Red Hat 3scale API Management 2.5

Migrating 3scale

Upgrade your 3scale API Management installation.

Last Updated: 2022-12-13

Red Hat 3scale API Management 2.5 Migrating 3scale

Upgrade your 3scale API Management installation.

Legal Notice

Abstract

This guide provides the information to upgrade your 3scale API Management installation to the latest version.

Table of Contents

PREFACE

This guide will help you to upgrade 3scale API Management.

PART I. 3SCALE API MANAGEMENT 2.4 TO 2.5 UPGRADE GUIDE

This section contains information about upgrading Red Hat 3scale API Management from version 2.4 to 2.5.

WARNING

This process causes disruption in the service. Make sure to have a maintenance window.

CHAPTER 1. BEFORE YOU BEGIN: DEPLOY OAUTH FLOWS PER SERVICE

In previous versions to 3scale 2.5, when you manually created or updated an application, the OAuth flow enabled by default by 3scale in Red Hat Single Sign On (RH-SSO) was the *standardFlowEnabled* (Authorization Code Flow). If you changed it to another flow such as Resource Owner Password, Implicit or Client Credentials, you could potentially have issues with this upgrade process.

To avoid having issues with the upgrade, you must perform these steps:

1. Scale down the **zync** pod to 0.
2. Upgrade to 3scale 2.5
3. For the services that use Open ID Connect (OIDC), review the OIDC configuration.
4. Scale up the **zync** pod.

Besides the upgrade process, you can set any of the four OAuth flows in 3scale 2.5 when manually creating or updating your application. To ensure correct flow settings, you can follow the steps in OAuth 2.0 supported flows.

CHAPTER 2. UPGRADING 3SCALE 2.4 TO 2.5

Prerequisites

- 3scale 2.4 deployed in a project.
- Tool prerequisites:
  - base64
  - jq

Procedure

To upgrade Red Hat 3scale API Management 2.4 to 2.5, go to the project where 3scale is deployed.

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

Then, you need to follow these steps:

- Section 2.1, “Upgrade images”
- Section 2.2, “Upgrade **system** configMap”
- Section 2.3, “Upgrade system-database secret fields”
- Section 2.4, “Fix Accounts table through database maintenance”
- Section 2.5, “Upgrade Zync Database PostgreSQL 9.5 to 10 (Highly recommended)”

2.1. UPGRADE IMAGES

1. Patch the **amp-system** image stream.
   - If 3scale is deployed with Oracle Database,
     a. Update the system image 2.5.0 image stream:

     ```
     $ 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. Update the build configuration for **3scale-amp-oracle** to fetch from the **2.5** tag:

     ```
     $ 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. Run the build:

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

   - If 3scale is deployed with a different database, use the following commands:

     ```
     $ 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. Patch the **amp-apicast** image stream.

```
$ 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. Patch the **amp-backend** image stream.

```
$ 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. Patch the **amp-zync** image stream.

```
$ 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. Patch the **amp-wildcard-router** image stream.

```
$ 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. Update the visible release version.

```
$ oc set env dc/system-app AMP_RELEASE=2.5.0
```

2.2. UPGRADE SYSTEM CONFIGMAP

1. In the project where 3scale is deployed, edit the configmap named **system**.

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

2. Add to *rolling_updates.yml* the correct value for **service_mesh_integration** and **policy_registry**.

```
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. Restart these pods: **system-app** and **system-sidekiq**.

2.3. UPGRADE SYSTEM-DATABASE SECRET FIELDS

As part of the changes of 3scale 2.5, some database environment variables in the **system-mysql** DeploymentConfig are assigned from secrets instead of having the values directly set. These include:

- MYSQL_USER environment variable gets its value from the **system-database** secret field DB_USER.
- MYSQL_PASSWORD environment variable gets its value from the **system-database** secret field DB_PASSWORD.

To upgrade the existing 3scale 2.4 installation to 2.5, follow the procedure below:

1. Verify that the following commands return an existing pod and DeploymentConfig, as well as that the output is not empty for both of them:

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

2. Save the current DeploymentConfig name, and values of the MySQL user and MySQL environment variables of the DeploymentConfig:

```
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. Verify that RESULT_MYSQL_USER and RESULT_MYSQL_PASSWORD have existing values and are not empty:

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

4. Save these values for future references.

5. Create a backup of the entire **system-mysql** environment by saving the output of the following command:

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

6. Additionally, create a backup the current values of the **system-database** secret and the **system-database** DeploymentConfig by saving the output of the following commands:

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

7. Add the current values of the MySQL user and password into the **system-database** secret:

```
$ 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. Verify that the secret has successfully been edited. The following commands should return the same contents of RESULT_MYSQL_USER and RESULT_MYSQL_PASSWORD respectively:

```
$ 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. Manually edit the **system-mysql** DeploymentConfig to set the values of MYSQL_USER and MYSQL_PASSWORD from the newly added fields to the **system-database** secret.

WARNING

Performing this step will trigger a redeployment of the **system-mysql** DeploymentConfig, causing temporarily service loss while the pod is recreated.

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

10. While editing, find the **env** section. You should find:

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

11. Replace that content with:

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

12. Save the changes and exit.

13. After this, the DeploymentConfig will redeploy the pod. Verify that the **system-mysql** pod is running again and in a correct state and that the platform works correctly again. For example you can check that the DeploymentConfig is set to 1:

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

And the pod should be running recently:

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

You can also verify that the environment variables of the system-mysql DeploymentConfig are now gathered from a secret:

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

You should see as part of the result the following output:

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

MYSQL_USER and MYSQL_PASSWORD do not contain specific values.

Additional notes:

- Failovers in **system-app**, **system-sidekiq** and **system-sphinx** pods may be expected while **system-mysql** restarts, but probably no need to redeploy them.
- In case the pods fail to recover, then redeploy the corresponding DeploymentConfigs manually.

- To prevent the failover, a maintenance window can be added to the migration procedure, scaling those DeploymentConfigs down to zero right before **system-mysql** reboots and up again after it is back, passing the readiness probe.

2.4. FIX ACCOUNTS TABLE THROUGH DATABASE MAINTENANCE

NOTE

These steps are for 3scale installations containing only Oracle databases. Removing and recreating these constraints need a maintenance window where the multitenant user interface and API must not be accessible to avoid creation of duplicate domain.

Due to an enhancement affecting the structure of Oracle tables, it is necessary to have a Oracle maintenance window when upgrading to 3scale 2.5.

To fix the Accounts table through database maintenance, follow these steps:

1. Scale down system-app, system-sidekiq and system-sphinx pods to 0.
2. Make the migration.
3. Scale back up the indicated pods.

2.5. UPGRADE ZYNC DATABASE POSTGRESQL 9.5 TO 10 (HIGHLY RECOMMENDED)

As part of the changes of 3scale 2.5, PostgreSQL version has been updated from 9 to 10. Upgrade Zync Database is highly recommended because support for Postgresql 9.5 will be available until May 2019, as indicated in the Red Hat Software Collections Product Life Cycle .

2.5.1. Create a backup of OpenShift objects

1. As a step to support the upgrading process, create a backup file containing the current values of the **postgresql** ImageStream and the **zync-database** DeploymentConfig by saving the output of the following commands:

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

2. Get the current value of the ImportPolicy of the existing PostgreSQL ImageStreamTag:

```
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. For later reference, check that the result is either *true* or *false* .

```
$ echo $IMPORT_POLICY_VAL
```

2.5.2. Create a backup of the database

1. Scale down the **zync** DeploymentConfig to 0 pods.

2. Run the following command to perform a backup of the **zync-database** database:

```
$ 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. Create new OpenShift objects

1. Add the new PostgreSQL 10 ImageStreamTag to the **postgresql** ImageStream:

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

2. Verify that the *ImageStream* tag for PostgreSQL has been added to the **postgresql** ImageStream:

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

   Keep running this command until you see that the status section contains an item with the following fields with their values:

   - **tag** = 10
   - **image** = value of the form sha256:<sha256identifier>
     Alternatively, to perform an additional verification you can also run:

```
$ oc get imagestream postgresql
```

     And verify that the tag field contains '10' and '9.5', separated by a comma.

     Another element you can verify is that the corresponding ImageStreamTag has been created:

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

     Verify that a result is returned and that the **DOCKER REF** field should have the URL of the PostgreSQL 10 with a specific sha256 value.

3. To trigger a new deployment of the Zync-Database deploymentConfig with PostgreSQL10, manually edit the **zync-database** DeploymentConfig:

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

4. Go to the *triggers* section, and look for the **name** field under *imageChangeParams*. There you should see that the trigger references the PostgreSQL 9.5 ImageStreamTag. You should see something like this:

```
triggers:
- type: ConfigChange
- imageChangeParams:
    automatic: true
    containerNames:
```

```
    - postgresql
    from:
      kind: ImageStreamTag
      name: postgresql:9.5
      namespace: <yournamespace>
```

5. Change the **name** field from **postgresql:9.5** to **postgresql:10**.

6. Then save the changes and exit the editor. This will trigger the changes: The existing pod will be terminated and, when it is terminated, this pod will start the new pod using the new PostgreSQL image.

7. To verify that the **zync-database** pod is recreated, is using the PostgreSQL10 image and is working correctly, you can run this command:

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

   And check that the current and desired values are both 1.

8. Also check that the zync-database pod is in status Running and has been recently created with:

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

9. Finally check that the Pod is using the PostgreSQL image:

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

   You should see that a **postgresql10** image URL is being referenced.

2.5.4. Import the database backup

1. To import the backup of the database you created earlier, run this command:

```
$ 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. Scale back up zync DeploymentConfig back.

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

2.5.5. Cleanup of the old *Image* tag

1. If all the pods are correctly using the new database version, remove the old PostgreSQL 9.5 ImageStreamTag:

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

2. You can verify it has been correctly removed by running:

```
$ oc get imagestream postgresql
```

   Verify that the tag field contains 10.

Another element you can verify is that the corresponding ImageStreamTag has been removed:

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

In this case, you will get an *error not found* message.

2.5.6. Rollback

In case you need to perform a rollback, you should:

1. Scale down the **zync** DeploymentConfig to 0 pods.
2. Rollback the PostgreSQL Image used to 9.5 with:

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

3. Go to the *triggers* section, and look for the **name** field under *imageChangeParams*. There you should see that the trigger references the PostgreSQL 10 ImageStreamTag.
4. Change the name field from **postgresql:10** to **postgresql:9.5**. This will trigger the changes: The existing pod will be terminated and, when this pod is terminated, it will start the new pod using the new PostgreSQL image.
5. Reimport the database backup following the same steps as the ones described in the Section 2.5.4, “Import the database backup” section.
6. Finally, scale up zync DeploymentConfig back.