Red Hat
Training and Certification

Student Workbook (ROLE)

Red Hat 3scale API Management 2.11 DO240

Cloud-native API Administration with Red Hat 3scale API Management

Edition 2

Red Hat
Learning Community

Join a community dedicated to learning open source

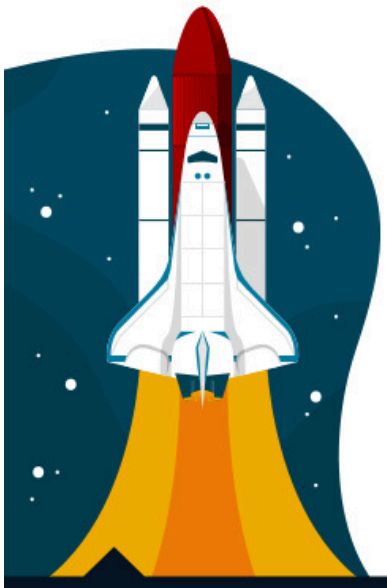
The Red Hat® Learning Community is a collaborative platform for users to accelerate open source skill adoption while working with Red Hat products and experts.

Network with tens of thousands of comunity members

Engage in thousands of active conversations and posts

Join and interact with hundreds of certified training instructors

Unlock badges as you participate and accomplish new goals



This knowledge-sharing platform creates a space where learners can connect, ask questions, and collaborate with other open source practitioners.

Access free Red Hat training videos

Discover the latest Red Hat Training and Certification news

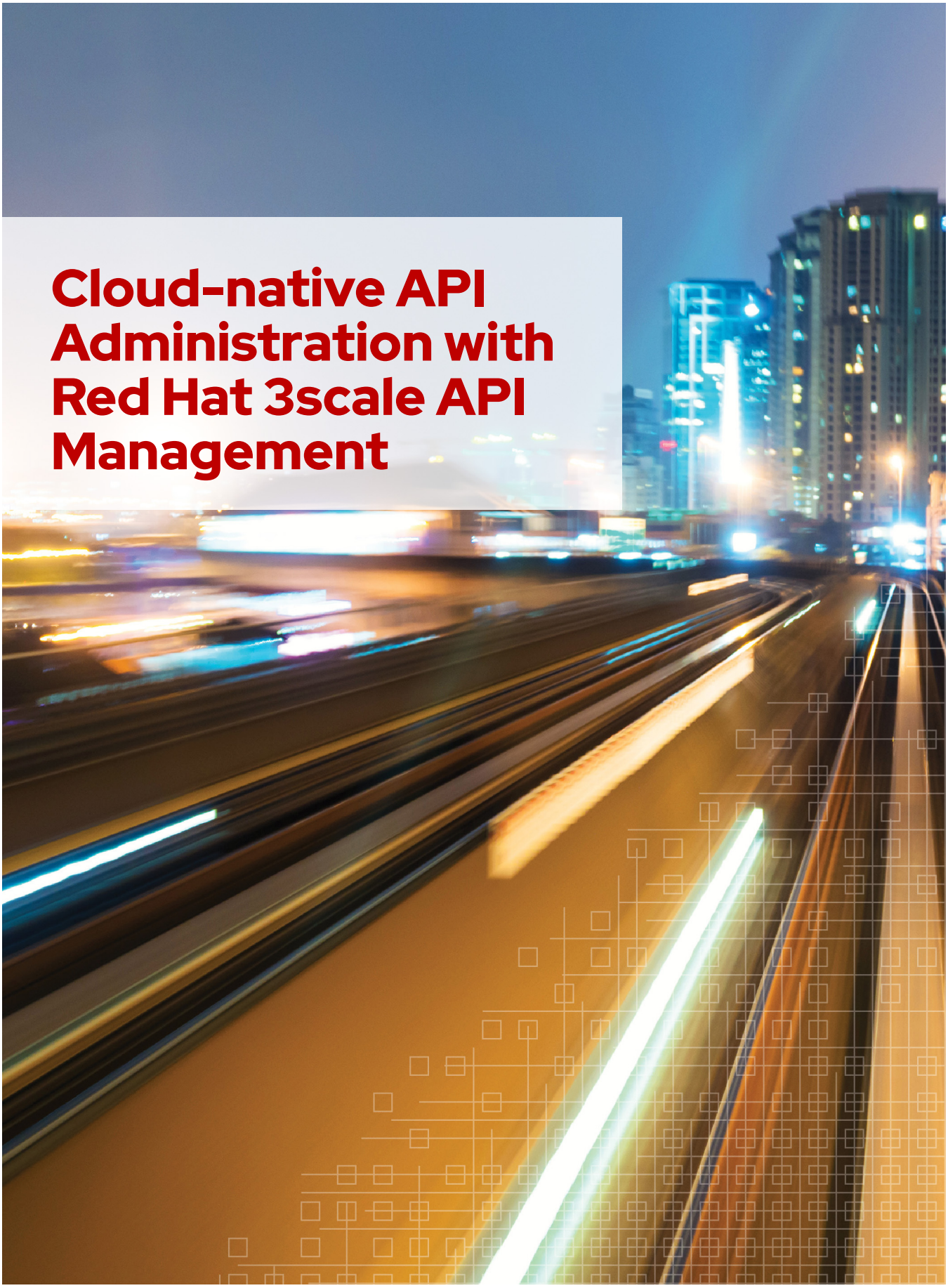
Connect with your instructor - and your classmates - before, after, and during your training course.

Join peers as you explore Red Hat products

Join the conversation learn.redhat.com

Red Hat

Cloud-native API Administration with Red Hat 3scale API Management



Red Hat 3scale API Management 2.11 DO240
Cloud-native API Administration with Red Hat 3scale API Management
Edition 2 20220405
Publication date 20220405

Authors:	Alejandro Serna-Borja, Enol Alvarez de Prado, Marek Czernek, Guy Bianco IV, Iván Chavero
Course Architect:	Ravishankar Srinivasan
DevOps Engineer:	Richard Allred
Editor:	Sam Ffrench

Contributors: Joel Birchler, Chetan Tiwary, Heider Souza

Document Conventions **ix**
ix

Introduction **xi**
Cloud-native API Administration with Red Hat 3scale API Management xi
Orientation to the Classroom Environment xii

1. Introducing Red Hat 3scale API Management **1**
Introducing Red Hat 3scale API Management 2
Guided Exercise: Creating a Lab Environment and Configuring the classroom environment 8
Installing Red Hat 3scale API Management 10
Guided Exercise: Installing Red Hat 3scale API Management 15
Summary 22

2. Managing APIs with Red Hat 3scale API Management **23**
Creating Backends and Products 24
Guided Exercise: Creating Backends and Products 29
Creating Application Plans and Rate Limits 34
Guided Exercise: Creating Application Plans and Rate Limits 38
Managing API Versions 46
Guided Exercise: Managing API Versions 51
Configuring Service Discovery 57
Guided Exercise: Configuring Service Discovery 62
Creating Multiple Tenants in Red Hat 3Scale API Management 71
Guided Exercise: Creating Multiple Tenants in Red Hat 3Scale API Management 79
Quiz: Managing APIs with Red Hat 3scale API Management 85
Summary 89

3. Managing and Customizing API Gateways **91**
Creating Self Managed APIcast Gateways 92
Guided Exercise: Creating Self Managed APIcast Gateways 97
Configuring Standard APIcast Policies 101
Guided Exercise: Configuring Standard APIcast Policies 104
Quiz: Managing and Customizing API Gateways 108
Summary 114

4. Securing APIs with Red Hat 3scale API Management **115**
Creating User Accounts for the 3Scale Admin Portal 116
Guided Exercise: Creating User Accounts for the 3Scale Admin Portal 120
Securing APIs with API Keys and API Key-pair Authentication 125
Guided Exercise: Securing APIs with API Keys and API Key-pair Authentication 129
Securing APIs with Red Hat Single Sign-on and OAuth 132
Guided Exercise: Securing APIs with Red Hat Single Sign-on and OAuth 137
Quiz: Securing APIs with Red Hat 3scale API Management 142
Summary 146

5. Creating a Developer Portal **147**
Creating an API Developer Portal 148
Guided Exercise: Creating an API Developer Portal 152
Customizing an API Developer Portal 155
Guided Exercise: Customizing an API Developer Portal 158
Importing OpenAPI Definitions 162
Guided Exercise: Importing OpenAPI Definitions 165
Quiz: Creating a Developer Portal 169
Summary 173

6. Monitoring APIs with Red Hat 3scale API Management **175**

Configuring API Analytics 176
Guided Exercise: Configuring API Analytics 180
Monitoring 3Scale Infrastructure 183
Guided Exercise: Monitoring 3Scale Infrastructure 186
Quiz: Monitoring APIs with Red Hat 3scale API Management 191
Summary 195

7. Monetizing APIs with Red Hat 3Scale API Management 197
Configuring Billing for APIs 198
Guided Exercise: Configuring Billing for APIs 202
Configuring Pricing Rules for API Consumption 205
Guided Exercise: Configuring Pricing Rules for API Consumption 208
Quiz: Monetizing APIs with Red Hat 3Scale API Management 210
Summary 212

Document Conventions

This section describes various conventions and practices used throughout all Red Hat Training courses.

Admonitions

Red Hat Training courses use the following admonitions:

References
These describe where to find external documentation relevant to a subject.

Note
These are tips, shortcuts, or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on something that makes your life easier.

Important
These provide details of information that is easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring these admonitions will not cause data loss, but may cause irritation and frustration.

Warning
These should not be ignored. Ignoring these admonitions will most likely cause data loss.

Inclusive Language

Red Hat Training is currently reviewing its use of language in various areas to help remove any potentially offensive terms. This is an ongoing process and requires alignment with the products and services covered in Red Hat Training courses. Red Hat appreciates your patience during this process.

Introduction

Cloud-native API Administration with Red Hat 3scale API Management

Cloud-native APIs with Red Hat 3scale API Management is a hands-on, lab-based course that gives SREs and administrators an introduction to managing cloud-native APIs with Red Hat® 3scale API Management. This course enables developers and administrators to install, administer, monetize, secure and document your cloud-native APIs.

This course is based on Red Hat 3scale API Management 2.11.

Course Objectives

- Installing, administering, monetizing, securing and documenting your cloud-native APIs.
- This course prepares the student to take the Red Hat Certified Specialist in API Management exam (EX240).

Audience

- Site Reliability Engineers and OpenShift Administrators interested in using 3scale API Management to manage, monitor, secure, and monetize their cloud-native APIs.

Prerequisites

- Take our free assessment to gauge whether this offering is the best fit for your skills.
- Complete Red Hat OpenShift Administration II: Operating a Production Kubernetes Cluster (DO280) or demonstrate equivalent knowledge.
- Familiarity with testing REST APIs and security concepts like API keys, tokens and OAuth/OIDC.

Orientation to the Classroom Environment

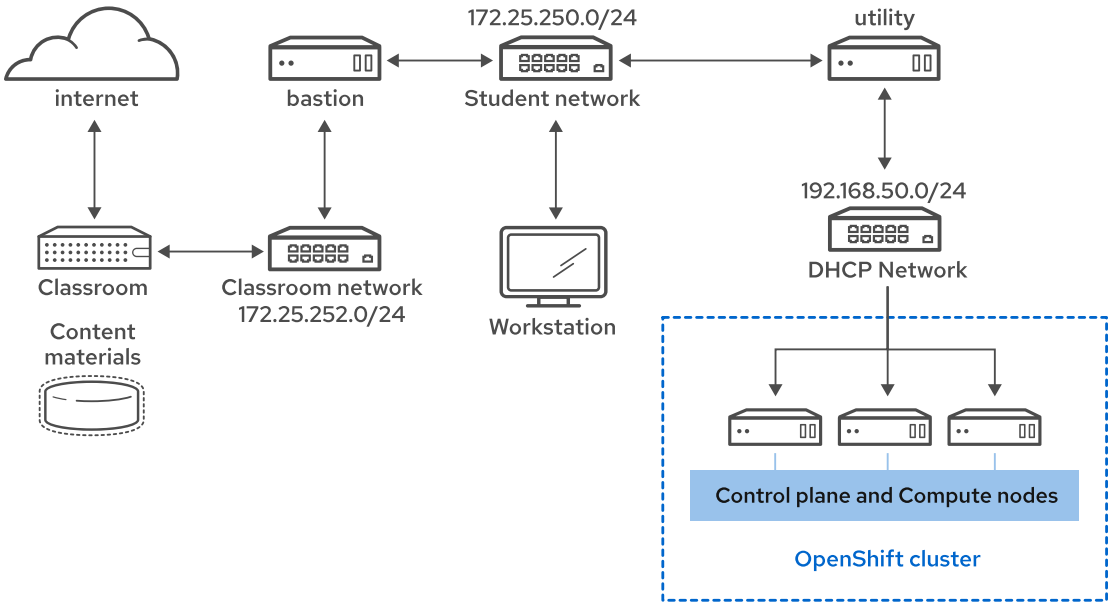


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is `workstation`. The system called `bastion` must always be running. These two systems are in the `lab.example.com` DNS domain.

All student computer systems have a standard user account, `student`, which has the password `student`. The `root` password on all student systems is `redhat`.

Classroom Machines

Machine name	IP addresses	Role
workstation.lab.example.com	172.25.250.9	Graphical workstation used by students
bastion.lab.example.com	172.25.250.254	Router linking student's VMs to classroom servers
classroom.lab.example.com	172.25.252.254	Server hosting the classroom materials required by the course
utility.lab.example.com	172.25.250.253	Server providing supporting services required by the RHOCP cluster including DHCP and NFS and routing to the RHOCP servers.

Machine name	IP addresses	Role
master01.ocp4.example.com	192.168.50.10	An OpenShift control plane and compute node.
master02.ocp4.example.com	192.168.50.11	An OpenShift control plane and compute node.
master03.ocp4.example.com	192.168.50.12	An OpenShift control plane and compute node.

The `bastion` system acts as a router between the network that connects the student machines and the classroom network. If `bastion` is down, other student machines may not function properly or may even hang during boot.

The `utility` system acts as a router between the network that connects the OpenShift cluster machines and the student network. If `utility` is down, the OpenShift cluster will not function properly or may even hang during boot.

Several systems in the classroom provide supporting services. Two servers, `content.example.com` and `materials.example.com`, are sources for software and lab materials used in hands-on activities. Information on how to use these servers is provided in the instructions for those activities.

Students use the `workstation` machine to access a dedicated OpenShift cluster, for which they have cluster administrator privileges.

Controlling Your Systems

You are assigned a remote computer in a Red Hat Online Learning classroom, which is accessed through a web application hosted at `http://rol.redhat.com/`. Students should log in to this site using their Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

The virtual machines in your classroom environment are controlled through a web page. The state of each virtual machine in the classroom is displayed on the page under the **Lab Environment** tab.

Machine States

Virtual Machine State	Description
active	The virtual machine is running and available (or, when booting, soon will be).
stopped	The virtual machine is completely shut down.
building	The initial creation of the virtual machine is being performed.

Depending on the state of a machine, a selection of the following actions is available.

Classroom/Machine Actions

Button or Action	Description
CREATE	Create the ROL classroom. Creates all of the virtual machines needed for the classroom and starts them. Can take several minutes to complete.
DELETE	Delete the ROL classroom. Destroys all virtual machines in the classroom. Caution: Any work generated on the disks is lost.
START	Start all virtual machines in the classroom.
STOP	Stop all virtual machines in the classroom.
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. Students can log in directly to the virtual machine and run commands. In most cases, you should log in to the workstation virtual machine and use ssh to connect to the other virtual machines.
ACTION > Start	Start (power on) the virtual machine.
ACTION > Shutdown	Gracefully shut down the virtual machine, preserving the contents of its disk.
ACTION > Power Off	Forcefully shut down the virtual machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION > Reset	Forcefully shut down the virtual machine and reset the disk to its initial state. Caution: Any work generated on the disk is lost.

At the start of an exercise, if instructed to reset a single virtual machine node, click **ACTION** > **Reset** for only the specific virtual machine.

At the start of an exercise, if instructed to reset all virtual machines, click **ACTION** > **Reset**

If you want to return the classroom environment to its original state at the start of the course, you can click **DELETE** to remove the entire classroom environment. After the lab has been deleted, you can click **CREATE** to provision a new set of classroom systems.

Warning
The **DELETE** operation cannot be undone. Any work you have completed in the classroom environment up to that point will be lost.

The Autostop Timer

The Red Hat Online Learning enrollment entitles students to a certain amount of computer time. To help conserve allotted computer time, the ROL classroom has an associated countdown timer, which shuts down the classroom environment when the timer expires.

To adjust the timer, click + to add one hour to the timer. Note that there is a maximum time of twelve hours.

Controlling Your Systems

You are assigned remote computers in a Red Hat Online Learning classroom. They are accessed through a web application hosted at rol.redhat.com [http://rol.redhat.com]. You should log in to this site using your Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

The virtual machines in your classroom environment are controlled through a web page. The state of each virtual machine in the classroom is displayed on the page under the **Online Lab** tab.

Machine States

Virtual Machine State	Description
STARTING	The virtual machine is in the process of booting.
STARTED	The virtual machine is running and available (or, when booting, soon will be).
STOPPING	The virtual machine is in the process of shutting down.
STOPPED	The virtual machine is completely shut down. Upon starting, the virtual machine boots into the same state as when it was shut down (the disk will have been preserved).
PUBLISHING	The initial creation of the virtual machine is being performed.
WAITING_TO_START	The virtual machine is waiting for other virtual machines to start.

Depending on the state of a machine, a selection of the following actions is available.

Classroom/Machine Actions

Button or Action	Description
PROVISION LAB	Create the ROL classroom. Creates all of the virtual machines needed for the classroom and starts them. Can take several minutes to complete.
DELETE LAB	Delete the ROL classroom. Destroys all virtual machines in the classroom. **Caution: Any work generated on the disks is lost.**
START LAB	Start all virtual machines in the classroom.
SHUTDOWN LAB	Stop all virtual machines in the classroom.
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. You can log in directly to the virtual machine and run commands. In most cases, you should log in to the `workstation` virtual machine and use `ssh` to connect to the other virtual machines.
ACTION > Start	Start (power on) the virtual machine.

Button or Action	Description
ACTION > Shutdown	Gracefully shut down the virtual machine, preserving the contents of its disk.
ACTION > Power Off	Forcefully shut down the virtual machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION > Reset	Forcefully shut down the virtual machine and reset the disk to its initial state. **Caution: Any work generated on the disk is lost.**

At the start of an exercise, if instructed to reset a single virtual machine node, click **ACTION** > **Reset** for only the specific virtual machine.

At the start of an exercise, if instructed to reset all virtual machines, click **ACTION** > **Reset**

If you want to return the classroom environment to its original state at the start of the course, you can click **DELETE LAB** to remove the entire classroom environment. After the lab has been deleted, you can click **PROVISION LAB** to provision a new set of classroom systems.

Warning
The **DELETE LAB** operation cannot be undone. Any work you have completed in the classroom environment up to that point will be lost.

The Autostop Timer

The Red Hat Online Learning enrollment entitles you to a certain amount of computer time. To help conserve allotted computer time, the ROL classroom has an associated countdown timer, which shuts down the classroom environment when the timer expires.

To adjust the timer, click **MODIFY** to display the **New Autostop Time** dialog box. Set the number of hours until the classroom should automatically stop. Note that there is a maximum time of ten hours. Click **ADJUST TIME** to apply this change to the timer settings.

Chapter 1

Introducing Red Hat 3scale API Management

Goal

Describe API management concepts and the architecture of Red Hat 3scale API Management.

Objectives

- Describe the architecture of Red Hat 3scale API Management and the benefits of API Management.
- Install Red Hat 3scale API Management on OpenShift and install the 3scale Toolbox CLI.

Sections

- Introducing Red Hat 3scale API Management (and Guided Exercise)
- Installing Red Hat 3scale API Management (and Guided Exercise)

Introducing Red Hat 3scale API Management

Objectives

After completing this section, you should be able to describe the architecture of Red Hat 3scale API Management and the benefits of API Management.

Introducing 3scale API Management

You can use Red Hat 3cale API Management to easily manage your APIs. You can focus on creating the business logic of your APIs, delegating to 3scale tasks such as authentication, rate limits, or user management.

You can deploy 3scale on-premises, in the cloud, as a managed service, or as a combination.

Features

- There are two different logical components: the API Manager and the API Gateway. If the API Manager is down, then the API Gateway still works.
- Routing policies, rate limiting, user management, and monetization.
- Administrators can manage their APIs through a rich user interface, the Admin Portal.
- Administrator can manage their APIs through a rich command-line utility, the 3scale Toolbox CLI.

Introducing the 3scale Architecture

The two main logical components of 3scale are the API Manager and the API Gateway. The API Manager holds all the information regarding routing policies, users, rate limits, etc. The API Gateway receives the actual external requests and handles them according to the API Manager information.

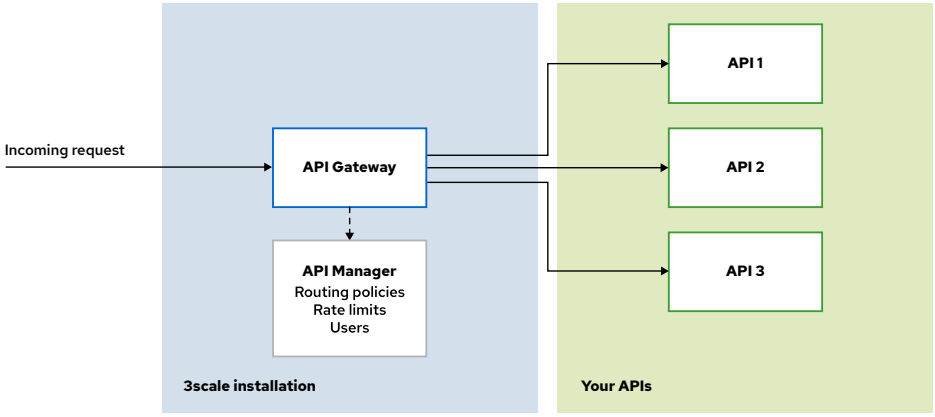


Figure 1.1: Architecture of the main two logical components of 3scale

The API Gateway fetches the API Manager information (routing policies, users, rate limits...) periodically and saves it. This means that the API Gateway does not have to get the information from the API Manager on every incoming request, therefore, reducing latency. At the same time, if the API Manager is down for some minutes, the API Gateway can work independently because it holds the cached information.

Describing the API Manager

You can consider the API Manager the administrative part of your 3scale installation. The API Manager provides the Admin Portal user interface for managing APIs, routing policies, users, rate limits, and more.

Routing policies

You can use a routing policy to define an external route where the consumers of your APIs will make requests. Then, after these requests reach 3scale, you define where they are delivered.

For example, consider that you have a book API available at `http://books.mystore.com/`. You want your clients to consume your API through 3scale to be able to manage different aspects of the API. Consider also that you define a 3scale route `/books`, which is the public external route that your clients will use. With the routing capabilities of 3scale, you can map the `/books` route to your external service `http://books.mystore.com/`. The following diagram illustrates this process.

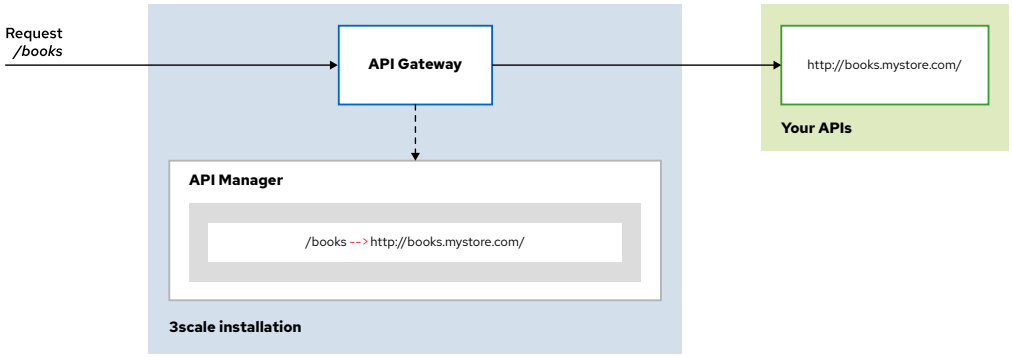


Figure 1.2: API Manager routing example

To create routing policies, 3scale provides two abstractions: products and backends. A *product* is used to define the external routes accessible by your clients. A *backend* is used to define your API services. Then, a product is mapped to one or several backends to achieve the routing policy. The following diagram illustrates the books example with a new route, `/authors`. The `/books` route is mapped to `http://books.mystore.com/` and the `/authors` route is mapped to `http://authors.mystore.com/`.

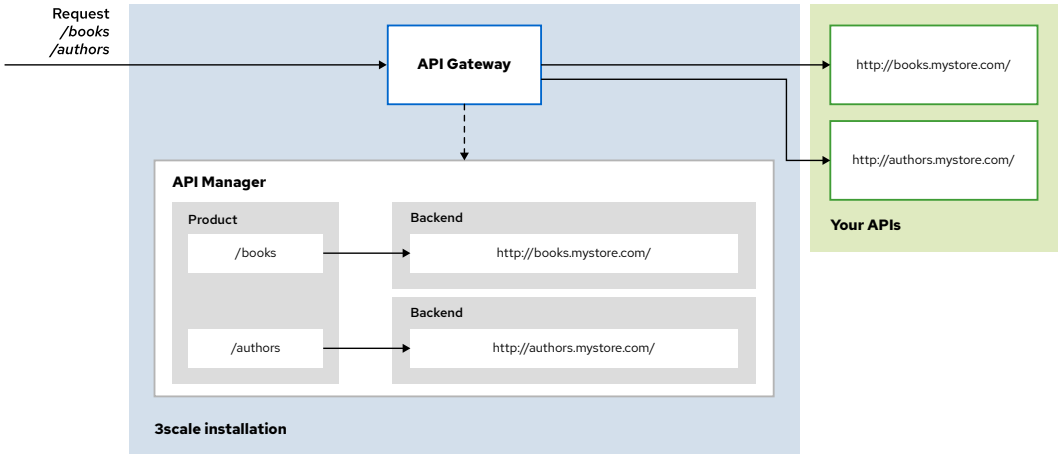


Figure 1.3: API Manager routing example based on products and backends

Both products and backends are covered later in this course.

Rate limits

You can use rate limits to control the number of requests that each of your clients can make to your APIs.

3scale provides the abstraction of an *application*. You can consider that each of your clients is a 3scale application. On every request, 3scale must identify which of your clients (i.e. applications) is making the request to be able to apply the rate limits. Every application is assigned a random user key. Then, your client must include this user key on every request.

At the same time, an application holds an *application plan*, which is used to define the rate limits for every application.

For example, consider the books example previously mentioned. If you have two clients, `client1` and `client2`, then you can create two applications. For every application, a random user key is generated, which must be provided when making external requests. For every application, you can create an application plan, which contains the rate limits of the application (i.e. how many requests the application can make in a specific period of time). The following diagram illustrates this behavior.

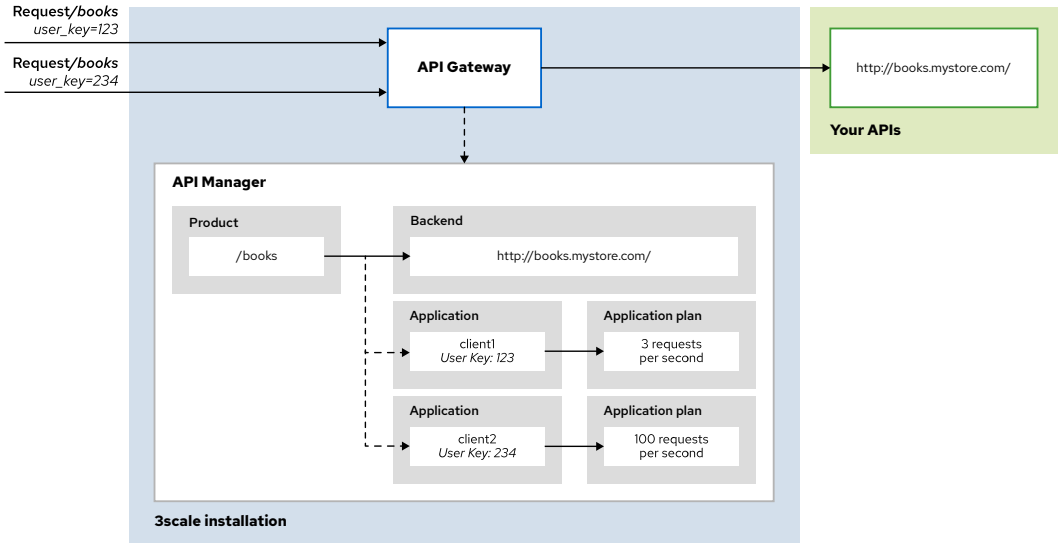


Figure 1.4: Routing limits example

Applications and application plans are covered later in this course.

Developer Portal

The *Developer Portal* is a public web page that you can use to provide information about your APIs. The Developer Portal is entirely customizable and it provides features such as Swagger definitions importing, authentication for developers, or email templates.

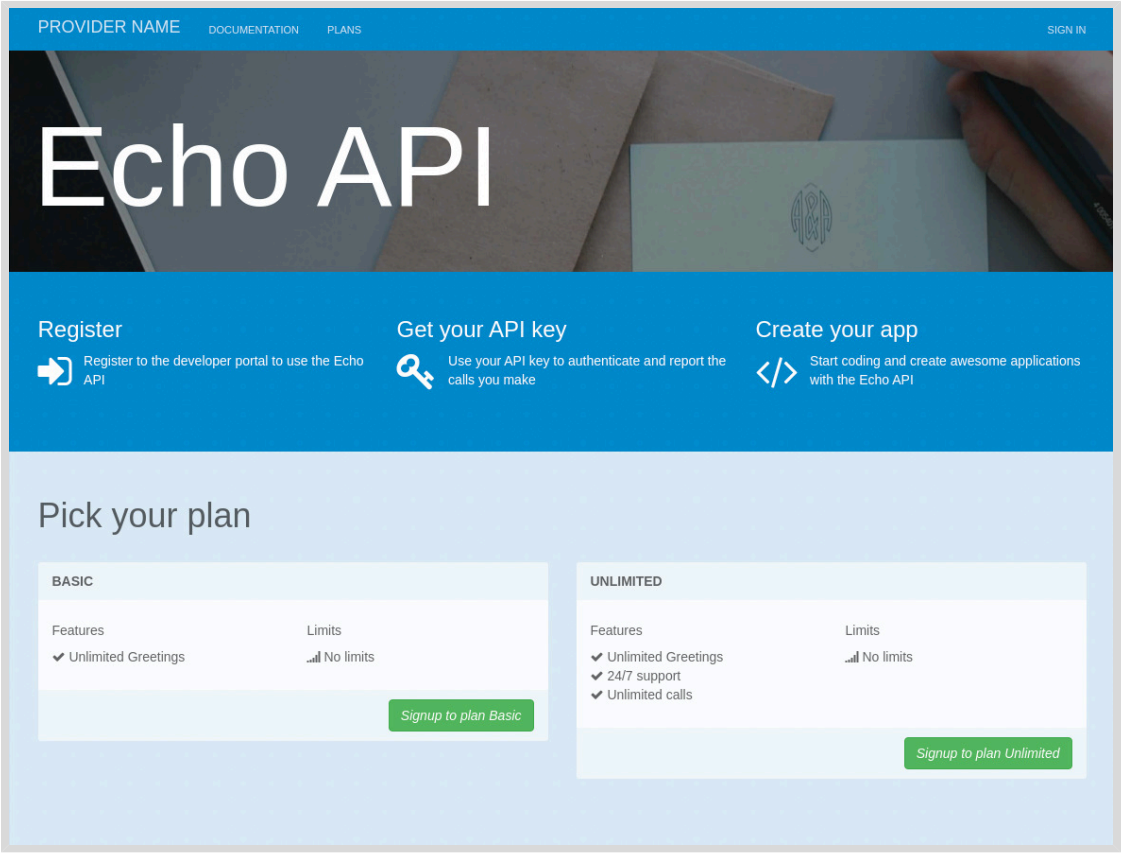


Figure 1.5: Screenshot of the default Developer Portal

The Developer Portal is covered later in this course.

Tenants

In the context of 3scale API Management, a *tenant* acts as an instance of 3scale. Within the same 3scale installation, you can create several instances of the product.

By using several tenants, you can create a logical separation of your APIs. For example, consider that you have two tenants: `orders` and `finance`. In the `order` tenants you create APIs related to orders, and in the `finance` tenant you create APIs related to payments. At the same time, you can restrict who has access to every tenant independently.

The default tenant is called `3scale`, and you can create as many tenants as you want depending on your needs.

Tenants are covered later in this course.

Describing the API Gateway

The *API Gateway* receives the actual external traffic and redirects or denies requests based on the rules provided by the API Manager. The API Gateway caches the API Manager information regarding routing and limits, so it can work independently in the case of a failure in the API Manager.

The implementation of the API Gateway is *APIcast*, an open-source NGINX-based API Gateway. The default 3scale installation comes with two APIcast instances: one for a `staging` environment and another one for a `production` environment.

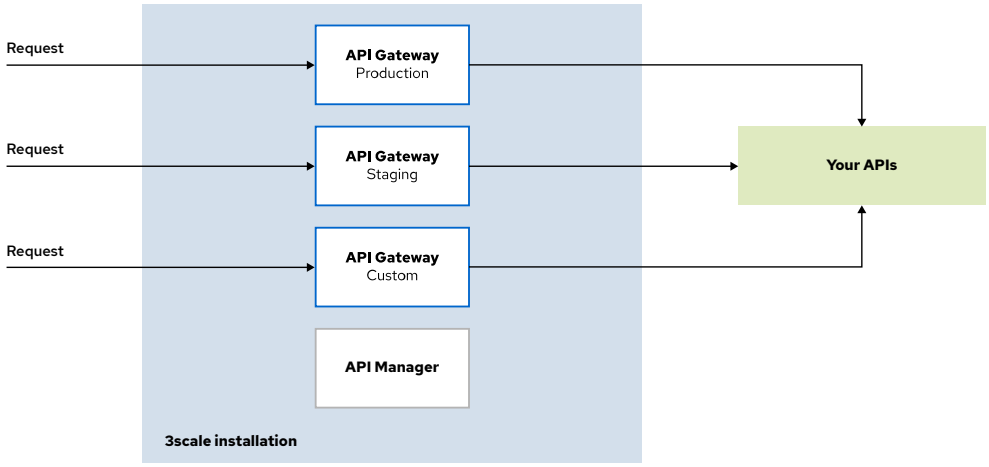


Figure 1.6: API Gateways architecture within the 3scale installation

APIcast is covered later in this course.

References

Red Hat Developer - Red Hat 3scale API Management
https://developers.redhat.com/products/3scale/overview

Guided Exercise

Creating a Lab Environment and Configuring the classroom environment

In this exercise, you will configure your development environment to work with 3scale, which runs on OpenShift.

Outcomes

You should be able to:

- Log in to your lab environment.
- Log in to OpenShift.
- Clone the `DO240-apps` GitHub repository.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start install-principles
```

Instructions

1. Log in to RHOCP.

   1.1. In a command-line terminal, run the `oc login` command with the following credentials:

   - **Username:** admin
   - **Passsword:** redhat

```
[student@workstation ~]$ oc login -u admin -p redhat api.ocp4.example.com:6443
Login successful.

...output omitted...
```

2. Clone the `DO240-apps` GitHub repository. This repository contains scripts and applications that you will use throughout the course.

   2.1. Run the `git clone` command to clone the repository.

```
[student@workstation ~]$ git clone \
 https://github.com/RedHatTraining/DO240-apps.git
...output omitted...
```

2.2. To execute the scripts later in the course, grant execution permissions to the files in the `scripts` directory.

```
[student@workstation ~]$ chmod +x DO240-apps/scripts/*
...output omitted...
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise.

```
[student@workstation ~]$ lab finish install-principles
```

This concludes the guided exercise.

Installing Red Hat 3scale API Management

Objectives

After completing this section, you should be able to install Red Hat 3scale API Management on OpenShift and install the 3scale Toolbox CLI.

Getting Red Hat Registry Credentials

Both the installation of the 3scale API Management and the 3scale Toolbox CLI require access to the Red Hat Registry (`registry.redhat.io`) to pull several container images. You can use your Red Hat account to get a valid username and token.

The `https://access.redhat.com/terms-based-registry/` web page displays your current Red Hat Registry Service Accounts, if you have any. Red Hat recommends creating a new Registry Service Account for the 3scale installation. Then, you can use those credentials during the installation process.

Installing 3scale API Management

Before you install 3scale, you should consider your system requirements, and create a Service Account for the Red Hat Registry.

System Requirements

Depending on your OpenShift installation and your application's workload, your system requirements might vary. Review the recommended parameters in 3scale official documentation [https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/installing_3scale/index#system-requirements-for-installing-threescale-on-openshift].

Container Registry Authentication

Most of the container images used by 3scale are stored in the Red Hat Registry. To download images from the Red Hat Registry, your OpenShift cluster must be authenticated.

You can authenticate your OpenShift cluster by creating an OpenShift `docker-registry` secret. This secret contains the username and token that is used when pulling images from the Red Hat Registry. Use the Red Hat Registry Service Account credentials.

```
[user@host ~]$ oc create secret docker-registry \
  threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username=REDHAT_USERNAME \ (1)
  --docker-password=REDHAT_TOKEN (2)
```

(1) Your Red Hat Registry Service Account username

(2) Your Red Hat Registry Service Account token

Then, link the secret to your OpenShift service account for pulling container images. You can link it to the `default` OpenShift service account.

```
[user@host ~]$ oc secrets link default \
 threescale-registry-auth --for=pull
```

Link the secret to the `builder` OpenShift service account for pushing and pulling build images.

```
[user@host ~]$ oc secrets link builder threescale-registry-auth
```

Installing 3scale API Management Through the Operator

The 3scale operator is available in the OperatorHub of OpenShift. The operator facilitates the installation of specific versions of 3scale and future updates.

The 3scale operator provides several custom resources used to deploy some of the product features. After installing the operator, you must deploy a new `APIManager` custom resource. The `APIManager` custom resource is used to deploy the 3scale API Management product itself. The following snippet is an `APIManager` example:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-sample
  namespace: 3scale (1)
spec:
  wildcardDomain: example.com (2)
```

(1) The namespace where the `APIManager` custom resource is deployed.

(2) The domain that is used to generate all the 3scale routes.

Note

Red Hat recommends to both install the operator and deploy the `APIManager` custom resource in a custom project. For example, in the previous snippet, the `3scale` project is used.

The `wildcardDomain` field must be a resolvable domain. Usually, it is the domain where your OpenShift cluster is running. For example, if your OpenShift cluster is running in `yourdomain.com`, then the deployment of the `APIManager` custom resource generates routes such as `3scale-admin.yourdomain.com`.

The most frequent issues that you might face when installing 3scale are:

Issue	How to identify it	Resolution
OpenShift cannot pull container images from the Red Hat Registry.	Some pods showing an error similar to `ErrImagePull` or `ImagePullBackOff`.	Review the `Container registry authentication` section of this document. Refer to the 3scale documentation [https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/installing_3scale/index#configuring-container-registry-authentication].

Issue	How to identify it	Resolution
Not enough memory and/or CPU to deploy 3scale	Some pods cannot get to the `Running` or `Completed` status after several minutes	Increase the memory and/or CPU of your OpenShift nodes.

Installing the 3scale Toolbox CLI

The 3scale installation provides several graphical interfaces to configure and manage most of the features of the product. If you want to use a command-line utility instead, then 3scale provides the 3cale Toolbox CLI.

You can use the Toolbox by running the official container image provided by 3scale, `registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11`. Because the container image is stored in the Red Hat Registry, you must log in with your Red Hat Registry credentials.

```
[user@host ~]$ podman login registry.redhat.io
Username: redhat_registry_username
Password: redhat_registry_token
```

Then, you can pull the image.

```
[user@host ~]$ podman pull \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11
Trying to pull registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11...
...output omitted...
```

You can use `podman run` to execute the Toolbox by using the following format:

```
podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale COMMAND
```

For example, if you want to execute the `help` command.

```
[user@host ~]$ podman run \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale help
NAME
    3scale - 3scale toolbox

USAGE
    3scale <sub-command> [options]
...output omitted...
```

Because executing `podman run` is too long, you can create a bash alias.

```
[user@host ~]$ alias 3scale="podman run \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale"
```

Adding a Remote to the Toolbox

A single 3scale installation can have more than one tenant, which encapsulates separate configurations.

To use the Toolbox, you must configure one or more remotes. A Toolbox remote holds the credentials to interact with a 3scale tenant. The following is an example of a Toolbox remote.

```
https://ACCESS_TOKEN@TENANT_ADMIN_PORTAL_URL
```

The `TENANT_ADMIN_PORTAL_URL` variable is your tenant's Admin Portal URL. The default tenant created when you install 3scale is `3scale`, therefore, the URL of the default tenant's Admin Portal is `3scale-admin.`*`YOUR_DOMAIN`*. The `ACCESS_TOKEN` variable is used to authenticate the remote against the tenant. You can create a new access token by using the Admin Portal of the tenant.

Then, you can use the `remote add` command to add the remote to the Toolbox.

```
[user@host ~]$ 3scale remote add 3scale-tenant -k \
https://ACCESS_TOKEN@TENANT_ADMIN_PORTAL_URL
```

Note

The `-k` option, provided in the previous command, allows connections to 3scale without an SSL certificate. Because the container created by `podman run` does not hold a proper certificate, you must add the `-k` option.

Because the `podman run` command creates a new container on every execution, the remote data is not kept between executions. You can solve this issue by creating a new container image that contains the remote information. This way, `podman run` creates a new container on every execution with an image that holds the remote.

For example, you can use `podman run` to create a named container that contains the remote information.

```
[user@host ~]$ podman run --name=toolbox-original \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale remote \
add 3scale-tenant -k https://ACCESS_TOKEN@TENANT_ADMIN_PORTAL_URL
```

Then, you can use `podman commit` to create a new image, `toolbox`, from the named container. Because the previous created container holds the remote information, the new image contains it too.

```
[user@host ~]$ podman commit toolbox-original toolbox
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

Now, you can update the `3scale` bash alias to use the `toolbox` image.

```
[user@host ~]$ alias 3scale="podman run toolbox 3scale -k"
```

Note

To persist this alias across sessions, add it to your user's `~/.bashrc` file.

References

Red Hat 3scale official documentation - Installing 3scale on OpenShift

https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/installing_3scale/index#install-threescale-on-openshift-guide

Guided Exercise

Installing Red Hat 3scale API Management

In this exercise, you will install Red Hat 3scale API Management by using the 3scale OpenShift operator, and then install the 3scale Toolbox CLI.

Outcomes

You should be able to:

- Install Red Hat 3scale API Management on OpenShift by using the 3scale OpenShift operator.
- Install the 3scale Toolbox CLI.

Before You Begin

You must be logged in to the `workstation` machine as the `student` user.

Run the following command to ensure that your OpenShift cluster is running properly.

```
[student@workstation ~]$ lab start install-setup
```

Instructions

1. Log in to the Red Hat OpenShift web console with an account that has cluster administrator privileges.

   1.1. Navigate to `https://console-openshift-console.apps.ocp4.example.com` to access the OpenShift web console. If a certificate errors appears, then accept the self-signed certificates.

   1.2. In the `Log in with` page, click **htpasswd_provider**.

   1.3. Log in with the following credentials:

   - **Username**: admin
   - **Password**: redhat

2. Create a new OpenShift project, where you will install the 3scale Operator.

   2.1. In the Administrator pane, click **Home** > **Projects**.

   2.2. Click **Create Project** and type `3scale` in the `Name` field. Then, click **Create**.

3. Install the 3scale OpenShift operator by using the OpenShift Console.

   3.1. In the Administrator pane, click **Operators** > **OperatorHub**.

   3.2. In the `Filter by keyword` text box, type `3scale`.

3.3. Select the `Red Hat Integration - 3scale` operator, and then click **Install**.

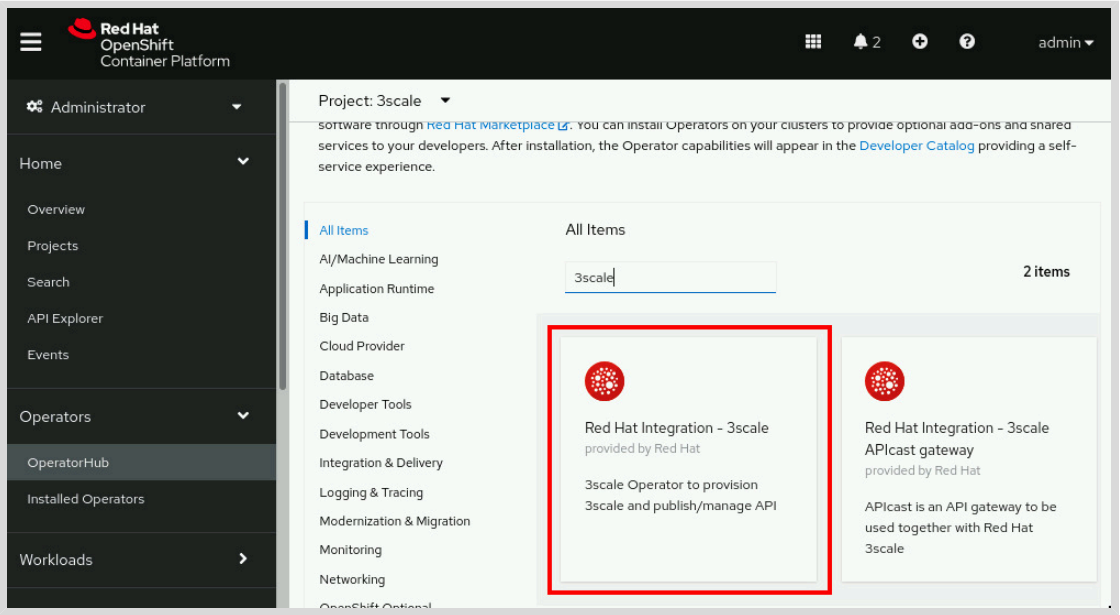


Figure 1.7: 3scale operator in the OperatorHub

3.4. Complete the form according to the following parameters, and then click **Install**.

- **Update channel**: `threescale-2.11`
- **Installed Namespace**: `3scale`
- **Update approval**: `Automatic`

The operator takes some time to install.

3.5. Click **View Operator** to access the operator details page.

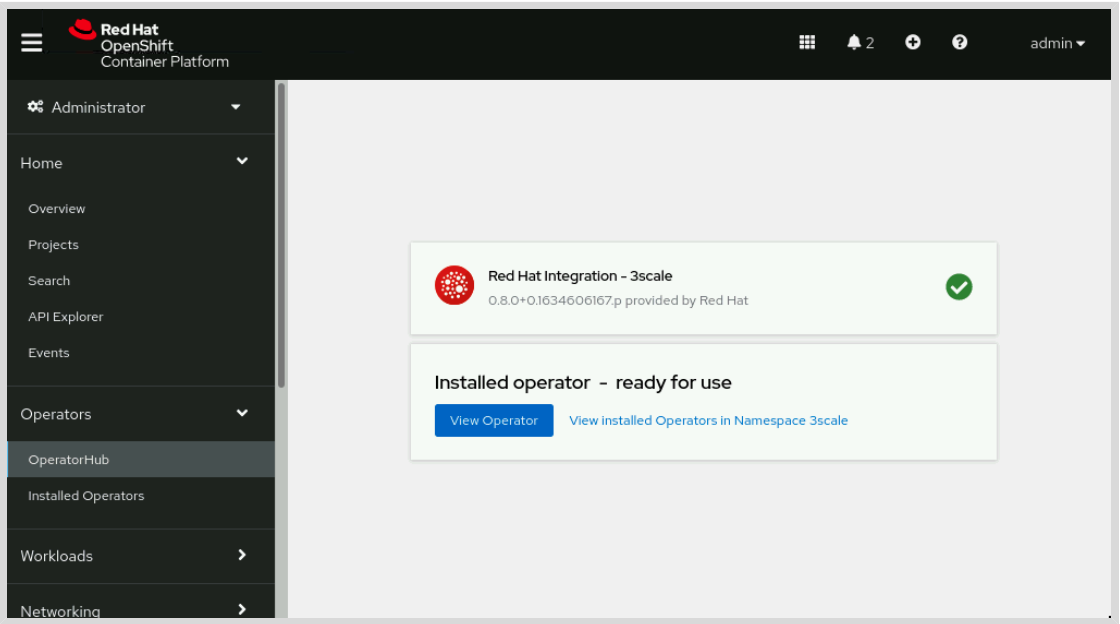


Figure 1.8: Operator successfully installed

4. Deploy the `APIManager` custom resource.

4.1. In the `Provided APIs` section, find `APIManager`. Click **Create instance**.

The `Create APIManager` form displays.

4.2. Click **YAML view** in the `Configure via` section to see the `APIManager` custom resource as a YAML file.

4.3. Update the YAML according to the following snippet.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-sample
  namespace: 3scale
spec:
  wildcardDomain: apps.ocp4.example.com
  resourceRequirementsEnabled: false
```

Note

The `wildcardDomain` field is used to create the accessible routes of 3scale. For example, the Admin Portal might be `3scale-admin.`*`wildcardDomain`*.

The `wildcardDomain` field should be a resolvable domain. Usually, it is the domain where you are running your OpenShift cluster. In the previous YAML snippet you use the `apps.ocp4.example.com` domain because it is the domain used by the OpenShift cluster in the `workstation` environment.

4.4. Click **Create**. The `APIManager` deployment creates several OpenShift resources, therefore, it takes several minutes for all the pods to be available.

In the meantime, you can proceed with the exercise by installing the 3scale Toolbox CLI.

5. Install the 3scale Toolbox CLI. The official way of installing the Toolbox is by using the 3scale Toolbox CLI container image, `registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11`.

5.1. In a web browser, navigate to `https://access.redhat.com/terms-based-registry/`. Log in with your Red Hat account if necessary. The page displays your current Red Hat Service Accounts, if you have any.

5.2. Click **New Service Account** to create a new Service Account. In the `Name` field type `do240`, and then click **Create**.

Note

Your username also includes the random string of numbers provided before the `Name` field. For example, if you provide the `do240` name, then a possible username string is `89238478|do240`.

5.3. The `Token Information` page displays your username and token. Keep this page open, as the contents are necessary later in this exercise.

5.4. In your command-line terminal, log in to the Red Hat Registry by using Podman. Use the username and token from the previous step.

```
[student@workstation ~]$ podman login registry.redhat.io
Username: username_from_previous_step
Password: token_from_previous_step
```

5.5. Pull the 3scale Toolbox CLI image from the Red Hat Registry by using Podman.

```
[student@workstation ~]$ podman pull \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11
Trying to pull registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11...
...output omitted...
```

5.6. Verify the installation. You can use `podman run` to execute the 3scale Toolbox CLI by using the following format:

```
podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale COMMAND
```

For example, if you want to execute the `help` command in the 3scale Toolbox CLI.

```
[student@workstation ~]$ podman run \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale help
NAME
    3scale - 3scale toolbox

USAGE
    3scale <sub-command> [options]
...output omitted...
```

5.7. Because using `podman run` is too long, you can create a bash alias.

```
[student@workstation ~]$ alias 3scale="podman run \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale"
```

Now you can run `3scale COMMAND`. For example, you can run the `3scale help` command.

```
[student@workstation ~]$ 3scale help
NAME
    3scale - 3scale toolbox

USAGE
    3scale <sub-command> [options]
...output omitted...
```

6. In the 3scale Toolbox CLI, add a remote for the default `3scale` tenant.

6.1. Log in to RHOCP:

```
[student@workstation ~]$ oc login -u admin -p redhat api.ocp4.example.com:6443
Login successful.

...output omitted...
```

6.2. In your command-line terminal, use the `oc get secret` command to retrieve the password for the `3scale` tenant Admin Portal.

```
[student@workstation ~]$ oc get secret system-seed -n 3scale \
--template={{.data.ADMIN_PASSWORD}} | base64 -d; echo
```

The default passwords of 3scale are stored in the `system-seed` secret. By executing the previous command, you retrieve the `ADMIN_PASSWORD` entry of the `system-seed` secret. Then, because the secret is encoded in Base64, you decode it.

6.3. Navigate to `https://3scale-admin.apps.ocp4.example.com`, the `3scale` tenant Admin Portal. Log in with the the following credentials:

- **Username**: admin
- **Password**: **Password from the `system-seed` secret**

Warning

As noted before, the deployment of the APIManager custom resource takes several minutes. If the Admin Portal is not accessible, try again in a few minutes.

If the `How does 3scale work` page displays, click **X** to close it.

6.4. In the top pane, click **Dashboard** to display the drop down menu. Then, click **Account Settings**.

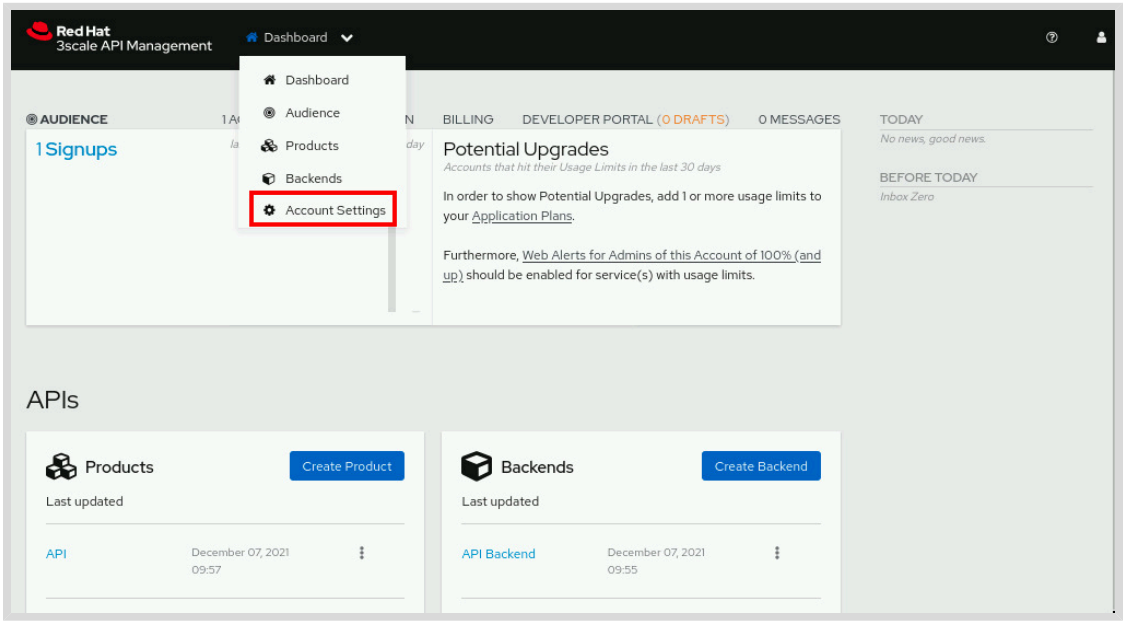


Figure 1.9: 3scale tenant administration portal

6.5. In the left pane, click **Personal** > **Tokens**.

6.6. In the `Access Tokens` section, click **Add Access Token**. Complete the form according to the following values:

- **Name:** `toolbox`
- **Scopes:** All (select `Billing API`, `Account Management API`, `Analytics API` and `Policy Registry API`)
- **Permission:** `Read & Write`

Then, click **Create Access token**. Save the access token displayed.

6.7. Using the official 3scale Toolbox CLI image, create a container with a name of `toolbox-original`. This image must contain the remote information.

```
[student@workstation ~]$ podman run --name=toolbox-original \
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.11 3scale remote \
add 3scale-tenant -k https://ACCESS_TOKEN@3scale-admin.apps.ocp4.example.com
```

Note

Because `podman run` creates a new container on every execution, the information added is not kept between executions. A solution to this issue is creating a new image from the original 3scale Toolbox CLI image that contains the remote information. This way, you use `podman run` to execute an image with the remote information on every execution.

If you make a mistake in this step, execute `podman rm toolbox-original`. Then, you can execute the preceding command again.

6.8. Create an image from the `toolbox-original` container.

```
[student@workstation ~]$ podman commit toolbox-original toolbox
Getting image source signatures
...output omitted...
Writing manifest to image destination
Storing signatures
```

6.9. Update the `3scale` bash alias to use the new image.

```
[student@workstation ~]$ alias 3scale="podman run toolbox 3scale -k"
```

Note

The classroom workstation includes a `.bashrc` file with the preceding alias configured. Future workstation sessions do not require setting the alias again.

Note that the `-k` option allows connections without an SSL certificate. Now you can list the remotes by using the `3scale` alias.

```
[student@workstation ~]$ 3scale remote list
3scale-tenant https://3scale-admin.apps.ocp4.example.com access-token
```

Finish

Run the following command

```
[student@workstation ~]$ lab finish install-setup
```

This concludes the guided exercise.

Summary

In this chapter, you learned:

- How to install 3scale via the OpenShift operator.
- How to configure your classroom workstation to administer RHOCP and Red Hat 3scale API Management.
- How to set up the 3scale Toolbox using Podman.

Chapter 2

Managing APIs with Red Hat 3scale API Management

Goal

Manage your cloud-native APIs with the Red Hat 3scale Admin Portal.

Objectives

- Create Backends and Products in the Admin Portal. Describe the process to promote APIs from Development to Production.
- Create application plans and set rate limits for APIs.
- Create and configure multiple versions of APIs.
- Discover applications deployed on OpenShift automatically.
- Create multiple tenants in Red Hat 3scale API Management.

Sections

- Creating Backends and Products (and Guided Exercise)
- Creating Application Plans and Rate Limits (and Guided Exercise)
- Managing API Versions (and Guided Exercise)
- Configuring Service Discovery (and Guided Exercise)
- Creating Multiple Tenants in Red Hat 3Scale API Management (and Guided Exercise)
- Managing APIs with Red Hat 3scale API Management (Quiz)

Creating Backends and Products

Objectives

After completing this section, you should be able to create Backends and Products in the Admin Portal. Describe the process to promote APIs from Development to Production.

Introducing 3scale API Management Backends and Products

3scale API Management acts as a configurable proxy to your applications. It also provides functionality for various API management tasks, such as billing and usage monitoring. Backends and products are two of the main types of objects in 3scale API Management.

Conceptually, you can imagine requests traversing through the various parts of your configuration.

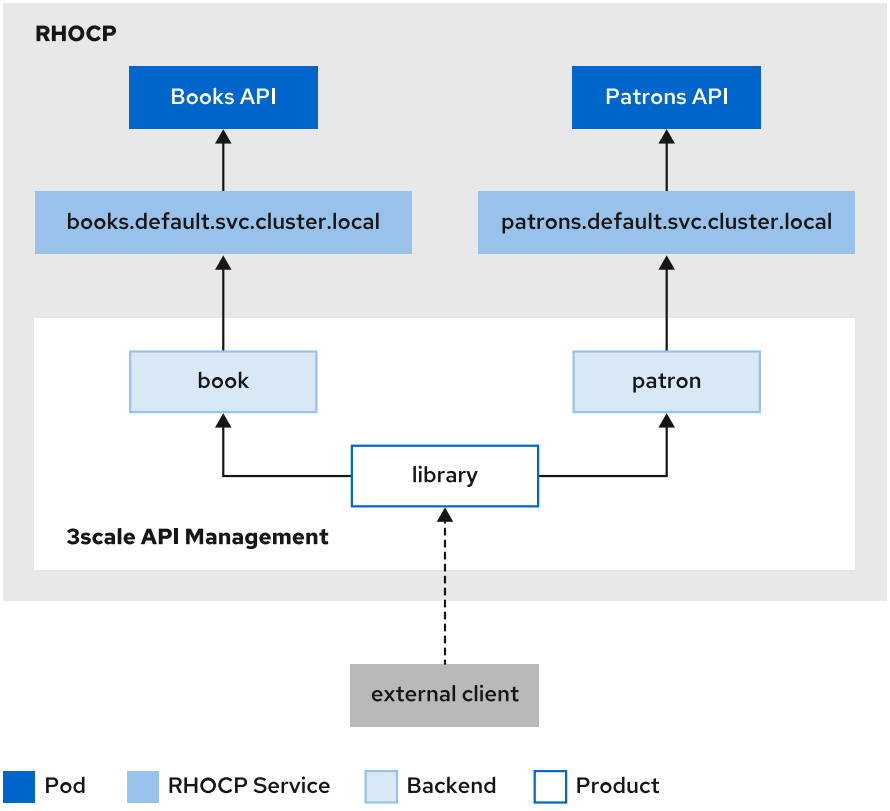


Figure 2.1: Conceptual request flow

In the preceding diagram, note that the underlying APIs are not publicly exposed.

Backends

A *backend* represents a deployed application by pointing to its root URL.

For example, a REST API is hosted at `http://myapp.example.com` and provides a `/ping` endpoint. The full path to the endpoint would be `http://myapp.example.com/ping`.

However, the URL of the 3scale backend would solely contain `http://myapp.example.com`. This allows the requester to reach multiple endpoints with a single backend for an application.

Backends are associated with one or more products. By associating a backend to a product, the product's users can access the underlying API.

Creating a Backend with the Admin Portal

Navigate to backend management with the top pane drop-down menu within the admin portal.

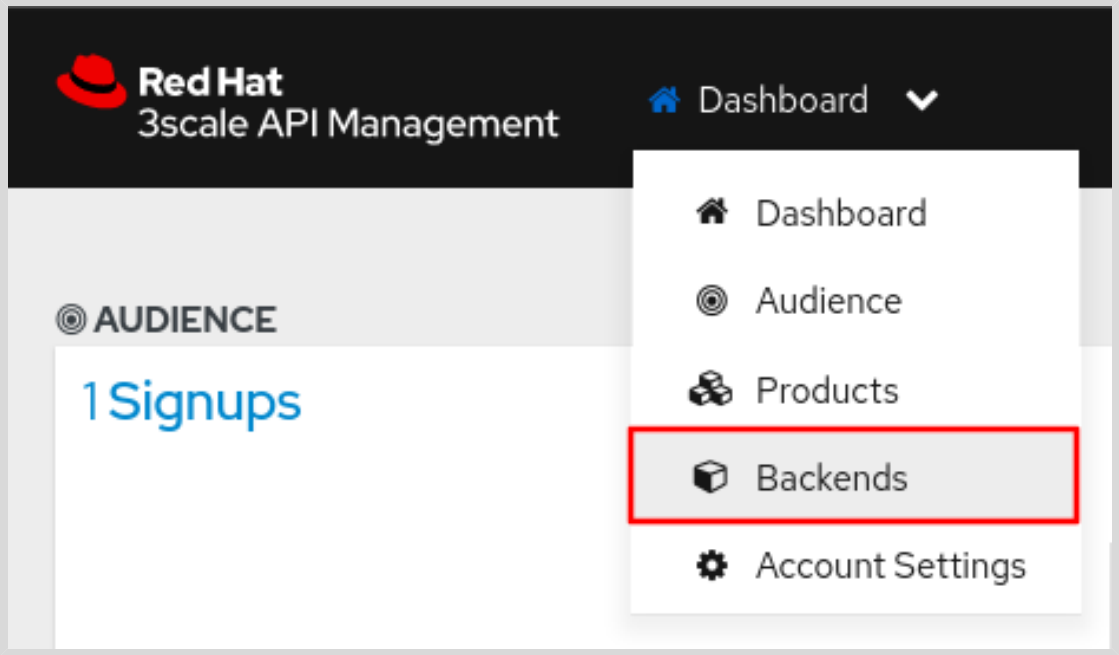


Figure 2.2: 3scale top pane navigation

From here, you can create new backends and manage existing ones.

To create a new backend, click **Create Backend**, fill out the form, and submit it.

Products

Products, previously called *services*, represent a collection of 3scale objects that pertain to a specific client or group of users.

Products contain the following 3scale configuration objects:

- Application plans
- Applications
- A map of backends to applications

Note that backends do not *belong* to a product. This is because the same backend can be associated with several different products. Different products do not need separate backends for the same underlying API.

Creating a Product with the 3scale Toolbox

Currently, the 3scale Toolbox CLI tool provides super-commands labeled `product` and `service`. Inspect the `service` super-command when you manage products, because not all of the features have been moved to the `product` super-command.

The `service` super-command provides the following sub-commands:

- Create
- Copy
- Delete
- Show
- List
- Apply

Note that `3scale service apply` is used to modify the service.

For example, to create a new product, use the following command:

```
[user@host ~]$ 3scale service create \ (1)
3scale-tenant \ (2)
library-product (3)
```

(1) Specifies that you want to create a new product.

(2) Puts the product in the `3scale-tenant` tenant.

(3) Names the product `library-product`.

Managing a Product with the Admin Portal

Navigate to product management with the top pane drop-down menu within the admin portal.

From here, you can create new products and manage existing ones.

After selecting a product to work within, the left-hand pane is used to navigate between the management pages for components within that product.

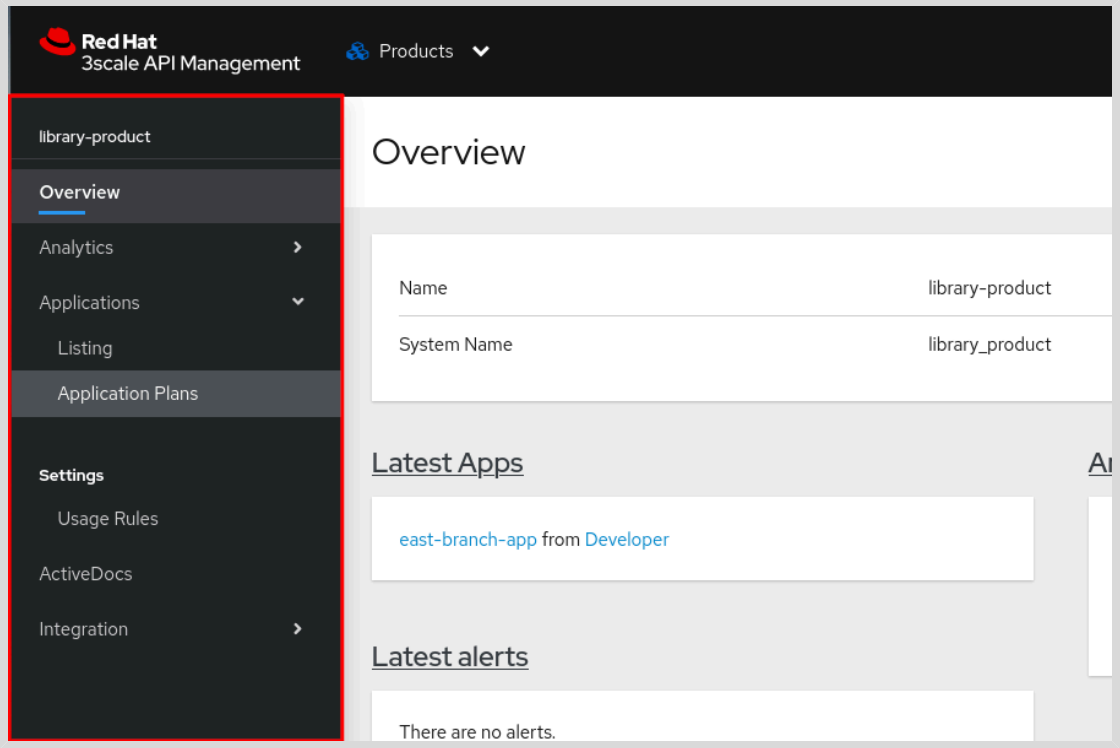


Figure 2.3: Product left pane navigation.

For example, in the previous screenshot, the `library-product` is selected. Click **Applications** > **Application Plans** to manage application plans within that product.

Associating Backends with Products by using the Admin Portal

To use a backend within a product, you must first associate the two. Within the product section in the admin portal, navigate to **Integration** > **Backends**.

This page shows the list of backends currently accessible by the product, their URLs, and the product path to access them. To add an existing backend to the product, click **Add Backend**, select the backend, and provide a path. This path is a prefix that maps to the backend.

Note that backend paths must be unique within a product, as they are what the router uses to direct requests to the correct backend.

Promoting Configurations to Staging and Production

Within the product section in the admin portal, navigate to **Integration** > **Configuration**. This page shows the status of your staging and production environments.

Changes you make to a product are not live until you promote the configuration. If you have any changes not yet in the staging environment then the **Configuration** item shows a warning icon.

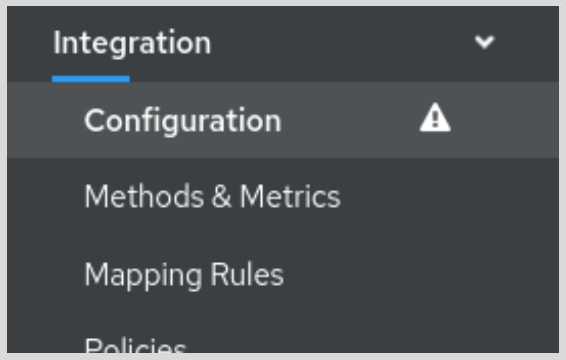


Figure 2.4: A warning icon shows that there are changes to be promoted.

The two environments are *staging* and *production*. Each environment can have separate configurations, but changes must go to the staging environment first.

To promote configuration changes to the staging environment, click **Promote v. *X* to Staging APIcast**, where *X* is an auto-incrementing version number.

After promoting changes to staging, you can then promote them to production by clicking **Promote v. *X* to Production APIcast**.

References

For more information, refer to the *Adding backends to a product* chapter in the *Red Hat 3scale API Management Getting Started Guide* at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.8/html/getting_started/first-steps#adding-backends-product

Guided Exercise

Creating Backends and Products

In this exercise, you will deploy a set of microservices for use in a regional library system. To expose these APIs, you will create one or more of the following resources:

- Backends
- Products
- Applications
- Application plans

Outcomes

You should be able to:

- Deploy microservices on Red Hat OpenShift Container Platform (RHOCP) in a way that is consumable by 3scale API Management.
- Create backends and products.
- Connect those backends and products to an application and corresponding application plan.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start applications-products
```

Instructions

1. Create a product called `library_product`. Within that product, create an application and application plan.

   1.1. Use the 3scale Toolbox to create a product named `library_product`:

```
[student@workstation ~]$ 3scale service create 3scale-tenant library_product
Service 'library_product' has been created with ID: 1
```

Note

In 3scale API Management, a *product* and a *service* are the same.

   1.2. Create an application plan named `east-branch-plan`:

```
[student@workstation ~]$ 3scale application-plan \
create 3scale-tenant library_product east-branch-plan
Created application plan id: 16. Default: false; Disabled: false
```

Note that the product is referenced via its system ID `library_product`.

1.3. Find the ID of the example developer named `John`:

```
[student@workstation ~]$ 3scale account find 3scale-tenant john
org_name => Developer
id => 3
```

1.4. Create an application named `east-branch-app` that references the `east-branch-plan` application plan and the example developer:

```
[student@workstation ~]$ 3scale application \
create 3scale-tenant 3 library_product east_branch_plan east-branch-app
Created application id: 9
```

2. Create a backend for each of the APIs.

2.1. Using a web browser, log in to the Admin Portal as the `admin` user. The portal is located at `https://3scale-admin.apps.ocp4.example.com`.

Note

If this is your first time accessing the admin portal, a setup wizard opens. Use the **X** button to exit this wizard.

2.2. From the top pane drop-down menu, select **Backends** to view the list of backends.

2.3. Create a backend for the book API by clicking **Create Backend**, providing the following values, and submitting the form. Leave all other fields as their default values.

Field	Value
Name	`book-backend`
Private Base URL	`http://books-api.book-library.svc.cluster.local:8080`

2.4. Create another backend for the patron API with the following values:

Field	Value
Name	`patron-backend`
Private Base URL	`http://patrons-api.book-library.svc.cluster.local:8080`

3. Associate `book-backend` and `patron-backend` with the `library_product`.

3.1. From the top pane drop-down menu, select **Products** to view the list of products. Click the name of the **library_product** product in the list.

3.2. Navigate to the list of backends associated with the product by selecting **Integration** > **Backends**. Associate a backend with the product by clicking **Add Backend**. Select `book-backend` as the backend, provide the unique path `/b-api`, and submit the form.

3.3. Repeat the previous step for the patron backend, but select the `patron-backend` and enter `/p-api` as the path.

4. Promote and test the product configuration.

4.1. Navigate to the product configuration by selecting **Integration** > **Configuration**.

4.2. Promote the product's configuration to the staging environment by clicking **Promote v.1 to Staging APIcast**.

4.3. Promote the product's configuration to the production environment by clicking **Promote v.1 to Production APIcast**.

4.4. From the `Production APIcast` section, copy the hostname from the URL value. In the command line, store this value in an environment variable called `API`, to use in a later step.

```
[student@workstation ~]$ API=https://library-product-3scale-apicast-production\
.apps.ocp4.example.com:443
```

4.5. From the `Staging APIcast` section, copy the `user_key` value from the example `curl` command. In the command line, store this value in a USER_KEY environment variable.

```
[student@workstation ~]$ USER_KEY=02e852852327972ebc6edfae27b9037a
```

Note

Your `user_key` will differ from the one listed here.

4.6. Retrieve the book list from the book application.

```
[student@workstation ~]$ curl -s $API/b-api/books?user_key=$USER_KEY | jq
[
  {
    "title": "Frankenstein",
    "authorName": "Mary Shelley",
    "year": 1818,
    "copies": 10
  },
  {
    "title": "A Christmas Carol",
    "authorName": "Charles Dickens",
    "year": 1843,
```

```
    "copies": 5
  },
  {
    "title": "Pride and Prejudice",
    "authorName": "Jane Austen",
    "year": 1813,
    "copies": 3
  }
]
```

4.7. Retrieve the patron list from the patron application.

```
[student@workstation ~]$ curl -s $API/p-api/patrons?user_key=$USER_KEY | jq
[
  {
    "name": "John Smith",
    "numBooks": 3,
    "cardNumber": "00001"
  },
  {
    "name": "Patty Jones",
    "numBooks": 1,
    "cardNumber": "00002"
  },
  {
    "name": "Frank Doe",
    "numBooks": 8,
    "cardNumber": "00003"
  }
]
```

4.8. Run the previous command several times to generate request metrics.

5. Review the endpoint metrics within the Admin Portal.

5.1. In the admin portal, navigate to **Analytics** > **Traffic**.

5.2. Use the selection menu to adjust the scale of the graph by selecting **Hits (hits)**. Make sure **24 hours** is selected so that the time scale is correct.

Note that your graph could look slightly different based on how many times you ran the `curl` command.

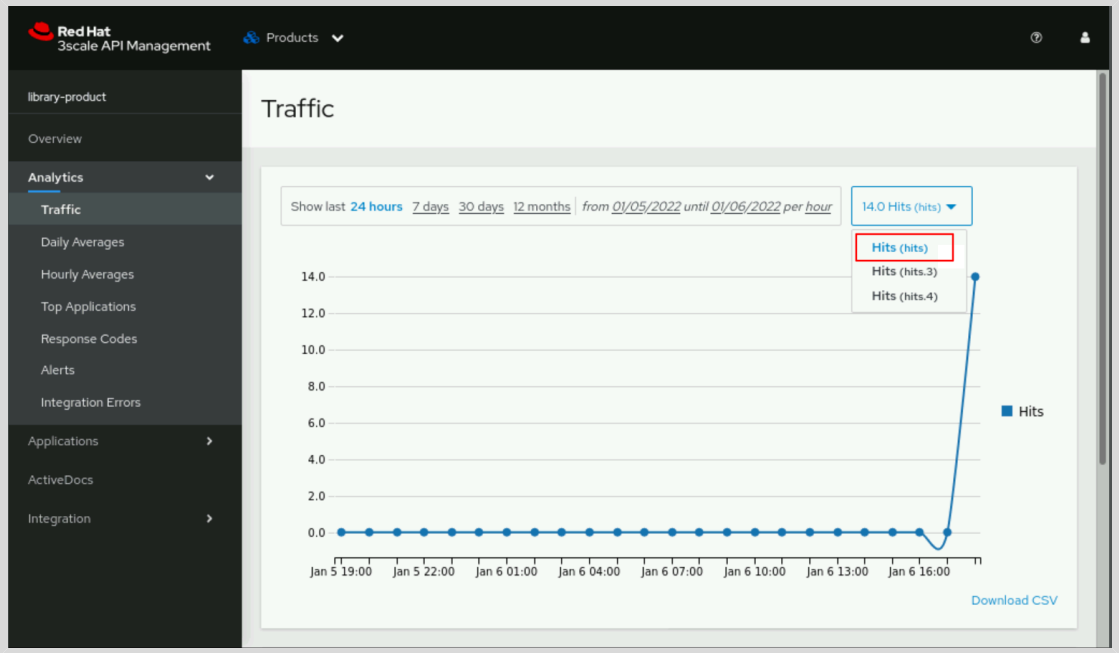


Figure 2.5: Graph scale selection menu

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish applications-products
```

This concludes the guided exercise.

Creating Application Plans and Rate Limits

Objectives

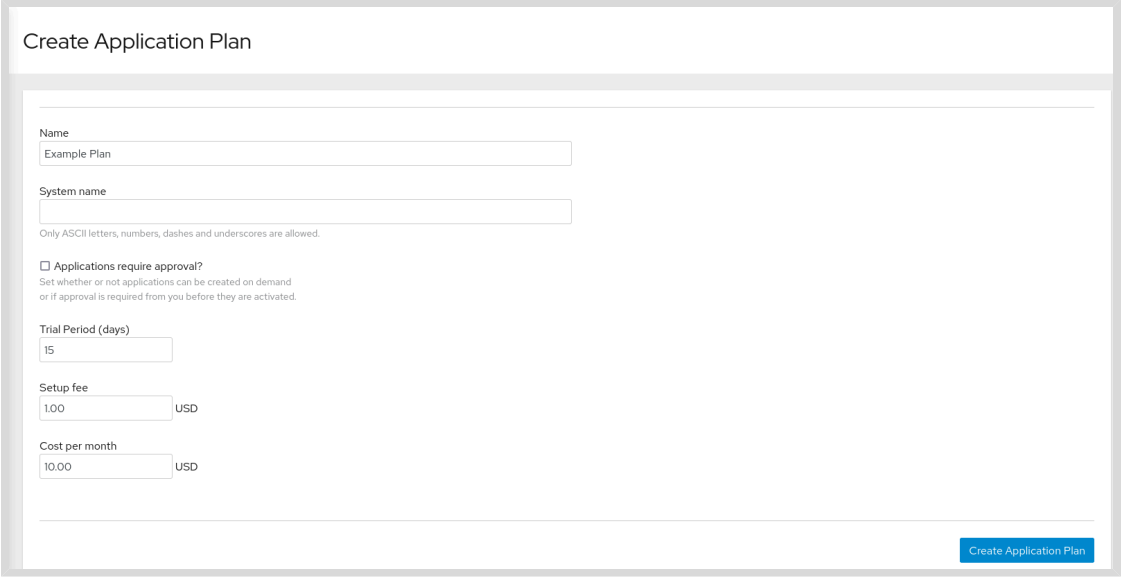
After completing this section, you should be able to create application plans and set rate limits for APIs.

Creating Application Plans using the GUI

Application plans are sets of access rights that allow users to define different rules for consumers of managed APIs. You can use application plans to set rate limits, enable features, resources, and methods for API users.

To create an application plan from the GUI:

- On your product detail page, navigate to **Applications** > **Application Plans**.
- Click the **Create Application Plan button** to go to the **Create Application Plan** section.
- Set a name and a system name for the new plan. System names must be unique because they are used by Red Hat 3scale API Management as unique identifiers.
- Select the **Applications require approval?** check box if applications need to be evaluated by the administrator before they become available.



To set an application plan as default:

- On your product detail page, navigate to **Applications** > **Application Plans**.
- Use the drop-down list below **Default Plan**, select the plan you want to set as default.

Provisioning Paid Plans

3scale API Management allows users to monetize APIs by the definition of subscription fees for accounts, products and backends. Pricing tiers are used on application plans for creating paid plans.

Pricing Models

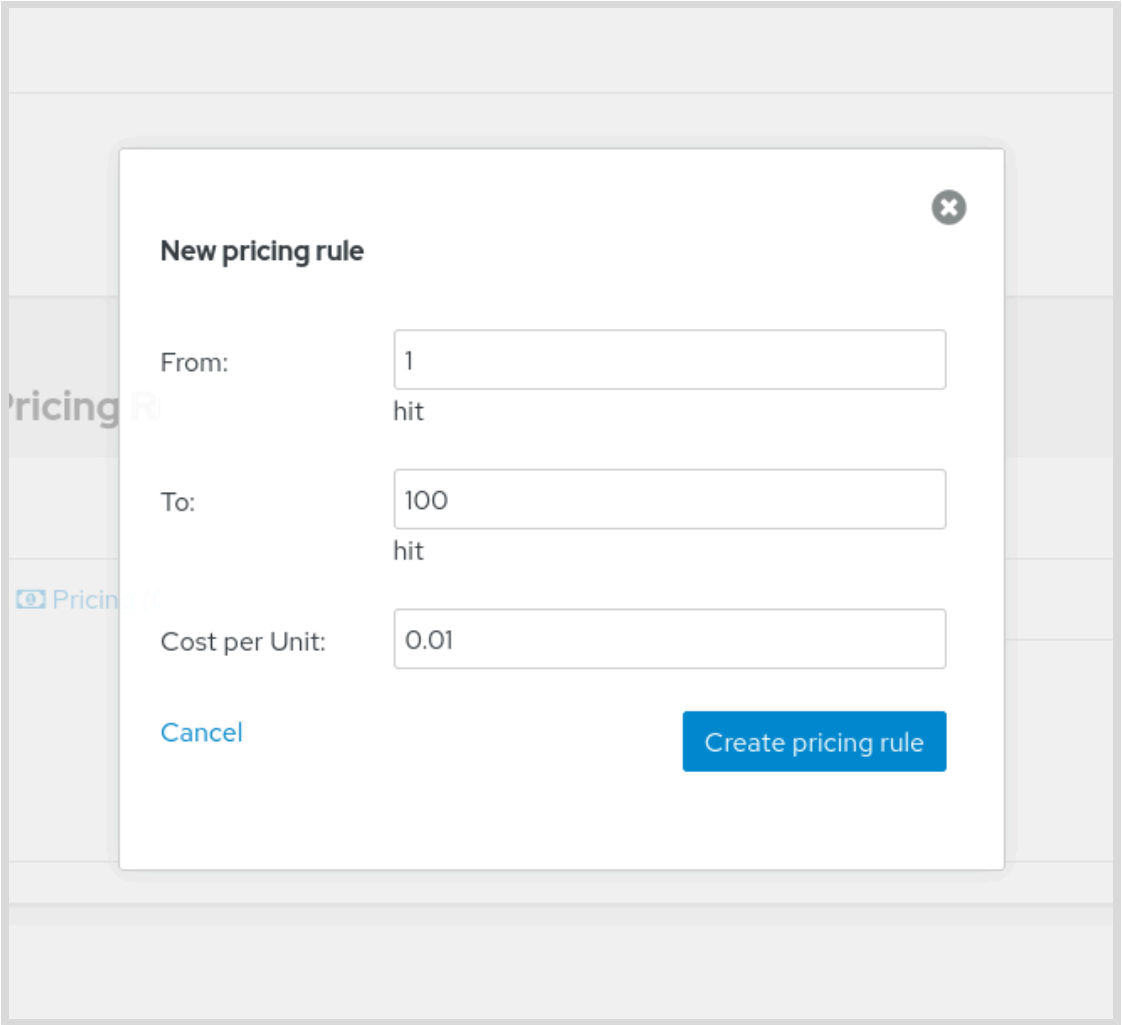
Pricing models are strategies that enable customers to pay for the return value they can get. Pricing models can be configured for volume, functionality and resources.

- Volume usage, you can assign global request count for calls on products or at the method level.
- Functionality, depending on the tier some features of the product can be enabled or disabled.
- Resources, you can create tiers for resource levels such as: bandwidth, number of users or transaction values.

Configure Pricing Rules for Application Plans

- On your product page, navigate to **Applications** > **Application Plans**
- Click on an application plan name.
- Navigate to **Metrics, Methods, Limits & Pricing Rules** where you can define pricing rules for total requests or in a more granular way at the method level.

You can define pricing rules for products in the **Product Level** section. In this section you can configure a range of requests and assign costs per unit.



Creating Application Plans by using the 3scale Toolbox CLI

You can use the toolbox client to create application plans, this is useful for automation or integration with third party applications.

To create an application plan using the toolbox CLI use the `application-plan create` option and set the `--approval-required`, `--cost-per-month`, `--setup-fee`, `--trial-period-days` for the tenant application plan. In this example the tenant name is: `3scale-tenant` and the application plan is: `new_plan`.

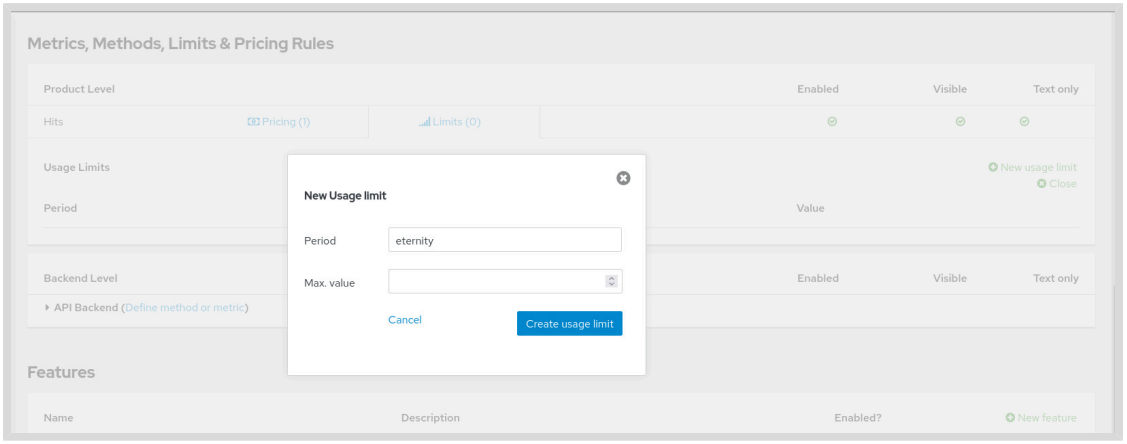
```
[student@workstation ~]$ 3scale application-plan create -p \
  --approval-required=false --cost-per-month=10.00 --setup-fee=1.0 \
  -t "new_plan" --trial-period-days=15 3scale-tenant 2 "Example Plan"
Created application plan id: 9. Default: false; Disabled: false
```

Managing Rate Limits

Rate limits enable you to restrict access to API resources, products, and backends. Application plans are also used to configure rate limits.

Specifying Rate Limits

- On your product page, navigate to **Applications** > **Application plan**.
- Click on the name of the application plan to configure.
- Navigate to **Metrics, Methods, Limits and Pricing Rules** by scrolling down.
- Click on **Limits**.
- Assign limits on the product or the backend level.
- Click on **Update Application plan** to save your changes.



References

For more information, refer to the *Application Plans* chapter in the *Red Hat 3Scale Api Management Admin Portal Guide* at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/admin_portal_guide/index#application-plans

Guided Exercise

Creating Application Plans and Rate Limits

In this exercise, you will manage application plans and rate limits.

Outcomes

You should be able to:

- Create application plans
- Provision paid plans
- Configure application plans rate limits
- Add application plans and rate limits to backends and applications
- Verify that application plans and rate limits work properly for backends and applications

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the 3scale API is online and deploys the `books` application.

```
[student@workstation ~]$ lab start applications-plans
```

Instructions

1. Create an application plan by using the toolbox CLI.

   1.1. Use the `3scale remote` command to get the list of 3scale tenants. Select the desired tenant to add an application plan in this case the remote name is: `3scale-tenant`.

```
[student@workstation ~]$ 3scale remote list
3scale-tenant https://3scale-admin.apps.ocp4.example.com 6GWLrvFdsKIIQxYs
```

Note

The 3scale client configuration should be performed as indicated in the Chapter 1 section: "Installing Red Hat 3scale API Management".

   1.2. List the services on the `3scale-tenant` and identify the `API` ID.

```
[student@workstation ~]$ 3scale service list 3scale-tenant
ID      NAME    SYSTEM_NAME
2       API     api
```

1.3. Add a new application plan to the `API` service with the following settings:

- Name: `Example Plan`.
- Trial Period (days): `15`.
- Setup Fee: `1.00`.
- Cost Per Month: `10.00`.

```
[student@workstation ~]$ 3scale application-plan create -p \
  --approval-required=false --cost-per-month=10.00 --setup-fee=1.0 \
  -t "example_plan" --trial-period-days=15 3scale-tenant 2 "Example Plan"
Created application plan id: 9. Default: false; Disabled: false
```

1.4. Verify that the new application plan was created successfully for the `3scale-tenant` service ID 2.

```
[student@workstation ~]$ 3scale application-plan list 3scale-tenant 2
ID      NAME    SYSTEM_NAME
7       Basic   basic
8       Unlimited       unlimited
9      Example Plan    example_plan
```

2. Add a new backend using the GUI.

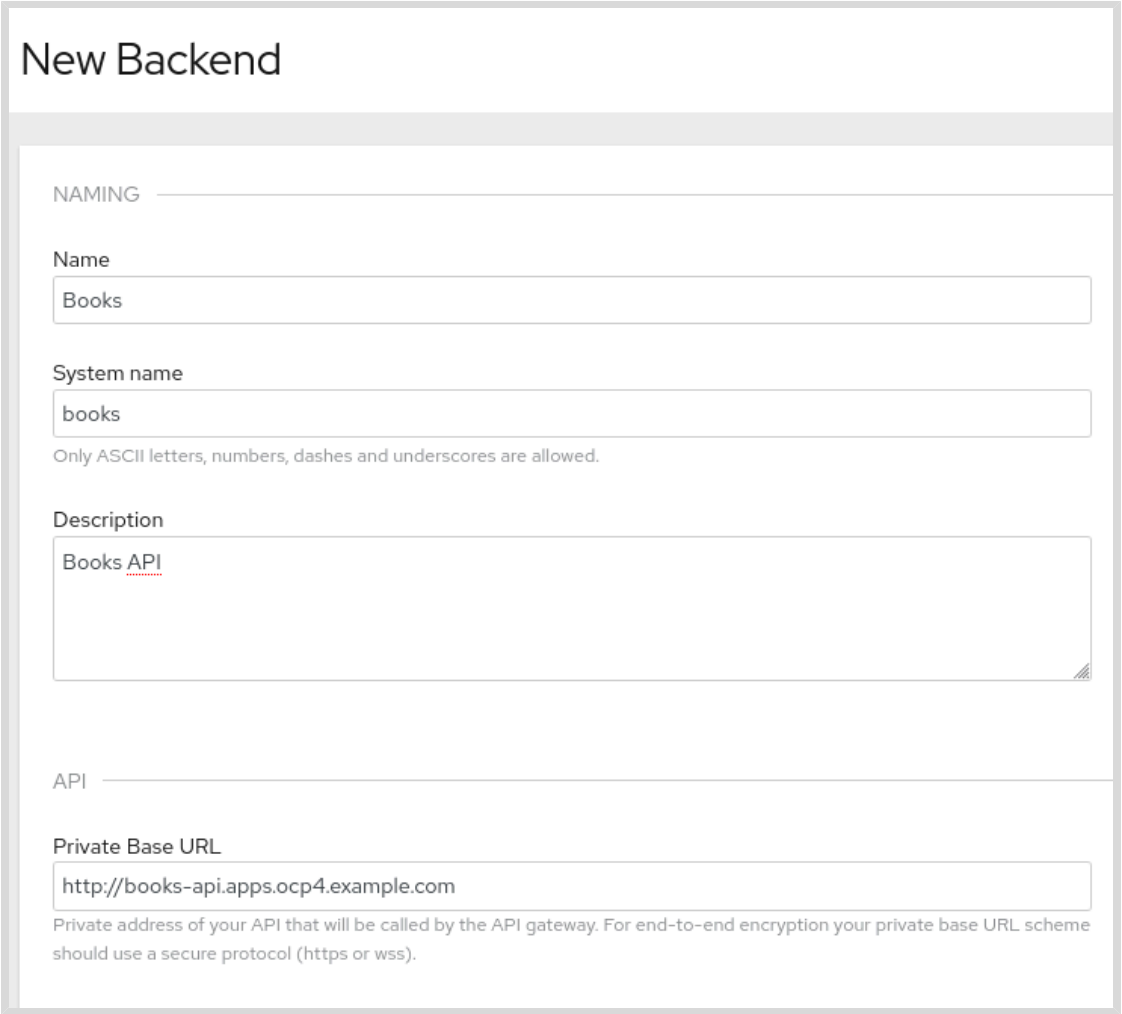
2.1. Verify that the books API application is available at the `http://books-api.apps.ocp4.example.com` URL.

```
[student@workstation ~]$ curl http://books-api.apps.ocp4.example.com/ping
pong[student@workstation ~]$
```

2.2. Navigate to **Backends** and click the **Create Backend** button.

2.3. On the `New Backend` section create a backend with the following values:

- **Name**: `Books`
- **System name**: `books`
- **Description**: `Books API`
- **Private Base URL**: `http://books-api.apps.ocp4.example.com`

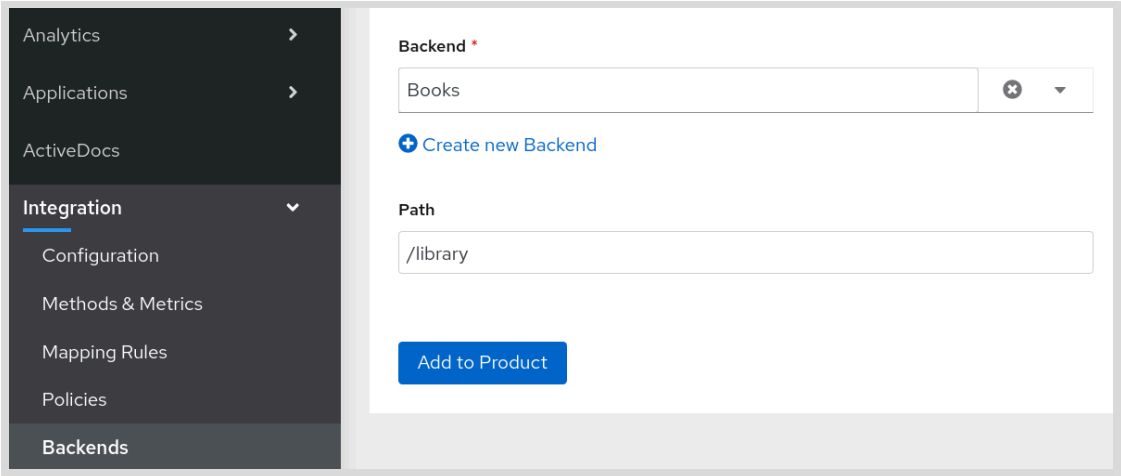


3. Add backend to the `API` product using the GUI.

   3.1. Navigate to **Products** and follow the **API** product link.

   3.2. Navigate to the **API** > **Integration** > **Backends** section and click the **Add Backend** button.

   3.3. Select the `Books` backend from the **Backend** drop-down, set the **Path** field to `/library`, and click the **Add to Product** button.



4. Promote the `API` backends to `Staging APIcast`.

   4.1. Navigate to the **API** > **Integration** > **Configuration** section.

   4.2. Click the **Promote v. 1 to Staging APIcast** button to promote the `API` backends to `Staging APIcast`.

   Your version number could differ if you previously promoted the `API` product.

   4.3. Copy the `Example curl for testing` command. This `curl` command has a `user_key` parameter containing the authentication token for the staging API.

   4.4. Verify the backend. On the terminal paste the `curl` command copied on the previous step and add the `library/books` path to the URL and execute it.

```
[student@workstation ~]$ curl \
  "https://api-3scale-apicast-staging.apps.ocp4.example.com:443/library/books/?
user_key=USER_KEY"
[{"title":"Frankenstein","authorName":"Mary Shelley","year":1818,"copies":10},
{"title":"A Christmas Carol","authorName":"Charles
 Dickens","year":1843,"copies":5},{"title":"Pride and
 Prejudice","authorName":"Jane Austen","year":1813,"copies":3}]
[student@workstation ~]$
```

5. Configure rate limits for the `Books` backend.

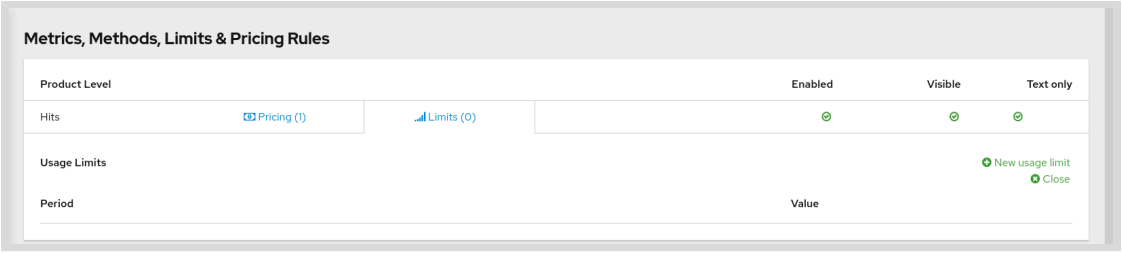
   5.1. Navigate to **API** > **Overview** > **Applications** > **Application plans**.

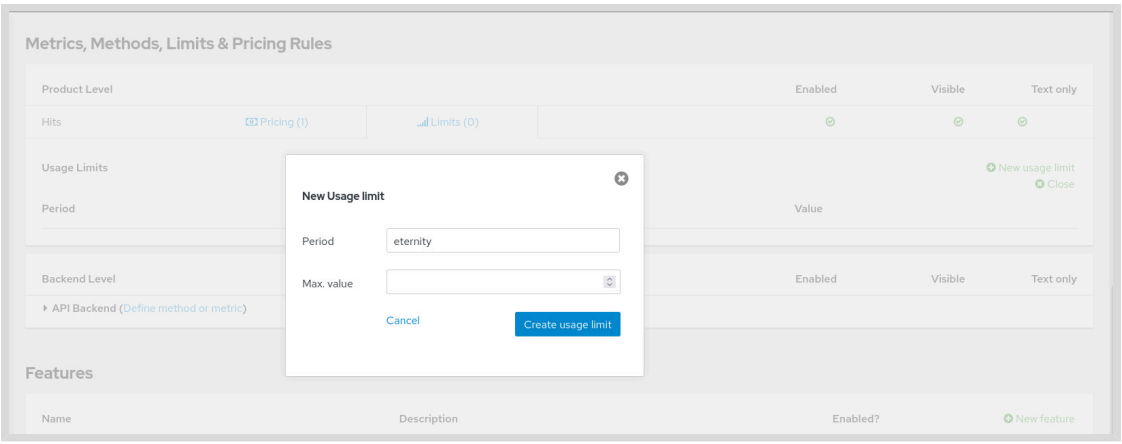
   5.2. Identify **Example Plan** and click on it.

   5.3. Scroll down to to **Metrics, Methods, Limits & Pricing Rules**.

   5.4. On the **Backend Level** section, click on the **Books** backend to expand the pricing and limits section.

   5.5. Click the **Limits** link to expand the limits section and click on the **New usage limit** link.





5.6. Set the following values to the fields of the **New Usage limit** modal.

- **Period**: `minute`
- **Max. Value**: 5

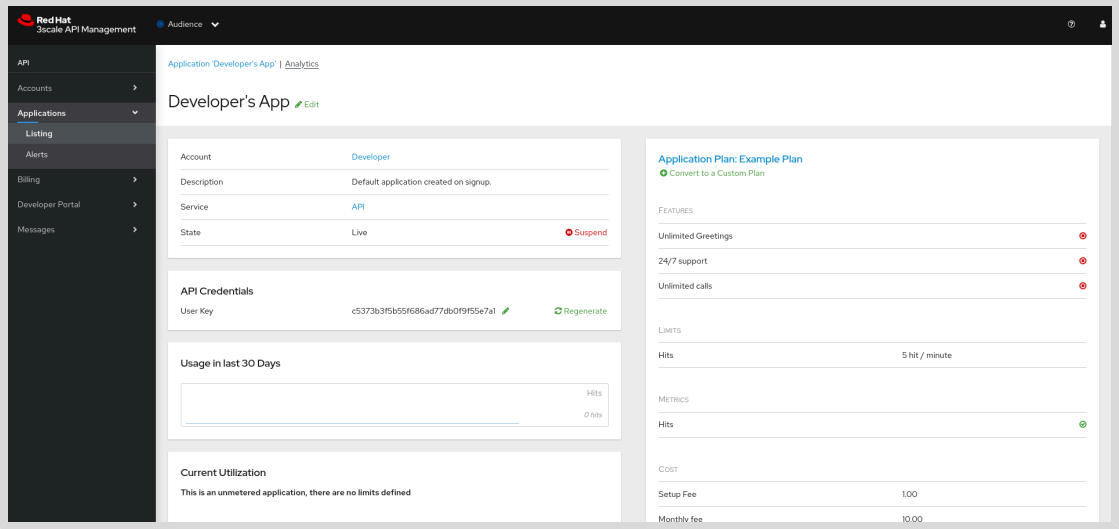
5.7. Click the **Create usage limit** button to create the new usage limit.

6. Change the application plan of the `Developer's App`

6.1. Navigate to the products management of the top pane drop-down menu and click on **API** > **Applications** > **Listing**.

6.2. Click **Developer's App**.

6.3. On the **Change plan** section select: **Example Plan** and click the **Change plan** button.



7. Add a mapping rule to the `Books` backend.

7.1. Navigate to **Backends** and click the **Books** backend.

7.2. Navigate to **Books** > **Mapping Rules** and click on the **Create Mapping Rule**.

7.3. Set the following values for the new mapping rule and submit the form.

- **Verb**: `GET`
- **Pattern**: `/books`
- **Metric**: `Hits`
- **Increment by**: `1`

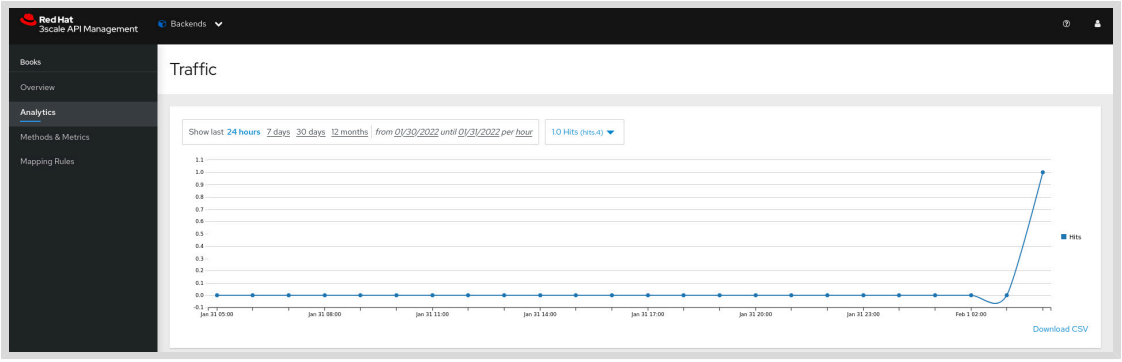
7.4. Navigate to **Products** > **API** > **Integration** > **Configuration** and click **Promote v. 2 to Staging APIcast**.

Your version number could differ if you previously promoted the `API` product.

7.5. Use the following `curl` command to generate a new hit to the mapping rule.

```
[student@workstation ~]$ curl \
  "https://api-3scale-apicast-staging.apps.ocp4.example.com:443/library/books?
user_key=USER_KEY"
[{"title":"Frankenstein","authorName":"Mary Shelley","year":1818,"copies":10},
{"title":"A Christmas Carol","authorName":"Charles
 Dickens","year":1843,"copies":5},{"title":"Pride and
 Prejudice","authorName":"Jane Austen","year":1813,"copies":3}]
[student@workstation ~]$
```

7.6. Verify the mapping rule: navigate to **Books** > **Analytics**, the graphic information should show an increment by 1.



8. Verify the application plan.

8.1. Use the `DO240/labs/applications-plans/test_api_plan.sh` script to generate requests to the books API. Provide the `user_key` as the only argument.

```
[student@workstation ~]$ DO240/labs/applications-plans/test_api_plan.sh USER_KEY
Making 100 requests to curl -s https://api-3scale-apicast-
staging.apps.ocp4.example.com:443/library/books?user_key=USER_KEY
Request No: 1
Request No: 2
...output omitted...
Request No: 100
Finished!
```

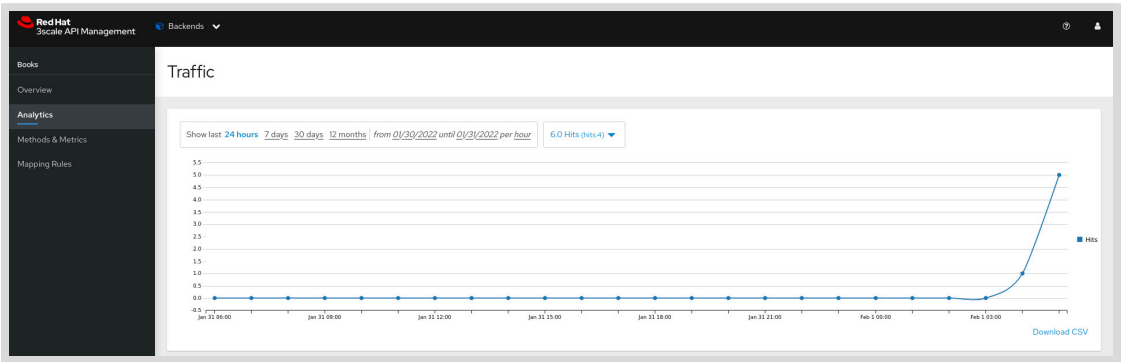
This script generates 100 requests to the API.

8.2. Verify that the usage limit was reached by using the following `curl` command.

```
[student@workstation ~]$ curl "https://api-3scale-apicast-
staging.apps.ocp4.example.com:443/library/books/?user_key=USER_KEY"
Usage limit exceeded[student@workstation ~]$
```

The books API call should respond with the: `Usage limit exceeded` message.

8.3. Verify the rate limit rule: navigate to **Books** > **Analytics**, the graphic information should show an increment only by 5.



Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish applications-plans
```

This concludes the guided exercise.

Managing API Versions

Objectives

After completing this section, you should be able to create and configure multiple versions of APIs.

Introducing API Versioning

API versioning is a way of releasing backward-incompatible changes to your API without breaking client code.

An example of a backward-incompatible change is removing a field from an API response. The client code that reads the removed field breaks if it tries to read a property that no longer exists.

You use API versioning when there is a need to change the client code before using a new incompatible version.

There are several ways to implement API versioning depending on where you place the version number, such as:

- *URL versioning* specifies the version number as part of the URL path or in a URL query parameter.
- *Request header versioning* specifies the version number in an HTTP header. This can be the `accept` header or a custom header such as `x-api-version`.

To implement URL API versioning in Red Hat 3scale API Management, create a backend per version of your application. Then, add them all to a single product, but prefix each with a unique path based on the version string.

3scale API Management also offers the `Routing` policy, which provides request header versioning.

URL Versioning

With URL versioning, requests are routed based on the version value taken from the request URL.

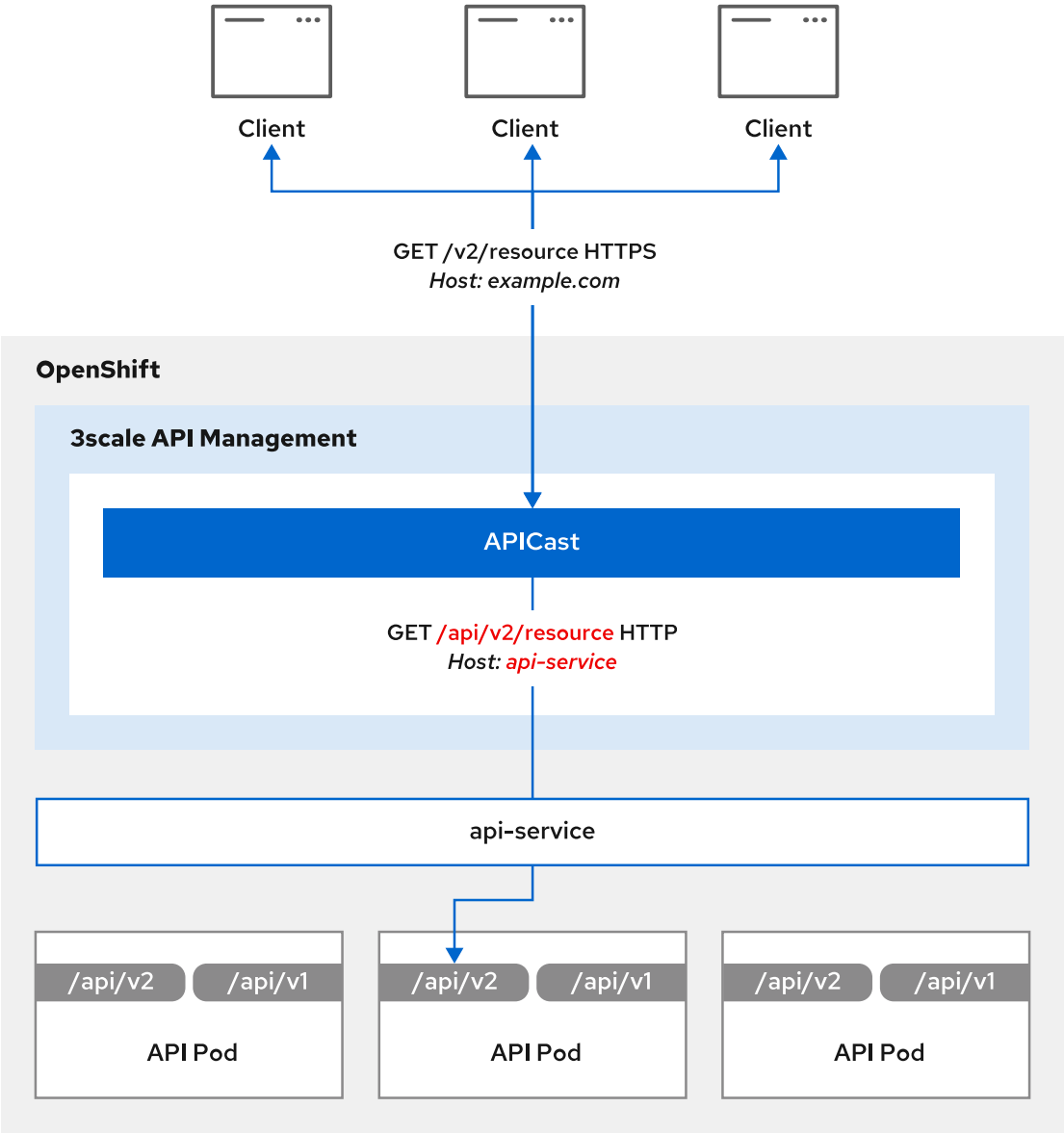


Figure 2.16: URL versioning request routing

URL versioning makes the version clear because the version is apparent from the URL. However, using URL versioning can break the one-to-one relationship between a resource and its REST URL. Namely, a resource that has no changes gets another URL for the new version.

Preparing the New Version

The design of the public-facing API must accept a version number in the URL path. Doing so ensures that client applications of the API supply the URL version during development.

For example, clients for the first API version use a URL such as `https://example.com/v1/resource-name`, where the `v1` indicates the version of the API the client wants to consume.

To release a breaking change, you must change the URL to use a new version.

Following the previous example, incrementing from `v1` to `v2` changes the URL to `https://example.com/v2/resource-name`, representing the new version.

The different versions in the request path route the request to the correct endpoint in the target API.

Using Products and Backends for URL Versioning

For URL versioning, create a backend for each of the versions and set the private base URL of the backend to point to the desired API version.

Having each backend point to a different path creates a logical separation for each version.

Backend	Private Base URL
`backend_v1`	`http://microservice-internal-url/api/v1`
`backend_v2`	`http://microservice-internal-url/api/v2`

For the preceding example, you assign both backends to the product by using a public path that represents the version.

For example, the public path for backend version `v2` is `/v2`.

When handling the request, 3scale API Management rewrites the path and eliminates the public path specified in the mapping between the product and the backend.

3scale API Management Request Translation.

Public URL	Private URL
`https://example.com/v1/resource-name`	`http://microservice-internal-url/api/v1/resource-name`
`https://example.com/v2/resource-name`	`http://microservice-internal-url/api/v2/resource-name`

Request Header Versioning

Request header versioning lets clients consume different versions of an API preserving the same URL. With this approach, if a REST resource does not change, then there is no need to release a new version for it. The downside is that it requires clients to know which versions of a resource are available before consuming the API.

Using the Routing Policy for Request Header Versioning

To implement request header versioning, use the `Routing` policy.

This policy routes requests based on the URL, request headers, or JWT claims.

The `Routing` policy must be placed before the APIcast policy in the policy chain.

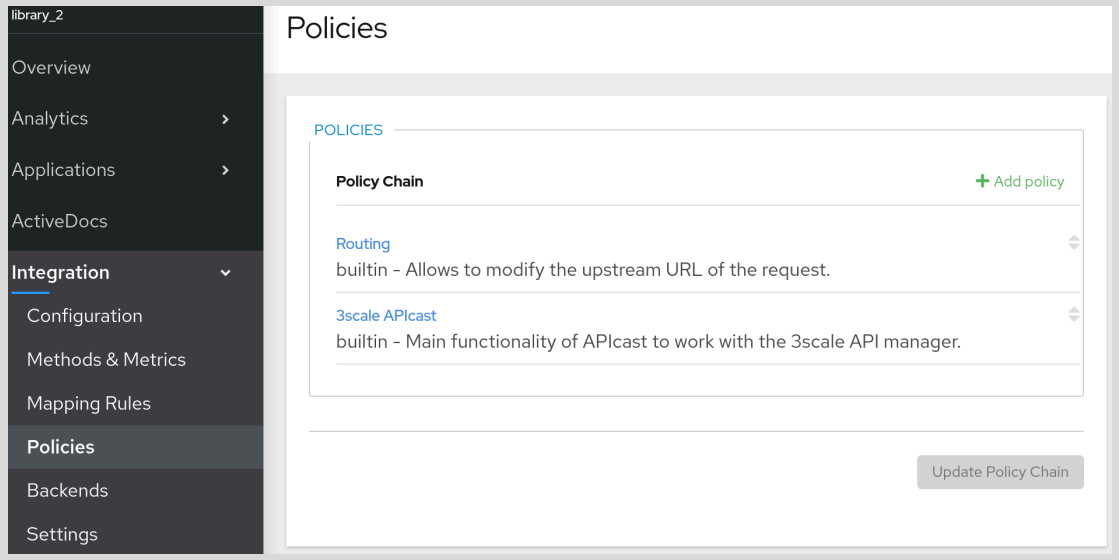


Figure 2.17: Product policies

You can configure the `Routing` policy to route requests by using a custom header, such as `x-api-version`.

For this, you can configure the `Routing` policy in the Admin Portal with the following values.

Routing policy for request header versioning

Field	Value
url	`http://microservice-internal-url/api/v2`
combine_op	`and`
match	`header`
value	v2
value_type	`Evaluate 'value' as plain text.`
op	`==`
header_name	`x-api-version`

With this configuration, the `Routing` policy routes requests containing the header `x-api-version: v2` to `http://microservice-interal-url/api/v2`.

After updating the policy configuration, you must click **Update Policy Chain** in the Policies section. Then, promote the configuration to staging to verify that it routes to the correct version.

You can use the `curl` utility to test the request header versioning by adding the version header to the request.

```
[student@workstation ~]$ curl -v -s --header "x-api-version: v2" \
  "https://example.com/resource?user_key=USER_KEY"
...output omitted...
> GET /resource?user_key=b2c9a5d6204ff848052dd4ba6944fbc7 HTTP/1.1
```

```
> Host: example.com
> User-Agent: curl/7.61.1
> Accept: /
> x-api-version: v2
>
...output omitted...
< HTTP/1.1 200 OK
...output omitted...
```

You can also use the 3scale Toolbox CLI to create the policy chain by executing the `policies` command with the `import` subcommand and an argument with a file containing the policy list.

```
[student@workstation ~]$ 3scale policies import \
  TENANT PRODUCT -f policy-chain.yaml
```

Where `policy-chain.yaml` contains a policy list with the `Routing` policy followed by the default `3scale APIcast` policy.

```
- name: routing
  version: builtin
  configuration:
    rules:
    - condition:
        combine_op: and
        operations:
        - value_type: plain
          match: header
          value: v2
          op: matches
          header_name: x-api-version
      url: http://example.com/api/v2
  enabled: true
- name: apicast
  version: builtin
  configuration: {}
  enabled: true
```

This command creates a new configuration for the product and promotes that configuration to the staging environment.

References

For more information, refer to the *URL versioning* section in the *Red Hat 3scale API Management Administering the API Gateway* at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html/administering_the_api_gateway/versioning-3scale#url_versioning

Guided Exercise

Managing API Versions

In this exercise, you will use URL API versioning to safely publish and monitor a breaking change of a `books-api` API under a new version `v2`.

Outcomes

You should be able to:

- Update an existing product to add a backend pointing to the new version.
- Create a mapping rule to allow public access to the new version.
- Create a metric for monitoring the new version usage.
- Promote product configuration changes to the staging environment.

Note

For faster feedback, this exercise is executed on the staging environment to avoid having to propagate changes to both staging and production.

You can promote the changes to production by navigating to the product's configuration in the product detail page. Then navigate to **Integration** > **Configuration** and click the promote to production button.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that:

- The `workstation` machine is ready and the OpenShift cluster is available.
- The 3scale Admin Portal is up and running.
- The `applications-versions` project is created and has the last `books-api` API version deployed.
- The `books-api` version `v1` is already in production and accessible through 3scale, but `v2` is only privately accessible.
- The RHOCP cluster has a pod named `curl-pod` to allow `curl` command execution within the cluster.

```
[student@workstation ~]$ lab start applications-versions
```

Instructions

1. Use the 3scale Toolbox CLI to extract the `USER_KEY` and `STAGING_DOMAIN` environment variables to use during this exercise.

1.1. Extract the `user_key` into a USER_KEY environment variable.

To extract the `user_key` get the application ID for the `applications_versions_app`.

```
[student@workstation ~]$ APP_ID=$(3scale application list 3scale-tenant -o json \
   | jq '.[] | select(.name == "applications_versions_app") | .id')
```

Use the APP_ID application ID to get the `user_key` value.

```
[student@workstation ~]$ USER_KEY=$(3scale application \
  show 3scale-tenant $APP_ID -o json \
  | jq -r '.user_key')
```

1.2. Extract the `staging` domain into a STAGING_DOMAIN environment variable.

```
[student@workstation ~]$ STAGING_DOMAIN=$(3scale proxy-config \
  show 3scale-tenant applications_versions sandbox -o json \
  | jq -r '.content.proxy.staging_domain')
```

1.3. Verify that the previous steps correctly assigned the STAGING_DOMAIN and USER_KEY environment variables.

```
[student@workstation ~]$ echo $STAGING_DOMAIN; echo $USER_KEY
applications-versions-3scale-apicast-staging.apps.ocp4.example.com
987f7abce12aaf49a8c297793bf44893
```

Note

Your USER_KEY will differ from the one listed here.

2. Compare versions `v1` and `v2` to verify that there is a breaking change in the `books-api` API. Only version `v1` is publicly accessible. To test `v2` you must log in to RHOCP as the `developer` user and execute a `curl` command within the cluster.

2.1. Test public access to the `books` resource in the `applications_versions` product to verify that only version `v1` is accessible.

```
[student@workstation ~]$ curl -s \
  "https://$STAGING_DOMAIN/v1/books?user_key=$USER_KEY" | jq
[
  {
    "authorName": "Mary Shelley",
    "title": "Frankenstein",
    "copies": 10,
    "year": 1818
  },
  ...output omitted...
]
```

Send a request for version `v2` of the `books` resource.

```
[student@workstation ~]$ curl -s \
  "https://$STAGING_DOMAIN/v2/books?user_key=$USER_KEY"; echo
No Mapping Rule matched
```

The response `No Mapping Rule matched` indicates that the resource is not publicly accessible for `v2`.

2.2. To verify that `v2` is actually deployed within the cluster, log in to RHOCP as the `developer` user. Then use the cluster pod `curl-pod` to query the `v2` of the `books` resource by using the FQDN for the `books-api` service.

Issue the login command to log into RHOCP.

```
[student@workstation ~]$ oc login \
  -u=developer -p=redhat --server=https://api.ocp4.example.com:6443
Logged into "https://api.ocp4.example.com:6443" as "developer" using existing
 credentials.
...output omitted...
```

Execute a curl command from within the cluster to verify that `v2` is deployed and that the books resource contains a breaking change.

```
[student@workstation ~]$ oc exec curl-pod -n applications-versions \
  -- curl -s \
 "http://books-api.applications-versions.svc.cluster.local/api/v2/books" |jq

[
  {
    "title": "Frankenstein",
    "author": {
      "name": "Mary Shelley",
      "birthYear": 1797
    },
    "year": 1818,
    "copies": 10
  },
  ...output omitted...
]
```

The version `v2` replaced `authorName` with the `author` object. This is a backward incompatible change and must be published under a new version to avoid breaking existing client code.

3. Navigate to the 3scale Admin Portal and create a backend called `books_backend_v2`. Set the backend URL to point to the service version `v2` by using the `http://books-api.applications-versions.svc.cluster.local:80/api/v2/books` URL.

3.1. Using a web browser, log in to the 3Scale Admin Portal as the `admin` user. The portal is located at `https://3scale-admin.apps.ocp4.example.com`.

3.2. From the top pane drop-down menu, select **Backends** to view the list of backends.

3.3. Create a backend for the `v2` by clicking **Create Backend**, providing the following values, and submitting the form. Leave all other fields as their default values.

Field	Value
Name	`books_backend_v2`
Private Base URL	`http://books-api.applications-versions.svc.cluster.local:80/api/v2`

4. Associate `books_backend_v2` backend with the `applications_versions` product.

4.1. From the top pane drop-down menu, click **Products**, and then click the **applications_versions** product.

4.2. Navigate to the list of backends associated with the product by selecting **Integration** > **Backends**. Associate a backend with the product by clicking **Add Backend**.

Select `books_backend_v2` as the backend, provide the unique path `/v2`, and submit the form.

This causes a path rewrite in the handling of the request. It eliminates `/v2` from the request path and proceeds to use the backend base URL, which is `http://books-api.applications-versions.svc.cluster.local/api/v2`.

3scale Request Translation

Public URL	Private URL
`https://applications-versions-3scale-apicast-staging.apps.ocp4.example.com/v2/books`	`http://books-api.applications-versions.svc.cluster.local/api/v2/books`

5. Use the 3scale Toolbox CLI to create a new metric `hits_v2` to count the number of requests to `v2`.

```
[student@workstation ~]$ 3scale metric create \
  3scale-tenant applications_versions hits_v2
Created metric id: 11. Disabled: false
```

6. To allow public access to `v2` create a mapping rule for the pattern `/v2` and assign it the metric `hits_v2`.

6.1. Navigate to the list of mapping rules associated with the product by selecting **Integration** > **Mapping Rules**.

6.2. Create a mapping rule for the `applications_versions` product by clicking **Create Mapping Rule**, providing the following values, and submitting the form. Leave all other fields as their default values.

Field	Value
Verb	`GET`
Pattern	`/v2`
Method or Metric to increment	Select **Metric**, then select `hits_v2`

7. Promote and test the product configuration.

7.1. Promote the `applications_versions` product configuration to the staging environment by using the 3scale Toolbox CLI.

Use the `proxy-config` command with the `deploy` subcommand to promote to staging.

```
[student@workstation ~]$ 3scale proxy-config deploy \
  3scale-tenant applications_versions
{
  "service_id": 3,
  "endpoint": "https://applications-versions-3scale-apicast-
production.apps.ocp4.example.com:443",
  "api_backend": "http://books-api.applications-versions.svc.cluster.local:80/api/
v1",
...output omitted...
```

To promote to production you can use the `proxy-config` command with the `promote` subcommand instead.

7.2. Verify that `v2` is publicly accessible by sending a request to the `books` resource.

```
[student@workstation ~]$ curl -s \
  "https://$STAGING_DOMAIN/v2/books?user_key=$USER_KEY" | jq
[
  {
    "title": "Frankenstein",
    "author": {
      "name": "Mary Shelley",
      "birthYear": 1797
    },
    "year": 1818,
    "copies": 10
  },
  ...output omitted...
]
```

8. In a command-line terminal, run the `/scripts/gen-load.sh` script in the `DO240-apps` repository to generate traffic under `v1` and `v2`. Increase the hit count on `v1` by 5 and the hit count on `v2` by 10.

```
[student@workstation ~]$ ~/DO240-apps/scripts/gen-load.sh \
  5 "https://$STAGING_DOMAIN/v1/books?user_key=$USER_KEY"
5/5 completed requests to  https://applications-versions-3scale-apicast-
staging.apps.ocp4.example.com/v1/books
```

```
[student@workstation ~]$ ~/DO240-apps/scripts/gen-load.sh \
  10 "https://$STAGING_DOMAIN/v2/books?user_key=$USER_KEY"
10/10 completed requests to  https://applications-versions-3scale-apicast-
staging.apps.ocp4.example.com/v2/books
```

9. Review the endpoint metrics within the 3scale Admin Portal and verify that `v2` has 5 more requests than `v1`.

   9.1. In the Admin Portal, navigate to **Analytics** > **Traffic**.

   9.2. Select metric `Hits` used by `v1` by clicking **Hits (hits)**. Note the number to compare it with `v2`

   9.3. Select metric `hits_v2` used by `v2` by clicking **hits_v2 (hits_v2)**. Note the resulting total in the graph. This number should be 5 units higher than the number for `v1`.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish applications-versions
```

This concludes the guided exercise.

Configuring Service Discovery

Objectives

After completing this section, you should be able to discover applications deployed on OpenShift automatically.

3scale Service Discovery

With service discovery, you can import products and backends from existing Red Hat OpenShift Container Platform (RHOCP) services. Service discovery also enables you to update the ActiveDocs OpenAPI specification.

To use service discovery:

- Enable it in the `system` configmap in the project where 3scale is installed.
- Add metadata to the services to enable service discovery and configure the service information.
- Make sure that 3scale has access to the services that you want to import.

Enabling Service Discovery

You can enable or disable service discovery globally by editing the `system` configmap in the `3scale` project. To do this you must edit the configmap `service_discovery.yml` entry, setting the value `production.enabled` to true.

If you installed 3scale by using the operator method then service discovery is enabled by default.

You can verify that it is enabled by describing the `system` configmap.

```
[student@workstation ~]$ oc describe configmap system -n 3scale
...output omitted...
service_discovery.yml:
----
production:
  enabled: <%= cluster_token_file_exists = File.exists?(cluster_token_file_path =
 '/var/run/secrets/kubernetes.io/serviceaccount/token') %>
...output omitted...
```

Making a Service Eligible for Discovery

Service discovery needs the service details to create products and backends. For this, 3scale looks for the following meta data to know how to import and configure each service it finds.

Service Discovery Metadata

Type	Key	Value
Label	discovery.3scale.net (Required)	[true|false] to enable/disable service discovery for the service

Type	Key	Value
Annotation	discovery.3scale.net/scheme (Required)	[http|https] scheme used by the service
Annotation	discovery.3scale.net/port (Required)	port number used by the service
Annotation	discovery.3scale.net/path	base path of the target service, if omitted the root path ("/") is used
Annotation	discovery.3scale.net/description-path	path to the service's OpenApi endpoint
Annotation	discovery.3scale.net/discovery-version	version of the 3scale discovery process

For example, to configure a service using http and listening on port 8080 with only the required configuration:

- Add the label to enable service discovery.

```
[student@workstation ~]$ oc label svc/SERVICE_NAME \
  discovery.3scale.net="true"
```

- Add the service schema annotation:

```
[student@workstation ~]$ oc annotate svc/SERVICE_NAME \
  discovery.3scale.net/scheme=http
```

- Add the service port annotation:

```
[student@workstation ~]$ oc annotate svc/SERVICE_NAME \
  discovery.3scale.net/port="8080"
```

You can also achieve the same result by editing the service resource definition:

```
[student@workstation ~]$ oc edit service books-api

apiVersion: v1
kind: Service
metadata:
  annotations:
    discovery.3scale.net/port: "8080"
    discovery.3scale.net/scheme: http
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: "2022-01-07T17:15:42Z"
  labels:
    app: books-api
    app.kubernetes.io/component: books-api
    app.kubernetes.io/instance: books-api
    discovery.3scale.net: "true"
  name: books-api
...output omitted...
```

Discovering Services from Other Projects

To Discover a service located in a different project than the 3scale project, 3scale needs view permissions into the project where that service exists. To give 3scale access to other projects, the 3scale `amp` service account must have the `view` role.

```
[student@workstation ~]$ oc policy add-role-to-user \
  view system:serviceaccount:3SCALE_PROJECT:amp -n PROJECT_NAME
```

You can also give 3scale access to all the projects in the cluster by assigning it the `view` role as a cluster role.

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user view \
  system:serviceaccount:3SCALE_PROJECT:amp
```

Importing Products And Backends from Discovered Services

To make use of the discovered services, navigate to the Admin Portal to import the discovered services into 3scale products and backends.

To import a discovered service, click **Products** on the top pane drop-down menu to navigate to the Products page. Then click **Create Product** to navigate to the New Product form.

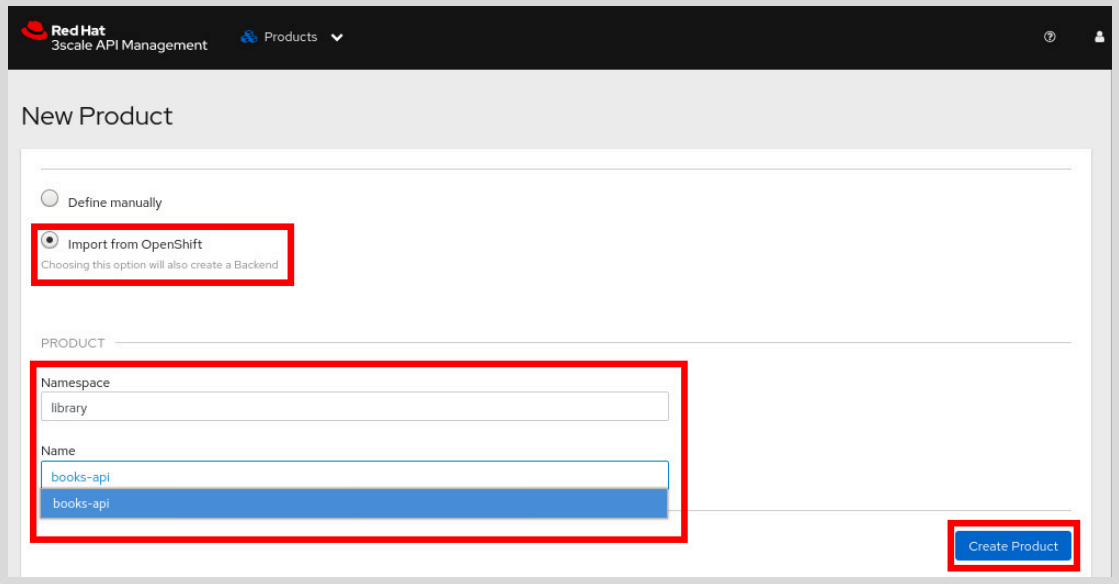


Figure 2.18: 3scale imported product

In the New Product form, select **Import from OpenShift**. This populates the Namespace and Name drop-down menus with the discovered services. Select the desired service, then click **Create Product**.

By clicking **Create Product**, 3scale creates the product and backend based on the 3scale metadata from the service.

Accessing the Discovered Services

Although the product import creates the product and backend resources, the product is not accessible yet.

To access the imported product and the backend associated with it, the product needs a 3scale application plan and an application.

To create an application plan named *APPLICATION_PLAN* you must reference the *TENANT* and the *PRODUCT_SYSTEM_NAME* resources.

```
[student@workstation ~]$ 3scale application-plan create \
  TENANT PRODUCT_SYSTEM_NAME APPLICATION_PLAN
```

To create the application you need a user account to link it to, the *PRODUCT_SYSTEM_NAME*, the previously created plan, and the name you want for the application.

```
[student@workstation ~]$ 3scale application create \
  TENANT USER_ACOUNT PRODUCT_SYSTEM_NAME APPLICATION_PLAN APPLICATION
```

This application provides us with a `user_key` to make requests in the Staging APIcast environment.

To verify the product import, navigate to the product detail page in the Admin Portal. Then click **Integration** > **Configuration** and scroll down to the `Staging APICast` section. There you can copy an example curl request to validate that your service is working properly in the staging environment.

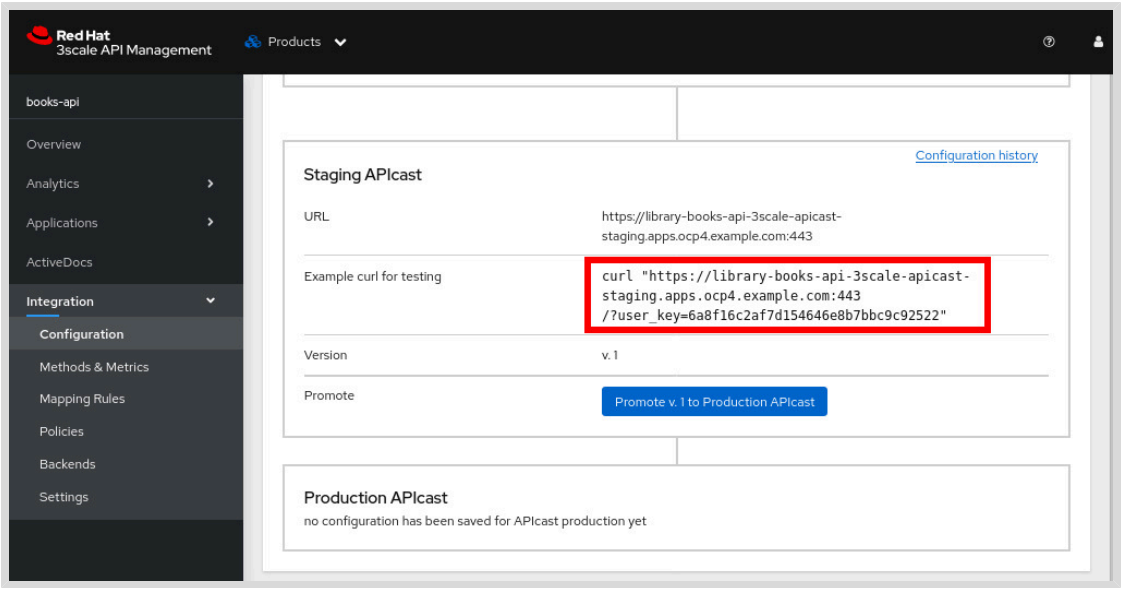


Figure 2.19: Staging APIcast configuration

References

For more information, refer to the *Service Discovery* chapter in the *Red Hat 3scale Api Management 2.11 Admin Portal Guide* at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html/admin_portal_guide/service-discovery_service-discovery

Guided Exercise

Configuring Service Discovery

In this exercise, you will use Red Hat 3scale API Management service discovery capabilities to automatically set up 3scale API Management products and backends from applications deployed on OpenShift.

Outcomes

You should be able to:

- Configure a Red Hat OpenShift Container Platform (RHOCP) service to make it discoverable by 3scale.
- Import a discovered OpenShift service to automatically create a 3scale product and 3scale backend.
- Make the product accessible through the staging APIcast gateway.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start applications-discovery
```

Instructions

1. Create an OpenShift project called `library` and deploy the sample application `books-api` from the Git repository.

   1.1. Log in to RHOCP as the `admin` user.

```
[student@workstation ~]$ oc login -u=admin -p=redhat \
  --server=https://api.ocp4.example.com:6443
Login successful.
```

   1.2. Create the `library` project.

```
[student@workstation ~]$ oc new-project library
Now using project "library" on server "https://api.ocp4.example.com:6443".
...output omitted...
```

   1.3. Deploy the `books-api` sample application and name it `books-api`.

```
[student@workstation ~]$ oc new-app \
  --name=books-api \
  --context-dir=library/books-api \
  --build-env NODE_ENV=development \
   https://github.com/RedHatTraining/DO240-apps.git
...output omitted...
  --> Success
    Build scheduled, use 'oc logs -f buildconfig/books-api' to track its progress.
...output omitted...
```

2. Make the service discoverable from 3scale. Add the required metadata to `books-api`, ensure service discovery is enabled, and give the `amp` service account view permissions to the `library` project.

   2.1. Add the service label `discovery.3scale.net: "true"`.

```
[student@workstation ~]$ oc label svc/books-api \
  discovery.3scale.net="true"
service/books-api labeled
```

   2.2. Add the service annotations `discovery.3scale.net/scheme: "http"` and `discovery.3scale.net/port: "8080"`.

```
[student@workstation ~]$ oc annotate svc/books-api  \
  discovery.3scale.net/port="8080"
service/books-api annotated
[student@workstation ~]$ oc annotate svc/books-api \
  discovery.3scale.net/scheme="http"
service/books-api annotated
```

   2.3. Verify that the `books-api` service has been modified accordingly.

```
[student@workstation ~]$ oc get service books-api -o yaml

apiVersion: v1
kind: Service
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
    discovery.3scale.net/scheme: http
    discovery.3scale.net/port: "8080"
  labels:
    app: books-api
    app.kubernetes.io/component: books-api
    app.kubernetes.io/instance: books-api
    discovery.3scale.net: "true"
  name: books-api
  namespace: library
spec:
...output omitted...
```

2.4. Validate that 3scale service discovery is enabled. Describe the `system` configmap in the `3scale` project and, under `service_discovery.yml`, verify that the following values show for the keys `production.enabled`, `production.bearer_token`, and `production.authentication_method`.

```
[student@workstation ~]$ oc describe configmap system -n 3scale
...output omitted...
service_discovery.yml:

  production:
    enabled: <%= cluster_token_file_exists = File.exists?(cluster_token_file_path
 ='/var/run/secrets/kubernetes.io/serviceaccount/token') %>
    bearer_token: "<%= File.read(cluster_token_file_path) if
 cluster_token_file_exists %>"
    authentication_method: service_account
...output omitted...
```

2.5. Grant the 3scale `amp` service account with `view` permissions on the `library` project so that 3scale can access the `books-api` service metadata.

```
[student@workstation ~]$ oc policy add-role-to-user \
  view system:serviceaccount:3scale:amp -n library
clusterrole.rbac.authorization.k8s.io/view added:
 "system:serviceaccount:3scale:amp"
```

2.6. Run the `ls-discoverable.sh` script from the `~/DO240-apps/scripts` directory to verify that the `books-api` service has the correct metadata. Also, verify that the 3scale `amp` service account has access to the `library` project.

The `ls-discoverable.sh` script lists services in all namespaces that have the label `discovery.3scale.net` set to `true` and have the required 3scale annotations. You must verify that the annotations values are correct because the script only determines the annotation existence.

```
[student@workstation ~]$ ~/DO240-apps/scripts/ls-discoverable.sh
{
  "service-name": "books-api",
  "service-namespace": "library",
  "labels": {
    "app": "books-api",
    "app.kubernetes.io/component": "books-api",
    "app.kubernetes.io/instance": "books-api",
    "discovery.3scale.net": "true"
  },
  "annotations": {
    "discovery.3scale.net/port": "8080",
    "discovery.3scale.net/scheme": "http",
    "openshift.io/generated-by": "OpenShiftNewApp"
  }
}
```

If `ls-discoverable.sh` does not print anything, verify that you added the required metadata correctly.

Run the following command to validate that 3scale can access your service definition.

```
[student@workstation ~]$ oc get rolebinding -o wide -A \
  | grep -E 'NAME|ClusterRole/view|3scale/amp'
NAMESPACE  NAME   ROLE                AGE     USERS   GROUPS   SERVICEACCOUNTS
library    view   ClusterRole/view    5d22h                    3scale/amp
```

From the previous command note that the `amp` service account has the `ClusterRole/view` in the `library` namespace. Therefore, our service is visible from 3scale.

3. Import the 3scale product and backend by using the 3scale Admin Portal.

3.1. Navigate to `https://3scale-admin.apps.ocp4.example.com` to access the Admin Portal by using the credentials provided in previous sections.

3.2. Click **Products** on the top pane's drop-down menu to go to the products page.

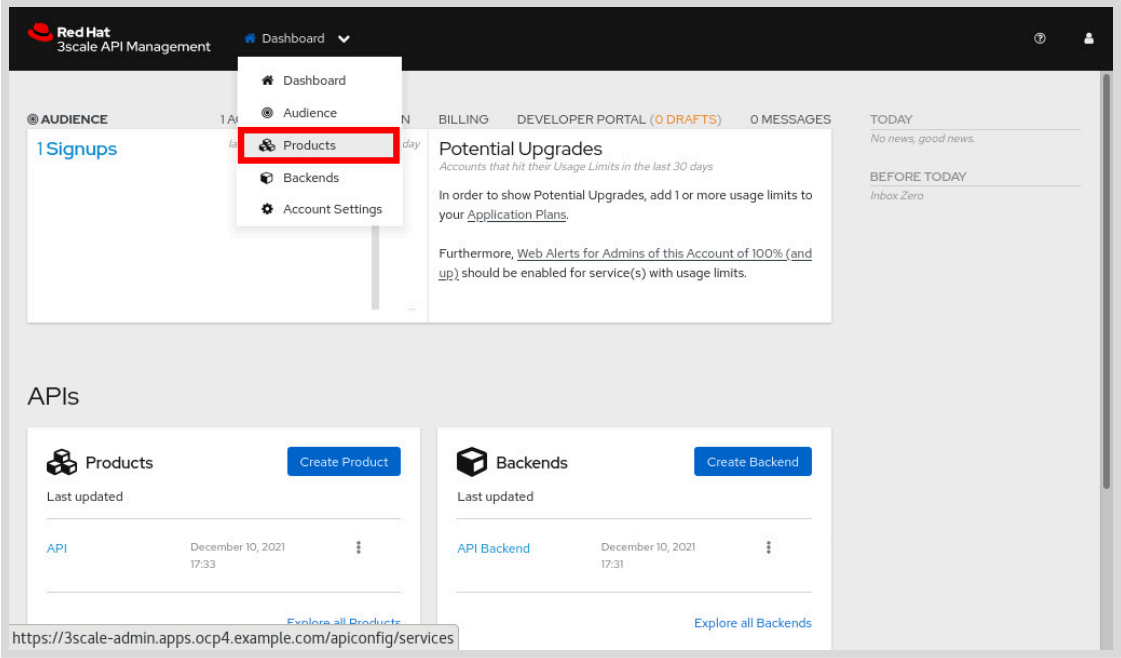


Figure 2.20: 3scale tenant administration portal

3.3. Click **Create Product**.

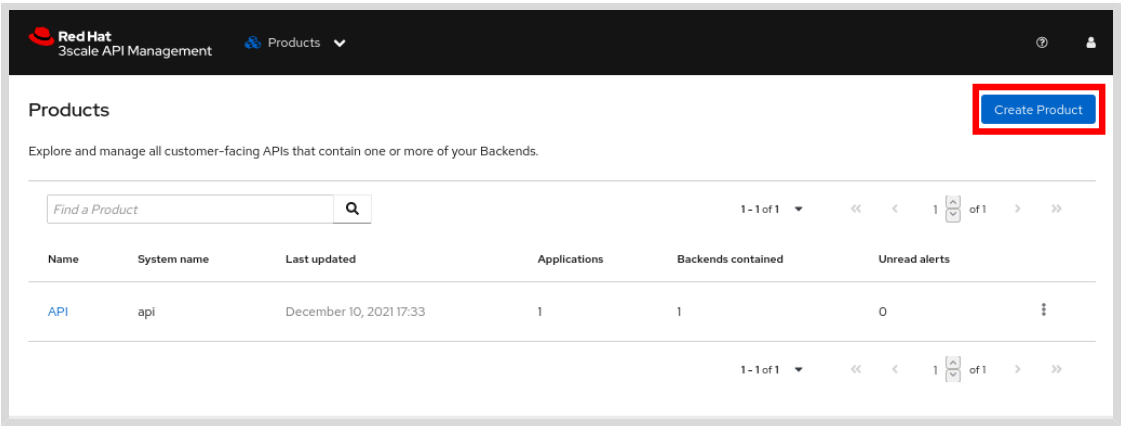


Figure 2.21: 3scale products page

3.4. In the `New Product` form, select **Import from OpenShift**. This populates the `Namespace` and `Name` drop-down menus with the discovered services. Select `books-api` in the `Name` selector field and click **Create Product**.

Note

If services do not show in the `Namespace` drop-down menu, verify that the label and the annotations are correct. Also, validate that the 3scale `amp` service account has view permissions on your service namespace.

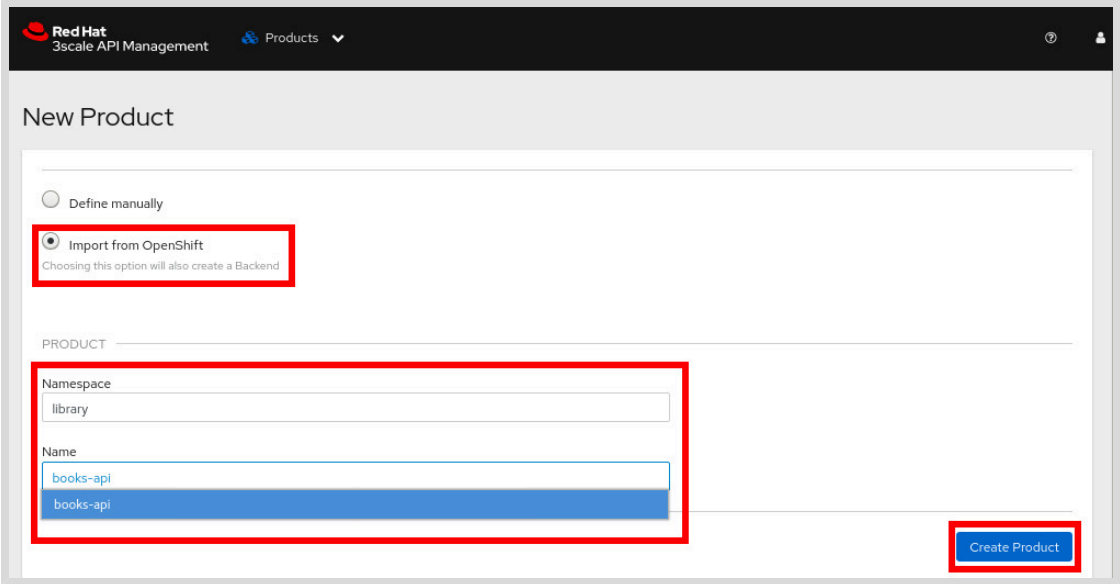


Figure 2.22: 3scale new product page

3.5. Go to the products page and verify that 3scale imported the discovered service.

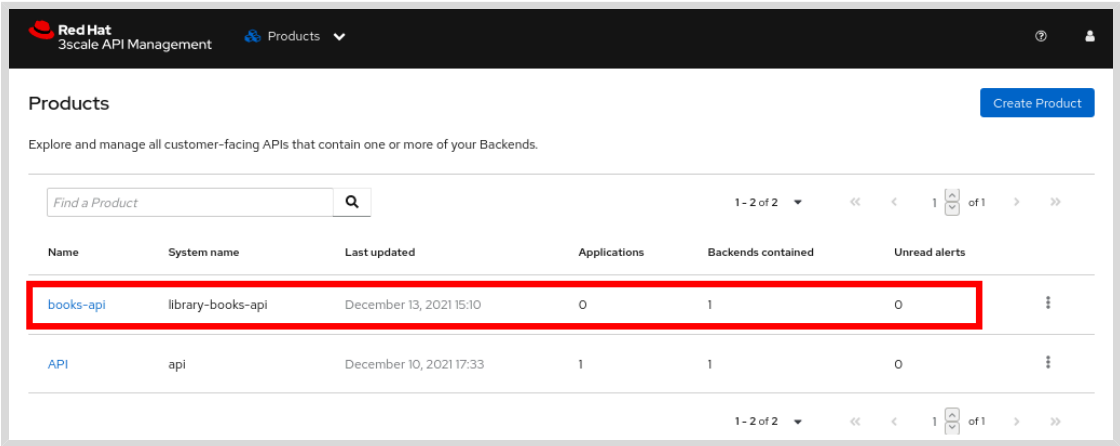


Figure 2.23: 3scale imported product

4. To access the `books-api` through 3scale, create an application plan and an application by using 3scale Toolbox CLI. Use the `3scale-tenant` remote configured in previous sections.

Note

The Toolbox CLI has the `-h` help option, which provides reference information for 3scale commands and subcommands.

To find out which arguments you need to create an application, you can execute the following command.

```
[student@workstation ~]$ 3scale application create -h
```

Note

The `3scale` alias created in previous sections uses the `-k` option because the 3scale Toolbox CLI container image does not have the proper certificate installed. If you get SSL certificate errors then you can either use the `-k` to avoid verifying the certificate's validity or install a valid certificate.

4.1. Create an application plan called `library_basic_plan` by using the 3scale Toolbox CLI in the command line.

```
[student@workstation ~]$ 3scale application-plan create \
  3scale-tenant library-books-api library_basic_plan
Created application plan id: 13. Default: false; Disabled: false
```

4.2. Create an application on the default `john` account, with the name `library-app`, and assign it the `library_basic_plan`.

```
[student@workstation ~]$ 3scale application create \
  3scale-tenant john library-books-api library_basic_plan library-app
Created application id: 7
```

5. Make a curl request to the `/books` endpoint in the staging APIcast service to determine whether we can reach the API through 3scale.

5.1. Go to the `books-api` product page in the 3scale Admin Portal. Then, go to **Integration** > **Configuration** and scroll down to the `Staging APIcast` section to copy the example curl command.

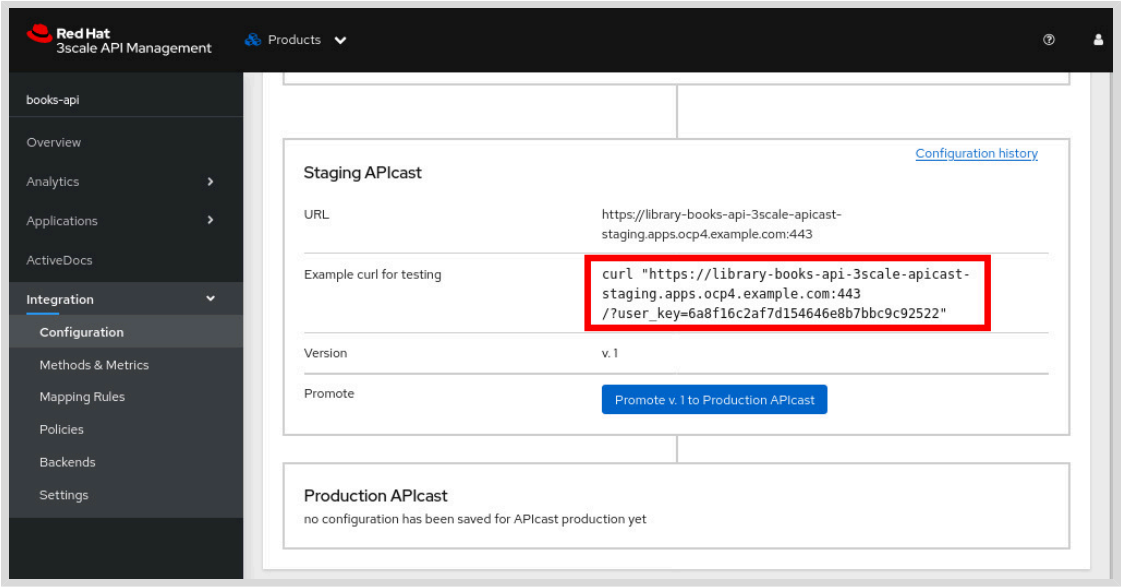


Figure 2.24: Staging APIcast configuration

5.2. Edit the copied curl command to send a request to the `/books` endpoint.

```
[student@workstation ~]$ curl \
   https://library-books-api-3scale-apicast-staging.apps.ocp4.example.com:443/
books?user_key=YOUR_USER_KEY
[{"title":"Frankenstein", ...
...output omitted...
```

6. Update the `books-api` OpenShift service port to `8181`. Repeat the curl request to see it fail.

6.1. Edit the `books-api` OpenShift service to make it listen on port `8181`.

```
[student@workstation ~]$ oc edit service books-api
...output omitted...
spec:
  ports:
  - name: 8080-tcp
    port: 8181
    protocol: TCP
    targetPort: 8080
```

6.2. Execute the curl command to verify that we can no longer reach `books-api`.

```
[student@workstation ~]$ curl \
   https://library-books-api-3scale-apicast-staging.apps.ocp4.example.com:443/
books?user_key=YOUR_USER_KEY
<html>
<head><title>502 Bad Gateway</title></head>
<body>
<center><h1>502 Bad Gateway</h1></center>
<hr><center>openresty</center>
</body>
</html>
```

6.3. Find the pod running the APIcast staging service.

```
[student@workstation ~]$ oc get pods -l threescale_component=apicast -n 3scale
NAME                          READY   STATUS    RESTARTS   AGE
apicast-production-1-j2jsl    1/1     Running   4          3d18h
apicast-staging-1-s8tq8       1/1     Running   4          3d18h
```

6.4. Inspect the logs in the APIcast staging service to see why it fails.

```
[student@workstation ~]$ oc logs apicast-staging-1-s8tq8 -n 3scale \
  | grep "No route to host"
...output omitted...
... [error] 22#22: \*1385 connect() failed (113: No route to host) ...
 upstream: "http://172.30.204.151:8080/books?user_key=6a...
...output omitted...
```

From the previous log, notice that 3scale is trying to reach our service on port 8080 and APIcast can not find anything on that port.

We can search for our `books-api` service to verify that it is running on port `8181`.

```
[student@workstation ~]$ oc get service books-api
NAME        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
books-api   ClusterIP   172.30.204.151   <none>        8181/TCP   3d18h
```

6.5. Revert the service port to its original value and send another request to see it succeed.

```
[student@workstation ~]$ oc edit service books-api
...output omitted...
spec:
  ports:
  - name: 8080-tcp
    port: 8080
    protocol: TCP
    targetPort: 8080
...output omitted...
```

```
[student@workstation ~]$ curl \
   https://library-books-api-3scale-apicast-staging.apps.ocp4.example.com:443/
books?user_key=YOUR_USER_KEY
[{"title":"Frankenstein", ...output omitted...
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish applications-discovery
```

This concludes the guided exercise.

Creating Multiple Tenants in Red Hat 3Scale API Management

Objectives

After completing this section, you should be able to create multiple tenants in Red Hat 3scale API Management.

Introducing Tenants

You can configure multiple 3scale API Management instances within the same installation. An instance of 3scale API Management is called a *tenant*. Every tenant can be managed through its own Admin Portal.

By default, the 3scale API Management installation creates the `3scale` tenant. In previous sections of this course, you have used the Admin Portal of the default `3scale` tenant to manage your APIs (`https://3scale-admin.apps.ocp4.example.com`).

While the default tenant, `3scale`, might be enough for most organizations, larger companies might prefer to have several, independent instances to manage their APIs. For example, a social network might choose to have a tenant for internal APIs, and another tenant for APIs that can be consumed by external developers.

Features of a Tenant

Every tenant provides several interesting features that you might consider when deciding the number of tenants that you want to have:

Feature	Explanation	URLs generated
Custom Developer Portal URL	For every tenant, a public Developer Portal is provided.	*TENANT_NAME*.*WILDCARD_DOMAIN*
Custom APIcast URL	Every tenant gets a staging and production APIcast URL containing the tenant name.	The URL for staging follows the pattern `api-`*TENANT_NAME*`-apicast-staging.`*WILDCARD_DOMAIN*. The URL for production follows the pattern `api-`*TENANT_NAME*`-apicast-production.`*WILDCARD_DOMAIN*.
Custom Admin Portal	Every tenant includes an Admin Portal. You can use the Admin Portal to create, update and delete 3scale resources in the tenant. The resources (products, backends, users...) that you create in the tenant are independent, sharing data across tenants is not possible.	*TENANT_NAME*`-admin.`*WILDCARD_DOMAIN*

Describing the Master Portal

The Master Portal is used to manage the tenants of the 3scale installation. The Master Portal graphic interface is similar to the Admin Portal one, but the Master Portal has a horizontal blue straight line at the top of every page that contains the word `Master`. The Master Portal's URL follows the pattern `master.`*`WILDCARD_DOMAIN`*.

Introducing Master Users

In the Master Portal context, a user allows you to connect to the Master Portal. By default, 3scale creates a user with the username `master` and saves its password in the `system-seed` OpenShift secret. The following command retrieves the default user's password and decodes it.

```
oc get secret system-seed -n 3scale \
--template={{.data.MASTER_PASSWORD}} | base64 -d
```

You can use these credentials to log in to the Master Portal.

To access the `Users` page, click **Account Settings** in the drop-down menu, located in the top pane of the Master Portal. Then, click **Users** > **Listing**.

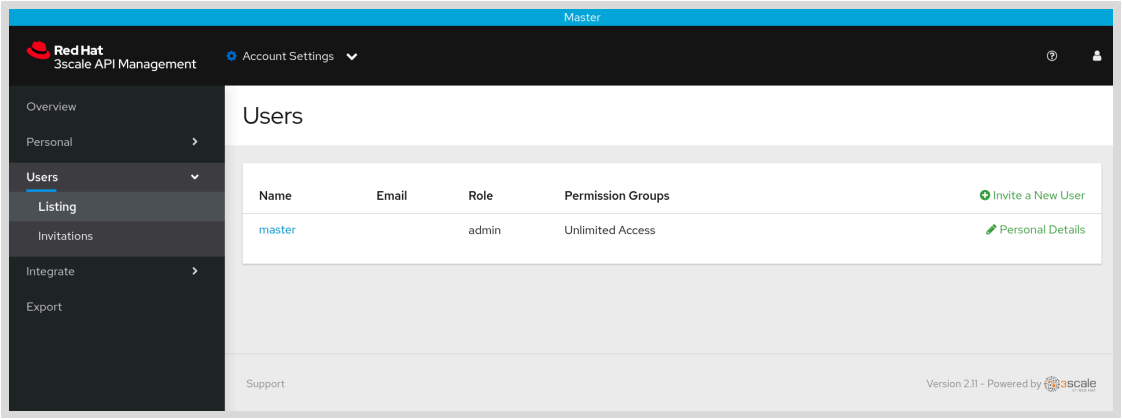


Figure 2.25: Master Portal User listing

The default `master` user is displayed. Click **Invite a New User** to create a new user by providing a valid email address. 3scale sends an email with a link to join the Master Portal. Click **Invitations** to list the invitations sent.

Introducing Master Accounts

In the Master Portal context, an `account` represents a tenant. When you create a master account, you are creating a new tenant.

To access the `Accounts` page, click **Audience** in the drop-down menu located in the top pane. This page lists the available accounts.

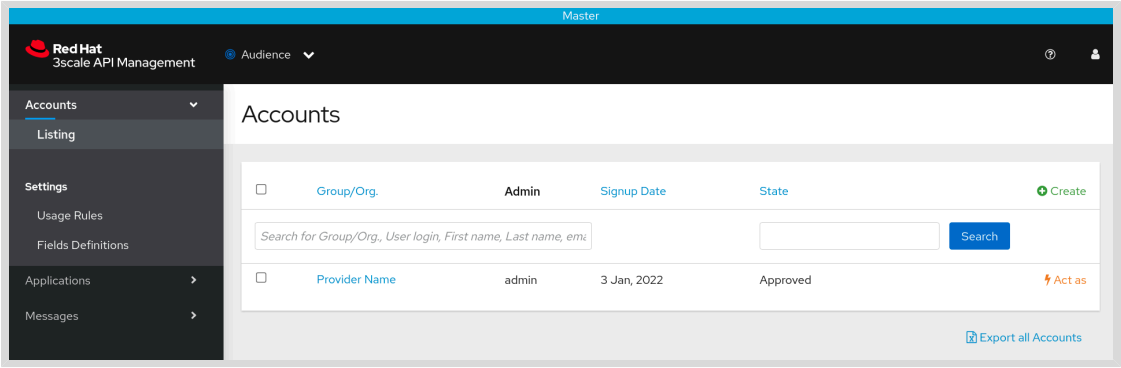


Figure 2.26: Master Portal Account listing

Click **Create** to create a new account. You must complete the form by taking into consideration what every field means.

Field	Explanation
Username	The username of the user that you use to access the Admin Portal of the tenant.
Password	The password of the user that you use to access the Admin Portal of the tenant.
Email	The email of the user that you use to access the Admin Portal of the tenant.
Organization/Group Name	The tenant name. This is used as a subdomain to generate the URLs of the Admin Portal, Developer Portal and APIcast.

For example, if you create a tenant with the following data:

Field	Value
Username	`do240-user`
Password	`do240-password`
Email	`do240@redhat.com`
Organization/Group Name	`do240`

Then the following URLs are created, considering that the wildcard domain is `example.com`:

Name	URL
Admin Portal	`do240-admin.example.com`
Developer Portal	`do240.example.com`
Staging APIcast URL	`api-do240-apicast-staging.example.com`
Production APIcast URL	`api-do240-apicast-production.example.com`

Describing the Tenant's Admin Portal

You can use Admin Portal to manage most of the features of the tenant, such as products and backends. You have already used the Admin Portal of the default tenant, `3scale`, in previous sections.

You can access the Admin Portal of your tenant by opening the URL *`TENANT_NAME`*`-admin.`*`WILDCARD_DOMAIN`* in a web browser. Log in by using the credentials (username and password) that you provided when creating the tenant.

Introducing Admin Portal Users

The Admin Portal Users allow you to connect to the Admin Portal. When you create a new tenant, you must provide a username and a password. These credentials are used to generate a default user for the Admin Portal of the tenant.

To access the `Users` page, click **Account Settings** in the drop-down menu of the Admin Portal. Then, click **Users** > **Listing** in the left pane. The list of users with access to the Admin Portal is displayed.

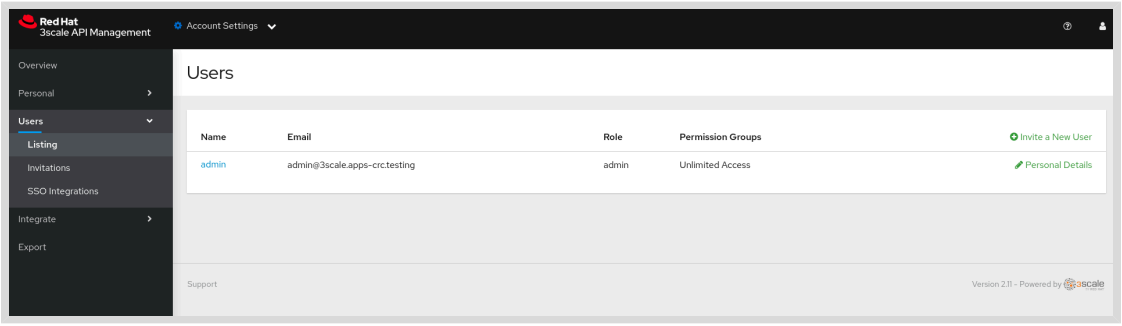


Figure 2.27: Admin Portal User listing

Click **Invite a New User** to create a new user by providing a valid email address. An email is sent to the email provided with a link to join the tenant. Click **Invitations** to list the invitations sent.

Describing a Multitenancy Example

Consider a clean 3scale installation with `WILDCARD_DOMAIN = apps.ocp4.example.com`. The `3scale` tenant is created by default with all the corresponding features, as shown in the following diagram:

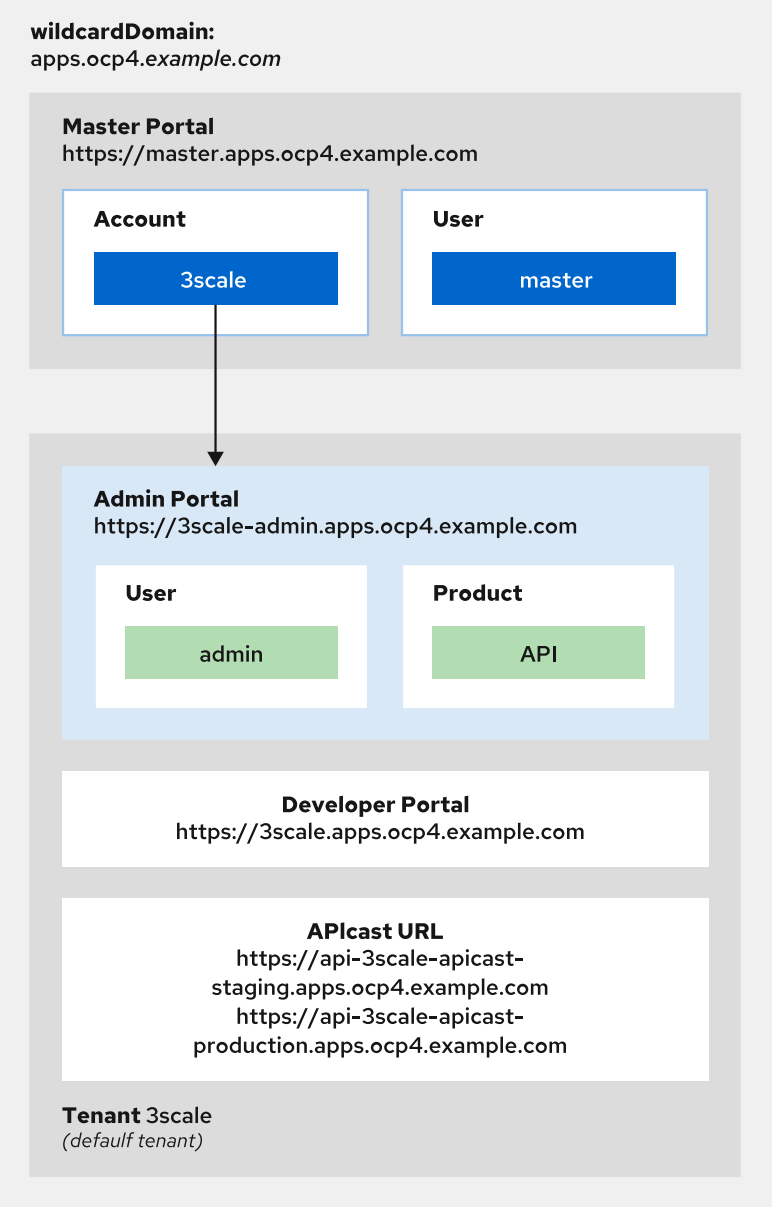


Figure 2.28: Default architecture of 3scale tenants

The Master Portal contains one account, `3scale`, which corresponds to the default `3scale` tenant. At the same time, the Master Portal has one user, `master`, which is the default user. You can log in to the Master Portal by opening the `https://master.apps.ocp4.example.com` URL in a web browser. Then, you can use the `master` user credentials to authenticate.

The `3scale` Admin Portal contains the default `admin` user, and the default `API` product. You can log in to the `3scale` Admin Portal by opening the `https://3scale-admin.apps.ocp4.example.com` in a web browser. Then, you can use the `admin` user credentials to authenticate.

The Developer Portal of the `3scale` tenant is available at `https://3scale.apps.ocp4.example.com`, and you can access your APIs at `https://api-3scale-apicast-staging.apps.ocp4.example.com`.

Consider also that your organization needs a separate tenant to manage the finance APIs independently. If you add a new `finance` Master account (i.e. tenant) with the following data:

- **Username**: `finance-user`
- **Password**: `finance-password`
- **Email**: `finance@redhat.com`
- **Organization/Group Name**: `finance`

Then the previous diagram displays as follows:

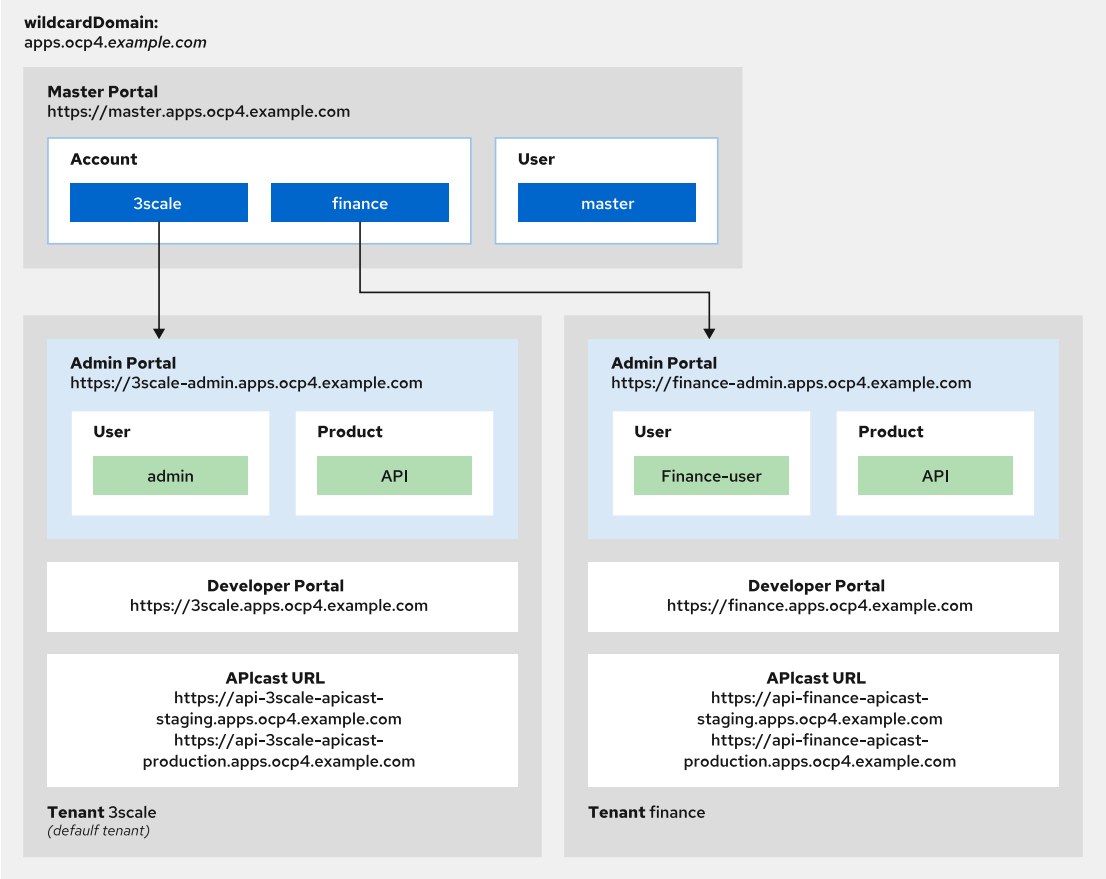


Figure 2.29: Tenants architecture with finance tenant added

The Master Portal contains two accounts, `3scale` and `finance`. The `finance` Admin Portal holds one user, `finance-user`, which is the username that you provided when creating the tenant. At the same time, the `finance` Admin Portal contains the `API` Product, which is created by default.

Note

The APIcast URLs displayed in the graphics assume that you are using the `API` default product. 3scale API Management provides a different URL prefix for every product by using the following pattern: *`PRODUCT`*`-`*`TENANT`*`-apicast-staging.apps.ocp4.example.com`.

Configuring SMTP

Because 3scale sends invitations through email, a SMTP server must be configured. 3scale does not provide an SMTP server by default, so if you want to create a Master Portal or Admin Portal user, then you must provide your own SMTP server. You can configure an SMTP server for the entire 3scale API Manager installation. Therefore, the Master Portal and all the tenants use the same email server.

When you have your SMTP server ready, you must configure 3scale to use the SMTP server. The `system-smtp` OpenShift secret contains the following parameters to connect to the server.

Parameter	Explanation
`address`	The host of the SMTP server.
`port`	Port where the server is listening for connections.
`username`	The username of the SMTP server authentication. Leave it blank if no authentication is required.
`password`	The password of the SMTP server authentication. Leave it blank if no authentication is required.
`domain`	The HELO domain.
`authentication`	The authentication type of the server. Values: `plain` (send the password in clear), `login` (sends the password base64 encoded), or `cram_md5`.
`openssl.verify.mode`	Specifies how OpenSSL verifies certificates when using TLS. Values: `none` or `peer`.

You can modify the `system-smtp` secret by using the OpenShift Console, or by executing the proper `oc patch` command. In the following command, you must replace both PARAMETER and VALUE by the parameter and value that you want to update, respectively.

```
oc patch secret system-smtp -p '{"stringData":{"PARAMETER":"VALUE"}}'
```

For example, the following command updates the `address` parameter.

```
oc patch secret system-smtp -p '{"stringData":{"address":"smtp"}}'
```

After you update the secret with the correct parameters for your SMTP server, you must restart the pods involved in the email sending.

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

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

References

3cale Multitenancy
https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html/installing_3scale/install-threescale-on-openshift-guide

Installing 3scale on OpenShift
https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.6/html/admin_portal_guide/multitenancy_2

Guided Exercise

Creating Multiple Tenants in Red Hat 3Scale API Management

In this exercise, you will use the Master Portal to create a new tenant. Then, you will use the tenant's Admin Portal to create a new user and modify the user's permissions.

Outcomes

You should be able to create a new 3scale tenant, create a new user for the tenant, and modify the user's permissions.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that the `workstation` machine is ready and the OpenShift cluster is available. At the same time, it deploys an email interceptor that you will use in the exercise.

```
[student@workstation ~]$ lab start applications-tenants
```

Instructions

1. Log in to the 3scale Master Portal by using the default installation password.

Important

Because the `start` function restarts some OpenShift pods to deploy the email server, the Master Portal might be temporary unavailable. If the Master Portal is not available, then try again in a few minutes.

1.1. Log in to OpenShift as the `admin` user.

```
[student@workstation ~]$ oc login -u=admin -p=redhat \
  --server=https://api.ocp4.example.com:6443
Login successful.
```

1.2. In a command-line terminal, retrieve the master password from the `system-seed` OpenShift secret.

```
[student@workstation ~]$ oc get secret system-seed -n 3scale \
--template={{.data.MASTER_PASSWORD}} | base64 -d
...output omitted...
```

Copy the password.

1.3. In a web browser, navigate to `https://master.apps.ocp4.example.com/`. Log in with the following credentials:

- **Username:** `master`
- **Password:** *Password retrieved in the previous step*

2. Create a new tenant called `do240` by using the Master Portal. Remember that in 3scale a tenant is a master account.

2.1. Click **Dashboard** to display the drop-down menu. Then, click **Audience**.

2.2. The `Accounts` page is displayed by default. This page lists all the available master accounts.

2.3. Click **Create** to create a new account.

2.4. Complete the form according to the following information:

- **Username:** do240-user
- **Email:** do240@redhat.com [mailto:do240@redhat.com]
- **Password:** do240
- **Password confirmation:** `do240`
- **Organization/Group name:** `do240`

Then, click **Create**. Remember that the organization name provided is used to create the Admin Portal subdomain.

2.5. The `do240` account page gives some information, such as the public domain of the tenant or the Admin Portal URL. By default, 3scale creates two users: `do240-user`, with the credentials that you provided previously, and another one called `3scale Admin`, which is used by the system. Both users have `admin` privileges.

3. Activate the `do240-user` so that you can use it to access the `do240` tenant Admin Portal.

3.1. Click **2 Users** in the top pane. The page displays the two created users for the tenant.

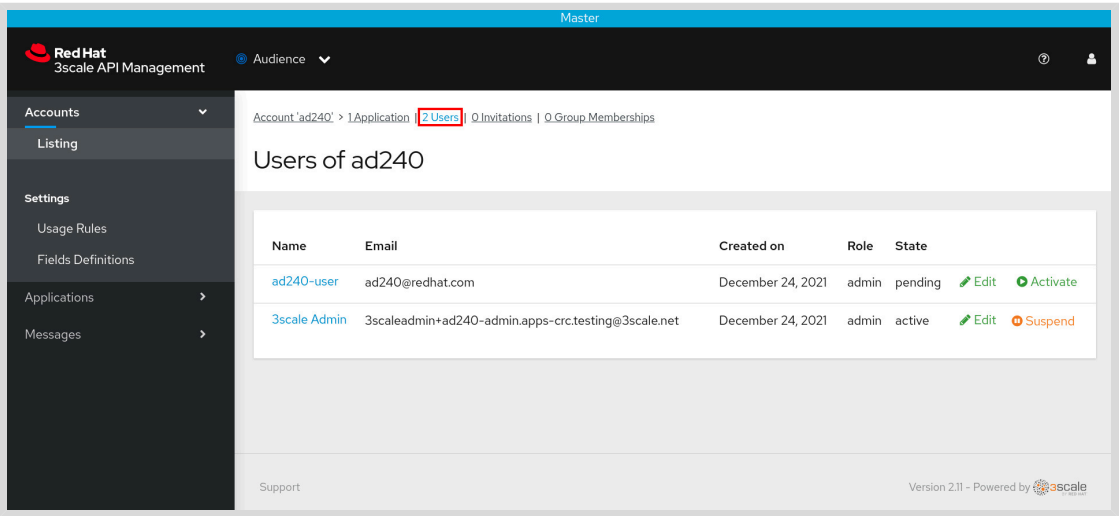


Figure 2.30: Account users in the Master Portal

3.2. Click **Activate** in the `do240-user` row to activate the user. Now you can use this user to access the `do240` tenant Admin Portal.

4. Verify that you can log in to the do240 tenant Admin Portal.

4.1. In a new tab, navigate to `https://do240-admin.apps.ocp4.example.com`, the `do240` tenant Admin Portal. Log in with the credentials that you provided in the previous steps.

- **Username:** `do240-user`
- **Password:** `do240`

4.2. If the `How does 3scale work` page displays, click **X** to close it. The dashboard shows the default `API` product.

5. Invite a new user to the `do240` tenant by using the Admin Portal. Then, use the link provided in the email to join the `do240` tenant.

5.1. In the `do240` tenant Admin Portal, click **Dashboard** to display the dropdown menu. Then, click **Account Settings**.

5.2. Click **Users**, and then click **Listing**. This section shows all the users that can log in to the Admin Portal. It is synchronized with the `Users` section of the `do240` tenant in the Master Portal.

5.3. In the left pane, click **Invitations**, and then click **Invite a New Team Member**.

5.4. In a real-world scenario, you should complete the form with the email of the person that you want to have access to the tenant. For the purpose of this exercise, use the `gls@redhat.com` email address. Then, click **Send**.

Important

In this exercise, the email is not sent. The `lab start` function that you executed at the beginning of the exercise deploys an email interceptor. You can fetch the email that would be sent in a production environment by using the email interceptor.

5.5. Log out of the `do240` tenant Admin Portal. Click on the `Session` icon in the top pane, and then click **Sign Out**.

5.6. In a command-line terminal, run the `/scripts/get-emails.sh` script in the `DO240-apps` repository to get the emails sent by 3scale.

```
[student@workstation ~]$ ~/DO240-apps/scripts/get-emails.sh
---------- MESSAGE FOLLOWS ----------
...output omitted...
You have been invited to join Provider Name on 3scale platform.

Please sign up by following this link: https://do240-admin.apps.ocp4.example.com/
p/signup/c9d30638114c6c9433867c5775689278

If you have any problems signing up or believe you received this email
 erroneously, please open a Support Case at https://access.redhat.com/support.
```

```
Thank you,

The 3scale API Team.
------------ END MESSAGE ------------
```

The email contains a link to join the tenant. Copy the link.

5.7. In a web browser, open the link from the previous step. Complete the sign-up form according to the following data:

Important

If you are still logged-in with the `do240-user`, then you cannot see the sign-up form. Log out first and try again.

- **Username:** gls-user
- **First name:** GLS
- **Last name:** Red Hat
- **Password:** gls-password
- **Password confirmation:** gls-password

Then, click **Sign up**. You are redirected to the log-in page.

5.8. Log in to the `do240` tenant Admin Portal with the credentials from the previous step.

- **Username:** gls-user
- **Password:** gls-password

By default, invited users get the `member` role assigned. Therefore, you do not have permissions to see any products.

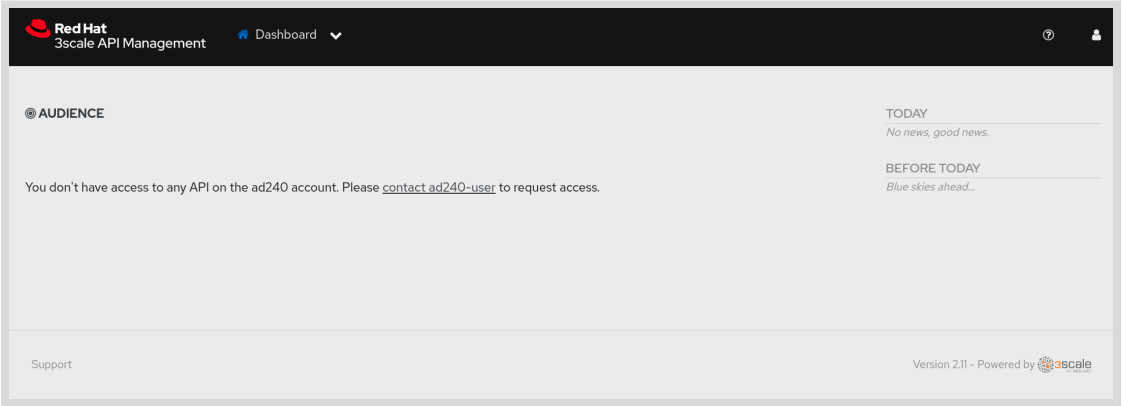


Figure 2.31: The gls-user does not have permissions to see any products.

5.9. Log out of the `do240` tenant Admin Portal. Click on the `Session` icon in the top pane, and then click **Sign Out**.

6. Change the `gls-user` user permissions, so that the user can view the `API` product.

6.1. Log in with the administrator credentials again.

- **Username:** `do240-user`
- **Password:** `do240`

6.2. Click **Dashboard** to display the dropdown menu. Then, click **Account Settings**.

6.3. Click **Users**, and then click **Listing**. The `GLS Red Hat` user (`gls-user`) is now displayed in this page. Click over its name to edit it.

6.4. The `Administrative` section allows you to change the role of user. The `Admin` role provides full access to the tenant. The `Member` role allows you to choose the sections that the user can access.

6.5. Click the `Create, read, update and delete developer accounts and applications of selected API products` checkbox.

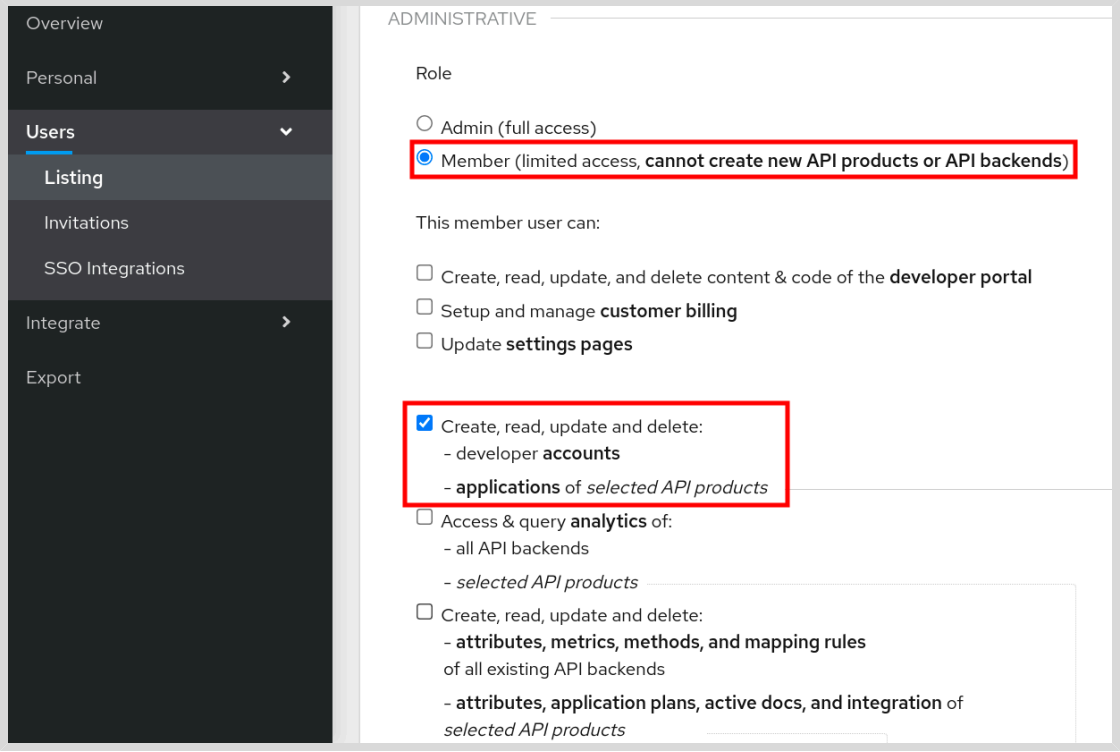


Figure 2.32: Enable the user to create accounts and applications

6.6. When you click this checkbox, the `All current and future API products` check is displayed. With this option, you grant access to all the products.

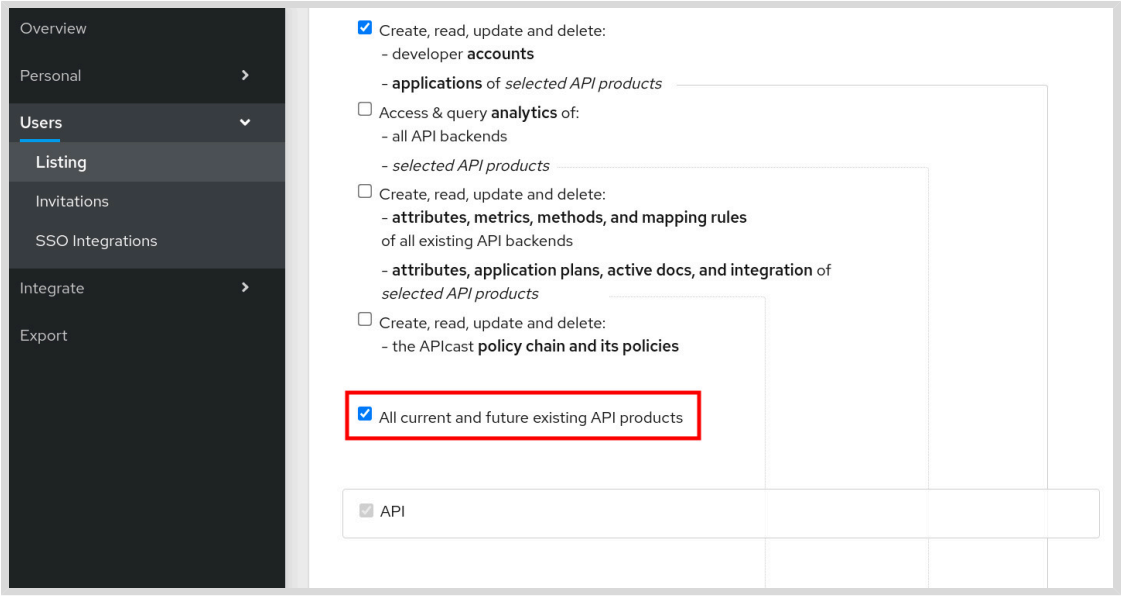


Figure 2.33: Enable the user to view all the products

6.7. Then, click **Update User**.

6.8. Log out of the `do240` tenant Admin Portal. Click on the `Session` icon in the top pane, and then click **Sign Out**.

7. Verify that the `gls-user` user can view the `API` Product.

7.1. Log in with the `gls-user` credentials:

- **Username:** `gls-user`
- **Password:** `gls-password`

The `Dashboard` section displays the `API` Product.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish applications-tenants
```

This concludes the guided exercise.

Quiz

Managing APIs with Red Hat 3scale API Management

Your company is releasing a new B2B product that includes paid-for access to APIs. You are tasked with configuring 3scale so that each of your client companies has their own access portal and none of the companies share information with one another.

The REST APIs your company exposes are as follows:

Fleet Management: `https://fleet.example.com:8080`

Metrics & Reporting: `https://reporting.example.com:3000`

GPS Location Tracking: `https://gps.example.com:8181`

Based on the preceding scenario, choose the correct answers to the following questions:

1. Which feature provides the necessary separation between the different companies?

a. Policies
b. Applications
c. Multi-tenancy
d. Client Separators

2. If you have four different client companies that each need access to all of your APIs, how many total 3scale backends do you need to create?

a. One
b. Three
c. Four
d. Twelve

3. One of your clients, Shadowman Software Inc, operates under a tenant named `shadowman`. Which of the following is the correct command to create a product named `trucking-product`?

a. `3scale service create shadowman trucking-product`
b. `3scale service new trucking-product`
c. `3scale product create shadowman trucking-product`
d. `3scale product create trucking_product`

4. Which set of objects are the minimum you must create to make an external request to an internal API via 3scale?

a. A product, backend, application, and application plan
b. Only a product and backend
c. Only an application and application plan
d. None, only the default objects are needed

5. For the Shadowman Software tenant, which of the following statements best describes how to create a backend for the Fleet Management service?

a. `3scale backend create shadowman fleet-management`

b. `3scale use-tenant shadowman` and `3scale create-backend fleet-management`

c. Use the Admin Portal web UI to create a backend specifying the URL `https://fleet.example.com:8080`.

d. Backends are created automatically, so you do not need to create one.

6. When creating an application plan via the 3scale Toolbox CLI, which of the following is the correct command option to specify a cost per month of $10?

a. `--cost-per-month=10.00`

b. `--cost 10`

c. `--cpm 10`

d. There is no way to specify the cost per month via the 3scale Toolbox CLI.

Solution

Managing APIs with Red Hat 3scale API Management

Your company is releasing a new B2B product that includes paid-for access to APIs. You are tasked with configuring 3scale so that each of your client companies has their own access portal and none of the companies share information with one another.

The REST APIs your company exposes are as follows:

Fleet Management: `https://fleet.example.com:8080`

Metrics & Reporting: `https://reporting.example.com:3000`

GPS Location Tracking: `https://gps.example.com:8181`

Based on the preceding scenario, choose the correct answers to the following questions:

1. Which feature provides the necessary separation between the different companies?

a. Policies
b. Applications
c. Multi-tenancy
d. Client Separators

2. If you have four different client companies that each need access to all of your APIs, how many total 3scale backends do you need to create?

a. One
b. Three
c. Four
d. Twelve

3. One of your clients, Shadowman Software Inc, operates under a tenant named `shadowman`. Which of the following is the correct command to create a product named `trucking-product`?

a. `3scale service create shadowman trucking-product`
b. `3scale service new trucking-product`
c. `3scale product create shadowman trucking-product`
d. `3scale product create trucking_product`

4. Which set of objects are the minimum you must create to make an external request to an internal API via 3scale?

a. A product, backend, application, and application plan
b. Only a product and backend
c. Only an application and application plan
d. None, only the default objects are needed

5. For the Shadowman Software tenant, which of the following statements best describes how to create a backend for the Fleet Management service?

a. `3scale backend create shadowman fleet-management`

b. `3scale use-tenant shadowman` and `3scale create-backend fleet-management`

c. Use the Admin Portal web UI to create a backend specifying the URL `https://fleet.example.com:8080`.

d. Backends are created automatically, so you do not need to create one.

6. When creating an application plan via the 3scale Toolbox CLI, which of the following is the correct command option to specify a cost per month of $10?

a. `--cost-per-month=10.00`

b. `--cost 10`

c. `--cpm 10`

d. There is no way to specify the cost per month via the 3scale Toolbox CLI.

Summary

In this chapter, you learned:

- How to create and manage 3scale products, backends, applications, and application plans.
- How to apply rate limits using application plans.
- How to manage API versions and the 3scale approach to API management.

Chapter 3

Managing and Customizing API Gateways

Goal

Configure and customize the API gateway using standard policies.

Objectives

- Allow external access to your API by configuring APIcast gateways.
- Customize request processing using APIcast policies.

Sections

- Creating Self Managed APIcast Gateways (and Guided Exercise)
- Configuring Standard APIcast Policies (and Guided Exercise)
- Managing and Customizing API Gateways (Quiz)

Creating Self Managed APIcast Gateways

Objectives

After completing this section, you should be able to allow external access to your API by configuring APIcast gateways.

Introducing APIcast Gateway

APIcast is an NGINX-based gateway in Red Hat 3scale API Management that is responsible for receiving and managing the incoming traffic from your APIs. The APIManager component contains information about mapping policies and rate limits and APIcast routes the traffic based on those rules.

3scale API Management provides two environments: `staging` and `production`. By default, 3scale API Management provides an APIcast gateway for each of these environments. Promoting the configuration of a product means propagating the changes to the specific APIcast gateway.

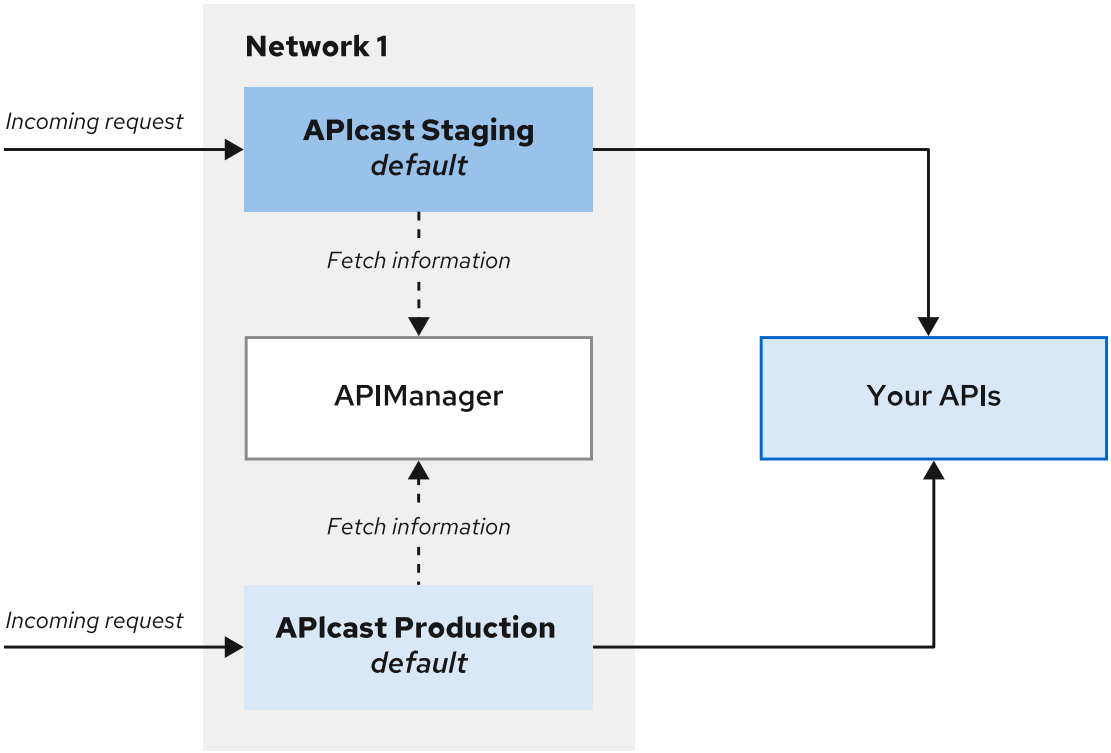


Figure 3.1: Default 3scale API Management installation

To have full control over the APIcast gateway, you can avoid using the default gateways. You might choose to deploy a self-managed gateway if you want to manage the full lifecycle of the application.

You can replace any of the default gateways (`staging` and `production`) with your self-managed gateway.

For example, consider that you have a 3scale API Management installation, with two default gateways. Consider also that you want to deploy a self-managed gateway for the production environment in a separate network. The self-managed gateway receives the incoming requests for the production environment and must fetch the information, such as the routing policies and rate limits, from the APIManager. The following diagram illustrates this behavior.

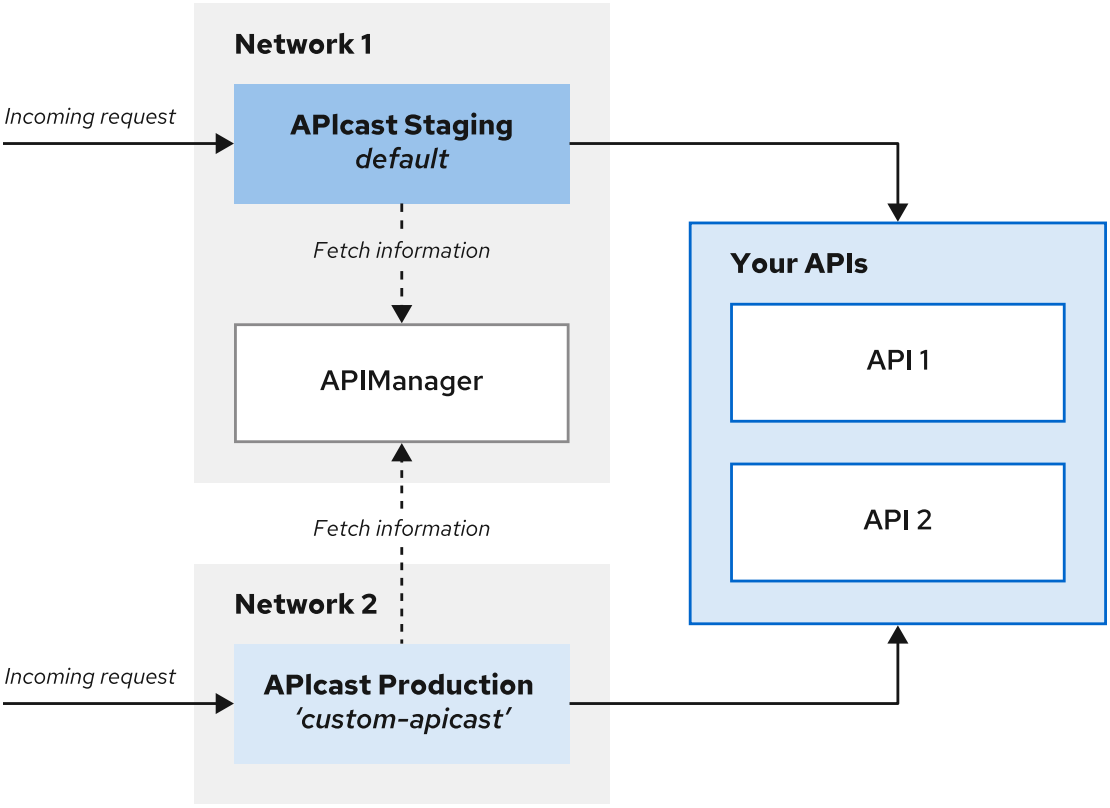


Figure 3.2: Self-managed gateway for the production environment

Deploying a Self-managed APIcast Gateway in Red Hat OpenShift Container Platform (RHOCP)

You can deploy an APIcast gateway by using a template or the official RHOCP operator. In this course, only the operator deployment is used.

Introducing the APIcast Gateway Operator

The APIcast gateway Operator makes it easier to deploy and manage a self-managed APIcast gateway. The operator includes an RHOCP custom resource definition (CRD), which you can use to set several configuration parameters of the gateway.

You can install the operator by using the RHOCP OperatorHub catalog.

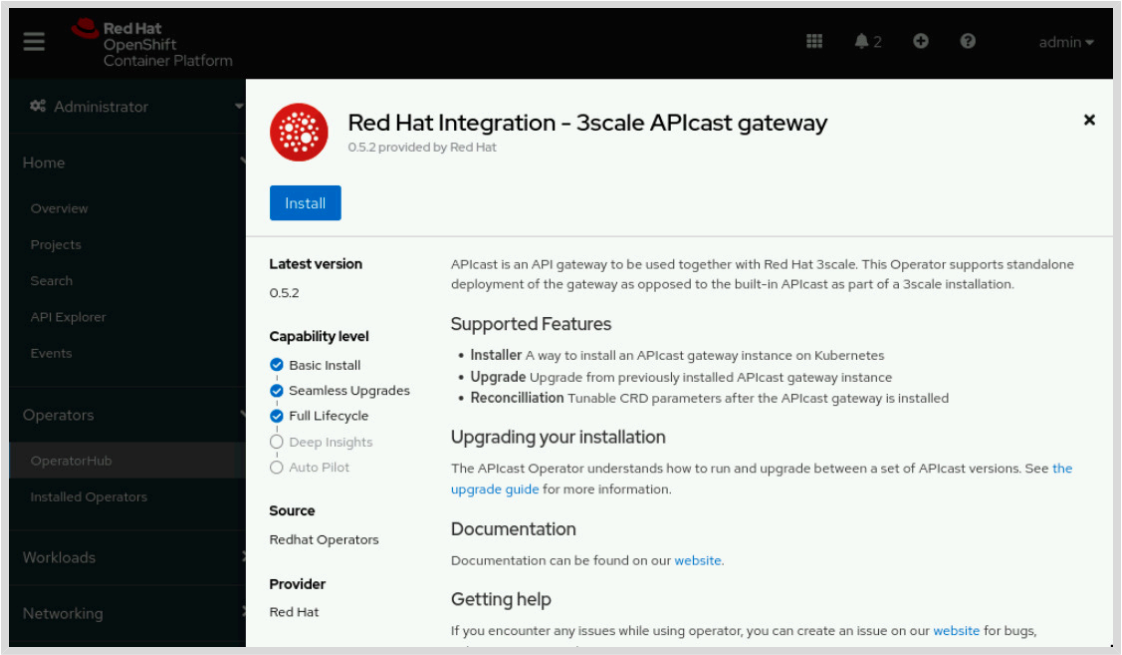


Figure 3.3: The APIcast gateway operator in the OperatorHub

Deploying an APIcast Custom Resource

The APIcast gateway operator includes the `APICast` custom resource, which you can use to deploy a new gateway. A sample `APIcast` manifest displays as follows:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: custom-apicast (1)
spec:
  adminPortalCredentialsRef:
    name: apicast-secret (2)
  deploymentEnvironment: production (3)
  exposedHost:
    host: custom-apicast.apps.ocp4.example.com (4)
    tls:
    - {}
```

(1) Name of the gateway.

(2) Name of the RHOCP secret that contains the authentication against the APIManager. This authentication is covered in detail later in this lecture.

(3) Environment that the gateway replaces (`staging` or `production`).

(4) RHOCP route where the gateway is accessible externally.

The APIcast operator creates an RHOCP route with the host provided in the manifest, so that your gateway is accessible externally.

Authenticating the Self-managed APIcast Gateway

The self-managed gateway must get routing information from the APIManager. Consequently, you must authenticate the gateway by using an RHOCP secret. Because you authenticate the gateway against a specific tenant, you must include an access token and the tenant's Admin Portal URL in the RHOCP secret.

```
[user@host]$ oc create secret generic SECRET_NAME \
 --from-literal=AdminPortalURL=https://ACCESS_TOKEN@TENANT_ADMIN_PORTAL
```

To create a new access token, navigate to the `Account Settings` page of your tenant's Admin Portal, and click **Personal** > **Tokens**. Then, click **Add Access Token**. The token must include the `Account Management API` scope.

For example, if you want to authenticate the gateway against the default tenant, `3scale`, and you have an access token `1234`, then the secret displays as follows:

```
[user@host]$ oc create secret generic apicast-secret \
 --from-literal=AdminPortalURL=https://1234@3scale-admin.apps.ocp4.example.com
```

Then, you authenticate your gateway by setting the `spec.adminPortalCredentialsRef.name` parameter in the `APIcast` manifest.

Using the Self-managed APIcast Gateway

When you create a 3scale API Management product, the default gateways are used. If you want to use your self-managed gateway, then you must change the settings of the product.

To change the settings of a product, click **Integration** > **Settings** in the product's page left pane. In the `DEPLOYMENT` section, select `APIcast self-managed` to enable a self-managed gateway for the product. Then, you use the `Staging Public Base URL` and `Production Public Base URL` fields to provide the external URL of your gateway for the staging and production environments (respectively).

For example, if you provide the `custom-apicast.apps.ocp4.example.com` host in the `APIcast` manifest, then you use the `https://custom-apicast.apps.ocp4.example.com:443` URL.

Deploying a Self-managed APIcast Gateway in Other Platforms

Although the main focus of this course is on deploying your self-managed gateway in RHOCP, you can use other platforms depending on your needs.

To deploy APIcast Gateway in your own server, you must install OpenResty and other dependencies. You can find a detailed installation guide in this link [https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.0/html/deployment_options/apicast-self-managed].

To deploy APIcast Gateway in a Docker containerized environment, you must download a ready to use container image. This image contains all the dependencies needed to run APIcast. You can find a detailed installation guide in this link [https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.0/html/deployment_options/apicast-docker].

References

Red Hat 3scale Documentation - Administering the API Gateway

https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/administering_the_api_gateway/index

Guided Exercise

Creating Self Managed APIcast Gateways

In this exercise, you will deploy a new APIcast Gateway that you will use in the staging environment.

Outcomes

You should be able to:

- Install the APIcast Gateway operator.
- Deploy a new APIcast Gateway in a different OpenShift project.
- Create an OpenShift route to access to the gateway externally.
- Test the gateway with a sample 3scale product.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start gateways-apicast
```

The `start` function creates a 3scale product called `apicast`, which points to a backend that uses the 3scale Echo API. You will use this product in this exercise to test the self-managed APIcast.

Instructions

1. Create a new OpenShift project called `apicast`, where you will deploy the APIcast custom resource. By deploying it in a different namespace, you simulate the logical isolation that you might require in a real-world scenario.

   1.1. Log in to RHOCP:

```
[student@workstation ~]$ oc login \
-u=admin -p=redhat --server=https://api.ocp4.example.com:6443
...output omitted...
```

   1.2. In a your command-line terminal, create the `apicast` project.

```
[student@workstation ~]$ oc new-project apicast
...output omitted...
```

2. Install the APIcast Gateway operator.

   2.1. In a web browser, navigate to `https://console-openshift-console.apps.ocp4.example.com`, and log in to the OpenShift Console with the following credentials by using the `htpasswd` provider:

- **Username:** admin
- **Password:** redhat

2.2. In the left pane, click **Operators** > **OperatorHub**.

2.3. Type `APIcast` in the search text box input, and select the `Red Hat Integration - 3scale APIcast gateway` option. Then, click **Install**.

2.4. Complete the `Install Operator` form with the following options.

- **Installed Namespace:** `apicast`

Leave other fields with the default selected options. Then, click **Install**. The operator takes some time to install.

3. Create a personal access token in the default `3scale` tenant, so that the APIcast operator can fetch the information from the APIManager.

3.1. In your command-line terminal, use the `oc get secret` command to retrieve the password for the `3scale` tenant Admin Portal, where you will create the access token.

```
[student@workstation ~]$ oc get secret system-seed -n 3scale \
--template={{.data.ADMIN_PASSWORD}} | base64 -d; echo
```

The default passwords of 3scale are stored in the `system-seed` secret. By executing the previous command, you retrieve the `ADMIN_PASSWORD` entry of the `system-seed` secret. Then, because the secret is encoded in Base64, you decode it.

3.2. Navigate to `https://3scale-admin.apps.ocp4.example.com`, the `3scale` tenant Admin Portal. Log in with the the following credentials:

- **Username**: admin
- **Password**: **Password from the `system-seed` secret**

3.3. In the top pane, click **Dashboard** to display the drop-down menu. Then, click **Account Settings**.

3.4. In the left pane, click **Personal** > **Tokens**.

3.5. In the `Access Tokens` section, click **Add Access Token**. Complete the form according to the following values:

- **Name:** `apicast`
- **Scopes:** `Account Management API`
- **Permission:** `Read Only`

Then, click **Create Access token**. Save the access token displayed.

3.6. Create an OpenShift secret that contains the access token. You will use this secret later to authenticate the APIcast operator.

```
[student@workstation ~]$ oc create secret generic apicast-secret \
--from-
literal=AdminPortalURL=https://ACCESS_TOKEN@3scale-admin.apps.ocp4.example.com
secret/apicast-secret created
```

In the preceding command, replace ACCESS_TOKEN with your access token.

4. Deploy the APIcast custom resource. Use the the OpenShift secret created in the previous step.

   4.1. By using a text editor of your preference, create a new file called `apicast.yaml` that contains the following snippet.

   You can use the `/home/student/DO240/labs/gateways-apicast/apicast.yaml` file.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: custom-apicast
spec:
  adminPortalCredentialsRef:
    name: apicast-secret
  deploymentEnvironment: staging
  exposedHost:
    host: custom-apicast.apps.ocp4.example.com
    tls:
    - {}
  resources:
    limits:
      cpu: '0'
      memory: 128Mi
```

The `deploymentEnvironment` parameter sets what environment this gateway replaces (either `staging` or `production`). The `host` parameter is the external route where your gateway is accessible.

   4.2. Apply the APIcast resource by using the `oc` command.

```
[student@workstation ~]$ oc apply -f apicast.yaml
apicast.apps.3scale.net/custom-apicast created
```

5. Update the `gateways_apicast` product settings to use the self-managed APIcast Gateway. The `gateways_apicast` product was created for you when you executed the `start` function of this exercise.

   5.1. In the `3scale` tenant Admin Portal, click **Products** in the drop-down menu. Then click **gateways_apicast**.

   5.2. In the left pane menu, click **Integration** > **Settings**. The settings page for the `gateways_apicast` product is displayed.

   5.3. In the `DEPLOYMENT` section, select `APIcast self-managed`.

5.4. In the `Staging Public Base URL` field, type the OpenShift route that you created previously, `https://custom-apicast.apps.ocp4.example.com`. Then, click **Update Product**.

6. Promote the new configuration to Staging.

6.1. In the left pane, click **Integration** > **Configuration**.

6.2. Click **Promote v. 1 to Staging APIcast** to replace the default Staging APIcast by your self-managed APIcast.

6.3. Copy the `curl` example and execute it in a command-line terminal.

```
[student@workstation ~]$ curl "https://custom-apicast.apps.ocp4.example.com:443/
echo?user_key=USER_KEY"
{
  "method": "GET",
  "path": "/",
  ...output omitted...
  "headers": {
    "HTTP_VERSION": "HTTP/1.1",
    "HTTP_HOST": "echo-api.3scale.net",
    ...output omitted...
  },
  ...output omitted...
}
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish gateways-apicast
```

This concludes the guided exercise.

Configuring Standard APIcast Policies

Objectives

After completing this section, you should be able to customize request processing using APIcast policies.

Establishing Policies' Role in the System

Red Hat 3scale API Management includes APIcast, which configures an underlying NGINX web server to act as a proxy for your APIs. Many objects in 3scale API Management, such as products and backends, are abstractions for this configuration.

Policies enable higher flexibility and control in how APIcast handles traffic by providing more-granular abstractions for the NGINX proxy configuration.

Configured policies exist within a product as part of a *policy chain*, which is an ordered list of policies that every request to the product is filtered through.

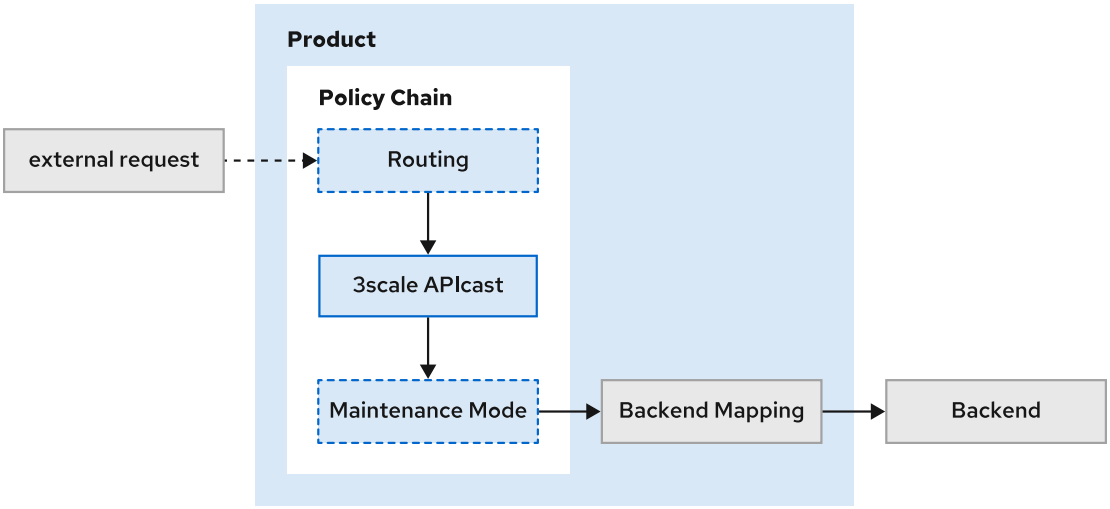


Figure 3.4: Example of a policy chain processing a request.

In the preceding example diagram, blue boxes represent the policies, and dashed outlines show which policies are optional. Note that the default `3scale APIcast` policy is required, as it represents 3scale API Management's routing configuration.

Each policy within a chain can alter or add information to the request before it is sent to the next policy.

Alternatively, a policy might conditionally alter how the request is handled. For example, an authorization policy might short-circuit a request and reject it before it reaches later polices or an API. In this way, policies can potentially improve the performance or resource usage of your system.

Exploring the Standard Policies

The standard policies provided by 3scale API Management address various needs, including the following:

- Routing
- Authentication and authorization
- Request and data conversion
- Utility

Policies that customize how APIcast routes incoming requests include the following:

- `Routing`: a general-purpose policy that gives you fine-grained control over how requests are routed.
- `Header Modification`: adds or removes specific headers from requests *or* responses.
- `IP Check`: configures allowlists and blocklists by using a list of IP addresses or blocks of addresses.

The collection of standard policies also includes policies to handle authentication and authorization tasks, such as the following:

- `RH-SSO/Keycloak Role Check`: integrates with RH-SSO to verify realm roles as a request allowlist or blocklist.
- `JWT Claim Check`: validates JWTs from requests and either allows or blocks the request based on claims within the token.
- `OAuth 2.0 Token Introspection`: communicates with an OIDC token issuer to authorize requests by using tokens.
- `Anonymous Access`: removes the `user_key` parameter requirement from all requests to the product by supplementing the request with a pre-configured user token.

Policies that provide miscellaneous utility functions include the following:

- `Custom Metrics`: adds hooks into the response to update custom metrics created in the Admin Portal.
- `Retry`: provides basic retry logic to reattempt failed requests to an API.
- `Maintenance Mode`: responds to all requests with a preconfigured status code and message, which is useful for temporarily disabling the product for maintenance.

Adding a Policy to a Product

Currently, you can manage the policies of a product in two ways.

Using the toolbox CLI, you can export and import policy chains by using the `3scale policies` super-command. For example, the following commands export an example policy chain as YAML and then re-imports it:

```
[student@workstation ~]$ 3scale policies export 3scale-tenant \
policy_sandbox >> policy_chain.yml
[student@workstation ~]$ cat policy_chain.yml
---
- name: routing
```

```
    version: builtin
    configuration: {}
    enabled: true
  - name: apicast
    version: builtin
    configuration: {}
    enabled: true
  - name: maintenance_mode
    version: builtin
    configuration: {}
    enabled: true
[student@workstation ~]$ 3scale policies import 3scale-tenant \
policy_sandbox -f policy_chain.yml
```

The name of the product is `policy_sandbox` and the `-f` option specifies the name of the file to import.

In most cases, you use the Admin Portal to manage policies. To add a policy via the Admin Portal, use the **Integration** > **Policies** section within the product. This graphical editor enables you to update the product's policy chain.

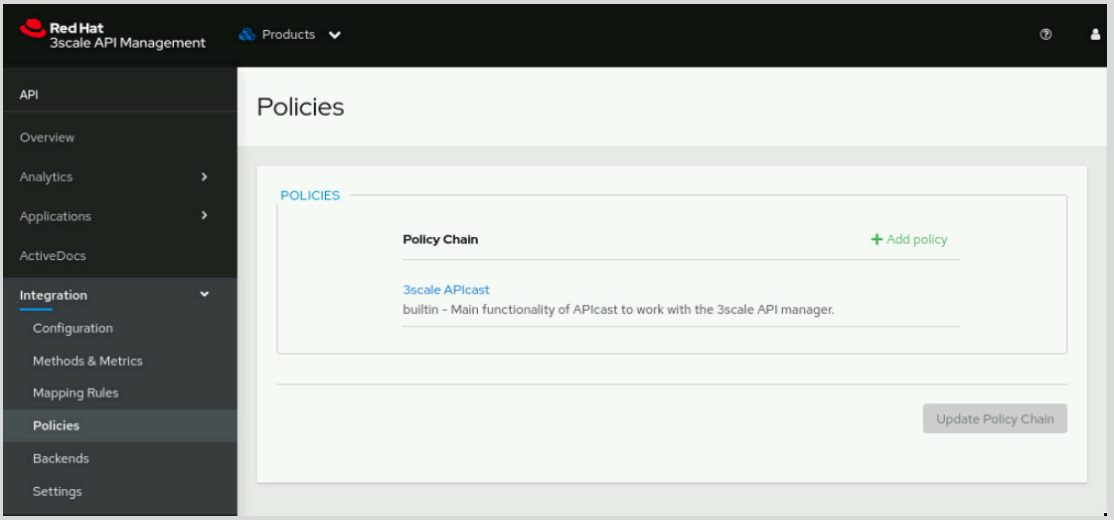


Figure 3.5: The Policies section within a product in the Admin Portal.

Once added, individual policies can be updated, removed, and reordered within the policy chain.

In the accompanying guided exercise, you practice adding, updating, reordering, and removing policies from a product by using the Admin Portal.

References

For a complete list of available policies, as well as more extensive documentation for the standard policies, please visit the `APIcast policies` chapter of the 3scale API Management documentation at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html/administering_the_api_gateway/apicast_policies

Guided Exercise

Configuring Standard APIcast Policies

In this exercise, you will use APIcast policies to customize how requests to a product are handled.

Outcomes

You should be able to add, update, reorder, and delete policies within a product.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command creates a new product called `gateways_policies` to house the policies. This product uses a basic echo backend that responds with information about the request.

```
[student@workstation ~]$ lab start gateways-policies
```

Instructions

1. Explore the product and backend before applying changes via policies.

   1.1. In a web browser, navigate to the Admin Portal hosted at `https://3scale-admin.apps.ocp4.example.com`.

   1.2. Using the top pane navigation drop-down, click **Products**, and then click **gateways_policies**.

   1.3. Navigate to **Integration** > **Configuration** and click **Promote v.1 to Staging APIcast** to promote the configuration to the staging environment.

   1.4. In a command-line terminal, run the example `curl` command. Be sure to use the correct `user_key` value.

```
[student@workstation ~]$ curl \
 https://gateways-policies-3scale-apicast-staging.apps.ocp4.example.com\
:443/echo?user_key=some-user-key
{
  "method": "GET",
  "path": "/",
  ...output omitted...
}
```

   Notice that the response echoes information from the request.

   1.5. Copy *just* the generated `user_key` value from the example `curl` command. You will use this in one of the following steps to configure a policy.

   For example, if your `curl` command is similar to the one in the preceding step, then the `user_key` to copy would be `some-user-key`.

1.6. Attempt to make a request to the endpoint without providing a `user_key`. This request fails due to missing authentication parameters.

```
[student@workstation ~]$ curl \
 https://gateways-policies-3scale-apicast-staging.apps.ocp4.example.com:443/echo
Authentication parameters missing
```

2. Create a policy that allows unauthenticated access to the product.

2.1. Within the Admin Portal, navigate to **Integration** > **Policies**.

Notice the existing policy chain contains a single policy: `3scale APIcast`.

2.2. Add a policy to this chain by clicking **Add policy**. Select `Anonymous Access` from the list of available policies.

2.3. Update the policy by clicking **Anonymous Access** within the policy chain. Ensure the policy is enabled and `user_key` is selected for the `auth_type` field.

Paste your user key value into the `user_key` field and click **Update Policy**.

2.4. Within the policy chain, use the arrow icon to drag the `Anonymous Access` policy so that it comes before the `3scale APIcast` policy.

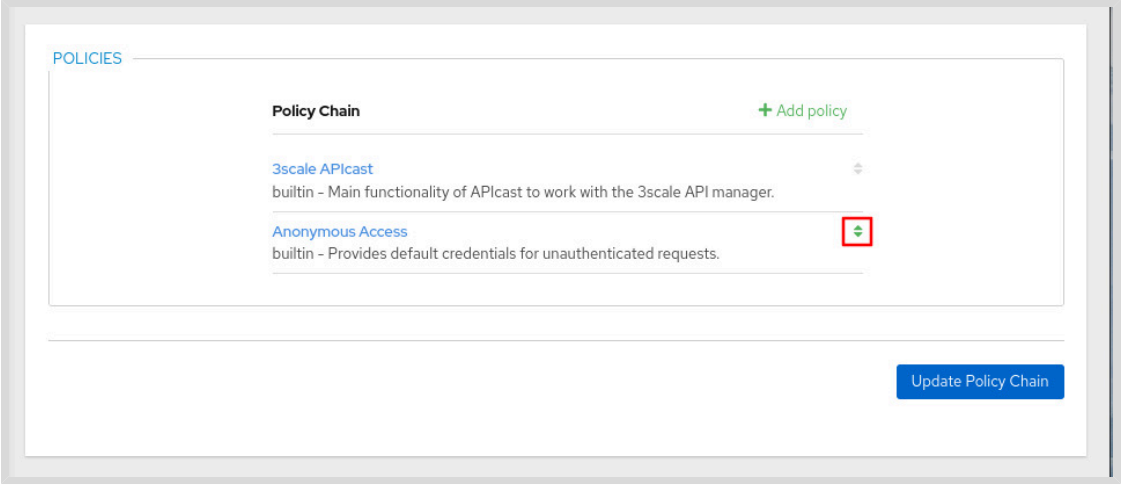


Figure 3.6: Reorder the policies so that Anonymous Access comes first.

2.5. Click **Update Policy Chain** to apply the new policy to the policy chain.

2.6. Deploy the changes to the staging environment by navigating to **Integration** > **Configuration** and clicking **Promote v.2 to Staging APIcast**.

2.7. From a command-line terminal, run the example `curl` command, but without providing the `user_key`.

```
[student@workstation ~]$ curl \
 https://gateways-policies-3scale-apicast-staging.apps.ocp4.example.com:443/echo
{
  "method": "GET",
  "path": "/",
  ...output omitted...
}
```

Notice that the request is processed by the echo API without providing a user key. Instead, the policy provides the configured user key.

Note

Use caution when applying the `Anonymous Access` policy, as anybody with a valid URL can make requests to any API available to the product. In particular, these requests have the same permissions as the user whose key is configured within the policy.

3. Create a policy that puts the product in maintenance mode and returns an appropriate response.

   3.1. Navigate back to **Integration** > **Policies**.

   3.2. Add a policy to this chain, selecting the `Maintenance Mode` policy.

   3.3. Click **Update Policy Chain** to apply the new policy.

   3.4. Deploy the changes to the staging environment by navigating to **Integration** > **Configuration** and clicking **Promote v.3 to Staging APIcast**.

   3.5. From a command-line terminal, run the example `curl` command. With the `Anonymous Access` policy still active, do not provide the `user_key`.

```
[student@workstation ~]$ curl -i \
 https://gateways-policies-3scale-apicast-staging.apps.ocp4.example.com:443/echo
HTTP/1.1 503 Service Temporarily Unavailable
...output omitted...
Service Unavailable - Maintenance
```

Notice that the service responds with a maintenance message and a 503 status code.

4. Remove the maintenance mode policy to restore the service.

   4.1. Navigate back to **Integration** > **Policies**.

   4.2. Click the **Maintenance Mode** policy to update it. At the bottom of the form, click **Remove**.

   4.3. Click **Update Policy Chain** to apply policy changes.

   4.4. Deploy the changes to the staging environment by navigating to **Integration** > **Configuration** and clicking **Promote v.4 to Staging APIcast**.

   4.5. From a command-line terminal, run the example `curl` command. With the `Anonymous Access` policy still active, do not provide the `user_key`.

```
[student@workstation ~]$ curl \
 https://gateways-policies-3scale-apicast-staging.apps.ocp4.example.com:443/echo
{
  "method": "GET",
  "path": "/",
  ...output omitted...
}
```

Notice that the service no longer responds with the maintenance message.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish gateways-policies
```

This concludes the guided exercise.

Quiz

Managing and Customizing API Gateways

Choose the correct answers to the following questions:

1. What is an APIcast gateway? (Choose one.)

a. The component that holds the information about routing policies.
b. The UI component where you can manage your APIs.
c. The database that 3scale uses for storing data.
d. An NGINX-based gateway that is responsible for receiving and managing the incoming traffic from your APIs.

2. Which two of the following statements about the APIcast gateway are correct? (Choose two.)

a. You can only deploy an APIcast gateway by using the official operator.
b. When you deploy a self-managed gateway, it is automatically applied to all products.
c. By default, two APIcast gateways are provided: `staging` and `production`.
d. You can deploy an APIcast gateway in your own machine. Then, you can replace any of the default gateways by your self-managed gateway.

3. Consider the following manifest. Which two of the following statements are correct? (Choose two.)

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: custom-apicast
spec:
  adminPortalCredentialsRef:
    name: apicast-auth
  deploymentEnvironment: staging
  exposedHost:
    host: custom-apicast.example.com
    tls:
    - {}
```

a. The self-managed APIcast gateway replaces the `production` default gateway.
b. The self-managed APIcast authenticates against the APIManager by using the `apicast-auth` secret.
c. The self-managed APIcast is accessible at the `custom-apicast.example.com` host.
d. The manifest is not correct.

4. Which two of the following statements about policies are correct? (Choose two.)

a. A policy might conditionally alter how the request is handled.

b. You can not manage policies by using the 3scale Toolbox CLI.

c. The `3scale APIcast` policy is not required.

d. Each policy within a chain can alter or add information to the request before it is sent to the next policy.

5. Which two of the following statements about policies are correct? (Choose two.)

a. The `IP Check` policy configures allowlists and blocklists by using a list of IP addresses or blocks of addresses.

b. The `OAuth 2.0 Token Introspection` policy integrates with RH-SSO to verify realm roles as a request allowlist or blocklist.

c. The `Maintenance Mode` policy responds to all requests with a preconfigured status code and message.

d. The `JWT Claim Check` removes the `user_key` parameter requirement from all requests to the product.

6. Consider that you have exported the policies of a product by using the `3scale policies export` command. Consider also that the following YAML is the result of the export command. Which one of the following statements about the YAML is correct? (Choose one.)

```
---
- name: custom_metrics
  version: builtin
  configuration:
    rules:
    - metric: status-203
      increment: '1'
      condition:
        combine_op: and
        operations:
        - left_type: liquid
          right_type: plain
          left: "{{status}}"
          right: '203'
          op: matches
  enabled: true
- name: apicast
  version: builtin
  configuration: {}
  enabled: true
```

a. The product contains two policies: `custom-metrics` and `apicast`.
b. One of the policies is not enabled.
c. The product contains three policies: `custom-metrics`, `apicast` and `default`.
d. The `apicast` policy has higher a priority than the `custom-metrics` policy.

Solution

Managing and Customizing API Gateways

Choose the correct answers to the following questions:

1. What is an APIcast gateway? (Choose one.)

a. The component that holds the information about routing policies.

b. The UI component where you can manage your APIs.

c. The database that 3scale uses for storing data.

d. An NGINX-based gateway that is responsible for receiving and managing the incoming traffic from your APIs.

2. Which two of the following statements about the APIcast gateway are correct? (Choose two.)

a. You can only deploy an APIcast gateway by using the official operator.

b. When you deploy a self-managed gateway, it is automatically applied to all products.

c. By default, two APIcast gateways are provided: `staging` and `production`.

d. You can deploy an APIcast gateway in your own machine. Then, you can replace any of the default gateways by your self-managed gateway.

3. Consider the following manifest. Which two of the following statements are correct? (Choose two.)

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: custom-apicast
spec:
  adminPortalCredentialsRef:
    name: apicast-auth
  deploymentEnvironment: staging
  exposedHost:
    host: custom-apicast.example.com
    tls:
    - {}
```

a. The self-managed APIcast gateway replaces the `production` default gateway.

b. The self-managed APIcast authenticates against the APIManager by using the `apicast-auth` secret.

c. The self-managed APIcast is accessible at the `custom-apicast.example.com` host.

d. The manifest is not correct.

4. Which two of the following statements about policies are correct? (Choose two.)

a. A policy might conditionally alter how the request is handled.

b. You can not manage policies by using the 3scale Toolbox CLI.

c. The `3scale APIcast` policy is not required.

d. Each policy within a chain can alter or add information to the request before it is sent to the next policy.

5. Which two of the following statements about policies are correct? (Choose two.)

a. The `IP Check` policy configures allowlists and blocklists by using a list of IP addresses or blocks of addresses.

b. The `OAuth 2.0 Token Introspection` policy integrates with RH-SSO to verify realm roles as a request allowlist or blocklist.

c. The `Maintenance Mode` policy responds to all requests with a preconfigured status code and message.

d. The `JWT Claim Check` removes the `user_key` parameter requirement from all requests to the product.

6. **Consider that you have exported the policies of a product by using the `3scale policies export` command. Consider also that the following YAML is the result of the export command. Which one of the following statements about the YAML is correct? (Choose one.)**

```
---
- name: custom_metrics
  version: builtin
  configuration:
    rules:
    - metric: status-203
      increment: '1'
      condition:
        combine_op: and
        operations:
        - left_type: liquid
          right_type: plain
          left: "{{status}}"
          right: '203'
          op: matches
  enabled: true
- name: apicast
  version: builtin
  configuration: {}
  enabled: true
```

a. The product contains two policies: `custom-metrics` and `apicast`.

b. One of the policies is not enabled.

c. The product contains three policies: `custom-metrics`, `apicast` and `default`.

d. The `apicast` policy has higher a priority than the `custom-metrics` policy.

Summary

In this chapter, you learned:

- APIcast is an nginx-based gateway in Red Hat 3scale API Management that is responsible for receiving and managing the incoming traffic of your APIs.
- 3scale API Management provides two environments: `staging` and `production`. By default, 3scale API Management provides an APIcast gateway for each of these environments.
- You can replace any of the default gateways (staging and production) with your self-managed gateway.
- Policies enable higher flexibility and control in how APIcast handles traffic by providing more-granular abstractions for the NGINX proxy configuration.
- Configured policies exist within a product as part of a policy chain, which is an ordered list of policies that every request to the product is filtered through.

Chapter 4

Securing APIs with Red Hat 3scale API Management

Goal

Secure access to APIs by using various mechanisms.

Objectives

- Create and configure Admin Portal users.
- Secure APIs by using API keys and API key-pair authentication.
- Secure APIs by using Red Hat Single Sign-On.

Sections

- Creating User Accounts for the 3Scale Admin Portal (and Guided Exercise)
- Securing APIs with Red Hat Single Sign-on and OAuth (and Guided Exercise)
- Securing APIs with Red Hat 3scale API Management (Quiz)

Creating User Accounts for the 3Scale Admin Portal

Objectives

After completing this section, you should be able to create and configure Admin Portal users.

Administering the Admin Portal

With Red Hat 3scale API Management you can have different types of users that administer your APIs by using the Admin Portal. These users manage the APIs and the Admin Portal and are not the same as the developer users, the people consuming your APIs.

Users administering APIs can have different roles:

Role `admin`
: Users with this role have unrestricted access to the Admin Portal and can invite other members.

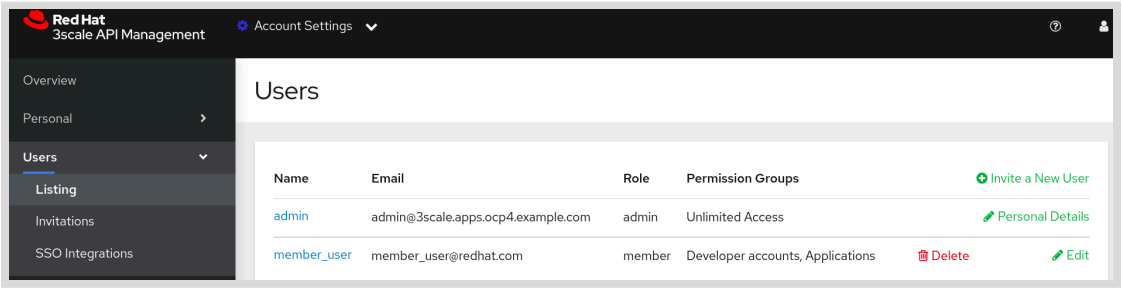
Role `member`
: Users with this role have limited access to the Admin Portal.

You can create `member` users with limited permissions to share the administration tasks for your APIs.

For example, an organization might have one or more administrators with the `admin` role. An administrator can create users for the sales department with the `member` role, and limited permissions to access only the billing section in the Admin Portal. Also, an administrator can create users with access permission to a specific product. This means that for example the API development team might access only the products related to the API versioning strategy.

If your organization has departments that are independently developing their APIs, then you can provide a stronger administrative separation by creating a tenant per department. This way, you can isolate administration tasks at the department level.

You can find the tools to manage Admin Portal users in the **Account Settings** page in the **Users** menu.



Creating Admin Portal Users

To create new users you can:

- Invite new users by sending an email invitation.

- Use Single Sign On (SSO) integration. 3scale API Management supports Red Hat Single Sign On (RHSSO) and Auth0 integrations.

In both cases, new users have the `member` role by default, and need additional permissions to do any work in the Admin Portal.

Managing User Access Control

You can configure `member` users to have fine-grained permissions in the Admin Portal.

To update user permissions you must navigate to **Users>Listing** in the **Accounts Settings** page and select the user you want to configure. There you can update the user role to be `admin` or configure the `member` permissions.

There are two sets of access permissions that you can configure:

- The first set of permissions applies at the Admin Portal level:
  - Manage content from the Developer Portal
  - Manage customer billing
  - Update settings in the following **Audience** sections: **Accounts**, **Applications**, **Billing**, **Developer Portal**, and **Messages**
- With the second set of permissions, which have finer granularity, you can grant a user access to specific products in the following Admin Portal sections:
  - Developer accounts and applications
  - Access to analytics information
  - Product and backend configuration
  - Policies and policy chains

The product level access permissions can either be granted to all current and future products, or only to a subset of the available products.

Notifications

As an Admin Portal user, you can subscribe to events related to the following 3scale API Management interactions.

Accounts
Notify developer account events.

Billing
Notify payment and billing events.

Applications
Notify application and application plan events.

Services, also known as products
Product deletion notifications.

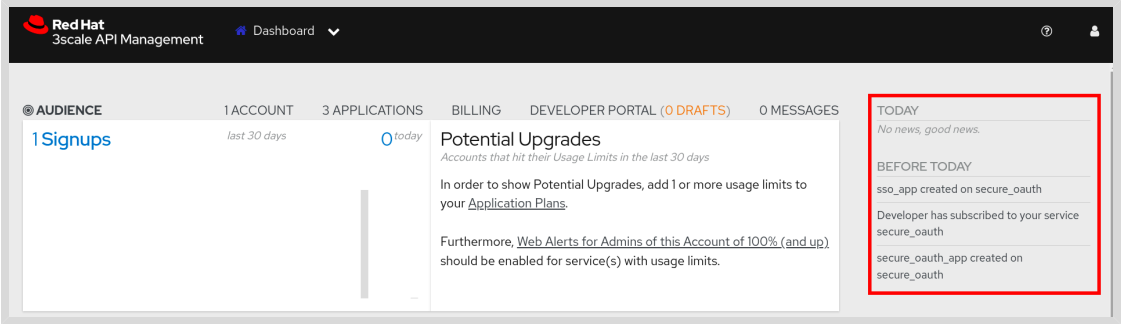
API usage alerts
Notify API usage information.

You can configure your notification preferences by navigating to **Personal>Notification Preferences** in the **Account Settings** page.

Users with the `admin` role have access to all the available types of notifications.

Users with the `member` role can only receive notifications that relate to the access permission they have.

Users receive notifications by mail. Notifications also show in the notifications section in the **Dashboard** page.



Providing Access to 3scale API Management APIs

You can interact with 3scale API Management by using the following APIs:

- Billing API
- Account Management API
- Analytics API
- Policy Registry API
- Service Management API

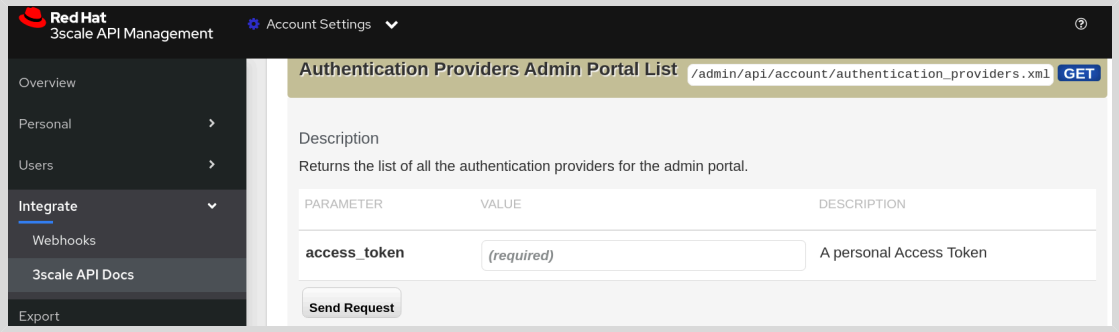
Depending on the API you want to use you need a different type of token.

- *Access Tokens* provide read-only or read-write access to the following 3scale management APIs.
  - Billing API
  - Account Management API
  - Analytics API
  - Policy Registry API
- *Service Tokens* provide per product access to the Service Management API.

Access tokens belong to the Admin Portal user, whereas service tokens are associated to the product. Therefore, different users will have their own access tokens, but they will share the service token for a product if they have access permission to that product.

As an Admin Portal user, you can create and edit the permissions for your own access tokens from the **Personal>Tokens** section in the **Account Settings** page. If your user has a `member` role then you can only create tokens for the APIs you have access to.

You can use access tokens and service tokens to explore the different 3scale API Management administration APIs in the **3scale API Docs** page.



References

For more information, refer to the *Inviting users and managing rights* chapter in the *Red Hat 3scale API Management Admin Portal Guide* at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/admin_portal_guide/index#inviting-users-managing-rights

Guided Exercise

Creating User Accounts for the 3Scale Admin Portal

In this exercise, you will configure different types of users for the Admin Portal.

Outcomes

You should be able to:

- Create a member user with limited permissions.
- Create a user with administrator privileges.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that:

- Red Hat OpenShift Container Platform (RHOCP) is ready.
- Red Hat 3scale API Management is ready.
- An email interceptor to accept account invites is ready.

```
[student@workstation ~]$ lab start secure-accounts
```

Instructions

1. Send an invitation to create a member user by using the Admin Portal.

Important

Because the `start` function restarts some 3scale API Management pods to deploy the email server, the Admin Portal might be temporary unavailable. If the Admin Portal is not available, then try again in a few minutes.

1.1. Log in to RHOCP:

```
[student@workstation ~]$ oc login \
-u=admin -p=redhat --server=https://api.ocp4.example.com:6443
...output omitted...
```

1.2. Retrieve the Admin Portal password.

```
[student@workstation ~]$ oc get secret system-seed -n 3scale \
  -o json | jq -r .data.ADMIN_PASSWORD | base64 -d; echo
...output omitted...
```

1.3. In a web browser, navigate to `https://3scale-admin.apps.ocp4.example.com/`. Log in to the Admin Portal with the following credentials:

- **Username:** `admin`
- **Password:** `ADMIN_PASSWORD` from the `system-seed` secret

1.4. Click **Account Settings** on the top pane drop-down menu. Then navigate to **Users** > **Invitations** and invite a new user by clicking **Invite a New Team Member**.

Fill the **Invite a New Team Member** form with the email for the invitation. Then submit the form.

- **Send invitation to:** `member_user@redhat.com`

1.5. Execute the `/scripts/get-emails.sh` script to receive the emails sent by 3scale API Management.

```
[student@workstation ~]$ ~/DO240-apps/scripts/get-emails.sh
---------- MESSAGE FOLLOWS ----------
...output omitted...
From: no-reply@apps.ocp4.example.com
To: member_user@redhat.com
...output omitted...
You have been invited to join Provider Name on 3scale platform.

Please sign up by following this link: https://3scale-admin.apps.ocp4.example.com/
p/signup/dbbfcd4fe317fae0e0bdc2187e70da6b

If you have any problems signing up or believe you received this email
 erroneously, please open a Support Case at https://access.redhat.com/support.

Thank you,

The 3scale API Team.
------------ END MESSAGE ------------
```

The email provides a signup link. Copy the link.

1.6. Log out of the Admin Portal by clicking the user icon and then clicking **Sign Out**.

1.7. Navigate to the sign up form by using the link copied from the invitation email. Complete the sign up form according to the following data:

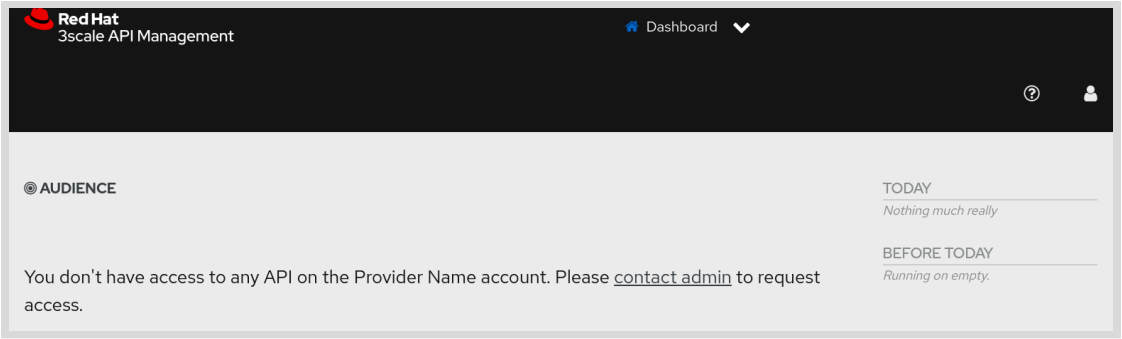
- **Username:** member_user
- **Password:** gls-password
- **Password confirmation:** gls-password

After submitting the form, the Admin Portal redirects the browser to the login form.

1.8. Log in to the Admin Portal as `member_user` by using the credentials from the previous step. If you are presented with the welcome screen then close it by pressing X.

2. Verify that `member_user` has restricted permission.

New users are created with the `member` role, which by default has no permissions. Therefore, on the welcome page, you see a message telling you that you do not have access to any API in the default tenant, the Provider Name account.



The new user neither has permissions to navigate to the **Products** or **Backends** pages.

3. Log in to the Admin Portal as the `admin` user and give `member_user` permissions to query the analytics of all API products.

3.1. Log out of the Admin Portal by clicking the user icon and then clicking **Sign Out**.

3.2. Log in to the Admin Portal as the `admin` user.

3.3. To edit `member_user` click **Account Settings** on the top pane drop-down menu. Then, click **Users** > **Listing** and select **member_user**.

3.4. Give `member_user` access to the analytics section by clicking **Access & query analytics of** in the **ADMINISTRATIVE** section. Select **All current and future existing API products** to access analytics in all the products and submit the form.

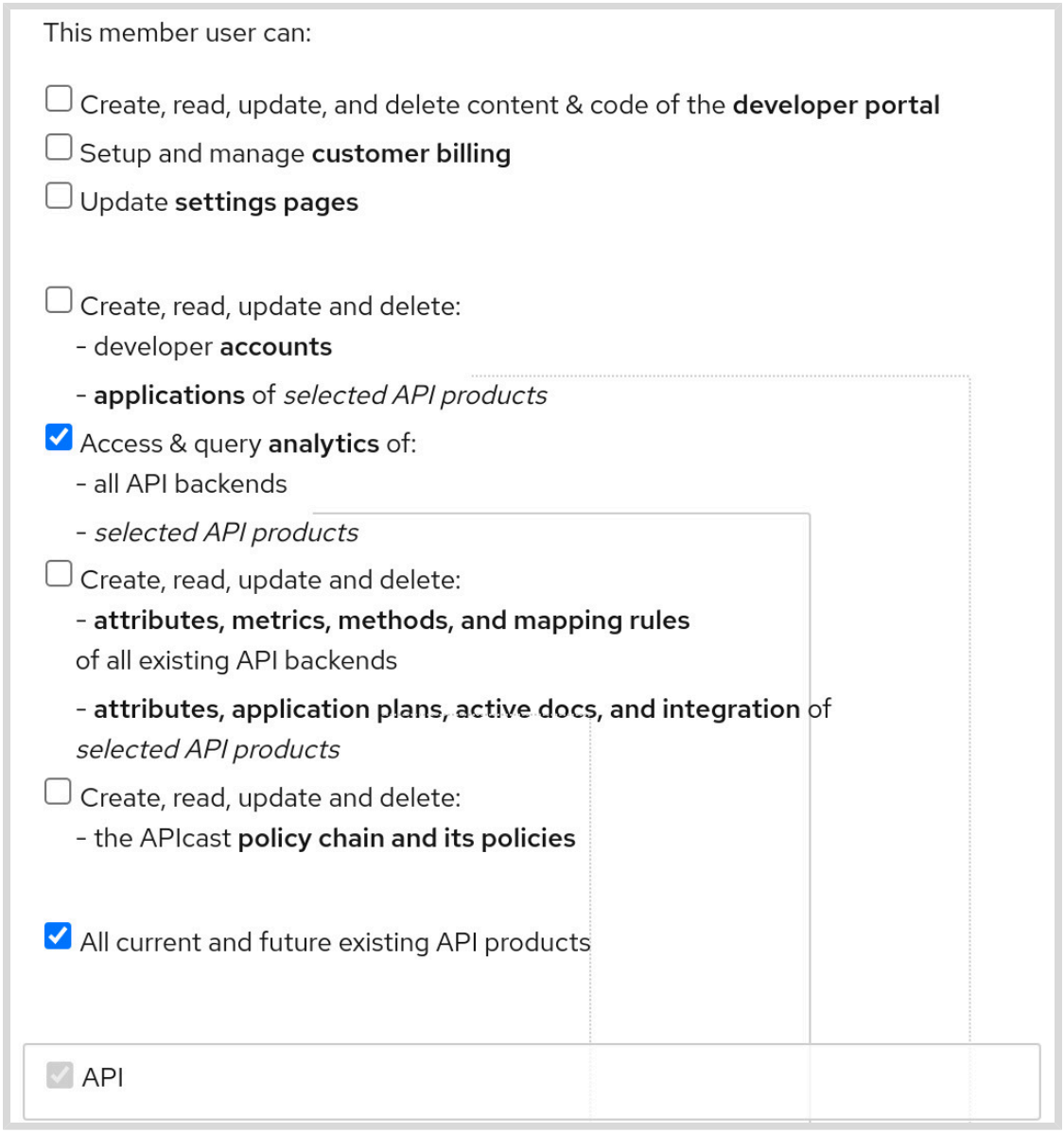


Figure 4.5: Member user permissions edit.

4. Log in to the Admin Portal as `member_user` and verify that the user can access analytics for the `API` product.

   4.1. Log in to the Admin Portal as `member_user`.

   4.2. Verify that `member_user` has access to product analytics by clicking **API** in the Products section, from the welcome page.

   In the `API` product detail page, you can verify that only the **Analytics** menu is available in the sidebar.

5. Create a new user, `admin_user`, and grant the user the `admin` role.

   5.1. Log in to the Admin Portal as the `admin` user and send the user an invitation to the `user_admin@redhat.com` address.

5.2. Run the `/scripts/get-emails.sh` script in the `DO240-apps` like you did in the previous step to copy the invitation link from the email.

5.3. Log out of the Admin Portal and use the previous invitation link to access the sign up form. Submit the form with the following values:

- **Username:** admin_user
- **Password:** gls-password
- **Password confirmation:** gls-password

5.4. Log in to the Admin Portal as `admin` and edit the `admin_user` by clicking **Users** > **Listing** and clicking **admin_user** in the **Accounts Settings** page. In the **ADMINISTRATIVE** section, select **Admin (full access)** and submit the form.

5.5. Log in to the Admin Portal as the `admin_user`, and verify that you have unrestricted access to the Admin Portal.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish secure-accounts
```

This concludes the guided exercise.

Securing APIs with API Keys and API Key-pair Authentication

Objectives

After completing this section, you should be able to secure APIs by using API keys and API key-pair authentication.

Classifying Red Hat 3scale API Management Authentication Patterns

Every call you make to an API managed by 3scale API Management requires authentication. Authenticating API calls provides a number of advantages, such as:

- Monitoring API usage for each application.
- Enforcing application plans and rate limits for each application.
- Providing a security layer for your APIs without the need to change or redeploy your application.

3scale API Management implements the following authentication patterns:

- API key
- App_ID and App_Key Pair (key-ID pair)
- Open ID Connect (OIDC) integration

For more information about OIDC, see *Securing APIs with Red Hat Single Sign-on and OAuth*.

Comparing the Authentication Patterns

When you use the API key authentication pattern, each call to the API requires you to provide a key, such as:

```
API_KEY = a6548169babc4d7847268ba63db15b65
```

An API key is a string that uniquely identifies each application that communicates with APIs managed by the 3scale Application Manager. The API key serves both as the ID and a secret token that is necessary to authenticate an API call. This is the simplest form of authentication, suitable for testing and environments with low security requirements.

The application key-ID pair decouples the API key from the application ID. Each call to the API that uses the key-ID authentication pattern requires the following information:

```
APP_ID  = 76b49d63
APP_KEY = d2fe3c9494f34a306b91c8e78f11a5f6
```

Consequently, each API call is identified by a unique ID and a unique secret token. Because the ID and secret token are decoupled from each other, you can create multiple application keys for a single application ID.

This is useful when regenerating your token. You can provision a new secret token, update your application, and then decommission the original secret token. Consequently, this means you can avoid disrupting your service due to regenerating a secret token.

Configuring Authentication Patterns

You can configure an authentication pattern implementation for each product in your 3scale API Management.

In the Admin Portal, select the product you want to configure. Then, click **Integration** > **Settings** and see the **Authentication** section.

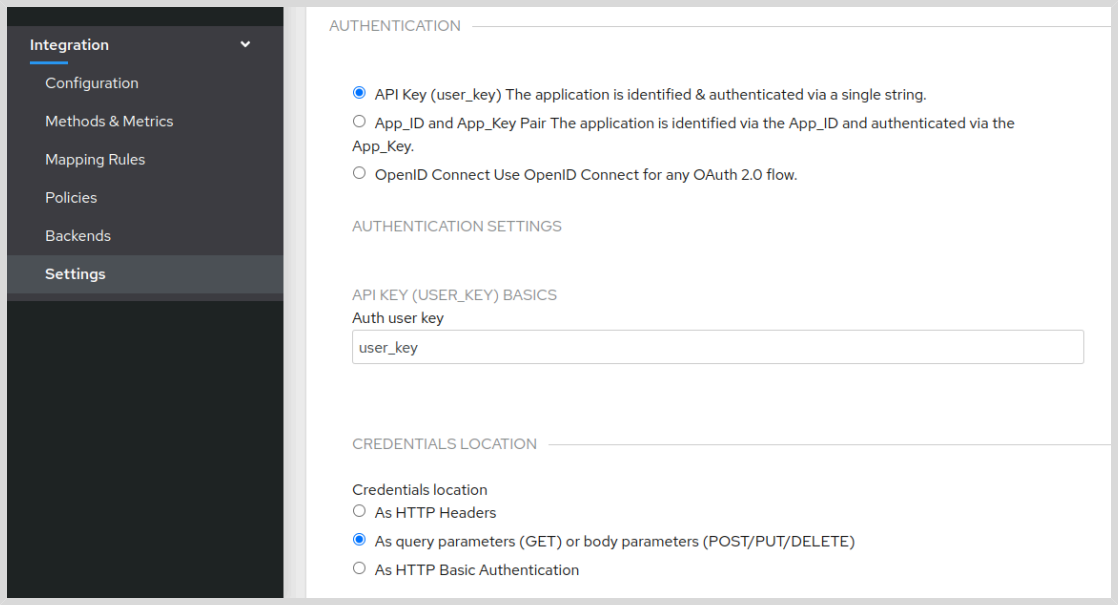


Figure 4.6: Configuring a product authentication pattern

Products use the API key authentication pattern by default. Select the **App_ID and App_Key Pair** option to require authentication by using a key-ID pair.

You can also configure:

- The name of your authentication parameters
- The way you pass authentication to your application

By default, each API call expects the `user_key` query parameter with the value of your API key, for example:

```
[user@host ~]$ curl "https://example-product-3scale-apicast-
staging.apps.ocp4.example.com:443/?user_key=a6548169babc4d7847268ba63db15b65"
```

If you enforce authentication by using the key-ID authentication pattern in the headers named `app_key` and `app_id` respectively, then the API call will look as follows:

```
[user@host ~]$ curl "https://example-product-3scale-apicast-
staging.apps.ocp4.example.com:443/" \
  --header 'app_id: 76b49d63' \
  --header 'app_key: 0a5eb0c64bd6748b6e37a7084a273e07'
```

Creating Credentials in 3scale API Management

Credentials in 3scale API Management are represented by the *application* objects. The application object maps to an application that communicates with your APIs. Each application object is mapped to a 3scale API Management account. Each application object is also associated with an application plan.

To create a new application object that is associated with your product, select your product in the Admin Portal. Then, click **Applications** > **Listing** and click **Create Application**.

After you create the application object, see the **API Credentials** section to view the API key or the key-ID pair generated for your application.

If your product uses the key-ID authentication pattern, then you can click the **Add Custom key** or **Add Random key** to associate multiple keys with your application ID. You can associate at most five keys with each application object.

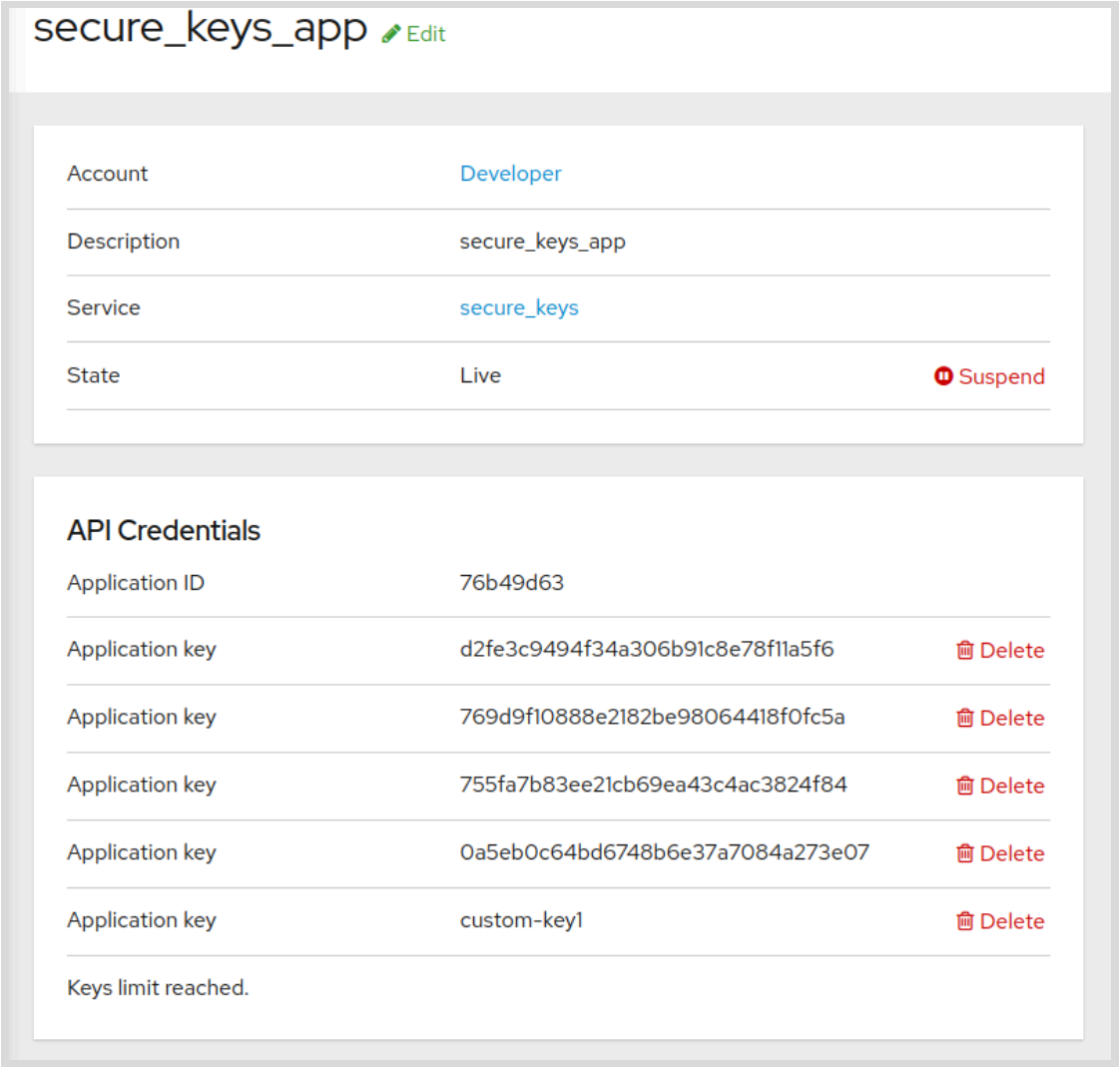


Figure 4.7: Configuring API keys for the application object

References

For more information, refer to the *API authentication* chapter in the *Administering the API Gateway* guide at
https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/administering_the_api_gateway/index#authentication-patterns

Guided Exercise

Securing APIs with API Keys and API Key-pair Authentication

In this exercise, you will secure an API by using an API key and an API key-ID pair.

Outcomes

You should be able to configure a Red Hat 3scale API Management product to require authentication by using an API key-ID pair.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command:

- Deploys the application you will use in this exercise.
- Configures a 3scale API Management product and backend.

```
[student@workstation ~]$ lab start secure-keys
```

Instructions

1. Configure the `secure_keys` product to require authentication by using the API key.

   1.1. Verify that the `secure_keys_basic` application plan in the `secure_keys` product does not have an associated application.

```
[student@workstation ~]$ 3scale application list 3scale-tenant \
  --plan=secure_keys_basic --service=secure_keys
ID NAME STATE ENABLED ACCOUNT_ID SERVICE_ID PLAN_ID
```

   1.2. Create a `secure_key_app` application that is associated with the `secure_keys_basic` application plan:

```
[student@workstation ~]$ 3scale application create 3scale-tenant \
  john secure_keys secure_keys_basic secure_keys_app
Created application id: 13
```

   1.3. List the API key that is associated with the `secure_key_app` application. Use the application ID from the preceding step.

```
[student@workstation ~]$ API_KEY=$(3scale application show 3scale-tenant 13 \
  -o json | jq -r '.user_key')
[student@workstation ~]$ echo $API_KEY
bc18aa400edba94148e37a4f632e34b2
```

1.4. Use the key to call the `books-api` service:

```
[student@workstation ~]$ curl \
  "https://secure-keys-3scale-apicast-staging.apps.ocp4.example.com:443/books?
user_key=$API_KEY" | jq
...output omitted...
```

2. Configure the `secure_keys` product to require authentication by using the API key-ID pair.

2.1. Log in to RHOCP:

```
[student@workstation ~]$ oc login \
-u=admin -p=redhat --server=https://api.ocp4.example.com:6443
...output omitted...
```

2.2. In a web browser, log in to the the 3scale API Management Administration Portal as the `admin` user.

Execute the following command to see the `ADMIN_PASSWORD`:

```
[student@workstation ~]$ oc get secret system-seed -n 3scale \
  -o json | jq -r .data.ADMIN_PASSWORD | base64 -d; echo
...output omitted...
```

2.3. In the **Products** section, click the `secure_keys` product.

2.4. Click **Integration** > **Settings**.

2.5. In the **Authentication** section, select `App_ID and App_Key Pair The application is identified via the App_ID and authenticated via the App_Key`.

2.6. Click **Update Product**. Then, click **Configuration** and click **Promote v. 2 to Staging APIcast**.

3. Use the key-ID pair to authenticate your API call.

3.1. Verify that using API key is no longer sufficient to authorize to the API:

```
[student@workstation ~]$ curl \
  "https://secure-keys-3scale-apicast-staging.apps.ocp4.example.com:443/books?
user_key=$API_KEY"; echo
Authentication parameters missing
```

3.2. Find the application ID for the application you created in the preceding steps. The following step uses ID `13` as the ID for the application:

```
[student@workstation ~]$ API_ID=$(3scale application show 3scale-tenant 13 \
  -o json | jq -r '.application_id')
[student@workstation ~]$ echo $API_ID
eb34878e
```

3.3. Authenticate your API call by using the `API_KEY` and `API_ID` variables:

```
[student@workstation ~]$ curl \
  "https://secure-keys-3scale-apicast-staging.apps.ocp4.example.com:443/books?
app_key=$API_KEY&app_id=$API_ID"| jq
...output omitted...
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish secure-keys
```

This concludes the guided exercise.

Securing APIs with Red Hat Single Sign-on and OAuth

Objectives

After completing this section, you should be able to secure APIs by using Red Hat Single Sign-On.

Describing OIDC Authentication Workflow in Red Hat 3scale API Management

You can configure 3scale API Management to require OpenID Connect (OIDC) as an authentication mechanism for your APIs. This means that 3scale API Management verifies the following:

- Each request contains a JSON Web Token (JWT).
- The JWT token is cryptographically signed by the configured OIDC provider.
- The JWT token is valid, for example not expired.
- The JWT token contains the necessary claims, such as the authorized party (`azp`) or audience (`aud`).

When you configure OIDC integration, APIcast reads the configuration of the OIDC provider by using the `https://OIDC_PROVIDER_HOST/.well-known/openid-configuration` discovery endpoint. APIcast caches information required for validating JWT tokens, such as public keys and signing algorithms.

Requests to the protected API then require a valid JWT token. The following diagram provides an example of an authentication request flow:

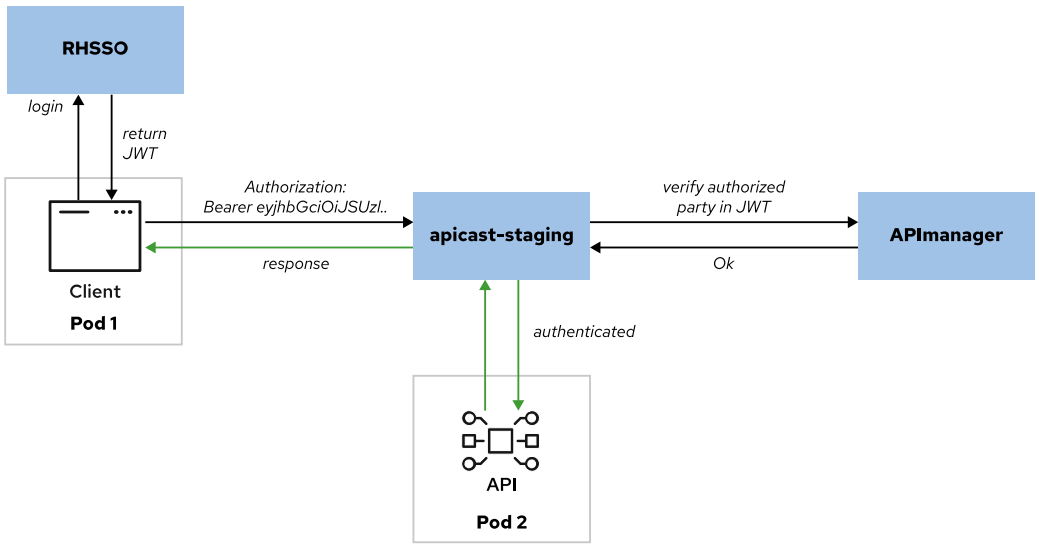


Figure 4.8: Authentication request flow

Integrating 3scale API Management with Red Hat Single Sign On (RHSSO)

When you configure OIDC integration in your product, 3scale API Management requires that the authorized party (`azp`) claim in the JWT token corresponds to the API credentials that you associate with an application in your product. However, RHSSO issues JWT tokens. Consequently, 3scale API Management must synchronize application credentials with RHSSO.

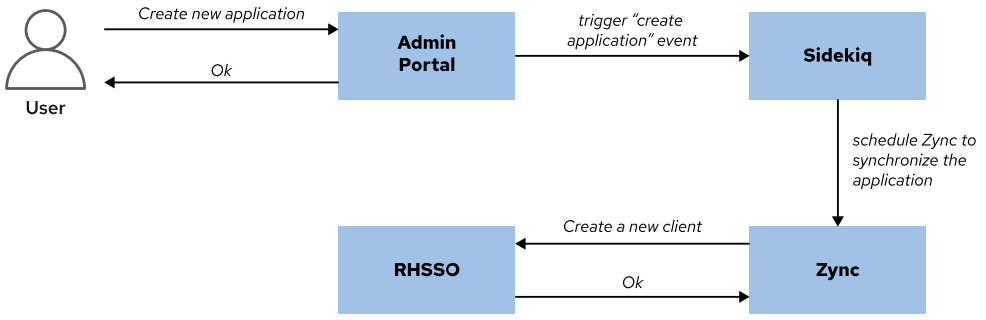


Figure 4.9: Account synchronization flow

Configuring Prerequisites for Credential Synchronization

The *Zync* component is responsible for the credential synchronization with RHSSO. When you create a new application, the *Sidekiq* component schedules Zync to synchronize the data with RHSSO. Zync asynchronously communicates with RHSSO and creates the API credentials.

For the synchronization to be successful, the Zync must be able to:

- Connect to RHSSO.
- Authenticate in RHSSO by using credentials with permissions to create clients.

For example, on Red Hat OpenShift Container Platform, if the `zync-queue` pods do not trust the RHSSO TLS certificates then the communication fails.

Additionally, Zync must authenticate with RHSSO by using a confidential `Service Account` client with the `manage-clients` role. Otherwise, Zync cannot create clients in RHSSO.

See the references section for more information about implementing the prerequisite steps.

Configuring OIDC

You can configure OIDC integration for each product in your 3scale API Management installation. When you configure OIDC integration for a particular product:

- 3scale API Management synchronizes new applications to RHSSO, provided that you configured the prerequisites.
- Clients must authenticate by using OIDC to reach the protected APIs. Authenticating with application credentials without JWT tokens is denied.

To configure OIDC for a product, navigate to the Admin Portal, select your product, and click **Integration** > **Settings**. Then, select the **OpenID Connect Use OpenID Connect for any OAuth 2.0 flow.** authentication option.

You must provide the OIDC issuer URL with the following parameters:

- **CLIENT_ID:CLIENT_SECRET**: 3scale API Management uses this client to authenticate with RHSSO.
- **PROVIDER_HOST:PROVIDER_PORT**: The location of your RHSSO.
- **REALM_NAME**: 3scale API Management creates RHSSO clients in this realm. Additionally, you can use users from this realm to authenticate your requests.

The following URL is an example of the RHSSO issuer URL:

```
https://zync-client:zync-secret@keycloak-rhsso.apps.ocp4.example.com/auth/
realms/example-realm
```

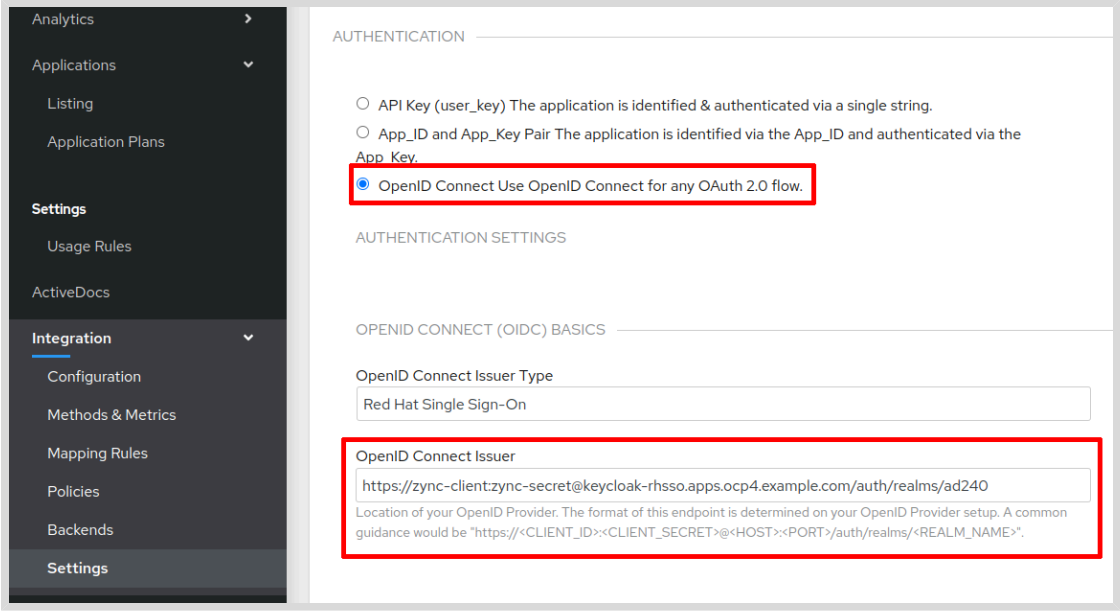


Figure 4.10: Configuring OIDC provider for a product

Configuring Advanced JWT Claim Checks

You can use the `JWT Claim Check` policy to configure advanced JWT claim checks. Consequently, you can secure resources and methods of your APIs by specifying additional JWT claim values that you require for a JWT token to be valid.

For example, consider the following policy:

```
{
  "name": "apicast.policy.jwt_claim_check",
  "configuration": {
    "error_message": "Invalid JWT check",
    "rules": [
      {
        "operations": [
          {"op": "==", "jwt_claim": "role", "jwt_claim_type": "plain", "value":
 "admin"}
        ],
        "combine_op":"and",
        "methods": ["GET"],
        "resource": "/resource",
```

```
          "resource_type": "plain"
        }
      ]
    }
  }
```

For the `GET /resource` request, the policy enforces that the provided JWT token contains the `role` claim with the value `admin`. Other resources and methods do not require the `role` claim check. All resources and methods still require a valid JWT token.

The preceding example requires RHSSO to provide the `role` claim for some JWT tokens.

You can configure the policy check for a particular product. Click **Integration** > **Policies**, then click **Add policy** and select `JWT Claim Check`.

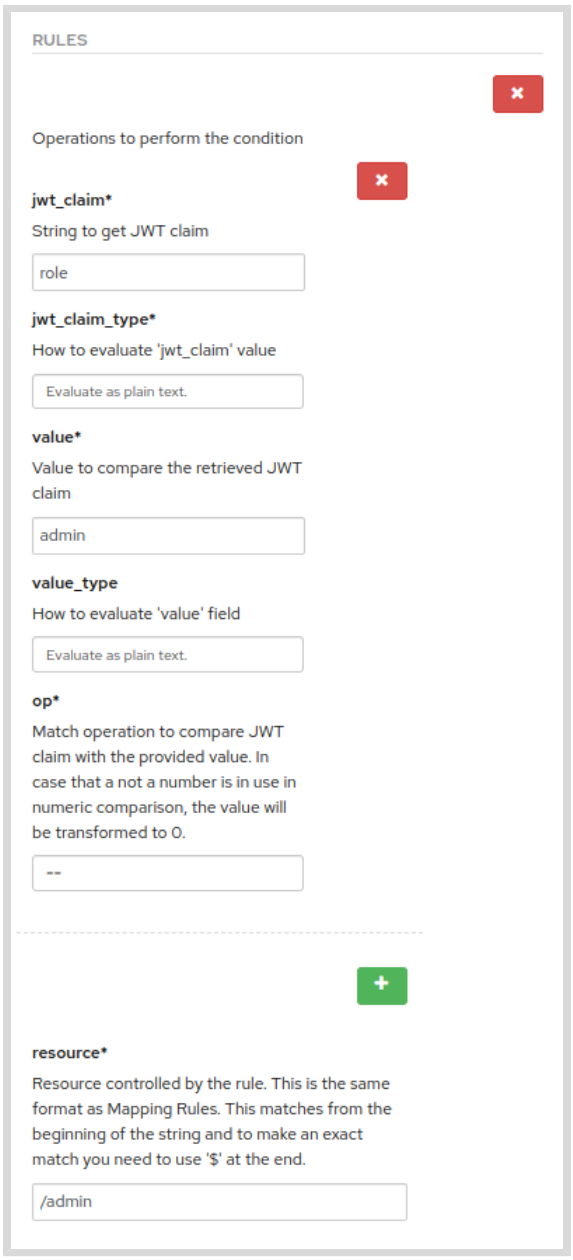


Figure 4.11: Example JWT Claim Check policy

Troubleshooting OIDC Configuration

The following list provides the most common sources of misconfiguration and errors when you integrate 3scale API Manager with RHSSO.

Zync does not synchronize clients to RHSSO

Verify that you have fulfilled the prerequisites for OIDC integration:

- The `zync` component can establish connection with RHSSO.
- The RHSSO client that you configured has sufficient permissions to create other clients.

You can check the `zync-queue` logs for more details. The following example shows a RHSSO client that does not have enough permissions to create other clients:

```
[user@host ~]$ oc logs -l threescale_component_element=zync-que
ERROR -- : "lib":"que",
"hostname":"zync-que-3-48n7z","pid":1,"thread":241980,
"event":"job_errored","job_id":162,"elapsed":0.142748975,
"error":"AbstractAdapter::InvalidResponseError:
"error"=>"insufficient_scope", "error_description"=>"Forbidden""
```

APIcast refuses your JWT token

APIcast can refuse your JWT token for multiple reasons. You can enable the debug logs for your APIcast, for example by using the `apicast.stagingSpec.logLevel: debug` operator property. Then, check the logs for more information.

The following example shows a JWT token that does not contain the audience (`aud`) claim on the stage APIcast:

```
[user@host ~]$ oc logs -l threescale_component_element=staging
2022/02/11 13:22:57 [debug] 24#24: *110 oidc.lua:191: verify(): [jwt]
 failed verification for token, reason: 'aud' claim is required.,
 requestID=567ee0a6ae81864ff959c2ece9392d43
2022/02/11 13:22:57 [debug] 24#24: *110 proxy.lua:287: rewrite(): oauth failed
 with 'aud' claim is required., requestID=567ee0a6ae81864ff959c2ece9392d43
```

A front end application fails to display successful API call results

If your front end application fails to display the result of a successful API call, then you might need to configure additional policies, such as CORS. Ensure that both RHSSO as well as your APIcast implement CORS policies.

You can implement APIcast CORS checks by adding the `CORS Request Handling` policy to your policy chain.

References

For more information about integrating 3scale API Management and RHSSO, refer to the *Integration of 3scale with Red Hat Single Sign-On* chapter in the *Administering the API Gateway* guide at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/administering_the_api_gateway/index#integration-threescale-rhsso_api-gateway-apicast

Guided Exercise

Securing APIs with Red Hat Single Sign-on and OAuth

In this exercise, you will secure an API by using Red Hat Single Sign On (RHSSO).

Outcomes

You should be able to:

- Configure 3scale API Management to synchronize with RHSSO.
- Configure 3scale API Management to require a valid JSON Web Token (JWT) to authenticate an API call.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command:

- Deploys and configures an RHSSO instance.
- Deploys the application you will use in this exercise, which consists of:
  - Front end React-based application.
  - Back end API application.
- Configures the product and backend in 3scale API Management.

```
[student@workstation ~]$ lab start secure-oauth
```

Instructions

1. Verify that the `books_api_v2` backend does not require Open ID Connect for authentication.

   1.1. Get the application ID for the current application plan:

```
[student@workstation ~]$ 3scale application list 3scale-tenant \
  --service=secure_oauth
ID NAME STATE ENABLED ACCOUNT_ID SERVICE_ID PLAN_ID
7 secure_oauth_app live true 3 3 10
```

   1.2. Get the user key for the application, for example for an application with the ID 7:

```
[student@workstation ~]$ 3scale application show 3scale-tenant 7 \
 -o json | jq -r '.user_key'
acaa2950a6124cd27a45194c023d2a3d
```

1.3. Use the key to call the `books-api` service:

```
[student@workstation ~]$ curl \
  "https://secure-oauth-3scale-apicast-staging.apps.ocp4.example.com/books?
user_key=acaa2950a6124cd27a45194c023d2a3d" \
  | jq
...output omitted...
```

The API call succeeds without a JWT token.

2. Add the RHSSO certificate chain to the 3scale API Management certificate chain.

Note

This step is necessary because the 3scale API Management Zync pod does not trust the self-signed certificates of RHSSO.

2.1. Examine the `/home/student/DO240/labs/secure-oauth/inject_rhsso_ca.sh` script.

2.2. Execute the script:

```
[student@workstation ~]$ sh \
  /home/student/DO240/labs/secure-oauth/inject_rhsso_ca.sh
...output omitted...
configmap/zync-new-ca-bundle created
deploymentconfig.apps.openshift.io/zync-que volume updated
deploymentconfig.apps.openshift.io/zync-que updated
```

2.3. Verify that the `zync` pod in the `3scale` project is in the `Running` state:

```
[student@workstation ~]$ oc -n 3scale get pods \
  -l threescale_component_element=zync-que
NAME              READY   STATUS    RESTARTS   AGE
zync-que-3-jmsst  1/1     Running   0          8s
```

3. Create the `zync-client` RHSSO client.

Use the pre-created `zync-client.json` file. The `zync-client.json` file creates a client configured as a service account in RHSSO.

3.1. Execute the following command to get the RHSSO password:

```
[student@workstation ~]$ oc -n rhsso get secret \
  credential-keycloak --template={{.data.ADMIN_PASSWORD}} \
  | base64 -d ; echo
7jpffNJgZTG8yA==
```

3.2. In a web browser, navigate to the following URL:

`https://keycloak-rhsso.apps.ocp4.example.com/auth/admin/master/console/#/realms/do240`

Log in as `admin` with the password from the previous step.

3.3. Click **Clients**, then click **Create**.

3.4. Click **Select file** and select the `/home/student/DO240/labs/secure-oauth/zync-client.json` file.

Then, click **Save**.

3.5. Click the **Service Account Roles** tab. In the **Client Roles** section, select **Realm Management**.

Click **manage clients**, then click **Add selected**.

3.6. Click the **Credentials** tab. You will require the **Secret** value later.

Keep the browser tab open.

4. Integrate the `secure_oauth` product with RHSSO.

4.1. In a new browser tab, log in to the the 3scale API Management Administration Portal as the `admin` user.

Execute the following command to see the `ADMIN_PASSWORD`:

```
[student@workstation ~]$ oc get secret system-seed -n 3scale \
  -o json | jq -r .data.ADMIN_PASSWORD | base64 -d; echo
...output omitted...
```

4.2. In the **Products** section, click the `secure_oauth` product.

4.3. Click **Integration** > **Settings**.

4.4. In the **Authentication** section, select `OpenID Connect Use OpenID Connect for any OAuth 2.0 flow.`

4.5. In the **OpenID Connect (OIDC) Basics** section, construct the **OpenID Connect Issuer** URL in the following format:

`https://CLIENT-ID:CLIENT-SECRET@RHSSO-URL`

Use the following parameters:

- **Client-id**: `zync-client`
- **Client-secret**: Copy the `Secret` value from the RHSSO web console.
- **RHSSO URL**: `keycloak-rhsso.apps.ocp4.example.com/auth/realms/do240`

For example:

`https://zync-client:f6f0..7dd4@keycloak-rhsso.apps.ocp4.example.com/auth/realms/do240`

4.6. Click **Update Product**. Then, click **Configuration** and click **Promote v. 2 to Staging APIcast**.

5. Create a new Application.

5.1. Click **Applications** > **Listing**. Then, click **Create Application**.

5.2. Select the `Developer` account, `secure_oauth` product, and the `secure_oauth_basic` application plan.

5.3. Enter the following:

- **Name**: `sso_app`
- **Description**: `sso_app`

Then, click **Create Application**.

Note the **Client ID** number of your application, for example `7308b6b9`.

6. Change the `sso_app` RHSSO client visibility.

6.1. In the RHSSO Administration Console browser tab, click **Clients**.

6.2. Click the newly created client, for example `7308b6b9`. Verify that the name of the client is `sso_app`.

6.3. Change the following properties:

- **Access Type**: `public`
- **Valid Redirect URIs**: *
- **Web Origins**: *

Then, click **Save**.

7. Modify the front end application to use the `sso_app` client.

7.1. Edit the `book-config` configuration map in the `secure-oauth` project, and change the `REACT_APP_CLIENT_ID` property to the client ID of the `sso_app`

```
[student@workstation ~]$ oc edit cm book-config -n secure-oauth
apiVersion: v1
data:
  REACT_APP_CLIENT_ID: 7308b6b9
...output omitted...
```

7.2. Delete the front end pod.

The new pod receives the updated `REACT_APP_CLIENT_ID` value.

```
[student@workstation ~]$ oc -n secure-oauth delete pod \
  -l app=books-frontend
pod "books-frontend-v2-f5658556-57vqh" deleted
```

8. Enable CORS for the `secure_oauth` product.

8.1. In the 3scale API Management Administration Portal, open the `secure_oauth` product.

8.2. Click **Integration** > **Policies**. Then, click **Add policy** and select the `CORS Request Handling` policy.

8.3. Move the `CORS Request Handling` policy to the start of the policy chain.

> **Warning**
> Failure to put the CORS policy as the first policy in the policy chain might result in the front end application not working.

8.4. Click the `CORS Request Handling` policy. Enter the following configuration:

- **allow_origin**: *

Then, click **Update Policy**, and **Update Policy Chain**.

8.5. Promote the configuration to the staging APIcast.

9. In a web browser, visit the following URL:

```
http://books-frontend-secure-oauth.apps.ocp4.example.com
```

Log in with the user `student` and password `redhat`.

You can examine the RHSSO `student` user by viewing the `/home/student/DO240/labs/secure-oauth/04_user.yml` file.

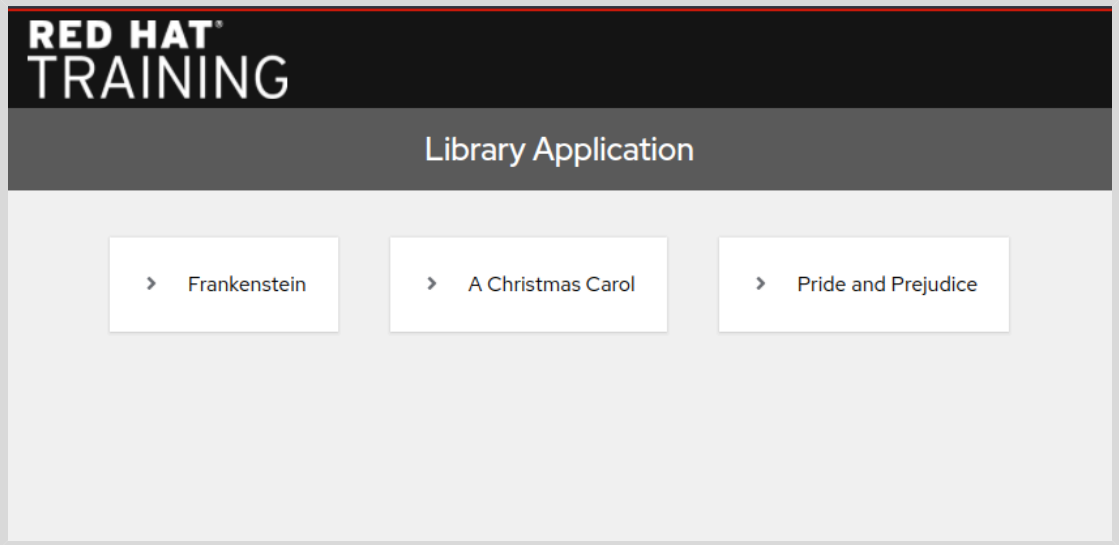


Figure 4.12: A working front end

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish secure-oauth
```

This concludes the guided exercise.

Quiz

Securing APIs with Red Hat 3scale API Management

Choose the correct answers to the following questions:

1. **Which one of the following statements about authentication in Red Hat 3scale API Management is correct?**
   a. APIs that are managed by 3scale API Management must use an adapter library to integrate with the authentication workflow.
   b. 3scale API Management does not support any cryptographically verified authentication method.
   c. You can require authentication for APIs that are managed by 3scale API Management without the need to redeploy or modify your API.
   d. 3scale API Management only supports authentication concerns when you integrate it with Red Hat Single Sign On.

2. **When you try to access a web application that communicates with an API managed by 3scale API Manager, you are presented with the following CORS error. Which one of the provided statements is the most likely cause and solution for the error?**

```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote
 resource at https://example-product-3scale-apicast-staging.apps.ocp4.example.com/
```

   a. You must modify your API to include a proper CORS header value in the API response.
   b. You can use the `CORS Request Handling` policy. This means APIcast will include a proper CORS header value in the API response.
   c. You must adapt your front end application to include a proper header in the request.
   d. 3scale API Management does not support interactions by using a browser.

3. **Consider that you configured 3scale Application Management to integrate with Red Hat Single Sign On (RHSSO). Which one of the following statements about account object synchronization is correct?**
   a. 3scale Application Management migrates all application objects to RHSSO. After the migration, 3scale Application Management deletes all local application objects and defers to RHSSO.
   b. You must recreate all application objects in RHSSO manually. You can keep the application objects in 3scale Application Management for reference.
   c. 3scale Application Management does not support integration with RHSSO.
   d. The Zync component synchronizes any application objects associated with the product that is configured to integrate with RHSSO.

4. Which one of the following statements about regenerating API keys in 3scale Application Management is correct?

a. You can generate up to five API keys when you use the key-ID authentication pattern.

b. The only way to regenerate an API key is to create a new application object.

c. You can generate up to five API keys when you use the API key authentication pattern.

d. You must regenerate your API keys periodically.

Solution

Securing APIs with Red Hat 3scale API Management

Choose the correct answers to the following questions:

1. Which one of the following statements about authentication in Red Hat 3scale API Management is correct?

a. APIs that are managed by 3scale API Management must use an adapter library to integrate with the authentication workflow.

b. 3scale API Management does not support any cryptographically verified authentication method.

c. You can require authentication for APIs that are managed by 3scale API Management without the need to redeploy or modify your API.

d. 3scale API Management only supports authentication concerns when you integrate it with Red Hat Single Sign On.

2. When you try to access a web application that communicates with an API managed by 3scale API Manager, you are presented with the following CORS error. Which one of the provided statements is the most likely cause and solution for the error?

```
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote
 resource at https://example-product-3scale-apicast-staging.apps.ocp4.example.com/
```

a. You must modify your API to include a proper CORS header value in the API response.

b. You can use the `CORS Request Handling` policy. This means APIcast will include a proper CORS header value in the API response.

c. You must adapt your front end application to include a proper header in the request.

d. 3scale API Management does not support interactions by using a browser.

3. Consider that you configured 3scale Application Management to integrate with Red Hat Single Sign On (RHSSO). Which one of the following statements about account object synchronization is correct?

a. 3scale Application Management migrates all application objects to RHSSO. After the migration, 3scale Application Management deletes all local application objects and defers to RHSSO.

b. You must recreate all application objects in RHSSO manually. You can keep the application objects in 3scale Application Management for reference.

c. 3scale Application Management does not support integration with RHSSO.

d. The Zync component synchronizes any application objects associated with the product that is configured to integrate with RHSSO.

4. Which one of the following statements about regenerating API keys in 3scale Application Management is correct?

a. You can generate up to five API keys when you use the key-ID authentication pattern.

b. The only way to regenerate an API key is to create a new application object.

c. You can generate up to five API keys when you use the API key authentication pattern.

d. You must regenerate your API keys periodically.

Summary

In this chapter, you learned:

- Red Hat 3scale Application Management Admin Portal users can have the `admin` or `member` role with configurable permission settings.
- You can require authentication for APIs managed by 3scale Application Management without the need to modify or redeploy the APIs.
- 3scale Application Management supports the following authentication patterns:
  - API key
  - App_ID and App_Key Pair
  - Integration with 3rd party identity provider, such as Red Hat Single Sign On
- 3scale Application Management can synchronize application objects to Red Hat Single Sign On.

Chapter 5

Creating a Developer Portal

Goal

Configure Developer Portals for external consumption and documentation of your APIs.

Objectives

- Create and configure a developer portal for API consumption.
- Configure a developer portal with custom look and feel.
- Import OpenAPI standard API definitions into Red Hat 3scale API Management.

Sections

- Creating an API Developer Portal (and Guided Exercise)
- Customizing an API Developer Portal (and Guided Exercise)
- Importing OpenAPI Definitions (and Guided Exercise)
- Creating a Developer Portal (Quiz)

Creating an API Developer Portal

Objectives

After completing this section, you should be able to create and configure a developer portal for API consumption.

Describing the Developer Portal

The Developer Portal aims to engage API users, and turn developers into customers. Developer Portal is a configurable portal where API developers can create application credentials, subscribe to application plans, read about billing plans, and read the documentation for your APIs. This means that your users can self-provision access to APIs managed by 3scale API Management.

Listing Developer Portal Content

You can create new content and change the look and feel of the Developer Portal by using the 3scale API Manager Admin Portal. This means that you can tailor the Developer Portal to use your branding and content.

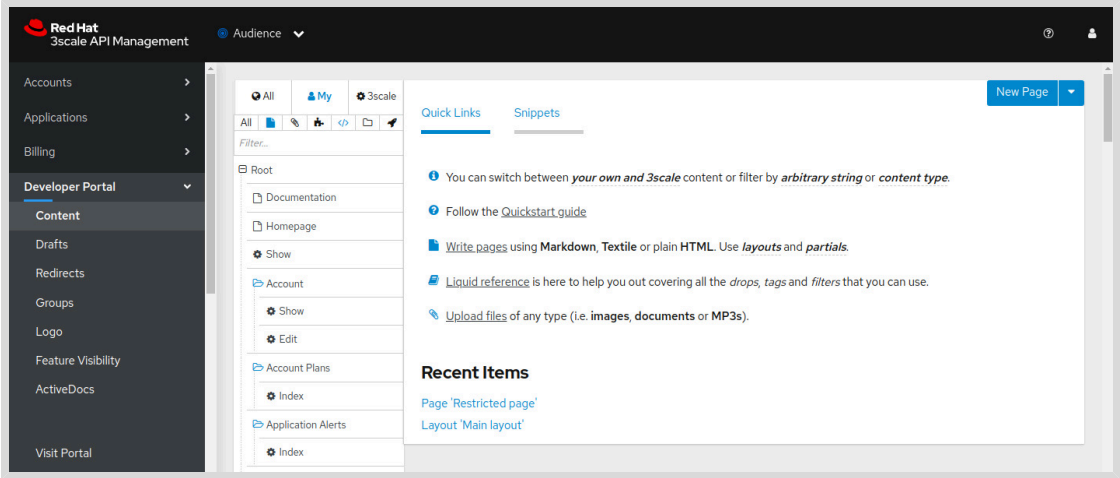


Figure 5.1: Managing content in the Developer Portal

To access the Developer Portal customization options, select **Audience** from the drop-down menu, and then click **Developer Portal**. The following list describes sections in the Developer Portal menu:

Content
: Contains Developer Portal content that you can create and manage, such as HTML, CSS, and JavaScript files.

Drafts
: Lists the content with the `Draft` status.

Redirects
: Create and manage redirects from one portal URL to another, for example when you deprecate a page on your Developer Portal.

Groups
Create and manage user groups. You can use groups to limit access to content sections.

Logo
Upload a Developer Portal logo.

Feature Visibility
Toggle advanced features of your Developer Portal, such as webhooks, or identity and access management (IAM) tools.

ActiveDocs
Contains the OpenAPI documentation for your APIs.

In the **Content** section, you can create and edit the following elements:

Page
A page typically contains liquid-templated HTML content to be shown when the users access a given URL. You can also create pages in Markdown or Textile.

Section
A section is a logical grouping of pages that you can use to create hierarchical page structures. For example, you can group all pages accessible by users that belong to a particular group under a common URL subpath.

Layout
Layouts define a common structure that is often used by multiple pages. For example, you might reference common CSS and JavaScript files in one layout and use the layout for all your pages.

Partial
Partials are reusable elements that you can embed in pages. Layouts and partials are distinct elements. Typically, layouts define the structure of pages whereas partials are contained in Pages.

For example, a layout defines a common header and footer for your pages, while a partial defines a disclaimer which you add to a subset of your pages.

File
Files are content uploaded to the Developer Portal that you want to reference or display, such as an image later referenced in a page.

Portlets
Portlets gather and show information from external sources. You can use the following portlets:

External RSS Feed
Fetches the RSS feed from an external source. For example, you might decouple announcing changes to the API by moving it to an RSS feed. This means that you can add this feed to a page such that there is no need to update the page every time an announcement is made.

Table of Contents
Generates a list of links for the pages in a given section.

Latest Forum Posts
Displays the latest posts in the Community Forum.

Creating Developer Portal Content

You can create new content by selecting the **Content** option in the **Developer Portal** section.

To add content to the portal, click **New Page**. To create another kind of content, use the drop-down menu of that button to select the desired content type.

For example in the **New Page** editor, provide the following data:

Title
Name of the page.

Section
Section to file the page under.

Path
URL path relative to the Developer Portal that shows the page.

Note
The page called **Homepage** has a special path (`/`) that does not include the file name. This is because homepage is implemented as an `index.html` file, which is automatically served if no file name is provided.

Layout
Layout to define the general structure of the page.

After you provide the details, use the content box below to write the content of the page. 3scale API Management supports content written in HTML, Markdown and Textile.

Use the **Preview** button to validate the results and view how this new content is presented to the client. You can view the content in a preview page with a preview toolbar. This toolbar shows information about the page and the templates used to render the page. You can also switch between publishing modes.

Restricting Access to Pages

You can limit access to some parts of the Developer Portal by creating restricted sections. These sections can contain content that is specific for logical groups of users. Users that are associated with that logical group can access the restricted section.

For example, consider that there is a group of developers called *partners*. You can create a new section with the following details:

Section data

Field	Value
Title	`Partner Pages`
Public	Unselected
Parent	`.Root`

Then, you create a page in the private section, for example with the following parameters:

Page data

Field	Value
Title	`How it works`
Section	`Partner Pages`
Path	`/howitworks`
Layout	`Main`
Content	`<h1>How restricted sections work</h1>`

After you create a restricted section, create a group and give it access to that section. Navigate to **Developer Portal** > **Groups** and click **Create Group**. Set the name of the group to `Partner's Group`, mark the `Partner pages` section as allowed, and click **Create Group**.

Finally, add users to this group to grant access to pages that belong to the Partner pages section. Navigate to **Dashboard** > **Audience**, then click **Accounts** > **Listing**, and select a user. In the account page, click **Group membership** and select the `Partner's Group`.

This means that the user has access to the *`PORTAL-URL`*`/howitworks` path.

References

For more information, refer to the *Creating the Developer Portal* guide at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/creating_the_developer_portal/index

Guided Exercise

Creating an API Developer Portal

In this exercise, you will publish and customize the content in your Developer Portal.

Outcomes

You should be able to:

- Publish your Developer Portal.
- Create a new user in the Developer Portal.
- Create restricted content in the Developer Portal.
- Grant select users access to the restricted content.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start portal-creating
```

Instructions

1. Navigate to the Developer Portal.

1.1. Log in to RHOCP:

```
[student@workstation ~]$ oc login \
-u=admin -p=redhat --server=https://api.ocp4.example.com:6443
...output omitted...
```

1.2. In a new browser tab, log in to the `3scale` tenant Admin Portal as the `admin` user. Execute the following command to see the `ADMIN_PASSWORD`:

```
[student@workstation ~]$ oc get secret system-seed -n 3scale \
  -o json | jq -r .data.ADMIN_PASSWORD | base64 -d; echo
...output omitted...
```

1.3. Click **Dashboard** > **Audience**, select **Developer Portal**, and click **Content**.

1.4. Click **Open your Portal to the world** and select **Ok**. Then, in the left side menu, click **Visit Portal**.

2. Create a new Developer Portal user.

2.1. Click **Signup to plan Basic**.

2.2. Provide the following details:

- **Organization/group name**: `GLS`
- **Username**: `portal-user`
- **Email**: `gls@redhat.com`
- **Password**: `redhat`
- **Password confirmation**: `redhat`

Then, click **Sign up**.

2.3. In the Admin Portal, click **Accounts** > **Listing**. Then, click **Activate** to enable the `portal-user` account.

3. Explore the created user.

3.1. Click the `GLS` link. Note that the registration created an application for the user.

3.2. Click the `GLS's App` link. Note that the user uses the `Basic` application plan, and has access to the `API` product.

4. Create a restricted page on the Developer Portal.

4.1. Click **Developer Portal** > **Content**. Then, click the arrow next to **New Page** and select **New Section**.

Enter the following details:

- **Title**: `Restricted section`
- **Public**: unchecked

Then, click **Create Section**.

4.2. Then, click the arrow next to **New Section** and select **New Page**.

Enter the following details:

- **Title**: `Restricted page`
- **Section**: `Restricted section`
- **Path**: `/restricted`

Enter `<h1>Welcome to the Restricted Section</h1>` as the HTML content of the page. Then, click **Create Page** and **Publish**.

5. Verify that the `/restricted` page is not visible.

5.1. Open a new browser window in an incognito mode, for example by pressing `ctrl+shift+n` in Firefox.

Note

If you visit the restricted section when logged in to the Admin Portal then you will see the content. You must either log out of the Admin Portal, open a different browser, or open a new browser window in the incognito mode.

5.2. Navigate to `https://3scale.apps.ocp4.example.com/restricted`. You are presented with a `Not found` error screen.

6. Allow the `portal-user` user to view the restricted content.

6.1. In the Admin Portal, click **Developer Portal** > **Groups**.

6.2. Click **Create Group** and provide the following details:

- **Name**: `privileged`

Then, click **Create Group**.

6.3. Click **Privileged**. Then, select the `Restricted section` and click **Save**.

Note

There is currently a bug that prevents selected sections from being properly assigned to a group. You must open the group, select the section, and then update the group.

6.4. Click **Accounts** > **Listing**. Then, click **GLS**.

6.5. Click **0 Group Memberships**. Then, select the `privileged` group and click **Save**.

7. Verify that the `portal-user` can view the restricted content.

7.1. Navigate to `https://3scale.apps.ocp4.example.com/` and log in as the `portal-user` with the `redhat` password.

7.2. Navigate to `https://3scale.apps.ocp4.example.com/restricted`. You are presented with the content.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish portal-creating
```

This concludes the guided exercise.

Customizing an API Developer Portal

Objectives

After completing this section, you should be able to configure a developer portal with custom look and feel.

Customizing the Developer Portal

The default Developer Portal includes features that most of your users need to manage access to your APIs. However, you might need to add additional features or customization. For example, you can change the look and feel of the portal to better match company brand standards by updating colors or adding a company name and logo.

On top of minor additions and tweaks, you can modify or replace many parts of the default Developer Portal.

Accessing and Modifying the Templates

Within the Admin Portal, access the default Developer Portal templates by navigating to **Audience** and selecting **Developer Portal** > **Content**.

This view is divided into two main panes. The left pane shows the structure of the various components and templates, and the right pane is used to edit a selected component.

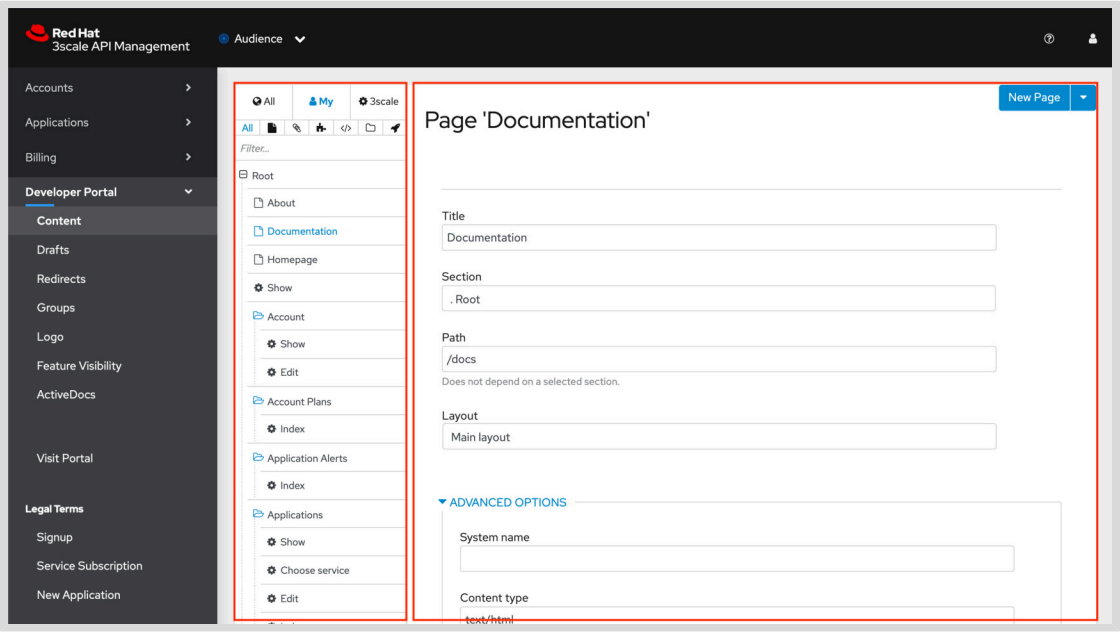


Figure 5.2: The left and right panes of the Developer Portal editing page.

The left pane is filterable by ownership and component type. Click **My** to only show components you own and **3scale** to show components owned by the system. Components owned by 3scale can be modified, but they cannot be deleted.

To filter the list by component type, select the icons representing the types you wish to see. For example, click the icon of a jigsaw puzzle piece to only show partials.

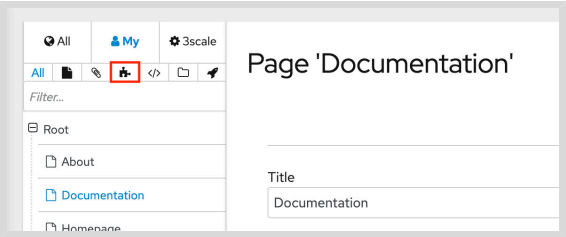


Figure 5.3: Use the filters to limit the list of components by type.

To edit the components, select one and edit it in the right pane. Test your changes by clicking **Preview**. Once you are happy with your changes, click **Publish** to deploy the changes to the Developer Portal.

Customizing Templates

The Developer Portal uses a templating engine called Liquid. By using a templating engine, each page can reuse elements and features without the need for duplicate code.

When a user visits a page, the server stitches together the parts before sending the response. This makes the portal feel like a cohesive web application rather than a set of disparate pages.

The default Liquid templates in the Developer Portal use standard HTML, JavaScript, and CSS. Liquid syntax enables you to define and reference variables within the pages, as well as create conditional and repeating structures.

Basic Liquid Syntax

The Liquid syntax uses double curly braces (`{{` and `}}`) to indicate parts to be parsed as Liquid syntax and replaced with the results of the expression.

For example, consider the following HTML and Liquid snippet:

```
<strong>{{ name }}</strong>
```

Assuming the variable `name` is set to `Bob`, the resulting HTML would be the following:

```
<strong>Bob</strong>
```

The curly brace and percent symbol syntax (`{%` and `%}`) indicates parts to be parsed as Liquid syntax, but does not get replaced with anything. These parts are used to define parts of the template that control the structure, such as loops.

For example, consider the following HTML and Liquid snippet:

```
<ul>
  {% for user in users %}
    <li>{{ user.name }}</li>
  {% endfor %}
</ul>
```

Assuming `users` is a list of objects with a `name` property, the resulting HTML would be an unordered list of the names of users.

Adding Branding to the Developer Portal

Within the Liquid templates, 3scale API Management provides a number of prepopulated variables. One of these variables is the `provider` variable, which includes several fields about your company. A reference of included variables is available by navigating to **Developer Portal** > **Docs** > **Liquid Reference** from the left-side navigation pane.

For example, the `provider.logo_url` and `provider.name` are set to the custom logo URL and company name, respectively.

The following HTML and Liquid snippet produces an anchor tag that has the company name and logo.

```
<a href="/">
  <img src="{{ provider.logo_url }}" />
  {{  provider.name }}
</a>
```

References

For more information, refer to the *Red Hat 3scale API Management Creating the Developer Portal* at
https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/creating_the_developer_portal

For more information on using Liquid, refer to the *Red Hat 3scale API Management Liquid Reference* at
https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html/liquid_reference/liquid-reference

Guided Exercise

Customizing an API Developer Portal

In this exercise, you will customize the default developer portal that comes with 3scale API Management.

Outcomes

You should be able to:

- Change the company name
- Upload a custom logo and use it within the developer portal
- Customize the developer portal colors and theming
- Create a custom page within the developer portal
- Add a navigation item to the developer portal

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command ensures that:

- Red Hat OpenShift Container Platform (RHOCP) is accessible
- 3scale API Management is installed
- Required files exist on the workstation machine

```
[student@workstation ~]$ lab start portal-customizing
```

Instructions

1. Update the company name for use within the developer portal.

1.1. Within a web browser, navigate to the 3scale API Management Admin Portal. The URL to the Admin Portal is `https://3scale-admin.apps.ocp4.example.com`.

1.2. Using the top pane navigation drop-down, click **Account Settings**.

1.3. Next to `Account Details`, click **Edit**.

1.4. Change the organization name to `Bob's Boxes` and click **Update Account**.

2. Review the default look and feel of the developer portal.

2.1. Using the top pane navigation drop-down, click **Audience**.

2.2. Preview the current version of the developer portal by clicking **Developer Portal** > **Visit Portal**.

Close the CMS toolbar by clicking the **X** (close) button in the top right corner.

2.3. Review the default style and theme of the portal. Notice the blue theme and that the company name is `Bob's Boxes`.

Leave this tab open because you will visit the portal several times to view your changes.

3. Add a custom company logo to the navigation of the portal.

3.1. Within the Admin Portal, navigate to **Developer Portal** > **Logo** and upload the file `~/DO240/labs/portal-customizing/icon.png`.

3.2. Navigate to **Developer Portal** > **Content**.

This area displays the structure and content of your developer portal. The right side of the page is further divided into two sections: the file hierarchy and file editing.

3.3. Within the file hierarchy, under the `Layouts` heading, select the file called `Main layout`. This file provides the structure for many of the pages within the default developer portal.

3.4. Modify the contents of the file by adding an `img` tag referencing the `provider.logo_url` variable. After your updates, the file should match the following contents:

```
...output omitted...
      <nav class="navbar navbar-fixed-top navbar-inverse" role="navigation">
        <div class="container tabbed">
          <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#navbar-1">
              <span class="sr-only">Toggle navigation</span>
              <span class="icon-bar"></span>
              <span class="icon-bar"></span>
              <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="/">
              <img src="{{ provider.logo_url }}" height="32px" />
              {{  provider.name }}
            </a>
          </div>
          {% include 'submenu'%}
        </div>
      </nav>
...output omitted...
```

3.5. Deploy your changes to the developer portal by clicking **Publish** at the bottom of the file editing panel.

3.6. Select the developer portal tab and refresh the page to view the logo in the navigation bar.

4. Customize the style of the developer portal by modifying the included CSS file.

4.1. Within the Admin Portal tab, select the `default.css` file from the file hierarchy.

4.2. Modify the background color of the `body` selector to orange. The file should match the following excerpt:

```
...output omitted...
body {
  background-color: darkorange; /* fallback */
  background-color: rgb(255, 140, 0);
  ...output omitted...
}
...output omitted...
```

4.3. Deploy your changes to the developer portal by clicking **Publish** at the bottom of the file editing panel.

Select the developer portal tab and refresh the page to view the color theme change to orange.

5. Add a new page to the developer portal that shows company information.

5.1. Within the Admin Portal, open the file called `Documentation`. Click **New Page** at the top right.

5.2. Fill out the form according to the following table:

Field	Value
Title	About
Path	/about

Within the `Advanced Options` section, make sure `Liquid enabled` is checked.

Leave all other fields as their default values and click **Create Page**.

5.3. Update the contents for the about page by pasting the following HTML snippet into the editor:

```
<h1>About</h1>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam accumsan est id
 neque interdum posuere. Ut eleifend et sapien nec pharetra. Vivamus condimentum
 magna vitae justo rutrum, non vestibulum odio ultricies. Morbi eget sem sit amet
 erat dapibus molestie. Nullam nec felis et massa tincidunt interdum. Integer
 imperdiet nisi ac lacus feugiat pretium. Suspendisse accumsan consequat augue, a
 feugiat metus commodo sed.</p>
```

5.4. Deploy your changes to the developer portal by clicking **Publish** at the bottom of the file editing panel.

Select the developer portal tab and navigate to `https://3scale.apps.ocp4.example.com/about` to view the new about page.

6. Add a button to the navigation for the about page.

6.1. Within the Admin Portal tab, open the file called `submenu` under the `Partials` heading.

6.2. Update the contents of the partial to include an anchor tag for the about page. The file should match the following excerpt:

```
...output omitted...
  {% if current_user %}
    ...output omitted...
        <li><a class="{% if urls.docs.active? %}active{% endif %}" href="/
docs">Documentation</a></li>

        <li class="{% if request.path == "/about" %}active{% endif %}">
          <a href="/about">About</a>
        </li>

      </ul>
  ...output omitted...
  {% else %}
    <ul class="nav navbar-nav">
      <li><a href="/docs">Documentation</a></li>
      <li><a href="/#plans">Plans</a></li>
      <li class="{% if request.path == "/about" %}active{% endif %}">
          <a href="/about">About</a>
      </li>
    </ul>
    ...output omitted...
  {% endif %}
</div>
```

6.3. Deploy your changes to the developer portal by clicking **Publish** at the bottom of the file editing panel.

Select the developer portal tab and refresh the page. Use the navigation at the top of the page to navigate between the different pages, including the new about page.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish portal-customizing
```

This concludes the guided exercise.

Importing OpenAPI Definitions

Objectives

After completing this section, you should be able to import OpenAPI standard API definitions into Red Hat 3scale API Management.

Introducing the OpenAPI Specification.

The OpenAPI Specification (OAS) is a way of describing REST APIs. It is an open source specification based on the Swagger Specification.

It offers the following benefits:

- It provides a standard for documenting your APIs.
- It allows API consumers to interact with your API from the documentation, for example for testing purposes.
- It does not require any binding to a programming language or a programming framework.
- It offers a way to generate clients and servers that comply with the OAS you define.

The OAS definition supports both YAML and JSON formats.

This specification is used by Red Hat 3scale API Management to provide interactive documentation for the Developer Portal. In 3scale API Management the OAS integration is called ActiveDocs.

Creating ActiveDocs with OAS Definitions

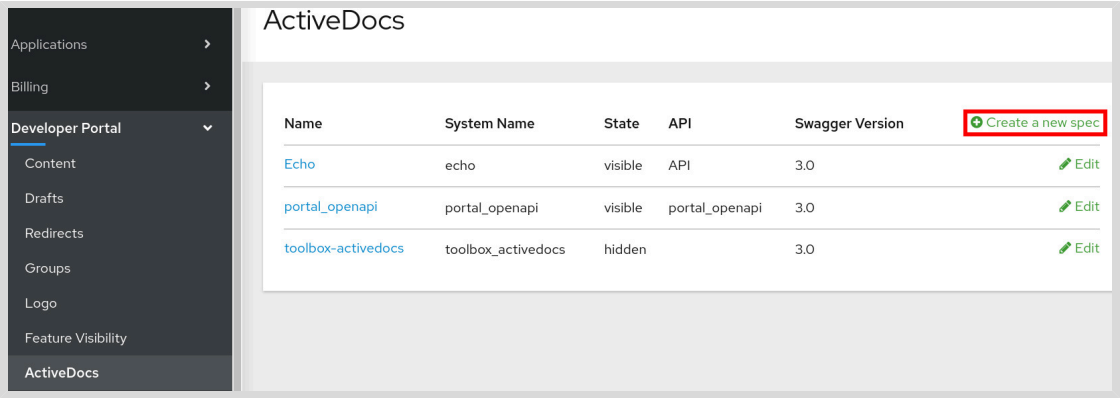
You can manage your ActiveDocs definitions either by using the Admin Portal or the 3scale Toolbox CLI.

Also, by using the 3scale Toolbox CLI and the `import` command you can generate all the 3scale API Management objects needed to create an application based on the OAS definition.

Creating ActiveDocs from the Admin Portal

You can create an ActiveDocs object either from the **Audience** page or from a product page. If you create the ActiveDocs from a product page then the ActiveDocs are assigned to that product.

To select any product for the OAS definition, create the ActiveDocs from the **Audience** page. For this you must navigate to **Developer Portal>ActiveDocs**, click **Create a new spec**, and complete the ActiveDocs form.



Managing ActiveDocs by Using the 3scale Toolbox CLI

The 3scale Toolbox CLI has the `activedocs` command, which offers several subcommands to manage ActiveDocs objects.

```
[user@host ~]$ 3scale activedocs -h
NAME
    activedocs - activedocs super command

USAGE
    3scale activedocs <sub-command> [options]

DESCRIPTION
    Manage your ActiveDocs

SUBCOMMANDS
    apply       Update activedocs
    create      Create an ActiveDocs
    delete      Delete an ActiveDocs
    list        List ActiveDocs
```

All these subcommands affect ActiveDocs objects exclusively. For example, you can use the delete subcommand as a way to delete ActiveDocs objects.

```
[user@host ~]$ 3scale activedocs delete \
  REMOTE \ (1)
  ACTIVE_DOCS (2)
```

(1) The target tenant.

(2) ID or system name of the ActiveDocs to delete.

Importing ActiveDocs from the 3scale Toolbox CLI

The 3scale Toolbox CLI offers a different way of creating ActiveDocs by importing the OAS definition. For this, you must use the `import` command and the `openapi` subcommand. You can get the OAS definition either from a file or from a URL.

This operation does more than creating the ActiveDocs object. It generates many 3scale API Management objects based on the OAS definition file. It creates:

- A product with a name taken from the `title` attribute in the OAS file.
- A backend with a name taken from the `title` attribute and a backend URL taken from the servers section in the OAS file.
- Mapping rules and methods based on the `paths` section from the OAS file.

This is an example command with some of the available options:

```
[user@host ~]$ 3scale import openapi \
  --target_system_name=SYSTEM_NAME \ (1)
  --destination=REMOTE \ (2)
  --default-credentials-userkey=ANONYMOUS_KEY (3)
  OAS_FILE (4)
```

(1) This is the system name that is used as the system name for the product, the backend and the ActiveDocs objects.

(2) The target tenant.

(3) This option adds an `Anonymous Access` policy. The value provided here is used for unauthenticated endpoints.

(4) The OAS file in JSON or YAML. This option also accepts a URL that returns the OAS definition.

After executing this command you must create an application plan and an application to make your API public.

To use the ActiveDocs interactive documentation you must add a `CORS Request Handling` policy and set the `allow_origin` to the Developer Portal domain. The reason for this is that the ActiveDocs request is sent from the Developer Portal domain, which is different from your API domain. When the domain from where the request is originated differs from the domain serving the request, the browser blocks the call unless you configure APIcast to allow it.

References

For more information, refer to the *How to write an OpenAPI document for use as a 3scale OpenAPI spec* chapter in the *Red Hat 3scale API Management Providing APIs in the Developer Portal* at
https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html/providing_apis_in_the_developer_portal/how-to-write-an-openapi-document-for-use-as-a-threescale-openapi-spec_creating-a-new-service-based-on-oas

OpenAPI Specification
https://github.com/OAI/OpenAPI-Specification

Guided Exercise

Importing OpenAPI Definitions

In this exercise, you will import the OpenApi definition from an existing API and explore the ActiveDocs generated by Red Hat 3scale API Management.

Outcomes

You should be able to:

- Import the OpenApi definition of an API into 3scale API Management.
- Use ActiveDocs in the Admin Portal to explore the API.
- Update the Documentation template in Admin Portal to show all the ActiveDocs in the Developer Portal.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command deploys a pet shelter API to the Red Hat OpenShift Container Platform (RHOCP) cluster.

```
[student@workstation ~]$ lab start portal-openapi
```

Instructions

1. Add the backend URL to the OpenApi definition. This is the pet shelter API internal URL accessible from within the RHOCP cluster.

   1.1. Examine the `~/DO240/labs/portal-openapi/openapi-definition.yaml` file, and search for the *CHANGE_ME* URL value in the `servers` section.

   1.2. Replace the *CHANGE_ME* URL value with the cluster API URL, `http://petshelter-api.portal-openapi.svc.cluster.local`

2. Import the `openapi-definition.yaml` OpenApi definition.

   2.1. Execute the `3scale_wrapper.sh` script to import the OpenApi definition. Name the product resulting from the import with `portal_openapi` by providing the `--target_system_name=portal_openapi` argument.

   The `3scale_wrapper.sh` script mounts the last parameter as a Podman volume file to run 3scale Toolbox CLI with Podman.

```
[student@workstation ~]$ \
  ~/DO240/labs/portal-openapi/3scale_wrapper.sh import openapi \
  --target_system_name=portal_openapi \
  --destination=3scale-tenant \
  DO240/labs/portal-openapi/openapi-definition.yaml
{
  "code": "E_3SCALE",
  "message": "User key must be provided by --default-credentials-userkey optional
 param",
  "class": "ThreeScaleToolbox::Error"
}
Created service id: 3, name: Pet Shelter
Service proxy updated
destroying all mapping rules
Created PUT /pet$ endpoint
Created POST /pet$ endpoint
Created GET /pet/findByStatus$ endpoint
Created GET /pet/{petId}$ endpoint
Created DELETE /pet/{petId}$ endpoint
```

You can ignore the error message.

Note

You can add the `--default-credentials-userkey=anonymous_api_key` option to allow for unauthenticated endpoints. This creates an `Anonymous Access` policy with the *anonymous_api_key* value.

2.2. Use the 3scale Toolbox CLI `proxy-config` command to get the `portal_openapi` product's proxy configuration, and verify that the previous step creates a backend with the correct URL.

```
[student@workstation ~]$ 3scale proxy-config show \
  3scale-tenant portal_openapi sandbox -o json \
  | jq -r '.content.proxy.api_backend'
 http://petshelter-api.portal-openapi.svc.cluster.local:80
```

3. Create an application plan and an application.

3.1. Create an application plan called `basic_plan` by using the 3scale Toolbox CLI from the command line.

```
[student@workstation ~]$ 3scale application-plan create \
  3scale-tenant portal_openapi basic_plan
Created application plan id: 13. Default: false; Disabled: false
```

3.2. Create an application on the default `john` account called `shelter-app`, and assign it the `basic_plan` application plan.

```
[student@workstation ~]$ 3scale application create \
  3scale-tenant john portal_openapi basic_plan shelter-app
Created application id: 7
```

4. Configure the `portal_openapi` product to allow access from the Admin Portal and the Developer Portal.

   4.1. Navigate to **Integration** > **Policies** from the `portal_openapi` product page. Then update the policy chain by adding the `CORS Request Handling` policy and placing it before the `3scale APIcast` policy in the policy chain.

   Configure the CORS policy by setting:

   - **allow_origin**: `(3scale-admin|3scale).apps.ocp4.example.com`

   The syntax between parentheses allows our API to be called from:

   - The Admin Portal domain: 3scale-admin.apps.ocp4.example.com
   - The Developer Portal domain: 3scale.apps.ocp4.example.com

   Then click **Update Policy** and then **Update Policy Chain**.

   4.2. Promote the `portal_openapi` configuration first to staging and then to production.

5. Verify that you can reach the `petshelter-api` RHOCP service by using the ActiveDocs in the Admin Portal.

   5.1. Click **ActiveDocs** from the side menu in the `portal_openapi` product. Then click **portal_openapi** to go to the interactive documentation.

   5.2. Click **GET /pet/findByStatus** to open the endpoint documentation.

   To test the call click **Try it out**, then:

   - Select the `available` value for the `status` parameter.
   - Click the **Fisrt userkey from latest 5 applications** drop-down menu and select the `shelter-app` to automatically fill the `user_key` parameter.
   - Click **Execute** to send the request.

   The response shows in the `200` code under the **Server response** section.

```
{
  "0": {
    "photoUrls": [
      "photo_1_url",
      "photo_2_url"
    ],
    "name": "spotty",
    "id": 0,
    "category": {
      "name": "dogs",
```

```
      "id": 6
    },
...output omitted...
```

6. Update the Developer Portal Documentation page to show all the ActiveDocs in the tenant.

   6.1. Open the `~/DO240/labs/portal-openapi/dev-portal-docs.liquid` file to copy its content.

   6.2. Navigate to the **Audience** page. Then click **Developer Portal** > **Content** and click **Documentation** to edit the documentation page.

   6.3. Scroll to the **Draft** section and replace the default template by pasting the contents from the `dev-portal-docs.liquid` file. Then click **Save** and **Publish**.

7. Use the 3scale Toolbox CLI to retrieve the user_key for the `shelter-app` application.

```
[student@workstation ~]$ 3scale application list \
  --service=portal_openapi 3scale-tenant -o json \
  | jq -r '.[] | select(.name == "shelter-app")|.user_key'
7acb0243fdc7a30882f39dbf95261ab7
```

8. Navigate to `https://3scale.apps.ocp4.example.com/docs` to test the ActiveDocs from the Developer Portal.

   Scroll to the `portal_openapi` documentation. Then click **GET /pet/findByStatus** to open the endpoint documentation.

   To test the call click **Try it out**, then:

   - Select the `available` value for the `status` parameter.
   - Paste the user_key from the previous step to fill the `user_key` parameter.
   - Click **Execute** to send the request.

   The response shows in the `200` code under the **Server response** section.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish portal-openapi
```

This concludes the guided exercise.

Quiz

Creating a Developer Portal

Choose the correct answers to the following questions:

1. **Consider that you want to create a new page in your Developer Portal, which is only visible to a restricted number of developers. Which of the following is the most appropriate? (Choose one.)**
   a. You create a new public page with a difficult and long path. Then, you share the URL with the corresponding developers.
   b. You create a new private page and a new group called `restricted-developers`.
   c. You create a new public page and a new group called `restricted-developers`. Then, you assign the corresponding users to the group.
   d. You create a new private page and a new group called `restricted-developers`. Then, you assign the corresponding users to the group.

2. **Which of the following are correct about the Developer Portal? (Choose two.)**
   a. Layouts are reusable elements that you can embed in pages.
   b. When you sign up in the Developer Portal, an account with a developer user and an application are created specifically for you.
   c. You can create pages by using Markdown instead of HTML.
   d. RSS is not supported.

3. **Consider the following code snippet from a layout of the Developer Portal. Which of the following statements are correct? (Choose two.)**

```
{% if provider.name == "Red Hat" %}
    <img src="{{ provider.logo_url }}" height="32px" />
{% else %}
    <div style="width: 100px; height: 100px; background-color: black; position:
 relative; color: white;">
        {{ request.path }}
    </div>
{% endif %}
```

   a. If the name of the Developer Portal is `Red Hat`, then the logo of the Developer Portal is rendered.
   b. If the name of the Developer Portal is NOT `Red Hat`, then a black square with the logo of the Developer Portal is rendered.
   c. If the name of the Developer Portal is `Red Hat`, then a black square is rendered.
   d. If the name of the Developer Portal is NOT `Red Hat`, then a black square with the current path is rendered.

4. Consider the following 3scale Toolbox CLI command. Which of the following statements are correct about the command? (Choose two.)

```
[user@host ~]$ 3scale import openapi \
  --target_system_name=redhat \
  --destination=3scale \
  /local/folder/definition.json
```

a. A product and a backend are created in the `redhat` tenant.
b. A product and a backend with the system name `redhat` are created.
c. A product and a backend are created in the `3scale` tenant.
d. A product and a backend called `3scale` are created.

Solution

Creating a Developer Portal

Choose the correct answers to the following questions:

1. **Consider that you want to create a new page in your Developer Portal, which is only visible to a restricted number of developers. Which of the following is the most appropriate? (Choose one.)**
   a. You create a new public page with a difficult and long path. Then, you share the URL with the corresponding developers.
   b. You create a new private page and a new group called `restricted-developers`.
   c. You create a new public page and a new group called `restricted-developers`. Then, you assign the corresponding users to the group.
   d. You create a new private page and a new group called `restricted-developers`. Then, you assign the corresponding users to the group.

2. **Which of the following are correct about the Developer Portal? (Choose two.)**
   a. Layouts are reusable elements that you can embed in pages.
   b. When you sign up in the Developer Portal, an account with a developer user and an application are created specifically for you.
   c. You can create pages by using Markdown instead of HTML.
   d. RSS is not supported.

3. **Consider the following code snippet from a layout of the Developer Portal. Which of the following statements are correct? (Choose two.)**

```
{% if provider.name == "Red Hat" %}
    <img src="{{ provider.logo_url }}" height="32px" />
{% else %}
    <div style="width: 100px; height: 100px; background-color: black; position:
 relative; color: white;">
        {{ request.path }}
    </div>
{% endif %}
```

   a. If the name of the Developer Portal is `Red Hat`, then the logo of the Developer Portal is rendered.
   b. If the name of the Developer Portal is NOT `Red Hat`, then a black square with the logo of the Developer Portal is rendered.
   c. If the name of the Developer Portal is `Red Hat`, then a black square is rendered.
   d. If the name of the Developer Portal is NOT `Red Hat`, then a black square with the current path is rendered.

4. Consider the following 3scale Toolbox CLI command. Which of the following statements are correct about the command? (Choose two.)

```
[user@host ~]$ 3scale import openapi \
  --target_system_name=redhat \
  --destination=3scale \
  /local/folder/definition.json
```

a. A product and a backend are created in the `redhat` tenant.

b. A product and a backend with the system name `redhat` are created.

c. A product and a backend are created in the `3scale` tenant.

d. A product and a backend called `3scale` are created.

Summary

In this chapter, you learned:

- The various parts of the Developer Portal in 3scale API Management.
- How to modify and customize the Developer Portal.
- How to restrict access to pages within the portal.
- How to use an OpenAPI specification to provide documentation for your APIs.

Chapter 6

Monitoring APIs with Red Hat 3scale API Management

Goal

Configure monitoring and analytics for APIs and the 3scale API management infrastructure.

Objectives

- Configure API analytics to gather runtime metrics for APIs.
- Configure monitoring and metrics collection for the 3scale API Management infrastructure.

Sections

- Configuring API Analytics (and Guided Exercise)
- Monitoring 3Scale Infrastructure (and Guided Exercise)
- Monitoring APIs with Red Hat 3scale API Management (Quiz)

Configuring API Analytics

Objectives

After completing this section, you should be able to configure API analytics to gather runtime metrics for APIs.

Introducing Metrics in 3scale

You use a *metric* to track the usage of your APIs deployed on Red Hat 3scale API Management. For example, you can define a new metric to track the number of times that the `403` HTTP status code is returned. Every time your API returns a `403` status code, the metric is incremented.

You can define a new metric at the 3cale API Management product level, therefore, a product is composed of one or many metrics. By default, 3scale creates the `Hits` metric, which counts the number of requests that reach the product at any path. For example, if you create a product called `product1`, then 3cale API Management creates the `Hits` metric for the `/` path, which means that a request to any path increments the metric.

A metric is composed of several fields. The following table gives an explanation of every field.

Field	Explanation
Friendly name	The name used in the Admin Portal UI.
System name	A unique name that identifies the metric within the 3scale API Management system.
Unit	The quantifiable measurement that is incremented.
Description	Additional information that you might find useful to identify the metric.

Managing Metrics

To manage the metrics you can use the Admin Portal or the CLI Toolbox.

Managing Metrics Through the Admin Portal

To manage the metrics through the Admin Portal, navigate to the product page of the product that you want to manage. Then, click **Integration:** > **Methods & Metrics**. The page displays the metrics for the product.

To create a new metric, click **New metric** and complete the form.

Managing Metrics Through the CLI Toolbox

To manage the metrics through the CLI Toolbox, you can use the `3scale metric` super command.

You can create a new metric by using the `3scale metric create` command. For example, if you want to create a metric `metric1` for the `product1` product in the 3scale tenant, then a possible command is as follows:

```
[student@workstation ~]$ 3scale metric create 3scale-tenant product1 metric1
 --unit=1
```

You can update an existing metric by using the `3scale metric apply` command. The following command updates the name for the `metric1` metric.

```
[student@workstation ~]$ 3scale metric apply 3scale-tenant product1 metric1
 --name=metric2
```

You can list the existing metrics by using the `3scale metric list`.

```
[student@workstation ~]$ 3scale metric list 3scale-tenant product1
ID FRIENDLY_NAME SYSTEM_NAME UNIT DESCRIPTION
8 Hits hits hit Number of API hits
10 status-203 status-203 1 status 203
```

The previous commands include several options to modify the metric, such as name or description. For a complete understanding of the options associated to every command, visit the official Red Hat documentation [https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/operating_3scale/index#creating-metrics].

Incrementing Metrics

The main purpose of a metric is to measure, therefore, it must be incremented based on your specific use cases.

Incrementing Metrics Linked to a Mapping Rule

A metric can be linked to a mapping rule. When you create a mapping rule, you have the option to select a metric to increment. If a metric is linked to a mapping rule, then the metric gets incremented every time that the mapping rule receives a request.

Incrementing Metrics Through an APIcast Custom Policy

If the metric is not linked to a mapping rule, then an external system must perform the incrementation. APIcast includes a policy, `Custom Metrics`, which you can use to increment a metric based on the HTTP status code of the response of the API. For example, if you want to track the number of 300 status codes that your API returns, then you can configure the `Custom Metrics` policy to increment a metric on every response that meets the criteria.

To add the policy, navigate to **Integration:** > **Policies** and click **Add policy**. Then, select `Custom Metrics` from the policies list. For the policy to work, you must give it priority over the default `APIcast policy` policy.

The policy is composed of one or several `operations` combined by a `condition`.

You can consider that every `operation` is an equality expression with two operands: left and right. The `operation` holds an `op` field, which is applied to the `left` and `right` operands.

```
left (op) right
```

The `left` and `right` fields can be plain text, or Liquid (the programming language) expressions. In the Liquid programming language, variables are enclosed with two brackets (for example, `{{variable}}`). The policy provides the built-in variable `{{status}}`, which is used to get the status code of the response. For example, if you only want the metric to be incremented when the status code is 404, then the expression is similar to the following:

```
404 matches {{status}}
```

3scale API Management evaluates the previous expression on every request, and only increments the metric if the evaluation is `true`. You can include complex expressions by creating several `operations` combined by a logical expression (`and` or `or`).

```
(404 matches {{status}}) OR (500 matches {{status}})
```

The previous expression increments the metric if the status code is equal to 404 or 500.

To configure the policy, you must complete the form, which includes several fields.

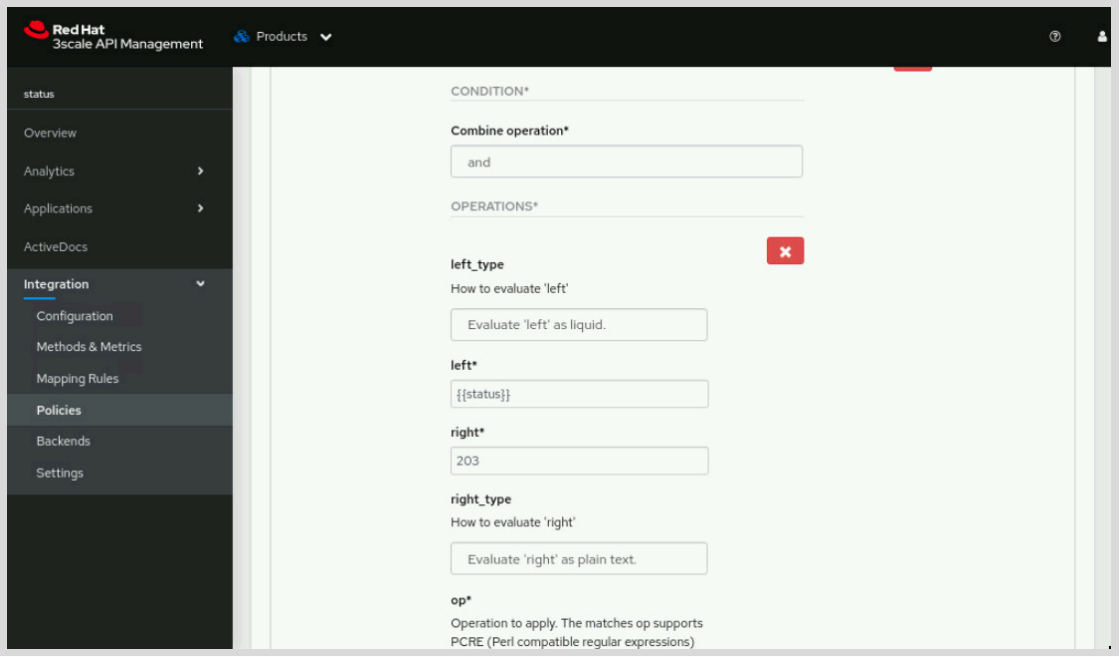


Figure 6.1: Sample product with a configured Custom Metrics policy

The following table explains the fields involved in the configuration of the policy.

Field	Explanation
left	The left operand of the expression.
left_type	The type of the left operand (either plain text or liquid expression).
op	The operation to apply to the left and right operands.

Field	Explanation
right	The right operand of the expression.
right_type	The type of the right operand (either plain text or liquid expression).
Metric to increment	The system name of the metric to increment if the expression evaluates to `true`.
Increment	The number that is added to the metric.

References

Red Hat 3scale API Management documentation - Creating Metrics
https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/operating_3scale/index#creating-metrics

Guided Exercise

Configuring API Analytics

In this exercise, you will use an APIcast policy to create a metric that measures the number of 203 HTTP status codes for a product.

Outcomes

You should be able to configure the `Custom Metrics` APIcast policy to increment a metric based on the HTTP status code of the response.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start monitoring-analytics
```

The `start` function deploys the `status-api` application. The application exposes an endpoint, `/status/{code}`, which returns the HTTP status code (200, 201 or 203) provided in the path parameter. You will use this API to track the number of responses that return the 203 status code. At the same time, a product and a backend, called `status`, are created to perform the exercise.

Instructions

1. Explore the `status` product and review the default metrics.

   1.1. In the 3scale Admin Portal, navigate to the `monitoring_analytics` product page.

   1.2. In the left pane, click **Analytics** > **Traffic**. The default metric, `Hits`, is incremented every time that a request for the product arrives.

2. Create a new metric to track the number of requests to the `/status` endpoint of the Status API.

   2.1. In the left pane, click **Integration:** > **Methods & Metrics**. Then, click **New metric**. Complete the form according to the following values:

Field	Value
Friendly name	`status`
System name	`status`
Unit	`1`
Description	Tracking requests to the status endpoint.

Click **Create Metric**.

2.2. In the `status` metric row, click **Add a mapping rule** to attach the metric to the `/status` path. Complete the form according to the following values:

Field	Value
Verb	`GET`
Pattern	`/status`
Method or Metric to increment	Select `Metric`. Then, select `status`.

Then, click **Create Mapping Rule**.

3. Create a new metric to track the 203 HTTP status codes returned by the `status` product.

3.1. In the left pane, click **Integration:** > **Methods & Metrics**. Then, click **New metric**. Complete the form according to the following values:

Field	Value
Friendly name	`status-203`
System name	`status-203`
Unit	`1`
Description	Tracking of 203 HTTP response codes.

Click **Create Metric**.

4. Add the `Custom Metrics` policy, so that APIcast can increment the `status-203` metric on every request that returns a 203 HTTP status code.

4.1. In the left pane, click **Integration:** > **Policies**. Then, click **Add policy**.

4.2. Within the policies list, select `Custom Metrics`. Then, use the arrow icon to move the `Custom Metrics` policy to the first position.

4.3. Click **Custom Metrics** to update the policy. Complete the form according to the following values:

Field	Value
Metric to increment	`status-203`
left	`{{status}}`
right	`203`
op	`matches`
right_type	`Evaluate 'right' as plain text`
left_type	`Evaluate 'left' as liquid`

This policy increments the `status-203` metrics whenever the status code of the response is `203`.

4.4. Click **Update Policy**. Then, click **Update Policy Chain**.

5. Promote the changes of the `status` product to staging.

5.1. Navigate to **Integration:** > **Configuration**.

5.2. Click **Promote v.1 to Staging APIcast** to promote the changes.

5.3. From the `curl` example, copy the `user_key` parameter. You use this later in this exercise.

6. Verify that the `status-203` metric is only incremented when a 203 HTTP status code is returned.

6.1. In your command-line terminal, run the `DO240/labs/monitoring-analytics/test_status_api.sh` script by providing your user key as a parameter. The script sends 20 requests to the Status API. The script also collects the number of times that every status code is returned.

```
[student@workstation ~]$ DO240/labs/monitoring-analytics/test_status_api.sh \
 USER_KEY
Making 20 requests
Responses with status code 200: 7
Responses with status code 201: 8
Responses with status code 203: 5
```

Track the number of responses with the 203 status code.

6.2. In the Admin Portal, navigate to **Analytics:** > **Traffic**. In the drop-down menu, select the `status-203` metric.

6.3. The number that the `status-203` metric displays should be equal to the number of responses with the 203 status code provided by the `test-status-api.sh` script. Although the Status API has returned other status codes, only the 203 status code has incremented the metric.

6.4. In the drop-down menu, select the `status` metric. The number displayed should be equal to 20, which is the number of total requests sent to the product.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish monitoring-analytics
```

This concludes the guided exercise.

Monitoring 3Scale Infrastructure

Objectives

After completing this section, you should be able to configure monitoring and metrics collection for the 3scale API Management infrastructure.

Outlining the Monitoring Stack

The 3scale API Management monitoring stack consists of Prometheus, Grafana, and a set of configuration objects to connect all three.

Prometheus is responsible for collecting and managing the monitoring and metrics data. *Grafana* provides an interface and widgets for creating dashboards with the metrics.

Both the Prometheus and Grafana operators provide *Custom Resource Definitions* (CRDs). With monitoring enabled, the 3scale API Management operator creates resources based on these CRDs to automatically configure Prometheus and Grafana.

Enabling Monitoring

To enable the monitoring functionality within 3scale API Management, you must do so within the APImanager definition.

For example, in the following APImanager definition, the highlighted `monitoring` attribute instructs the 3scale API Management operator to create applicable Prometheus and Grafana resources.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-sample
  namespace: 3scale
spec:
  wildcardDomain: apps.ocp4.example.com
  monitoring:
    enabled: true
```

Collecting Data With Prometheus

Prometheus serves the role of collecting and storing metrics data, as well as providing a query language, *PromQL*, for the data.

There are multiple ways to run Prometheus within Red Hat OpenShift Container Platform (RHOCP). However, this course only focuses on using the Prometheus operator.

The 3scale API Management operator is designed to integrate with the Prometheus operator. Specifically, the 3scale API Management operator configures a standard set of Prometheus alerting rules and pod monitors.

Alerting rules define certain conditions that, when met, should trigger actions. For example, you might use an alert rule to send an email notification whenever CPU usage exceeds a certain threshold. *Pod monitors* gather metrics data from pods within the cluster.

In addition to installing the operator, you must also create a Prometheus instance by creating a resource of type `Prometheus`. The following is an excerpt of an example Prometheus definition:

```
kind: Prometheus
...output omitted...
spec:
  serviceAccountName: prometheus-k8s (1)
  podMonitorSelector:
   matchExpressions:
   - key: app (2)
     operator: In
     values:
     - my-app
```

(1) The name of the Red Hat OpenShift service account that the Prometheus instance will use. This service account should have permissions to access all selected resources.

(2) This expression configures Prometheus to look for pod monitors with the `app=my-app` label. Alert rules and service monitors are selected similarly by using the `ruleSelector` and `podMonitorSelector` attributes, respectively.

Displaying Data With Grafana

Although Prometheus can be directly queried, its display functionality is limited. Grafana can consume data directly from Prometheus and display easy-to-read and customizable graphs.

Similar to Prometheus, this course focuses on installing Grafana by using an operator. You must also create a Grafana instance by creating a resource of type `Grafana`. The following is an excerpt of an example Grafana definition:

```
kind: Grafana (1)
...output omitted...
spec:
  config: (2)
    ...output omitted...
  dashboardLabelSelector:
  - matchExpressions:
    - key: app (3)
      operator: In
      values:
      - my-app
```

(1) The resource type must be `Grafana`.

(2) You must provide a configuration object which is passed to the Grafana instance.

(3) This expression configures Grafana to look for dashboard resources with the `app=my-app` label.

For the Grafana instance to retrieve data from the Prometheus instance, you also need a resource of type `GrafanaDataSource`, which provides connection information. The following is an excerpt of an example Grafana data source definition:

```
kind: GrafanaDataSource (1)
...output omitted...
spec:
  name: middleware
  datasources:
    - name: Prometheus
      url:  http://prometheus-operated:9090 (2)
      ...output omitted...
```

(1) The resource type must be `GrafanaDataSource`.

(2) You must provide a connection string that resolves to the Prometheus instance.

Available Dashboards

Once you have monitoring set up, you can view the default metrics and dashboards.

The 3scale API Management operator creates seven Grafana dashboards. Each of these uses the data that Prometheus gathers from the various components within 3scale API Management.

The following are some of the default dashboards and the metrics that they display:

- `Apicast`: Shows APIcast instance metrics, such as the number of error requests in the last hour. Use the **environment** selection menu to choose between the `staging` and `production` APIcast instances.
- `System`: Shows metrics about 3scale API Management itself, such as tenants requests per second.
- `Zync`: Shows metrics from the Zync component and its RHOCP pods, such as the number of Zync requests per second and CPU usage.

References

For more information, refer to the *Monitoring 3scale* chapter in the *Red Hat 3scale API Management 2.11 Operating 3scale* guide at https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html/operating_3scale/monitoring-threescale

Guided Exercise

Monitoring 3Scale Infrastructure

In this exercise, you will configure the monitoring of 3scale API Management resources.

Outcomes

You should be able to:

- Install the Prometheus operator
- Install the Grafana operator
- Enable monitoring within a 3scale API Management installation

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start monitoring-infrastructure
```

Instructions

1. Review the available metrics by using the APIcast staging pod.

   1.1. Within a command line terminal, log in as the `admin` user and select the `3scale` project with the `oc` command. When prompted, use the password `redhat`.

```
[student@workstation ~]$ oc login -u admin  https://api.ocp4.example.com:6443
...output omitted...
Login successful.
...output omitted...
[student@workstation ~]$ oc project 3scale
Now using project "3scale" ...output omitted...
```

   1.2. Open a remote shell connection to the APIcast staging pod.

```
[student@workstation ~]$ oc rsh svc/apicast-staging
sh-4.4$ **
```

   1.3. Make a request to `localhost`, which is the APIcast pod itself, to retrieve the list of available metrics.

```
sh-4.4$ curl  http://localhost:9421/metrics
...output omitted...
nginx_http_connections{state="accepted"} 129
nginx_http_connections{state="active"} 1
nginx_http_connections{state="handled"} 129
...output omitted...
```

Exit the remote shell by running `exit` or pressing `Control-d`.

2. Deploy the Prometheus operator to the `3scale` project.

   2.1. In a web browser, log in to the Red Hat OpenShift Container Platform (RHOCP) web console by using the `htpasswd` provider, username `admin`, and password `redhat`.

   The URL for the RHOCP web console is `https://console-openshift-console.apps.ocp4.example.com`.

   2.2. Navigate to **Operators** > **OperatorHub** and search for `prometheus`.

   2.3. Make sure the `3scale` project is selected by using the **Project** selection menu at the top of the page.

   2.4. Click the **Prometheus Operator** entry and click **Install**.

   2.5. Make sure the `Installed Namespace` is set to `3scale` and click **Install**.

   2.6. Review the `DO240/labs/monitoring-infrastructure/prometheus-3scale-monitor.yml` resource file.

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: 3scale-monitor
  namespace: 3scale
spec:
  serviceAccountName: prometheus-k8s
  podMonitorSelector:
   matchExpressions:
   - key: app
     operator: In
     values:
     - 3scale-api-management
  ruleSelector:
    matchExpressions:
    - key: app
      operator: In
      values:
      - 3scale-api-management
```

Applying the preceding resource creates a new Prometheus instance within the `3scale` project. This instance watches for pod monitors and Prometheus rules that have the RHOCP label `app=3scale-api-management`.

   2.7. Within a command line terminal, apply the `prometheus-3scale-monitor.yml` file. Make sure you are logged in as the `admin` user.

```
[student@workstation ~]$ oc apply -f \
DO240/labs/monitoring-infrastructure/prometheus-3scale-monitor.yml
prometheus.monitoring.coreos.com/3scale-monitor created
```

3. Install the Grafana operator to the `3scale` project.

   3.1. Within a web browser, navigate back to **Operators** > **OperatorHub** and search for `grafana`.

   3.2. Click the **Grafana Operator** entry and click **Install**.

   3.3. Make sure the `Installed Namespace` is set to `3scale` and click **Install**.

   3.4. Review the `DO240/labs/monitoring-infrastructure/grafana-3scale.yml` file.

```
apiVersion: integreatly.org/v1alpha1
kind: Grafana
metadata:
  name: grafana
  namespace: 3scale
spec:
  ingress:
    enabled: True
  config:
    log:
      mode: "console"
      level: "warn"
    auth:
      disable_login_form: False
      disable_signout_menu: True
    auth.anonymous:
      enabled: True
  dashboardLabelSelector:
  - matchExpressions:
    - key: app
      operator: In
      values:
      - 3scale-api-management
```

   Applying the preceding resource creates a Grafana instance within the `3scale` project. This instance also watches for the `app=3scale-api-management` RHOCP label.

   3.5. Within a command line terminal, apply the `grafana-3scale.yml` file. The Grafana operator creates the relevant resources, including a Grafana pod.

```
[student@workstation ~]$ oc apply -f \
DO240/labs/monitoring-infrastructure/grafana-3scale.yml
grafana.integreatly.org/grafana created
```

   3.6. Review the `DO240/labs/monitoring-infrastructure/grafana-prometheus-datasource.yml` file.

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDataSource
metadata:
  name: prometheus
  namespace: 3scale
spec:
  name: middleware
  datasources:
    - name: Prometheus
      type: prometheus
      access: proxy
      url: http://prometheus-operated:9090
      isDefault: true
      version: 1
      editable: true
      jsonData:
        timeInterval: "5s"
```

3.7. Apply the `grafana-prometheus-datasource.yml` file. This creates a new data source that enables Grafana to connect to the Prometheus instance that you created previously.

```
[student@workstation ~]$ oc apply -f \
DO240/labs/monitoring-infrastructure/grafana-prometheus-datasource.yml
grafanadatasource.integreatly.org/prometheus created
```

4. Enable monitoring for 3scale API Management by updating the APIManager.

4.1. Observe that no pod monitors or rules exist in the `3scale` project.

```
[student@workstation ~]$ oc get podmonitors
No resources found in 3scale namespace.
[student@workstation ~]$ oc get prometheusrules
No resources found in 3scale namespace.
```

4.2. Edit the APIManager you created as part of the 3scale API Management set up:

```
[student@workstation ~]$ oc edit apimanager apimanager-sample
```

This opens an editor to modify the resource definition.

4.3. Within the `spec` definition, add the following snippet:

```
...output omitted...
spec:
  ...output omitted...
  monitoring:
    enabled: true
  ...output omitted...
```

Save and quit the editor to update the APIManager.

4.4. Observe that the 3scale API Management operator creates relevant Prometheus resources.

```
[student@workstation ~]$ oc get podmonitors
...output omitted...
apicast-production   1s
apicast-staging      1s
...output omitted...
[student@workstation ~]$ oc get prometheusrules
...output omitted...
apicast                         51s
backend-listener                52s
...output omitted...
```

5. View the 3scale API Management metrics within the Grafana web interface.

5.1. In a web browser, navigate to `https://grafana-route-3scale.apps.ocp4.example.com/dashboards`.

5.2. Click **3scale** to expand the list of available dashboards. Then, click **Apicast** to view a dashboard showing APIcast metrics.

5.3. In the top right of the page, select **Last 5 minutes** from the selection menu. This makes it easier to see the data because Prometheus does not receive any data until monitoring is enabled.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish monitoring-infrastructure
```

This concludes the guided exercise.

Quiz

Monitoring APIs with Red Hat 3scale API Management

Choose the correct answers to the following questions:

1. **Which two of the following components are a part of the 3scale API Management monitoring stack?**
   a. Prometheus
   b. Logstash
   c. Grafana
   d. Kibana

2. **What one of the following is the primary role of a Prometheus instance?**
   a. Collecting metrics data.
   b. Collecting service URLs.
   c. Displaying graphs.
   d. Managing the Grafana instance.

3. **Which one of the following metrics is the default metric within a 3scale API Management product and what does it collect?**
   a. `Hits`, which counts the number of requests that 3scale API Management processes across all products.
   b. `Hits`, which counts the number of requests that reach the product at any path.
   c. `Requests`, which counts the number of requests that reach the product at any path.
   d. `Requests`, which counts the number of requests that reach a specified backend.

4. **Other than creating a mapping rule, which one of the following responses best describes a 3scale API Management feature that enables you to increment a custom metric?**
   a. The `Increment Metric` policy
   b. The `Custom Metrics` policy
   c. Prometheus and Grafana
   d. You must create a mapping rule.

5. **Which one of the following definitions best describes a Prometheus alert rule?**
   a. Alert rules establish the connection between Prometheus and Grafana.
   b. Alert rules define how a graph appears within the Prometheus user interface.
   c. Alert rules are used to trigger configured actions, such as sending an email notification once a certain condition is met.
   d. Alert rules are not a feature of Prometheus.

6. Your team has a product called `my-app` in which you would like to track the number of request errors with HTTP status code 500. Which of the following best describes the process you should follow?

a. Create a new metric in the product, add the `Custom Metrics` policy to the product's policy chain, and promote the configuration.
b. Create a new metric in the product, add a mapping rule, and promote the configuration.
c. Create a new metric in the product and then create an alerting rule in Prometheus.
d. The 3scale API Management creates a default Grafana dashboard with this metric already included.

7. For the same scenario as outlined in question 6, when setting up the operands for the `Custom Metrics` policy, what would be the `left` and `right` values, respectively? Consider that `left_type = liquid` and `right_type = plain text`.

a. `left` and `500`
b. `500` and `{{status}}`
c. `{{status}}` and `500`
d. `status` and `{{500}}`

Solution

Monitoring APIs with Red Hat 3scale API Management

Choose the correct answers to the following questions:

1. Which two of the following components are a part of the 3scale API Management monitoring stack?

a. Prometheus
b. Logstash
c. Grafana
d. Kibana

2. What one of the following is the primary role of a Prometheus instance?

a. Collecting metrics data.
b. Collecting service URLs.
c. Displaying graphs.
d. Managing the Grafana instance.

3. Which one of the following metrics is the default metric within a 3scale API Management product and what does it collect?

a. `Hits`, which counts the number of requests that 3scale API Management processes across all products.
b. `Hits`, which counts the number of requests that reach the product at any path.
c. `Requests`, which counts the number of requests that reach the product at any path.
d. `Requests`, which counts the number of requests that reach a specified backend.

4. Other than creating a mapping rule, which one of the following responses best describes a 3scale API Management feature that enables you to increment a custom metric?

a. The `Increment Metric` policy
b. The `Custom Metrics` policy
c. Prometheus and Grafana
d. You must create a mapping rule.

5. Which one of the following definitions best describes a Prometheus alert rule?

a. Alert rules establish the connection between Prometheus and Grafana.
b. Alert rules define how a graph appears within the Prometheus user interface.
c. Alert rules are used to trigger configured actions, such as sending an email notification once a certain condition is met.
d. Alert rules are not a feature of Prometheus.

6. Your team has a product called `my-app` in which you would like to track the number of request errors with HTTP status code 500. Which of the following best describes the process you should follow?

a. Create a new metric in the product, add the `Custom Metrics` policy to the product's policy chain, and promote the configuration.

b. Create a new metric in the product, add a mapping rule, and promote the configuration.

c. Create a new metric in the product and then create an alerting rule in Prometheus.

d. The 3scale API Management creates a default Grafana dashboard with this metric already included.

7. For the same scenario as outlined in question 6, when setting up the operands for the `Custom Metrics` policy, what would be the `left` and `right` values, respectively? Consider that `left_type = liquid` and `right_type = plain text`.

a. `left` and `500`

b. `500` and `{{status}}`

c. `{{status}}` and `500`

d. `status` and `{{500}}`

Summary

In this chapter, you learned:

- How to create custom metrics within 3scale API Management.
- How to integrate custom metrics with your services by using a policy.
- How to use the Prometheus and Grafana operators to monitor 3scale API Management.

Chapter 7

Monetizing APIs with Red Hat 3Scale API Management

Goal

Monetize APIs by configuring billing plans and pricing rules.

Objectives

- Configure billing for API consumers.
- Configure pricing rules for APIs.

Sections

- Configuring Billing for APIs
- Configuring Billing for APIs (Guided Exercise)
- Configuring Pricing Rules for API Consumption (and Guided Exercise)
- Monetizing APIs with Red Hat 3Scale API Management (Quiz)

Configuring Billing for APIs

Objectives

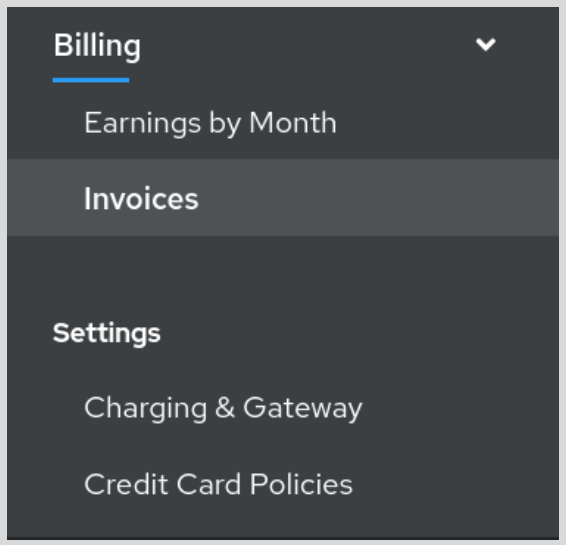
After completing this section, you should be able to configure billing for API consumers.

Billing in Red Hat 3scale API Management

With 3scale API Management you can set up billing and handle customer payments, either manually or automatically.

3scale API Management billing generates a monthly invoice for the customer based on the application plans the customer signed up for. Customers in 3scale API Management are the developer accounts that signed up to use your APIs.

To configure billing in the Admin Portal navigate to the **Audience** page. There you can access the **Billing** menu.



Automatic billing is enabled by default for all the accounts. You can disable automatic billing for individual accounts by navigating to the account and clicking **Disable** in the billing section from the account details page.

You can enable automatic payments globally. For this, you must navigate to **Billing** > **Settings** > **Charging & Gateway** and click **Charging enabled**. This affects every developer account in the tenant, but you can disable it for individual accounts in the billing section from the account details page.

To use automatic payments you must configure a payment gateway. 3scale API Management offers integration's with credit card gateways like Stripe or Braintree.

Billing Configuration

To configure billing, navigate to **Billing** > **Settings** > **Charging & Gateway**.

There you have the following options to customize the invoices that 3scale API Management sends to customers.

Mode
: You can select `Prepaid` or `Postpaid` depending on which month you want clients to pay in the current month invoice.

Charging enabled
: You must enable this if you want to have automated payments by using a payment gateway.

Currency
: You can pick the currency for the invoice. If you set up automatic payments, pick a currency supported by your payment gateway.

Invoice footnote
: The text you provide shows at the bottom of every PDF invoice.

Text to show if VAT/Sales Tax is 0%
: The text that shows when VAT/Sales Tax is explicitly set to 0.

Billing periods for invoice IDs
: Invoice identifier format used for administrative purposes. You can set it to `monthly` or `yearly` string formats, which correspond to `YYYY-MM-XXXXXXXX` and `YYYY-XXXXXXXX` respectively. This setting only affects the formatting of the identifier. Only automated monthly billing is supported.

Managing Invoices

You can list and search the existing invoices by navigating to **Billing** > **Invoices**.

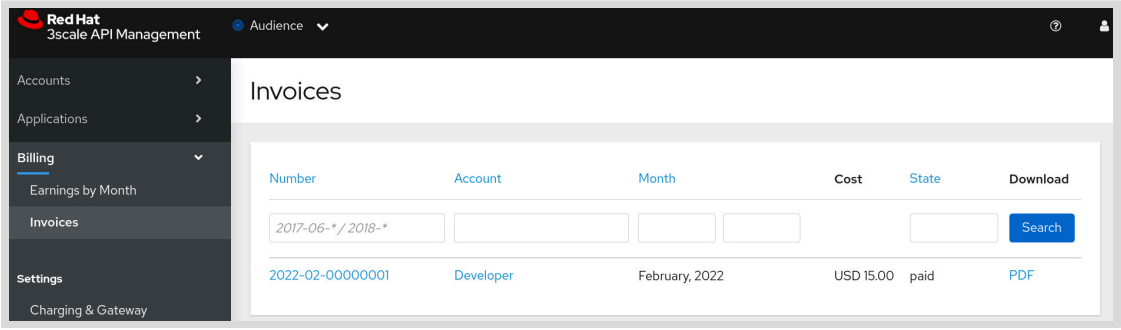


Figure 7.2: Invoice list

From the invoice list you can see some invoice details like identifier, the developer account, and the invoice state.

In 3scale API Management an invoice can be in the following states.

Open
: The invoice is created and the invoice contents can be updated.

Finalized
: The invoice has all the current billing period charges added to it.

Pending
: 3scale API Management issued the invoice to the customer and the invoice is awaiting payment.

Unpaid
: The invoice payment failed, but the system will retry.

Paid
The invoice payment terminated with success.

Failed
The invoice payment failed after several attempts and 3scale API Management stops retrying.

Canceled
The administrator canceled the invoice.

Billing Modes and the Billing Process

3scale API Management offers two billing modes:

Prepaid
3scale API Management bills fixed and set-up fees at the beginning of the month. Variable costs for the current month are billed at the beginning of the following month.

Postpaid
3scale API Management bills all the fees at the beginning of next month.

The billing process executes every day. The tasks to execute differ depending on whether it is the the first day of the month, and whether the billing mode is set to `prepaid` or `postpaid`.

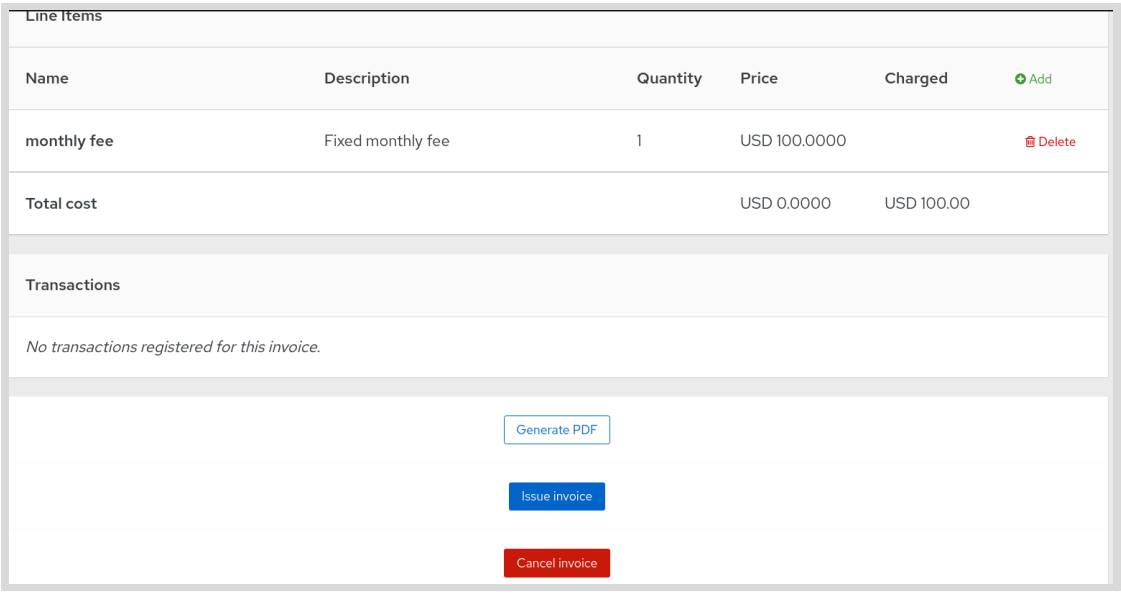
The following tasks execute the first day of the month:

- Prepaid
  - Bill fixed costs for current month.
  - Bill variable costs for previous month.
  - Finalize the invoice for the current month.
- Postpaid
  - Bill variable cost for previous month.
  - Finalize `open` invoices for the previous month.
  - Create an `open` invoice for this month and bill fixed costs.

The following tasks execute every day of the month, including the first day:

- Create an `open` invoice for today's new contracts and expired trials.
- If the payment method is `prepaid`, finalize invoices with an `open` state.
- Issue invoices to customers and change the state from `finalized` to `pending`.
- Charge invoices. Invoices in an `unpaid` and `pending` state are charged if the `Due On` invoice property is set for today or earlier. A failed payment changes the state to `unpaid`. After three failed payment attempts the invoice is set to `failed`.
- Notify the credit cards that are about to expire.

You can manually trigger these tasks from the Admin Portal. For this, navigate to the selected invoice by clicking the invoice identifier in the invoice list. If the selected invoice is not `paid` you can perform different actions. For example, if the invoice is `open` then you can issue the invoice by clicking **Issue invoice** and leave it in a `pending` state.



References

Credit card gateways for payments

https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/admin_portal_guide/index#credit-card-gateways-for-payments

Setting VAT/Sales Tax Values

https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/admin_portal_guide/index#vat-sales

Adding currencies

https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/admin_portal_guide/index#yaml_configuration_for_currencies

Guided Exercise

Configuring Billing for APIs

In this exercise, you will configure billing in Red Hat 3scale API Management to prepaid mode to automatically generate invoices for the paid plans for your API.

Outcomes

You should be able to:

- Configure 3scale API Management to `prepaid` mode.
- Change an account to a paid application plan.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

This command:

- Deploys the application you will use in this exercise.
- Configures a 3scale API Management product and backend.

```
[student@workstation ~]$ lab start monetizing-billing
```

Instructions

1. Set billing to `prepaid` mode in the Admin Portal.

   1.1. In a web browser, log in to the the 3scale API Management Admin Portal as the `admin` user.

   1.2. From the **Audience** page, navigate to **Billing** > **Settings** > **Charging & Gateway** and click **Switch to PREPAID**.

   When a prompt shows asking for confirmation, click **OK**.

2. Create a monthly plan with a set-up fee of $30 and a monthly fee of $10.

```
[student@workstation ~]$ 3scale application-plan create --publish \
  --approval-required=false --cost-per-month=10.0 --setup-fee=30.0 \
  3scale-tenant monetizing_billing "monthly_plan"
Created application plan id: 11. Default: false; Disabled: false
```

3. Create an application for the `john` user account and subscribe to the `monthly_plan`.

```
[student@workstation ~]$ 3scale application create 3scale-tenant \
  john monetizing_billing monthly_plan monetizing_billing_app
Created application id: 7
```

4. Update the 3scale API Management billing process to run every minute.

4.1. Examine the `/home/student/DO240/labs/monetizing-billing/sidekiq_schedule.yml` file.

It contains the job scheduling configuration for the billing process. Every cron field has a * character to run the billing job every minute.

Warning

The sidekiq_shcedule.yml file for this exercise is intentionally incomplete and contains only the cron definition for billing jobs. If you leave this configuration other types of jobs will not execute.

4.2. Log in to Red Hat OpenShift Container Platform (RHOCP) as the `admin` user.

```
[student@workstation ~]$ oc login -u=admin -p=redhat \
  --server=https://api.ocp4.example.com:6443
Login successful.
```

4.3. Patch the `system` configmap to include the new schedule for the billing job.

```
[student@workstation ~]$ oc -n 3scale patch cm system --type merge \
  --patch-file /home/student/DO240/labs/monetizing-billing/sidekiq_schedule.yml
configmap/system patched
```

4.4. Patch the `system-sidekiq` deploymentconfig object to add the `sidekiq_schedule.yml` to the files that override the default configuration.

```
[student@workstation ~]$ oc -n 3scale patch dc system-sidekiq \
  --patch-file /home/student/DO240/labs/monetizing-billing/system-volumes.yml
deploymentconfig.apps.openshift.io/system-sidekiq patched
```

5. Run the `/home/student/DO240/labs/monetizing-billing/update_trial_dates.sh` script to set the trial expiration date of `monetizing_billing_app` to yesterday. The billing process generates invoices the day after the free trial expires.

```
[student@workstation ~]$ sh \
  /home/student/DO240/labs/monetizing-billing/update_trial_dates.sh
mysql: [Warning] Using a password on the command line interface can be insecure.
```

6. Verify that the billing process generates an invoice for the `Developer` account.

6.1. Navigate to the **Audience** page in the Admin Portal. Then click **Billing** > **Invoices**.

6.2. Verify that the billing process generated an invoice for the `Developer` account and that the invoice state is set to Finalized.

The billing process job could take a minute until it runs again. Refresh the page if the invoice list is empty.

6.3. Navigate to the invoice details by clicking the invoice number. Then verify that two line items show in the **Line Items** section:

- *Fixed fee ('monthly_plan')* with a max value of $10 depending on how close today is to the beginning of the month.
- *Setup fee ('monthly_plan')* with a value of $30.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish monetizing-billing
```

This concludes the guided exercise.

Configuring Pricing Rules for API Consumption

Objectives

After completing this section, you should be able to configure pricing rules for APIs.

Describing Pricing

Along with exposing your APIs to the public, you might want to charge your consumers for using your APIs. By using Red Hat 3scale API Management you can set price rates for the requests at the product level.

For example, if you have a product called `example` with two paths, `/test` and `/test1`, you might want to charge your consumers for using one or both paths.

In 3scale API Management, pricing rates are linked with application plans. When you create an application plan, you can choose fixed and variable charges for your APIs. Then, you link an application plan with an application, and your consumers use the application's user key to make requests.

Describing Fixed Costs

The fixed costs are applied regardless of the number of requests that the product receives.

You can set a *setup fee*, which is a one-time charge applied upon subscription to the service. It is only charged the first month and it is not applied when an application moves to another application plan.

You can also set a *cost per month* fee, which is a fixed charge applied monthly, regardless of the number of requests. If the subscription happens in the middle of the month, then the fee is prorated.

To access the fixed costs, navigate to the product page of the product that you want to use. Then, click **Applications** > **Application Plans** and choose the application plan that you want to modify.

In the form you can see two fields: `Setup fee` and `Cost per month`.

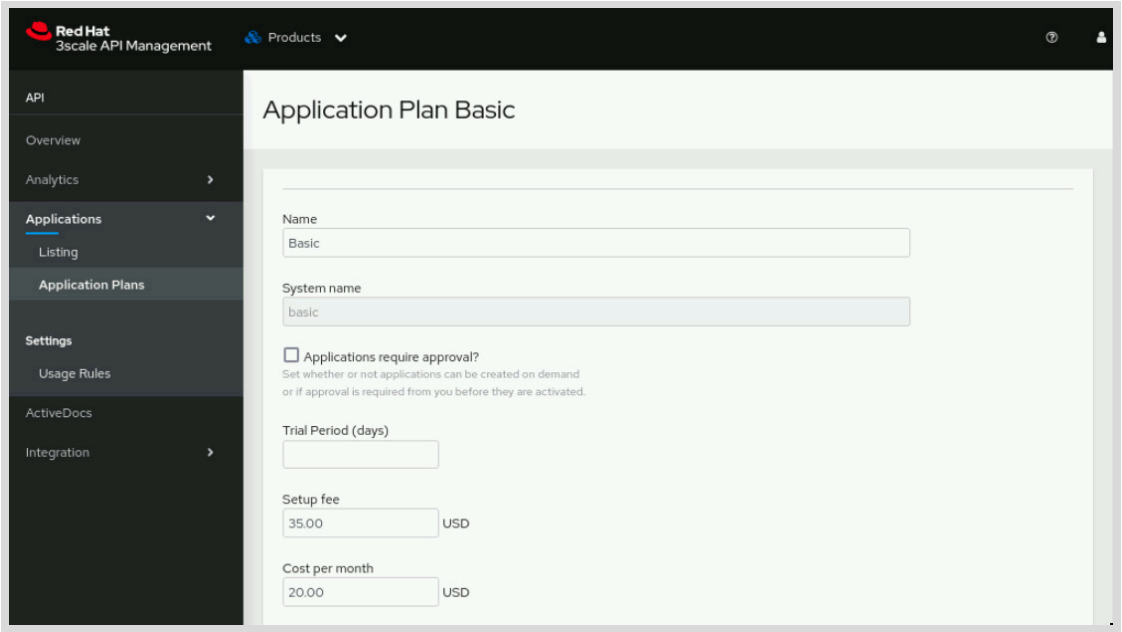


Figure 7.4: Fixed cost fields from a sample product

Describing Variable Costs

Variable costs are costs that depend on the number of requests that every product receives. You can set variable costs at the request level by using *pricing rules*.

You create pricing rules for a metric or a method, and you specify the cost of every request linked with the metric or method.

For example, consider that you have a method mapped to the `/test` path. You create a pricing rule for the method with a cost of 1 USD per request. Therefore, every request to the `/test` path costs 1 USD.

To create a pricing rule, select an application plan first. Then, in the `Metrics, Methods, Limits & Pricing Rules` section, click **Pricing (X)**.

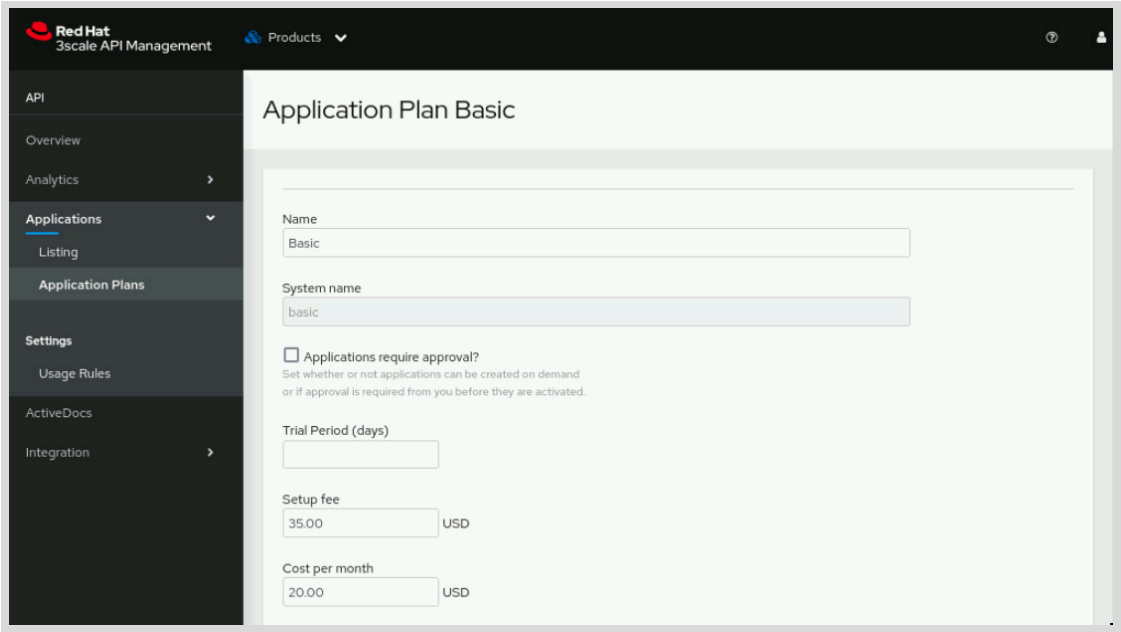


Figure 7.5: Pricing rules from a sample product

You set the pricing for a range of requests. The `From` field sets the lower side of the range, and the `To` field sets the higher side of the range. The `Cost per request` field sets the price of every request in the range.

For example, consider two pricing rules as the following:

```
From = 1
To = 124
Cost per request = 0.5 USD
```

```
From = 125
To = 1000
Cost per request = 0.2 USD
```

In the previous example, from the first to the 124th request, every request is charged at 0.5 USD. Then, from the 125th to the 1000th request, every request is charged at 0.2 USD.

Note

Keep in mind that, because the default `Hits` metric captures all paths (`/`), the pricing rules linked to the `Hits` metrics are applied to all paths.

References

Red Hat 3scale API Management Official Documentation - Pricing

https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.11/html-single/admin_portal_guide/index#pricing

Guided Exercise

Configuring Pricing Rules for API Consumption

In this exercise, you will configure both fixed and variable costs for a product. Because invoices are generated monthly, you will not be able to verify the reports immediately.

Outcomes

You should be able to:

- Configure fixed costs for a product.
- Create pricing rules for a metric.

Before You Begin

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise.

```
[student@workstation ~]$ lab start monetizing-pricing
```

Instructions

1. View the `echo` product, which points to the 3scale Echo API.

   1.1. In the 3scale Admin Portal, navigate to the `Products` page. Click **echo**.

   1.2. In the left pane, click **Applications** > **Application Plans**. The `custom` application plan is displayed.

   1.3. Click **custom** to access the settings page of the plan.

2. Update the `custom` application plan to include a setup fee of 5 USD, and a cost per month of 3 USD.

   2.1. Complete the `Setup fee` field with `5.00`.

   2.2. Complete the `Cost per month` field with `3.00`.

   2.3. Click **Update Application plan** to save the changes.

3. Update the `custom` application plan to charge 0.5 USD per request for the first 120 requests.

   3.1. Click **custom** to access the settings page of the plan.

   3.2. In the `Metrics, Methods, Limits & Pricing Rules` section, click **Pricing (0)** for the `Hits` metric row. The pricing rules are applied to the `Hits` metrics, which captures all paths.

   3.3. Click **new pricing rule** and complete the form:

Field	Value
From	`1`
To	`120`
Cost Per Unit	`0.5`

Then, click **Create pricing rule**.

4. Update the `custom` application plan to charge 0.3 USD per request for the following 1000 requests.

   4.1. Click **new pricing rule** and complete the form:

Field	Value
From	`121`
To	`1121`
Cost Per Unit	`0.3`

Then, click **Create pricing rule**.

5. Update the `custom` application to charge 0.1 USD per request from the 1122th request.

   5.1. Click **new pricing rule** and complete the form:

Field	Value
From	`1122`
To	*(blank)*
Cost Per Unit	`0.1`

Then, click **Create pricing rule**.

Important
To verify that the pricing rules work as expected, you can use the invoice reports generated by 3scale API Management. Because the reports are generated monthly, you can not verify the pricing rules of this guided exercise immediately.

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish monetizing-pricing
```

This concludes the guided exercise.

Quiz

Monetizing APIs with Red Hat 3Scale API Management

Choose the correct answers to the following questions:

1. Which two of the following statements about billing are correct? (Choose two.)

a. Automatic billing is disabled for all accounts by default.

b. In the prepaid billing mode, fixed and set-up costs are billed at the beginning of the month.

c. You can not use Red Hat 3scale API Management to set up a billing payment gateway.

d. You can see your invoices by using the Admin Portal.

2. Which of the following statements about billing is correct? (Choose one.)

a. You can not modify the contents of an invoice in the `Open` state.

b. In the postpaid billing mode, fixed costs from the previous month are billed.

c. An invoice is set to the `failed` state after three failed payment attempts.

d. An invoice in the `Pending` state means that an administrator must take action.

3. Which two of the following statements about pricing rules are correct? (Choose two.)

a. Fixed costs are applied regardless of the number of requests that the product receives.

b. The set-up fee is applied when an application moves to another application plan.

c. If the subscription happens in the middle of the month, then the monthly fee is prorated.

d. You can update the fixed costs in the settings page of a product.

4. Which two of the following statements about pricing rules are correct? (Choose two.)

a. You apply pricing rules to a range of requests. When creating a pricing rule, you select a *from* and a *to* to specify the lower and upper limits of the range.

b. All ranges must have the same cost per request.

c. You can create pricing rules by using the Master Portal.

d. Pricing rules are applied to a metric or a method.

Solution

Monetizing APIs with Red Hat 3Scale API Management

Choose the correct answers to the following questions:

1. Which two of the following statements about billing are correct? (Choose two.)

a. Automatic billing is disabled for all accounts by default.

b. In the prepaid billing mode, fixed and set-up costs are billed at the beginning of the month.

c. You can not use Red Hat 3scale API Management to set up a billing payment gateway.

d. You can see your invoices by using the Admin Portal.

2. Which of the following statements about billing is correct? (Choose one.)

a. You can not modify the contents of an invoice in the `Open` state.

b. In the postpaid billing mode, fixed costs from the previous month are billed.

c. An invoice is set to the `failed` state after three failed payment attempts.

d. An invoice in the `Pending` state means that an administrator must take action.

3. Which two of the following statements about pricing rules are correct? (Choose two.)

a. Fixed costs are applied regardless of the number of requests that the product receives.

b. The set-up fee is applied when an application moves to another application plan.

c. If the subscription happens in the middle of the month, then the monthly fee is prorated.

d. You can update the fixed costs in the settings page of a product.

4. Which two of the following statements about pricing rules are correct? (Choose two.)

a. You apply pricing rules to a range of requests. When creating a pricing rule, you select a *from* and a *to* to specify the lower and upper limits of the range.

b. All ranges must have the same cost per request.

c. You can create pricing rules by using the Master Portal.

d. Pricing rules are applied to a metric or a method.

Summary

In this chapter, you learned:

- Invoices are generated monthly, and 3scale API Management offers two billing modes: prepaid and postpaid.
- It is possible to set up a billing payment gateway for your invoices.
- You can use an application plan to configure fixed and variables costs.
- Pricing rules are applied to a metric or a method.
- When creating a pricing rule, you select a range of requests. A cost per request is selected for every range of requests.