Red Hat 3scale API Management 2.12

迁移 3scale

迁移或升级 3scale API 管理及其组件

Last Updated: 2024-01-10

Red Hat 3scale API Management 2.12 迁移 3scale

迁移或升级 3scale API 管理及其组件

法律通告

摘要

将 3scale 从模板迁移到基于 Operator 的安装。另外，查找要将 3scale 及其组件升级到最新版本的信息。

目录

前言

本指南提供了将 Red Hat 3scale API Management 从模板迁移到基于 Operator 的安装的信息、将 3scale 安装从 2.10 升级到 2.11 所需的详细信息，以及在基于 Operator 的部署中升级 APIcast 的步骤。

要从基于模板的部署迁移到基于 Operator 的部署，请参阅 3scale 迁移指南中列出的步骤。

要将 3scale On-premises 部署从 2.10 升级到 2.11，请参阅以下步骤之一，具体取决于安装类型：

- 使用模板将 3scale 版本 2.10 升级到 2.11
- 3scale 基于 operator 的升级指南

要在基于 operator 的部署中升级 APIcast，请参阅 APIcast 升级指南中列出的步骤。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看我们的 CTO Chris Wright 信息。

第 1 章 3SCALE 迁移指南：从模板到基于 OPERATOR 的部署

本节介绍使用 Red Hat OpenShift 3.11 将 Red Hat 3scale API 管理从基于模板的部署迁移到使用 Red Hat OpenShift 4.x 基于 Operator 的部署。

警告

为了了解所需的条件和程序，请在应用列出的步骤前阅读整个迁移指南。迁移过程会破坏服务的调配，直到过程完成为止。因为这个过程需要涉及到系统中断，请确保计划有一个维护窗口进行。

1.1. 执行迁移的先决条件

在将 3scale 安装从模板迁移到基于 Operator 的部署之前，请通过咨询以下指南确认您的部署受支持：

- 备份 3scale 基于模板的部署。
- 在基于 Operator 的部署中恢复备份。

1.2. 将 3SCALE 模板迁移到基于 OPERATOR 的部署

迁移前的基本设置是 3scale 指向 OCP3 域：**3scale.example.com → ocp3.example.com**

要将 3scale 从使用 Red Hat OpenShift 3.11 的基于模板的部署迁移到使用 Red Hat OpenShift 4.1 的基于 operator 的部署，按照以下步骤进行：

1. 从基于模板的部署创建一个 3scale 备份。
2. 使用 operator 部署 3scale。
3. 在基于 Operator 的部署中恢复备份。
4. 将 3scale WILDCARD_DOMAIN（在本例中为 **3scale.example.com** ）指向 **ocp4.example.com**。

执行所有列出的步骤后，3scale 从模板迁移至基于 Operator 的部署现已完成。

第 2 章 使用模板将 3SCALE 版本 2.11 升级到 2.12 版本

您可以使用 OpenShift 3.11 中的基于模板的部署将 Red Hat 3scale API Management 从 2.11 升级到 2.12 版本。

重要

这是支持使用 OpenShift 3.11 中模板部署的 3scale 的最后一个发行版本。您应该遵循 3scale 迁移指南：从模板到基于 operator 的部署，以便在以后的版本中保持受支持。

重要

要了解所需的条件和程序，请务必先阅读整个升级指南，然后再应用列出的步骤。升级过程会破坏服务的调配，直到过程完成为止。因为这个过程需要涉及到系统中断，请确保计划有一个维护窗口进行。

2.1. 执行升级的先决条件

本节介绍了在基于模板的安装中将 3scale 从 2.11 升级到 2.12 所需的配置、任务和工具。

2.1.1. 配置

- 3scale 支持使用 OpenShift 3.11 中的模板从 2.11 升级到 2.12 的升级路径。

2.1.2. 准备步骤

- 确保 OpenShift CLI 工具已在部署了 3scale 的同一项目中配置。
- 对您用于 3scale 的数据库执行备份。备份过程特定于每种数据库类型和设置。

2.1.3. 工具

您需要这些工具来执行升级：

- 3scale 2.11 部署有 OpenShift 3.11 项目中的模板。
- Bash shell：要运行升级过程中详述的命令，请执行以下操作：
- base64：对机密信息进行编码和解码。
- jq:用于 JSON 转换目的。

2.2. 在基于模板的安装中，从 2.11 升级到 2.12

按照本节所述步骤，在基于模板的安装中将 3scale 2.11 升级到 2.12。

要开始升级，请转至部署了 3scale 的项目。

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

然后，按照以下顺序执行这些步骤：

1. 创建 3scale 项目的备份

2. 删除未使用的 AMP_RELEASE 变量
3. 升级 MySQL 配置
4. 升级 3scale 镜像
5. 其他镜像更改
6. 确认镜像 URL
7. 删除未使用的 MessageBus 变量

2.2.1. 创建 3scale 项目的备份

前一步

无。

当前步骤

此步骤列出了创建 3scale 项目的备份所需的操作。

流程

1. 根据与 3scale 一起使用的数据库，使用以下值之一设置 ${SYSTEM_DB}：
   - 如果数据库是 MySQL，则为 **SYSTEM_DB=system-mysql**。
   - 如果数据库是 PostgreSQL，则为 **SYSTEM_DB=system-postgresql**。
2. 使用现有 DeploymentConfig 创建备份文件：

```
$ 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. 备份通过 **export all** 命令导出的项目中所有现有 OpenShift 资源：

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

4. 使用没有使用 **export all** 命令导出的额外元素创建备份文件：

```
$ 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. 验证所有生成的文件是否都为空，并且所有这些文件都具有预期内容。

后续步骤

删除未使用的 AMP_RELEASE 变量

2.2.2. 删除未使用的 AMP_RELEASE 变量

前一步

创建 3scale 项目的备份

当前步骤

此步骤从 **system-app** 容器 (**system-app** pre hook) 中移除未使用的 AMP_RELEASE 变量，然后验证 AMP_RELEASE 不存在。

流程

1. 从 **system-app** 容器中删除该变量：
   - 请注意变量名称后面的横线字符。

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

2. 从 **system-app** pre hook 中删除该变量：

```
$ INDEX=$(oc get dc system-app -o json | jq
'.spec.strategy.rollingParams.pre.execNewPod.env | map(.name == "AMP_RELEASE") |
index(true)')
oc patch dc/system-app --type=json -p "[{'op': 'remove', 'path':
'/spec/strategy/rollingParams/pre/execNewPod/env/$INDEX'}]"
```

3. 验证 AMP_RELEASE 不存在：

```
$ oc get dc system-app -o yaml | grep AMP_RELEASE
```

后续步骤

升级 MySQL 配置

2.2.3. 升级 MySQL 配置

注意

如果您的 3scale 部署启用了外部数据库模式并使用 MySQL 8.0，请将身份验证插件设置为 3scale 2.12 的 **mysql_native_password**。

在 MySQL 配置文件中添加以下内容：

```
[mysqld]
default_authentication_plugin=mysql_native_password
```

前一步

删除未使用的 AMP_RELEASE 变量

当前步骤

此步骤对 MySQL 配置 **configmap** 进行补丁，以启用升级到 MySQL 8.0。

注意

只有在您当前的 3scale 安装中存在 **system-mysql** 部署时才应遵循这个步骤。

流程

1. 对 **configmap** 进行补丁：

```
$ oc patch configmap/mysql-extra-conf --type merge -p '{"data": {"mysql-default-
authentication-plugin.cnf": "
[mysqld]\ndefault_authentication_plugin=mysql_native_password"}}'
```

2. 验证 **configmap** 已更新：

```
$ oc get cm mysql-extra-conf -o jsonpath='{.data.mysql-default-authentication-plugin\.cnf}'
```

- 应该返回：

```
[mysqld]
default_authentication_plugin=mysql_native_password
```

后续步骤

升级 3scale 镜像

2.2.4. 升级 3scale 镜像

前一步

升级 MySQL 配置

当前步骤

此步骤更新升级过程所需的 3scale 镜像。

2.2.4.1. 对 system 镜像进行补丁

1. 创建新镜像流标签：

```
$ oc patch imagestream/amp-system --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "AMP system 2.12"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/system-rhel7:3scale2.12"}, "name":
"2.12", "referencePolicy": {"type": "Source"}}}]'
```

2. 要继续这个过程，请考虑 3scale 部署中使用的数据库：

- 如果数据库是 Oracle DB，请按照 Patching system 镜像中列出的步骤进行：3scale 使用 Oracle 数据库
- 如果数据库与 Oracle DB 不同，请按照补丁系统镜像中列出的步骤进行操作：3scale 与其他数据库

2.2.4.1.1. 修补系统镜像：3scale 使用 Oracle 数据库

1. 要使用 Oracle 数据库启动对 3scale 的系统镜像进行补丁，您必须构建系统镜像：
   - 从 GitHub 存储库下载 3scale OpenShift 模板并提取存档：

     ```
     tar -xzf 3scale-amp-openshift-templates-3scale-2.12.0-GA.tar.gz
     ```

   - 将 Oracle Database Instant Client Package 文件放在 **3scale-amp-openshift-templates-3scale-2.12.0-GA/amp/system-oracle/oracle-client-files** 目录中。
   - 使用 **-f** 选项运行 **oc process** 命令，并使用 **oc apply** 命令指定 **build.yml** OpenShift 模板，并使用 **-f** 选项指定现有构建：

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

   - 输入 **oc start-build** 命令以构建新系统镜像：

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

2. 对 **system-app** ImageChangeTrigger 进行补丁：
   a. 删除旧的 **2.11-oracle** 触发器：

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

   b. 添加新的特定于版本的触发器：

      ```
      $ oc set triggers dc/system-app --from-image=amp-system:2.12-oracle --
      containers=system-master,system-developer,system-provider
      ```

      这会触发 **system-app** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

3. 对 **system-sidekiq** ImageChange 触发器进行补丁：
   a. 删除旧的 **2.11-oracle** 触发器：

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

   b. 添加新的特定于版本的触发器：

      ```
      $ oc set triggers dc/system-sidekiq --from-image=amp-system:2.12-oracle --
      containers=system-sidekiq,check-svc
      ```

      这会触发 **system-sidekiq** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

4. 对 **system-sphinx** ImageChange 触发器进行补丁：
   a. 删除旧的 **2.11-oracle** 触发器：

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

b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/system-sphinx --from-image=amp-system:2.12-oracle --
containers=system-sphinx,system-master-svc
```

这会触发 **system-sphinx** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

5. 如果要缩减，请缩减 3scale。

2.2.4.1.2. 修补系统镜像：3scale 与其他数据库

1. 对 **system-app** ImageChange 触发器进行补丁：

a. 删除旧的 **2.11** 触发器：

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

b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/system-app --from-image=amp-system:2.12 --containers=system-
master,system-developer,system-provider
```

这会触发 **system-app** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

2. 对 **system-sidekiq** ImageChange 触发器进行补丁：

a. 删除旧的 **2.11** 触发器：

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

b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/system-sidekiq --from-image=amp-system:2.12 --containers=system-
sidekiq,check-svc
```

这会触发 **system-sidekiq** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

3. 对 **system-sphinx** ImageChange 触发器进行补丁：

a. 删除旧的 **2.11** 触发器：

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

b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/system-sphinx --from-image=amp-system:2.12 --containers=system-sphinx,system-master-svc
```

这会触发 **system-sphinx** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

2.2.4.2. 对 **apicast** 镜像进行补丁

1. 对 **amp-apicast** 镜像流进行补丁：

```
$ oc patch imagestream/amp-apicast --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP APIcast 2.12"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.12"}, "name": "2.12", "referencePolicy": {"type": "Source"}}}]'
```

2. 对 **apicast-staging** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/apicast-staging --from-image=amp-apicast:2.12 --containers=apicast-staging
```

   这会触发 **apicast-staging** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

3. 对 **apicast-production** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/apicast-production --from-image=amp-apicast:2.12 --containers=apicast-production,system-master-svc
```

   这会触发 **apicast-production** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

2.2.4.3. 修补 **backend** 镜像

1. 对 **amp-backend** 镜像流进行补丁：

```
$ oc patch imagestream/amp-backend --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "AMP Backend 2.12"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/backend-rhel8:3scale2.12"}, "name": "2.12", "referencePolicy": {"type": "Source"}}}]'
```

2. 对 **backend-listener** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/backend-listener --from-image=amp-backend:2.12 --
   containers=backend-listener
   ```

   这会触发 **backend-listener** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

3. 对 **backend-worker** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/backend-worker --from-image=amp-backend:2.12 --
   containers=backend-worker,backend-redis-svc
   ```

   这会触发 **backend-worker** 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

4. 对 **backend-cron** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/backend-cron --from-image=amp-backend:2.12 --
   containers=backend-cron,backend-redis-svc
   ```

   此命令会触发 **backend-cron** 的重新部署。等待它重新部署、对应的新容器集就绪，并且前面的容器集终止。

2.2.4.4. 对 **zync** 镜像进行补丁

1. 对 **amp-zync** 镜像流进行补丁：

```
$ oc patch imagestream/amp-zync --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value":
{"annotations": {"openshift.io/display-name": "AMP Zync 2.12"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/3scale-amp2/zync-rhel8:3scale2.12"}, "name":
"2.12", "referencePolicy": {"type": "Source"}}}]'
```

2. 对 **zync** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/zync --from-image=amp-zync:2.12 --containers=zync,zync-db-svc
   ```

   这会触发重新部署 **zync**。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

3. 对 **zync-que** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/zync-que --from-image=amp-zync:2.12 --containers=que
   ```

   这会触发重新部署 **zync-que**。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

2.2.4.5. 对 **system-memcached** 镜像进行补丁

1. 对 **system-memcached** 镜像流进行补丁：

```
$ oc patch imagestream/system-memcached --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.12 Memcached"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/3scale-amp2/memcached-rhel7:3scale2.12"}, "name": "2.12", "referencePolicy": {"type": "Source"}}}]'
```

2. 对 **system-memcache** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/system-memcache --from-image=system-memcached:2.12 --containers=memcache
   ```

   这会触发 **system-memcache** DeploymentConfig 的重新部署。等待它重新部署、对应的新容器集就绪，并且旧容器集终止。

2.2.4.6. 对 **zync-database-postgresql** 镜像进行补丁

1. 对 **zync-database-postgresql** 镜像流进行补丁：

```
$ oc patch imagestream/zync-database-postgresql --type=json -p '[{"op": "add", "path":
"/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "Zync 2.12 PostgreSQL"},
"from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"},
"name": "2.12", "referencePolicy": {"type": "Source"}}}]'
```

- 此 patch 命令更新 **zync-database-postgresql** 镜像流，使其包含 2.12 标签。您可以使用这些步骤验证 2.12 标签是否已创建：

  a. 运行这个命令：

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

  b. 检查 *Tags* 列是否显示 **2.12** 标签。

2. 对 **zync-database** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/zync-database --from-image=zync-database-postgresql:2.12 --
   containers=postgresql
   ```

   如果镜像有新的更新，此补丁还可能触发重新部署 **zync-database** DeploymentConfig。如果发生这种情况，请等待新容器集重新部署并就绪，并且旧容器集终止。

2.2.4.7. 其他镜像更改

如果您的 3scale 2.11 安装中提供了一个或多个 DeploymentConfig，请点击适用链接来获取如何继续的更多信息：

- **backend-redis** DeploymentConfig
- **system-redis** DeploymentConfig
- **system-mysql** DeploymentConfig
- **system-postgresql** DeploymentConfig
- 确认镜像 URL

backend-redis DeploymentConfig

如果在当前 3scale 安装中存在 **backend-redis** DeploymentConfig，请为 **backend-redis** 对 **redis** 镜像进行补丁：

1. 对 **backend-redis** 镜像流进行补丁：

```
$ oc patch imagestream/backend-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "Backend 2.12 Redis"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/rhscl/redis-5-rhel7:5"}, "name": "2.12",
"referencePolicy": {"type": "Source"}}}]'
```

这个补丁更新了 backend-redis 镜像流，使其包含 2.12 标签。使用以下命令，如果 *Tags* 列显示了 2.12，则确认标签已创建：

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

2. 对 **backend-redis** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/backend-redis --from-image=backend-redis:2.12 --containers=backend-redis
   ```

   如果镜像有新的更新，此补丁还可能触发 **backend-redis** DeploymentConfig 的重新部署。如果发生这种情况，请等待新容器集重新部署并就绪，并且旧容器集终止。

system-redis DeploymentConfig

如果在当前 3scale 安装中存在 **system-redis** DeploymentConfig，请为 **system-redis** 对 **redis** 镜像进行补丁。

1. 对 **system-redis** 镜像流进行补丁：

```
$ oc patch imagestream/system-redis --type=json -p '[{"op": "add", "path": "/spec/tags/-", "value": {"annotations": {"openshift.io/display-name": "System 2.12 Redis"}, "from": { "kind": "DockerImage", "name": "registry.redhat.io/rhscl/redis-5-rhel7:5"}, "name": "2.12", "referencePolicy": {"type": "Source"}}}]'
```

此补丁更新了 **system-redis** 镜像流，使其包含 2.12 标签。使用以下命令，如果 *Tags* 列显示了 2.12，则确认标签已创建：

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

2. 对 **system-redis** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

   ```
   $ oc set triggers dc/system-redis --from-image=system-redis:2.12 --containers=system-redis
   ```

   如果镜像有新的更新，此补丁还可能触发 **system-redis** DeploymentConfig 的重新部署。如果发生这种情况，请等待新容器集重新部署并就绪，并且旧容器集终止。

system-mysql DeploymentConfig

如果当前 3scale 安装中存在 **system-mysql** DeploymentConfig，请为 **system-mysql** 修补 MySQL 镜像。

1. 对 **system-mysql** 镜像流进行补丁：

```
$ oc patch imagestream/system-mysql --type=json -p '[{"op": "add", "path": "/spec/tags/-",
"value": {"annotations": {"openshift.io/display-name": "System 2.12 MySQL"}, "from": { "kind":
"DockerImage", "name": "registry.redhat.io/rhel8/mysql-80:1"}, "name": "2.12",
"referencePolicy": {"type": "Source"}}}]'
```

此补丁更新了 **system-mysql** 镜像流，使其包含 2.12 标签。使用以下命令，如果 *Tags* 列显示了 2.12，则确认标签已创建：

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

2. 对 **system-mysql** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

   b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/system-mysql --from-image=system-mysql:2.12 --
containers=system-mysql
```

   如果镜像上有新的更新，此补丁还可能触发 **system-mysql** DeploymentConfig 的重新部署。如果发生这种情况，请等待新容器集重新部署并就绪，并且旧容器集终止。

system-postgresql DeploymentConfig

如果当前 3scale 安装中存在 **system-postgresql** DeploymentConfig，请为 **system-postgresql** 修补 PostgreSQL 镜像。

1. 对 **system-postgresql** 镜像流进行补丁：

```
$ oc patch imagestream/system-postgresql --type=json -p '[{"op": "add", "path": "/spec/tags/-
", "value": {"annotations": {"openshift.io/display-name": "System 2.12 PostgreSQL"}, "from": {
"kind": "DockerImage", "name": "registry.redhat.io/rhscl/postgresql-10-rhel7"}, "name": "2.12",
"referencePolicy": {"type": "Source"}}}]'
```

这个补丁更新了 **system-postgresql** 镜像流，使其包含 2.12 标签。使用以下命令，如果 *Tags* 列显示了 2.12，则确认标签已创建：

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

2. 对 **system-postgresql** ImageChange 触发器进行补丁：

   a. 删除旧的 **2.11** 触发器：

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

b. 添加新的特定于版本的触发器：

```
$ oc set triggers dc/system-postgresql --from-image=system-postgresql:2.12 --
containers=system-postgresql
```

如果镜像上有新的更新，此补丁还可能触发 **system-postgresql** DeploymentConfig 的重新部署。如果发生这种情况，请等待新容器集重新部署并就绪，并且旧容器集终止。

2.2.4.8. 确认镜像 URL

确认 DeploymentConfig 的所有镜像 URL 包含新的镜像 registry URL，并在各个 URL 地址末尾添加一个哈希：

```
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
```

后续步骤

删除未使用的 MessageBus 变量

2.2.5. 删除未使用的 MessageBus 变量

前一步

升级 3scale 镜像

当前步骤

此步骤删除未使用的 MESSAGE_BUS_REDIS_* 变量。

2.2.5.1. 从 system-app deploymentconfig 中删除 MESSAGE_BUS_REDIS_* 变量

1. 从 **system-app** 容器中移除 MESSAGE_BUS_REDIS_* 变量：

   - 请注意变量名称后面的横线字符。

```
$ oc set env dc/system-app MESSAGE_BUS_REDIS_URL-
$ oc set env dc/system-app MESSAGE_BUS_REDIS_NAMESPACE-
$ oc set env dc/system-app MESSAGE_BUS_REDIS_SENTINEL_HOSTS-
$ oc set env dc/system-app MESSAGE_BUS_REDIS_SENTINEL_ROLE-
```

2. 从 **system-app** pre hook 中删除 MESSAGE_BUS_REDIS_* 变量：

```
$ INDEX=$(oc get dc system-app -o json | jq
'.spec.strategy.rollingParams.pre.execNewPod.env | map(.name ==
"MESSAGE_BUS_REDIS_URL") | index(true)')
oc patch dc/system-app --type=json -p "[{'op': 'remove', 'path':
'/spec/strategy/rollingParams/pre/execNewPod/env/$INDEX'}]"

$ INDEX=$(oc get dc system-app -o json | jq
'.spec.strategy.rollingParams.pre.execNewPod.env | map(.name ==
"MESSAGE_BUS_REDIS_NAMESPACE") | index(true)')
```

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

$ INDEX=$(oc get dc system-app -o json | jq
'.spec.strategy.rollingParams.pre.execNewPod.env | map(.name ==
"MESSAGE_BUS_REDIS_SENTINEL_HOSTS") | index(true)')
oc patch dc/system-app --type=json -p "[{'op': 'remove', 'path':
'/spec/strategy/rollingParams/pre/execNewPod/env/$INDEX'}]"

$ INDEX=$(oc get dc system-app -o json | jq
'.spec.strategy.rollingParams.pre.execNewPod.env | map(.name ==
"MESSAGE_BUS_REDIS_SENTINEL_ROLE") | index(true)')
oc patch dc/system-app --type=json -p "[{'op': 'remove', 'path':
'/spec/strategy/rollingParams/pre/execNewPod/env/$INDEX'}]"
```

3. 验证 MESSAGE_BUS_REDIS_* 环境变量不存在：

```
$ oc get dc system-app -o yaml | grep MESSAGE_BUS_REDIS
```

2.2.5.2. 从 system-sidekiq deploymentconfig 中删除 MESSAGE_BUS_REDIS_* 变量

1. 从 **system-sidekiq** 容器中删除 MESSAGE_BUS_REDIS_* 变量：

   - 请注意变量名称后面的横线字符。

```
$ oc set env dc/system-sidekiq MESSAGE_BUS_REDIS_URL-
$ oc set env dc/system-sidekiq MESSAGE_BUS_REDIS_NAMESPACE-
$ oc set env dc/system-sidekiq MESSAGE_BUS_REDIS_SENTINEL_HOSTS-
$ oc set env dc/system-sidekiq MESSAGE_BUS_REDIS_SENTINEL_ROLE-
```

2. 从 **system-sidekiq** init-container 中删除 MESSAGE_BUS_REDIS_* 变量：

```
$ INDEX=$(oc get dc system-sidekiq -o json | jq '.spec.template.spec.initContainers[].env |
map(.name == "MESSAGE_BUS_REDIS_URL") | index(true)')
oc patch dc/system-sidekiq --type=json -p "[{'op': 'remove', 'path':
'/spec/template/spec/initContainers/0/env/$INDEX'}]"

$ INDEX=$(oc get dc system-sidekiq -o json | jq '.spec.template.spec.initContainers[].env |
map(.name == "MESSAGE_BUS_REDIS_NAMESPACE") | index(true)')
oc patch dc/system-sidekiq --type=json -p "[{'op': 'remove', 'path':
'/spec/template/spec/initContainers/0/env/$INDEX'}]"

$ INDEX=$(oc get dc system-sidekiq -o json | jq '.spec.template.spec.initContainers[].env |
map(.name == "MESSAGE_BUS_REDIS_SENTINEL_HOSTS") | index(true)')
oc patch dc/system-sidekiq --type=json -p "[{'op': 'remove', 'path':
'/spec/template/spec/initContainers/0/env/$INDEX'}]"

$ INDEX=$(oc get dc system-sidekiq -o json | jq '.spec.template.spec.initContainers[].env |
map(.name == "MESSAGE_BUS_REDIS_SENTINEL_ROLE") | index(true)')
oc patch dc/system-sidekiq --type=json -p "[{'op': 'remove', 'path':
'/spec/template/spec/initContainers/0/env/$INDEX'}]"
```

3. 验证 MESSAGE_BUS_REDIS_* 环境变量不存在：

```
$ oc get dc system-sidekiq -o yaml | grep MESSAGE_BUS_REDIS
```

2.2.5.3. 从 system-sphinx deploymentconfig 中删除 MESSAGE_BUS_REDIS_* 变量

1. 从 **system-sphinx** 容器中删除 MESSAGE_BUS_REDIS_* 变量：
   - 请注意变量名称后面的横线字符。

```
$ oc set env dc/system-sphinx MESSAGE_BUS_REDIS_URL-
$ oc set env dc/system-sphinx MESSAGE_BUS_REDIS_NAMESPACE-
$ oc set env dc/system-sphinx MESSAGE_BUS_REDIS_SENTINEL_HOSTS-
$ oc set env dc/system-sphinx MESSAGE_BUS_REDIS_SENTINEL_ROLE-
```

2. 验证 MESSAGE_BUS_REDIS_* 环境变量不存在：

```
$ oc get dc system-sphinx -o yaml | grep MESSAGE_BUS_REDIS
```

2.2.5.4. 从 system-redis secret 中删除 MESSAGE_BUS_REDIS_* 变量

1. 从 **system-redis** secret 中删除 MESSAGE_BUS_REDIS_* 变量：

```
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_URL'}]"
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_NAMESPACE'}]"
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_SENTINEL_HOSTS'}]"
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_SENTINEL_ROLE'}]"
```

2. 验证 MESSAGE_BUS_REDIS_* 环境变量不存在：

```
$ oc get secret system-redis -o yaml | grep MESSAGE_BUS
```

2.2.5.5. 使用 PostgreSQL 10 和 PostgreSQL 13 使用外部 system-database 升级

此升级支持使用 PostgreSQL 10 的外部 **system-database**。您应当先完成 3scale 升级，然后升级到 PostgreSQL 13。

后续步骤

无。执行所有列出的步骤后，基于模板的部署中的 3scale 从 2.11 升级到 2.12 现已完成。

2.3. 在基于模板的安装中使用 ORACLE 数据库升级 3SCALE

本节介绍在 OpenShift 3.11 基于模板的安装中使用带有 Oracle 数据库的 3scale 系统镜像时，如何更新 Red Hat 3scale API 管理。

先决条件

带有 Oracle 数据库的 3scale 安装.请参阅使用 Oracle 数据库设置 3scale 系统镜像。

要在基于模板的安装中使用 Oracle 数据库升级 3scale 系统镜像，请执行以下步骤：

- 使用 Oracle 19c 升级 3scale

2.3.1. 使用 Oracle 19c 升级 3scale

此流程指导您从现有 3scale 2.11 安装中对 3scale 2.12 进行 3scale 2.12 的 Oracle 数据库 19c 更新过程。

重要：丢失与数据库的连接可能会损坏 3scale。在继续升级前进行备份。如需更多信息，请参阅 Oracle 数据库文档：Oracle 数据库备份和恢复用户指南。

先决条件

- 3scale 2.11 安装。
- Oracle 数据库 19c 安装。
  - 有关使用 Oracle 配置 3scale 的更多信息，请参阅 准备 Oracle 数据库。

流程

1. 从 GitHub 存储库下载 3scale OpenShift 模板并提取存档：

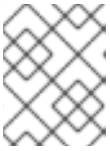
```
tar -xzf 3scale-amp-openshift-templates-3scale-2.12.0-GA.tar.gz
```

2. 将 Oracle Database Instant Client Package 文件放在 **3scale-amp-openshift-templates-3scale-2.12.0-GA/amp/system-oracle/oracle-client-files** 目录中。
3. 使用 **-f** 选项运行 **oc process** 命令并指定 **build.yml** OpenShift 模板：

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

4. 使用 **-f** 选项运行 **oc new-app** 命令以指示 **amp.yml** OpenShift 模板，并使用 **-p** 选项指定带有 OpenShift 集群域的 **WILDCARD_DOMAIN** 参数：

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



注意

以下步骤是可选的。如果在安装或系统升级后删除 **ORACLE_SYSTEM_PASSWORD**，请使用它们。

5. 输入以下 **oc patch** 命令，将 **SYSTEM_PASSWORD** 替换为您在准备 Oracle 数据库中设置的 Oracle Database **system** 密码：

```
$ 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. 输入以下命令，替换 **DATABASE_URL** 以指向在准备 Oracle 数据库中指定的 Oracle 数据库 ：

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

7. 输入 **oc start-build** 命令以构建新系统镜像：

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

8. 等待构建完成。要查看构建的状态，请运行以下命令：

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

   a. 等待构建处于 Complete 状态。

9. 使用 Oracle 数据库设置 3scale 系统镜像后，从 **system-app** DeploymentConfig 中删除 **ORACLE_SYSTEM_PASSWORD**。在升级到新版本的 3scale 前，不需要再次执行它。

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

open_cursors 参数设置验证

您必须确认此数据库中的 **open_cursors** 参数设置为大于 **1000** 的值。

要做到这一点，以 SYSTEM 用户身份登录 Oracle 数据库并运行以下命令：

```
show parameter open_cursors;
```

返回值应至少为 **1000**。如果没有，请在 Oracle 对 **开放光**标 的下面放置内容将参数更改为大于 **1000** 的值。

如果 **open_cursors** 参数之前配置为小于 **1000** 的一些限制，并且您不增加值，您可能会在其中一个 OpenShift **system-app** pod 日志中看到以下错误：

```
ORA-01000: maximum open cursors exceeded
```

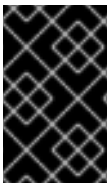
其他资源

有关 3scale 和 Oracle 数据库支持的更多信息，请参阅 Red Hat 3scale API 管理支持的配置。

第 3 章 3SCALE 基于 OPERATOR 的升级指南：从 2.11 升级到 2.12

在基于 operator 的安装中，将 Red Hat 3scale API Management 从 2.11 升级到 2.12，以在 OpenShift 4.x 上管理 3scale。

要自动获得 3scale 的微版本，请确保自动更新已经启动。要进行检查，请参阅 为微发行版本设置 3scale operator。



重要

为了了解所需的条件和程序，请在应用列出的步骤前阅读整个升级指南。升级过程会破坏服务的调配，直到过程完成为止。因为这个过程需要涉及到系统中断，请确保计划有一个维护窗口进行。

3.1. 执行升级的先决条件

本节介绍了在基于 Operator 的安装中，将 3scale 从 2.11 升级到 2.12 所需的配置。

- 具有管理员访问权限的 OpenShift Container Platform (OCP) 4.6, 4.7, 4.8, 4.9, 或 4.10 集群。
- 3scale 2.11 以前通过 3scale Operator 部署。
- 确保使用 **threescale-2.11** 频道的最新 CSV。检查它：
  - 如果订阅的批准设置是 *automatic* ，您应该已位于该频道的最新 CSV 版本。
  - 如果订阅的批准设置是 *manual*，请确保您批准所有待处理的 *InstallPlans* 并具有最新的 CSV 版本。
  - 请记住，如果有一个待处理的安装计划，可能还有更多待处理的安装计划，只有在安装了现有待定计划后才会显示这些计划。

3.2. 在基于 OPERATOR 的安装中，从 2.11 升级到 2.12

在基于 Operator 的部署中，将 3scale 从 2.11 升级到 2.12：

1. 使用具有管理员特权的帐户登录 OCP 控制台。
2. 选择部署了 *3scale-operator* 的项目。
3. 点 **Operators > Installed Operators**。
4. 选择 **Red Hat Integration - 3scale > Subscription > Channel**。
5. 选择 *3scale-2.12* 并保存更改，以编辑订阅的频道。
   这将开始升级过程。
6. 查询项目中 pod 的状态，直到您看到所有新版本都在运行并就绪且没有错误：

```
$ oc get pods
```

注意

- pod 在升级过程中可能会出现临时错误。
- 升级 pod 所需的时间可能从 5 到 10 分钟。

7. 新 pod 版本运行后，通过登录 3scale 管理门户并检查它是否按预期工作，确认升级是否成功。
8. 运行以下命令，检查 *APIManager* 对象的状态并获取 *YAML* 内容。<myapimanager> 代表 *APIManager* 的名称：

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

- 带有值的新注解应如下所示：

```
apps.3scale.net/apimanager-threescale-version: "2.12"
apps.3scale.net/threescale-operator-version: "0.9.0"
```

执行所有步骤后，基于 Operator 的部署中的 3scale 从 2.11 升级到 2.12 已完成。

3.2.1. 将外部 MySQL 版本升级到 8.0

如果您的 3scale 部署启用了外部数据库模式并使用 MySQL 8.0，请将身份验证插件设置为 3scale 2.12 的 **mysql_native_password**。

在 MySQL 配置文件中添加以下内容：

```
[mysqld]
default_authentication_plugin=mysql_native_password
```

3.2.2. 删除未使用的 MessageBus 变量

如果您的 3scale 部署启用了外部数据库模式，您必须手动从 **system-redis** secret 中删除 **MESSAGE_BUS_*** 字段。

1. 从 **system-redis** secret 中删除 **MESSAGE_BUS_*** 字段。如果 secret 不是由外部控制器管理，使用以下命令手动更新：

```
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_URL'}]"
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_NAMESPACE'}]"
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_SENTINEL_HOSTS'}]"
$ oc patch secret/system-redis --type=json -p "[{'op': 'remove', 'path':
'/data/MESSAGE_BUS_SENTINEL_ROLE'}]"
```

2. 验证 **MESSAGE_BUS_*** 字段不存在。以下命令应该返回空的：

```
$ oc get secret system-redis -o yaml | grep MESSAGE_BUS
```

3.2.3. 使用 PostgreSQL 10 和 PostgreSQL 13 使用外部 system-database 升级

此升级支持使用 PostgreSQL 10 的外部 **system-database**。您应当先完成 3scale 升级，然后升级到 PostgreSQL 13。

第 4 章 基于 APICAST OPERATOR 的升级指南：从 2.11 升级到 2.12

在基于 Operator 的安装中，将 APIcast 从 2.11 升级到 2.12，可帮助您使用 APIcast API 网关将您的内部和外部 API 服务与 3scale 集成。

重要

为了了解所需的条件和程序，请在应用列出的步骤前阅读整个升级指南。升级过程会破坏服务的调配，直到过程完成为止。因为这个过程需要涉及到系统中断，请确保计划有一个维护窗口进行。

4.1. 执行升级的先决条件

要在基于 operator 的安装过程中执行 APIcast 从 2.11 升级到 2.12，需要满足以下先决条件：

- 具有管理员访问权限的 OpenShift Container Platform (OCP) 4.6, 4.8, 4.9, 或 4.10 集群。
- APIcast 2.11 之前通过 APIcast operator 部署。
- 确保使用 **threescale-2.11** 频道的最新 CSV。检查它：
  - 如果订阅的批准设置是 *automatic* ，您应该已位于该频道的最新 CSV 版本。
  - 如果订阅的批准设置是 *manual*，请确保您批准所有待处理的 *InstallPlans* 并具有最新的 CSV 版本。
  - 请记住，如果有一个待处理的安装计划，可能还有更多待处理的安装计划，只有在安装了现有待定计划后才会显示这些计划。

4.2. 在基于 OPERATOR 的安装中，将 APICAST 从 2.11 升级到 2.12

在基于 Operator 的安装中，将 APIcast 从 2.11 升级到 2.12，以便 APIcast 可以在 3scale 安装中作为 API 网关运行。

流程

1. 使用具有管理员特权的帐户登录 OCP 控制台。
2. 选择部署了 *APIcast Operator* 的项目。
3. 点 **Operators > Installed Operators**。
4. 在 **Subscription > Channel** 中，选择 *Red Hat Integration - 3scale APIcast gateway* 。
5. 选择 *3scale-2.12* 频道并保存更改，以编辑订阅的频道。
   这将开始升级过程。
6. 查询项目中 pod 的状态，直到您看到所有新版本都在运行并就绪且没有错误：

```
$ oc get pods
```

注意

- pod 在升级过程中可能会出现临时错误。
- 升级 pod 所需的时间可能从 5 到 10 分钟。

7. 运行以下命令，检查 *APIcast* 对象的状态，并获取 *YAML* 内容：

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

- 带有值的新注解应如下所示：

```
apicast.apps.3scale.net/operator-version: “0.6.0”
```

执行所有列出的步骤后，基于 operator 的部署中的 APIcast 从 2.11 升级到 2.12 现已完成。