Red Hat 3scale API Management 2.6

3scale の移行

3scale API Management インストールのアップグレード

Last Updated: 2023-06-20

Red Hat 3scale API Management 2.6 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

前書き

本ガイドは、3scale API Management のアップグレードに役立ちます。

パート I. 3SCALE API MANAGEMENT アップグレードガイド: 2.5 から 2.6 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 2.5 から 2.6 にアップグレードする方法について説明します。

警告

このプロセスによりサービスが中断され、アップグレードが完了するまで Zync 処理ジョブが一時的に停止します。メンテナンス期間があることを確認してください。

第1章 始める前に

アップグレードを進める前に、以下の点を考慮する必要があります。

サポートされる構成

- 3scale は、OpenShift 3.11 で **テンプレート** を使用する場合にも、2.5 から 2.6 へのアップグレードパスをサポートします。

3scale の以前のバージョン

- 3scale 2.5 が **amp.yml** 標準シナリオテンプレートでデプロイされたと仮定して、新しい 3scale 2.6 **amp.yml** テンプレートをダウンロードしてデプロイし、新しい OpenShift 要素を作成 します。
  - 3scale 2.6 **amp.yml** テンプレートをダウンロードするには、ノードと資格の設定 を参照してください。
- 2.4 より前のバージョンからのマルチバージョンアップグレードの場合、**システム環境** の ConfigMap があることを確認します。

```
$ oc get configmap system-environment
```

  - **NotFound** エラーメッセージが表示された場合は、2.4 アップグレードガイドの ConfigMap の作成 を参照してください。

ツール

- データベースのバックアップを実行します。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。
- 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。
- bash シェルで次のコマンドを実行します。
- このアップグレードでは、次の手順に従ってパッチファイルをダウンロードして取得します。
  1. templates-migration-2.5-to-2.6 をクリックします。
  2. ファイルをダウンロードして展開します。

ドキュメント全体で、ダウンロードした圧縮ファイルの内容に関連するいくつかのファイル参照が表示されます。たとえば、**$(cat db-imagestream-patches/backend-redis-json.patch)** です。

第2章 3SCALE 2.5 から 2.6 へのアップグレード

前提条件

- Red Hat 3scale API Management 2.5 がプロジェクトにデプロイされている。
- ツールの前提条件
  - base64
  - jq

手順

3scale API Management 2.5 を 2.6 にアップグレードするには、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. 認証済みレジストリーのサポート設定
3. OpenShift リソースの作成
4. Redis Enterprise および Redis Sentinel のシステム DeploymentConfig 設定
5. Redis Sentinel 環境変数の修正
6. Zync 環境変数の修正
7. DeploymentConfig データベースの ImageStreams への移行
8. 3scale イメージのアップグレード
9. WildcardRouter から Zync Route Management への移行

2.1. 3SCALE プロジェクトのバックアップディレクトリーの作成

3scale プロジェクトのバックアップディレクトリーを作成するには、以下の手順に従います。**{BACKUP_DIR}** は、3scale バックアップのマシン上の場所であることに **注意** してください。

手順

1. バックアップディレクトリーを作成します。

```
mkdir ${BACKUP_DIR}
```

2. バックアップ用のディレクトリーおよびサブディレクトリーを作成します。

```
mkdir ${BACKUP_DIR}/system-database ${BACKUP_DIR}/zync-database
${BACKUP_DIR}/system-redis ${BACKUP_DIR}/backend-redis ${BACKUP_DIR}/system-
app ${BACKUP_DIR}/openshift
```

```
mkdir ${BACKUP_DIR}/openshift/configmaps/
${BACKUP_DIR}/openshift/deploymentConfigs ${BACKUP_DIR}/openshift/imageStreams
${BACKUP_DIR}/openshift/other/ ${BACKUP_DIR}/openshift/routes/
${BACKUP_DIR}/openshift/secrets/ ${BACKUP_DIR}/openshift/services/
```

2.1.1. system-database (MySQL) のバックアップ

system-mysql データベースをダンプし、ダンプを **${BACKUP_DIR}/system-database/system-mysql-backup.gz** 内に保存します。

```
oc rsh $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq -r '.items[0].metadata.name')
bash -c 'export MYSQL_PWD=${MYSQL_ROOT_PASSWORD}; mysqldump --single-transaction -
hsystem-mysql -uroot system' | gzip > ${BACKUP_DIR}/system-database/system-mysql-backup.gz
```

2.1.2. zync-database のバックアップ

zync-database PostrgreSQL データベースをダンプし、ダンプを **${BACKUP_DIR}/zync-database/zync-database-backup.gz** 内に保存します。

```
oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq '.items[0].metadata.name' -r)
bash -c 'pg_dumpall -c --if-exists' | gzip > ${BACKUP_DIR}/zync-database/zync-database-backup.gz
```

2.1.3. system-redis のバックアップ

${BACKUP_DIR}/system-redis/system-redis-dump.rdb 内に **system-redis** ダンプを抽出します

```
oc cp $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq '.items[0].metadata.name' -
r):/var/lib/redis/data/dump.rdb ${BACKUP_DIR}/system-redis/system-redis-dump.rdb
```

2.1.4. backend-redis のバックアップ

${BACKUP_DIR}/backend-redis/backend-redis-dump.rdb 内に **backend-redis** ダンプを抽出します。

```
oc cp $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq '.items[0].metadata.name' -
r):/var/lib/redis/data/dump.rdb ${BACKUP_DIR}/backend-redis/backend-redis-dump.rdb
```

2.1.5. system-app 永続データのバックアップ

${BACKUP_DIR}/system-app/ 内に **system-app** 永続データをコピーします。

```
oc rsync $(oc get pods -l 'deploymentConfig=system-app' -o json | jq '.items[0].metadata.name' -
r):/opt/system/public/system ${BACKUP_DIR}/system-app/
```

2.1.6. DeploymentConfig のバックアップ

```
for object in `oc get dc | awk '{print $1}' | grep -v NAME`; do oc get -o yaml --export dc ${object} >
${BACKUP_DIR}/openshift/deploymentConfigs/${object}_dc.yaml; done
```

2.1.7. ImageStream のバックアップ

```
for object in `oc get is | awk '{print $1}' | grep -v NAME`; do oc get -o yaml --export is ${object} >
${BACKUP_DIR}/openshift/imageStreams/${object}_is.yaml; done
```

2.1.8. シークレットのバックアップ

トークンとシークレットのデフォルトのビルダーとデプロイヤーを除くすべてをバックアップします。

```
for object in `oc get secret | awk '{print $1}' | grep -v NAME | grep -v default | grep -v builder | grep -v
deployer`; do oc get -o yaml --export secret ${object} >
${BACKUP_DIR}/openshift/secrets/${object}_secret.yaml; done
```

2.1.9. サービスのバックアップ

```
for object in `oc get svc | awk '{print $1}' | grep -v NAME`; do oc get -o yaml --export svc ${object} >
${BACKUP_DIR}/openshift/services/${object}_svc.yaml; done
```

2.1.10. アップルートのバックアップ

```
for object in `oc get routes | awk '{print $1}' | grep -v NAME`; do oc get -o yaml --export route
${object} > ${BACKUP_DIR}/openshift/routes/${object}_route.yaml; done
```

2.1.11. ConfigMap のバックアップ

```
for object in `oc get cm | awk '{print $1}' | grep -v NAME`; do oc get -o yaml --export cm ${object} >
${BACKUP_DIR}/openshift/configmaps/${object}_cm.yaml; done
```

2.1.12. その他のリソースのバックアップ

バックアップオブジェクトではないその他のカスタムリソースを処理するために、2 番目のバックアップを作成します。

```
oc get -o yaml --export all > ${BACKUP_DIR}/openshift/other/threescale-project-elements.yaml

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 > ${BACKUP_DIR}/openshift/other/$object.yaml; done
```

2.2. 認証済みレジストリーのサポート設定

3scale 2.6 リリースの一環で、コンテナーイメージが **registry.access.redhat.com** から **registry.redhat.io** にある認証済みレジストリーに移行されました。次の手順に従って、既存の 3scale インフラストラクチャーを準備し、新しい認証済みレジストリーをサポートします。

1. **registry.redhat.io** にある新しい Red Hat 認証済みレジストリーに認証情報を作成します。

- レジストリーサービスアカウントとも呼ばれるレジストリートークンを作成します。このレジストリートークンは、**registry.redhat.io** に対して認証するために 3scale プラットフォームで使用することを目的としています。
- 認証情報を作成する方法の詳細は、Red Hat Container Registry Authentication を参照してください。

2. レジストリーサービスアカウントが利用可能になったら、3scale インフラストラクチャーがデプロイされている OpenShift プロジェクトで、その認証情報を含む新しいシークレットを作成します。
   a. Red Hat Service Accounts パネルに移動して、OpenShift シークレット定義を取得します。
   b. 3scale インフラストラクチャーに使用するレジストリーサービスアカウントを選択します。
   c. **OpenShift シークレット** タブを選択し、シークレットのダウンロードリンクをクリックします。
3. Red Hat サービスアカウントパネルから OpenShift シークレットをダウンロードした後に、YAML ファイルの **metadata** セクションの **name** フィールドを変更し、既存の名前を **threescale-registry-auth** 名に置き換えます。
   シークレットは次のようになります。

```
apiVersion: v1
kind: Secret
metadata:
  name: threescale-registry-auth
data:
  .dockerconfigjson: a-base64-encoded-string-containing-auth-credentials
type: kubernetes.io/dockerconfigjson
```

4. 変更を保存し、現在 3scale 2.5 がデプロイされている OpenShift プロジェクトにシークレットを作成します。

```
$ oc create -f the-secret-name.yml
```

5. シークレットを作成したら、その存在を確認できます。次のコマンドを実行すると、コンテンツを含むシークレットが返されます。

```
$ oc get secret threescale-registry-auth
```

6. **threescale-registry-auth** シークレットを使用する **amp** サービスアカウントを作成します。これには、次の内容でファイル **amp-sa.yml** を作成します。

```
apiVersion: v1
kind: ServiceAccount
imagePullSecrets:
- name: threescale-registry-auth
metadata:
  name: amp
```

7. **amp** サービスアカウントをデプロイします。

```
$ oc create -f amp-sa.yml
```

8. **amp** サービスアカウントが正しく作成されていることを確認します。次のコマンドは、**imagePullSecrets** セクションの要素の 1 つとして **threescale-registry-auth** を含む、作成されたサービスアカウントをコンテンツとともに返します。

```
$ oc get sa amp -o yaml
```

9. 既存の 3scale プロジェクトのデフォルトサービスアカウントに適用された権限が、新しい **amp** サービスアカウントにレプリケートされることを確認します。

   - サービスアカウントの認証モードでサービスディカバリーが設定されている場合に、Configuring without OAuth server の説明に従い、cluster-role の表示権限を **system:serviceaccount:<3scale-project>:default** ユーザーに割り当て、同じ権限を **system:serviceaccount:<3scale-project>:amp** にも適用する必要があります。

     ```
     $ oc adm policy add-cluster-role-to-user view system:serviceaccount:<3scale-project>:amp
     ```

10. 既存のすべての DeploymentConfigs を更新して、新しい **amp** サービスアカウントを使用します。

    ```
    $ THREESCALE_DC_NAMES="apicast-production apicast-staging apicast-wildcard-router backend-cron backend-listener backend-redis backend-worker system-app system-memcache system-mysql system-redis system-sidekiq system-sphinx zync zync-database" for component in ${THREESCALE_DC_NAMES}; do oc patch dc $component --patch '{"spec":{"template": {"spec": {"serviceAccountName": "amp"}}}}' ; done
    ```

    コマンドの出力には、次の行が含まれます。

    ```
    deploymentconfig.apps.openshift.io/apicast-production patched
    deploymentconfig.apps.openshift.io/apicast-staging patched
    deploymentconfig.apps.openshift.io/apicast-wildcard-router patched
    deploymentconfig.apps.openshift.io/backend-cron patched
    deploymentconfig.apps.openshift.io/backend-listener patched
    deploymentconfig.apps.openshift.io/backend-redis patched
    deploymentconfig.apps.openshift.io/backend-worker patched
    deploymentconfig.apps.openshift.io/system-app patched
    deploymentconfig.apps.openshift.io/system-memcache patched
    deploymentconfig.apps.openshift.io/system-mysql patched
    deploymentconfig.apps.openshift.io/system-redis patched
    deploymentconfig.apps.openshift.io/system-sidekiq patched
    deploymentconfig.apps.openshift.io/system-sphinx patched
    deploymentconfig.apps.openshift.io/zync patched
    deploymentconfig.apps.openshift.io/zync-database patched
    ```

    前のコマンドは、再起動をトリガーする 3scale の既存の DeploymentConfig もすべて再デプロイします。

11. DeploymentConfig が再起動されている間、それらのステータスが変化する場合があります。すべての DeploymentConfig が **Ready** になるまで待ちます。

- 次のコマンドを入力し、DeploymentConfig ごとに、**Desired** 列と **Current** 列の値が同じであり、0 以外であることをチェックして、DeploymentConfig のステータスを確認できます。

```
$ oc get dc
```

12. また、すべての Pod が **Running** ステータスであり、**Ready** になっていることを確認します。

```
$ oc get pods
```

13. 次のコマンドを使用して、すべての DeploymentConfig に **amp** サービスアカウントが設定されていることを確認します。

```
$ for component in ${THREESCALE_DC_NAMES}; do oc get dc $component -o yaml | grep -i serviceAccountName; done
```

14. 前のコマンドの結果として、以前に設定された THREESCALE_DC_NAMES 環境変数で定義した要素数だけ次の行が繰り返されます。

```
serviceAccountName: amp
```

15. この時点で、DeploymentConfigurations は Red Hat 認証レジストリーイメージを使用する準備ができています。

2.3. OPENSHIFT リソースの作成

このセクションでは、これらの新しい要素の作成に必要な手順について説明します。3scale 2.6 リリースの一環として、次の OpenShift 要素が追加されました。

- データベース用の新しい ImageStreams:
  - backend-redis
  - system-redis
  - system-memcached
  - system-mysql
  - zync-database-postgresql
- 以下の OpenShift オブジェクトを含む新しい **zync-que** コンポーネント:
  - **zync-que** DeploymentConfig
  - **zync-que-sa** ServiceAccount
  - **zync-que** ロール
  - **zync-que-rolebinding** RoleBinding

新しい OpenShift 要素を作成するには、次の手順に従います。

1. 3scale 2.5 のデプロイ時に設定された WildcardDomain を含む次の環境変数を作成します。

```
$ THREESCALE_WILDCARD_DOMAIN=$(oc get configmap system-environment  -o json |
jq .data.THREESCALE_SUPERDOMAIN -r)
```

2. THREESCALE_WILDCARD_DOMAIN 環境変数が空ではなく、3scale 2.5 のデプロイ時に設定されたワイルドカードドメインと同じ値であることを確認します。

```
$ echo ${THREESCALE_WILDCARD_DOMAIN}
```

3. ImageStreams に設定された **ImportPolicy** ImageStream 値を含む次の環境変数を作成します。

```
$ IMPORT_POLICY_VAL=$(oc get imagestream amp-system -o json | jq -r
".spec.tags[0].importPolicy.insecure")
if [ "$IMPORT_POLICY_VAL" == "null" ]; then
  IMPORT_POLICY_VAL="false"
fi
```

4. IMPORT_POLICY_VAL 環境変数が **true** または **false** であることを確認します。

```
$ echo ${IMPORT_POLICY_VAL}
```

5. 3scale Pod の **アプリ** Kubernetes ラベルの現在の値を含む次の環境変数を作成します。たとえば、**backend-listener** Pod から取得します。

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

6. DEPLOYED_APP_LABEL 環境変数が空でも **null** でもないことを確認します。

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

7. 3scale 2.6 **amp.yml** 標準シナリオテンプレートを使用して、2.6 リリース用の新しい OpenShift オブジェクトをデプロイします。

```
$ oc new-app -f amp.yml --param
WILDCARD_DOMAIN=${THREESCALE_WILDCARD_DOMAIN} --param
IMAGESTREAM_TAG_IMPORT_INSECURE=${IMPORT_POLICY_VAL} --param
APP_LABEL=${DEPLOYED_APP_LABEL}
```

いくつかのエラーが表示されます。一部の要素は 3scale 2.5 にすでに存在していたため、これらは想定範囲内です。エラーでない行で、表示されているものには、以下が挙げられます。

```
imagestream.image.openshift.io "zync-database-postgresql" created
imagestream.image.openshift.io "backend-redis" created
imagestream.image.openshift.io "system-redis" created
imagestream.image.openshift.io "system-memcached" created
imagestream.image.openshift.io "system-mysql" created
role.rbac.authorization.k8s.io "zync-que-role" created
serviceaccount "zync-que-sa" created
rolebinding.rbac.authorization.k8s.io "zync-que-rolebinding" created
deploymentconfig.apps.openshift.io "zync-que" created
```

8. 前述のすべての新しい ImageStreams と、すべての新しい zync-que 関連要素が存在することを確認します。

```
$ oc get is system-redis
$ oc get is system-mysql
$ oc get is system-memcached
$ oc get is zync-database-postgresql
$ oc get is backend-redis
$ oc get role zync-que-role
$ oc get sa zync-que-sa
$ oc get rolebinding zync-que-rolebinding
$ oc get dc zync-que
```

前のコマンドはすべて、それらが作成されたことを示す出力を返します。また、次のように入力した場合:

```
$ oc get pods | grep -i zync-que
```

そのステータスが **エラー** またはクラッシュしていることを示すその他のエラーであることがわかります。この時点では Zync イメージが更新されていないため、これは想定範囲内です。これは、「3scale イメージのアップグレード」セクション のポイント 4 で行われます。

2.4. REDIS ENTERPRISE および REDIS SENTINEL のシステム DEPLOYMENTCONFIG 設定

このセクションは、作成したシークレットフィールドを使用するように既存の **システム** の DeploymentConfigs を設定する場合に役立ちます。これらのシークレットフィールドは、**system-redis** で環境変数として使用されます。

1. **system-redis** シークレットのシステム接続の Redis Enterprise 互換性に関連するフィールドを追加します。

```
$ oc patch secret/system-redis --patch '{"stringData":
{"MESSAGE_BUS_SENTINEL_HOSTS": "", "MESSAGE_BUS_SENTINEL_ROLE": "",
"SENTINEL_HOSTS": "", "SENTINEL_ROLE": "", "MESSAGE_BUS_NAMESPACE": "",
"MESSAGE_BUS_URL": "", "NAMESPACE": ""}}'
```

2. 新しい環境変数を **system-app** コンテナーに追加します。

```
$ oc patch dc/system-app -p "$(cat redis-patches/system-app-podcontainers.patch)"
```

このコマンドは、**system-app** DeploymentConfig の再起動をトリガーします。DeploymentConfig Pod が再起動され、再び **Ready** ステータスになるまで待ちます。

3. 次のコマンドを使用して、DeploymentConfig のすべての環境変数を一覧表示します。

```
$ oc set env dc a-deployment-config-name --list
```

- このコマンドを実行して、このステップの項目の各パッチコマンドの前後に環境変数のリストを取得します。
- 以下は、環境変数を一覧表示するコマンドを使用できず、特定のコマンドが必要な特殊なケースです。
  - **pre-hook** pod:

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

- **system-sidekiq** initContainer

```
$ oc get dc system-sidekiq -o json | jq .spec.template.spec.initContainers[0].env
```

4. 新しい環境変数を **system-app** pre-hook Pod に追加します。

```
$ oc patch dc/system-app -p "$(cat redis-patches/system-app-prehookpod-json.patch)" --
type json
```

前のコマンドを実行した後、既存の環境変数は変更されずに残ります。さらに、新しい変数が system-app の **pre-hook** pod と system-app のすべてのコンテナー (system-master、system-developer、system-provider) に追加され、**system-secret** シークレットがそのソースとして使用されます。

- REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_URL
- MESSAGE_BUS_REDIS_SENTINEL_HOSTS
- MESSAGE_BUS_REDIS_SENTINEL_ROLE
- REDIS_SENTINEL_HOSTS
- REDIS_SENTINEL_ROLE
- BACKEND_REDIS_SENTINEL_HOSTS
- BACKEND_REDIS_SENTINEL_ROLE

5. 新しい環境変数を **system-sidekiq** に追加します。

```
$ oc patch dc/system-sidekiq -p "$(cat redis-patches/system-sidekiq.patch)"
```

このコマンドは、**system-sidekiq** DeploymentConfig の再起動をトリガーします。DeploymentConfig Pod が再起動され、再び準備完了状態になるまで待ちます。

前のコマンドの実行後に、次の環境変数が追加され、既存のものは変更されずに、**system-sidekiq** の **system-sidekiq** InitContainer に追加されました。

- REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_URL
- MESSAGE_BUS_REDIS_SENTINEL_HOSTS
- MESSAGE_BUS_REDIS_SENTINEL_ROLE
- REDIS_SENTINEL_HOSTS

- REDIS_SENTINEL_ROLE
  さらに、次の環境変数が **system-sidekiq** Pod に追加されました。
- REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_URL
- MESSAGE_BUS_REDIS_SENTINEL_HOSTS
- MESSAGE_BUS_REDIS_SENTINEL_ROLE
- REDIS_SENTINEL_HOSTS
- REDIS_SENTINEL_ROLE
- BACKEND_REDIS_SENTINEL_HOSTS
- BACKEND_REDIS_SENTINEL_ROLE

6. 新しい環境変数を **system-sphinx** に追加します。

```
$ oc patch dc/system-sphinx -p "$(cat redis-patches/system-sphinx.patch)"
```

このコマンドは、**system-sphinx** DeploymentConfig の再起動をトリガーします。DeploymentConfig Pod が再起動され、再び準備完了状態になるまで待ちます。

前のコマンドの実行後に、次の環境変数が追加され、既存のものは変更されずに、**system-sphinx** pod に追加されました。

- REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_NAMESPACE
- MESSAGE_BUS_REDIS_URL
- MESSAGE_BUS_REDIS_SENTINEL_HOSTS
- MESSAGE_BUS_REDIS_SENTINEL_ROLE
- REDIS_SENTINEL_HOSTS
- REDIS_SENTINEL_ROLE
- REDIS_URL

2.5. REDIS SENTINEL 環境変数の修正

このステップには、Redis Sentinel 接続設定が **backend-worker** および **backend-cron** Pod 機能の障害となっていた 3scale 2.5 の問題に対する修正が含まれます。

1. 次のコマンドを実行して、DeploymentConfig InitContainer の既存の環境変数をすべて表示します。

```
$ oc get dc a-deployment-config-name -o json | jq .spec.template.spec.initContainers[0].env
```

以下のコマンドを使用して、この手順で実行される各パッチコマンドの前後に環境変数のリストを取得し、すべてが期待どおりに機能していることを確認します。

2. backend-worker に Redis Sentinel 接続の修正を適用します。

```
$ oc patch dc/backend-worker -p "$(cat redis-patches/backend-worker.patch)"
```

このコマンドを実行すると、次の環境変数が **backend-worker** DeploymentConfig の **backend-worker** InitContainer に追加されます。

- CONFIG_REDIS_PROXY
- CONFIG_REDIS_SENTINEL_HOSTS
- CONFIG_REDIS_SENTINEL_ROLE
- CONFIG_QUEUES_SENTINEL_HOSTS
- CONFIG_QUEUES_SENTINEL_ROLE
- RACK_ENV

3. **backend-cron** に Redis Sentinel 接続の修正を適用します。

```
$ oc patch dc/backend-cron -p "$(cat redis-patches/backend-cron.patch)"
```

このコマンドを実行すると、次の環境変数が **backend-cron** DeploymentConfig の **backend-cron** InitContainer に追加されます。

- CONFIG_REDIS_PROXY
- CONFIG_REDIS_SENTINEL_HOSTS
- CONFIG_REDIS_SENTINEL_ROLE
- CONFIG_QUEUES_SENTINEL_HOSTS
- CONFIG_QUEUES_SENTINEL_ROLE
- RACK_ENV

2.6. ZYNC 環境変数の修正

- 次のコマンドを実行して、zync 環境を更新します。

```
oc patch dc zync -p '{"spec": {"template": {"spec": {"containers": [{"name": "zync", "env": [{"name": "POD_NAME", "valueFrom": {"fieldRef": {"apiVersion": "v1", "fieldPath": "metadata.name"}}}, {"name": "POD_NAMESPACE", "valueFrom": {"fieldRef": {"apiVersion": "v1", "fieldPath": "metadata.namespace"}}}]}]}}}}'
```

2.7. DEPLOYMENTCONFIG データベースの IMAGESTREAMS への移行

2.6 では、データベースを含むデプロイ済みの 3scale DeploymentConfig は、イメージ URL への直接参照ではなく、ImageStreams からコンテナーイメージを取得するように移行されました。

1. **backend-redis** DeploymentConfig を移行して、backend-redis ImageStream を使用します。

```
$ oc patch dc/backend-redis -p "$(cat db-imagestream-patches/backend-redis-json.patch)" --type json
```

   - これにより、**backend-redis** DeploymentConfig の再デプロイがトリガーされ、DeploymentConfig には **backend-redis** ImageStream を参照する **ImageChange** トリガーが含まれるようになりました。
   - **backend-worker**、**backend-cron** または **backend-listener** は、**backend-redis** Pod が再デプロイされるまで一時的に失敗する場合があります。
     DeploymentConfig Pod が再起動され、再び準備完了状態になるまで待ちます。

2. **system-redis** DeploymentConfig を移行して、 **system-redis** ImageStream を使用します。

```
$ oc patch dc/system-redis -p "$(cat db-imagestream-patches/system-redis-json.patch)" --type json
```

   - これにより、**system-redis** DeploymentConfig の再デプロイがトリガーされ、DeploymentConfig には **backend-redis** ImageStream を参照する **ImageChange** トリガーが含まれるようになりました。
   - DeploymentConfig Pod が再起動され、再び準備完了状態になるまで待ちます。

3. **system-memcached** ImageStream を使用するように **system-memcache** DeploymentConfig を移行します。

```
$ oc patch dc/system-memcache -p "$(cat db-imagestream-patches/system-memcached-json.patch)" --type json
```

   - これにより、**system-memcache** DeploymentConfig の再デプロイがトリガーされ、DeploymentConfig には、**system-memcached** ImageStream を参照する **ImageChange** トリガーが含まれるようになりました。
   - DeploymentConfig Pod が再起動され、再び準備完了状態になるまで待ちます。

4. **system-mysql** DeploymentConfig を移行して、 **system-mysql** ImageStream を使用します。

```
$ oc patch dc/system-mysql -p "$(cat db-imagestream-patches/system-mysql-json.patch)" --type json
```

   - これにより、**system-mysql** DeploymentConfig の再デプロイがトリガーされ、DeploymentConfig には、**system-mysql** ImageStream を参照する **ImageChange** トリガーが含まれるようになりました。
   - DeploymentConfig Pod が再起動され、再び準備完了状態になるまで待ちます。

5. **zync-database** DeploymentConfig を移行して、 **zync-database-postgresql** ImageStream を使用します。

```
$ oc patch dc/zync-database -p "$(cat db-imagestream-patches/zync-database-postgresql.patch)"
```

- これにより、**zync-database** DeploymentConfig の再デプロイがトリガーされ、DeploymentConfig には、**zync-database-postgresql** ImageStream を参照する **ImageChange** トリガーが含まれるようになりました。
- **zync** DeploymentConfig Pod は、 **zync-database** が再び使用可能になるまで一時的に失敗する可能性があり、再び準備完了状態になるまで時間がかかる場合があります。数分後、すべての zyncDeploymentConfig Pod が **Ready** ステータスになっていることを確認します。
- 続行する前に、DeploymentConfig Pod が再起動され、再び準備完了状態になるまで待ちます。

6. 使用されなくなった **postgresql** ImageStream を削除します。

```
$ oc delete ImageStream postgresql
```

7. 成功を確認するには、次のことを確認します。
   - すべてのデータベース関連の DeploymentConfigs が ImageStream を使用するようになりました。対応するデータベース ImageStream を指す **ImageChange** トリガーが作成されたことを確認できます。
   - **ImageChange** トリガーには、**registry.redhat.io** を指す URL を含む **lastTriggeredImage** という名前のフィールドがあります。

2.8. 3SCALE イメージのアップグレード

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

```
$ oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "AMP system 2.6"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp26/system"}, "name": "2.6",
"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": "2.6"}, "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。

注記

Oracle データベースを使用している場合は、上記の手順を実行した後、Oracle データベースを使用した 3scale システムイメージ の手順に従って、システムイメージを再構築する必要があります。

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 2.6"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp26/apicast-gateway"}, "name": "2.6",
"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": "2.6"}, "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 2.6"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp26/backend"}, "name": "2.6",
"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": "2.6"}, "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 2.6"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp26/zync"}, "name": "2.6",
"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": "2.6"}, "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。
- さらに、前のセクションで作成したときに **Error** 状態だった **zync-que** が現在実行されており、その Pod が **Ready** 状態になっていることがわかります。

5. 表示されるリリースバージョンを更新します。

```
$ oc set env dc/system-app AMP_RELEASE=2.6
```

これにより、**system-app** DeploymentConfig の再デプロイメントがトリガーされます。再デプロイが実行され、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. 最終的に、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 apicast-wildcard-router
backend-cron backend-listener backend-redis backend-worker system-app system-
memcache system-mysql system-redis system-sidekiq system-sphinx zync zync-database"
```

```
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
```

2.9. WILDCARDROUTER から ZYNC ROUTE MANAGEMENT への移行

3scale 2.6 では、WildcardRouter コンポーネントとワイルドカード OpenShift ルートが削除され、Zync サブシステムによって管理される個別の OpenShift ルートとして作成されるようになりました。この手順では、WildcardRouter から Zync へのルート管理の移行について詳しく説明します。

この時点で、すべての 3scale イメージが 3scale 2.6 にアップグレードされています。3scale サービスとテナントに対応する OpenShift ルートの作成と削除は、Zync サブシステムによって自動的に管理されます。さらに、そのために必要なすべての新しい Zync インフラストラクチャーは、前のセクション新しく OpenShift 要素を追加することで利用可能になります。

OpenShift ルート管理を WildcardRouter から Zync に移行するには、OpenShift ルートとワイルドカードルートに関連する古い 3scale テナントとサービスを削除してから、Zync による既存のサービスとテナントの強制的な再評価を実行する必要があります。これにより、Zync は現在持っているものと同等のルートを作成します。

パブリックベース URL が変更されると、**system-app** でイベントがトリガーされ、**system-redis** に保存されているジョブキューを介して **system-sidekiq** に通知されます。ジョブはバックグラウンドで処理され、zync に送信され、データが **zync-db** にすでに存在するかどうかがチェックされます。変更を検出すると、**zync-que** で処理されたジョブを介して新しいルートを作成します。

重要

SSL 証明書をいくつかのルートに手動でインストールした場合は、操作を行う前に、ルートに割り当てられた証明書をコピーし、各証明書が割り当てられたルートをメモしておく必要があります。証明書の機能を維持する場合は、Zync で作成される同等の新規ルートに証明書をインストールする必要があります。

注記

- サービスはデフォルトで、**hosted** オプションを使用して移行されます。
- パブリックベース URL は自動的に入力され、ルートは Zync によって作成されます。
- ステップ 1 は、オプション **self_managed** を使用して外部 APIcast ゲートウェイを設定する場合にのみ必要です。
- **3scale_managed** オプションを選択すると、ルートは Zync によって自動的に管理されます。

手順

1. Zync が外部ゲートウェイのルートを管理しない場合、提案された代替案のいずれかの手順に従って、Zync で管理されない各サービスのデプロイメントオプションを変更できます。
   - 3scale で以下の操作を行います。
     a. **Integration** ページに移動し、**edit integration settings** をクリックします。
     b. 適切なデプロイメントオプションを選択し、変更があれば保存します。

- API の使用

  a. サービス識別子 (**ID**) とアクセストークン (**ACCESS_TOKEN**) およびテナントエンドポイント (**TENANT_URL**) でサービスを更新します。

  ```
  $ curl -XPUT "${TENANT_URL}/admin/api/services/${ID}.json" -d
  deployment_option=self_managed -d access_token="${ACCESS_TOKEN}"
  ```

  または、ホストされている APIcast を使用している場合は、次のコマンドを使用できます。

  ```
  $ curl -XPUT "${TENANT_URL}/admin/api/services/${ID}.json" -d
  deployment_option=hosted -d access_token="${ACCESS_TOKEN}"
  ```

  b. 各テナントのサービスごとに、3scale または API を介して **deployment_option** フィールドを変更します。

  - **deployment_option** を **self_managed** に設定できるケースは次のとおりです。
    - APIcast は、OpenShift のカスタムルートにリンクされています。
    - APIcast は OpenShift の外部でホストされます。
    - **APICAST_PATH_ROUTING** が **true** に設定されています。
  - それ以外の場合は、**deployment_option** を **hosts** に設定します。

2. 存在する可能性のあるルートの中で、デフォルトのルートの中で 3scale が 2.5 で自動的に作成したものがあります。それらを削除することから始めます。

```
$ oc delete route system-master
$ oc delete route system-provider-admin
$ oc delete route system-developer
$ oc delete route api-apicast-production
$ oc delete route api-apicast-staging
```

- **WILDCARD_POLICY=** Subdomain を指定して 3scale 2.5 をデプロイした場合は、以下を使用してワイルドカードルートを削除する必要があります。

```
$ oc delete route apicast-wildcard-router
```

- それ以外の場合、**WILDCARD_POLICY=** Subdomain を指定せずに 3scale 2.5 をデプロイした場合は、Zync が作成するルートの重複を避けるために、3scale テナントおよびサービス用に手動で作成したルートを削除する必要があります。

この時点で、サービスとテナントに関連するすべてのルートが削除されている必要があります。ここで、Zync による同等のルートの作成を実行します。

1. Zync を使用して、すべての 3scale サービスとテナント OpenShift ルートを強制的に再度同期します。

```
$ SYSTEM_SIDEKIQ_POD=$(oc get pods | grep sidekiq | awk '{print $1}')
```

2. SYSTEM_SIDEKIQ_POD 環境変数の結果が空でないことを確認します。

```
$ echo ${SYSTEM_SIDEKIQ_POD}
```

3. 最後に、再同期を実行します。

```
$ oc exec -it ${SYSTEM_SIDEKIQ_POD} -- bash -c 'bundle exec rake zync:resync:domains'
```

システムへの通知に関する情報を含むこのスタイルの出力が表示されます。

```
No valid API key has been set, notifications will not be sent
ActiveMerchant MODE set to 'production'
[Core] Using http://backend-listener:3000/internal/ as URL
OpenIdAuthentication.store is nil. Using in-memory store.
[EventBroker] notifying subscribers of Domains::ProviderDomainsChangedEvent 59a554f6-
7b3f-4246-9c36-24da988ca800
[EventBroker] notifying subscribers of ZyncEvent caa8e941-b734-4192-acb0-0b12cbaab9ca
Enqueued ZyncWorker#d92db46bdba7a299f3e88f14 with args: ["caa8e941-b734-4192-
acb0-0b12cbaab9ca", {:type=>"Provider", :id=>1, :parent_event_id=>"59a554f6-7b3f-4246-
9c36-24da988ca800", :parent_event_type=>"Domains::ProviderDomainsChangedEvent",
:tenant_id=>1}]
[EventBroker] notifying subscribers of Domains::ProviderDomainsChangedEvent 9010a199-
2af1-4023-9b8d-297bd618096f
…
```

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

プロセスが完了するまでに、以下が作成されたことが分かります。

- 1 つのマスター管理ポータルルート。
  3scale テナントごとに、以下の 2 つのルートが作成されます。
- テナントの管理ポータルルート。
- テナントの開発者ポータルルート。
  3scale サービスごとに 2 つのルートが作成されます。
- サービスに対応する APIcast staging Route。
- サービスに対応する APIcast production Route。

4. 上記で説明した予想されるすべてのルートが、既存のすべてのサービスとテナントに対して作成されていることを確認します。次のコマンドを実行すると、すべてのルートを表示できます。

```
$ oc get routes
```

前のコマンドの出力として表示されるホスト/ポートフィールドは、ルートの URL である必要があります。

- WILDCARD_POLICY を Subdomain に設定して **3scale** 2.5 をデプロイした場合には、新しいルートはすべて、古い OpenShift ワイルドカードルートと同じベースの WildcardDomain を指定する必要があります。

- それ以外の場合、WILDCARD_POLICY=Subdomain なしで 3scale 2.5 をデプロイした場合に、新しいルートは、2.5 リリースで 3scale によって自動的に作成されたものを含め、削除した古いルートと同じホストを指定する必要があります。

5. 最後に、古いワイルドカードルート、または手動で作成された古いルートにカスタム SSL 証明書を使用していた場合は、その証明書を Zync で作成された新しいルートにインストールします。これには、OpenShift Web パネルでルートを編集し、証明書をルートに追加します。

6. この移行の前に存在していたサービスとテナントが、新しいルートを使用して引き続き解決できることを確認します。このタスクを実行するには、以下のテストを実行します。

   a. この移行の前にすでに存在していた 3scale サービスに関連付けられた既存の APIcast 実稼働 URL のルートを解決します。

   b. この移行の前にすでに存在していた 3scale サービスに関連付けられた既存の APIcast ステージング URL のルートを解決します。

   c. この移行の前にすでに存在していた既存のテナントのルートを解決します。

7. 新しい Zync 機能が動作していることを確認するときに、新しいテナントとサービスの作成時に新しいルートが生成されることを確認します。このタスクを実行するには、以下のテストを実行します。

   a. マスターパネルから新しいテナントを作成し、数秒後にそれに関連付けられた新しいルートが OpenShift に表示されることを確認します。

   b. 既存のテナントの 1 つで新しいサービスを作成し、数秒後にそれに関連付けられた新しいルートが OpenShift に表示されることを確認します。

8. apicast-wildcard-router サービスを削除します。

```
$ oc delete service apicast-wildcard-router
```

9. 非推奨の WildcardRouter サブシステムを削除します。

```
$ oc delete ImageStream amp-wildcard-router
$ oc delete DeploymentConfig apicast-wildcard-router
```

リストされたすべてのステップを実行すると、2.5 から 2.6 への 3scale のアップグレードが完了します。