Red Hat 3scale API Management 2.11

Migrating 3scale

Migrate or upgrade 3scale API Management and its components

Last Updated: 2022-12-13

Red Hat 3scale API Management 2.11 Migrating 3scale

Migrate or upgrade 3scale API Management and its components

Legal Notice

Abstract

Migrate 3scale from a template to an operator-based installation. Also, find the information to upgrade 3scale and its components to the latest version.

Table of Contents

PREFACE

This guide provides the information to migrate Red Hat 3scale API Management from a template to an operator-based installation, the details required to upgrade your 3scale installation from 2.10 to 2.11, as well as the steps to upgrade APIcast in an operator-based deployment.

To migrate from a template-based to an operator-based deployment, refer to the procedures listed in the 3scale migration guide.

To upgrade your 3scale On-premises deployment from 2.10 to 2.11, refer to one of the following guides depending on the installation type:

- Upgrading 3scale version 2.10 to version 2.11 using templates
- Upgrading 3scale version 2.11.0 to version 2.11.1 using templates
- 3scale operator-based upgrade guide

To upgrade APIcast in an operator-based deployment, refer to the steps listed in the APIcast upgrade guide.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message .

CHAPTER 1. 3SCALE 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.

1.1. PREREQUISITES TO PERFORM 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 .

1.2. MIGRATING 3SCALE TEMPLATE TO OPERATOR-BASED DEPLOYMENTS

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.

CHAPTER 2. UPGRADING 3SCALE VERSION 2.10 TO VERSION 2.11 USING TEMPLATES

You can upgrade Red Hat 3scale API Management from version 2.10 to version 2.11 using a template-based deployment on OpenShift 3.11.

IMPORTANT

To understand the required conditions and procedures, be sure to 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, be sure to have a maintenance window.

2.1. PREREQUISITES TO PERFORM THE UPGRADE

This section describes the required configurations, tasks, and tools to upgrade 3scale from 2.10 to 2.11 in a template-based installation.

2.1.1. Configurations

- 3scale supports upgrade paths from 2.10 to 2.11 with templates on OpenShift 3.11.

2.1.2. Preliminary tasks

- Ensure your OpenShift CLI tool is configured in the same project where 3scale is deployed.
- Perform a backup of the database you are using with 3scale. The procedure of the backup is specific to each database type and setup.

2.1.3. Tools

You need these tools to perform the upgrade:

- 3scale 2.10 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.

2.2. UPGRADING FROM 2.10 TO 2.11 IN A TEMPLATE-BASED INSTALLATION

Follow the procedure described in this section to upgrade 3scale 2.10 to 2.11 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:

2.2.1. 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.

Procedure

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 backup 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 backup 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

Updating 3scale version number

2.2.2. Updating 3scale version number

Previous step

Creating a backup of the 3scale project

Current step

This step updates the 3scale release version number from *2.10* to *2.11* in the **system-environment** ConfigMap. AMP_RELEASE is a ConfigMap entry referenced in some DeploymentConfig container environments.

Procedure

1. To patch AMP_RELEASE, run this command:

```
$ oc patch cm system-environment --patch '{"data": {"AMP_RELEASE": "2.11"}}'
```

2. Confirm that the AMP_RELEASE key in the system-environment ConfigMap has the **2.11** value:

```
$ oc get cm system-environment -o json | jq '.data["AMP_RELEASE"]'
```

Next step

Updating BACKEND_ROUTE environment variable

2.2.3. Updating BACKEND_ROUTE environment variable

Previous step

Updating 3scale version number

Current step

This step updates the BACKEND_ROUTE environment variable from **system-app** and **system-sidekiq** pods to use the **backend-listener** Kubernetes service instead of the OpenShift route.

Procedure

1. Update the variable in the **system-app** pre-hook pod by editing the system-app DeploymentConfig:

```
$ oc edit dc system-app
```

You will enter an interactive editor session. Find the BACKEND_ROUTE environment variable in the **.spec.strategy.rollingParams.pre.execNewPod.env** array section.

a. Replace the following entry:

```
- name: BACKEND_ROUTE
  valueFrom:
    secretKeyRef:
      key: route_endpoint
      name: backend-listener
```

b. With this entry:

```
- name: BACKEND_ROUTE
  value: http://backend-listener:3000/internal/
```

Save your changes and exit the interactive editor session.

2. Update the entry on **system-app** containers:

```
$ oc set env dc/system-app BACKEND_ROUTE="http://backend-listener:3000/internal/"
```

This command triggers a redeployment of **system-app**. Wait until it is redeployed, its corresponding new pods are ready, and the previous pods are terminated.

3. Update it on **system-sidekiq** container:

```
$ oc set env dc/system-sidekiq BACKEND_ROUTE="http://backend-listener:3000/internal/"
```

This command triggers a redeployment of **system-sidekiq**. Wait until it is redeployed, its corresponding new pods are ready, and the previous pods are terminated.

Next step

Moving ‘zync’ DeploymentConfig monitoring annotations from DeploymentConfig annotations to PodTemplate annotations

2.2.4. Moving ‘zync’ DeploymentConfig monitoring annotations from DeploymentConfig annotations to PodTemplate annotations

Previous step

Updating BACKEND_ROUTE environment variable

Current step

This step moves the **prometheus.io/port and prometheus.io/scrape** annotations from the **zync** DeploymentConfig annotations to the PodTemplate annotations.

Procedure

1. Take note of the current values for the **prometheus.io/port and prometheus.io/scrape** annotations by running:

```
$ oc get dc zync -o json | jq .metadata.annotations
```

2. Add the annotations to **zync** DeploymentConfig’s PodTemplate annotations. If the **prometheus.io/port and prometheus.io/scrape** annotation values are different than the ones shown in the command below replace them with the values that are currently set in the **zync** DeploymentConfig as shown by the previous command:

```
$ oc patch dc zync --patch '{"spec":{"template":{"metadata":{"annotations":
{"prometheus.io/port":"9393","prometheus.io/scrape":"true"}}}}}'
```

3. Remove the original annotations from the zync DeploymentConfig annotations:

```
$ oc annotate dc zync prometheus.io/scrape-
$ oc annotate dc zync prometheus.io/port-
```

This command triggers a redeployment of zync. Wait until it is redeployed, its corresponding new pods are ready, and the previous pods are terminated.

Next step

Increasing **backend-cron** DeploymentConfig resource requirements

2.2.5. Increasing backend-cron DeploymentConfig resource requirements

Previous step

Moving ‘zync’ DeploymentConfig monitoring annotations from DeploymentConfig annotations to PodTemplate annotations

Current step

As of 3scale 2.11, **backend-cron** DeploymentConfig might consume more memory than earlier versions. Use this procedure to increase the maximum memory limits from the currently set values.

The required **backend-cron** resource in 3scale 2.11 are:

```
{
  "limits": {
 "cpu": "500m",
 "memory": "500Mi"
  },
  "requests": {
 "cpu": "100m",
 "memory": "100Mi"
  }
}
```

If the current **backend-cron** deployment has no memory limits or the resource requirements are higher, you do not need to complete the following procedure.

Procedure

1. Check the current resource requirements set for **backend-cron** with the following command:

```
$ oc get dc backend-cron -o json | jq .spec.template.spec.containers[0].resources
```

If the output is empty or **null** it means no resource requirements are set.

2. To increase the current **backend-cron** resource requirements, run the following command:

```
$ oc patch dc backend-cron --patch '{"spec":{"template":{"spec":{"containers":
[{"name":"backend-cron","resources":{"limits":{"memory":"500Mi", "cpu": "500m"}, "requests":
{"memory":"100Mi", "cpu": "100m"}}}]}}}}'
```

This command triggers a redeployment of **backend-cron**. Wait until it is redeployed, its corresponding new pods are ready, and the previous pods are terminated.

Next step

Upgrading 3scale images

2.2.6. Upgrading 3scale images

Previous step

Increasing **backend-cron** DeploymentConfig resource requirements

Current step

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

2.2.6.1. Patch the **system** image

1. Create the new image stream tag:

```
$ oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "AMP system 2.11"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/system-rhel7:3scale2.11"}, "name":
"2.11", "referencePolicy": {"type": "Source"}}}]'
```

2. To continue the procedure, consider the database used with your 3scale deployment:

   - If the database is Oracle DB, follow the steps listed in Patching the system image: 3scale with Oracle Database
   - If the database is different from Oracle DB, follow the steps listed in Patching the system image: 3scale with other databases

2.2.6.1.1. Patching the system image: 3scale with Oracle Database

1. To start patching the system image of 3scale with an Oracle Database, perform steps 1, 2, 4, and 8 in Building the system image.
2. Patch the **system-app** ImageChangeTrigger:

   a. Remove the old **2.10-oracle** trigger:

   ```
   $ oc set triggers dc/system-app --from-image=amp-system:2.10-oracle --
   containers=system-master,system-developer,system-provider --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/system-app --from-image=amp-system:2.11-oracle --
   containers=system-master,system-developer,system-provider
   ```

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

3. Patch the **system-sidekiq** ImageChange trigger:

   a. Remove the old **2.10-oracle** trigger:

   ```
   $ oc set triggers dc/system-sidekiq --from-image=amp-system:2.10-oracle --
   containers=system-sidekiq,check-svc --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/system-sidekiq --from-image=amp-system:2.11-oracle --
   containers=system-sidekiq,check-svc
   ```

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

4. Patch the **system-sphinx** ImageChange trigger:

   a. Remove the old **2.10-oracle** trigger:

   ```
   $ oc set triggers dc/system-sphinx --from-image=amp-system:2.10-oracle --
   containers=system-sphinx,system-master-svc --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/system-sphinx --from-image=amp-system:2.11-oracle --
   containers=system-sphinx,system-master-svc
   ```

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

5. Scale 3scale back if you scaled it down.

2.2.6.1.2. Patching the system image: 3scale with other databases

1. Patch the **system-app** ImageChange trigger:

   a. Remove the old **2.10** trigger:

   ```
   $ oc set triggers dc/system-app --from-image=amp-system:2.10 --containers=system-
   master,system-developer,system-provider --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/system-app --from-image=amp-system:2.11 --containers=system-
   master,system-developer,system-provider
   ```

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

2. Patch the **system-sidekiq** ImageChange trigger:

   a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/system-sidekiq --from-image=amp-system:2.10 --containers=system-sidekiq,check-svc --remove
```

b. Add the new version-specific trigger:

```
$ oc set triggers dc/system-sidekiq --from-image=amp-system:2.11 --containers=system-sidekiq,check-svc
```

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

3. Patch the **system-sphinx** ImageChange trigger:

   a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/system-sphinx --from-image=amp-system:2.10 --containers=system-sphinx,system-master-svc --remove
```

   b. Add the new version-specific trigger:

```
$ oc set triggers dc/system-sphinx --from-image=amp-system:2.11 --containers=system-sphinx,system-master-svc
```

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

2.2.6.2. Patch the apicast image

1. 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 2.11"}, "from": {"kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.11"}, "name": "2.11", "referencePolicy": {"type": "Source"}}}]'
```

2. Patch the **apicast-staging** ImageChange trigger:

   a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/apicast-staging --from-image=amp-apicast:2.10 --containers=apicast-staging --remove
```

   b. Add the new version-specific trigger:

```
$ oc set triggers dc/apicast-staging --from-image=amp-apicast:2.11 --containers=apicast-staging
```

This triggers a redeployment of **apicast-staging**. Wait until it is redeployed, its corresponding new pods are ready, and the old ones terminated.

3. Patch the **apicast-production** ImageChange trigger:

   a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/apicast-production --from-image=amp-apicast:2.10 --
containers=apicast-production,system-master-svc --remove
```

b. Add the new version-specific trigger:

```
$ oc set triggers dc/apicast-production --from-image=amp-apicast:2.11 --
containers=apicast-production,system-master-svc
```

This triggers a redeployment of **apicast-production**. Wait until it is redeployed, its corresponding new pods are ready, and the old ones terminated.

2.2.6.3. Patch the backend image

1. 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 2.11"}, "from": {"kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/backend-rhel8:3scale2.11"}, "name":
"2.11", "referencePolicy": {"type": "Source"}}}]'
```

2. Patch the **backend-listener** ImageChange trigger:

a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/backend-listener --from-image=amp-backend:2.10 --
containers=backend-listener --remove
```

b. Add the new version-specific trigger:

```
$ oc set triggers dc/backend-listener --from-image=amp-backend:2.11 --
containers=backend-listener
```

This triggers a redeployment of **backend-listener**. Wait until it is redeployed, its corresponding new pods are ready, and the old ones terminated.

3. Patch the **backend-worker** ImageChange trigger:

a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/backend-worker --from-image=amp-backend:2.10 --
containers=backend-worker,backend-redis-svc --remove
```

b. Add the new version-specific trigger:

```
$ oc set triggers dc/backend-worker --from-image=amp-backend:2.11 --
containers=backend-worker,backend-redis-svc
```

This triggers a redeployment of **backend-worker**. Wait until it is redeployed, its corresponding new pods are ready, and the old ones terminated.

4. Patch the **backend-cron** ImageChange trigger:

a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/backend-cron --from-image=amp-backend:2.10 --
containers=backend-cron,backend-redis-svc --remove
```

b. Add the new version-specific trigger:

```
$ oc set triggers dc/backend-cron --from-image=amp-backend:2.11 --
containers=backend-cron,backend-redis-svc
```

This command triggers a redeployment of **backend-cron**. Wait until it is redeployed, its corresponding new pods are ready, and the previous pods are terminated.

2.2.6.4. Patch the zync image

1. 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 2.11"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/zync-rhel8:3scale2.11"}, "name":
"2.11", "referencePolicy": {"type": "Source"}}}]'
```

2. Patch the **zync** ImageChange trigger:

a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/zync --from-image=amp-zync:2.10 --containers=zync,zync-db-svc --
remove
```

b. Add the new version-specific trigger:

```
$ oc set triggers dc/zync --from-image=amp-zync:2.11 --containers=zync,zync-db-svc
```

This triggers a redeployment of **zync**. Wait until it is redeployed, its corresponding new pods are ready, and the old ones terminated.

3. Patch the **zync-que** ImageChange trigger:

a. Remove the old **2.10** trigger:

```
$ oc set triggers dc/zync-que --from-image=amp-zync:2.10 --containers=que --remove
```

b. Add the new version-specific trigger:

```
$ oc set triggers dc/zync-que --from-image=amp-zync:2.11 --containers=que
```

This triggers a redeployment of **zync-que**. Wait until it is redeployed, its corresponding new pods are ready, and the old ones terminated.

2.2.6.5. Patch the system-memcached image

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

```
$ oc patch imagestream/system-memcached --type=json -p '[{"op": "add", "path":
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.11
Memcached"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-
amp2/memcached-rhel7:3scale2.11"}, "name": "2.11", "referencePolicy": {"type": "Source"}}}]'
```

2. Patch the **system-memcache** ImageChange trigger:

   a. Remove the old **2.10** trigger:

   ```
   $ oc set triggers dc/system-memcache --from-image=system-memcached:2.10 --
   containers=memcache --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/system-memcache --from-image=system-memcached:2.11 --
   containers=memcache
   ```

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

2.2.6.6. Patch the zync-database-postgresql image

1. 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 2.11 PostgreSQL"},
"from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"},
"name": "2.11", "referencePolicy": {"type": "Source"}}}]'
```

- This patch command updates the **zync-database-postgresql** image stream to contain the **2.11** tag. You can verify that the **2.11** tag has been created with these steps:

  a. Run this command:

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

  b. Check that the *Tags* column shows the **2.11** tag.

2. Patch the **zync-database** ImageChange trigger:

   a. Remove the old **2.10** trigger:

   ```
   $ oc set triggers dc/zync-database --from-image=zync-database-postgresql:2.10 --
   containers=postgresql --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/zync-database --from-image=zync-database-postgresql:2.11 --
   containers=postgresql
   ```

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

2.2.6.7. Additional image changes

If one or more of the following DeploymentConfigs are available in your 3scale 2.10 installation, click the links that apply to obtain more information on how to proceed:

- **backend-redis** DeploymentConfig
- **system-redis** DeploymentConfig
- **system-mysql** DeploymentConfig
- **system-postgresql** DeploymentConfig

backend-redis DeploymentConfig

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

1. 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 2.11 Redis"}, "from": { "kind":
   "DockerImage", "name": "registry.redhat.io/rhscl/redis-5-rhel7:5"}, "name": "2.11",
   "referencePolicy": {"type": "Source"}}}]'
   ```

   This patch updates the backend-redis image stream to contain the **2.11** tag. With the command below, you can confirm that the tag has been created if the *Tags* column shows **2.11**:

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

2. Patch the **backend-redis** ImageChange trigger:

   a. Remove the old **2.10** trigger:

   ```
   $ oc set triggers dc/backend-redis --from-image=backend-redis:2.10 --
   containers=backend-redis --remove
   ```

   b. For 3scale 2.11 redis image is upgraded from Redis 3 to 5, which contains a different binary path to Redis. The **backend-redis** deployment container command must be updated to use the new path. **Note:** Applying this change will temporarily leave the **backend-redis** deployment in an error state until you *add the new version-specific trigger* in the next substep:

   ```
   $ oc patch dc backend-redis --patch '{"spec":{"template":{"spec":{"containers":
   [{"name":"backend-redis","command":["/opt/rh/rh-redis5/root/usr/bin/redis-server"]}]}}}}'
   ```

   c. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/backend-redis --from-image=backend-redis:2.11 --
   containers=backend-redis
   ```

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

system-redis DeploymentConfig

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

1. 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 2.11 Redis"}, "from": { "kind":
   "DockerImage", "name": "registry.redhat.io/rhscl/redis-5-rhel7:5"}, "name": "2.11",
   "referencePolicy": {"type": "Source"}}}]'
   ```

   This patch updates the **system-redis** image stream to contain the **2.11** tag. With the command below, you can confirm that the tag has been created if the *Tags* column shows **2.11**:

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

2. Patch the **system-redis** ImageChange trigger:

   a. Remove the old **2.10** trigger:

      ```
      $ oc set triggers dc/system-redis --from-image=system-redis:2.10 --containers=system-
      redis --remove
      ```

   b. For 3scale 2.11 redis image is upgraded from Redis 3 to 5, which contains a different binary path to Redis. The **system-redis** deployment container command must be updated to use the new path. **Note:** Applying this change will temporarily leave the **system-redis** deployment in an error state until you *add the new version-specific trigger* in the next substep:

      ```
      $ oc patch dc system-redis --patch '{"spec":{"template":{"spec":{"containers":
      [{"name":"system-redis","command":["/opt/rh/rh-redis5/root/usr/bin/redis-server"]}]}}}}'
      ```

   c. Add the new version-specific trigger:

      ```
      $ oc set triggers dc/system-redis --from-image=system-redis:2.11 --containers=system-
      redis
      ```

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

system-mysql DeploymentConfig

If the **system-mysql** DeploymentConfig exists in your current 3scale installation, patch the MySQL image for **system-mysql**.

1. 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 2.11 MySQL"}, "from": { "kind":
   "DockerImage", "name": "registry.redhat.io/rhscl/mysql-57-rhel7:5.7"}, "name": "2.11",
   "referencePolicy": {"type": "Source"}}}]'
   ```

   This patch updates the **system-mysql** image stream to contain the **2.11** tag. With the command below, you can confirm that the tag has been created if the *Tags* column shows **2.11**:

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

2. Patch the **system-mysql** ImageChange trigger:

   a. Remove the old **2.10** trigger:

   ```
   $ oc set triggers dc/system-mysql --from-image=system-mysql:2.10 --
   containers=system-mysql --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/system-mysql --from-image=system-mysql:2.11 --
   containers=system-mysql
   ```

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

system-postgresql DeploymentConfig

If the **system-postgresql** DeploymentConfig exists in your current 3scale installation, patch the PostgreSQL image for **system-postgresql**.

1. 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 2.11 PostgreSQL"}, "from": {
"kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"}, "name": "2.11",
"referencePolicy": {"type": "Source"}}}]'
```

This patch updates the **system-postgresql** image stream to contain the **2.11** tag. With the command below, you can confirm that the tag has been created if the *Tags* column shows **2.11**:

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

2. Patch the **system-postgresql** ImageChange trigger:

   a. Remove the old **2.10** trigger:

   ```
   $ oc set triggers dc/system-postgresql --from-image=system-postgresql:2.10 --
   containers=system-postgresql --remove
   ```

   b. Add the new version-specific trigger:

   ```
   $ oc set triggers dc/system-postgresql --from-image=system-postgresql:2.11 --
   containers=system-postgresql
   ```

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

2.2.6.8. Confirm image URLs

Confirm 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

None. After you have performed all the listed steps, 3scale upgrade from 2.10 to 2.11 in a template-based deployment is now complete.

2.3. UPGRADING 3SCALE WITH AN ORACLE DATABASE IN A TEMPLATE-BASED INSTALLATION

This section explains how to update Red Hat 3scale API Management when you are using a 3scale system image with an Oracle Database, in a template-based installation with OpenShift 3.11.

Prerequisites

A 3scale installation with the Oracle Database. See Setting up your 3scale system image with an Oracle Database.

To upgrade your 3scale system image with an Oracle Database in a template-based installation, perform the procedure below:

- Upgrading 3scale with Oracle 19c

2.3.1. Upgrading 3scale with Oracle 19c

This procedure guides you through an Oracle Database 19c update for 3scale 2.11 from an existing 3scale 2.10 installation.

IMPORTANT: Loss of connection to the database can potentially corrupt 3scale. Make a backup before proceeding to perform the upgrade. For more information see the Oracle Database documentation: Oracle Database Backup and Recovery User's Guide .

Prerequisites

- A 3scale 2.10 installation.
- An Oracle Database 19c installation.
  - For more information about configuring 3scale with Oracle, see Preparing the Oracle Database.

Procedure

1. Download 3scale OpenShift templates from the GitHub repository and extract the archive:

   ```
   tar -xzf 3scale-amp-openshift-templates-3scale-2.11.1-GA.tar.gz
   ```

2. Place your Oracle Database Instant Client Package files into the **3scale-amp-openshift-templates-3scale-2.11.1-GA/amp/system-oracle/oracle-client-files** directory.

3. Run the **oc process** command with the **-f** option and specify the **build.yml** OpenShift template:

```
$ oc process -f build.yml | oc apply -f -
```

4. Run the **oc new-app** command with the **-f** option to indicate the **amp.yml** OpenShift template, and the **-p** option to specify the **WILDCARD_DOMAIN** parameter with the domain of your OpenShift cluster:

```
$ oc new-app -f amp.yml -p WILDCARD_DOMAIN=mydomain.com
```

NOTE

The following steps are optional. Use them if you remove **ORACLE_SYSTEM_PASSWORD** after the installation or a system upgrade.

5. Enter the following **oc patch** commands, replacing **SYSTEM_PASSWORD** with the Oracle Database **system** password you set up in Preparing the Oracle Database:

```
$ oc patch dc/system-app -p '[{"op": "add", "path":
"/spec/strategy/rollingParams/pre/execNewPod/env/-", "value": {"name":
"ORACLE_SYSTEM_PASSWORD", "value": "SYSTEM_PASSWORD"}}]' --type=json

$ oc patch dc/system-app -p '{"spec": {"strategy": {"rollingParams": {"post":{"execNewPod":
{"env": [{"name": "ORACLE_SYSTEM_PASSWORD", "value":
"SYSTEM_PASSWORD"}]}}}}}}'
```

6. Enter the following command, replacing **DATABASE_URL** to point to your Oracle Database, specified in Preparing the Oracle Database:

```
$ oc patch secret/system-database -p '{"stringData": {"URL": "DATABASE_URL"}}'
```

7. Enter the **oc start-build** command to build the new system image:

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

8. Wait until the build completes. To see the state of the build, run the following command:

```
$ oc get build <build-name> -o jsonpath="{.status.phase}"
```

   a. Wait until the build is in a Complete state.

9. Once you have set up your 3scale system image with your Oracle Database, remove **ORACLE_SYSTEM_PASSWORD** from the **system-app** DeploymentConfig. It is not necessary again until you upgrade to a new version of 3scale.

```
$ oc set env dc/system-app ORACLE_SYSTEM_PASSWORD-
```

Additional resources

For more information about 3scale and Oracle Database support, see Red Hat 3scale API Management Supported Configurations.

CHAPTER 3. UPGRADING 3SCALE VERSION 2.11.0 TO VERSION 2.11.1 USING TEMPLATES

You can upgrade Red Hat 3scale API Management from version 2.11.0 to version 2.11.1 using a template-based deployment on OpenShift 3.11.

IMPORTANT

To understand the required conditions and procedures, be sure to 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, be sure to have a maintenance window.

3.1. PREREQUISITES TO PERFORM THE UPGRADE

This section describes the required configurations, tasks, and tools to upgrade 3scale from 2.11.0 to 2.11.1 in a template-based installation.

3.1.1. Configurations

- 3scale supports upgrade paths from 2.11.0 to 2.11.1 with templates on OpenShift 3.11.

3.1.2. Preliminary tasks

- Ensure your OpenShift CLI tool is configured in the same project where 3scale is deployed.
- Perform a backup of the database you are using with 3scale. The procedure of the backup is specific to each database type and setup.

3.1.3. Tools

You need these tools to perform the upgrade:

- 3scale 2.11.0 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.

3.2. UPGRADING FROM 3SCALE 2.11.0 TO 2.11.1 IN A TEMPLATE-BASED INSTALLATION

Follow the procedure described in this section to upgrade 3scale 2.11.0 to 2.11.1 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. Creating a backup of the 3scale project
2. Increasing **backend-cron** DeploymentConfig resource requirements
3. Upgrading 3scale images

3.2.1. 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.

Procedure

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 backup 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 backup 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

Increasing **backend-cron** DeploymentConfig resource requirements

3.2.2. Increasing backend-cron DeploymentConfig resource requirements

Previous step

Creating a backup of the 3scale project

Current step

In this release **backend-cron** DeploymentConfig has potentially more resources usage. Use this procedure to increase the resource requirements from the currently set values.

The required **backend-cron** resource in 3scale 2.11.1 are:

```
{
  "limits": {
 "cpu": "500m",
 "memory": "500Mi"
  },
  "requests": {
 "cpu": "100m",
 "memory": "100Mi"
  }
}
```

If the current **backend-cron** deployment has no memory limits or the resource requirements are higher, you do not need to complete the following procedure.

Procedure

1. Check the current resource requirements set for **backend-cron** with the following command:

   ```
   $ oc get dc backend-cron -o json | jq .spec.template.spec.containers[0].resources
   ```

   If the output is empty or **null** it means no resource requirements are set.

2. To increase the current **backend-cron** resource requirements, run the following command:

   ```
   $ oc patch dc backend-cron --patch '{"spec":{"template":{"spec":{"containers":
   [{"name":"backend-cron","resources":{"limits":{"memory":"500Mi", "cpu": "500m"}, "requests":
   {"memory":"100Mi", "cpu": "100m"}}}]}}}}'
   ```

   This command triggers a redeployment of **backend-cron**. Wait until it is redeployed, its corresponding new pods are ready, and the previous pods are terminated.

Next step

Upgrading 3scale images

3.2.3. Upgrading 3scale images

Previous step

Increasing **backend-cron** DeploymentConfig resource requirements

Current step

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

3.2.3.1. Patch the **system** image

1. To continue the procedure, consider the database used with your 3scale deployment:
   - If the database is Oracle DB, follow the steps listed in Patching the system image: 3scale with Oracle Database
   - If the database is different from Oracle DB, follow the steps listed in Patching the system image: 3scale with other databases

3.2.3.2. Patching the system image: 3scale with Oracle Database

1. To start patching the system image of 3scale with an Oracle Database, perform steps 1, 2, 4, and 8 in Building the system image .
2. Import the latest **amp-system** Oracle Database image:

   ```
   oc import-image amp-system:2.11-oracle
   ```

3.2.3.3. Patching the system image: 3scale with other databases

Import the latest **system-app** image:

```
oc import-image amp-system:2.11
```

3.2.3.4. Patch the **apicast** image

Import the latest **amp-apicast** image:

```
oc import-image amp-apicast:2.11
```

3.2.3.5. Patch the **backend** image

Import the latest **amp-backend** image:

```
oc import-image amp-backend:2.11
```

3.2.3.6. Patch the **zync** image

Import the latest **amp-zync** image:

```
oc import-image amp-zync:2.11
```

3.2.3.7. Patch the **system-memcached** image

Import the latest **system-memcached** image:

```
oc import-image system-memcached:2.11
```

Next step

None. After you have performed all the listed steps, 3scale upgrade from 2.11.0 to 2.11.1 in a template-based deployment is now complete.

CHAPTER 4. 3SCALE OPERATOR-BASED UPGRADE GUIDE: FROM 2.10 TO 2.11

Upgrade Red Hat 3scale API Management from version 2.10 to 2.11, in an operator-based installation to manage 3scale on OpenShift 4.x.

To automatically obtain a micro-release of 3scale, make sure automatic updates is on. To check this, see Setting up the 3scale operator for micro releases.

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.

4.1. PREREQUISITES TO PERFORM THE UPGRADE

This section describes the required configurations to upgrade 3scale from 2.10 to 2.11 in an operator-based installation.

- An OpenShift Container Platform (OCP) 4.6, 4.7, or 4.8 cluster with administrator access.
  - You must perform the upgrade of 3scale from 2.10 to 2.11 before upgrading to OCP 4.9.
  - **Note:** If you upgrade OCP to 4.9 or greater before upgrading 3scale you will get a non-working installation.
- 3scale 2.10 previously deployed via the 3scale operator.
- Make sure the latest CSV of the **threescale-2.10** channel is in use. To check it:
  - If the approval setting for the subscription is *automatic* you should already be in the latest CSV version of the channel.
  - If the approval setting for the subscription is *manual* make sure you approve all pending *InstallPlans* and have the latest CSV version.
  - Keep in mind if there is a pending install plan, there might be more pending install plans, which will only be shown after the existing pending plan has been installed.

4.2. UPGRADING FROM 2.10 TO 2.11 IN AN OPERATOR-BASED INSTALLATION

To upgrade 3scale from version 2.10 to 2.11 in an operator-based deployment:

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 **Red Hat Integration - 3scale** > **Subscription** > **Channel**
5. Edit the channel of the subscription by selecting *threescale-2.11* and save the changes.

This will start the upgrade process.

6. Query the pods' status on the project until you see all the new versions are running and ready without errors:

   ```
   $ oc get pods
   ```

   NOTE

   - The pods might have temporary errors during the upgrade process.
   - The time required to upgrade pods can vary from 5-10 minutes.

7. After new pod versions are running, confirm a successful upgrade by logging in to the 3scale Admin Portal and checking that it works as expected.

8. Check the status of the *APIManager* objects and get the *YAML* content by running the following command. <myapimanager> represents the name of your *APIManager*:

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

   - The new annotations with the values should be as follows:

     ```
     apps.3scale.net/apimanager-threescale-version: "2.11"
     apps.3scale.net/threescale-operator-version: "0.8.0"
     ```

After you have performed all steps, 3scale upgrade from 2.10 to 2.11 in an operator-based deployment is complete.

CHAPTER 5. APICAST OPERATOR-BASED UPGRADE GUIDE: FROM 2.10 TO 2.11

Upgrading APIcast from 2.10 to 2.11 in an operator-based installation helps you use the APIcast API gateway to integrate your internal and external API services with 3scale.

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.

5.1. PREREQUISITES TO PERFORM THE UPGRADE

To perform the upgrade of APIcast from 2.10 to 2.11 in an operator-based installation, the following required prerequisites must already be in place:

- An OpenShift Container Platform (OCP) 4.6, 4.7, or 4.8 cluster with administrator access.
  - You must perform the upgrade of APIcast from 2.10 to 2.11 before upgrading to OCP 4.9
  - **Note:** If you upgrade OCP to 4.9 or greater before upgrading APIcast you will get a non-working installation.
- APIcast 2.10 previously deployed via the APIcast operator.
- Make sure the latest CSV of the **threescale-2.10** channel is in use. To check it:
  - If the approval setting for the subscription is *automatic* you should already be in the latest CSV version of the channel.
  - If the approval setting for the subscription is *manual* make sure you approve all pending *InstallPlans* and have the latest CSV version.
  - Keep in mind if there is a pending install plan, there might be more pending install plans, which will only be shown after the existing pending plan has been installed.

5.2. UPGRADING APICAST FROM 2.10 TO 2.11 IN AN OPERATOR-BASED INSTALLATION

Upgrade APIcast from 2.10 to 2.11 in an operator-based installation so that APIcast can function as the API gateway in your 3scale installation.

Procedure

1. Log in to the OCP console using the account with administrator privileges.
2. Select the project where the *APIcast operator* has been deployed.
3. Click **Operators > Installed Operators**.
4. In **Subscription > Channel**, select *Red Hat Integration - 3scale APIcast gateway* .

5. Edit the channel of the subscription by selecting the *threescale-2.11* channel and save the changes.
   This will start the upgrade process.

6. Query the pods status on the project until you see all the new versions are running and ready without errors:

   ```
   $ oc get pods
   ```

   > **NOTE**
   >
   > - The pods might have temporary errors during the upgrade process.
   > - The time required to upgrade pods can vary from 5-10 minutes.

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

   ```
   $ oc get apicast <myapicast> -o yaml
   ```

   - The new annotations with the values should be as follows:

     ```
     apicast.apps.3scale.net/operator-version: “0.5.0”
     ```

After you have performed all the listed steps, APIcast upgrade from 2.10 to 2.11 in an operator-based deployment is now complete.