.example.com**
  - Red Hat OpenShift 4.x (OCP4): **ocp4.example.com**
  - 3scale: **3scale.example.com**

手順

移行前の基本セットアップは、3scale が OCP3 ドメインをポイントする設定です:
3scale.example.com → ocp3.example.com

3scale を Red Hat OpenShift 3.11 を使用するテンプレートベースのデプロイメントから Red Hat OpenShift 4.1 を使用する operator ベースのデプロイメントに移行するには、以下の手順に従います。

1. テンプレートベースのデプロイメントから 3scale のバックアップを作成する。
2. operator を使用して 3scale をデプロイする。

3. operator ベースのデプロイメントでバックアップを復元する。

4. 3scale WILDCARD_DOMAIN (ここでは **3scale.example.com**) を **ocp4.example.com** にポイントする。

上記の手順をすべて実施すると、3scale のテンプレートベースから operator ベースのデプロイメントへの移行が完了します。