Red Hat 3scale API Management 2.5

3scale の移行

3scale API Management インストールのアップグレード

Last Updated: 2023-03-26

Red Hat 3scale API Management 2.5 3scale の移行

3scale API Management インストールのアップグレード

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

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 2.4 から 2.5 へのアップグレードガイド

このセクションには、Red Hat 3scale API Management のバージョン 2.4 から 2.5 へのアップグレードに関する情報が含まれています。

警告

このプロセスにより、サービスが中断されます。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: サービス別の OAUTH フローのデプロイ

3scale 2.5 の以前のバージョンでは、アプリケーションを手動で作成または更新した場合に、Red Hat Single Sign On (RH-SSO) で 3scale によってデフォルトで有効化された OAuth フローは **standardFlowEnabled** (Authorization Code Flow) でした。Resource Owner Password、Implicit、または Client Credentials など、別のフローに変更した場合、このアップグレードプロセスで問題が発生する可能性があります。

アップグレードで問題が発生しないようにするには、次の手順を実行する必要があります。

1. **zync** Pod を 0 にスケールダウンします。
2. 3scale 2.5 へのアップグレード
3. Open ID Connect (OIDC) を使用するサービスについては、OIDC 設定を確認してください。
4. **zync** Pod をスケールアップします。

アップグレードプロセスに加えて、アプリケーションを手動で作成または更新するときに、3scale 2.5 で 4 つの OAuth フローのいずれかを設定できます。フロー設定が正しいことを確認するには、OAuth 2.0 でサポートされているフロー の手順に従ってください。

第2章 3SCALE 2.4 から 2.5 へのアップグレード

前提条件

- プロジェクトにデプロイされた 3scale 2.4。
- ツールの前提条件
  - base64
  - jq

手順

3scale API Management 2.4 を 2.5 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>
```

次に、以下の手順に従う必要があります。

- 「イメージのアップグレード」
- 「**システム** 設定マップのアップグレード」
- 「システムデータベースのシークレットフィールドのアップグレード」
- 「データベースのメンテナンスによるアカウントテーブルの修正」
- 「Zync Database PostgreSQL 9.5 の 10 へのアップグレード (強く推奨)」

2.1. 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。
   - 3scale が Oracle データベースとともにデプロイされている場合:
     a. システムイメージ 2.5.0 イメージストリームを更新します。

     ```
     $ oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP system 2.5.0"}, "from": { "kind": "DockerImage", "name": "registry.access.redhat.com/3scale-amp25/system"}, "name": "2.5.0", "referencePolicy": {"type": "Source"}}}]'
     ```

     b. **2.5** タグからフェッチするように **3scale-amp-oracle** のビルド設定を更新します。

     ```
     $ oc patch bc 3scale-amp-system-oracle --type json -p '[{"op": "replace", "path": "/spec/strategy/dockerStrategy/from/name", "value": "amp-system:2.5.0"}]'
     ```

     c. ビルドを実行します。

     ```
     $ oc start-build 3scale-amp-system-oracle --from-dir=.
     ```

   - 3scale が別のデータベースでデプロイされている場合は、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 2.5.0"}, "from": {
"kind": "DockerImage", "name": "registry.access.redhat.com/3scale-amp25/system"},
"name": "2.5.0", "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": "2.5.0"}, "name": "latest", "referencePolicy": {"type":
"Source"}}}]'
```

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 2.5.0"}, "from": { "kind":
"DockerImage", "name": "registry.access.redhat.com/3scale-amp25/apicast-gateway"},
"name": "2.5.0", "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": "2.5.0"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

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 2.5.0"}, "from": { "kind":
"DockerImage", "name": "registry.access.redhat.com/3scale-amp25/backend"}, "name":
"2.5.0", "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": "2.5.0"}, "name": "latest", "referencePolicy": {"type":
"Source"}}}]'
```

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 2.5.0"}, "from": { "kind":
"DockerImage", "name": "registry.access.redhat.com/3scale-amp25/zync"}, "name": "2.5.0",
"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": "2.5.0"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

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

```
$ oc patch imagestream/amp-wildcard-router --type=json -p '[{"op": "add", "path":
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP APIcast Wildcard
Router 2.5.0"}, "from": { "kind": "DockerImage", "name": "registry.access.redhat.com/3scale-
amp22/wildcard-router"}, "name": "2.5.0", "referencePolicy": {"type": "Source"}}}]'
$ oc patch imagestream/amp-wildcard-router --type=json -p '[{"op": "add", "path":
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP APIcast Wildcard
Router (latest)"}, "from": { "kind": "ImageStreamTag", "name": "2.5.0"}, "name": "latest",
"referencePolicy": {"type": "Source"}}}]'
```

6. 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=2.5.0
```

-

2.2. システム 設定マップのアップグレード

1. 3scale がデプロイされているプロジェクトで、**system** という名前の configmap を編集します。

```
$ $ oc edit configmap system
```

2. **rolling_updates.yml** に **service_mesh_integration** と **policy_registry** の正しい値を追加します。

```
rolling_updates.yml: |
  production:
    old_charts: false
    new_provider_documentation: false
    proxy_pro: false
    instant_bill_plan_change: false
    service_permissions: true
    async_apicast_deploy: false
    duplicate_application_id: true
    duplicate_user_key: true
    plan_changes_wizard: false
    require_cc_on_signup: false
    apicast_per_service: true
    new_notification_system: true
    cms_api: false
    apicast_v2: true
    forum: false
    published_service_plan_signup: true
    apicast_oidc: true
    policies: true
    policy_registry: true
    proxy_private_base_path: true
    service_mesh_integration: true
```

3. **system-app** および **system-sidekiq** の Pod を再起動します。

2.3. システムデータベースのシークレットフィールドのアップグレード

3scale 2.5 の変更の一環として、**system-mysql** DeploymentConfig の一部のデータベース環境変数は、値を直接設定する代わりにシークレットから割り当てられます。これらには以下が含まれます。

- MYSQL_USER 環境変数は、**システムデータベース** のシークレットフィールド DB_USER から値を取得します。
- MYSQL_PASSWORD 環境変数は、**システムデータベース** のシークレットフィールド DB_PASSWORD から値を取得します。

既存の 3scale 2.4 インストールを 2.5 にアップグレードするには、以下の手順に従います。

1. 次のコマンドが既存の Pod と DeploymentConfig を返すこと、および両方の出力が空でないことを確認します。

```
$ oc get pod | grep -i system-mysql | awk '{print $1}'
$ oc get dc system-mysql
```

2. 現在の DeploymentConfig 名、および DeploymentConfig の MySQL ユーザーと MySQL 環境変数の値を保存します。

```
MYSQL_DC="system-mysql"
RESULT_MYSQL_USER=$(oc get dc ${MYSQL_DC} -o json | jq -r
'.spec.template.spec.containers[0].env[] | select(.name == "MYSQL_USER").value')
RESULT_MYSQL_PASSWORD=$(oc get dc ${MYSQL_DC} -o json | jq -r
'.spec.template.spec.containers[0].env[] | select(.name == "MYSQL_PASSWORD").value')
```

3. RESULT_MYSQL_USER と RESULT_MYSQL_PASSWORD に既存の値があり、空でないことを確認します。

```
$ echo $RESULT_MYSQL_USER
$ echo $RESULT_MYSQL_PASSWORD
```

4. 後で参照できるように、これらの値を保存します。

5. 次のコマンドの出力を保存して、**system-mysql** 環境全体のバックアップを作成します。

```
$ oc set env "dc/${MYSQL_DC}" --list
```

6. さらに、次のコマンドの出力を保存して、**system-database** secret と **system-database** DeploymentConfig の現在の値のバックアップを作成します。

```
$ oc get secret system-database -o yaml
$ oc get dc system-mysql -o yaml
```

7. MySQL ユーザーとパスワードの現在の値を **system-database** シークレットに追加します。

```
$ oc patch secret/system-database -p "{\"stringData\": {\"DB_USER\":
\"${RESULT_MYSQL_USER}\"}}"
$ oc patch secret/system-database -p "{\"stringData\": {\"DB_PASSWORD\":
\"${RESULT_MYSQL_PASSWORD}\"}}"
```

8. シークレットが正常に編集されたことを確認します。次のコマンドは、それぞれ RESULT_MYSQL_USER と RESULT_MYSQL_PASSWORD の同じ内容を返す必要があります。

```
$ oc get secret system-database -o json | jq -r '.data["DB_USER"]' | base64 -d
$ oc get secret system-database -o json | jq -r '.data["DB_PASSWORD"]' | base64 -d
```

9. **system-mysql** DeploymentConfig を手動で編集して、MYSQL_USER および MYSQL_PASSWORD の値を新しく追加されたフィールドから **system-database** シークレットに設定します。

警告

この手順を実行すると、**system-mysql** DeploymentConfig の再デプロイがトリガーされ、Pod の再作成中に一時的にサービスが失われます。

```
$ oc edit dc $MYSQL_DC
```

10. 編集中に、**env** セクションを見つけます。以下が表示されるはずです。

```
- name: MYSQL_USER
  value: <current_mysql_user_value>
- name: MYSQL_PASSWORD
  value: <current_mysql_password_value>
```

11. そのコンテンツを次のものに置き換えます。

```
- name: MYSQL_USER
  valueFrom:
    secretKeyRef:
      key: DB_USER
      name: system-database
- name: MYSQL_PASSWORD
  valueFrom:
    secretKeyRef:
      key: DB_PASSWORD
      name: system-database
```

12. 変更を保存して終了します。

13. この後、DeploymentConfig は Pod を再デプロイします。**system-mysql** Pod が再び正しい状態で実行され、プラットフォームが再び正しく動作することを確認します。たとえば、DeploymentConfig が 1 に設定されていることを確認できます。

```
$ oc get dc ${MYSQL_DC}
```

そして、Pod は最近実行されているはずです。

```
$ oc get pods | grep -i system-mysql
```

system-mysql DeploymentConfig の環境変数がシークレットから収集されていることも確認できます。

```
$ oc set env "dc/${MYSQL_DC}" --list
```

結果の一部として、次の出力が表示されます。

```
# MYSQL_USER from secret system-database, key DB_USER
# MYSQL_PASSWORD from secret system-database, key DB_PASSWORD
```

MYSQL_USER と MYSQL_PASSWORD には特定の値は含まれていません。

その他の注意事項:

- **system-mysql** の再起動中に、**system-app**、**system-sidekiq**、および **system-sphinx** Pod のフェイルオーバーが予想される場合がありますが、それらを再デプロイする必要はおそらくありません。
- Pod の復元に失敗した場合は、対応する DeploymentConfig を手動で再デプロイします。
- フェイルオーバーを防ぐために、移行手順にメンテナンスウィンドウを追加して、**system-mysql** が再起動する直前に DeploymentConfig をゼロにスケールダウンし、再起動後に再びスケールアップして、readiness プローブを渡すことができます。

2.4. データベースのメンテナンスによるアカウントテーブルの修正

注記

これらの手順は、Oracle データベースのみを含む 3scale インストール用です。これらの制約を削除して再作成するには、重複ドメインの作成を避けるために、マルチテナントのユーザーインターフェイスと API にアクセスできないようにするメンテナンスウィンドウが必要です。

Oracle テーブルの構造に影響を与える機能強化のため、3scale 2.5 にアップグレードする際には Oracle メンテナンスウィンドウが必要です。

データベースのメンテナンスによって Accounts テーブルを修正するには、次の手順に従います。

1. system-app、system-sidekiq、および system-sphinx Pod を 0 にスケールダウンします。
2. 移行を行います。
3. 指定の Pod を再度スケールアップします。

2.5. ZYNC DATABASE POSTGRESQL 9.5 の 10 へのアップグレード (強く推奨)

3scale 2.5 の変更の一環として、PostgreSQL のバージョンが 9 から 10 に更新されています。Red Hat Software Collections Product Life Cycle に示されているように、Postgresql 9.5 のサポートは 2019 年 5 月まで利用できるため、Zync データベースのアップグレードを強くお勧めします。

2.5.1. OpenShift オブジェクトのバックアップ作成

1. アップグレードプロセスをサポートする手順として、次のコマンドの出力を保存して、**postgresql** ImageStream と **zync-database** DeploymentConfig の現在の値を含むバックアップファイルを作成します。

```
$ oc get imagestream postgresql -o yaml
$ oc get dc zync-database -o yaml
```

2. 既存の PostgreSQL ImageStreamTag の ImportPolicy の現在の値を取得します。

```
IMPORT_POLICY_VAL=$(oc get imagestream postgresql -o json | jq -r
```

```
".spec.tags[0].importPolicy.insecure")
if [ "$IMPORT_POLICY_VAL" == "null" ]; then
  IMPORT_POLICY_VAL="false"
fi
```

3. 後で参照できるように、結果が **true** または **false** であることを確認してください。

```
$ echo $IMPORT_POLICY_VAL
```

2.5.2. データベースのバックアップ作成

1. **zync** DeploymentConfig を 0 Pod にスケールダウンします。
2. 次のコマンドを実行して、**zync-database** データベースのバックアップを実行します。

```
$ oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq
'.items[0].metadata.name' -r) bash -c 'pg_dumpall' | gzip - > zync-database-backup-for-2.5.gz
```

2.5.3. OpenShift オブジェクトの新規作成

1. 新しい PostgreSQL 10 ImageStreamTag を **postgresql** ImageStream に追加します。

```
$ oc tag --source=docker registry.access.redhat.com/rhscl/postgresql-10-rhel7 postgresql:10
--insecure=${IMPORT_POLICY_VAL}
```

2. PostgreSQL の **ImageStream** タグが **postgresql** ImageStream に追加されていることを確認します。

```
$ oc get imagestream postgresql -o yaml
```

ステータスセクションに次のフィールドとその値を持つ項目が含まれていることが判明するまで、このコマンドを実行し続けます。

- **tag** = 10
- **image** = sha256:<sha256identifier> 形式の値
  または、追加の検証を実行するには、次を実行することもできます。

  ```
  $ oc get imagestream postgresql
  ```

  タグフィールドに 10 と 9.5 がコンマで区切られていることを確認します。

  確認できるもう 1 つの要素は、対応する ImageStreamTag が作成されていることです。

  ```
  $ oc get imagestreamtag postgresql:10
  ```

  結果が返され、**DOCKER REF** フィールドに特定の sha256 値を持つ PostgreSQL 10 の URL が含まれていることを確認します。

3. PostgreSQL10 で Zync-Database deploymentConfig の新しいデプロイメントをトリガーするには、**zync-database** DeploymentConfig を手動で編集します。

```
$ oc edit dc zync-database
```

4. **triggers** セクションに移動し、**imageChangeParams** の下の **name** フィールドを探します。トリガーが PostgreSQL 9.5 ImageStreamTag を参照していることがわかります。以下のような出力が表示されるはずです。

```
triggers:
- type: ConfigChange
- imageChangeParams:
    automatic: true
    containerNames:
    - postgresql
    from:
      kind: ImageStreamTag
      name: postgresql:9.5
      namespace: <yournamespace>
```

5. **name** フィールドを **postgresql:9.5** から **postgresql:10** に変更します。

6. 変更を保存し、エディターを終了します。これにより、変更がトリガーされます。既存の Pod が終了します。終了したら、この Pod は新しい PostgreSQL イメージを使用して新しい Pod を開始します。

7. **zync-database** Pod が再作成され、PostgreSQL10 イメージを使用し、正しく動作していることを確認するには、次のコマンドを実行できます。

```
$ oc get deploymentconfig zync-database
```

そして、現在の値と目的の値がいずれも 1 であることを確認します。

8. また、zync-database Pod のステータスが Running であり、最近作成されたことを確認します。

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

9. 最後に、Pod が PostgreSQL イメージを使用していることを確認します。

```
$ oc get pod $(oc get pods | grep -i zync-database | awk '{print $1}') -o json | jq
'.spec.containers[0]'.image
```

postgresql10 イメージ URL が参照されていることがわかります。

2.5.4. データベースのバックアップのインポート

1. 前に作成したデータベースのバックアップをインポートするには、次のコマンドを実行します。

```
$ zcat zync-database-backup-for-2.5.gz | oc rsh $(oc get pods -l 'deploymentConfig=zync-
database' -o json | jq '.items[0].metadata.name' -r) bash -c 'psql -d postgres -f -'
```

2. zync DeploymentConfig を元にスケールアップします。

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

2.5.5. 古い Image タグのクリーンアップ

1. すべての Pod が新しいデータベースバージョンを正しく使用している場合は、古い PostgreSQL 9.5 ImageStreamTag を削除します。

```
$ oc tag -d postgresql:9.5
```

2. 次のコマンドを実行して、正しく削除されたことを確認できます。

```
$ oc get imagestream postgresql
```

タグフィールドに 10 が含まれていることを確認します。

確認できるもう 1 つの要素は、対応する ImageStreamTag が削除されていることです。

```
$ oc get imagestreamtag postgresql:9.5
```

この場合、**error not found** というメッセージが表示されます。

2.5.6. ロールバック

ロールバックを実行する必要がある場合は、次のことを行う必要があります。

1. **zync** DeploymentConfig を 0 Pod にスケールダウンします。
2. 以下を使用して、PostgreSQL イメージを 9.5 にロールバックします。

```
$ oc edit dc zync-database
```

3. **triggers** セクションに移動し、**imageChangeParams** の下の **name** フィールドを探します。トリガーが PostgreSQL 10 ImageStreamTag を参照していることがわかります。
4. name フィールドを **postgresql:10** から **postgresql:9.5** に変更します。これにより、変更がトリガーされます。既存の Pod が終了します。終了したら、この Pod は新しい PostgreSQL イメージを使用して新しい Pod を開始します。
5. 「データベースのバックアップのインポート」セクションで説明されているのと同じ手順に従って、データベースバックアップを再インポートします。
6. 最後に、zync DeploymentConfig を元にスケールアップします。