Installing Red Hat 3scale API Management

Install and configure 3scale API Management.
Install and configure 3scale API Management.
Abstract
This guide provides the information to install and configure 3scale API Management.
# Table of Contents

**PREFACE** ................................................................. 5

**MAKING OPEN SOURCE MORE INCLUSIVE** ........................................ 6

**CHAPTER 1. REGISTRY SERVICE ACCOUNTS FOR 3SCALE** ........................................... 7
  1.1. CREATING A REGISTRY SERVICE ACCOUNT ........................................ 7
  1.2. CONFIGURING CONTAINER REGISTRY AUTHENTICATION ............................. 7
  1.3. MODIFYING A REGISTRY SERVICE ACCOUNT ........................................... 8
  1.4. ADDITIONAL RESOURCES .................................................................... 9

**CHAPTER 2. INSTALLING 3SCALE ON OPENSIFT** .................................................. 10
  2.1. SYSTEM REQUIREMENTS FOR INSTALLING 3SCALE ON OPENSIFT ................. 10
    2.1.1. Environment requirements ............................................................... 10
    2.1.2. Hardware requirements .................................................................. 11
  2.2. INSTALLING THE 3SCALE OPERATOR ON OPENSIFT .................................. 12
    2.2.1. Creating a new OpenShift project ..................................................... 12
    2.2.2. Installing and configuring the 3scale operator using the OLM .................. 13
    2.2.2.1. Restrictions in disconnected environments ...................................... 14
    2.2.3. Upgrading the 3scale operator using the OLM ................................. 15
    2.2.3.1. Configuring automated application of micro releases ......................... 16
  2.3. INSTALLING THE APICAST OPERATOR ON OPENSIFT ............................... 17
  2.4. DEPLOYING 3SCALE USING THE OPERATOR .............................................. 17
    2.4.1. Deploying the APIManager custom resource ........................................ 18
    2.4.2. Getting the Admin Portal URL .......................................................... 19
    2.4.3. Getting the APIManager Admin Portal and Master Admin Portal credentials .................................................. 20
    2.4.4. External databases for 3scale using the operator .................................. 20
  2.5. DEPLOYMENT CONFIGURATION OPTIONS FOR 3SCALE ON OPENSIFT USING THE OPERATOR .......................................................... 21
    2.5.1. Configuring proxy parameters for embedded APIcast ........................... 21
    2.5.2. Injecting custom environments with the 3scale operator ....................... 24
    2.5.3. Injecting custom policies with the 3scale operator ............................... 25
    2.5.4. Configuring OpenTracing with the 3scale operator ............................... 27
    2.5.5. Enabling TLS at the pod level with the 3scale operator ......................... 29
    2.5.6. Proof of concept for evaluation deployment ........................................ 30
    2.5.6.1. Default deployment configuration .................................................. 30
    2.5.6.2. Evaluation installation .................................................................. 31
    2.5.7. External databases installation ............................................................. 31
    2.5.7.1. Backend Redis secret .................................................................... 32
    2.5.7.2. System Redis secret ..................................................................... 33
    2.5.7.3. System database secret ................................................................. 33
    2.5.7.4. Zync database secret .................................................................... 34
    2.5.7.5. APIManager custom resources to deploy 3scale ................................. 35
    2.5.8. Enabling pod affinity in the 3scale operator ......................................... 35
    2.5.8.1. Customizing node affinity and tolerations at component level .............. 36
    2.5.9. Multiple clusters in multiple availability zones ..................................... 37
    2.5.9.1. Prerequisites for multiple clusters installations .................................... 38
    2.5.9.2. Active-passive clusters on the same region with shared databases ......... 38
    2.5.9.3. Configuring and installing shared databases ...................................... 39
    2.5.9.4. Manual failover shared databases ................................................. 40
    2.5.9.5. Active-passive clusters on different regions with synced databases ....... 41
    2.5.9.6. Configuring and installing synced databases ..................................... 41
    2.5.9.7. Manual failover synced databases .................................................. 42
    2.5.10. Amazon Simple Storage Service 3scale fileStorage installation ............ 42

---

3scale
2.5.10.1. Amazon S3 bucket creation
2.5.10.2. Create an OpenShift secret
2.5.10.3. Manual mode with STS
2.5.11. PostgreSQL installation
2.5.12. Configuring SMTP variables (optional)
2.5.13. Customizing compute resource requirements at component level
  2.5.13.1. Default APIManager components compute resources
  2.5.13.1.1. CPU and memory units
2.5.14. Customizing node affinity and tolerations at component level
2.5.15. Pod priority of 3scale components
2.5.16. Setting backend client to skip certificate verification
2.5.17. Setting custom annotations
2.5.18. Reconciliation
  2.5.18.1. Resources
  2.5.18.2. Backend replicas
  2.5.18.3. APIcast replicas
  2.5.18.4. System replicas
  2.5.18.5. Zync replicas
2.6. INSTALLING 3SCALE WITH THE OPERATOR USING ORACLE AS THE SYSTEM DATABASE
  2.6.1. Preparing the Oracle Database
  2.6.2. Building a custom system container image
  2.6.3. Installing 3scale with Oracle using the operator
2.7. TROUBLESHOOTING COMMON 3SCALE INSTALLATION ISSUES
  2.7.1. Previous deployment leaving dirty persistent volume claims
  2.7.2. Wrong or missing credentials of the authenticated image registry
  2.7.3. Incorrectly pulling from the Docker registry
  2.7.4. Permission issues for MySQL when persistent volumes are mounted locally
  2.7.5. Unable to upload logo or images
  2.7.6. Test calls not working on OpenShift
  2.7.7. APIcast on a different project from 3scale failing to deploy
2.8. ADDITIONAL RESOURCES

CHAPTER 3. INSTALLING APICAST

3.1. APICAST DEPLOYMENT OPTIONS
3.2. APICAST ENVIRONMENTS
3.3. CONFIGURING THE INTEGRATION SETTINGS
3.4. CONFIGURING YOUR PRODUCT
  3.4.1. Declaring the API backend
  3.4.2. Configuring the authentication settings
  3.4.3. Configuring the API test call
3.5. DEPLOYING APICAST ON THE DOCKER CONTAINERIZED ENVIRONMENT
  3.5.1. Installing the Docker containerized environment
  3.5.2. Running the Docker containerized environment gateway
    3.5.2.1. The docker command options
    3.5.2.2. Testing APIcast
  3.5.3. Additional resources
  3.5.4. Deploying APIcast on Podman
    3.5.4.1. Installing the Podman container environment
    3.5.4.2. Running the Podman environment
      3.5.4.2.1. Testing APIcast with Podman
    3.5.4.3. The podman command options
    3.5.4.4. Additional resources
3.6. DEPLOYING AN APICAST GATEWAY SELF-MANAGED SOLUTION USING THE OPERATOR
3.6.1. APICast deployment and configuration options
   3.6.1.1. Providing a 3scale system endpoint
     3.6.1.1.1. Verifying the APICast gateway is running and available
     3.6.1.1.2. Exposing APICast externally via a Kubernetes Ingress
   3.6.1.2. Providing a configuration secret
     3.6.1.2.1. Verifying APICast gateway is running and available
   3.6.1.3. Injecting custom environments with the APICast operator
   3.6.1.4. Injecting custom policies with the APICast operator
   3.6.1.5. Configuring OpenTracing with the APICast operator

3.7. ADDITIONAL RESOURCES

CHAPTER 4. EXTERNAL REDIS DATABASE CONFIGURATION FOR HIGH AVAILABILITY SUPPORT IN 3SCALE

4.1. SETTING UP REDIS FOR ZERO DOWNTIME
4.2. CONFIGURING BACK-END COMPONENTS FOR 3SCALE
   4.2.1. Creating backend-redis and system-redis secrets
   4.2.2. Deploying a fresh installation of 3scale for HA
   4.2.3. Migrating a non-HA deployment of 3scale to HA
      4.2.3.1. Using Redis Enterprise
      4.2.3.2. Using Redis Sentinel
4.3. REDIS DATABASE SHARDING AND REPLICATION
4.4. ADDITIONAL INFORMATION

CHAPTER 5. CONFIGURING AN EXTERNAL MYSQL DATABASE

5.1. EXTERNAL MYSQL DATABASE LIMITATIONS
5.2. EXTERNALIZING THE MYSQL DATABASE
5.3. ROLLING BACK
5.4. ADDITIONAL INFORMATION
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 Red Hat 3scale API Management 2.14, you must use a Registry Service Account instead of an individual user’s Customer Portal credentials.

IMPORTANT

Before deploying 3scale on OpenShift via the operator, follow the steps outlined in this chapter, as the deployment uses registry authentication.

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

- Creating a registry service account
- Configuring container registry authentication
- 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. CONFIGURING CONTAINER REGISTRY AUTHENTICATION
As a 3scale administrator, configure authentication with registry.redhat.io before you deploy 3scale on OpenShift.

**Prerequisites**

- A Red Hat OpenShift Container Platform (OCP) account with administrator credentials.
- OpenShift oc client tool is installed. For more details, see the OpenShift CLI documentation.

**Procedure**

1. Log into your OpenShift cluster as administrator:
   ```bash
   $ oc login -u <admin_username>
   ```

2. Open the project in which you want to deploy 3scale:
   ```bash
   $ 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:
   ```bash
   $ 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"
   
   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:
   ```bash
   $ 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:
   ```bash
   $ 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

### 1.3. 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.

### 1.4. ADDITIONAL RESOURCES

- Red Hat Container Registry Authentication
- Authentication enabled Red Hat registry
CHAPTER 2. INSTALLING 3SCALE ON OPENSHEET

This section walks you through steps to deploy Red Hat 3scale API Management 2.14 on OpenShift.

The 3scale solution for on-premises deployment is composed of:

- Two application programming interface (API) gateways: embedded APIcast
- One 3scale Admin Portal and Developer Portal with persistent storage

**NOTE**

- When deploying 3scale, 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 3scale. 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 Creating a registry service account.

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

- System requirements for installing 3scale on OpenShift
- Deploying 3scale using the operator
- External databases for 3scale using the operator
- Deployment configuration options for 3scale on OpenShift using the operator
- Installing 3scale with the operator using Oracle as the system database
- Troubleshooting common 3scale installation issues

2.1. SYSTEM REQUIREMENTS FOR INSTALLING 3SCALE ON OPENSHEET

This section lists the system requirements for installing 3scale on OpenShift.

2.1.1. Environment requirements

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

The requirements for persistent volumes vary between different deployment types. When deploying with external databases, persistent volumes are not necessary. For some deployment types, an Amazon S3 bucket can serve as a substitute for persistent volumes. If you use local file system storage, consider the specific deployment type and its associated requirements for persistent volumes.

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.

NOTE

Network File System (NFS) is supported on 3scale for the RWX volume only.

For IBM Power (ppc64le) and IBM Z (s390x), provision local storage using the following:

Storage

- NFS

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
- 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.

**Additional resources**

- Understanding persistent storage

### 2.2. INSTALLING THE 3SCALE OPERATOR ON OPENSSHIFT

**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 2.2.1, “Creating a new OpenShift project”
- Section 2.2.2, “Installing and configuring the 3scale operator using the OLM”

#### 2.2.1. 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.

2.2.2. 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) 4.6 cluster through the OperatorHub in the OCP console. You can install the 3scale operator using the following installation modes:

- Cluster-wide in which the operator is available in all namespaces on the cluster.
- A specific namespace on the cluster

**NOTE**

If you are using the OpenShift Container Platform on a restricted network or a disconnected cluster, the Operator Lifecycle Manager 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.

   **NOTE**

   The menu structure depends on the version of OpenShift you are using:

2. Click *Operators > OperatorHub*

3. In the *Filter by keyword* box, type *3scale operator* to find *Red Hat Integration - 3scale*.

4. Click *Red Hat Integration - 3scale*. Information about the operator is displayed.

5. Read the information about the operator and click *Install*. The *Install Operator* page opens.

6. On the *Install Operator* page, select the desired channel to update in the *Update channel* section.
7. In the *Installation mode* section, select where to install the operator.
   
a. **All namespaces on the cluster (default)** - The operator will be available in all namespaces on the cluster.

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

8. Click **Install**.

9. After the installation is complete, the system displays a confirmation message indicating that the operator is ready for use.

10. Verify that the 3scale operator ClusterServiceVersion (CSV) is correctly installed. Also check if it reports that the installation of the operator has been successful:

    - Click **Operators > Installed Operators**
    - Click on the **Red Hat Integration - 3scale** operator.
    - In the **Details** tab, scroll down to the **Conditions** section, where the **Succeeded** condition should read **InstallSucceeded** under the **Reason** column.

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.

### 2.2.2.1. Restrictions in disconnected environments

The following list outlines current restrictions in a disconnected environment for 3scale 2.14:

- 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](https://docs.openshift.com).
For more information about using the OLM on restricted networks, see Using Operator Lifecycle Manager on restricted networks.

For more information about preparing your installation on restricted networks, see Mirroring images for a disconnected installation.

For more information about supported configurations, see the Red Hat 3scale API Management Supported Configurations page.

2.2.3. Upgrading the 3scale operator using the OLM

To upgrade the 3scale operator from a single namespace to a cluster-wide installation in all namespaces on an operator-based deployment, you must remove the 3scale operator from the namespace and then reinstall the operator on the cluster.

Cluster administrators can delete installed operators from a selected namespace by using the web console. Uninstalling the operator does not uninstall an existing 3scale instance.

After the 3scale operator is uninstalled from the namespace, you can use OLM to install the operator in the cluster-wide mode.

Prerequisites

- 3scale administrator permissions or an OpenShift role that has delete permissions for the namespace.

Procedure

1. In the OpenShift Container Platform console, log in using an account with administrator privileges.

   NOTE
   The menu structure depends on the version of OpenShift you are using:

2. Click **Operators > OperatorHub**. The installed Operators page is displayed.

3. Enter **3scale** into the **Filter by name** to find the Operator and click on it.

4. On the **Operator Details** page, select **Uninstall Operator** from the **Actions** drop-down menu to remove it from a specific namespace.

5. An **Uninstall Operator?** dialog box is displayed, reminding you that:

   Removing the operator will not remove any of its custom resource definitions or managed resources. If your operator has deployed applications on the cluster or configured off-cluster resources, these will continue to run and need to be cleaned up manually. This action removes the operator as well as the Operator deployments and pods, if any. Any operands and resources managed by the operator, including CRDs and CRs, are not removed. The web console enables dashboards and navigation items for some operators. To remove these after uninstalling the operator, you might need to manually delete the operator CRDs.

6. Select **Uninstall**. This operator stops running and no longer receives updates.
7. In the OpenShift Container Platform console click **Operators > OperatorHub**.

8. In the **Filter by keyword** box, type **3scale operator** to find **Red Hat Integration - 3scale**.

9. Click **Red Hat Integration - 3scale**. Information about the operator is displayed.

10. Click **Install**. The **Install Operator** page opens.

11. On the **Install Operator** page, select the desired channel to update in the **Update channel** section.

12. In the **Installation mode** section, select **All namespaces on the cluster (default)** The operator will be available in all namespaces on the cluster.

13. Click **Subscribe**. The **3scale operator** details page is displayed and you can see the **Subscription Overview**.

14. Confirm that the subscription **Upgrade Status** is displayed as **Up to date**.

15. Verify that the 3scale operator ClusterServiceVersion (CSV) is displayed.

**Additional Resources**

- For more information about installing the 3scale operator, see **Install and configure 3scale API Management**.

---

### 2.2.3.1. Configuring automated application of micro releases

The 3scale operator’s approval strategy must be set to **Automatic** to obtain micro release updates and have them be applied automatically. 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 at 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**
2.3. INSTALLING THE APICAST OPERATOR ON OPENSIFT

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

Prerequisites

- OCP 4.x or later with administrator privileges.

Procedure

1. Create new project **operator-test** in **Projects > Create Project**

2. Click **Operators > OperatorHub**

3. In the **Filter by keyword** box, type **apicast operator** to find **Red Hat Integration - 3scale APIcast gateway**.

4. Click **Red Hat Integration - 3scale APIcast gateway**. Information about the APIcast operator is displayed.

5. Click **Install**. The **Create Operator Subscription** page opens.

6. Click **Install** to accept all of the default selections on the **Create Operator Subscription** page.

   **NOTE**

   You can select different operator versions and installation modes, such as cluster-wide or namespace-specific options. There can only be one cluster-wide installation per cluster.

   a. The subscription upgrade status is shown as **Up to date**.

7. Click **Operators > Installed Operators** to verify that the APIcast operator **ClusterServiceVersion** (CSV) status displays to **InstallSucceeded** in the **operator-test** project.

2.4. DEPLOYING 3SCALE USING THE OPERATOR

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

**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.
  • 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 Admin Portal URL

• Getting the APIManager Admin Portal and Master Admin Portal credentials

• External databases for 3scale using the operator

2.4.1. Deploying the APIManager custom resource

NOTE

If you decide to use Amazon Simple Storage Service (Amazon S3), see Amazon Simple Storage Service 3scale FileStorage installation.

Deploying the APIManager CR 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 Red Hat Integration - 3scale.

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 value of the `wildcardDomain` parameter must be a valid domain name that resolves to the address of your OpenShift Container Platform (OCP) router. For example, `apps.mycluster.example.com`.

- APIManager CR with minimum requirements:

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

- APIManager CR with replicas configured:

```yaml
apiVersion: apps.3scale.net/v1alpha1
cr?: 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
      apiicast:
        productionSpec:
          replicas: 1
        stagingSpec:
          replicas: 1
      wildcardDomain: example.com
```

### 2.4.2. 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 the installation. Open this unique URL in a browser using the this command:

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

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

### 2.4.3. 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:

   ```bash
   $ 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:

   ```bash
   $ 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](#).

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

### 2.4.4. External databases for 3scale using the operator

**IMPORTANT**

When you externalize databases from a Red Hat 3scale API Management deployment, this means to provide isolation from the application and resilience against service disruptions at the database level. The resilience to service disruptions depends on the service level agreements (SLAs) provided by the infrastructure or platform provider where you host the databases. This is not offered by 3scale. For more details on externalizing of databases offered by your chosen deployment, see the associated documentation.

When you use an external databases for 3scale using the operator, the aim is to provide uninterrupted uptime if, for example, one or more databases were to fail.

If you use external databases in your 3scale operator-based deployment, note the following:

- Configure and deploy 3scale critical databases externally. Critical databases include the system database, system redis, and backend redis components. Ensure that you deploy and configure these components in a way that makes them highly available.
Specify the connection endpoints to those components for 3scale by creating their corresponding Kubernetes secrets before deploying 3scale.

- See External databases installation for more information.
- See Enabling Pod Disruption Budgets for more information about non-database deployment configurations.

In the APIManager CR, set the .spec.externalComponents attribute to specify that system database, system redis, and backend redis are external:

```yaml
externalComponents:
  backend:
    redis: true
  system:
    database: true
    redis: true
  zync:
    database: true
```

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 creating its corresponding Kubernetes secret before deploying 3scale.
  - See Zync database secret.
- Configure 3scale by setting the .spec.externalComponents.zync.database attribute in the APIManager CR to true to specify that the zync database is an external database.

## 2.5. 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.

### 2.5.1. 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 (CR). In other words, you are using the 3scale operator, an APIManager CR
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:

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

```yaml
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:

```yaml
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:

```yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
```
2.5.2. 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.

```lua
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 },
}
```

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. 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 (CR) 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
description: APIManager
description: metadata:
  name: apimanager-apicast-custom-environment
description: spec:
  wildcardDomain: <desired-domain>
description: apicast:
    productionSpec:
      customEnvironments:
      - secretRef:
        name: custom-env-1
description: stagingSpec:
  customEnvironments:
  - secretRef:
    name: custom-env-1
```

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

4. Create the APIManager CR 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 (CR) 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 redeploy APIcast again by setting `spec.apicast.productionSpec.replicas` or `spec.apicast.stagingSpec.replicas` back to its previous value.

**2.5.3. 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 (CR)

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 CR. 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:

```bash
$ 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. Use the 3scale operator to monitor secret changes. Add the apimanager.apps.3scale.net/watched-by=apimanager label to begin the 3scale operator secret changes monitoring:

```bash
$ oc label secret my-first-custom-policy-secret apimanager.apps.3scale.net/watched-by=apimanager
```

**NOTE**

By default, changes to the secret are not tracked by the 3scale operator. With the label in place, the 3scale operator automatically updates the APImcast deployment whenever you make changes to the secret. This happens in both staging and production environments where the secret is in use. The 3scale operator will not take ownership of the secret in any way.

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

```yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
```
An APIManager CR can reference multiple secrets that define different custom policies. The operator loads each custom policy.

4. Create the APIManager CR that references the secrets that contain the custom policies. For example:

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

Next steps

When you apply the `apimanager.apps.3scale.net/watched-by=apimanager` label, the 3scale operator begins monitoring changes in the secret. Now, you can modify the custom policy within the secret, and the operator will initiate a rolling update, loading 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 redeploy APIcast again by setting `spec.apicast.productionSpec.replicas` or `spec.apicast.stagingSpec.replicas` back to its previous value.

### 2.5.4. 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.

#### 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 following example shows a valid secret definition:

```yaml
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 -f myjaeger.yaml
```

3. Define an `APIManager` custom resource (CR) 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.

```yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager1
spec:
  apicast:
    stagingSpec:
```
Create the `APIManager` CR that configures OpenTracing. For example, if you saved the `APIManager` CR 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

**2.5.5. 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` custom resource (CR). You create the secret and then reference the name of the secret in the `APIManager` CR 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`, 3scale 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 definitions:

      ```yaml
      spec:
        apicast:
          productionSpec:
            httpsPort: 8443
            httpsVerifyDepth: 1
            httpsCertificateSecretRef:
              name: mycertsecret
      ```
   b. If enabling for **staging**, configure the following YAML definitions:

      ```yaml
      spec:
        apicast:
          stagingSpec:
            httpsPort: 8443
            httpsVerifyDepth: 1
            httpsCertificateSecretRef:
              name: mycertsecret
      ```
7. Click **Create**.

### 2.5.6. 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.

### 2.5.6.1. 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 overridden with specific field values in the APIManager custom resource (CR). 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.

### 2.5.6.2. Evaluation installation

For an evaluation installation, 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

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

Additional resources

- APIManager

### 2.5.7. External databases installation
IMPORTANT

When you externalize databases from a Red Hat 3scale API Management deployment, this means to provide isolation from the application and resilience against service disruptions at the database level. The resilience to service disruptions depends on the service level agreements (SLAs) provided by the infrastructure or platform provider where you host the databases. This is not offered by 3scale. For more details on externalizing of databases offered by your chosen deployment, see the associated documentation.

An external databases installation is suitable for production where you want to provide uninterrupted uptime or where you plan to reuse your own databases.

IMPORTANT

When enabling the 3scale external databases installation mode, you can configure one or more of the following databases as external to 3scale:

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

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

<table>
<thead>
<tr>
<th>Database</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis</td>
<td>5.0</td>
</tr>
<tr>
<td>MySQL</td>
<td>8.0</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>13</td>
</tr>
</tbody>
</table>

Before creating an APIManager CR to deploy 3scale, you must provide the following connection settings for the external databases by using OpenShift secrets.

Additional resources

- Red Hat 3scale API Management Supported Configurations

2.5.7.1. Backend Redis secret

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

```yaml
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"

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

### 2.5.7.2. System Redis secret

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

```yaml
apiVersion: v1
class: 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: ""

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

### 2.5.7.3. 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**

```yaml
apiVersion: v1
class: Secret
metadata:
  name: system-database
stringData:
  URL: "mysql://system-database"
  DATABASE: "example"
  NAMESPACE: ""

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

If you use MySQL 8.0 with 3scale 2.12, you must set the authentication plugin to `mysql_native_password`. Add the following to the MySQL configuration file:

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

PostgreSQL system database secret

```yaml
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

```yaml
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

- `{DB_USER}` and `{DB_PASSWORD}` are the username and password of the regular non-system user.
- `{DB_NAME}` is the Oracle Database service name.
- `ORACLE_SYSTEM_PASSWORD` is optional, see Configure a database user.

2.5.7.4. Zync database secret

In a zync database setup, when the `spec.externalComponents.zync.database` field is set to true, you must create a secret named `zync` before you deploy 3scale. In this secret, set the `DATABASE_URL` and `DATABASE_PASSWORD` fields to the values that point to your external zync database, for example:

```yaml
apiVersion: v1
kind: Secret
```
The zync database must be in high-availability mode.

### 2.5.7.5. APIManager custom resources to deploy 3scale

**NOTE**

- When you enable external components, you must create a secret for each external component (backend-redis, system-redis, system-database, zync) before you deploy 3scale.

- For an external system-database, choose only one type of database to externalize.

Configuration of the APIManager custom resource (CR) depends on whether or not your choice of database is external to your 3scale deployment.

If backend-redis, system-redis, or system-database is external to 3scale, populate the APIManager CR **externalComponents** object as shown in the following example:

```yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  externalComponents:
    system:
      database: true
```

Additional resources

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

### 2.5.8. Enabling pod affinity in the 3scale operator

You can enable pod affinities in the 3scale operator for every component. This ensures distribution of pod replicas from each `deploymentConfig` across different nodes of the cluster, so they will be evenly balanced across different availability zones (AZ).
2.5.8.1. Customizing node affinity and tolerations at component level

Customize Kubernetes affinity and tolerations in your 3scale solution through the `APIManager` CR attributes. You can then customize to schedule different 3scale components onto Kubernetes nodes.

For example, to set a custom node affinity for `backend-listener` and custom tolerations for `system-memcached`, do the following:

Custom affinity and tolerations

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

  system:
    memcachedTolerations:
      - key: key1
        value: value1
        operator: Equal
        effect: NoSchedule
      - key: key2
        value: value2
        operator: Equal
        effect: NoSchedule

affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
```

Add the following affinity block to `apicastProductionSpec` or to any non-database `deploymentConfig`. This adds a soft `podAntiAffinity` configuration using `preferredDuringSchedulingIgnoredDuringExecution`. The scheduler will try to run this set of `apicast-production` pods in different hosts from different AZs. If it is not possible, then allow them to run elsewhere:

Soft podAntiAffinity

```yaml
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
```
In the following example, a hard podAntiAffinity configuration is set using requiredDuringSchedulingIgnoredDuringExecution. Conditions must be met to schedule a pod onto a node. A risk exists, for example, that you will not be able to schedule new pods on a cluster with low free resources:

**Hard podAntiAffinity**

```yaml
affinity:
podAntiAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    podAffinityTerm:
      labelSelector:
        matchLabels:
        deploymentConfig: apicast-production
        topologyKey: topology.kubernetes.io/zone

- weight: 99
  podAffinityTerm:
    labelSelector:
      matchLabels:
      deploymentConfig: apicast-production
      topologyKey: kubernetes.io/hostname
```

In case of failure, bringing a passive cluster into active mode disrupts the provision of the service until the procedure finishes. Due to this disruption, be sure to have a maintenance window.

This documentation focuses on deployment using Amazon Web Services (AWS). The same configuration options apply to other public cloud vendors where the provider’s managed database services offer, for example, support for multiple availability zones and multiple regions.

When you want to install 3scale on several OpenShift clusters and high availability (HA) zones, there are options available which you can refer to here.
In multiple cluster installation options, clusters work in an active/passive configuration, with the failover procedure involving a few manual steps.

2.5.9.1. Prerequisites for multiple clusters installations

Use the following in 3scale installations that involve using several OpenShift clusters:

- Use pod affinities with both `kubernetes.io/hostname` and `topology.kubernetes.io/zone` rules in the `APIManager` custom resource (CR).
- Use pod disruption budgets in the `APIManager` CR.
- A 3scale installation over multiple clusters must use the same shared `wildcardDomain` attribute specifications in the `APIManager` CR. The use of a different domain for each cluster is not allowed in this installation mode, as the information stored in the database would be conflicting.
- You must manually deploy the secrets containing credentials, such as tokens and passwords, in all clusters with the same values. The 3scale operator creates them with secure random values on every cluster. In this case, you need to have the same credentials in both clusters. You will find the list of secrets and how to configure them in the 3scale operator documentation. The following is the list of secrets you must mirror in both clusters:
  - `backend-internal-api`
  - `system-app`
  - `system-events-hook`
  - `system-master-apicast`
  - `system-seed`
    - You must manually deploy secrets with the database connection strings for `backend-redis`, `system-redis`, `system-database` and `zync`. See [External databases installation](#).
  - Databases shared among clusters must use the same values on all clusters.
  - If each cluster have their own databases, they must use different values on each cluster.

2.5.9.2. Active-passive clusters on the same region with shared databases

This setup consists of having two or more clusters in the same region and deploying 3scale in active-passive mode. One cluster is active, receiving traffic. The others are in standby mode without receiving traffic, therefore passive, but prepared to assume the active role in case there is a failure in the active cluster.

In this installation option, only a single region is in use and databases will be shared among all clusters.
### 2.5.9.3. Configuring and installing shared databases

**Procedure**

1. Create two or more OpenShift clusters in the same region using different availability zones (AZs). A minimum of three zones is recommended.

2. Create all required AWS ElastiCache (EC) instances with Amazon Relational Database Service (RDS) Multi-AZ enabled:
   a. One AWS EC for Backend Redis database
   b. One AWS EC for System Redis database

3. Create all required AWS RDS instances with Amazon RDS Multi-AZ enabled:
   a. One AWS RDS for the System database
   b. One AWS RDS for Zync database

4. Configure a AWS S3 bucket for the system assets.

5. Create a custom domain in AWS Route53 or your DNS provider and point it to the OpenShift router of the active cluster. This must coincide with the `wildcardDomain` attribute from `APIManager` custom resource (CR).
6. Install 3scale in the passive cluster. The `APIManager` CR should be identical to the one used in the previous step. When all pods are running, change the `APIManager` to deploy 0 replicas for all the `backend`, `system`, `zync` and `APIcast` pods.

   a. Set replicas to 0 to avoid consuming jobs from active database. Deployment will fail due to pod dependencies if each replica is set to 0 at first. For example, pods checking that others are running. First deploy as normal, then set replicas to 0 as shown in the `APIManager` spec example:

   ```yaml
   spec:
     apicast:
       stagingSpec:
         replicas: 0
       productionSpec:
         replicas: 0
     backend:
       listenerSpec:
         replicas: 0
       workerSpec:
         replicas: 0
       cronSpec:
         replicas: 0
     zync:
       appSpec:
         replicas: 0
       queSpec:
         replicas: 0
     system:
       appSpec:
         replicas: 0
       sidekiqSpec:
         replicas: 0
   ```

2.5.9.4. Manual failover shared databases

**Procedure**

1. In the active cluster, scale down the replicas of the `backend`, `system`, `zync`, and `APIcast` pods to 0.
   a. This becomes the new passive cluster, so you ensure that the new passive cluster will not consume jobs from active databases. Downtime starts here.

2. In the passive cluster, edit the `APIManager` to scale up the replicas of the `backend`, `system`, `zync`, and `APIcast` pods that were set to 0, so it will become the active cluster.

3. In the new active cluster, recreate the OpenShift routes created by `zync`.
   a. Run the `zync:resync:domains` command from the `system-master` container of the `system-app` pod:

   ```bash
   bundle exec rake zync:resync:domains
   ```

4. Point the custom domain created in AWS Route53 to the OpenShift router of the new active cluster.
2.5.9.5. Active-passive clusters on different regions with synced databases

This setup consists of having two or more clusters in different regions and deploying 3scale in active-passive mode. One cluster is active, receiving traffic, the others are in standby mode without receiving traffic, therefore passive, but prepared to assume the active role in case there is a failure in the active cluster.

To ensure good database access latency, each cluster has its own database instances. The databases from the active 3scale installation are replicated to the read-replica databases of the 3scale passive installations so the data is available and up to date in all regions for a possible failover.

2.5.9.6. Configuring and installing synced databases

Procedure

1. Create two or more OpenShift clusters in different regions using different availability zones. A minimum of three zones is recommended.

2. Create all required AWS ElastiCache instances with Amazon RDS Multi-AZ enabled on every region:
2. Create all required AWS RDS instances with Amazon RDS Multi-AZ enabled on every region:
   a. Two AWS RDS for the System database.
   b. Two AWS RDS for Zync database.
   c. Use cross-region replication, so the databases from passive regions are read-replicas from
      the master databases at the active region.

3. Configure an AWS S3 bucket for the system assets on every region using cross-region replication.

4. Create a custom domain in AWS Route53 or your DNS provider and point it to the OpenShift
   Router of the active cluster. This must coincide with the wildcardDomain attribute from
   APIManager CR.

5. Install 3scale in the passive cluster. The APIManager CR should be identical to the one used in
   the previous step. When all pods are running, change the APIManager to deploy 0 replicas for all
   the backend, system, zync, and APICAST pods:
   a. Set replicas to 0 to avoid consuming jobs from active database. Deployment will fail due to
      pod dependencies if each replica is set to 0 at first. For example, pods checking that others
      are running. First deploy as normal, then set replicas to 0.

2.5.9.7. Manual failover synced databases

Procedure

1. Do steps 1, 2 and 3 from Manual Failover shared databases.
   a. Every cluster has its own independent databases: read-replicas from the master at the
      active region.
   b. You must manually execute a failover on every database to select the new master on the
      passive region, which then becomes the active region.

2. Manual failovers of the databases to execute are:
   a. AWS RDS: System and Zync.
   b. AWS ElastiCaches: Backend and System.


2.5.10. Amazon Simple Storage Service 3scale fileStorage installation

Before creating APIManager custom resource (CR) to deploy 3scale, provide connection settings for
the Amazon Simple Storage Service (Amazon S3) service by using an OpenShift secret.
2.5.10.1. Amazon S3 bucket creation

Prerequisites

- You must have an Amazon Web Services (AWS) account.

Procedure

1. Create a bucket for storing the system assets.

2. Disable the public access blocker of S3 when using the Logo feature of the Developer Portal.

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

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "s3:ListAllMyBuckets",
         "Resource": "arn:aws:s3:::*"
       },
       {
         "Effect": "Allow",
         "Action": "s3:*",
         "Resource": ["arn:aws:s3::<target_bucket_name>", 1 "arn:aws:s3::<target_bucket_name>/" 2 ]
       }
     ]
   }
   ``

   1 Replace `<target_bucket_name>` with your own value.

   2 Replace `<target_bucket_name>` with your own value.
4. Create a **CORS configuration** with the following rules:

```xml
<?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>
```

### 2.5.10.2. Create an OpenShift secret

The following examples show 3scale **fileStorage** using Amazon S3 instead of persistent volume claim (PVC).

**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 **fileStorage S3 credentials secret fields** table for more details.

In the following example, **Secret** name can be anything, as it is be referenced in the **APIManager CR**.

```yaml
apiVersion: v1
class: Secret
metadata:
  creationTimestamp: null
  name: aws-auth
stringData:
  AWS_ACCESS_KEY_ID: <ID_123456>
  AWS_SECRET_ACCESS_KEY: <ID_9876544>
  AWS_BUCKET: <mybucket.example.com>
  AWS_REGION: eu-west-1
type: Opaque
```

Lastly, create the **APIManager CR** to deploy 3scale.

```yaml
apiVersion: apps.3scale.net/v1alpha1
class: APIManager
metadata:
  name: <example_apimanager>
  namespace: <3scale_test>
spec:
  wildcardDomain: lvh.me
  system:
    fileStorage:
      simpleStorageService:
        configurationSecretRef:
          name: aws-auth
```

Check **APIManager SystemS3Spec**.

The following table shows the **fileStorage** Amazon S3 credentials secret field requirements for Identity and Access Management (IAM) and Security Token Service (STS) settings:
The S3 authentication method using Secure Token Service (STS) is for short-term, limited-privilege security credentials.

S3 Identity and Access Management (IAM) is for long-term privilege security credentials.

Table 2.1. fileStorage S3 credentials secret fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Required for IAM</th>
<th>Required for STS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS_ACCESS_KEY_ID</td>
<td>AWS Access Key ID to use in S3 Storage for system’s fileStorage</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AWS_SECRET_ACCESS_KEY</td>
<td>AWS Access Key Secret to use in S3 Storage for system’s fileStorage</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AWS_BUCKET</td>
<td>The S3 bucket to be used as system’s fileStorage for assets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AWS_REGION</td>
<td>The region of the S3 bucket to be used as system’s fileStorage for assets</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AWS_HOSTNAME</td>
<td>Default: Amazon endpoints - An AWS S3 compatible provider endpoint hostname</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AWS_PROTOCOL</td>
<td>Default: HTTPS An AWS S3 compatible provider endpoint protocol</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AWS_PATH_STYLE</td>
<td>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</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AWS_ROLE_ARN</td>
<td>ARN of the Role which has a policy attached to authenticate using AWS STS</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Required for IAM</td>
<td>Required for STS</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>AWS_WEB_IDENTITY_TOKEN_FILE</td>
<td>Path to mounted token file location. For example: /var/run/secrets/openshift/serviceaccount/token</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

2.5.10.3. Manual mode with STS

STS authentication mode must be enabled from the APIManager CR. You can define your audience, however, the default value is *openshift*.

**Prerequisites**

- Configure OpenShift to use temporary credentials for different components with AWS Security Token Service (STS). For further detail see [Using manual mode with Amazon Web Services Secure Token Service](#).

```yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: <apimanager_sample>
  namespace: <3scale_test>
spec:
  wildcardDomain: lvh.me
  system:
    fileStorage:
      simpleStorageService:
        configurationSecretRef:
          name: s3-credentials
          sts:
            enabled: true
            audience: openshift
```

The secret generated by the cloud credential tooling looks different from the IAM secret. There are two new fields **AWS_ROLE_ARN** and **AWS_WEB_IDENTITY_TOKEN_FILE** instead of **AWS_ACCESS_KEY_ID** and **AWS_SECRET_ACCESS_KEY**.

**STS secret example**

```yaml
kind: Secret
apiVersion: v1
metadata:
  name: s3-credentials
  namespace: 3scale
data:
  AWS_ROLE_ARN: arn:aws:iam::ID:role/ROLE
  AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/openshift/serviceaccount/token
```
With STS, the 3scale operator adds the projected volume to request the token. The following pods have a projected volume:

- **system-app**
- **system-app hook pre**
- **system-sidekiq**

Pod example for STS

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: system-sidekiq-1-zncrz
  namespace: 3scale-test
spec:
  containers:

  volumeMounts:
  - mountPath: /var/run/secrets/openshift/serviceaccount
    name: s3-credentials
    readOnly: true

  volumes:
  - name: s3-credentials
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
        audience: openshift
        expirationSeconds: 3600
        path: token
```

Additional resources

- **APIManager SystemS3Spec**
- **S3 secret reference**
- **Using manual mode with Amazon Web Services Secure Token Service**
- **Short lived Credentials with AWS Security Token Service**

2.5.11. PostgreSQL installation

A MySQL internal relational database is the default deployment. This deployment configuration can be overridden to use PostgreSQL instead.
2.5.12. Configuring SMTP variables (optional)

3scale 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:
   ```
ooc login
   ```

2. 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:

   **Table 2.2. system-smtp**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>This is the address (hostname or IP) of the remote mail server to use. If this is set to a value different than <code>''</code>, system will use the mail server to send mails related to events that happen in the API management solution.</td>
<td><code>''</code></td>
</tr>
<tr>
<td>port</td>
<td>This is the port of the remote mail server to use.</td>
<td><code>''</code></td>
</tr>
<tr>
<td>domain</td>
<td>Use domain if the mail server requires a HELO domain.</td>
<td><code>''</code></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
<td>Default value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>authentication</td>
<td>Use if the mail server requires authentication. Set the authentication types: <strong>plain</strong> to send the password in the clear, <strong>login</strong> to send password Base64 encoded, or <strong>cram_md5</strong> to combine a challenge/response mechanism based on the HMAC-MD5 algorithm.</td>
<td>****</td>
</tr>
<tr>
<td>username</td>
<td>Use <strong>username</strong> if the mail server requires authentication and the authentication type requires it.</td>
<td>****</td>
</tr>
<tr>
<td>password</td>
<td>Use <strong>password</strong> if the mail server requires authentication and the authentication type requires it.</td>
<td>****</td>
</tr>
<tr>
<td>openssl.verify.mode</td>
<td>When using TLS, you can set how OpenSSL checks the certificate. This is useful if you need to validate a self-signed and/or a wildcard certificate. You can use the name of an OpenSSL verify constant: <strong>none</strong> or <strong>peer</strong>.</td>
<td>****</td>
</tr>
<tr>
<td>from_address</td>
<td><strong>from</strong> address value for the no-reply mail.</td>
<td>****</td>
</tr>
</tbody>
</table>

**Examples**

```
$ 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>"}}'
```

3. 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
```

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

```
$ oc rollout status dc/system-app
$ oc rollout status dc/system-sidekiq
```
2.5.13. Customizing compute resource requirements at component level

Customize Kubernetes Compute Resource Requirements in your 3scale solution through the APIManager custom resource (CR) 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:

```yaml
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

- APIManager CRD reference

2.5.13.1. 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.

2.5.13.1.1. 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 2.3. Compute resources default values**

<table>
<thead>
<tr>
<th>Component</th>
<th>CPU requests</th>
<th>CPU limits</th>
<th>Memory requests</th>
<th>Memory limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>system-app’s</td>
<td>50m</td>
<td>1000m</td>
<td>600Mi</td>
<td>800Mi</td>
</tr>
<tr>
<td>system-master</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system-app’s</td>
<td>50m</td>
<td>1000m</td>
<td>600Mi</td>
<td>800Mi</td>
</tr>
<tr>
<td>system-provider</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system-app’s</td>
<td>50m</td>
<td>1000m</td>
<td>600Mi</td>
<td>800Mi</td>
</tr>
<tr>
<td>system-developer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system-sidekiq</td>
<td>100m</td>
<td>1000m</td>
<td>500Mi</td>
<td>2Gi</td>
</tr>
<tr>
<td>system-sphinx</td>
<td>80m</td>
<td>1000m</td>
<td>250Mi</td>
<td>512Mi</td>
</tr>
<tr>
<td>system-redis</td>
<td>150m</td>
<td>500m</td>
<td>256Mi</td>
<td>32Gi</td>
</tr>
<tr>
<td>system-mysql</td>
<td>250m</td>
<td>No limit</td>
<td>512Mi</td>
<td>2Gi</td>
</tr>
<tr>
<td>system-postgresql</td>
<td>250m</td>
<td>No limit</td>
<td>512Mi</td>
<td>2Gi</td>
</tr>
<tr>
<td>backend-listener</td>
<td>500m</td>
<td>1000m</td>
<td>550Mi</td>
<td>700Mi</td>
</tr>
<tr>
<td>backend-worker</td>
<td>150m</td>
<td>1000m</td>
<td>50Mi</td>
<td>300Mi</td>
</tr>
<tr>
<td>backend-cron</td>
<td>50m</td>
<td>150m</td>
<td>40Mi</td>
<td>80Mi</td>
</tr>
<tr>
<td>backend-redis</td>
<td>1000m</td>
<td>2000m</td>
<td>1024Mi</td>
<td>32Gi</td>
</tr>
<tr>
<td>apicast-production</td>
<td>500m</td>
<td>1000m</td>
<td>64Mi</td>
<td>128Mi</td>
</tr>
<tr>
<td>apicast-staging</td>
<td>50m</td>
<td>100m</td>
<td>64Mi</td>
<td>128Mi</td>
</tr>
<tr>
<td>zync</td>
<td>150m</td>
<td>1</td>
<td>250M</td>
<td>512Mi</td>
</tr>
<tr>
<td>Component</td>
<td>CPU requests</td>
<td>CPU limits</td>
<td>Memory requests</td>
<td>Memory limits</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
<td>------------</td>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>zync-que</td>
<td>250m</td>
<td>1</td>
<td>250M</td>
<td>512Mi</td>
</tr>
<tr>
<td>zync-database</td>
<td>50m</td>
<td>250m</td>
<td>250M</td>
<td>2G</td>
</tr>
</tbody>
</table>

### 2.5.14. Customizing node affinity and tolerations at component level

Customize Kubernetes **Affinity** and **Tolerations** in your Red Hat 3scale API Management solution through the APIManager CR 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**:

```yaml
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

- APIManager CRD reference

### 2.5.15. Pod priority of 3scale components
As a 3scale administrator, you can set up the pod priority for various 3scale installed components by modifying the APIManager custom resource (CR). Use the optional priorityClassName available in the following components:

- apicast-production
- apicast-staging
- backend-cron
- backend-listener
- backend-worker
- backend-redis
- system-app
- system-sidekiq
- system-searchd
- system-memcache
- system-mysql
- system-postgresql
- system-redis
- zync
- zync-database
- zync-que

For example:

```yaml
apiversion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: api.vmogilev01.0nno.s1.devshift.org
  resourceRequirementsEnabled: false
apicast:
  stagingSpec:
    priorityClassName: openshift-user-critical
  productionSpec:
    priorityClassName: openshift-user-critical
backend:
  listenerSpec:
    priorityClassName: openshift-user-critical
  cronSpec:
    priorityClassName: openshįt-user-critical
  workerSpec:
    priorityClassName: openshift-user-critical
```
2.5.16. Setting backend client to skip certificate verification

When a controller processes an object, it generates a new backend client for making API calls. By default, this client is set up to confirm the server’s certificate chain. However, during development and testing, you might need the client to skip certificate verification when processing an object. To achieve this, add the annotation "insecure_skip_verify": "true" to the following objects:

- ActiveDoc
- Application
- Backend
- CustomPolicyDefinition
- DeveloperAccount
- DeveloperUser
- OpenAPI - backend and product
- Product
- ProxyConfigPromote
- Tenant

OpenAPI CR example:

```yaml
apiVersion: capabilities.3scale.net/v1beta1
```
2.5.17. Setting custom annotations

In 3scale, the components' pods have annotations. These are key/value pairs used for configurations. You can change these annotations for any 3scale component using the APIManager CR.

**NOTE**

If you remove an annotation defined in a custom resource (CR), it is not automatically removed from the associated DeploymentConfig (DC). You must manually remove the annotation from the DC.

APIManager annotations for apicast-staging and backend-listener

```yaml
kind: OpenAPI
metadata:
  name: ownertest
  namespace: threescale
  annotations:
    "insecure_skip_verify": "true"

spec:
  openapiRef:
    secretRef:
      name: myopenapi
      namespace: threescale
    productSystemName: testProduct
```

2.5.18. Reconciliation

Once 3scale has been installed, the 3scale operator enables updating a given set of parameters from the custom resource (CR) 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 CR definitions (CRDs) are reconcilable.

The following is a list of reconcilable parameters:

```yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: example.com
  apicast:
    stagingSpec:
      annotations:
        anno-sample1: anno1
  backend:
    listenerSpec:
      annotations:
        anno-sample2: anno2
```
2.5.18.1. Resources

Resource limits and requests for all 3scale components.

```yaml
apiVersion: apps.3scale.net/v1alpha1
class: APIManager
metadata:
  name: example-apimanager
spec:
  resourceRequirementsEnabled: true/false
```

2.5.18.2. Backend replicas

*Backend* components pod count.

```yaml
apiVersion: apps.3scale.net/v1alpha1
class: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      replicas: X
    workerSpec:
      replicas: Y
    cronSpec:
      replicas: Z
```

2.5.18.3. APIcast replicas

*APIcast* staging and production components pod count.

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

2.5.18.4. System replicas
**System** app and system **sidekiq** components pod count

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

**2.5.18.5. Zync replicas**

**Zync** app and que components pod count

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

## 2.6. 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 2.14 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.

**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.

- In this documentation `myregistry.example.com` is used as an example of the registry URL. Replace it with your registry URL.

- Disclaimer: Links contained herein 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.
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 CR, as it will be created in the following procedure.

- A Registry service account for 3scale.

- A supported version of the Oracle Database accessible from your OpenShift cluster.

- Access to the Oracle Database SYSTEM user for installation procedures.

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

- Preparing the Oracle Database
- Building a custom system container image
- Installing 3scale with Oracle using the operator

2.6.1. Preparing the Oracle Database

As a 3scale administrator, you must fully prepare the Oracle Database for your 3scale installation when you decide to use it for the System component.

Procedure

1. Create a new database.

2. Apply the following settings:

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

3. Configure a database user

   There are two options for setting up Oracle Database integration in 3scale: with or without providing the Oracle SYSTEM user password.

   3scale uses the SYSTEM user only for the initial setup, which consist in creating a regular user and granting it the required privileges. The following SQL commands will set up a regular user with proper permissions. ( `{DB_USER}` and `{DB_PASSWORD}` are placeholders that need to be replaced with actual values):

   ```sql
   CREATE USER `{DB_USER}` IDENTIFIED BY `{DB_PASSWORD}`;
   GRANT unlimited tablespace TO `{DB_USER}`;
   GRANT create session TO `{DB_USER}`;
   GRANT create table TO `{DB_USER}`;
   GRANT create view TO `{DB_USER}`;
   GRANT create sequence TO `{DB_USER}`;
   GRANT create trigger TO `{DB_USER}`;
   GRANT create procedure TO `{DB_USER}`;
   ```

   a. Using the SYSTEM user:

      - Provide the SYSTEM user password in ORACLE SYSTEM PASSWORD field of the
- Provide the `SYSTEM` user password in `ORACLE_SYSTEM_PASSWORD` field of the `system-database` secret.

- The regular user does not need to exist before the installation. It will be created by the 3scale initialization script.

- Provide the desired username and password for the regular user in the connection string (e.g. `oracle-enhanced://(DB_USER):(DB_PASSWORD)@[DB_HOST]: (DB_PORT)/(DB_NAME)`) in the `URL` field of the `system-database` secret.

- The password for the regular Oracle Database non-system user must be unique and not match the `SYSTEM` user password.

- If the user with the specified username already exists, the 3scale initialization script will attempt to update the password using the following command:

  ```sql
  ALTER USER {DB_USER} IDENTIFIED BY {DB_PASSWORD}
  ```

  Your database configuration might prevent this command from completing successfully if the parameters `PASSWORD_REUSE_TIME` and `PASSWORD_REUSE_MAX` are set in a way that restricts reusing the same password.

b. Manual setup of the regular database user:

- You don’t need to provide the `ORACLE_SYSTEM_PASSWORD` in the `system-database` secret.

- The regular database user (not `SYSTEM`) specified in the connection string in the `URL` field of the `system-database` secret needs to exist prior to the 3scale installation.

- The regular user used for the installation must have all the privileges listed above.

Additional resources

- For information on creating a new database, see the Oracle Database 19c documentation.

2.6.2. Building a custom system container image

Procedure

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

   ```
   tar -xzf 3scale-2.14.0-GA.tar.gz
   ```

2. 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)
3. Check the table for the following Oracle software component versions:

- Oracle Instant Client Package: Basic or Basic Light
- Oracle Instant Client Package: SDK
- Oracle Instant Client Package: ODBC

<table>
<thead>
<tr>
<th>Table 2.4. Oracle 19c example packages for 3scale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oracle 19c package name</strong></td>
</tr>
<tr>
<td>Basic</td>
</tr>
<tr>
<td>Basic Light</td>
</tr>
<tr>
<td>SDK</td>
</tr>
<tr>
<td>ODBC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2.5. Oracle 19c example packages for ppc64le and 3scale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oracle 19c package name</strong></td>
</tr>
<tr>
<td>Basic</td>
</tr>
<tr>
<td>Basic Light</td>
</tr>
<tr>
<td>SDK</td>
</tr>
<tr>
<td>ODBC</td>
</tr>
</tbody>
</table>

**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.

4. Place your Oracle Database Instant Client Package files into the `system-oracle-3scale-2.13.0-GA/oracle-client-files` directory.

5. Login to your `registry.redhat.io` account using the credentials you created in Creating a Registry Service Account.
6. 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:2.14.0-1
```

7. 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:2.14.0-1
```

### 2.6.3. Installing 3scale with Oracle using the operator

**Procedure**

1. 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.

2. Install your 3scale solution by creating an `APIManager` CR. Follow the instructions in [Deploying 3scale using the operator](#).

   - The `APIManager` CR must specify the `.spec.system.image` field set to the system's Oracle-based image you previously built:

     ```yaml
     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:2.14.0-1"
     ```

### 2.7. 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
2.7.1. 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.

2.7.2. 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:
CHAPTER 2. INSTALLING 3SCALE ON OPENSФIFT

$ 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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3scale-operator-66cc6d857b-sxhgm</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>17h</td>
</tr>
<tr>
<td>apicast-production-1-deploy</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>17m</td>
</tr>
<tr>
<td>apicast-production-1-pxkqm</td>
<td>0/1</td>
<td>Pending</td>
<td>0</td>
<td>17m</td>
</tr>
<tr>
<td>apicast-staging-1-dbwcw</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>17m</td>
</tr>
<tr>
<td>apicast-staging-1-deploy</td>
<td>0/1</td>
<td>Completed</td>
<td>0</td>
<td>17m</td>
</tr>
<tr>
<td>backend-cron-1-deploy</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>17m</td>
</tr>
</tbody>
</table>

2.7.3. 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.

2.7.4. 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.
CHAPTER 2. INSTALLING 3SCALE ON OPENSIFT

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

2.7.5. 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.

2.7.6. 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.

2.7.7. 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>
```

2.8. ADDITIONAL RESOURCES

- External Components Specification
- System database
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 product
- Deploying APIcast on the Docker containerized environment
- 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: A 3scale API Management installation includes two default APIcast gateways, staging and production. These gateways come preconfigured and ready for immediate use.

- 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, 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.
• **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_product_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 PRODUCT

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 product:

- 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:

  ```bash
  $ curl "https://echo-api.3scale.net:443"
  
  You will get the following response:
  ```

  ```json
  {
    "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

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

<table>
<thead>
<tr>
<th>Response code</th>
<th>Response body</th>
</tr>
</thead>
<tbody>
<tr>
<td>403</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>403</td>
<td>Authentication parameters missing</td>
</tr>
<tr>
<td>404</td>
<td>No Mapping Rule matched</td>
</tr>
<tr>
<td>429</td>
<td>Usage limit exceeded</td>
</tr>
</tbody>
</table>

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.


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:
   ```bash
   $ sudo systemctl start docker.service
   ```

2. Check if the Docker daemon is running:
   ```bash
   $ sudo systemctl status docker.service
   ```

3. Download a ready to use Docker container engine image from the Red Hat registry:
   ```bash
   $ sudo docker pull registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.14
   ```

4. Run APIcast in a Docker container engine:
   ```bash
   $ 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-rhel8:3scale2.14
   ```

   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/9.x
- Podman 4.2.0/4.1.1

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 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-rhel8:3scale2.14
  ```

2. Run APIcast in a Podman:
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

- Red Hat 3scale API Management Supported Configurations
- Basic Setup and Use of Podman

### 3.6. 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.
The default settings are for production environment when you deploy APIcast. You can always adjust these settings for deploying a staging environment. For example, use the following `oc` command:

```
$ oc patch apicast/{apicast_name} --type=merge -p '{"spec":
{"deploymentEnvironment":"staging","configurationLoadMode":"lazy"}}'
```

For more information, see the: [APIcast Custom Resource reference](#).

**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 *APIcast > Create APIcast*.

### 3.6.1. 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](#)

#### 3.6.1.1. 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

```bash
$ 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

```yaml
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:

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

3.6.1.1.1. 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`:

   ```bash
   $ 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:

   ```bash
   $ curl 127.0.0.1:8080/test -H "Host: localhost"
   ```
3.6.1.1.2. 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`:

```json
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**

<table>
<thead>
<tr>
<th>json/yaml field</th>
<th>Type</th>
<th>Required</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>host</code></td>
<td>string</td>
<td>Yes</td>
<td>N/A</td>
<td>Domain name being routed to the gateway</td>
</tr>
<tr>
<td><code>tls</code></td>
<td>[]networkv1.IngressTLS</td>
<td>No</td>
<td>N/A</td>
<td>Array of ingress TLS objects. See more on TLS.</td>
</tr>
</tbody>
</table>

3.6.1.2. Providing a configuration secret

**Procedure**

1. Create a secret with the configuration file:
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:

```bash
$ 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:

```bash
$ 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

2. Create an APIcast custom resource:

```bash
$ 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

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

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

The `spec.eddedConfigurationSecretRef.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:

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

3.6.1.2.1. 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`:

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

   a. Next: Make a request to a configured 3scale service and verify a successful HTTP response.

3.6.1.3. 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.

   ```lua
   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**

If you update your custom environment be sure to re-create its secret so the secret contains the update. The APIcast operator watches for updates and automatically redeployes when it finds an update.

**3.6.1.4. 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
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:

   ```bash
   $ 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.

   ```yaml
   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:

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

Next steps

- If you update your custom policy be sure to re-create its secret so the secret contains the update. The

  ```bash
  $ 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 update your custom policy be sure to re-create its secret so the secret contains the update. The **APIcast** operator watches for updates and automatically redeploys when it finds an update.

**Additional resources**

- [APIcast custom resource definition](#)

### 3.6.1.5. 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.

**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 following example shows a valid secret definition:

```yaml
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
      }
    }
```
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 -f 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.

```yaml
apiVersion: apps.3scale.net/v1alpha1
customResourceDefinition
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**

**3.7. ADDITIONAL RESOURCES**

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

- [Red Hat 3scale API Management Supported Configurations](https://example.com)
- [Red Hat 3scale API Management - Component Details](https://example.com)
Important

Red Hat supports 3scale configurations that use an external Redis database. However, 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).

NOTE

When you externalize databases from a Red Hat 3scale API Management deployment, this means to provide isolation from the application and resilience against service disruptions at the database level. The resilience to service disruptions depends on the service level agreements (SLAs) provided by the infrastructure or platform provider where you host the databases. This is not offered by 3scale. For more details on externalizing of databases offered by your chosen deployment, see the associated documentation.

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.

4.1. 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.

## 4.2. 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 provide sentinel configuration in either backend-redis, system-redis, or both secrets.

### 4.2.1. 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

### 4.2.2. Deploying a fresh installation of 3scale for HA

**Procedure**

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

   **backend-redis**

<table>
<thead>
<tr>
<th>REDIS_QUEUES_SENTINEL_HOSTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>REDIS_QUEUES_SENTINEL_ROLE</td>
</tr>
<tr>
<td>REDIS_QUEUES_URL</td>
</tr>
<tr>
<td>REDIS_STORAGE_SENTINEL_HOSTS</td>
</tr>
<tr>
<td>REDIS_STORAGE_SENTINEL_ROLE</td>
</tr>
<tr>
<td>REDIS_STORAGE_URL</td>
</tr>
</tbody>
</table>
system-redis

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 using the operator.

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

4.2.3. 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.

```yaml
name: REDIS_SENTINEL_HOSTS
valueFrom:
  secretKeyRef:
    key: SENTINEL_HOSTS
    name: system-redis

name: REDIS_SENTINEL_ROLE
valueFrom:
  secretKeyRef:
    key: SENTINEL_ROLE
    name: system-redis
```

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

### 4.2.3.1. Using Redis Enterprise

1. Use Redis Enterprise deployed in OpenShift, with three different `redis-enterprise` instances:
   
   a. Edit `system-redis` secret:
      
      i. Set the system redis database in `system-redis` to `URL`.
   
   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`.

### 4.2.3.2. Using Redis Sentinel

**NOTE**

You can optionally apply Redis Sentinels to any of the databases. However, Red Hat recommends that you apply Redis Sentinels to all of them for HA.

1. Backend redis for statistics: update `backend-redis` secret and provide values for:
   
   - `REDIS_STORAGE_URL`
   
   - `REDIS_STORAGE_SENTINEL_ROLE`
   
   - `REDIS_STORAGE_SENTINEL_HOSTS`
     
     Set `REDIS_STORAGE_SENTINEL_ROLE` to a comma-separated list of sentinels hosts and ports, for example: `:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722`
   
2. Backend redis for queue: update `backend-redis` secret and provide values for:
   
   - `REDIS_QUEUES_URL`
   
   - `REDIS_QUEUES_SENTINEL_ROLE`
   
   - `REDIS_QUEUES_SENTINEL_HOSTS`
Set `REDIS_QUEUES_SENTINEL_ROLE` to a comma-separated list of sentinels hosts and ports, for example: `:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722`

3. Use Redis Sentinel, with these Redis databases:

4. System redis for data: update `system-redis` secret and provide values for:

   **NOTE**
   Edit `system-redis` secret: URL

   - **SENTINEL_ROLE**
   - **NAMESPACE**
   - **URL**
   - **SENTINEL_HOSTS**
     Set `SENTINEL_HOSTS` to a comma-separated list of sentinels hosts and ports, for example: `:sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722`

**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.

**4.3. 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:
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.

4.4. 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 5. CONFIGURING AN EXTERNAL MYSQL DATABASE

IMPORTANT

When you externalize databases from a Red Hat 3scale API Management deployment, this means to provide isolation from the application and resilience against service disruptions at the database level. The resilience to service disruptions depends on the service level agreements (SLAs) provided by the infrastructure or platform provider where you host the databases. This is not offered by 3scale. For more details on externalizing of databases offered by your chosen deployment, see the associated documentation.

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

This guide provides information for externalizing the MySQL database. This is useful where there are several infrastructure issues, such as network or filesystem, using the default system-mysql pod.

Prerequisites

- Access to an OpenShift Container Platform 4.x cluster using an account with administrator privileges.
- A 3scale instance installation on the OpenShift cluster. See Installing 3scale on OpenShift.

To configure an external MySQL database, perform the steps outlined in the following sections:

- Section 5.1, “External MySQL database limitations”
- Section 5.2, “Externalizing the MySQL database”
- Section 5.3, “Rolling back”

5.1. 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.

5.2. 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 pods -n <3scale_namespace>
   
   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 8.0, for Linux (x86_64)
   --
   -- Host: localhost    Database:
   -- ------------------------------------------------------
   -- Server version   8.0
   /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
   /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
   ```
7. Scale down the `system-mysql` pod and leave it with 0 (zero) replicas:

   ```shell
   $ 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:

   ```shell
   $ 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:

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

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

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

    ```shell
    $ 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:

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

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

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

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

    ```shell
    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.

### 5.3. 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>
```

### 5.4. ADDITIONAL INFORMATION

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