Red Hat 3scale API Management 2.8

Migrating 3scale

Upgrade your 3scale API Management installation.

Last Updated: 2022-12-13

Red Hat 3scale API Management 2.8 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 helps you to migrate and upgrade Red Hat 3scale API Management.

To upgrade your 3scale installation from 2.7 to 2.8, there are two guides depending on the installation type:

- 3scale template-based upgrade guide
- 3scale operator-based upgrade guide

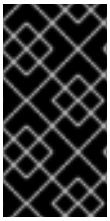
Post-upgrade step for provisioning APIs in the Developer Portal

- After a successful upgrade of 3scale, to configure OpenAPI Specification 3.0 (OAS 3.0) in the Developer Portal, see the following: Configuring the Developer Portal with OAS 3.0 .

To migrate from a template-based to an operator-based deployment, follow the procedures listed in Chapter 3, *3scale API Management migration guide: from template to operator-based deployments* .

CHAPTER 1. 3SCALE TEMPLATE-BASED UPGRADE GUIDE: FROM 2.7 TO 2.8

This section contains information about upgrading Red Hat 3scale API Management from version 2.7 to 2.8, in a template-based deployment.



IMPORTANT

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

1.1. GETTING READY FOR THE UPGRADE

This chapter describes the conditions to meet before the upgrade of 3scale. It also lists the tasks and tools you need to have as prerequisites necessary to perform the upgrade.

1.1.1. Conditions for the upgrade

Before proceeding with the upgrade, you must consider these points:

- 3scale supports upgrade paths from 2.7 to 2.8 with templates on OpenShift 3.11.
- Ensure your OpenShift CLI tool is configured in the same project where 3scale is deployed.

1.1.2. Prerequisites to perform the upgrade

This section describes the required tasks and tools to upgrade 3scale from 2.7 to 2.8 in a template-based installation.

Preliminary tasks

- Perform a backup of the database you are using with 3scale. The procedure of the backup is specific to each database type and setup.

Tools

You need these tools to perform the upgrade:

- 3scale 2.7 deployed with templates in an OpenShift 3.11 project.
- Bash shell: To run the commands detailed in the upgrade procedure.
- base64: To encode and decode secret information.
- jq: For JSON transformation purposes.

1.2. UPGRADING FROM 2.7 TO 2.8 IN A TEMPLATE-BASED INSTALLATION

Follow the procedure described in this section to upgrade 3scale 2.7 to 2.8 in a template-based installation.

To start with the upgrade, go to the project where 3scale is deployed.

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

Then, follow these steps in this order:

1. Section 1.2.1, "Creating a backup of the 3scale project"
2. Section 1.2.2, "Migrating the **smtp** ConfigMap to **system-smtp** secret"
3. Section 1.2.3, "Updating the **pre-hook pod** command of the **system-app** DeploymentConfig"
4. Section 1.2.4, "Patching the **pre-hook pod** environment of the **system-app** DeploymentConfig"
5. Section 1.2.5, "Patching the environment of the **system-app** DeploymentConfig containers"
6. Section 1.2.6, "Patching the environment of the **system-sidekiq** DeploymentConfig container"
7. Section 1.2.7, "Migrating S3 specific configuration"
8. Section 1.2.8, "Updating 3scale version number"
9. Section 1.2.9, "Upgrading 3scale images"
10. Section 1.2.10, "Deleting **smtp** ConfigMap"

1.2.1. Creating a backup of the 3scale project

Previous step

None.

Current step

This step lists the actions necessary to create a backup 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 of the generated files are not empty, and that all of them have the expected content.

Next step

Section 1.2.2, "Migrating the **smtp** ConfigMap to **system-smtp** secret"

1.2.2. Migrating the smtp ConfigMap to system-smtp secret

Previous step

Section 1.2.1, "Creating a backup of the 3scale project"

Current step

The goal of this step is to migrate the SMTP configuration of **system**, from a ConfigMap to a Secret. This migration involves mailing-related aspects, because the SMTP configuration contains some sensitive information. To protect this information, secrets are more secure than ConfigMaps.

1. Gather the current value of the **app** label:

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

- You can run following command to verify that DEPLOYED_APP_LABEL is not empty:

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

2. Gather the current contents of the **smtp** ConfigMap:

```
$ CFGMAP_DATA_CONTENTS=$(oc get configmap smtp -o json | jq -r .data)
```

- You can run following command to verify that CFGMAP_DATA_CONTENTS is not empty:

```
$ echo  ${CFGMAP_DATA_CONTENTS}
```

- You can confirm the value of CFGMAP_DATA_CONTENTS by running this command:

```
$ oc get configmap smtp -o json | jq -r .data
```

3. Create the **system-smtp** secret with the contents of the **smtp** ConfigMap:

```
$ cat <<EOF | oc create -f -
{
```

```
  "apiVersion": "v1",
  "kind": "Secret",
  "metadata": {
    "creationTimestamp": null,
    "labels": {
      "app": "${DEPLOYED_APP_LABEL}",
      "threescale_component": "system",
      "threescale_component_element": "smtp"
    },
    "name": "system-smtp"
  },
  "stringData": ${CFGMAP_DATA_CONTENTS}
}
EOF
```

4. Confirm that the **system-smtp** secret has been created by executing:

```
$ oc get secret system-smtp -o yaml
```

Verify that all data keys and associated values are the same in both **system-smtp** secret and the **smtp** ConfigMap. Data values in the **system-smtp** secret are base64 encoded, so they have to be decoded to look at the real value. For example, if a key in the secret data is named *mykey* you can copy the value associated to that key and decode it with the following command to see the real value:

```
$ oc get secret system-smtp -o json | jq -r .data.mykey | base64 -d
```

If the associated value to a key is an empty string, the result of the previous command will have no output.

Next step

Section 1.2.3, "Updating the **pre-hook pod** command of the **system-app** DeploymentConfig"

1.2.3. Updating the pre-hook pod command of the system-app DeploymentConfig

Previous step

Section 1.2.2, "Migrating the **smtp** ConfigMap to **system-smtp** secret"

Current step

To get the latest features from 3scale, this step explains how to update the **pre-hook pod** command in the **system-app** DeploymentConfig.

1. Within the **system-app** DeploymentConfig, update the pre-hook pod command to the new one needed for this release:

```
oc patch dc/system-app -p '{"spec":{"strategy":{"rollingParams":{"pre":{"execNewPod":
{"command":["bash","-c","bundle exec rake boot openshift:deploy"]}}}}}}'
```

2. Verify that the **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"
]
```

Next step

Section 1.2.4, "Patching the **pre-hook pod** environment of the **system-app** DeploymentConfig"

1.2.4. Patching the pre-hook pod environment of the system-app DeploymentConfig

Previous step

Section 1.2.3, "Updating the **pre-hook pod** command of the **system-app** DeploymentConfig"

Current step

This step adds environment variables to the **system-app** DeploymentConfig in the **pre-hook pod** environment. This addition ensures that SMTP-related environment variables point to the newly created **system-smtp** secret. This addition guarantees that the variables related to the modification of the **pre-hook pod** command are correctly configured.

1. Patch the **pre-hook pod** environment variables in the **system-app** DeploymentConfig:

```
oc get dc system-app -o json | jq 'del(.spec.strategy.rollingParams.pre.execNewPod.env[] |
select(.name == "SMTP_ADDRESS" // .name == "SMTP_USER_NAME" // .name ==
"SMTP_PASSWORD" // .name == "SMTP_DOMAIN" // .name == "SMTP_PORT" // .name
== "SMTP_AUTHENTICATION" // .name == "SMTP_OPENSSL_VERIFY_MODE")) |
.spec.strategy.rollingParams.pre.execNewPod.env +=
[{"name":"SMTP_ADDRESS","valueFrom":{"secretKeyRef":{"key":"address","name":"system-
smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"secretKeyRef":
{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":
{"secretKeyRef":{"key":"password","name":"system-smtp"}}},
{"name":"SMTP_DOMAIN","valueFrom":{"secretKeyRef":{"key":"domain","name":"system-
smtp"}}},{"name":"SMTP_PORT","valueFrom":{"secretKeyRef":{"key":"port","name":"system-
smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"secretKeyRef":
{"key":"authentication","name":"system-smtp"}}},
{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"secretKeyRef":
{"key":"openssl.verify.mode","name":"system-smtp"}}},
{"name":"MASTER_ACCESS_TOKEN","valueFrom":{"secretKeyRef":
{"key":"MASTER_ACCESS_TOKEN","name":"system-seed"}}}]' | oc apply -f -
```

2. Verify that **pre-hook pod** environment has been patched by following these action points:

   a. Check that MASTER_ACCESS_TOKEN has been set as a secret reference in the **system-app** pre-hook pod:

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env |
map(select(.name == "MASTER_ACCESS_TOKEN")) | length'
```

   Expected output: **1**

- You can confirm that MASTER_ACCESS_TOKEN has been set correctly pointing to the **system-seed** secret:

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env
| map(select(.name == "MASTER_ACCESS_TOKEN"))'
```

Expected output:

```
[
  {
    "name": "MASTER_ACCESS_TOKEN",
    "valueFrom": {
        "secretKeyRef": {
            "key": "MASTER_ACCESS_TOKEN",
            "name": "system-seed"
        }
    }
  }
]
```

b. Check that all SMTP_* env vars have been set as a secret reference in the **system-app** pre-hook pod:

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env |
map(select(.name | contains("SMTP")))'
```

- Each environment variable from the output list below should be a reference to the **system-smtp** secret key:
  - SMTP_ADDRESS
  - SMTP_USER_NAME
  - SMTP_PASSWORD
  - SMTP_DOMAIN
  - SMTP_PORT
  - SMTP_AUTHENTICATION
  - SMTP_OPENSSL_VERIFY_MODE

Next step

Section 1.2.5, “Patching the **system-app** DeploymentConfig containers”

1.2.5. Patching the environment of the system-app DeploymentConfig containers

Previous step

Section 1.2.4, “Patching the **pre-hook pod** environment of the **system-app** DeploymentConfig”

Current step

This procedure adds and modifies environment variables to the **system-app** container environments. It makes sure SMTP-related environment variables point to the newly created **system-smtp** secret.

1. Patch the container environment variables in **system-app** DeploymentConfig:

```
oc patch dc/system-app -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-master","env":[{"name":"SMTP_ADDRESS","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"authentication","name":"system-smtp"}}},{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-smtp"}}}]},{"name":"system-provider","env":[{"name":"SMTP_ADDRESS","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"authentication","name":"system-smtp"}}},{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-smtp"}}}]},{"name":"system-developer","env":[{"name":"SMTP_ADDRESS","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"authentication","name":"system-smtp"}}},{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-smtp"}}}]}]}}}}'
```

2. Verify that all SMTP_* env vars have been set as a secret reference in the **system-app** containers listed here:

- **system-developer**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name == "system-developer"))[].env | map(select(.name | contains("SMTP")))'
```

- **system-provider**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name == "system-provider"))[].env | map(select(.name | contains("SMTP")))'
```

- **system-master**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name == "system-master"))[].env | map(select(.name | contains("SMTP")))'
```

In these containers, the environment variable from the output list below should be a reference to the **system-smtp** secret key:

- SMTP_ADDRESS
- SMTP_USER_NAME
- SMTP_PASSWORD
- SMTP_DOMAIN
- SMTP_PORT
- SMTP_AUTHENTICATION
- SMTP_OPENSSL_VERIFY_MODE

Next step

Section 1.2.6, "Patching the environment of the **system-sidekiq** DeploymentConfig container"

1.2.6. Patching the environment of the system-sidekiq DeploymentConfig container

Previous step

Section 1.2.5, "Patching the environment of the **system-app** DeploymentConfig containers"

Current step

This procedure adds and modifies environment variables to the **system-sidekiq** pod environment. The steps listed here ensure that SMTP-related environment variables point to the newly created **system-smtp** secret.

1. Patch the environment variables of **system-sidekiq** DeploymentConfig:

```
oc patch dc/system-sidekiq -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-sidekiq","env":[{"name":"SMTP_ADDRESS","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"address","name":"system-smtp"}}},{"name":"SMTP_USER_NAME","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"username","name":"system-smtp"}}},{"name":"SMTP_PASSWORD","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"password","name":"system-smtp"}}},{"name":"SMTP_DOMAIN","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"domain","name":"system-smtp"}}},{"name":"SMTP_PORT","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"port","name":"system-smtp"}}},{"name":"SMTP_AUTHENTICATION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"authentication","name":"system-smtp"}}},{"name":"SMTP_OPENSSL_VERIFY_MODE","valueFrom":{"configMapKeyRef":null,"secretKeyRef":{"key":"openssl.verify.mode","name":"system-smtp"}}}]}]}}}}'
```

2. Confirm that all SMTP_* environment variables have been set as a secret reference:

```
oc get dc system-sidekiq -o json | jq '.spec.template.spec.containers | map(select(.name == "system-sidekiq"))[].env | map(select(.name | contains("SMTP")))'
```

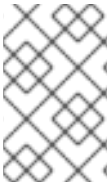
Each environment variable from the output list below should be a reference to the **system-smtp** secret key:

- SMTP_ADDRESS
- SMTP_USER_NAME
- SMTP_PASSWORD
- SMTP_DOMAIN
- SMTP_PORT
- SMTP_AUTHENTICATION
- SMTP_OPENSSL_VERIFY_MODE

Next step

- If you installed the **amp-s3** template in 3scale 2.7, Section 1.2.2, "Migrating the **smtp** ConfigMap to **system-smtp** secret".
- If you have not installed the **amp-s3** template in 3scale 2.7, Section 1.2.8, "Updating 3scale version number"

1.2.7. Migrating S3 specific configuration



NOTE

If you installed the **amp-s3** template in 3scale 2.7, follow the instructions of this step. Otherwise, continue the upgrade with the next step: Section 1.2.8, "Updating 3scale version number"

Previous step

Section 1.2.6, "Patching the environment of the **system-sidekiq** DeploymentConfig container"

Current step

This step lists the tasks to migrate the configuration specific to S3, from the **system-environment** ConfigMap to the **aws-auth** secret.

1. Add the values into the existing **aws-auth** secret:

```
oc patch secret aws-auth --patch "{\"stringData\": $(oc get configmap system-environment -o json | jq '.data | {"AWS_BUCKET": .AWS_BUCKET, "AWS_REGION": .AWS_REGION } ')}"
```

- Confirm that the keys and its values have been added to the **aws-auth** secret. These values are base64 encoded:

```
oc get secret aws-auth -o yaml
```

2. Patch the pre-hook pod environment variables from **system-app** DeploymentConfig:

```
oc get dc system-app -o json | jq 'del(.spec.strategy.rollingParams.pre.execNewPod.env[] |
select(.name == "AWS_BUCKET" // .name == "AWS_REGION")) |
.spec.strategy.rollingParams.pre.execNewPod.env +=
[{"name":"AWS_BUCKET","valueFrom":{"secretKeyRef":
{"key":"AWS_BUCKET","name":"aws-auth"}}},{"name":"AWS_REGION","valueFrom":
{"secretKeyRef":{"key":"AWS_REGION","name":"aws-auth"}}},
{"name":"AWS_PROTOCOL","valueFrom":{"secretKeyRef":
{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]' | oc apply -f -
```

- Check that all AWS_* environment variables have been set as a secret reference in the **system-app** pre-hook pod:

```
oc get dc system-app -o json | jq '.spec.strategy.rollingParams.pre.execNewPod.env |
map(select(.name | contains("AWS")))'
```

- Each environment variable from the output list below should be a reference to the **aws-auth** secret key:
  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_BUCKET
  - AWS_REGION
  - AWS_PROTOCOL
  - AWS_HOSTNAME
  - AWS_PATH_STYLE

3. Patch the containers environment variables in **system-app** DeploymentConfig:

```
oc patch dc/system-app -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-
master","env":[{"name":"AWS_BUCKET","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},
{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":
{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]},{"name":"system-
provider","env":[{"name":"AWS_BUCKET","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},
{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
```

```
{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":
{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]},{"name":"system-
developer","env":[{"name":"AWS_BUCKET","valueFrom":
{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},
{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":
{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]}]}}}}'
```

Verify that all AWS_* environment variables have been set as a secret reference in the three containers of **system-app**.

- **system-developer**:

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-developer"))[].env | map(select(.name | contains("AWS")))'
```

- **system-master**:

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-master"))[].env | map(select(.name | contains("AWS")))'
```

- **system-provider**

```
oc get dc system-app -o json | jq '.spec.template.spec.containers | map(select(.name ==
"system-provider"))[].env | map(select(.name | contains("AWS")))'
```

For all the three containers, each environment variable from the output list below should be a reference to the **aws-auth** secret key:

  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_BUCKET
  - AWS_REGION
  - AWS_PROTOCOL
  - AWS_HOSTNAME
  - AWS_PATH_STYLE

4. Patch the container environment variables in **system-sidekiq** DeploymentConfig:

```
oc patch dc/system-sidekiq -p '{"spec":{"template":{"spec":{"containers":[{"name":"system-
sidekiq","env":[{"name":"AWS_BUCKET","valueFrom":
```

```
{"configMapKeyRef":null,"secretKeyRef":{"key":"AWS_BUCKET","name":"aws-auth"}}},
{"name":"AWS_REGION","valueFrom":{"configMapKeyRef":null,"secretKeyRef":
{"key":"AWS_REGION","name":"aws-auth"}}},{"name":"AWS_PROTOCOL","valueFrom":
{"secretKeyRef":{"key":"AWS_PROTOCOL","name":"aws-auth", "optional": true}}},
{"name":"AWS_HOSTNAME","valueFrom":{"secretKeyRef":
{"key":"AWS_HOSTNAME","name":"aws-auth", "optional": true}}},
{"name":"AWS_PATH_STYLE","valueFrom":{"secretKeyRef":
{"key":"AWS_PATH_STYLE","name":"aws-auth", "optional": true}}}]}}}}}'
```

- Verify that all AWS_* environment variables have been set as a secret reference:

```
oc get dc system-sidekiq -o json | jq '.spec.template.spec.containers | map(select(.name
== "system-sidekiq"))[].env | map(select(.name | contains("AWS")))'
```

  Each environment variable from the output list below should be a reference to the **aws-auth** secret key:

  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY
  - AWS_BUCKET
  - AWS_REGION
  - AWS_PROTOCOL
  - AWS_HOSTNAME
  - AWS_PATH_STYLE

5. Delete the unused **system-environment** ConfigMap keys:

```
oc patch configmap system-environment --patch '{"data": {"AWS_BUCKET": null,
"AWS_REGION": null}}'
```

Next step

Section 1.2.8, “Updating 3scale version number”

1.2.8. Updating 3scale version number

Previous step

- If you installed the **amp-s3** template in 3scale 2.7, Section 1.2.2, “Migrating the **smtp** ConfigMap to **system-smtp** secret”.
- If you have not installed the **amp-s3** template in 3scale 2.7, Section 1.2.6, “Patching the environment of the **system-sidekiq** DeploymentConfig container”

Current step

This step updates the 3scale release version number from *2.7* to *2.8* in the **system-environment** ConfigMap. AMP_RELEASE is a ConfigMap entry referenced in some DeploymentConfig container environments.

1. To patch AMP_RELEASE, run this command:

```
oc patch cm system-environment --patch '{"data": {"AMP_RELEASE": "2.8"}}'
```

2. Verify that the AMP_RELEASE key in the system-environment ConfigMap has the **2.8** value:

```
oc get cm system-environment -o json | jq .data.AMP_RELEASE
```

Next step

Section 1.2.9, "Upgrading 3scale images"

1.2.9. Upgrading 3scale images

Previous step

Section 1.2.8, "Updating 3scale version number"

Current step

This step updates the 3scale images required for the upgrade process.

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 from Oracle DB, use this command:

```
oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "AMP system 2.8"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/system-rhel7:3scale2.8"},
"name": "2.8", "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": "2.8"}, "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 2.8"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.8"},
"name": "2.8", "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": "2.8"}, "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 2.8"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/backend-rhel7:3scale2.8"}, "name":
"2.8", "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": "2.8"}, "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 2.8"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/zync-rhel7:3scale2.8"}, "name":
"2.8", "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": "2.8"}, "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 2.8 Memcached"}, "from": {
"kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/memcached-
rhel7:3scale2.8"}, "name": "2.8", "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": "2.8"}, "name": "latest", "referencePolicy": {"type":
"Source"}}}]'
```

This triggers redeployment of the **system-memcache** DeploymentConfig. Wait until it is 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 2.8 PostgreSQL"},
"from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"},
"name": "2.8", "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": "2.8"}, "name": "latest",
"referencePolicy": {"type": "Source"}}}]'
```

- This patch command updates the **zync-database-postgresql** image stream to contain the 2.8 tag. You can verify that the 2.8 tag has been created by executing:

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

  Then, check that the **tags** column shows the **2.8** tag.

- 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 2.7 installation, click the link for the DeploymentConfigs that apply to obtain more information on how to proceed:
   - Section 1.2.9.1.1, "**backend-redis** DeploymentConfig"
   - Section 1.2.9.1.2, "**system-redis** DeploymentConfig"
   - Section 1.2.9.1.3, "**system-mysql** DeploymentConfig"
   - Section 1.2.9.1.4, "**system-postgresql** DeploymentConfig"
8. 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 address:

```
$ 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 zync-que"

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
```

Next step

Section 1.2.10, "Deleting **smtp** ConfigMap"

1.2.9.1. Additional steps with existing DeploymentConfigs

1.2.9.1.1. **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 2.8 Redis"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/redis-32-rhel7:3.2"}, "name": "2.8", "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": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **backend-redis** image stream to contain the **2.8** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.8**:

```
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.

1.2.9.1.2. 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 2.8 Redis"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/redis-32-rhel7:3.2"}, "name": "2.8", "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": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **system-redis** image stream to contain the **2.8** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.8**:

```
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.

1.2.9.1.3. 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 2.8 MySQL"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/mysql-57-rhel7:5.7"}, "name": "2.8", "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": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **system-mysql** image stream to contain the **2.8** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.8**:

```
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.

1.2.9.1.4. 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 2.8 PostgreSQL"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7
"}, "name": "2.8", "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": "2.8"}, "name": "latest", "referencePolicy": {"type": "Source"}}}]'
```

- This patch updates the **system-postgresql** image stream to contain the **2.8** tag. With the command below, you can confirm that the tag has been created if the **tags** column shows **2.8**:

```
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.

1.2.10. Deleting smtp ConfigMap

Previous step

Section 1.2.9, “Upgrading 3scale images”

Current step

This step removes the **smtp** ConfigMap, because this ConfigMap has been migrated to the **system-smtp** secret.

To remove the **smtp** ConfigMap, run this command:

```
$ oc delete cm smtp
```

If the command does not return an error, it has worked correctly.

Next step

None. After you have performed all the listed steps, 3scale upgrade from 2.7 to 2.8 in a template-based deployment is now complete.

CHAPTER 2. 3SCALE OPERATOR-BASED UPGRADE GUIDE: FROM 2.7 TO 2.8

This section contains information about upgrading Red Hat 3scale API Management from version 2.7 to 2.8, in an operator-based deployment.

IMPORTANT

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

Prerequisites

- 3scale 2.7 previously deployed via the 3scale operator.
- An OpenShift Container Platform (OCP) 4.x cluster with administrator access.

2.1. UPGRADING 3SCALE 2.7 TO 2.8

To upgrade 3scale from version 2.7 to 2.8 in an operator-based deployment, use the following procedure.

Procedure

1. Log in to the OCP console using the account with administrator privileges.
2. Select the project where the *3scale-operator* has been deployed.
3. Click **Operators > Installed Operators**.
4. Select 3scale operator **Subscription > Channel**
5. Edit the channel of the subscription by selecting the *threescale-2.8* and save the changes.
   - This will start the upgrade process.
   - Wait until the upgrade process 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 temporary errors 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. Confirm the upgrade process has been successful, by logging in to the 3scale Admin Portal and check that it works as expected.

8. Check the status of the *APIManager* objects and get the *YAML* content by running the following command:

```
oc get apimanager <myapimanager> -o yaml
```

   a. The new annotations with the values should be shown as follows:

```
apps.3scale.net/apimanager-threescale-version: "2.8"
apps.3scale.net/threescale-operator-version: "0.5.0"
```

   After you have performed all the listed steps, 3scale upgrade from 2.7 to 2.8 in an operator-based deployment is now complete.

CHAPTER 3. 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.

3.1. 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.

3.2. MIGRATING 3SCALE TEMPLATE TO OPERATOR-BASED DEPLOYMENTS

Prerequisites

- Red Hat 3scale API Management 2.8 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.