Red Hat 3scale API Management 2.7

Migrating 3scale

Upgrade your 3scale API Management installation.

Last Updated: 2022-12-13

Red Hat 3scale API Management 2.7 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.

By upgrading 3scale 2.6 to 2.7, the following will happen:

- Each existing API will be split into one API product and one API backend.
- All existing mapping rules, metrics, and methods will be defined at the product level.
- The private URL of the API will move to the API backend level.
- The API backend usage in the API product will have '/' as the path.
- This change will not affect the users of your API, because all endpoints stay the same.

For more details about products and backends, refer to the *Release Notes for Red Hat 3scale API Management 2.7 On-premises*, under Major features.

PART I. 3SCALE TEMPLATE-BASED UPGRADE GUIDE: FROM 2.6 TO 2.7

This section contains information about upgrading Red Hat 3scale API Management from version 2.6 to 2.7, in a template-based deployment.

WARNING

This process causes disruption in the service until the upgrade is finished. Make sure to have a maintenance window.

CHAPTER 1. BEFORE YOU BEGIN

Prior to proceeding with the upgrade, you must consider these points:

- 3scale supports upgrade paths from 2.6 to 2.7 with templates on OpenShift 3.11.
- Ensure your OpenShift CLI tool is configured in the same project where 3scale is deployed.
- Run the commands below in the bash shell.
- Perform a backup of the databases. The procedure of the backup is specific to each database type and setup.

CHAPTER 2. UPGRADING 3SCALE 2.6 TO 2.7 USING TEMPLATES

Prerequisites

- 3scale 2.6 deployed with templates in an OpenShift 3.11 project.
- Tool prerequisites:
  - base64
  - jq

Procedure

To upgrade Red Hat 3scale API Management 2.6 to 2.7 using templates, go to the project where 3scale is deployed.

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

Then, follow these steps in this order:

1. Section 2.1, “Create a back-up of the 3scale project”
2. Section 2.2, “Update the pre-hook pod from system”
3. Section 2.3, “Upgrade 3scale images”
4. Section 2.4, “Migrate from WildcardRouter to Zync Route Management”

2.1. CREATE A BACK-UP OF THE 3SCALE PROJECT

1. Depending on the database used with 3scale, set **${SYSTEM_DB}** with one of the following values:
   - If the database is MySQL, **SYSTEM_DB=system-mysql**.
   - If the database is PostgreSQL, **SYSTEM_DB=system-postgresql**.
2. Create a back-up file with the existing DeploymentConfigs:

   ```
   $ 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. Backup all existing OpenShift resources in the project that are exported through the **export all** command:

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

4. Create a back-up file with the additional elements that are not exported with the **export all** command:

   -

```
$ 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. Verify that all generated files are not empty and that all of them have the expected content.

2.2. UPDATE THE PRE-HOOK POD FROM SYSTEM

To add post-migration actions required for the upgrade, you must update the pre-hook pod command from system.

1. Get the current MASTER_ACCESS_TOKEN value:

```
$ MASTER_ACCESS_TOKEN=$(oc get secret system-seed -o json | jq -r
.data.MASTER_ACCESS_TOKEN | base64 -d)
```

2. Verify that MASTER_ACCESS_TOKEN is not empty and has the existing value:

```
$ echo ${MASTER_ACCESS_TOKEN}
```

3. Update the pre-hook pod command from **system-app** DeploymentConfig to the new command needed for this release:

```
$ oc patch dc/system-app -p "{\"spec\":{\"strategy\":{\"rollingParams\":{\"pre\":
{\"execNewPod\":{\"command\":[\"bash\",\"-c\",\"bundle exec rake boot openshift:deploy
MASTER_ACCESS_TOKEN=\\\"${MASTER_ACCESS_TOKEN}\\\" && bundle exec rake
services:create_backend_apis services:update_metric_owners
proxy:update_proxy_rule_owners\"]}}}}}}"
```

4. Verify that pre-hook pod command has changed to the new value:

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

- The result of the previous command should be:

```
[
  "bash",
  "-c",
  "bundle exec rake boot openshift:deploy MASTER_ACCESS_TOKEN=\"<your-master-
access-token>\" && bundle exec rake services:create_backend_apis
services:update_metric_owners proxy:update_proxy_rule_owners"
]
```

2.3. UPGRADE 3SCALE IMAGES

This section describes the steps required to upgrade the 3scale images.

2.3.1. Procedure

1. Patch the **amp-system** image stream:
   To patch the **amp-system** image stream, you need to consider the database used with your 3scale deployment.

   - If 3scale is deployed with Oracle Database, perform these steps to build the system image with an Oracle Database: 1, 2, 4, 8 and 9.

   - If the database is different, use this command:

     ```
     $ oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP system 2.7"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/system-rhel7:3scale2.7"}, "name": "2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
     ```

     This triggers redeployments of **system-app**, **system-sphinx** and **system-sidekiq** DeploymentConfigs. Wait until they are redeployed, its corresponding new pods are ready, and the old ones terminated.

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 2.7"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/apicast-gateway-rhel7:3scale2.7"}, "name": "2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
   ```

   This triggers redeployments of **apicast-production** and **apicast-staging** DeploymentConfigs. Wait until they are redeployed, its corresponding new pods are ready, and the old ones terminated.

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 2.7"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/backend-rhel7:3scale2.7"}, "name": "2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
   ```

   This triggers redeployments of **backend-listener**, **backend-worker**, and **backend-cron** DeploymentConfigs. Wait until they are redeployed, its corresponding new pods are ready, and the old ones terminated.

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 2.7"}, "from": { "kind":
   ```

```
"DockerImage", "name": "registry.redhat.io/3scale-amp2/zync-rhel7:3scale2.7"}, "name":
"2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This triggers redeployments of **zync** and **zync-que** DeploymentConfigs. Wait until they are redeployed, its corresponding new pods are ready, and the old ones terminated.

5. Patch the **system-memcached** ImageStream:

```
$ oc patch imagestream/system-memcached --type=json -p '[{"op": "add", "path":
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.7
Memcached"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-
amp2/memcached-rhel7:3scale2.7"}, "name": "2.7", "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": "2.7"}, "name": "latest",
"referencePolicy": {"type": "Source"}}}]'
```

This triggers redeployment of the **system-memcache** DeploymentConfig. Wait until they are redeployed, its corresponding new pods are ready, and the old ones terminated.

6. Patch the **zync-database-postgresql** image stream:

```
$ oc patch imagestream/zync-database-postgresql --type=json -p '[{"op": "add", "path":
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "Zync 2.7 PostgreSQL"},
"from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"},
"name": "2.7", "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": "2.7"}, "name": "latest",
"referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **zync-database-postgresql** image stream to contain the 2.7 tag. You can confirm that the 2.7 tag has been created with this command:

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

- This patch might also trigger a redeployment of the **zync-database** DeploymentConfig in case there are new updates on the image. If this happens, wait until the new pods are redeployed and ready, and the old pods terminated.

7. If one or more of the following DeploymentConfigs exist in your 3scale 2.6 installation, click the link for the DeploymentConfigs that apply to obtain more information on how to proceed:

- Section 2.4.1.1, "**backend-redis** DeploymentConfig"
- Section 2.4.1.2, "**system-redis** DeploymentConfig"
- Section 2.4.1.3, "**system-mysql** DeploymentConfig"
- Section 2.4.1.4, "**system-postgresql** DeploymentConfig"

8. Update the visible release version:

```
$ oc set env dc/system-app AMP_RELEASE=2.7
```

This triggers a redeployment of the **system-app** DeploymentConfig. Wait until it is performed and its corresponding new pods are ready and the old ones terminated.

9. Verify that all the image URLs of the DeploymentConfigs contain the new image registry URLs, with a hash added at the end of each URL:

```
$ 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"

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
```

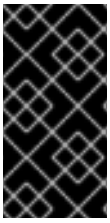
2.4. MIGRATE FROM WILDCARDROUTER TO ZYNC ROUTE MANAGEMENT

In 3scale 2.6, the WildcardRouter component and wildcard OpenShift routes have been removed and are now created as individual OpenShift routes managed by the Zync subsystem. This procedure details the migration of route management from WildcardRouter to Zync.

At this point, all 3scale images have been upgraded to 3scale 2.6. Creation and deletion of OpenShift routes corresponding to 3scale services and tenants are automatically managed by the Zync subsystem. Moreover, all the new Zync infrastructure needed to do so are available by the addition of new OpenShift elements that has been done in previous sections.

To migrate the OpenShift route management from WildcardRouter to Zync, the old 3scale tenants and services related to OpenShift routes and wildcard routes must be removed, and then a forced reevaluation of existing services and tenants by Zync must be performed. This will make Zync create equivalent routes to what you currently have.

When the Public Base URL is modified, an event is triggered in **system-app** and notifies **system-sidekiq** via the job queue stored in **system-redis**. The job is then processed in the background and sent to zync where it checks if the data exists already or not on the **zync-db**. If it detects changes, it creates a new route via a job processed in **zync-que**.



IMPORTANT

Before doing anything, if you have manually installed SSL certificates into some routes, you must copy the certificates assigned to the routes and keep note of what routes was each certificate assigned to. You will have to install them into the new equivalent routes that will be created by Zync, in case you want to keep the certificates functionality.

NOTE

- Your services is migrated by default with the option *hosted*.
- The Public Base URL will be populated automatically and the routes will be created by Zync.
- Step 1 is only necessary when configuring external APIcast gateways with the option *self_managed*.
- If you select the *3scale_managed* option, routes are managed automatically by Zync.

Procedure

1. Given that Zync does not manage the routes of external gateways, you can modify the deployment option of each service not managed by Zync, by following the steps under one of the proposed alternatives:
   - In 3scale:
     a. Go to the **Integration** page, and click **edit integration settings**.
     b. Choose the correct deployment option, and save your changes if any.
   - Using the API:
     a. Update the service with a service identifier (**ID**) with an access token (**ACCESS_TOKEN**) and the tenant endpoint (**TENANT_URL**):

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

     Alternatively, you can use the command below if you are using APIcast hosted:

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

     b. For each service of each tenant, modify the **deployment_option** field via 3scale or the API:
        - These are the cases where you can set **deployment_option** to **self_managed**:
          - APIcast is linked to a custom route in OpenShift.
          - APIcast is hosted outside of OpenShift.
          - **APICAST_PATH_ROUTING** is set to **true**.
        - In other cases, set **deployment_option** to **hosted**.
2. Among the potentially existing routes, some default routes were automatically created in 2.5 by 3scale. Start by removing them:

```
$ 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
```

- In case you deployed 3scale 2.5 with **WILDCARD_POLICY=Subdomain** you must remove the wildcard route with:

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

- Otherwise, if you deployed 3scale 2.5 without **WILDCARD_POLICY=Subdomain**, you must remove the routes you manually created for the 3scale tenants and services, to avoid having duplications of the routes that Zync will create.

At this point, all the routes related to services and tenants must have been removed. Now, you will perform the creation of equivalent routes by Zync:

1. Force the resync of all 3scale services and tenants OpenShift routes with Zync:

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

2. Check that SYSTEM_SIDEKIQ_POD environment variable result is not empty:

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

3. Finally, perform the resynchronization:

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

You will see output of this style with information about notifications to system:

```
$ 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
…
```

New routes are created for all the existing tenants and services, after forcing Zync to reevaluate them. Route creation might take some minutes depending on the number of services and tenants.

By the end of the process, you will see created:

- One Master Admin Portal route.
  For every 3scale tenant two routes are created:

- Tenant's Admin Portal route.

- Tenant's Developer Portal route.
  For every 3scale service two routes are created:
- APIcast staging Route corresponding to the service.
- APIcast production Route corresponding to the service.

4. Verify that all the expected routes explained above have been created for all your existing services and tenants. You can see all the routes by running:

```
$ oc get routes
```

   The host/port field shown as the output of the previous command must be the URL of the routes.

   - In case you deployed 3scale 2.5 with the WILDCARD_POLICY set to **Subdomain**, all of the new routes must have the same base WildcardDomain as the old OpenShift wildcard Route.
   - Otherwise, in case you deployed 3scale 2.5 without WILDCARD_POLICY=Subdomain the new routes must have the same host as the old routes that you have removed, including the ones that were automatically created by 3scale in the 2.5 release.

5. Finally, in case you were using custom SSL certificates for the old wildcard route, or the old manually created routes, install them into the new ones created by Zync. You can do so by editing the routes via the OpenShift web panel and adding the certificate/s into them.

6. Verify that Services and Tenants that existed before this migration are still resolvable using the new routes. To do so perform the following tests:

   a. Resolve the route of an existing APIcast production URL associated to a 3scale service that already existed before this migration.

   b. Resolve the route of an existing APIcast staging URL associated to a 3scale service that already existed before this migration.

   c. Resolve the route of an existing Tenant that already existed before this migration.

7. When verifying that the new Zync functionality is working, confirm that new routes are generated when creating new tenants and services. To do so perform the following tests:

   a. Create a new tenant from the 'master' panel and verify that after some seconds the new Routes associated to it appear in OpenShift.

   b. Create a new Service in one of your existing tenants and verify that after some seconds the new Routes associated to it appear in OpenShift.

8. Remove the apicast-wildcard-router service:

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

9. Remove the deprecated WildcardRouter subsystem:

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

After you have performed all the listed steps, 3scale upgrade from 2.6 to 2.7 in a template-based deployment is now complete.

2.4.1. Additional steps with existing DeploymentConfigs

2.4.1.1. backend-redis DeploymentConfig

If the **backend-redis** DeploymentConfig exists in your current 3scale installation, patch the **backend-redis** image stream:

```
$ oc patch imagestream/backend-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "Backend 2.7 Redis"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/redis-32-rhel7:3.2"}, "name": "2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **backend-redis** image stream to contain the **2.7** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.7**:

```
$ oc get is backend-redis
```

- This patch might also trigger a redeployment of the **backend-redis** DeploymentConfig in case there are new updates on the image. If this happens, wait until the new pods are redeployed and ready, and the old pods terminated.

Continue upgrading 3scale images.

2.4.1.2. system-redis DeploymentConfig

If the **system-redis** DeploymentConfig exists in your current 3scale installation, patch the **system-redis** image stream:

```
$ oc patch imagestream/system-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.7 Redis"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/redis-32-rhel7:3.2"}, "name": "2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **system-redis** image stream to contain the **2.7** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.7**:

```
$ oc get is system-redis
```

- This patch might also trigger a redeployment of the **system-redis** DeploymentConfig in case there are new updates on the image. If this happens, wait until the new pods are redeployed and ready, and the old pods terminated.

Continue upgrading 3scale images.

2.4.1.3. system-mysql DeploymentConfig

If the **system-mysql** DeploymentConfig exists in your current 3scale installation, patch the **system-mysql** image stream:

```
$ oc patch imagestream/system-mysql --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.7 MySQL"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/mysql-57-rhel7:5.7"}, "name": "2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **system-mysql** image stream to contain the **2.7** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.7**:

```
$ oc get is system-mysql
```

- This patch might also trigger a redeployment of the **system-mysql** DeploymentConfig in case there are new updates on the image. If this happens, wait until the new pods are redeployed and ready, and the old pods terminated.

Continue upgrading 3scale images.

2.4.1.4. system-postgresql DeploymentConfig

If the **system-postgresql** DeploymentConfig exists in your current 3scale installation, patch the **system-postgresql** image stream:

```
$ oc patch imagestream/system-postgresql --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.7 PostgreSQL"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"}, "name": "2.7", "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": "2.7"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **system-postgresql** image stream to contain the **2.7** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.7**:

```
$ oc get is system-postgresql
```

- This patch might also trigger a redeployment of the **system-postgresql** DeploymentConfig in case there are new updates on the image. If this happens, wait until the new pods are redeployed and ready, and the old pods terminated.

Continue upgrading 3scale images.

PART II. 3SCALE OPERATOR-BASED UPGRADE GUIDE: FROM 2.6 TO 2.7

This section contains information about upgrading Red Hat 3scale API Management from version 2.6 to 2.7 via the 3scale operator.

WARNING

This process causes disruption in the service until the upgrade is finished. Make sure to have a maintenance window.

CHAPTER 3. UPGRADING 3SCALE 2.6 TO 2.7

To upgrade 3scale from version 2.6 to 2.7 in an operator-based deployment, you must first upgrade the 3scale operator.

3.1. PREPARING THE APIMANAGER OBJECTS

Procedure

1. Edit the *APIManager* object by adding the following annotations:

NOTE

The *APIManager* object is your 3scale installation that you intend to upgrade.

```
apps.3scale.net/apimanager-threescale-version: "2.6"
apps.3scale.net/threescale-operator-version: "0.3.0"
```

2. Get all existing APIManagers in an existing OpenShift project:

```
oc get apimanagers
```

3. Edit the APIManager:

```
oc annotate apimanager <apimanager-name>
```

4. Add the previous tags in the annotations section when editing the object.
5. If the annotations section is not available, create this section.
6. Save the changes to finish editing.

3.2. UPGRADING 3SCALE 2.6 TO 2.7 VIA THE OPERATOR UPGRADE

This applies to the deployment of 3scale using operators.

Prerequisites

- 3scale 2.6 previously deployed via the 3scale operator.
- An OpenShift Container Platform 4 cluster with administrator access.
  - For more information about supported configurations, see the Red Hat 3scale API Management Supported Configurations page.

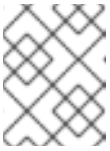
Procedure

1. Open the OpenShift user interface.
2. Select the project where the APIManager, that is, your operator-based 3scale installation and 3scale operator are deployed.

3. The menu structure depends on the OpenShift version you are using:
   - For OCP 4.1, click **Catalog > Operator Management**
   - For OCP 4.2, click **Operators > Installed Operators**
4. Select the 3scale *Operator Subscription.*
5. Edit the channel of that subscription, select the *threescale-2.7* channel and save the changes.
   - This should start the upgrade process.
   - Wait until the upgrade procedure finishes for the APIManager.
6. Query the pods status on the project:

```
oc get pods
```

   - Wait until all the new versions are running and ready without errors.
   - They might have errors temporarily during the upgrade process.



NOTE

Times can vary from 5-10 minutes approximately. Be sure to keep checking the state of the pods until all of them are running, ready, and without errors.

7. To confirm that the upgrade process has been successful, log in to the 3scale Admin Portal and check that it works as expected.
   After you have performed all the listed steps, the 3scale upgrade via the 3scale operator from 2.6 to 2.7 is now complete.

PART III. 3SCALE API MANAGEMENT MIGRATION GUIDE: FROM TEMPLATE TO OPERATOR-BASED DEPLOYMENTS

This section contains information about migrating Red Hat 3scale API Management from a template-based deployment using Red Hat OpenShift 3.11, to an operator-based deployment using Red Hat OpenShift 4.x.

WARNING

In order to understand the required conditions and procedure, read the entire migration guide before applying the listed steps. The migration process disrupts the provision of the service until the procedure finishes. Due to this disruption, make sure to have a maintenance window.

CHAPTER 4. GETTING READY FOR THE MIGRATION

Before migrating your 3scale installation from a template to an operator-based deployment, confirm that your deployment is supported by consulting the following guides:

- Backing up 3scale a template-based deployment.
- Restoring the backup in an operator-based deployment.

CHAPTER 5. MIGRATING 3SCALE TEMPLATE TO OPERATOR-BASED DEPLOYMENTS

Prerequisites

- Red Hat 3scale API Management 2.7 deployed in both environments.
- A domain for each OpenShift cluster, and another WILDCARD_DOMAIN for 3scale. Examples:
  - Red Hat OpenShift 3.11 (OCP3): **ocp3.example.com**
  - Red Hat OpenShift 4.x (OCP4): **ocp4.example.com**
  - 3scale: **3scale.example.com**

Procedure

The basic setup before migration is that 3scale points to the OCP3 domain: **3scale.example.com → ocp3.example.com**

To migrate 3scale from a template-based deployment using Red Hat OpenShift 3.11 to an operator-based deployment using Red Hat OpenShift 4.1, follow these steps:

1. Create a 3scale backup from the template-based deployment.
2. Deploy 3scale using the operator.
3. Restore the backup in the operator-based deployment.
4. Point the 3scale WILDCARD_DOMAIN, in this case **3scale.example.com**, to **ocp4.example.com**.

After you have performed all the listed steps, 3scale migration from a template to an operator-based deployment is now complete.