Red Hat 3scale API Management 2.11

Installing 3scale

Install and configure 3scale API Management.

Last Updated: 2023-03-29

Red Hat 3scale API Management 2.11 Installing 3scale

Install and configure 3scale API Management.

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide provides the information to install and configure 3scale API Management.

Table of Contents

PREFACE

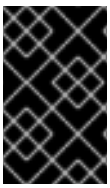
This guide will help you to install and configure 3scale

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. REGISTRY SERVICE ACCOUNTS FOR 3SCALE

To use container images from **registry.redhat.io** in a shared environment with 3scale 2.11, you must use a *Registry Service Account* instead of an individual user's *Customer Portal* credentials.



IMPORTANT

It is a requirement of deploying 3scale that you follow the steps outlined in this chapter before deploying either on OpenShift using a template or via the operator, as both options use registry authentication.

To create and modify a registry service account, perform the steps outlined in the following sections:

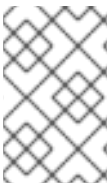
- Section 1.1, "Creating a registry service account"
- Section 1.2, "Modifying a registry service account"

1.1. CREATING A REGISTRY SERVICE ACCOUNT

To create a registry service account, follow the procedure below.

Procedure

1. Navigate to the Registry Service Accounts page and log in.
2. Click **New Service Account**
3. Fill in the form on the *Create a New Registry Service Account* page.
   a. Add a name for the *service account*.
   Note: You will see a fixed-length, randomly generated numerical string before the form field.
   b. Enter a *Description*.
   c. Click **Create**.
4. Navigate back to your *Service Accounts*.
5. Click the *Service Account* you created.
6. Make a note of the username, including the prefix string, for example **12345678|username**, and your password. This username and password will be used to log in to **registry.redhat.io**.



NOTE

There are tabs available on the *Token Information* page that show you how to use the authentication token. For example, the *Token Information* tab shows the username in the format **12345678|username** and the password string below it.

1.2. MODIFYING A REGISTRY SERVICE ACCOUNT

You can edit or delete service accounts from the *Registry Service Account* page, by using the pop-up menu to the right of each authentication token in the table.

WARNING

The regeneration or removal of *service accounts* will impact systems that are using the token to authenticate and retrieve content from **registry.redhat.io**.

A description for each function is as follows:

- **Regenerate token:** Allows an authorized user to reset the password associated with the *Service Account*.
  Note: You cannot modify the username for the *Service Account*.
- **Update Description:** Allows an authorized user to update the description for the *Service Account*.
- **Delete Account:** Allows an authorized user to remove the *Service Account*.

1.3. ADDITIONAL RESOURCES

- Red Hat Container Registry Authentication
- Authentication enabled Red Hat registry

CHAPTER 2. INSTALLING 3SCALE ON OPENSHIFT

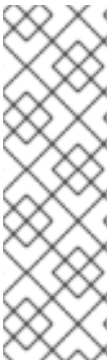
This section walks you through steps to deploy Red Hat 3scale API Management 2.11 on OpenShift.

The Red Hat 3scale API Management solution for on-premises deployment is composed of:

- Two API gateways: embedded APIcast
- One 3scale Admin Portal and Developer Portal with persistent storage

There are two ways to deploy a 3scale solution:

- Section 2.3, "Deploying 3scale on OpenShift using a template"
- Section 2.6, "Deploying 3scale using the operator"



NOTE

- Whether deploying 3scale using the operator or via templates, you must first configure registry authentication to the Red Hat container registry. See Configuring container registry authentication.
- The 3scale Istio Adapter is available as an optional adapter that allows labeling a service running within the Red Hat OpenShift Service Mesh, and integrate that service with Red Hat 3scale API Management. Refer to 3scale adapter documentation for more information.

Prerequisites

- You must configure 3scale servers for UTC (Coordinated Universal Time).
- Create user credentials using the step in Chapter 1, *Registry service accounts for 3scale*

To install 3scale on OpenShift, perform the steps outlined in the following sections:

- Section 2.1, "System requirements for installing 3scale on OpenShift"
- Section 2.2, "Configuring nodes and entitlements"
- Section 2.3, "Deploying 3scale on OpenShift using a template"
- Section 2.5, "Parameters of the 3scale template"
- Section 2.6, "Deploying 3scale using the operator"
- Section 2.7, "Deployment configuration options for 3scale on OpenShift using the operator"
- Section 2.8, "Installing 3scale with the operator using Oracle as the system database"
- Section 2.9, "Troubleshooting common 3scale installation issues"

2.1. SYSTEM REQUIREMENTS FOR INSTALLING 3SCALE ON OPENSHIFT

This section lists the requirements for the 3scale - OpenShift template.

2.1.1. Environment requirements

Red Hat 3scale API Management requires an environment specified in supported configurations.

If you are using local filesystem storage:

Persistent volumes

- 3 RWO (ReadWriteOnce) persistent volumes for Redis and MySQL persistence
- 1 RWX (ReadWriteMany) persistent volume for Developer Portal content and System-app Assets

Configure the RWX persistent volume to be group writable. For a list of persistent volume types that support the required access modes, see the OpenShift documentation.

If you are using an Amazon Simple Storage Service (Amazon S3) bucket for content management system (CMS) storage:

Persistent volumes

- 3 RWO (ReadWriteOnce) persistent volumes for Redis and MySQL persistence

Storage

- 1 Amazon S3 bucket
- Network File System (NFS)

2.1.2. Hardware requirements

Hardware requirements depend on your usage needs. Red Hat recommends that you test and configure your environment to meet your specific requirements. The following are the recommendations when configuring your environment for 3scale on OpenShift:

- Compute optimized nodes for deployments on cloud environments (AWS c4.2xlarge or Azure Standard_F8).
- Very large installations may require a separate node (AWS M4 series or Azure Av2 series) for Redis if memory requirements exceed your current node's available RAM.
- Separate nodes between routing and compute tasks.
- Dedicated computing nodes for 3scale specific tasks.
- Set the **PUMA_WORKERS** variable of the back-end listener to the number of cores in your compute node.

2.2. CONFIGURING NODES AND ENTITLEMENTS

Before deploying 3scale on OpenShift, you must configure the necessary nodes and the entitlements for the environment to fetch images from the Red Hat Ecosystem Catalog . Perform the following steps to configure the nodes and entitlements:

Procedure

1. Install Red Hat Enterprise Linux (RHEL) on each of your nodes.
2. Register your nodes with Red Hat using the Red Hat Subscription Manager (RHSM), via the interface or the command line.
3. Attach your nodes to your 3scale subscription using RHSM.
4. Install OpenShift on your nodes, complying with the following requirements:
   - Use a supported OpenShift version.
   - Configure persistent storage on a file system that supports multiple writes.
5. Install the OpenShift command line interface.
6. Enable access to the **rhel-7-server-3scale-amp-2-rpms** repository using the subscription manager:

```
sudo subscription-manager repos --enable=rhel-7-server-3scale-amp-2-rpms
```

7. Install the 3scale template called **3scale-amp-template**. This will be saved at **/opt/amp/templates**.

```
sudo yum install 3scale-amp-template
```

2.2.1. Configuring Amazon Simple Storage Service

IMPORTANT

Skip this section, if you are deploying 3scale with the local filesystem storage.

If you want to use an Amazon Simple Storage Service (Amazon S3) bucket as storage, you must configure your bucket before you can deploy 3scale on OpenShift.

Perform the following steps to configure your Amazon S3 bucket for 3scale:

1. Create an Identity and Access Management (IAM) policy with the following minimum permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "arn:aws:s3:::*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::targetBucketName",
        "arn:aws:s3:::targetBucketName/*"
      ]
```

```
      }
    ]
}
```

2. Create a CORS configuration with the following rules:

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
   <AllowedOrigin>https://*</AllowedOrigin>
   <AllowedMethod>GET</AllowedMethod>
</CORSRule>
</CORSConfiguration>
```

2.3. DEPLOYING 3SCALE ON OPENSHIFT USING A TEMPLATE

NOTE

OpenShift Container Platform (OCP) 4.x supports deployment of 3scale using the operator only. See Deploying 3scale using the operator.

Prerequisites

- An OpenShift cluster configured as specified in the Configuring nodes and entitlements section.
- A domain that resolves to your OpenShift cluster.
- Access to the Red Hat Ecosystem Catalog.
- (Optional) An Amazon Simple Storage Service (Amazon S3) bucket for content management system (CMS) storage outside of the local filesystem.
- (Optional) A deployment with PostgreSQL.
    - This is the same as the default deployment on Openshift, however it uses PostgreSQL as an internal system database.
- (Optional) A working SMTP server for email functionality.

NOTE

Deploying 3scale on OpenShift using a template is based on OpenShift Container Platform 3.11

Follow these procedures to install 3scale on OpenShift using a **.yml** template:

- Configuring container registry authentication
- Creating registry service accounts
- Modifying registry service accounts
- Importing the 3scale template

- Getting the Admin Portal URL
- Deploying 3scale with Amazon Simple Storage Service
- Deploying 3scale with PostgreSQL
- Configuring SMTP variables

2.4. CONFIGURING CONTAINER REGISTRY AUTHENTICATION

As a 3scale administrator, configure authentication with **registry.redhat.io** before you deploy 3scale container images on OpenShift.

Prerequisites

- Cluster administrator access to an OpenShift Container Platform cluster.
- OpenShift **oc** client tool is installed. For more details, see the OpenShift CLI documentation.

Procedure

1. Log into your OpenShift cluster as administrator:

```
$ oc login -u system:admin
```

2. Open the project in which you want to deploy 3scale:

```
oc project your-openshift-project
```

3. Create a **docker-registry** secret using your Red Hat Customer Portal account, replacing **threescale-registry-auth** with the secret to create:

```
$ oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username=CUSTOMER_PORTAL_USERNAME \
  --docker-password=CUSTOMER_PORTAL_PASSWORD \
  --docker-email=EMAIL_ADDRESS
```

   You will see the following output:

```
secret/threescale-registry-auth created
```

4. Link the secret to your service account to use the secret for pulling images. The service account name must match the name that the OpenShift pod uses. This example uses the **default** service account:

```
$ oc secrets link default threescale-registry-auth --for=pull
```

5. Link the secret to the **builder** service account to use the secret for pushing and pulling build images:

```
$ oc secrets link builder threescale-registry-auth
```

Additional resources

For more details on authenticating with Red Hat for container images:

- Red Hat container image authentication
- Red Hat registry service accounts

2.4.1. Creating registry service accounts

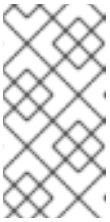
To use container images from **registry.redhat.io** in a shared environment with 3scale 2.11 deployed on OpenShift, you must use a *Registry Service Account* instead of an individual user's *Customer Portal* credentials.

NOTE

It is a requirement for 3scale 2.8 and greater that you follow the steps outlined below before deploying either on OpenShift using a template or via the operator, as both options use registry authentication.

Procedure

1. Navigate to the Registry Service Accounts page and log in.
2. Click **New Service Account**. Fill in the form on the *Create a New Registry Service Account* page.
    a. Add a name for the *service account*.
    Note: You will see a fixed-length, randomly generated number string before the form field.
3. Enter a *Description*.
4. Click **Create**.
5. Navigate back to your *Service Accounts*.
6. Click the *Service Account* you created.
7. Make a note of the username, including the prefix string, for example **12345678|username**, and your password.
    a. This username and password is used to log in to **registry.redhat.io**.



NOTE

There are tabs available on the Token Information page that show you how to use the authentication token. For example, the Token Information tab shows the username in the format **12345678|username** and the password string below it.

2.4.2. Modifying registry service accounts

Service accounts can be modified or deleted. This can done from the *Registry Service Account* page using the pop-up menu to the right of each authentication token in the table.

WARNING

The regeneration or removal of service accounts impacts systems that are using the token to authenticate and retrieve content from **registry.redhat.io**.

A description for each function is as follows:

- **Regenerate token:** Allows an authorized user to reset the password associated with the *Service Account*.
  Note: The username for the *Service Account* cannot be changed.
- **Update Description:** Allows an authorized user to update the description for the *Service Account*.
- **Delete Account:** Allows an authorized user to remove the *Service Account*.

Additional resources

- Red Hat Container Registry Authentication
- Authentication enabled Red Hat registry

2.4.3. Importing the 3scale template

NOTE

- Wildcard routes have been removed as of 3scale 2.6.
  - This functionality is handled by Zync in the background.
- When API providers are created, updated, or deleted, routes automatically reflect those changes.

Perform the following steps to import the 3scale template into your OpenShift cluster:

Procedure

1. From a terminal session log in to OpenShift as the cluster administrator:

```
oc login
```

2. Select your project, or create a new project:

```
oc project <project_name>
```

```
oc new-project <project_name>
```

3. Enter the **oc new-app** command:

a. Specify the **--file** option with the path to the *amp.yml* file you downloaded as part of Configuring nodes and entitlements.

b. Specify the **--param** option with the **WILDCARD_DOMAIN** parameter set to the domain of your OpenShift cluster:

```
oc new-app --file /opt/amp/templates/amp.yml --param WILDCARD_DOMAIN=
<WILDCARD_DOMAIN>
```

The terminal shows the master and tenant URLs and credentials for your newly created 3scale Admin Portal. This output should include the following information:

- master admin username
- master password
- master token information
- tenant username
- tenant password
- tenant token information

4. Log in to https://user-admin.3scale-project.example.com as admin/xXxXyz123.

```
* With parameters:

 * ADMIN_PASSWORD=xXxXyz123 # generated
 * ADMIN_USERNAME=admin
 * TENANT_NAME=user

 * MASTER_NAME=master
 * MASTER_USER=master
 * MASTER_PASSWORD=xXxXyz123 # generated

--> Success
Access your application via route 'user-admin.3scale-project.example.com'
Access your application via route 'master-admin.3scale-project.example.com'
Access your application via route 'backend-user.3scale-project.example.com'
Access your application via route 'user.3scale-project.example.com'
Access your application via route 'api-user-apicast-staging.3scale-project.example.com'
Access your application via route 'api-user-apicast-production.3scale-project.example.com'
```

5. Make a note of these details for future reference.

6. The 3scale deployment on OpenShift has been successful when the following command returns:

```
oc wait --for=condition=available --timeout=-1s $(oc get dc --output=name)
```

NOTE

When the 3scale deployment on OpenShift has been successful, your login credentials will work.

2.4.4. Getting the Admin Portal URL

When you deploy 3scale using the template, a default tenant is created, with a fixed URL: **3scale-admin.${wildcardDomain}**

The 3scale Dashboard shows the new portal URL of the tenant. As an example, if the *<wildCardDomain>* is **3scale-project.example.com**, the Admin Portal URL is: **https://3scale-admin.3scale-project.example.com**.

The **wildcardDomain** is the *<wildCardDomain>* parameter you provided during installation. Open this unique URL in a browser using the this command:

```
xdg-open https://3scale-admin.3scale-project.example.com
```

Optionally, you can create new tenants on the *MASTER portal URL*: **master.${wildcardDomain}**

2.4.5. Deploying 3scale with Amazon Simple Storage Service

Deploying 3scale with Amazon Simple Storage Service (Amazon S3) is an optional procedure. Deploy 3scale with Amazon S3 using the following steps:

Procedure

1. Download amp-s3.yml.
2. Log in to OpenShift from a terminal session :

```
oc login
```

3. Select your project, or create a new project:

```
oc project <project_name>
```

OR

```
oc new-project <project_name>
```

1. Enter the oc new-app command:
   - Specify the **--file** option with the path to the *amp-s3.yml* file.
   - Specify the **--param** options with the following values:
     - **WILDCARD_DOMAIN**: the parameter set to the domain of your OpenShift cluster.
     - **AWS_BUCKET**: with your target bucket name.
     - **AWS_ACCESS_KEY_ID**: with your AWS credentials ID.
     - **AWS_SECRET_ACCESS_KEY**: with your AWS credentials KEY.
     - **AWS_REGION: with the AWS**: region of your bucket.
     - **AWS_HOSTNAME**: Default: Amazon endpoints - AWS S3 compatible provider endpoint hostname.

- **AWS_PROTOCOL**: Default: HTTPS - AWS S3 compatible provider endpoint protocol.
- **AWS_PATH_STYLE**: Default: **false** - When set to **true**, the bucket name is always left in the request URI and never moved to the host as a sub-domain.

- Optionally, specify the **--param** option with the **TENANT_NAME** parameter to set a custom name for the Admin Portal. If omitted, this defaults to 3scale

```
oc new-app --file /path/to/amp-s3.yml \
 --param WILDCARD_DOMAIN=<a-domain-that-resolves-to-your-ocp-cluster.com> \
 --param TENANT_NAME=3scale \
 --param AWS_ACCESS_KEY_ID=<your-aws-access-key-id> \
 --param AWS_SECRET_ACCESS_KEY=<your-aws-access-key-secret> \
 --param AWS_BUCKET=<your-target-bucket-name> \
 --param AWS_REGION=<your-aws-bucket-region> \
 --param FILE_UPLOAD_STORAGE=s3
```

The terminal shows the master and tenant URLs, as well as credentials for your newly created 3scale Admin Portal. This output should include the following information:

- master admin username
- master password
- master token information
- tenant username
- tenant password
- tenant token information

2. Log in to https://user-admin.3scale-project.example.com as admin/xXxXyz123.

```
...

* With parameters:
 * ADMIN_PASSWORD=xXxXyz123 # generated
 * ADMIN_USERNAME=admin
 * TENANT_NAME=user
 ...

 * MASTER_NAME=master
 * MASTER_USER=master
 * MASTER_PASSWORD=xXxXyz123 # generated
 ...

--> Success
Access your application via route 'user-admin.3scale-project.example.com'
Access your application via route 'master-admin.3scale-project.example.com'
Access your application via route 'backend-user.3scale-project.example.com'
Access your application via route 'user.3scale-project.example.com'
Access your application via route 'api-user-apicast-staging.3scale-project.example.com'
Access your application via route 'api-user-apicast-production.3scale-project.example.com'
```

```
Access your application via route 'apicast-wildcard.3scale-project.example.com'

...
```

3. Make a note of these details for future reference.
4. The 3scale deployment on OpenShift has been successful when the following command returns:

```
oc wait --for=condition=available --timeout=-1s $(oc get dc --output=name)
```

NOTE

When the 3scale deployment on OpenShift has been successful, your login credentials will work.

2.4.6. Deploying 3scale with PostgreSQL

Deploying 3scale with PostgreSQL is an optional procedure. Deploy 3scale with PostgreSQL using the following steps:

Procedure

1. Download amp-postgresql.yml.
2. Log in to OpenShift from a terminal session :

```
oc login
```

3. Select your project, or create a new project:

```
oc project <project_name>
```

OR

```
oc new-project <project_name>
```

1. Enter the oc new-app command:
   - Specify the **--file** option with the path to the *amp-postgresql.yml* file.
   - Specify the **--param** options with the following values:
   - **WILDCARD_DOMAIN**: the parameter set to the domain of your OpenShift cluster.
   - Optionally, specify the **--param** option with the **TENANT_NAME** parameter to set a custom name for the Admin Portal. If omitted, this defaults to 3scale

```
oc new-app --file /path/to/amp-postgresql.yml \
 --param WILDCARD_DOMAIN=<a-domain-that-resolves-to-your-ocp-cluster.com> \
 --param TENANT_NAME=3scale \
```

The terminal shows the master and tenant URLs, as well as the credentials for your newly created 3scale Admin Portal. This output should include the following information:

- master admin username
- master password
- master token information
- tenant username
- tenant password
- tenant token information

2. Log in to https://user-admin.3scale-project.example.com as admin/xXxXyz123.

```
...

* With parameters:
 * ADMIN_PASSWORD=xXxXyz123 # generated
 * ADMIN_USERNAME=admin
 * TENANT_NAME=user
 ...

 * MASTER_NAME=master
 * MASTER_USER=master
 * MASTER_PASSWORD=xXxXyz123 # generated
 ...

--> Success
Access your application via route 'user-admin.3scale-project.example.com'
Access your application via route 'master-admin.3scale-project.example.com'
Access your application via route 'backend-user.3scale-project.example.com'
Access your application via route 'user.3scale-project.example.com'
Access your application via route 'api-user-apicast-staging.3scale-project.example.com'
Access your application via route 'api-user-apicast-production.3scale-project.example.com'
Access your application via route 'apicast-wildcard.3scale-project.example.com'

...
```

3. Make a note of these details for future reference.

4. The 3scale deployment on OpenShift has been successful when the following command returns:

```
oc wait --for=condition=available --timeout=-1s $(oc get dc --output=name)
```

NOTE

When the 3scale deployment on OpenShift has been successful, your login and credentials will work.

2.4.7. Configuring SMTP variables (optional)

OpenShift uses email to send notifications and invite new users. If you intend to use these features, you must provide your own SMTP server and configure SMTP variables in the **system-smtp** secret.

Perform the following steps to configure the SMTP variables in the **system-smtp** secret:

Procedure

1. If you are not already logged in, log in to OpenShift:

```
oc login
```

   a. Using the **oc patch** command, specify the **secret** type where **system-smtp** is the name of the secret, followed by the **-p** option, and write the new values in JSON for the following variables:

Variable	Description
address	Allows you to specify a remote mail server as a relay
username	Specify your mail server username
password	Specify your mail server password
domain	Specify a HELO domain
port	Specify the port on which the mail server is listening for new connections
authentication	Specify the authentication type of your mail server. Allowed values: **plain** (sends the password in the clear), **login** (send password Base64 encoded), or **cram_md5** (exchange information and a cryptographic Message Digest 5 algorithm to hash important information)
openssl.verify.mode	Specify how OpenSSL checks certificates when using TLS. Allowed values: **none** or **peer**.

Example

```
oc patch secret system-smtp -p '{"stringData":{"address":"<your_address>"}}'
oc patch secret system-smtp -p '{"stringData":{"username":"<your_username>"}}'
oc patch secret system-smtp -p '{"stringData":{"password":"<your_password>"}}'
```

2. After you have set the secret variables, redeploy the **system-app** and **system-sidekiq** pods:

```
oc rollout latest dc/system-app
oc rollout latest dc/system-sidekiq
```

3. Check the status of the rollout to ensure it has finished:

```
oc rollout status dc/system-app
oc rollout status dc/system-sidekiq
```

2.5. PARAMETERS OF THE 3SCALE TEMPLATE

Template parameters configure environment variables of the 3scale (*amp.yml*) template during and after deployment.

Table 2.1. Template parameters

Name	Description	Default Value	Required?
APP_LABEL	Used for object app labels	**3scale-api-management**	yes
ZYNC_DATABASE_PASSWORD	Password for the PostgreSQL connection user. Generated randomly if not provided.	N/A	yes
ZYNC_SECRET_KEY_BASE	Secret key base for Zync. Generated randomly if not provided.	N/A	yes
ZYNC_AUTHENTICATION_TOKEN	Authentication token for Zync. Generated randomly if not provided.	N/A	yes
AMP_RELEASE	3scale release tag.	**2.11.0**	yes
ADMIN_PASSWORD	A randomly generated 3scale administrator account password.	N/A	yes
ADMIN_USERNAME	3scale administrator account username.	**admin**	yes
APICAST_ACCESS_TOKEN	Read Only Access Token that APIcast will use to download its configuration.	N/A	yes
ADMIN_ACCESS_TOKEN	Admin Access Token with all scopes and write permissions for API access.	N/A	no

Name	Description	Default Value	Required?
WILDCARD_DOMAIN	Root domain for the wildcard routes. For example, a root domain **example.com** will generate **3scale-admin.example.com**.	N/A	yes
TENANT_NAME	Tenant name under the root that Admin Portal will be available with -admin suffix.	**3scale**	yes
MYSQL_USER	Username for MySQL user that will be used for accessing the database.	**mysql**	yes
MYSQL_PASSWORD	Password for the MySQL user.	N/A	yes
MYSQL_DATABASE	Name of the MySQL database accessed.	**system**	yes
MYSQL_ROOT_PASSWORD	Password for Root user.	N/A	yes
SYSTEM_BACKEND_USERNAME	Internal 3scale API username for internal 3scale api auth.	**3scale_api_user**	yes
SYSTEM_BACKEND_PASSWORD	Internal 3scale API password for internal 3scale api auth.	N/A	yes
REDIS_IMAGE	Redis image to use	**registry.redhat.io/rhscl/redis-5-rhel7:5.0**	yes
MYSQL_IMAGE	Mysql image to use	**registry.redhat.io/rhscl/mysql-57-rhel7:5.7**	yes
MEMCACHE_SERVERS	Comma-delimited string of memcache servers, creating a ring of memcache servers to be used by **system-*** pods.	**system-memcache:11211**	yes

Name	Description	Default Value	Required?
	For example: **MEMCACHE_SERVERS="cache-1.us-east.domain.com:11211,cache-3.us-east.domain.com:11211,cache-2.us-east.domain.com:11211"**		
MEMCACHED_IMAGE	Memcached image to use	**registry.redhat.io/3scale-amp2/memcached-rhel7:3scale2.11**	yes
POSTGRESQL_IMAGE	Postgresql image to use	**registry.redhat.io/rhscl/postgresql-10-rhel7**	yes
AMP_SYSTEM_IMAGE	3scale System image to use	**registry.redhat.io/3scale-amp2/system-rhel7:3scale2.11**	yes
AMP_BACKEND_IMAGE	3scale Backend image to use	**registry.redhat.io/3scale-amp2/backend-rhel7:3scale2.11**	yes
AMP_APICAST_IMAGE	3scale APIcast image to use	**registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.11**	yes
AMP_ZYNC_IMAGE	3scale Zync image to use	**registry.redhat.io/3scale-amp2/zync-rhel7:3scale2.11**	yes
SYSTEM_BACKEND_SHARED_SECRET	Shared secret to import events from backend to system.	N/A	yes
SYSTEM_APP_SECRET_KEY_BASE	System application secret key base	N/A	yes
APICAST_MANAGEMENT_API	Scope of the APIcast Management API. Can be disabled, status or debug. At least status required for health checks.	**status**	no

Name	Description	Default Value	Required?
APICAST_OPENSSL_VERIFY	Turn on/off the OpenSSL peer verification when downloading the configuration. Can be set to true/false.	**false**	no
APICAST_RESPONSE_CODES	Enable logging response codes in APIcast.	true	no
APICAST_REGISTRY_URL	A URL which resolves to the location of APIcast policies	**http://apicast-staging:8090/policies**	yes
MASTER_USER	Master administrator account username	**master**	yes
MASTER_NAME	The subdomain value for the master Admin Portal, will be appended with the **-master** suffix	**master**	yes
MASTER_PASSWORD	A randomly generated master administrator password	N/A	yes
MASTER_ACCESS_TOKEN	A token with master level permissions for API calls	N/A	yes
IMAGESTREAM_TAG_IMPORT_INSECURE	Set to true if the server may bypass certificate verification or connect directly over HTTP during image import.	**false**	yes

2.6. DEPLOYING 3SCALE USING THE OPERATOR

This section takes you through installing and deploying the 3scale solution via the 3scale operator, using the *APIManager* custom resource.

NOTE

- Wildcard routes have been removed since 3scale 2.6.
  - This functionality is handled by Zync in the background.
- When API providers are created, updated, or deleted, routes automatically reflect those changes.

Prerequisites

- Configuring container registry authentication
- To make sure you receive automatic updates of micro releases for 3scale, you must have enabled the automatic approval functionality in the 3scale operator. *Automatic* is the default approval setting. To change this at any time based on your specific needs, use the steps for Configuring automated application of micro releases.
- Deploying 3scale using the operator first requires that you follow the steps in Installing the 3scale Operator on OpenShift
- OpenShift Container Platform 4
  - A user account with administrator privileges in the OpenShift cluster.
  - **Note:** OCP 4 supports deployment of 3scale using the operator only.
  - For more information about supported configurations, see the Red Hat 3scale API Management Supported Configurations page.

Follow these procedures to deploy 3scale using the operator:

- Deploying the APIManager custom resource
- Getting the APIManager Admin Portal and Master Admin Portal credentials
- Getting the Admin Portal URL
- Configuring automated application of micro releases
- High availability in 3scale using the operator

2.6.1. Deploying the *APIManager* custom resource

Deploying the *APIManager* custom resource will make the operator begin processing and will deploy a 3scale solution from it.

Procedure

1. Click **Operators > Installed Operators**.
   a. From the list of *Installed Operators*, click *3scale Operator*.
2. Click the *API Manager* tab.
3. Click **Create APIManager**.
4. Clear the sample content and add the following *YAML* definitions to the editor, then click **Create**.
   - Before 3scale 2.8, you could configure the automatic addition of replicas by setting the **highAvailability** field to **true**. From 3scale 2.8, the addition of replicas is controlled through the replicas field in the APIManager CR as shown in the following example.

NOTE

The *wildcardDomain* parameter can be any desired name you wish to give that resolves to an IP address, which is a valid DNS domain.

- APIManager CR with minimum requirements:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-sample
spec:
  wildcardDomain: example.com
```

- APIManager CR with replicas configured:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-sample
spec:
  system:
    appSpec:
      replicas: 1
    sidekiqSpec:
      replicas: 1
  zync:
    appSpec:
      replicas: 1
    queSpec:
      replicas: 1
  backend:
    cronSpec:
      replicas: 1
    listenerSpec:
      replicas: 1
    workerSpec:
      replicas: 1
  apicast:
    productionSpec:
      replicas: 1
    stagingSpec:
      replicas: 1
  wildcardDomain: example.com
```

2.6.2. Getting the *APIManager* Admin Portal and Master Admin Portal credentials

To log in to either the 3scale Admin Portal or Master Admin Portal after the operator-based deployment, you need the credentials for each separate portal. To get these credentials:

1. Run the following commands to get the Admin Portal credentials:

```
oc get secret system-seed -o json | jq -r .data.ADMIN_USER | base64 -d
oc get secret system-seed -o json | jq -r .data.ADMIN_PASSWORD | base64 -d
```

a. Log in as the Admin Portal administrator to verify these credentials are working.

2. Run the following commands to get the Master Admin Portal credentials:

```
oc get secret system-seed -o json | jq -r .data.MASTER_USER | base64 -d
oc get secret system-seed -o json | jq -r .data.MASTER_PASSWORD | base64 -d
```

a. Log in as the Master Admin Portal administrator to verify these credentials are working.

Additional resources

For more information about the *APIManager* fields, refer to the Reference documentation.

2.6.3. Getting the Admin Portal URL

When you deploy 3scale using the operator, a default tenant is created, with a fixed URL: **3scale-admin.${wildcardDomain}**

The 3scale Dashboard shows the new portal URL of the tenant. As an example, if the *<wildCardDomain>* is **3scale-project.example.com**, the Admin Portal URL is: **https://3scale-admin.3scale-project.example.com**.

The **wildcardDomain** is the *<wildCardDomain>* parameter you provided during installation. Open this unique URL in a browser using the this command:

```
xdg-open https://3scale-admin.3scale-project.example.com
```

Optionally, you can create new tenants on the *MASTER portal URL*: **master.${wildcardDomain}**

2.6.4. Configuring automated application of micro releases

To obtain micro release updates and have them be applied automatically, the 3scale operator's approval strategy must be set to *Automatic*. The following describes the differences between *Automatic* and *Manual* settings and outlines the steps in a procedure to change from one to the other.

Automatic and manual:

- During installation, the *Automatic* setting is the selected option by default. Installation of new updates occur as they become available. You can change this during the install or any time afterwards.
- If you select the *Manual* option during installation or at any time afterwards, you will receive updates when they are available. Next, you must approve the *Install Plan* and apply it yourself.

Procedure

1. Click **Operators > Installed Operators**.
2. Click **3scale API Management** from the list of *Installed Operators*.
3. Click the **Subscription** tab. Under the *Subscription Details* heading you will see the subheading *Approval*.
4. Click the link below *Approval*. The link is set to **Automatic** by default. A modal with the heading, *Change Update Approval Strategy* will pop up.

5. Choose the option of your preference: *Automatic (default)* or *Manual*, and then click **Save**.

Additional resources

- See *Approval Strategy* under Operator installation with OperatorHub.

2.6.5. High availability in 3scale using the operator

High availability (HA) in 3scale using the operator aims to provide uninterrupted uptime if, for example, if one or more databases were to fail.

If you want HA in your 3scale operator-based deployment, note the following:

- Deploy and configure 3scale critical databases externally, specifically system database, system redis, and backend redis. Make sure you deploy and configure those databases in a way they are highly available.
- Specify the connection endpoints to those databases for 3scale by pre-creating their corresponding Kubernetes Secrets.
  - See External databases installation for more information.
  - See Enabling Pod Disruption Budgets for more information about non-database deployment configurations.
- Set the **.spec.highAvailability.enabled** attribute to **true** when deploying the APIManager CR to enable external database mode for the critical databases: system database, system redis, and backend redis.

Additionally, if you want the zync database to be highly available to avoid zync potentially losing queue jobs data on restart, note the following:

- Deploy and configure the zync database externally. Make sure you deploy and configure the database in a way that it is highly available.
- Specify the connection endpoint to the zync database for 3scale by pre-creating its corresponding Kubernetes Secrets.
  - See Zync database secret for more information.
- Deploy 3scale setting the **spec.highAvailability.externalZyncDatabaseEnabled** attribute to true to specify zync database as an external database.

2.7. DEPLOYMENT CONFIGURATION OPTIONS FOR 3SCALE ON OPENSHIFT USING THE OPERATOR

This section provides information about the deployment configuration options for Red Hat 3scale API Management on OpenShift using the operator.

Prerequisites

- Configuring container registry authentication
- Deploying 3scale using the operator first requires that you follow the steps in Installing the 3scale Operator on OpenShift

- OpenShift Container Platform 4.x
  - A user account with administrator privileges in the OpenShift cluster.

2.7.1. Configuring proxy parameters for embedded APIcast

As a 3scale administrator, you can configure proxy parameters for embedded APIcast staging and production. This section provides reference information for specifying proxy parameters in an **APIManager** custom resource. In other words, you are using the 3scale operator (an **APIManager** custom resource) to deploy 3scale on OpenShift.

You can specify these parameters when you deploy an **APIManager** CR for the first time or you can update a deployed **APIManager** CR and the operator will reconcile the update. See Deploying the APIManager custom resource.

There are four proxy-related configuration parameters for embedded APIcast:

- **allProxy**
- **httpProxy**
- **httpsProxy**
- **noProxy**

allProxy

The **allProxy** parameter specifies an HTTP or HTTPS proxy to be used for connecting to services when a request does not specify a protocol-specific proxy.

After you set up a proxy, configure APIcast by setting the **allProxy** parameter to the address of the proxy. Authentication is not supported for the proxy. In other words, APIcast does not send authenticated requests to the proxy.

The value of the **allProxy** parameter is a string, there is no default, and the parameter is not required. Use this format to set the **spec.apicast.productionSpec.allProxy** parameter or the **spec.apicast.stagingSpec.allProxy** parameter:

<scheme>://<host>:<port>

For example:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      allProxy: http://forward-proxy:80
    stagingSpec:
      allProxy: http://forward-proxy:81
```

httpProxy

The **httpProxy** parameter specifies an HTTP proxy to be used for connecting to HTTP services.

After you set up a proxy, configure APIcast by setting the **httpProxy** parameter to the address of the proxy. Authentication is not supported for the proxy. In other words, APIcast does not send authenticated requests to the proxy.

The value of the **httpProxy** parameter is a string, there is no default, and the parameter is not required. Use this format to set the **spec.apicast.productionSpec.httpProxy** parameter or the **spec.apicast.stagingSpec.httpProxy** parameter:

http://<host>:<port>

For example:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      httpProxy: http://forward-proxy:80
    stagingSpec:
      httpProxy: http://forward-proxy:81
```

httpsProxy

The **httpsProxy** parameter specifies an HTTPS proxy to be used for connecting to services.

After you set up a proxy, configure APIcast by setting the **httpsProxy** parameter to the address of the proxy. Authentication is not supported for the proxy. In other words, APIcast does not send authenticated requests to the proxy.

The value of the **httpsProxy** parameter is a string, there is no default, and the parameter is not required. Use this format to set the **spec.apicast.productionSpec.httpsProxy** parameter or the **spec.apicast.stagingSpec.httpsProxy** parameter:

https://<host>:<port>

For example:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      httpsProxy: https://forward-proxy:80
    stagingSpec:
      httpsProxy: https://forward-proxy:81
```

noProxy

The **noProxy** parameter specifies a comma-separated list of hostnames and domain names. When a request contains one of these names, APIcast does not proxy the request.

If you need to stop access to the proxy, for example during maintenance operations, set the **noProxy** parameter to an asterisk (*). This matches all hosts specified in all requests and effectively disables any proxies.

The value of the **noProxy** parameter is a string, there is no default, and the parameter is not required. Specify a comma-separated string to set the **spec.apicast.productionSpec.noProxy** parameter or the **spec.apicast.stagingSpec.noProxy** parameter. For example:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      noProxy: theStore,company.com,big.red.com
    stagingSpec:
      noProxy: foo,bar.com,.extra.dot.com
```

Additional resources

- Custom resource definition for APIcast staging configuration in an **APIManager** custom resource
- Custom resource definition for APIcast production configuration in an **APIManager** custom resource

2.7.2. Injecting custom environments with the 3scale operator

In a 3scale installation that uses embedded APIcast, you can use the 3scale operator to inject custom environments. Embedded APIcast is also referred to as managed or hosted APIcast. A custom environment defines behavior that APIcast applies to all upstream APIs that the gateway serves. To create a custom environment, define a global configuration in Lua code.

You can inject a custom environment before or after 3scale installation. After injecting a custom environment and after 3scale installation, you can remove a custom environment. The 3scale operator reconciles the changes.

Prerequisites

- The 3scale operator is installed.

Procedure

1. Write Lua code that defines the custom environment that you want to inject. For example, the following **env1.lua** file shows a custom logging policy that the 3scale operator loads for all services.

```
local cjson = require('cjson')
local PolicyChain = require('apicast.policy_chain')
local policy_chain = context.policy_chain

local logging_policy_config = cjson.decode([[
{
  "enable_access_logs": false,
```

```
  "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}"
}
]])

policy_chain:insert( PolicyChain.load_policy('logging', 'builtin', logging_policy_config), 1)

return {
  policy_chain = policy_chain,
  port = { metrics = 9421 },
}
```

2. Create a secret from the Lua file that defines the custom environment. For example:

```
oc create secret generic custom-env-1 --from-file=./env1.lua
```

A secret can contain multiple custom environments. Specify the `**–from-file** option for each file that defines a custom environment. The operator loads each custom environment.

3. Define an **APIManager** custom resource that references the secret you just created. The following example shows only content relative to referencing the secret that defines the custom environment.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-apicast-custom-environment
spec:
  wildcardDomain: <desired-domain>
  apicast:
    productionSpec:
      customEnvironments:
        - secretRef:
            name: custom-env-1
    stagingSpec:
      customEnvironments:
        - secretRef:
            name: custom-env-1
```

An **APIManager** custom resource can reference multiple secrets that define custom environments. The operator loads each custom environment.

4. Create the **APIManager** custom resource that adds the custom environment. For example:

```
oc apply -f apimanager.yaml
```

Next steps

You cannot update the content of a secret that defines a custom environment. If you need to update the custom environment you can do either of the following:

- The recommended option is to create a secret with a different name and update the **APIManager** custom resource field, **customEnvironments[].secretRef.name**. The operator triggers a rolling update and loads the updated custom environment.
- Alternatively, you can update the existing secret, redeploy APIcast by setting **spec.apicast.productionSpec.replicas** or **spec.apicast.stagingSpec.replicas** to 0, and then

redploy APIcast again by setting **spec.apicast.productionSpec.replicas** or **spec.apicast.stagingSpec.replicas** back to its previous value.

2.7.3. Injecting custom policies with the 3scale operator

In a 3scale installation that uses embedded APIcast, you can use the 3scale operator to inject custom policies. Embedded APIcast is also referred to as managed or hosted APIcast. Injecting a custom policy adds the policy code to APIcast. You can then use either of the following to add the custom policy to an API product's policy chain:

- 3scale API
- **Product** custom resource

To use the 3scale Admin Portal to add the custom policy to a product's policy chain, you must also register the custom policy's schema with a **CustomPolicyDefinition** custom resource. Custom policy registration is a requirement only when you want to use the Admin Portal to configure a product's policy chain.

You can inject a custom policy as part of or after 3scale installation. After injecting a custom policy and after 3scale installation, you can remove a custom policy by removing its specification from the **APIManager** CR. The 3scale operator reconciles the changes.

Prerequisites

- You are installing or you previously installed the 3scale operator.
- You have defined a custom policy as described in Write your own policy. That is, you have already created, for example, the **my-policy.lua**, **apicast-policy.json**, and **init.lua** files that define a custom policy,

Procedure

1. Create a secret from the files that define one custom policy. For example:

```
oc create secret generic my-first-custom-policy-secret \
 --from-file=./apicast-policy.json \
 --from-file=./init.lua \
 --from-file=./my-first-custom-policy.lua
```

   If you have more than one custom policy, create a secret for each custom policy. A secret can contain only one custom policy.

2. Define an **APIManager** custom resource that references each secret that contains a custom policy. You can specify the same secret for APIcast staging and APIcast production. The following example shows only content relative to referencing secrets that contain custom policies.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-apicast-custom-policy
spec:
  apicast:
    stagingSpec:
```

```
    customPolicies:
      - name: my-first-custom-policy
        version: "0.1"
        secretRef:
          name: my-first-custom-policy-secret
      - name: my-second-custom-policy
        version: "0.1"
        secretRef:
          name: my-second-custom-policy-secret
  productionSpec:
    customPolicies:
      - name: my-first-custom-policy
        version: "0.1"
        secretRef:
          name: my-first-custom-policy-secret
      - name: my-second-custom-policy
        version: "0.1"
        secretRef:
          name: my-second-custom-policy-secret
```

An **APIManager** custom resource can reference multiple secrets that define different custom policies. The operator loads each custom policy.

3. Create the **APIManager** custom resource that references the secrets that contain the custom policies. For example:

```
oc apply -f apimanager.yaml
```

Next steps

You cannot update the content of a secret that defines a custom policy. If you need to update the custom policy you can do either of the following:

- The recommended option is to create a secret with a different name and update the **APIManager** custom resource **customPolicies** section to refer to the new secret. The operator triggers a rolling update and loads the updated custom policy.
- Alternatively, you can update the existing secret, redeploy APIcast by setting **spec.apicast.productionSpec.replicas** or **spec.apicast.stagingSpec.replicas** to 0, and then redploy APIcast again by setting **spec.apicast.productionSpec.replicas** or **spec.apicast.stagingSpec.replicas** back to its previous value.

2.7.4. Configuring OpenTracing with the 3scale operator

In a 3scale installation that uses embedded APIcast, you can use the 3scale operator to configure OpenTracing. You can configure OpenTracing in the staging or production environments or both environments. By enabling OpenTracing, you get more insight and better observability on the APIcast instance.

Prerequisites

- The 3scale operator is installed or you are in the process of installing it.
- Prerequisites listed in Configuring APIcast to use OpenTracing .
- Jaeger is installed.

Procedure

1. Define a secret that contains your OpenTracing configuration details in **stringData.config**. This is the only valid value for the attribute that contains your OpenTracing configuration details. Any other specification prevents APIcast from receiving your OpenTracing configuration details. The folowing example shows a valid secret definition:

```
apiVersion: v1
kind: Secret
metadata:
  name: myjaeger
stringData:
  config: |-
      {
      "service_name": "apicast",
      "disabled": false,
      "sampler": {
        "type": "const",
        "param": 1
      },
      "reporter": {
        "queueSize": 100,
        "bufferFlushInterval": 10,
        "logSpans": false,
        "localAgentHostPort": "jaeger-all-in-one-inmemory-agent:6831"
      },
      "headers": {
        "jaegerDebugHeader": "debug-id",
        "jaegerBaggageHeader": "baggage",
        "TraceContextHeaderName": "uber-trace-id",
        "traceBaggageHeaderPrefix": "testctx-"
      },
      "baggage_restrictions": {
          "denyBaggageOnInitializationFailure": false,
          "hostPort": "127.0.0.1:5778",
          "refreshInterval": 60
      }
      }
type: Opaque
```

2. Create the secret. For example, if you saved the previous secret definition in the **myjaeger.yaml** file, you would run the following command:

```
oc create secret generic myjaeger --from-file myjaeger.yaml
```

3. Define an **APIManager** custom resource that specifies **OpenTracing** attributes. In the CR definition, set the **openTracing.tracingConfigSecretRef.name** attribute to the name of the secret that contains your OpenTracing configuration details. The following example shows only content relative to configuring OpenTracing.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager1
spec:
```

```
apicast:
  stagingSpec:
    ...
    openTracing:
      enabled: true
      tracingLibrary: jaeger
      tracingConfigSecretRef:
        name: myjaeger
  productionSpec:
    ...
      openTracing:
        enabled: true
        tracingLibrary: jaeger
        tracingConfigSecretRef:
          name: myjaeger
```

4. Create the **APIManager** custom resource that configures OpenTracing. For example, if you saved the **APIManager** custom resource in the **apimanager1.yaml** file, you would run the following command:

```
oc apply -f apimanager1.yaml
```

Next steps

Depending on how OpenTracing is installed, you should see the traces in the Jaeger service user interface.

Additional resource

- APIManager custom resource definition

2.7.5. Enabling TLS at the pod level with the 3scale operator

3scale deploys two APIcast instances, one for production and the other for staging. TLS can be enabled for only production or only staging, or for both instances.

Prerequisites

- A valid certificate for enabling TLS.

Procedure

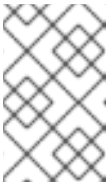
1. Create a secret from your valid certificate, for example:

```
oc create secret tls mycertsecret --cert=server.crt --key=server.key
```

The configuration exposes secret references in the **APIManager** CRD. You create the secret and then reference the name of the secret in the **APIManager** custom resource as follows:

- Production: The **APIManager** CR exposes the certificate in the **.spec.apicast.productionSpec.httpsCertificateSecretRef** field.
- Staging: The **APIManager** CR exposes the certificate in the **.spec.apicast.stagingSpec.httpsCertificateSecretRef** field.
Optionally, you can configure the following:

- **httpsPort** indicates which port APIcast should start listening on for HTTPS connections. If this clashes with the HTTP port APIcast uses this port for HTTPS only.
- **httpsVerifyDepth** defines the maximum length of the client certificate chain.



NOTE

Provide a valid certificate and reference from the **APImanager** CR. If the configuration can access **httpsPort** but not **httpsCertificateSecretRef**, APIcast uses an embedded self-signed certificate. This is not recommended.

2. Click **Operators > Installed Operators**.
3. From the list of **Installed Operators**, click **3scale Operator**.
4. Click the **API Manager** tab.
5. Click **Create APIManager**.
6. Add the following YAML definitions to the editor.
   a. If enabling for *production*, configure the following YAML defintions:

```
spec:
  apicast:
    productionSpec:
      httpsPort: 8443
      httpsVerifyDepth: 1
      httpsCertificateSecretRef:
        name: mycertsecret
```

   b. If enabling for *staging*, configure the following YAML defintions:

```
spec:
  apicast:
    stagingSpec:
      httpsPort: 8443
      httpsVerifyDepth: 1
      httpsCertificateSecretRef:
        name: mycertsecret
```

7. Click **Create**.

2.7.6. Proof of concept for evaluation deployment

The following sections describe the configuration options applicable to the proof of concept for an evaluation deployment of 3scale. This deployment uses internal databases as default.

IMPORTANT

The configuration for external databases is the standard deployment option for production environments.

2.7.6.1. Default deployment configuration

- Containers will have Kubernetes resource limits and requests .
  - This ensures a minimum performance level.
  - It limits resources to allow external services and allocation of solutions.
- Deployment of internal databases.
- File storage will be based on Persistence Volumes (PV).
  - One will require read, write, execute (RWX) access mode.
  - OpenShift configured to provide them upon request.
- Deploy MySQL as the internal relational database.

The default configuration option is suitable for proof of concept (PoC) or evaluation by a customer.

One, many, or all of the default configuration options can be overriden with specific field values in the *APIManager* custom resource. The 3scale operator allows all available combinations whereas templates allow fixed deployment profiles. For example, the 3scale operator allows deployment of 3scale in evaluation mode and external databases mode. Templates do not allow this specific deployment configuration. Templates are only available for the most common configuration options.

2.7.6.2. Evaluation installation

For and evaluation installtion, containers will not have kubernetes resource limits and requests specified. For example:

- Small memory footprint
- Fast startup
- Runnable on laptop
- Suitable for presale/sales demos

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  resourceRequirementsEnabled: false
```

Additional resources

- See *APIManager* custom resource for more information.

2.7.7. External databases installation

An external databases installation is suitable for production use where high availability (HA) is a requirement or where you plan to reuse your own databases.

IMPORTANT

When enabling the 3scale external databases installation mode, all of the following databases are externalized:

- **backend-redis**
- **system-redis**
- **system-database** (**mysql**, **postgresql**, or **oracle**)

3scale 2.8 and above has been tested is supported with the following database versions:

Database	Version
Redis	5.0
MySQL	5.7
PostgreSQL	10.6

Before creating *APIManager custom resource* to deploy 3scale, you must provide the following connection settings for the external databases using OpenShift secrets.

2.7.7.1. Backend Redis secret

Deploy two external Redis instances and fill in the connection settings as shown in the following example:

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-redis
stringData:
  REDIS_STORAGE_URL: "redis://backend-redis-storage"
  REDIS_STORAGE_SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-
1.example.com:26379, redis://sentinel-2.example.com:26379"
  REDIS_STORAGE_SENTINEL_ROLE: "master"
  REDIS_QUEUES_URL: "redis://backend-redis-queues"
  REDIS_QUEUES_SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-
1.example.com:26379, redis://sentinel-2.example.com:26379"
  REDIS_QUEUES_SENTINEL_ROLE: "master"
type: Opaque
```

The *Secret* name must be **backend-redis**.

2.7.7.2. System Redis secret

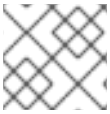
Deploy two external Redis instances and fill in the connection settings as shown in the following example:

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: system-redis
stringData:
  URL: "redis://system-redis"
  SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-1.example.com:26379,
redis://sentinel-2.example.com:26379"
  SENTINEL_ROLE: "master"
  NAMESPACE: ""
  MESSAGE_BUS_URL: "redis://system-redis-messagebus"
  MESSAGE_BUS_SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-
1.example.com:26379, redis://sentinel-2.example.com:26379"
  MESSAGE_BUS_SENTINEL_ROLE: "master"
  MESSAGE_BUS_NAMESPACE: ""
type: Opaque
```

The *Secret* name must be **system-redis**.

2.7.7.3. System database secret



NOTE

The *Secret* name must be **system-database**.

When you are deploying 3scale, you have three alternatives for your system database. Configure different attributes and values for each alternative's related secret.

- MySQL
- PostgreSQL
- Oracle Database

To deploy a MySQL, PostgreSQL, or an Oracle Database system database secret, fill in the connection settings as shown in the following examples:

MySQL system database secret

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "mysql2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
type: Opaque
```

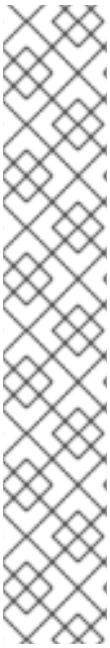
PostgreSQL system database secret

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
```

```
stringData:
  URL: "postgresql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
type: Opaque
```

Oracle system database secret

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "oracle-enhanced://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
  ORACLE_SYSTEM_PASSWORD: "{SYSTEM_PASSWORD}"
type: Opaque
```



NOTE

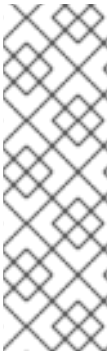
- The Oracle **system** user executes commands with system privileges. Some are detailed in this GitHub repository. The latest can be executed in the Oracle Database initializer when the tables are initialized in the database. There may be other commands executed not listed in these links.
- The **system** user is also required for upgrades when there are any schema migrations to run, so other commands not included in the previous links may be executed.
- Disclaimer: *Links contained in this note to external website(s) are provided for convenience only. Red Hat has not reviewed the links and is not responsible for the content or its availability. The inclusion of any link to an external website does not imply endorsement by Red Hat of the website or their entities, products or services. You agree that Red Hat is not responsible or liable for any loss or expenses that may result due to your use of (or reliance on) the external site or content.*

2.7.7.4. Zync database secret

In a zync database setup, when HighAvailability is enabled, and if the **externalZyncDatabaseEnabled** field is also enabled, the user has to pre-create a secret named **zync**. Then set **zync** with the **DATABASE_URL** and **DATABASE_PASSWORD** fields with the values pointing to your externally database. The external database must be in high-availability mode. See the following example:

```
apiVersion: v1
kind: Secret
metadata:
  name: zync
stringData:
  DATABASE_URL: postgresql://<zync-db-user>:<zync-db-password>@<zync-db-host>:<zync-db-port>/zync_production
  ZYNC_DATABASE_PASSWORD: <zync-db-password>
type: Opaque
```

2.7.7.5. APIManager custom resources to deploy 3scale



NOTE

- When you enable **highAvailability**, you must pre-create the **backend-redis**, **system-redis**, and **system-database** secrets.
- When you enable **highAvailability** and the **externalZyncDatabaseEnabled** fields together, you must pre-create the zync database secret.
  - Choose only one type of database to externalize in the case of **system-database**.

Configuration of the *APIManager* custom resource will depend on whether or not your choice of database is external to your 3scale deployment.

If your backend Redis, system Redis, and system database will be external to 3scale, the *APIManager* custom resource must have **highAvailability** set to **true**. See the following example:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  highAvailability:
    enabled: true
```

If your zync database will be external, the *APIManager* custom resource must have **highAvailability** set to **true** and **externalZyncDatabaseEnabled** must also be set to **true**. See the following example:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  highAvailability:
    enabled: true
    externalZyncDatabaseEnabled: true
```

Additional resources

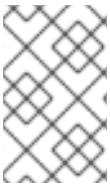
- Backend redis secret
- System database secret
- APIManager HighAvailabilitySpec
- Zync secret

2.7.8. Amazon Simple Storage Service 3scale *Filestorage* installation

The following examples show 3scale *FileStorage* using Amazon Simple Storage Service (Amazon S3) instead of persistent volume claim (PVC).

Before creating *APIManager* custom resource to deploy 3scale, connection settings for the S3 service needs to be provided using an openshift secret.

2.7.8.1. Amazon S3 secret



NOTE

AN AWS S3 compatible provider can be configured in the S3 secret with **AWS_HOSTNAME**, **AWS_PATH_STYLE**, and **AWS_PROTOCOL** optional keys. See the S3 secret reference for more details.

In the following example, *Secret* name can be anything, as it will be referenced in the *APIManager* custom resource.

```
kind: Secret
metadata:
  creationTimestamp: null
  name: aws-auth
stringData:
  AWS_ACCESS_KEY_ID: 123456
  AWS_SECRET_ACCESS_KEY: 98765544
  AWS_BUCKET: mybucket.example.com
  AWS_REGION: eu-west-1
type: Opaque
```

NOTE

Amazon S3 region and Amazon S3 bucket settings are provided directly in the *APIManager* custom resource. The Amazon S3 secret name is provided directly in the *APIManager* custom resource.

Lastly, create the *APIManager* custom resource to deploy 3scale.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  system:
    fileStorage:
      simpleStorageService:
        configurationSecretRef:
          name: aws-auth
```

Check *APIManager SystemS3Spec* for reference.

2.7.9. PostgreSQL installation

A MySQL internal relational database is the default deployment. This deployment configuration can be overriden to use PostgreSQL instead.

```
apiVersion: apps.3scale.net/v1alpha1
```

```
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  system:
    database:
      postgresql: {}
```

Additional resources

- See *APIManager DatabaseSpec* for more information.

2.7.10. Customizing compute resource requirements at component level

Customize Kubernetes Compute Resource Requirements in your 3scale solution through the *APIManager* custom resource attributes. Do this to customize compute resource requirements, which is CPU and memory, assigned to a specific *APIManager* component.

The following example outlines how to customize compute resource requirements for the system-master's **system-provider** container, for the **backend-listener** and for the **zync-database**:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      resources:
        requests:
          memory: "150Mi"
          cpu: "300m"
        limits:
          memory: "500Mi"
          cpu: "1000m"
  system:
    appSpec:
      providerContainerResources:
        requests:
          memory: "111Mi"
          cpu: "222m"
        limits:
          memory: "333Mi"
          cpu: "444m"
  zync:
    databaseResources:
      requests:
        memory: "111Mi"
        cpu: "222m"
      limits:
        memory: "333Mi"
        cpu: "444m"
```

Additional resources

See APIManager CRD reference for more information about how to specify component-level custom resource requirements.

2.7.10.1. Default APIManager components compute resources

When you configure the *APIManager* **spec.resourceRequirementsEnabled** attribute as **true**, the default compute resources are set for the *APIManager* components.

The specific compute resources default values that are set for the *APIManager* components are shown in the following table.

2.7.10.1.1. CPU and memory units

The following list explains the units you will find mentioned in the compute resources default values table. For more information on CPU and memory units, see Managing Resources for Containers.

Resource units explanation

- m - milliCPU or millicore
- Mi - mebibytes
- Gi - gibibyte
- G - gigabyte

Table 2.2. Compute resources default values

Component	CPU requests	CPU limits	Memory requests	Memory limits
system-app's system-master	50m	1000m	600Mi	800Mi
system-app's system-provider	50m	1000m	600Mi	800Mi
system-app's system-developer	50m	1000m	600Mi	800Mi
system-sidekiq	100m	1000m	500Mi	2Gi
system-sphinx	80m	1000m	250Mi	512Mi
system-redis	150m	500m	256Mi	32Gi
system-mysql	250m	No limit	512Mi	2Gi
system-postgresql	250m	No limit	512Mi	2Gi
backend-listener	500m	1000m	550Mi	700Mi

Component	CPU requests	CPU limits	Memory requests	Memory limits
backend-worker	150m	1000m	50Mi	300Mi
backend-cron	50m	150m	40Mi	80Mi
backend-redis	1000m	2000m	1024Mi	32Gi
apicast-production	500m	1000m	64Mi	128Mi
apicast-staging	50m	100m	64Mi	128Mi
zync	150m	1	250M	512Mi
zync-que	250m	1	250M	512Mi
zync-database	50m	250m	250M	2G

2.7.11. Customizing node affinity and tolerations at component level

Customize Kubernetes Affinity and Tolerations in your Red Hat 3scale API Management solution through the *APIManager* custom resource attributes to customize where and how the different 3scale components of an installation are scheduled onto Kubernetes Nodes.

The following example sets a custom node affinity for the backend. It also sets listener and custom tolerations for the **system-memcached**:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: "kubernetes.io/hostname"
                operator: In
                values:
                - ip-10-96-1-105
              - key: "beta.kubernetes.io/arch"
                operator: In
                values:
                - amd64
  system:
    memcachedTolerations:
    - key: key1
      value: value1
```

```
      operator: Equal
      effect: NoSchedule
    - key: key2
      value: value2
      operator: Equal
      effect: NoSchedule
```

Additional resources

See APIManager CDR reference for a full list of attributes related to affinity and tolerations.

2.7.12. Reconciliation

Once 3scale has been installed, the 3scale operator enables updating a given set of parameters from the custom resource to modify system configuration options. Modifications are made by *hot swapping*, that is, without stopping or shutting down the system.

Not all the parameters of the *APIManager* custom resource definitions (CRDs) are reconcilable.

The following is a list of reconcilable parameters:

- Section 2.7.12.1, “Resources”
- Section 2.7.12.2, “Backend replicas”
- Section 2.7.12.3, “APIcast replicas”
- Section 2.7.12.4, “System replicas”

2.7.12.1. Resources

Resource limits and requests for all 3scale components.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  ResourceRequirementsEnabled: true/false
```

2.7.12.2. Backend replicas

Backend components pod count.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      replicas: X
    workerSpec:
```

```
    replicas: Y
  cronSpec:
    replicas: Z
```

2.7.12.3. APIcast replicas

APIcast staging and production components pod count.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      replicas: X
    stagingSpec:
      replicas: Z
```

2.7.12.4. System replicas

System app and system *sidekiq* components pod count

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  system:
    appSpec:
      replicas: X
    sidekiqSpec:
      replicas: Z
```

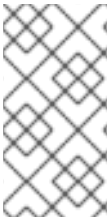
2.7.12.5. Zync replicas

Zync app and que components pod count

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  zync:
    appSpec:
      replicas: X
    queSpec:
      replicas: Z
```

2.8. INSTALLING 3SCALE WITH THE OPERATOR USING ORACLE AS THE SYSTEM DATABASE

As a Red Hat 3scale API Management administrator, you can install the 3scale with the operator using the Oracle Database. By default, 3scale 2.11 has a component called **system** that stores configuration data in a MySQL database. You can override the default database and store your information in an external Oracle Database. Follow the steps below to build a custom system container image with your own Oracle Database client binaries and deploy 3scale to OpenShift.



NOTE

- The Oracle Database is not supported with OpenShift Container Platform (OCP) versions 4.2 and 4.3 when you are performing an operator-only installation of 3scale. For more information, refer to the Red Hat 3scale API Management Supported Configurations page.

Prerequisites

- A container registry to push container images, accessible by the OCP cluster where 3scale installed.
- An installation of the 3scale operator.
  - Do not install the *APIManager* custom resource, as it will be created in the following procedure.
- A Registry service account for 3scale .

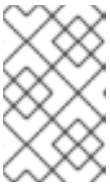
To install 3scale with the operator using Oracle as the system database, use the following steps:

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. Follow the prerequisites in Setting up your 3scale system image with an Oracle Database .



NOTE

If the client packages versions downloaded and stored locally do not match with the ones 3scale expects, 3scale will automatically download and use the appropriate ones in the following steps.

3. 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.
4. Login to your **registry.redhat.io** account using the credentials you created in Creating a Registry Service Account.

```
docker login registry.redhat.io
```

5. Build the custom system Oracle-based image. The image tag must be a fixed image tag as in the following example:

```
docker build . --tag myregistry.example.com/system-oracle:2.11.0-1
```

6. Push the system Oracle-based image to a container registry accessible by the OCP cluster. This container registry is where your 3scale solution is going to be installed:

```
docker push myregistry.example.com/system-oracle:2.11.0-1
```

7. Set up the Oracle Database URL connection string and Oracle Database system password by creating the **system-database** secret with the corresponding fields. See, External databases installation for the Oracle Database.

8. Install your 3scale solution by creating an *APIManager* custom resource. Follow the instructions in Deploying 3scale using the operator.

   - The *APIManager* custom resource must specify the **.spec.system.image** field set to the system's Oracle-based image you previous built:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
 imagePullSecrets:
  - name: threescale-registry-auth
  - name: custom-registry-auth
 system:
  image: "myregistry.example.com/system-oracle:2.11.0-1"
 highAvailability:
  enabled: true
```

2.9. TROUBLESHOOTING COMMON 3SCALE INSTALLATION ISSUES

This section contains a list of common installation issues and provides guidance for their resolution.

- Previous deployment leaving dirty persistent volume claims
- Wrong or missing credentials of the authenticated image registry
- Incorrectly pulling from the Docker registry
- Permission issues for MySQL when persistent volumes are mounted locally
- Unable to upload logo or images
- Test calls not working on OpenShift
- APIcast on a different project from 3scale failing to deploy

2.9.1. Previous deployment leaving dirty persistent volume claims

Problem

A previous deployment attempt leaves a dirty Persistent Volume Claim (PVC) causing the MySQL container to fail to start.

Cause

Deleting a project in OpenShift does not clean the PVCs associated with it.

Solution

Procedure

1. Find the PVC containing the erroneous MySQL data with the **oc get pvc** command:

```
# oc get pvc
NAME                    STATUS   VOLUME    CAPACITY   ACCESSMODES   AGE
backend-redis-storage   Bound    vol003    100Gi      RWO,RWX       4d
mysql-storage           Bound    vol006    100Gi      RWO,RWX       4d
system-redis-storage    Bound    vol008    100Gi      RWO,RWX       4d
system-storage          Bound    vol004    100Gi      RWO,RWX       4d
```

2. Stop the deployment of the system-mysql pod by clicking **cancel deployment** in the OpenShift UI.

3. Delete everything under the MySQL path to clean the volume.

4. Start a new **system-mysql** deployment.

2.9.2. Wrong or missing credentials of the authenticated image registry

Problem

Pods are not starting. ImageStreams show the following error:

```
! error: Import failed (InternalError): ...unauthorized: Please login to the Red Hat Registry
```

Cause

While installing 3scale on OpenShift 4.x, OpenShift fails to start pods because ImageStreams cannot pull the images they reference. This happens because the pods cannot authenticate against the registries they point to.

Solution

Procedure

1. Type the following command to verify the configuration of your container registry authentication:

```
$ oc get secret
```

   - If your secret exists, you will see the following output in the terminal:

```
threescale-registry-auth        kubernetes.io/dockerconfigjson        1        4m9s
```

   - However, if you do not see the output, you must do the following:

2. Use the credentials you previously set up while Creating a registry service account to create your secret.

3. Use the steps in Configuring registry authentication in OpenShift, replacing **<your-registry-service-account-username>** and **<your-registry-service-account-password>** in the **oc create secret** command provided.

4. Generate the **threescale-registry-auth** secret in the same namespace as the *APIManager* resource. You must run the following inside the **<project-name>**:

```
oc project <project-name>
oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username="<your-registry-service-account-username>" \
  --docker-password="<your-registry-service-account-password>"
  --docker-email="<email-address>"
```

5. Delete and recreate the *APIManager* resource:

```
$ oc delete -f apimanager.yaml
apimanager.apps.3scale.net "example-apimanager" deleted

$ oc create -f apimanager.yaml
apimanager.apps.3scale.net/example-apimanager created
```

Verification

1. Type the following command to confirm that deployments have a status of **Starting** or **Ready**. The pods then begin to spawn:

```
$ oc describe apimanager
(...)
Status:
  Deployments:
    Ready:
      apicast-staging
      system-memcache
      system-mysql
      system-redis
      zync
      zync-database
      zync-que
    Starting:
      apicast-production
      backend-cron
      backend-worker
      system-sidekiq
      system-sphinx
    Stopped:
      backend-listener
      backend-redis
      system-app
```

2. Type the following command to see the status of each pod:

```
$ oc get pods
NAME                          READY   STATUS          RESTARTS   AGE
3scale-operator-66cc6d857b-sxhgm   1/1     Running         0        17h
```

```
apicast-production-1-deploy      1/1     Running          0         17m
apicast-production-1-pxkqm       0/1     Pending          0         17m
apicast-staging-1-dbwcw          1/1     Running          0         17m
apicast-staging-1-deploy         0/1     Completed        0         17m
backend-cron-1-deploy            1/1     Running          0         17m
```

2.9.3. Incorrectly pulling from the Docker registry

Problem

The following error occurs during installation:

```
svc/system-redis - 1EX.AMP.LE.IP:6379
  dc/system-redis deploys docker.io/rhscl/redis-32-rhel7:3.2-5.3
    deployment #1 failed 13 minutes ago: config change
```

Cause

OpenShift searches for and pulls container images by issuing the **docker** command. This command refers to the **docker.io** Docker registry instead of the **registry.redhat.io** Red Hat Ecosystem Catalog.

This occurs when the system contains an unexpected version of the Docker containerized environment.

Solution

Procedure

Use the appropriate version of the Docker containerized environment.

2.9.4. Permission issues for MySQL when persistent volumes are mounted locally

Problem

The system-msql pod crashes and does not deploy causing other systems dependant on it to fail deployment. The pod log displays the following error:

```
[ERROR] Cannot start server : on unix socket: Permission denied
[ERROR] Do you already have another mysqld server running on socket: /var/lib/mysql/mysql.sock ?
[ERROR] Aborting
```

Cause

The MySQL process is started with inappropriate user permissions.

Solution

Procedure

1. The directories used for the persistent volumes MUST have the write permissions for the root group. Having read-write permissions for the root user is not enough as the MySQL service runs as a different user in the root group. Execute the following command as the root user:

```
chmod -R g+w /path/for/pvs
```

2. Execute the following command to prevent SElinux from blocking access:

```
chcon -Rt svirt_sandbox_file_t /path/for/pvs
```

2.9.5. Unable to upload logo or images

Problem

Unable to upload a logo - **system-app** logs display the following error:

```
Errno::EACCES (Permission denied @ dir_s_mkdir - /opt/system/public//system/provider-name/2
```

Cause

Persistent volumes are not writable by OpenShift.

Solution

Procedure

Ensure your persistent volume is writable by OpenShift. It should be owned by root group and be group writable.

2.9.6. Test calls not working on OpenShift

Problem

Test calls do not work after creation of a new service and routes on OpenShift. Direct calls via curl also fail, stating: **service not available**.

Cause

3scale requires HTTPS routes by default, and OpenShift routes are not secured.

Solution

Procedure

Ensure the **secure route** checkbox is clicked in your OpenShift router settings.

2.9.7. APIcast on a different project from 3scale failing to deploy

Problem

APIcast deploy fails (pod does not turn blue). You see the following error in the logs:

```
update acceptor rejected apicast-3: pods for deployment "apicast-3" took longer than 600 seconds to become ready
```

You see the following error in the pod:

```
Error synching pod, skipping: failed to "StartContainer" for "apicast" with RunContainerError: "GenerateRunContainerOptions: secrets \"apicast-configuration-url-secret\" not found"
```

Cause

The secret was not properly set up.

Solution

Procedure

When creating a secret with APIcast v3, specify **apicast-configuration-url-secret**:

```
oc create secret generic apicast-configuration-url-secret --from-
literal=password=https://<ACCESS_TOKEN>@<TENANT_NAME>-admin.<WILDCARD_DOMAIN>
```

2.10. ADDITIONAL RESOURCES

- For more information about **HighAvailabilitySpec**, see the APIManager CRD reference documentation.
- For more information about secrets and fields for **system-database**, see the APIManager CRD reference documentation.

CHAPTER 3. INSTALLING APICAST

APIcast is an NGINX based API gateway used to integrate your internal and external API services with the Red Hat 3scale API Management Platform. APIcast does load balancing by using round-robin.

In this guide you will learn about deployment options, environments provided, and how to get started.

Prerequisites

APIcast is not a standalone API gateway. It needs connection to 3scale API Manager.

- A working 3scale On-Premises instance.

To install APIcast, perform the steps outlined in the following sections:

- APIcast deployment options
- APIcast environments
- Configuring the integration settings
- Configuring your service
- Deploying APIcast on the Docker containerized environment
- Deploying APIcast using the OpenShift template
- Deploying an APIcast gateway self-managed solution using the operator

3.1. APICAST DEPLOYMENT OPTIONS

You can use hosted or self-managed APIcast. In both cases, APIcast must be connected to the rest of the 3scale API Management platform:

- **Embedded APIcast**: Two APIcast gateways (staging and production) come by default with the 3scale API Management installation. They come pre-configured and ready to use out-of-the-box.
- **Self-managed APIcast**: You can deploy APIcast wherever you want. Here are a few recommended options to deploy APIcast:
  - Deploying APIcast on the Docker containerized environment: Download a ready to use Docker-formatted container image, which includes all of the dependencies to run APIcast in a Docker-formatted container.
  - Running APIcast on Red Hat OpenShift: Run APIcast on a supported version of OpenShift. You can connect self-managed APIcasts to a 3scale On-premises installation or to a 3scale Hosted (SaaS) account. For this, you have the option to deploy APIcast using the Openshift template or to deploy an APIcast gateway self-managed solution using the operator .

3.2. APICAST ENVIRONMENTS

By default, when you create a 3scale account, you get embedded APIcast in two different environments:

- **Staging**: Intended to be used only while configuring and testing your API integration. When you have confirmed that your setup is working as expected, then you can choose to deploy it to the

production environment. The OpenShift template sets the parameters of the Staging APIcast in a way that the configuration is reloaded on each API call (**APICAST_CONFIGURATION_LOADER: lazy**, **APICAST_CONFIGURATION_CACHE: 0**). It is useful to test the changes in APIcast configuration quickly.

- **Production**: This environment is intended for production use. The following parameters are set for the Production APIcast in the OpenShift template: **APICAST_CONFIGURATION_LOADER: boot**, **APICAST_CONFIGURATION_CACHE: 300**. This means that the configuration will be fully loaded when APIcast is started, and will be cached for 300 seconds (5 minutes). After 5 minutes the configuration will be reloaded. This means that when you promote the configuration to production, it may take up to 5 minutes to be applied, unless you trigger a new deployment of APIcast.

3.3. CONFIGURING THE INTEGRATION SETTINGS

As a 3scale administrator, configure the integration settings for the environment you require 3scale to run in.

Prerequisites

A 3scale account with administrator privileges.

Procedure

1. Navigate to **[Your_API_name] > Integration > Settings**
2. Under **Deployment**, the default options are as follows:
   - Deployment Option: APIcast 3scale managed
   - Authentication mode: API key.
3. Change to your preferred option.
4. To save your changes, click **Update Product**.

3.4. CONFIGURING YOUR SERVICE

You must declare your API back-end in the *Private Base URL* field, which is the endpoint host of your API back-end. APIcast will redirect all traffic to your API back-end after all authentication, authorization, rate limits and statistics have been processed.

This section will guide you through configuring your service:

- Declaring the API backend
- Configuring the authentication settings
- Configuring the API test call

3.4.1. Declaring the API backend

Typically, the Private Base URL of your API will be something like **https://api-backend.yourdomain.com:443**, on the domain that you manage (**yourdomain.com**). For instance, if you were integrating with the Twitter API the Private Base URL would be **https://api.twitter.com/**.

In this example, you will use the **Echo API** hosted by 3scale, a simple API that accepts any path and returns information about the request (path, request parameters, headers, etc.). Its Private Base URL is **https://echo-api.3scale.net:443**.

Procedure

- Test your private (unmanaged) API is working. For example, for the Echo API you can make the following call with **curl** command:

```
curl "https://echo-api.3scale.net:443"
```

You will get the following response:

```
{
   "method": "GET",
   "path": "/",
   "args": "",
   "body": "",
   "headers": {
     "HTTP_VERSION": "HTTP/1.1",
     "HTTP_HOST": "echo-api.3scale.net",
     "HTTP_ACCEPT": "*/*",
     "HTTP_USER_AGENT": "curl/7.51.0",
     "HTTP_X_FORWARDED_FOR": "2.139.235.79, 10.0.103.58",
     "HTTP_X_FORWARDED_HOST": "echo-api.3scale.net",
     "HTTP_X_FORWARDED_PORT": "443",
     "HTTP_X_FORWARDED_PROTO": "https",
     "HTTP_FORWARDED": "for=10.0.103.58;host=echo-api.3scale.net;proto=https"
   },
   "uuid": "ee626b70-e928-4cb1-a1a4-348b8e361733"
 }
```

3.4.2. Configuring the authentication settings

You can configure authentication settings for your API in the **AUTHENTICATION** section under **[Your_product_name] > Integration > Settings**

Table 3.1. Optional authentication fields

Field	Description
Auth user key	Set the user key associated with the credentials location.
Credentials location	Define whether credentials are passed as HTTP headers, query parameters or as HTTP basic authentication.
Host Header	Define a custom Host request header. This is required if your API backend only accepts traffic from a specific host.

Field	Description
Secret Token	Used to block direct developer requests to your API backend. Set the value of the header here, and ensure your backend only allows calls with this secret header.

Furthermore, you can configure the **GATEWAY RESPONSE** error codes under **[Your_product_name] > Integration > Settings**. Define the *Response Code*, *Content-type*, and *Response Body* for the errors: Authentication failed, Authentication missing, and No match.

Table 3.2. Response codes and default response body

Response code	Response body
403	Authentication failed
403	Authentication parameters missing
404	No Mapping Rule matched
429	Usage limit exceeded

3.4.3. Configuring the API test call

Configuring the API involves testing the backends with a product and promoting the APIcast configuration to staging and production environments to make tests based on request calls.

For each product, requests get redirected to their corresponding backend according to the path. This path is configured when you add the backend to the product. For example, if you have two backends added to a product, each backend has its own path.

Prerequisites

- One or more backends added to a product .
- A mapping rule for each backend added to a product.
- An application plan to define the access policies.
- An application that subscribes to the application plan.

Procedure

1. Promote an APIcast configuration to Staging, by navigating to **[Your_product_name] > Integration > Configuration**.
2. Under *APIcast Configuration*, you will see the mapping rules for each backend added to the product. Click **Promote v.[n] to Staging APIcast**

- **v.[n]** indicates the version number to be promoted.

3. Once promoted to staging, you can promote to Production. Under *Staging APIcast*, click **Promote v.[n] to Production APIcast**
   - **v.[n]** indicates the version number to be promoted.
4. To test requests to your API in the command line, use the command provided in *Example curl for testing*.
   - The curl command example is based on the first mapping rule in the product.

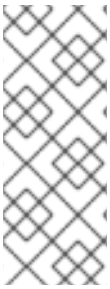
When testing requests to your API, you can modify the mapping rules by adding methods and metrics.

Every time you modify the configuration and before making calls to your API, make sure you promote to the Staging and Production environments. When there are pending changes to be promoted to the Staging environment, you will see an exclamation mark in the Admin Portal, next to the **Integration** menu item.

3scale Hosted APIcast gateway does the validation of the credentials and applies the rate limits that you defined for the application plan of your API. If you make a call without credentials, or with invalid credentials, you will see the error message, **Authentication failed**.

3.5. DEPLOYING APICAST ON THE DOCKER CONTAINERIZED ENVIRONMENT

This is a step-by-step guide to deploy APIcast inside a Docker container engine that is ready to be used as a Red Hat 3scale API Management API gateway.



NOTE

When deploying APIcast on the Docker containerized environment, the supported versions of Red Hat Enterprise Linux (RHEL) and Docker are as follows:

- RHEL 7.7
- Docker 1.13.1

Prerequisites

- You must configure APIcast in your 3scale Admin Portal as per Chapter 3, *Installing APIcast*.
- Access to the Red Hat Ecosystem Catalog .
  - To create a registry service account, see Creating and modifying registry service accounts .

To deploy APIcast on the docker containerized environment, perform the steps outlined in the following sections:

- Section 3.5.1, “Installing the Docker containerized environment”
- Section 3.5.2, “Running the Docker containerized environment gateway”

3.5.1. Installing the Docker containerized environment

This guide covers the steps to set up the Docker containerized environment on RHEL 7.x.

The Docker container engine provided by Red Hat is released as part of the Extras channel in RHEL. To enable additional repositories, you can use either the Subscription Manager or the *yum-config-manager* option. For details, see the RHEL product documentation.

To deploy RHEL 7.x on an Amazon Web Services (AWS), Amazon Elastic Compute Cloud (Amazon EC2) instance, take the following steps:

Procedure

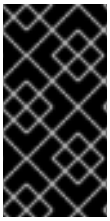
1. List all repositories: **sudo yum repolist all**.
2. Find the ***-extras** repository.
3. Enable the **extras** repository: **sudo yum-config-manager --enable rhui-REGION-rhel-server-extras**.
4. Install the Docker containerized environment package: **sudo yum install docker**.

Additional resources

For other operating systems, refer to the following Docker documentation:

- Installing the Docker containerized environment on Linux distributions
- Installing the Docker containerized environment on Mac
- Installing the Docker containerized environment on Windows

3.5.2. Running the Docker containerized environment gateway



IMPORTANT

In 3scale 2.11, support for an APIcast deployment running as a container in RHEL 7 and Docker is deprecated. In future releases, 3scale will support only RHEL 8 and Podman. If you are running APIcast self-managed as a container, upgrade your installation to the supported configuration.

To run the docker containerized environment gateway, do the following:

Procedure

1. Start the Docker daemon:

```
sudo systemctl start docker.service
```

2. Check if the Docker daemon is running:

```
sudo systemctl status docker.service
```

3. Download a ready to use Docker container engine image from the Red Hat registry:

```
sudo docker pull registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.11
```

4. Run APIcast in a Docker container engine:

```
sudo docker run --name apicast --rm -p 8080:8080 -e
THREESCALE_PORTAL_ENDPOINT=https://<access_token>@<domain>-admin.3scale.net
registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.11
```

Here, **<access_token>** is the Access Token for the 3scale Account Management API. You can use the Provider Key instead of the access token. **<domain>-admin.3scale.net** is the URL of your 3scale Admin Portal.

This command runs a Docker container engine called *"apicast"* on port **8080** and fetches the JSON configuration file from your 3scale Admin Portal. For other configuration options, see Installing APIcast.

3.5.2.1. The docker command options

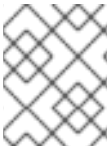
You can use the following options with the **docker run** command:

- **--rm**: Automatically removes the container when it exits.
- **-d** or **--detach**: Runs the container in the background and prints the container ID. When it is not specified, the container runs in the foreground mode and you can stop it using **CTRL + c**. When started in the detached mode, you can reattach to the container with the **docker attach** command, for example, **docker attach apicast**.
- **-p** or **--publish**: Publishes a container's port to the host. The value should have the format **<host port="">:<container port="">**, so **-p 80:8080** will bind port **8080** of the container to port **80** of the host machine. For example, the Management API uses port **8090**, so you may want to publish this port by adding **-p 8090:8090** to the **docker run** command.
- **-e** or **--env**: Sets environment variables.
- **-v** or **--volume**: Mounts a volume. The value is typically represented as **<host path="">:<container path="">[:<options>]**. **<options>** is an optional attribute; you can set it to **:ro** to specify that the volume will be read only (by default, it is mounted in read-write mode). Example: **-v /host/path:/container/path:ro**.

3.5.2.2. Testing APIcast

The preceding steps ensure that your Docker container engine is running with your own configuration file and the Docker container image from the 3scale registry. You can test calls through APIcast on port **8080** and provide the correct authentication credentials, which you can get from your 3scale account.

Test calls will not only verify that APIcast is running correctly but also that authentication and reporting is being handled successfully.



NOTE

Ensure that the host you use for the calls is the same as the one configured in the *Public Base URL* field on the **Integration** page.

Additional resources

- For more information on available options, see Docker run reference.

3.5.3. Additional resources

- For more information about tested and supported configuration, see Red Hat 3scale API Management Supported Configurations

3.5.4. Deploying APIcast on Podman

This is a step-by-step guide for deploying APIcast on a Pod Manager (Podman) container environment to be used as a Red Hat 3scale API Management API gateway.

NOTE

When deploying APIcast on a Podman container environment, the supported versions of Red Hat Enterprise Linux (RHEL) and Podman are as follows:

- RHEL 8.x
- Podman 1.4.2

Prerequisites

- You must configure APIcast in your 3scale Admin Portal as per Chapter 3, *Installing APIcast*.
- Access to the Red Hat Ecosystem Catalog .
  - To create a registry service account, see Creating and modifying registry service accounts .

To deploy APIcast on the Podman container environment, perform the steps outlined in the following sections:

- Section 3.5.4.1, “Installing the Podman container environment”
- Section 3.5.4.2, “Running the Podman environment”

3.5.4.1. Installing the Podman container environment

This guide covers the steps to set up the Podman container environment on RHEL 8.x. Docker is not included in RHEL 8.x, therefore, use Podman for working with containers.

For more details about Podman with RHEL 8.x, see the Container command-line reference.

Procedure

- Install the Podman container environment package:
  sudo dnf install podman

Additional resources

For other operating systems, refer to the following Podman documentation:

- Podman Installation Instructions

3.5.4.2. Running the Podman environment

To run the Podman container environment, follow the procedure below.

Procedure

1. Download a ready to use Podman container image from the Red Hat registry:

```
podman pull registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.11
```

2. Run APIcast in a Podman:

```
podman run --name apicast --rm -p 8080:8080 -e
THREESCALE_PORTAL_ENDPOINT=https://<access_token>@<domain>-admin.3scale.net
registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.11
```

   Here, **<access_token>** is the Access Token for the 3scale Account Management API. You can use the Provider Key instead of the access token. **<domain>-admin.3scale.net** is the URL of your 3scale Admin Portal.

This command runs a Podman container engine called *"apicast"* on port **8080** and fetches the JSON configuration file from your 3scale Admin Portal. For other configuration options, see Installing APIcast.

3.5.4.2.1. Testing APIcast with Podman

The preceding steps ensure that your Podman container engine is running with your own configuration file and the Podman container image from the 3scale registry. You can test calls through APIcast on port **8080** and provide the correct authentication credentials, which you can get from your 3scale account.

Test calls will not only verify that APIcast is running correctly but also that authentication and reporting is being handled successfully.

NOTE

Ensure that the host you use for the calls is the same as the one configured in the *Public Base URL* field on the **Integration** page.

3.5.4.3. The podman command options

You can use the following option examples with the **podman** command:

- **-d**: Runs the container in *detached mode* and prints the container ID. When it is not specified, the container runs in the foreground mode and you can stop it using **CTRL + c**. When started in the detached mode, you can reattach to the container with the **podman attach** command, for example, **podman attach apicast**.
- **ps** and **-a**: Podman **ps** is used to list creating and running containers. Adding **-a** to the **ps** command will show all containers, both running and stopped, for example, **podman ps -a**.
- **inspect** and **-l**: Inspect a running container. For example, use **inspect** to see the ID that was assigned to the container. Use **-l** to get the details for the latest container, for example, **podman inspect -l | grep Id\":**.

3.5.4.4. Additional resources

- For more information about tested and supported configurations, see Red Hat 3scale API Management Supported Configurations.
- For information about getting started with Podman, see Basic Setup and Use of Podman .

3.6. DEPLOYING APICAST USING THE OPENSHIFT TEMPLATE

You can deploy the APIcast API Gateway using the OpenShift template. Deploying the APIcast API Gateway helps protect your APIs, and you can analyze and monitor the traffic to them.

Prerequisites

- You must configure APIcast in your Red Hat 3scale API Management Admin Portal as per Installing APIcast.
- Make sure *Self-managed Gateway* is selected as the deployment option in the integration settings.
- You should have both staging and production environment configured to proceed.

Procedure

1. By default you are logged in as *developer* and can proceed to the next step.
   Otherwise login into OpenShift using the **oc login** command from the OpenShift Client tools you downloaded and installed in the previous step. The default login credentials are *username = "developer"* and *password = "developer"*:

   ```
   oc login https://OPENSHIFT-SERVER-IP:8443
   ```

   You should see **Login successful.** in the output.

2. Create your project. This example sets the display name as *gateway*

   ```
   oc new-project "3scalegateway" --display-name="gateway" --description="3scale gateway demo"
   ```

   The response should look like this:

   ```
   Now using project "3scalegateway" on server "https://172.30.0.112:8443"
   ```

   Ignore the suggested next steps in the text output at the command prompt and proceed to the next step below.

3. Create a new secret to reference your project by replacing **<access_token>** and **<domain>** with your own credentials. See below for more information about the **<access_token>** and **<domain>**.

   ```
   oc create secret generic apicast-configuration-url-secret --from-literal=password=https://<access_token>@<admin_portal_domain>  --type=kubernetes.io/basic-auth
   ```

   Here **<access_token>** is an Access Token for the 3scale account, and **<domain>-admin.3scale.net** is the URL of your 3scale Admin Portal.

   The response should look like this:

   ```
   secret/apicast-configuration-url-secret
   ```

4. Create an application for your APIcast gateway from the template, and start the deployment:

```
oc new-app -f https://raw.githubusercontent.com/3scale/3scale-amp-openshift-templates/2.11.0.GA/apicast-gateway/apicast.yml
```

You should see the following messages at the bottom of the output:

```
--> Creating resources with label app=3scale-gateway ...
    deploymentconfig "apicast" created
    service "apicast" created
--> Success
    Run 'oc status' to view your app.
```

3.7. DEPLOYING AN APICAST GATEWAY SELF-MANAGED SOLUTION USING THE OPERATOR

This guide provides steps for deploying an APIcast gateway self-managed solution using the APIcast operator via the Openshift Container Platform console.

Prerequisites

- OpenShift Container Platform (OCP) 4.x or later with administrator privileges.
- You followed the steps in Installing the APIcast operator on OpenShift .

Procedure

1. Log in to the OCP console using an account with administrator privileges.
2. Click **Operators > Installed Operators**.
3. Click the *APIcast Operator* from the list of *Installed Operators*.
4. Click the **APIcast > Create APIcast**

3.7.1. APICast deployment and configuration options

You can deploy and configure an APIcast gateway self-managed solution using two approaches:

- Providing a 3scale system endpoint
- Providing a configuration secret

See also:

- Injecting custom environments with the APIcast operator
- Injecting custom policies with the APIcast operator
- Configuring OpenTracing with the APIcast operator

3.7.1.1. Providing a 3scale system endpoint

Procedure

1. Create an OpenShift secret that contains 3scale System Admin Portal endpoint information:

```
oc create secret generic ${SOME_SECRET_NAME} --from-
literal=AdminPortalURL=${MY_3SCALE_URL}
```

   - **${SOME_SECRET_NAME}** is the name of the secret and can be any name you want as long as it does not conflict with an existing secret.
   - **${MY_3SCALE_URL}** is the URI that includes your 3scale access token and 3scale System portal endpoint. For more details, see **THREESCALE_PORTAL_ENDPOINT**

     Example

```
oc create secret generic 3scaleportal --from-literal=AdminPortalURL=https://access-
token@account-admin.3scale.net
```

     For more information about the contents of the secret see the Admin portal configuration secret reference.

2. Create the OpenShift object for APIcast

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
spec:
  adminPortalCredentialsRef:
    name: SOME_SECRET_NAME
```

   The **spec.adminPortalCredentialsRef.name** must be the name of the existing OpenShift secret that contains the 3scale system Admin Portal endpoint information.

3. Verify the APIcast pod is running and ready, by confirming that the **readyReplicas** field of the OpenShift Deployment associated with the APIcast object is *1*. Alternatively, wait until the field is set with:

```
$ echo $(oc get deployment apicast-example-apicast -o jsonpath='{.status.readyReplicas}')
1
```

3.7.1.1.1. Verifying the APIcast gateway is running and available

Procedure

1. Ensure the OpenShift Service APIcast is exposed to your local machine, and perform a test request. Do this by port-forwarding the APIcast OpenShift Service to **localhost:8080**:

```
oc port-forward svc/apicast-example-apicast 8080
```

2. Make a request to a configured 3scale service to verify a successful HTTP response. Use the domain name configured in **Staging Public Base URL** or **Production Public Base URL** settings of your service. For example:

```
$ curl 127.0.0.1:8080/test -H "Host: myhost.com"
```

3.7.1.1.2. Exposing APIcast externally via a Kubernetes Ingress

To expose APIcast externally via a Kubernetes Ingress, set and configure the **exposedHost** section. When the **host** field in the **exposedHost** section is set, this creates a Kubernetes Ingress object. The Kubernetes Ingress object can then be used by a previously installed and existing Kubernetes Ingress Controller to make APIcast accessible externally.

To learn what Ingress Controllers are available to make APIcast externally accessible and how they are configured see the Kubernetes Ingress Controllers documentation .

The following example to expose APIcast with the hostname **myhostname.com**:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
spec:
  ...
  exposedHost:
    host: "myhostname.com"
  ...
```

The example creates a Kubernetes Ingress object on the port 80 using HTTP. When the APIcast deployment is in an OpenShift environment, the OpenShift default Ingress Controller will create a Route object using the Ingress object APIcast creates which allows external access to the APIcast installation.

You may also configure TLS for the **exposedHost** section. Details about the available fields in the following table:

Table 3.3. APIcastExposedHost reference table

json/yaml field	Type	Required	Default value	Description
host	string	Yes	N/A	Domain name being routed to the gateway
tls	[]extensions.IngressTLS	No	N/A	Array of ingress TLS objects. See more on TLS.

3.7.1.2. Providing a configuration secret

Procedure

1. Create a secret with the configuration file:

```
$ curl
https://raw.githubusercontent.com/3scale/APIcast/master/examples/configuration/echo.json -
o $PWD/config.json

oc create secret generic apicast-echo-api-conf-secret --from-file=$PWD/config.json
```

The configuration file must be called **config.json**. This is an APIcast CRD reference requirement.

For more information about the contents of the secret see the Admin portal configuration secret reference.

2. Create an APIcast custom resource:

```
$ cat my-echo-apicast.yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: my-echo-apicast
spec:
  exposedHost:
    host: YOUR DOMAIN
  embeddedConfigurationSecretRef:
    name: apicast-echo-api-conf-secret

$ oc apply -f my-echo-apicast.yaml
```

   a. The following is an example of an embedded configuration secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: SOME_SECRET_NAME
type: Opaque
stringData:
  config.json: |
    {
      "services": [
       {
         "proxy": {
          "policy_chain": [
           { "name": "apicast.policy.upstream",
             "configuration": {
               "rules": [{
                 "regex": "/",
                 "url": "http://echo-api.3scale.net"
               }]
             }
           }
          ]
         }
       }
      ]
    }
```

3. Set the following content when creating the APIcast object:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
```

```
spec:
  embeddedConfigurationSecretRef:
    name: SOME_SECRET_NAME
```

The **spec.embeddedConfigurationSecretRef.name** must be the name of the existing OpenShift secret that contains the configuration of the gateway.

4. Verify the APIcast pod is running and ready, by confirming that the **readyReplicas** field of the OpenShift Deployment associated with the APIcast object is *1*. Alternatively, wait until the field is set with:

```
$ echo $(oc get deployment apicast-example-apicast -o jsonpath='{.status.readyReplicas}')
1
```

3.7.1.2.1. Verifying APIcast gateway is running and available

Procedure

1. Ensure the OpenShift Service APIcast is exposed to your local machine, and perform a test request. Do this by port-forwarding the APIcast OpenShift Service to **localhost:8080**:

```
oc port-forward svc/apicast-example-apicast 8080
```

2. Make a request to a configured 3scale Service to verify a successful HTTP response. Use the domain name configured in **Staging Public Base URL** or **Production Public Base URL** settings of your service. For example:

```
$  curl 127.0.0.1:8080/test -H "Host: localhost"
{
  "method": "GET",
  "path": "/test",
  "args": "",
  "body": "",
  "headers": {
    "HTTP_VERSION": "HTTP/1.1",
    "HTTP_HOST": "echo-api.3scale.net",
    "HTTP_ACCEPT": "*/*",
    "HTTP_USER_AGENT": "curl/7.65.3",
    "HTTP_X_REAL_IP": "127.0.0.1",
    "HTTP_X_FORWARDED_FOR": ...
    "HTTP_X_FORWARDED_HOST": "echo-api.3scale.net",
    "HTTP_X_FORWARDED_PORT": "80",
    "HTTP_X_FORWARDED_PROTO": "http",
    "HTTP_FORWARDED": "for=10.0.101.216;host=echo-api.3scale.net;proto=http"
  },
  "uuid": "603ba118-8f2e-4991-98c0-a9edd061f0f0"
```

3.7.1.3. Injecting custom environments with the APIcast operator

In a 3scale installation that uses self-managed APIcast, you can use the **APIcast** operator to inject custom environments. A custom environment defines behavior that APIcast applies to all upstream APIs that the gateway serves. To create a custom environment, define a global configuration in Lua code.

You can inject a custom environment as part of or after APIcast installation. After injecting a custom environment, you can remove it and the **APIcast** operator reconciles the changes.

Prerequisites

- The APIcast operator is installed.

Procedure

1. Write Lua code that defines the custom environment that you want to inject. For example, the following **env1.lua** file shows a custom logging policy that the **APIcast** operator loads for all services.

```
local cjson = require('cjson')
local PolicyChain = require('apicast.policy_chain')
local policy_chain = context.policy_chain

local logging_policy_config = cjson.decode([[
{
  "enable_access_logs": false,
  "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}"
}
]])

policy_chain:insert( PolicyChain.load_policy('logging', 'builtin', logging_policy_config), 1)

return {
  policy_chain = policy_chain,
  port = { metrics = 9421 },
}
```

2. Create a secret from the Lua file that defines the custom environment. For example:

```
oc create secret generic custom-env-1 --from-file=./env1.lua
```

A secret can contain multiple custom environments. Specify the **–from-file** option for each file that defines a custom environment. The operator loads each custom environment.

3. Define an **APIcast** custom resource that references the secret you just created. The following example shows only content relative to referencing the secret that defines the custom environment.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  customEnvironments:
    - secretRef:
        name: custom-env-1
```

An **APIcast** custom resource can reference multiple secrets that define custom environments. The operator loads each custom environment.

4. Create the **APIcast** custom resource that adds the custom environment. For example, if you saved the **APIcast** custom resource in the **apicast.yaml** file, run the following command:

```
oc apply -f apicast.yaml
```

Next steps

You cannot update the content of a secret that defines a custom environment. If you need to update the custom environment, update the Lua file that defines the custom environment and then do either of the following:

- The recommended option is to create a secret with a different name and then update the **APIcast** custom resource field, **spec.customEnvironments[].secretRef.name**, for the custom environment that you updated. The operator triggers a rolling update and loads the updated custom environment.
- Alternatively, you can update the existing secret, redeploy APIcast by setting **spec.replicas** to 0, and then redploy APIcast again by setting **spec.replicas** back to its previous value.

3.7.1.4. Injecting custom policies with the APIcast operator

In a 3scale installation that uses self-managed APIcast, you can use the **APIcast** operator to inject custom policies. Injecting a custom policy adds the policy code to APIcast. You can then use either of the following to add the custom policy to an API product's policy chain:

- 3scale API
- **Product** custom resource

To use the 3scale Admin Portal to add the custom policy to a product's policy chain, you must also register the custom policy's schema with a **CustomPolicyDefinition** custom resource. Custom policy registration is a requirement only when you want to use the Admin Portal to configure a product's policy chain.

You can inject a custom policy as part of or after APIcast installation. After injecting a custom policy, you can remove it and the **APIcast** operator reconciles the changes.

Prerequisites

- The APIcast operator is installed or you are in the process of installing it.
- You have defined a custom policy as described in Write your own policy. That is, you have already created, for example, the **my-first-custom-policy.lua**, **apicast-policy.json**, and **init.lua** files that define a custom policy,

Procedure

1. Create a secret from the files that define one custom policy. For example:

```
oc create secret generic my-first-custom-policy-secret \
 --from-file=./apicast-policy.json \
 --from-file=./init.lua \
 --from-file=./my-first-custom-policy.lua
```

   If you have more than one custom policy, create a secret for each custom policy. A secret can contain only one custom policy.

2. Define an **APIcast** custom resource that references the secret you just created. The following example shows only content relative to referencing the secret that defines the custom policy.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  customPolicies:
    - name: my-first-custom-policy
      version: "0.1"
      secretRef:
        name: my-first-custom-policy-secret
```

   An **APIcast** custom resource can reference multiple secrets that define custom policies. The operator loads each custom policy.

3. Create the **APIcast** custom resource that adds the custom policy. For example, if you saved the **APIcast** custom resource in the **apicast.yaml** file, run the following command:

```
oc apply -f apicast.yaml
```

Next steps

You cannot update the content of a secret that defines a custom policy. If you need to update the custom policy, update its files and then do either of the following:

- The recommended option is to create a secret with a different name and then update the **APIcast** custom resource field, **spec.customPolicies[].secretRef.name**, for the custom policy that you updated. The operator triggers a rolling update and loads the updated custom policy.
- Alternatively, you can update the existing secret, redeploy APIcast by setting **spec.replicas** to 0, and then redploy APIcast again by setting **spec.replicas** back to its previous value.

Additional resources

- APIcast custom resource definition

3.7.1.5. Configuring OpenTracing with the APIcast operator

In a 3scale installation that uses self-managed APIcast, you can use the **APIcast** operator to configure OpenTracing. By enabling OpenTracing, you get more insight and better observability on the APIcast instance.

Prerequisites

- The **APIcast** operator is installed or you are in the process of installing it.
- Prerequisites listed in Configuring APIcast to use OpenTracing .
- Jaeger is installed.

Procedure

1. Define a secret that contains your OpenTracing configuration details in **stringData.config**. This

is the only valid value for the attribute that contains your OpenTracing configuration details. Any other specification prevents APIcast from receiving your OpenTracing configuration details. The folowing example shows a valid secret definition:

```
apiVersion: v1
kind: Secret
metadata:
  name: myjaeger
stringData:
  config: |-
      {
      "service_name": "apicast",
      "disabled": false,
      "sampler": {
        "type": "const",
        "param": 1
      },
      "reporter": {
        "queueSize": 100,
        "bufferFlushInterval": 10,
        "logSpans": false,
        "localAgentHostPort": "jaeger-all-in-one-inmemory-agent:6831"
      },
      "headers": {
        "jaegerDebugHeader": "debug-id",
        "jaegerBaggageHeader": "baggage",
        "TraceContextHeaderName": "uber-trace-id",
        "traceBaggageHeaderPrefix": "testctx-"
      },
      "baggage_restrictions": {
          "denyBaggageOnInitializationFailure": false,
          "hostPort": "127.0.0.1:5778",
          "refreshInterval": 60
      }
      }
type: Opaque
```

2. Create the secret. For example, if you saved the previous secret definition in the **myjaeger.yaml** file, you would run the following command:

```
oc create secret generic myjaeger --from-file myjaeger.yaml
```

3. Define an **APIcast** custom resource that specifies the **OpenTracing** attributes. In the CR definition, set the **spec.tracingConfigSecretRef.name** attribute to the name of the secret that contains your OpenTracing configuration details. The following example shows only content relative to configuring OpenTracing.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  ...
  openTracing:
    enabled: true
```

```
    tracingConfigSecretRef:
      name: myjaeger
    tracingLibrary: jaeger
...
```

4. Create the **APIcast** custom resource that configures OpenTracing. For example, if you saved the **APIcast** custom resource in the **apicast1.yaml** file, you would run the following command:

```
oc apply -f apicast1.yaml
```

Next steps

Depending on how OpenTracing is installed, you should see the traces in the Jaeger service user interface.

Additional resource

- APIcast custom resource definition

3.8. ADDITIONAL RESOURCES

To get information about the latest released and supported version of APIcast, see the articles:

- Red Hat 3scale API Management Supported Configurations
- Red Hat 3scale API Management - Component Details .

CHAPTER 4. INSTALLING THE 3SCALE OPERATOR ON OPENSHIFT

NOTE

3scale supports the last two general availability (GA) releases of OpenShift Container Platform (OCP). For more information, see the Red Hat 3scale API Management Supported Configurations page.

This documentation shows you how to:

- Create a new project.
- Deploy a Red Hat 3scale API Management instance.
- Install the 3scale operator through Operator Lifecycle Manager (OLM).
- Deploy the custom resources once the operator has been deployed.

Prerequisites

- Access to a supported version of an OpenShift Container Platform 4 cluster using an account with administrator privileges.
  - For more information about supported configurations, see the Red Hat 3scale API Management Supported Configurations page.

WARNING

Deploy the 3scale operator and custom resource definitions (CRDs) in a separate newly created, empty *project*. If you deploy them in an existing project containing infrastructure, it could alter or delete existing elements.

To install the 3scale operator on OpenShift, perform the steps outlined in the following sections:

- Section 4.1, “Creating a new OpenShift project”
- Section 4.2, “Installing and configuring the 3scale operator using the OLM”

4.1. CREATING A NEW OPENSHIFT PROJECT

This procedure explains how to create a new OpenShift project named **3scale-project**. Replace this project name with your own.

Procedure

To create a new OpenShift project:

- Indicate a valid name using alphanumeric characters and dashes. As an example, run the command below to create **3scale-project**:

```
oc new-project 3scale-project
```

This creates the new *OpenShift project* where the operator, the *APIManager* custom resource (CR), and the *Capabilities* custom resources will be installed. The operator manages the custom resources through OLM in that project.

4.2. INSTALLING AND CONFIGURING THE 3SCALE OPERATOR USING THE OLM

Use Operator Lifecycle Manager (OLM) to install the 3scale operator on an OpenShift Container Platform (OCP) 4.3 cluster through the OperatorHub in the OCP console.

NOTE

- When using the OCP on a restricted network or a disconnected cluster, OLM can no longer use the OperatorHub. Follow the instructions for setting up and using the OLM in the guide titled Using Operator Lifecycle Manager on restricted networks.

Prerequisites

- You must install and deploy the 3scale operator in the project that you defined in Creating a new OpenShift project.

Procedure

1. In the OpenShift Container Platform console, log in using an account with administrator privileges.
2. The menu structure depends on the version of OpenShift you are using:
   - Click **Operators > OperatorHub**
3. In the **Filter by keyword** box, type *3scale operator* to find the 3scale operator.
4. Click the 3scale operator. Information about the Operator is displayed.
5. Read the information about the operator and click **Install**. The *Create Operator Subscription* page opens.
6. On the **Create Operator Subscription** page, accept all of the default selections and click **Subscribe**.

NOTE

The operator will only be available in the specific single namespace on the cluster that you have selected.

The *3scale-operator* details page is displayed, where you can see the *Subscription Overview*.

7. Confirm that the subscription **upgrade status** is shown as **Up to date**.

8. Verify that the 3scale operator ClusterServiceVersion (CSV) is displayed, and the **Status** of the operator ultimately resolves to **InstallSucceeded** in the project you defined in Creating a new OpenShift project:
    - Click **Operators > Installed Operators**. In this case, successful installation will register the *APIManager* CRD, and the CRDs related to the *Capabilities* functionality of the operator in the *OpenShift API server*.
9. After successful installation, query the resource types defined by the CRDs via **oc get**.
    a. For example, to verify that the *APIManager* CRD has been correctly registered, execute the following command:

```
oc get apimanagers
```

10. You should see the following output:

```
No resources found.
```

Besides the indicated procedure, create a list of the allowed domains you intend to use in the 3scale Developer Portal while using OCP on restricted networks. Consider the following examples:

- Any link you intend to add to the Developer Portal.
- SSO integrations through third party SSO providers such as GitHub.
- Billing.
- Webhooks that trigger an external URL.

4.2.1. Restrictions in disconnected environments

The following list outlines current restrictions in a disconnected environment for 3scale 2.11:

- The GitHub login to the Developer Portal is not available.
- Support links are not operational.
- Links to external documentation are not operational.
- The validator for the OpenAPI Specification (OAS) in the Developer Portal is not operational, affecting links to external services.
- In the product *Overview* page in **ActiveDocs**, links to OAS are not operational.
    - It is also necessary to check the option *Skip swagger validations* when you create a new ActiveDocs specification.

Additional resources

- For troubleshooting information, see the OpenShift Container Platform documentation.
- For more information about using the OLM on retricted networks, see Using Operator Lifecycle Manager on restricted networks.

- For more information about preparing your installation on restricted networks, see Creating a mirror registry for installation in a restricted network.
- For more information about supported configurations, see the Red Hat 3scale API Management Supported Configurations page.

CHAPTER 5. INSTALLING THE APICAST OPERATOR ON OPENSHIFT

This guide provides steps for installing the APIcast operator through the OpenShift Container Platform (OCP) console.

Procedure

1. Log in to the OCP console using an account with administrator privileges.
2. Create new project **operator-test** in **Projects > Create Project**
3. Click **Operators > Installed Operators**
4. Type *apicast* in the *Filter by keyword* box to find the APIcast operator. Do not use the community version.
5. Click the APIcast operator. You will see information about the APIcast operator.
6. Click *Install*. The *Create Operator Subscription* page opens.
7. Click *Subscribe* to accept all of the default selections on the *Create Operator Subscription* page.
    a. The subscription upgrade status is shown as *Up to date*.
8. Click **Operators > Installed Operators** to verify that the APIcast operator *ClusterServiceVersion* (CSV) status displays to *InstallSucceeded* in the **operator-test** project.

CHAPTER 6. 3SCALE HIGH AVAILABILITY AND EVALUATION TEMPLATES

This document describes the templates for High Availability and Evaluation used by Red Hat 3scale API Management 2.11 installation.

Prerequisites

- You need to have an available OpenShift cluster to deploy elements of the High Availability and Evaluation templates.

IMPORTANT

The 3scale High Availability and Evaluation templates are a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

To deploy High Availability and Evaluation templates, perform the steps outlined in the following sections:

- Section 6.1, “High Availability template”
- Section 6.2, “Evaluation template”

6.1. HIGH AVAILABILITY TEMPLATE

The High Availability (HA) template allows you to have a HA setting for critical databases.

Prerequisites

- Before deploying the HA template, you must deploy and configure the external databases, and configure them in a HA configuration with a load-balanced endpoint.

Using the HA template

For HA, the template named **amp-ha-tech-preview.yml** allows you to deploy critical databases externally to OpenShift. This excludes:

- Memcached
- Sphinx
- Zync

Differences between the standard **amp.yml** template and **amp-ha-tech-preview.yml** include:

- Removal of the following elements:
  - backend-redis and its related components
  - system-redis and its related components

    - system-mysql and its related components
    - Redis and MySQL related ConfigMaps
    - MYSQL_IMAGE, REDIS_IMAGE, MYSQL_USER, MYSQL_ROOT_PASSWORD parameters
- By default, increased from 1 to 2 the number of replicas for non-database **DeploymentConfig** object types.
- Addition of the following mandatory parameters, allowing you the control of the location of external databases:
    - BACKEND_REDIS_STORAGE_ENDPOINT
    - BACKEND_REDIS_QUEUES_ENDPOINT
    - SYSTEM_REDIS_URL
    - APICAST_STAGING_REDIS_URL
    - APICAST_PRODUCTION_REDIS_URL
    - SYSTEM_DATABASE_URL

With **amp-ha-tech-preview.yml**, you need to configure database connections (excluding **system-memcache**, **zync-database** and **system-sphinx** that do not contain permanent data) out of the cluster via the newly added mandatory parameters. The endpoints require database load-balanced connection strings, including authentication information. Also, for the non-database deployments, the number of pod replicas is increased to 2 by default to have redundancy at application-level.

6.1.1. Setting RWX_STORAGE_CLASS for high availability

ReadWriteMany (RWX) PersistentVolumeClaims (PVCs) uses the storage class RWX_STORAGE_CLASS.

required: false

value: **null**

- Set this to **null** to signal OpenShift that you want the storage class to be auto-discovered (no value).
- If you set this to an empty string or no default value, it signals OpenShift that you want the string storage empty. This is an invalid setting.

6.2. EVALUATION TEMPLATE

For evaluation purposes, there is a template named **amp-eval-tech-preview.yml** that deploys a 3scale environment without resource requests nor limits.

The only functional difference compared to the standard **amp.yml** template is that the resource limits and requests have been removed. This means that in this version the minimum hardware requirements have been removed on the pods at CPU and Memory level. This template is intended only for evaluation, testing, and development purposes as it tries to deploy the components in a best-effort way with the given hardware resources.

CHAPTER 7. REDIS HIGH AVAILABILITY (HA) SUPPORT FOR 3SCALE

IMPORTANT

Red Hat does not officially support setting up Redis for zero downtime, configuring back-end components for 3scale, or Redis database replication and sharding. The content is for reference only. Additionally, Redis **cluster mode** is not supported in 3scale.

High availability (HA) is provided for most components by the OpenShift Container Platform (OCP). For more information see OpenShift Container Platform 3.11 Chapter 30. High Availability .

The database components for HA in Red Hat 3scale API Management include:

- **backend-redis**: used for statistics storage and temporary job storage.
- **system-redis**: provides temporary storage for background jobs for 3scale and is also used as a message bus for *Ruby* processes of **system-app** pods.

Both **backend-redis** and **system-redis** work with supported Redis high availability variants for Redis Sentinel and Redis Enterprise.

If the Redis pod comes to a stop, or if the OpenShift Container Platform stops it, a new pod is automatically created. Persistent storage will restore the data so the pod continues to work. In these scenarios, there will be a small amount of downtime while the new pod starts. This is due to a limitation in Redis that does not support a multi-master setup. You can reduce downtime by preinstalling the Redis images onto all nodes that have Redis deployed to them. This will speed up the pod restart time.

Set up Redis for zero downtime and configure back-end components for 3scale:

- Setting up Redis for zero downtime
- Configuring back-end components for 3scale
- Redis database sharding and replication

Prerequisites

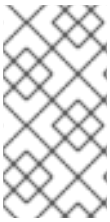
- A 3scale account with an administrator role.

7.1. SETTING UP REDIS FOR ZERO DOWNTIME

As a 3scale administrator, configure Redis outside of OCP if you require zero downtime. There are several ways to set it up using the configuration options of 3scale pods:

- Set up your own self-managed Redis
- Use Redis Sentinel: Reference Redis Sentinel Documentation
- Redis provided as a service:
  For example by:
  - Amazon ElastiCache

- Redis Labs



NOTE

Red Hat does not provide support for the above mentioned services. The mention of any such services does not imply endorsement by Red Hat of the products or services. You agree that Red Hat is not responsible or liable for any loss or expenses that may result due to your use of (or reliance on) any external content.

7.2. CONFIGURING BACK-END COMPONENTS FOR 3SCALE

As a 3scale administrator, configure Redis HA (failover) for the **back-end** component environment variables in the following deployment configurations: **backend-cron**, **backend-listener**, and **backend-worker**. These configurations are necessary for Redis HA in 3scale.

NOTE

If you want to use Redis with sentinels, you must create the **system-redis** secret with all fields in order to configure the Redis you want to point to before deploying 3scale. The fields are not provided as parameters in the back end as of 3scale.

7.2.1. Creating backend-redis and system-redis secrets

Follow these steps to create **backend-redis** and **system-redis** secrets accordingly:

- Deploying a fresh installation of 3scale for HA
- Migrating a non-HA deployment of 3scale to HA

7.2.2. Deploying a fresh installation of 3scale for HA

To prevent key collision when deploying with single-database Redis instances, set different *namespaces* for **sidekiq** and **message_bus** Redis keys. This applies to both Redis Enterprise and Redis Cluster.

For other deployments where **sidekiq** and **message_bus** read and write to different Redis databases, *namespaces* are not necessary.

The following parameters are used to set Redis key *namespaces*:

- **NAMESPACE**: for entries related to job queues stored by **system-app** and **system-sidekiq** in the Redis database.
- **MESSAGE_BUS_NAMESPACE**: for entries related to interprocess **message_bus** communication stored by **system-app** in the Redis database.

Procedure

1. Create the **backend-redis** and **system-redis** secrets with the fields below:

   backend-redis

   ```
   REDIS_QUEUES_SENTINEL_HOSTS
   REDIS_QUEUES_SENTINEL_ROLE
   REDIS_QUEUES_URL
   ```

```
REDIS_STORAGE_SENTINEL_HOSTS
REDIS_STORAGE_SENTINEL_ROLE
REDIS_STORAGE_URL
```

system-redis

```
MESSAGE_BUS_NAMESPACE
MESSAGE_BUS_SENTINEL_HOSTS
MESSAGE_BUS_SENTINEL_ROLE
MESSAGE_BUS_URL
NAMESPACE
SENTINEL_HOSTS
SENTINEL_ROLE
URL
```

- When configuring for Redis with sentinels, the corresponding **URL** fields in **backend-redis** and **system-redis** refer to the Redis group in the format **redis://[:redis-password@]redis-group[/db]**`, where [x] denotes optional element **x** and **redis-password**, **redis-group**, and **db** are variables to be replaced accordingly:

  Example

  ```
  redis://:redispwd@mymaster/5
  ```

- The **SENTINEL_HOSTS** fields are comma-separated lists of sentinel connection strings in the following format:

  ```
  redis://:sentinel-password@sentinel-hostname-or-ip:port
  ```

  - For each element of the list, [x] denotes optional element **x** and **sentinel-password**, **sentinel-hostname-or-ip**, and **port** are variables to be replaced accordingly:

    Example

    ```
    :sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722
    ```

- The **SENTINEL_ROLE** fields are either **master** or **slave**.

2. Deploy 3scale as indicated in Deploying 3scale on OpenShift using a template , using the latest version of the templates.

   a. Ignore the errors due to **backend-redis** and **system-redis** already present.

7.2.3. Migrating a non-HA deployment of 3scale to HA

1. Edit the **backend-redis** and **system-redis** secrets with all fields as shown in Deploying a fresh installation of 3scale for HA.

2. Make sure the following **backend-redis** environment variables are defined for the back-end pods.

```
name: BACKEND_REDIS_SENTINEL_HOSTS
 valueFrom:
  secretKeyRef:
```

```
      key: REDIS_STORAGE_SENTINEL_HOSTS
      name: backend-redis
name: BACKEND_REDIS_SENTINEL_ROLE
 valueFrom:
  secretKeyRef:
    key: REDIS_STORAGE_SENTINEL_ROLE
    name: backend-redis
```

3. Make sure the following **system-redis** environment variables are defined for the **system-(app|sidekiq|sphinx)** pods.

```
name: REDIS_SENTINEL_HOSTS
 valueFrom:
  secretKeyRef:
    key: SENTINEL_HOSTS
    name: system-redis
name: REDIS_SENTINEL_ROLE
 valueFrom:
  secretKeyRef:
    key: SENTINEL_ROLE
    name: system-redis
name: MESSAGE_BUS_REDIS_SENTINEL_HOSTS
 valueFrom:
  secretKeyRef:
    key: MESSAGE_BUS_SENTINEL_HOSTS
    name: system-redis
name: MESSAGE_BUS_REDIS_SENTINEL_ROLE
 valueFrom:
  secretKeyRef:
    key: MESSAGE_BUS_SENTINEL_ROLE
    name: system-redis
```

4. Proceed with instructions to continue Upgrading 3scale using templates.

7.2.3.1. Using Redis Enterprise

1. Use Redis Enterprise deployed in OpenShift, with three different **redis-enterprise** instances:
   a. Edit **system-redis** secret:
      i. Set distinct values to **MESSAGE_BUS_NAMESPACE** and **NAMESPACE**.
      ii. Set **URL** and **MESSAGE_BUS_URL** to the same database.
   b. Set the back-end database in **backend-redis** to **REDIS_QUEUES_URL**.
   c. Set the third database to **REDIS_STORAGE_URL** for **backend-redis**.

7.2.3.2. Using Redis Sentinel

1. Use Redis Sentinel, with three or four different Redis databases:
   a. Edit **system-redis** secret:
      i. Set distinct values to **MESSAGE_BUS_NAMESPACE** and **NAMESPACE**.

ii. Set **URL** and **MESSAGE_BUS_URL** to the proper Redis group, for example: **redis://:redispwd@mymaster/5**

iii. Set **SENTINEL_HOSTS** and **MESSAGE_BUS_SENTINEL_HOSTS** to a comma-separated list of sentinels hosts and ports, for example: **:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722**

iv. Set **SENTINEL_ROLE** and **MESSAGE_BUS_SENTINEL_ROLE** to *master*

2. Set the **backend-redis** secret for back-end with the values:
   - **REDIS_QUEUES_URL**
   - **REDIS_QUEUES_SENTINEL_ROLE**
   - **REDIS_QUEUES_SENTINEL_HOSTS**
3. Set the variables in the third database as follows:
   - **REDIS_STORAGE_URL**
   - **REDIS_STORAGE_SENTINEL_ROLE**
   - **REDIS_STORAGE_SENTINEL_HOSTS**

Notes

- The *system-app* and *system-sidekiq* components connect directly to **back-end** Redis for retrieving statistics.
  - As of 3scale 2.7, these system components can also connect to **back-end** Redis (storage) when using sentinels.
- The *system-app* and *system-sidekiq* components uses **only backend-redis** storage, not **backend-redis** queues.
  - Changes made to the system components support **backend-redis** storage with sentinels.

7.3. REDIS DATABASE SHARDING AND REPLICATION

Sharding, sometimes referred to as partitioning, separates large databases in to smaller databases called shards. With replication, your database is set up with copies of the same dataset hosted on separate machines.

Sharding

Sharding facilitates adding more leader instances, which is also useful when you have so much data that it does not fit in a single database, or when the CPU load is close to 100%.

With Redis HA for 3scale, the following two reasons are why sharding is important:

- Spliting and scaling large volumes of data and adjusting the number of shards for a particular index to help avoid bottlenecks.
- Distributing operations across different node, therefore increasing performance, for example, when multiple machines are working on the same query.

The three main solutions for Redis database sharding with cluster mode disabled are:

- Amazon ElastiCache
- Standard Redis via Redis sentinels
- Redis Enterprise

Replication

Redis database replication ensures redundancy by having your dataset replicated across different machines. Using replication allows you to keep Redis working when the leader goes down. Data is then pulled from a single instance, the leader, ensuring high availability.

With Redis HA for 3scale, database replication ensures high availability replicas of a primary shard. The principles of operation involve:

- When the primary shard fails, the replica shard will automatically be promoted to the new primary shard.
- Upon recovery of the original primary shard, it automatically becomes the replica shard of the new primary shard.

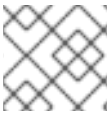
The three main solutions for Redis database replication are:

- Redis Enterprise
- Amazon ElastiCache
- Standard Redis via Redis sentinels

Sharding with twemproxy

For Amazon ElastiCache and Standard Redis, sharding involves splitting data up based on keys. You need a proxy component that given a particular key knows which shard to find, for example **twemproxy**. Also known as nutcracker, **twemproxy** is a lightweight proxy solution for Redis protocols that finds shards based on specific keys or server maps assigned to them. Adding sharding capabilities to your Amazon ElastiCache or Standard Redis instance with **twemproxy**, has the following advantages:

- The capability of sharding data automatically across multiple servers.
- Support of multiple hashing modes and consistent hashing and distribution.
- The capability to run in multiple instances, which allows clients to connect to the first available proxy server.
- Reduce the number of connections to the caching servers on the backend.



NOTE

Redis Enterprise uses its own proxy, so it does not need **twemproxy**.

Additional resources

- Redis Sentinel Documentation.
- **twemproxy**.

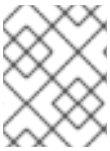
7.4. ADDITIONAL INFORMATION

- For more information about 3scale and Redis database support, see Red Hat 3scale API Management Supported Configurations.
- For more information about Amazon ElastiCache for Redis, see the official Amazon ElastiCache Documentation.
- For more information about Redis Enterprise, see the latest Documentation.

CHAPTER 8. CONFIGURING AN EXTERNAL MYSQL DATABASE

This guide provides information for externalizing the MySQL database for Chapter 6, *3scale High Availability and Evaluation templates*. This can be done by using the default *amp.yml* file. This is useful where there are several infrastructure issues, such as network or filesystem, using the default **system-mysql** pod.

The difference between this approach and the one in Chapter 6, *3scale High Availability and Evaluation templates* is that this provides a way for externalizing a MySQL database in case Red Hat 3scale API Management was initially using the default *amp.yml* template.



NOTE

Red Hat supports 3scale configurations that use an external MySQL database. However, the database itself is not within the scope of support.

Prerequisites

- Access to an OpenShift Container Platform 3.11 cluster using an account with administrator privileges.
- A 3scale instance installation on the OpenShift cluster. See Chapter 2, *Installing 3scale on OpenShift*.

To configure an external MySQL database for High Availability (HA), perform the steps outlined in the following sections:

- Section 8.1, "External MySQL database limitations"
- Section 8.2, "Externalizing the MySQL database"
- Section 8.3, "Rolling back"

8.1. EXTERNAL MYSQL DATABASE LIMITATIONS

There are limitations with the process of externalizing your MySQL database:

3scale On-premises versions

It has only been tested and verified on the 2.5 On-premises and 2.6 On-premises versions from 3scale.

MySQL database user

The URL must be in the following format:

<database_scheme>://<admin_user>:<admin_password>@<database_host>/<database_name>

An **<admin_user>** must be an existing user in the external database with full permissions on the **<database_name>** logical database. The **<database_name>** must be an already existing logical database in the external database.

MySQL host

Use the *IP address* from the external MySQL database instead of the *hostname* or it will not resolve. For example, use *1.1.1.1* instead of *mysql.mydomain.com*.

8.2. EXTERNALIZING THE MYSQL DATABASE

Use the following steps to fully externalize the MySQL database.

WARNING

This will cause downtime in the environment while the process is ongoing.

Procedure

1. Login to the OpenShift node where your 3scale On-premises instance is hosted and change to its project:

```
oc login -u <user> <url>
oc project <3scale-project>
```

   Replace **<user>**, **<url>**, and **<3scale-project>** with your own credentials and the project name.

2. Follow the steps below in the order shown to scale down all the pods. This will avoid loss of data.

Stop 3scale On-premises

From the OpenShift web console or from the command line interface (CLI), scale down all the deployment configurations to zero replicas in the following order:

- **apicast-wildcard-router** and **zync** for versions before 3scale 2.6 or **zync-que** and **zync** for 3scale 2.6 and above.
- **apicast-staging** and **apicast-production**.
- **system-sidekiq**, **backend-cron**, and **system-sphinx**.
  - 3scale 2.3 includes **system-resque**.
- **system-app**.
- **backend-listener** and **backend-worker**.
- **backend-redis**, **system-memcache**, **system-mysql**, **system-redis**, and **zync-database**. The following example shows how to perform this in the CLI for **apicast-wildcard-router** and **zync**:

```
oc scale dc/apicast-wildcard-router --replicas=0
oc scale dc/zync --replicas=0
```

NOTE

The deployment configuration for each step can be scaled down at the same time. For example, you could scale down **apicast-wildcard-router** and **zync** together. However, it is better to wait for the pods from each step to terminate before scaling down the ones that follow. The 3scale instance will be completely inaccessible until it is fully started again.

3. To confirm that no pods are running on the 3scale project use the following command:

```
oc get pod
```

The command should return *No resources found*.

4. Scale up the database level pods again using the following command:

```
oc scale dc/{backend-redis,system-memcache,system-mysql,system-redis,zync-database} --replicas=1
```

5. Ensure that you are able to login to the external MySQL database through the **system-mysql** pod before proceeding with the next steps:

```
oc rsh system-mysql-<system_mysql_pod_id>
mysql -u root -p -h <host>
```

- *<system_mysql_pod_id>*: The identifier of the system-mysql pod.
- The user should always be root. For more information see External MySQL database limitations.

    a. The CLI will now display **mysql>**. Type *exit*, then press *return*. Type *exit* again at the next prompt to go back to the OpenShift node console.

6. Perform a full MySQL dump using the following command:

```
oc rsh system-mysql-<system_mysql_pod_id> /bin/bash -c "mysqldump -u root --single-transaction --routines --triggers --all-databases" > system-mysql-dump.sql
```

- Replace *<system_mysql_pod_id>* with your unique **system-mysql** pod *ID* .
- Validate that the file **system-mysql-dump.sql** contains a valid MySQL level dump as in the following example:

```
$ head -n 10 system-mysql-dump.sql
-- MySQL dump 10.13  Distrib 5.7.24, for Linux (x86_64)
--
-- Host: localhost    Database:
-- ------------------------------------------------------
-- Server version   5.7.24

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
```

```
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
*/;
/*!40101 SET NAMES utf8 */;
```

7. Scale down the **system-mysql** pod and leave it with 0 (zero) replicas:

```
oc scale dc/system-mysql --replicas=0
```

8. Find the *base64* equivalent of the URL **mysql2://root:<password>@<host>/system**, replacing *<password>* and *<host>* accordingly:

```
echo "mysql2://root:<password>@<host>/system" | base64
```

9. Create a default *'user'@'%'* on the remote MySQL database. It only needs to have SELECT privileges. Also find its *base64* equivalents:

```
echo "user" | base64
echo "<password>" | base64
```

- Replace <password> with the password for *'user'@'%'*.

10. Perform a backup and edit the OpenShift secret **system-database**:

```
oc get secret system-database -o yaml > system-database-orig.bkp.yml
oc edit secret system-database
```

- **URL**: Replace it with the value from [step-8].
- **DB_USER** and **DB_PASSWORD**: Use the values from the previous step for both.

11. Send **system-mysql-dump.sql** to the remote database server and import the dump into it. Use the command to import it:

12. Use the command below to send **system-mysql-dump.sql** to the remote database server and import the dump into the server:

```
mysql -u root -p < system-mysql-dump.sql
```

13. Ensure that a new database called *system* was created:

```
mysql -u root -p -se "SHOW DATABASES"
```

14. Use the following instructions to *Start 3scale On-premises*, which scales up all the pods in the correct order.

Start 3scale On-premises

- **backend-redis**, **system-memcache**, **system-mysql**, **system-redis**, and **zync-database**.
- **backend-listener** and **backend-worker**.
- **system-app**.
- **system-sidekiq**, **backend-cron**, and **system-sphinx**

    - 3scale 2.3 includes **system-resque**.

- **apicast-staging** and **apicast-production**.

- **apicast-wildcard-router** and **zync** for versions before 3scale 2.6 or **zync-que** and **zync** for 3scale 2.6 and above.
  The following example shows how to perform this in the CLI for **backend-redis**, **system-memcache**, **system-mysql**, **system-redis**, and **zync-database**:

```
oc scale dc/backend-redis --replicas=1
oc scale dc/system-memcache --replicas=1
oc scale dc/system-mysql --replicas=1
oc scale dc/system-redis --replicas=1
oc scale dc/zync-database --replicas=1
```

  The **system-app** pod should now be up and running without any issues.

15. After validation, scale back up the other pods in the order shown.

16. Backup the **system-mysql** *DeploymentConfig* object. You may delete after a few days once you are sure everything is running properly. Deleting **system-mysql** *DeploymentConfig* avoids any future confusion if this procedure is done again in the future.

8.3. ROLLING BACK

Perform a rollback procedure if the **system-app** pod is not fully back online and the root cause for it could not be determined or addressed after following step 14.

1. Edit the secret **system-database** using the original values from **system-database-orig.bkp.yml**. See [step-10]:

```
oc edit secret system-database
```

   Replace *URL*, *DB_USER*, and *DB_PASSWORD* with their original values.

2. Scale down all the pods and then scale them back up again, including **system-mysql**. The **system-app** pod and the other pods to be started after it should be up and running again. Run the following command to confirm all pods are back up and running:

```
oc get pods -n <3scale-project>
```

8.4. ADDITIONAL INFORMATION

- For more information about 3scale and MySQL database support, see Red Hat 3scale API Management Supported Configurations.

CHAPTER 9. SETTING UP YOUR 3SCALE SYSTEM IMAGE WITH AN ORACLE DATABASE

NOTE

- The Oracle Database is only supported with OpenShift Container Platform (OCP) 3.11 when you are performing a template-based installation of 3scale.
- If you are performing a 3scale deployment with the operator using the Oracle Database, see Installing 3scale with the operator using the Oracle Database .
- As of 3scale 2.10, Oracle Database 12c is no longer supported.
- For more information about supported configurations, see the Red Hat 3scale API Management Supported Configurations page.

This section explains how a Red Hat 3scale API Management administrator sets up the 3scale system image with an Oracle Database. By default, 3scale 2.11 has a component called system that stores configuration data in a MySQL database. You can override the default database and store your information in an external Oracle Database. Follow the steps in this chapter to build a custom system container image with your own Oracle Database client binaries and deploy 3scale to OpenShift.

Prerequisites

1. From the *Instant Client Downloads* page, download:
   - A client: It can be either *basic-lite* or *basic*.
   - The *ODBC driver*.
   - The *SDK* for Oracle Database 19c.
     - For 3scale, use Instant Client Downloads for Linux x86-64 (64-bit)
     - For ppc64le and 3scale, use Oracle Instant Client Downloads for Linux on Power Little Endian (64-bit)
2. Check the Red Hat 3scale API Management Supported Configurations for the following Oracle software components:
   - Oracle Instant Client Package: Basic or Basic Light
   - Oracle Instant Client Package: SDK
   - Oracle Instant Client Package: ODBC

Table 9.1. Oracle 19c example packages for 3scale

Oracle 19c package name	Compressed file name
Basic	instantclient-basic-linux.x64-19.8.0.0.0dbru.zip
Basic Light	instantclient-basiclite-linux.x64-19.8.0.0.0dbru.zip

Oracle 19c package name	Compressed file name
SDK	instantclient-sdk-linux.x64-19.8.0.0.0dbru.zip
ODBC	instantclient-odbc-linux.x64-19.8.0.0.0dbru.zip

Table 9.2. Oracle 19c example packages for ppc64le and 3scale

Oracle 19c package name	Compressed file name
Basic	instantclient-basic-linux.leppc64.c64-19.3.0.0.0dbru.zip
Basic Light	instantclient-basiclite-linux.leppc64.c64-19.3.0.0.0dbru.zip
SDK	instantclient-sdk-linux.leppc64.c64-19.3.0.0.0dbru.zip
ODBC	instantclient-odbc-linux.leppc64.c64-19.3.0.0.0dbru.zip

To set up your 3scale system image with and Oracle Database, perform the steps outlined in the following sections:

- Section 9.1, “Preparing the Oracle Database”
- Section 9.2, “Building the system image”

9.1. PREPARING THE ORACLE DATABASE

This section provides steps for preparing your Oracle Database.

Prerequisites

- A supported version of the Oracle Database accessible from your OpenShift cluster.
- Access to the Oracle Database **system** user for installation procedures.

Procedure

1. Create a new database.
   To configure 3scale with Oracle Database, use the following settings:

   ```
   ALTER SYSTEM SET max_string_size=extended SCOPE=SPFILE;
   ```

2. Collect the database details.
   You need the following information for 3scale configuration:

- The Oracle Database URL address.
- The Oracle Database. service name
- The Oracle Database **system** password.
  The **DATABASE_URL** parameter must follow this format: **oracle-enhanced://${user}:${password}@${host}:${port}/${database}**

Example

```
DATABASE_URL="oracle-enhanced://user:password@my-oracle-database.com:1521/threescalepdb"
```

Additional resources

- For information on creating a new database in Oracle Database, see the Oracle documentation.

9.2. BUILDING THE SYSTEM IMAGE

This section provides steps to build the system image.

Prerequisites

- You should have already carried out the steps in 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. Download the 3scale 2.11 *amp.yml* template.
4. Run the **oc new-app** command with the **-f** option and specify the **build.yml** OpenShift template:

   ```
   $ oc new-app -f build.yml
   ```

5. 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
   ```

6. 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"}]}}}}}}'
```

7. 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"}}'
```

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

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

9. 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.

9.2.1. Updating ImageChange triggers

Update the ImageChange triggers of the DeploymentConfigs that use the System image so they use the new Oracle-based System image.

Prerequisites

- First carry out the steps in Building the system image.

Procedure

1. Save the current 3scale release in an environment variable:

```
$ export THREESCALE_RELEASE=2.11
```

2. Update the **system-app** ImageChange trigger:

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

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

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

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

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

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

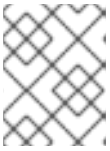
This triggers a redeployment of the **system-sidekiq** DeploymentConfig. Wait until it is redeployed, its corresponding new pods are ready, and the old ones have stopped.

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

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

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

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



NOTE

The following step is optional. Use it to remove **ORACLE_SYSTEM_PASSWORD** after the installation of 3scale.

5. 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.