Red Hat 3scale API Management 2.6

Migrating 3scale

Upgrade your 3scale API Management installation.

Last Updated: 2023-03-13

Red Hat 3scale API Management 2.6 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 UPGRADE GUIDE: FROM 2.5 TO 2.6

This section contains information about upgrading Red Hat 3scale API Management from version 2.5 to 2.6, in a template-based deployment.

WARNING

This process causes disruption in the service and temporarily stops Zync processing jobs 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:

Supported configurations

- 3scale supports upgrade paths from 2.5 to 2.6 with **templates only**, on OpenShift 3.11.

Previous 3scale versions

- Assuming 3scale 2.5 was deployed with the **amp.yml** standard scenario template, download the new 3scale 2.6 **amp.yml** template and then deploy it to create the new OpenShift elements.
  - To download the 3scale 2.6 **amp.yml** template, see Configuring nodes and entitlements.
- For multi-version upgrades from earlier versions than 2.4, confirm the existence of the **system-environment** ConfigMap:

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

  - If you get **NotFound** error message, refer to the 2.4 Upgrade Guide, under Creating ConfigMaps.

Tooling

- Perform a backup of the databases. The procedure of the backup is specific to each database type and setup.
- Ensure your OpenShift CLI tool is configured in the same project where 3scale is deployed.
- Run the commands below in the bash shell.
- For this upgrade, download and get the patch files by following these steps:
  1. Click templates-migration-2.5-to-2.6.
  2. Download and extract the files.

You will see some file references throughout the document that are relative to the contents of the compressed file you have downloaded. For example, **$(cat db-imagestream-patches/backend-redis-json.patch)**.

CHAPTER 2. UPGRADING 3SCALE 2.5 TO 2.6

Prerequisites

- Red Hat 3scale API Management 2.5 deployed in a project.
- Tool prerequisites:
  - base64
  - jq

Procedure

To upgrade 3scale API Management 2.5 to 2.6, go to the project where 3scale is deployed.

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

Then, follow these steps in this order:

1. Create a backup of the 3scale project
2. Configure support for the authenticated registry
3. Create OpenShift resources
4. Configure system DeploymentConfigs for Redis Enterprise and Redis Sentinel
5. Fix Redis Sentinel environment variables
6. Fix Zync environment variables
7. Migrate DeploymentConfig databases to ImageStreams
8. Upgrade 3scale Images
9. Migrate from WildcardRouter to Zync Route Management

2.1. CREATE BACKUP DIRECTORIES FOR THE 3SCALE PROJECT

Use the following procedure to create backup directories for the 3scale project. **Note** that **{BACKUP_DIR}** is the location on your machine of the 3scale backup.

Procedure

1. Create a backup directory:

```
mkdir ${BACKUP_DIR}
```

2. Create directories and subdirectories for your backups:

```
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. Backup system-database (MySQL)

Dump the **system-mysql** database and store the dump inside **${BACKUP_DIR}/system-database/system-mysql-backup.gz**:

```
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. Backup zync-database

Dump the **zync-database** PostrgreSQL database and store the dump inside **${BACKUP_DIR}/zync-database/zync-database-backup.gz**:

```
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. Backup system-redis

Extract the **system-redis** dump inside **${BACKUP_DIR}/system-redis/system-redis-dump.rdb**

```
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. Backup backend-redis

Extract the **backend-redis** dump inside **${BACKUP_DIR}/backend-redis/backend-redis-dump.rdb**

```
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. Backup system-app persistent data

Copy the **system-app** persistent data inside **${BACKUP_DIR}/system-app/**:

```
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. Backup DeploymentConfigs

```
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. Backup ImageStreams

```
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. Backup secrets

Back up everything except tokens and secrets default builder and deployer:

```
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. Backup services

```
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. Backup routes

```
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. Backup ConfigMaps

```
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. Backup other resources

Make a second backup, to deal with other custom resources that are not backed-up objects

```
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. CONFIGURE SUPPORT FOR THE AUTHENTICATED REGISTRY

As part of 3scale 2.6 release, container images have been migrated from **registry.access.redhat.com** to the authenticated registry located in **registry.redhat.io**. Follow the next steps to prepare the existing 3scale infrastructure to support the new authenticated registry:

1. Create credentials in the new Red Hat authenticated registry, located in **registry.redhat.io**.
    - Create a Registry Token, also called Registry Service Account. This registry token is intended to be used in the 3scale platform to authenticate against **registry.redhat.io**.
    - For more details on how to create credentials, see Red Hat Container Registry Authentication.

2. Once a Registry Service Account is available, create a new secret containing its credentials in the OpenShift project where the 3scale infrastructure is deployed:

   a. Obtain the OpenShift secret definition by navigating to your Red Hat Service Accounts panel.

   b. Choose the Registry Service Account to be used for 3scale infrastructure.

   c. Select the **OpenShift Secret** tab, and click the download secret link.

3. After downloading the OpenShift secret from the Red Hat Service Accounts panel, modify the *name* field in the **metadata** section of the YAML file, replacing the existing name with the **threescale-registry-auth** name.
   The secret looks something similar to this:

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

4. Save the changes, and create the secret in the OpenShift project where 3scale 2.5 is currently deployed:

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

5. After creating the secret, you can check its existence. The following command returns a secret with content:

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

6. Create the **amp** service account that will use the **threescale-registry-auth** secret. To do so, create the file **amp-sa.yml** with the following content:

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

7. Deploy the **amp** service account:

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

8. Ensure that the **amp** service account was correctly created. The following command returns the created service account with content, and having **threescale-registry-auth** as one of the elements in the **imagePullSecrets** section:

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

9. Verify that permissions once applied to the default service account of the existing 3scale project are replicated to the new **amp** service account.

    - If Service Discovery was configured in Service Account authentication mode, following the instructions available in Configuring without OAuth server, and cluster-role view permission was granted to the **system:serviceaccount:<3scale-project>:default** user, then that same permission needs now to be applied to **system:serviceaccount:<3scale-project>:amp**:

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

10. Update all existing DeploymentConfigs to use the new **amp** service account:

    ```
    $ 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
    ```

    The output of the command contains these lines:

    ```
    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
    ```

    The previous command will also redeploy all 3scale existing DeploymentConfigs triggering a reboot of them.

11. While DeploymentConfigs are rebooted, you might observe changes in their status. Wait until all the DeploymentConfigs are *Ready*.

    - You can check the status of DeploymentConfigs by typing the following command, and verifying that for each DeploymentConfig the **Desired** and **Current** columns have the same value and are different to zero:

    ```
    $ oc get dc
    ```

12. Also, verify that all pods are in *Running* status and all of them are *Ready*.

    ```
    $ oc get pods
    ```

13. Check that all DeploymentConfigs have the **amp** service account set with this command:

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

14. As a result of the previous command, the following line repeated as many times as the number of elements defined in the previously set THREESCALE_DC_NAMES environment variable is visible:

```
serviceAccountName: amp
```

15. At this point the DeploymentConfigurations are ready to use Red Hat authenticated registry images.

2.3. CREATE OPENSHIFT RESOURCES

This section provides the steps required for the creation of these new elements. As part of the 3scale 2.6 release the following OpenShift elements have been added:

- New ImageStreams for the databases:
  - backend-redis
  - system-redis
  - system-memcached
  - system-mysql
  - zync-database-postgresql
- New **zync-que** component, which contains the following OpenShift objects:
  - **zync-que** DeploymentConfig
  - **zync-que-sa** ServiceAccount
  - **zync-que** Role
  - **zync-que-rolebinding** RoleBinding

To create the new OpenShift elements, follow the next steps:

1. Create the following environment variable that contains the WildcardDomain set when 3scale 2.5 was deployed:

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

2. Verify that the THREESCALE_WILDCARD_DOMAIN environment variable is not empty and it has the same value as the Wildcard Domain that was set when deploying 3scale 2.5.

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

3. Create the following environment variable that contains the **ImportPolicy** ImageStream value set in the ImageStreams:

```
$ 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. Verify that the IMPORT_POLICY_VAL environment variable is either *true* or *false*:

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

5. Create the following environment variable that contains the current value of the **app** Kubernetes label in the 3scale pods. For example taking it from the **backend-listener** pod:

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

6. Verify that the DEPLOYED_APP_LABEL environment variable is not empty or **null**:

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

7. Deploy the new OpenShift objects for the 2.6 release using the 3scale 2.6 **amp.yml** standard scenario template:

```
$ 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}
```

   You will see several errors. These are expected because some of the elements already existed in 3scale 2.5. The only visible lines that are not errors include:

```
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. Verify that all the new ImageStreams described before exist, and also all the new zync-que related elements:

```
$ 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
```

All of the previous commands return an output showing that they have been created. Also, if you enter:

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

You will see that its status is *Error* or some other error that indicates that is crashing. That is expected because Zync images have not been updated at this point. This is done in point 4 of the Section 2.8, "Upgrade 3scale Images" section.

2.4. CONFIGURE SYSTEM DEPLOYMENTCONFIGS FOR REDIS ENTERPRISE AND REDIS SENTINEL

This section will help you configure the existing **system** DeploymentConfigs to use the secret fields you created. These secret fields are used as environment variables in **system-redis**.

1. Add the fields related to the Redis Enterprise compatibility for system connections in the **system-redis** secret:

```
$ 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. Add the new environment variables into **system-app** containers:

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

This command triggers a reboot of the **system-app** DeploymentConfig. Wait until the DeploymentConfig pods are rebooted and in *Ready* status again.

3. List all the environment variables of a DeploymentConfig with this command:

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

- Run this command to retrieve the list of environment variables before and after each patch command in the items of this step.
- The following are special cases where the command to list environment variables cannot be used and require specific commands:
  - The **pre-hook** pod:

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

  - The **system-sidekiq** initContainer

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

4. Add the new environment variables into the **system-app** pre-hook pod:

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

After running the previous commands, the existing environment variables remain without changes. Additionally, new variables are added to the **pre-hook** pod of system-app, and to all containers of system-app (system-master, system-developer, system-provider), using the **system-secret** secret as its source:

- 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. Add the new environment variables into **system-sidekiq**:

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

This command will trigger a reboot of the **system-sidekiq** DeploymentConfig. Wait until the DeploymentConfig pods are rebooted and in ready status again.

After running the previous command, the following environment variables have been added, keeping the existing ones unaltered, to the **system-sidekiq** InitContainer of the **system-sidekiq** pod:

- 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
  Moreover, the following environment variables have been added to the **system-sidekiq** pod:
- 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. Add the new environment variables to **system-sphinx**:

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

This command triggers a reboot of the **system-sphinx** DeploymentConfig. Wait until the DeploymentConfig pods are rebooted and in ready status again.

After running the previous command, the following environment variables have been added, keeping the existing ones unaltered, to the **system-sphinx** pod:

- 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. FIX REDIS SENTINEL ENVIRONMENT VARIABLES

This step involves fixing an issue in 3scale 2.5 that prevented a Redis Sentinel connection configuration to work in **backend-worker** and **backend-cron** pods.

1. Run the following command to see all the existing environment variables of a DeploymentConfig InitContainer:

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

Use this command to retrieve the list of environment variables before and after each patch command that is executed in this procedure to verify everything has worked as expected.

2. Apply the Redis Sentinel connections fix in backend-worker:

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

After running this command, the following environment variables have been added to the **backend-worker** InitContainer of the **backend-worker** DeploymentConfig:

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

3. Apply the Redis Sentinel connections fix in **backend-cron**:

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

After running this command, the following environment variables have been added to the **backend-cron** InitContainer of the **backend-cron** DeploymentConfig:

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

2.6. FIX ZYNC ENVIRONMENT VARIABLES

- Run the following command to update the zync environment:

```
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. MIGRATE DEPLOYMENTCONFIG DATABASES TO IMAGESTREAMS

In 2.6, the deployed 3scale DeploymentConfigs that contain a database have been migrated to obtain the container images from ImageStreams, instead of a direct reference to the image URL.

1. Migrate **backend-redis** DeploymentConfig to use backend-redis ImageStream:

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

- This triggers a redeployment of the **backend-redis** DeploymentConfig, and the DeploymentConfig has now an *ImageChange* trigger referencing the **backend-redis** ImageStream.

- **backend-worker**, **backend-cron** or **backend-listener** might temporarily fail until **backend-redis** pod is redeployed.
  Wait until the DeploymentConfig pods are rebooted and in ready status again.

2. Migrate **system-redis** DeploymentConfig to use **system-redis** ImageStream:

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

- This triggers a redeployment of the **system-redis** DeploymentConfig, and the DeploymentConfig has now an *ImageChange* trigger referencing the **backend-redis** ImageStream.

- Wait until the DeploymentConfig pods are rebooted and in ready status again.

3. Migrate the **system-memcache** DeploymentConfig to use **system-memcached** ImageStream:

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

- This triggers a redeployment of the **system-memcache** DeploymentConfig, and the DeploymentConfig has now an *ImageChange* trigger referencing the **system-memcached** ImageStream.

- Wait until the DeploymentConfig pods are rebooted and in ready status again.

4. Migrate **system-mysql** DeploymentConfig to use **system-mysql** ImageStream:

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

- This triggers a redeployment of the **system-mysql** DeploymentConfig, and the DeploymentConfig has now an *ImageChange* trigger referencing the **system-mysql** ImageStream.

- Wait until the DeploymentConfig pods are rebooted and in ready status again.

5. Migrate **zync-database** DeploymentConfig to use **zync-database-postgresql** ImageStream:

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

- This triggers a redeployment of the **zync-database** DeploymentConfig, and the DeploymentConfig has now an *ImageChange* trigger referencing the **zync-database-postgresql** ImageStream.

- The **zync** DeploymentConfig pod might temporarily fail until **zync-database** is available again, and it might take some time until it is in ready status again. Verify that after some minutes all 'zync' DeploymentConfig pods are in *Ready* status.

- Before you continue, wait until the DeploymentConfig pods are rebooted and in ready status again.

6. Remove the **postgresql** ImageStream that is no longer used:

```
$ oc delete ImageStream postgresql
```

7. To confirm success, verify that:

   - All database-related DeploymentConfigs are now using the ImageStream. You can verify that an *ImageChange* trigger pointing to the corresponding database ImageStream has been created.
   - The *ImageChange* trigger has a field named **lastTriggeredImage** that contains a URL pointing to **registry.redhat.io**.

2.8. UPGRADE 3SCALE IMAGES

1. Patch the **amp-system** image stream:

```
$ 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"}}}]'
```

   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.

NOTE

If you are using Oracle Database, you must rebuild the system image after executing the instructions above, by following the instructions in 3scale system image with Oracle Database

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 2.6"}, "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"}}}]'
```

   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 2.6"}, "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"}}}]'
```

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 2.6"}, "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"}}}]'
```

- 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.
- Additionally, you will see that **zync-que**, which was in *Error* status when it was created in previous sections, is running now and its pods in *Ready* status.

5. Update the visible release version:

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

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.

6. Finally, you can 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 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. 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 2.5 to 2.6 is now complete.