Red Hat 3scale 2-saas

Installing Red Hat 3scale API Management

Install and configure 3scale API Management.

Last Updated: 2023-12-22

Red Hat 3scale 2-saas Installing Red Hat 3scale API Management

Install and configure 3scale API Management.

Legal Notice

Abstract

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

Table of Contents

PREFACE

This guide will help you to install and configure 3scale

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. REGISTRY SERVICE ACCOUNTS FOR 3SCALE

To use container images from **registry.redhat.io** in a shared environment with Red Hat 3scale 2-saas, you must use a *Registry Service Account* instead of an individual user's *Customer Portal* credentials.

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

- Creating a registry service account
- Configuring container registry authentication
- Modifying a registry service account

1.1. CREATING A REGISTRY SERVICE ACCOUNT

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

Procedure

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

NOTE

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

1.2. CONFIGURING CONTAINER REGISTRY AUTHENTICATION

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

Prerequisites

- A Red Hat OpenShift Container Platform (OCP) account with administrator credentials.

- OpenShift **oc** client tool is installed. For more details, see the OpenShift CLI documentation.

Procedure

1. Log into your OpenShift cluster as administrator:

```
$ oc login -u <admin_username>
```

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

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

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

```
$ oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username="customer_portal_username" \
  --docker-password="customer_portal_password" \
  --docker-email="email_address"
```

   You will see the following output:

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

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

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

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

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

Additional resources

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

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

1.3. MODIFYING A REGISTRY SERVICE ACCOUNT

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

WARNING

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

A description for each function is as follows:

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

1.4. ADDITIONAL RESOURCES

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

CHAPTER 2. INSTALLING APICAST

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

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

Prerequisites

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

- A 3scale Hosted account.

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

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

2.1. APICAST DEPLOYMENT OPTIONS

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

- Hosted APIcast: 3scale hosts APIcast in the cloud. In this case, APIcast is already deployed for you and it is limited to 50,000 calls per day.
- **Self-managed APIcast**: You can deploy APIcast wherever you want. Here are a few recommended options to deploy APIcast:
    - Deploying APIcast on the Docker containerized environment : Download a ready to use Docker-formatted container image, which includes all of the dependencies to run APIcast in a Docker-formatted container.
    - Running APIcast on Red Hat OpenShift: Run APIcast on a supported version of OpenShift. You can connect self-managed APIcasts to a 3scale On-premises installation or to a 3scale Hosted (SaaS) account. For this, deploy an APIcast gateway self-managed solution using the operator.

2.2. APICAST ENVIRONMENTS

By default, when you create a 3scale account or create a new API service, you get an APIcast **hosted** in two different environments:

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

- **Production**: Limited to 50,000 calls per day and supports the following out-of-the-box authentication options: API key, and App ID and App key pair, OpenID Connect.

When you use a Self-managed deployment, you still have the same two environments, and you need to deploy an APIcast instance for each. You can specify which configuration (Staging or Production) the APIcast instance will use by setting the environment variable **THREESCALE_DEPLOYMENT_ENV**, which can take values **staging** or **production**.

2.3. CONFIGURING THE INTEGRATION SETTINGS

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

Prerequisites

A 3scale account with administrator privileges.

Procedure

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

2.4. CONFIGURING YOUR PRODUCT

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

This section will guide you through configuring your product:

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

2.4.1. Declaring the API backend

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

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

Procedure

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

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

You will get the following response:

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

2.4.2. Configuring the authentication settings

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

Table 2.1. Optional authentication fields

Field	Description
Auth user key	Set the user key associated with the credentials location.
Credentials location	Define whether credentials are passed as HTTP headers, query parameters or as HTTP basic authentication.
Host Header	Define a custom Host request header. This is required if your API backend only accepts traffic from a specific host.
Secret Token	Used to block direct developer requests to your API backend. Set the value of the header here, and ensure your backend only allows calls with this secret header.

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

Table 2.2. Response codes and default response body

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

2.4.3. Configuring the API test call

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

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

Prerequisites

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

Procedure

1. Promote an APIcast configuration to Staging, by navigating to **[Your_product_name] > Integration > Configuration**.
2. Under *APIcast Configuration*, you will see the mapping rules for each backend added to the product. Click **Promote v.[n] to Staging APIcast**
   - **v.[n]** indicates the version number to be promoted.
3. Once promoted to staging, you can promote to Production. Under *Staging APIcast*, click **Promote v.[n] to Production APIcast**
   - **v.[n]** indicates the version number to be promoted.
4. To test requests to your API in the command line, use the command provided in *Example curl for testing*.
   - The curl command example is based on the first mapping rule in the product.

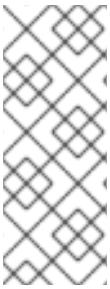
When testing requests to your API, you can modify the mapping rules by adding methods and metrics.

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

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

2.5. DEPLOYING APICAST ON THE DOCKER CONTAINERIZED ENVIRONMENT

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



NOTE

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

- RHEL 7.7
- Docker 1.13.1

Prerequisites

- You must configure APIcast in your 3scale Admin Portal as per Chapter 2, *Installing APIcast*.
- Access to the Red Hat Ecosystem Catalog .
  - To create a registry service account, see Creating and modifying registry service accounts .

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

- Section 2.5.1, “Installing the Docker containerized environment”
- Section 2.5.2, “Running the Docker containerized environment gateway”

2.5.1. Installing the Docker containerized environment

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

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

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

Procedure

1. List all repositories: **sudo yum repolist all**.

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

Additional resources

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

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

2.5.2. Running the Docker containerized environment gateway

IMPORTANT

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

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

Procedure

1. Start the Docker daemon:

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

2. Check if the Docker daemon is running:

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

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

```
$ sudo docker pull registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.13
```

4. Run APIcast in a Docker container engine:

```
$ sudo docker run --name apicast --rm -p 8080:8080 -e
THREESCALE_PORTAL_ENDPOINT=https://<access_token>@<domain>-admin.3scale.net
registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.13
```

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

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

2.5.2.1. The docker command options

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

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

2.5.2.2. Testing APIcast

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

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

NOTE

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

Additional resources

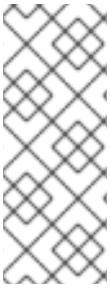
- For more information on available options, see Docker run reference.

2.5.3. Additional resources

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

2.5.4. Deploying APIcast on Podman

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



NOTE

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

- RHEL 8.x/9.x
- Podman 4.2.0/4.1.1

Prerequisites

- You must configure APIcast in your 3scale Admin Portal as per Chapter 2, *Installing APIcast*.
- Access to the Red Hat Ecosystem Catalog .
  - To create a registry service account, see Creating and modifying registry service accounts .

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

- Section 2.5.4.1, “Installing the Podman container environment”
- Section 2.5.4.2, “Running the Podman environment”

2.5.4.1. Installing the Podman container environment

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

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

Procedure

- Install the Podman container environment package:

```
$ sudo dnf install podman
```

Additional resources

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

- Podman Installation Instructions

2.5.4.2. Running the Podman environment

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

Procedure

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

```
$ podman pull registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.13
```

2. Run APIcast in a Podman:

```
$ podman run --name apicast --rm -p 8080:8080 -e
THREESCALE_PORTAL_ENDPOINT=https://<access_token>@<domain>-admin.3scale.net
registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.13
```

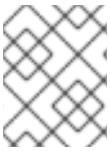
Here, **<access_token>** is the Access Token for the 3scale Account Management API. You can use the Provider Key instead of the access token. **<domain>-admin.3scale.net** is the URL of your 3scale Admin Portal.

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

2.5.4.2.1. Testing APIcast with Podman

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

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



NOTE

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

2.5.4.3. The podman command options

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

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

2.5.4.4. Additional resources

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

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

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

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

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

For more information, see the: APIcast Custom Resource reference

Prerequisites

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

Procedure

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

2.6.1. APICast deployment and configuration options

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

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

See also:

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

2.6.1.1. Providing a 3scale system endpoint

Procedure

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

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

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

Example

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

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

2. Create the OpenShift object for APIcast

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

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

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

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

2.6.1.1.1. Verifying the APIcast gateway is running and available

Procedure

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

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

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

```
$ curl 127.0.0.1:8080/test -H "Host: localhost"
{
  "method": "GET",
  "path": "/test",
  "args": "",
  "body": "",
  "headers": {
    "HTTP_VERSION": "HTTP/1.1",
    "HTTP_HOST": "echo-api.3scale.net",
    "HTTP_ACCEPT": "*/*",
    "HTTP_USER_AGENT": "curl/7.65.3",
    "HTTP_X_REAL_IP": "127.0.0.1",
    "HTTP_X_FORWARDED_FOR": ...
```

```
    "HTTP_X_FORWARDED_HOST": "echo-api.3scale.net",
    "HTTP_X_FORWARDED_PORT": "80",
    "HTTP_X_FORWARDED_PROTO": "http",
    "HTTP_FORWARDED": "for=10.0.101.216;host=echo-api.3scale.net;proto=http"
  },
  "uuid": "603ba118-8f2e-4991-98c0-a9edd061f0f0"
```

2.6.1.1.2. Exposing APIcast externally via a Kubernetes Ingress

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

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

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

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

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

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

Table 2.3. APIcastExposedHost reference table

json/yaml field	Type	Required	Default value	Description
host	string	Yes	N/A	Domain name being routed to the gateway
tls	[]networkv1.Ingress TLS	No	N/A	Array of ingress TLS objects. See more on TLS.

2.6.1.2. Providing a configuration secret

Procedure

1. Create a secret with the configuration file:

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

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

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

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

2. Create an APIcast custom resource:

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

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

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

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

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

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

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

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

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

2.6.1.2.1. Verifying APIcast gateway is running and available

Procedure

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

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

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

2.6.1.3. Injecting custom environments with the APIcast operator

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

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

Prerequisites

- The APIcast operator is installed.

Procedure

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

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

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

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

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

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

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

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

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

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

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

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

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

Next steps

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

2.6.1.4. Injecting custom policies with the APIcast operator

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

- 3scale API

- **Product** custom resource

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

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

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

Next steps

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

Additional resources

- APIcast custom resource definition

2.6.1.5. Configuring OpenTracing with the APIcast operator

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

Prerequisites

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

Procedure

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

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

```
        "hostPort": "127.0.0.1:5778",
        "refreshInterval": 60
    }
    }
type: Opaque
```

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

```
$ oc create -f myjaeger.yaml
```

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

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

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

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

Next steps

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

Additional resource

- APIcast custom resource definition

2.7. ADDITIONAL RESOURCES

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

- Red Hat 3scale API Management Supported Configurations
- Red Hat 3scale API Management - Component Details .
- For the updates on the hosted APIcast version please refer to Red Hat 3scale API Management Platform SaaS Release Notes.

CHAPTER 3. APICAST HOSTED

Once you complete this tutorial, you will have your API fully protected by a secure gateway in the cloud.

APIcast hosted is the best deployment option if you want to launch your API as fast as possible, or if you want to make the minimum infrastructure changes on your side.

Prerequisites

- You have reviewed Chapter 3, *APIcast hosted* for deployment alternatives and decided to use APIcast hosted to integrate your API with 3scale.
- Your API backend service is accessible over the public Internet. Secure communication will be established to prevent users from bypassing the access control gateway.
- You do not expect demand for your API to exceed the limit of 50,000 hits/day. Beyond beyond this, we recommend upgrading to the self-managed gateway.
- Section 3.1, "Deploying your API with APIcast hosted in a staging environment"
- Section 3.2, "Deploying your API with the APIcast hosted into production"

3.1. DEPLOYING YOUR API WITH APICAST HOSTED IN A STAGING ENVIRONMENT

The first step is to configure your API and test it in your staging environment:

Procedure

1. Define the private base URL and its endpoints.
2. Choose the placement of credentials and other configuration details. For more information see the APIcast Hosted documentation.
3. Save the configuration settings by clicking **Update Product**. These settings go through the APIcast staging instance to your API.

You will see a green confirmation message when your configuration is complete.

Before moving on to the next step, make sure that you have configured a secret token to that you backend service validates. You can define the value for the secret token under **Authentication Settings**. This will ensure that nobody can bypass APIcast access control.

3.2. DEPLOYING YOUR API WITH THE APICAST HOSTED INTO PRODUCTION

At this point, you are ready to take your API configuration to a production environment. To deploy your 3scale Hosted APIcast instance:

Procedure

1. Go back to the **Integration and Configuration** page and click on the **'Promote to v.x to Production'** button.

2. Repeat this step to promote further changes in your staging environment to your production environment.

It will take between 5 and 7 minutes for your configuration to deploy and propagate to all the cloud APIcast instances. During redeployment, your API will not experience any downtime. API calls may return different responses depending on which instance serves the call. Deployment has been successful when the box around your production environment has turned green.

Both the staging and production APIcast instances have base URLs on the **apicast.io** domain. You can tell them apart because the staging environment URLs have a staging subdomain. For example:

- staging: https://api-2445581448324.staging.apicast.io:443
- production: https://api-2445581448324.apicast.io:443

3.3. ADDITIONAL INFORMATION

- 50,000 hits/day is the maximum allowed for your API through the APIcast production cloud instance. You can check your API usage in the Analytics section of your Admin Portal.
- There is a hard throttle limit of 20 hits/second on any spike in API traffic.
- Above the throttle limit, APIcast returns a response code of **403**. This is the same as the default for an application over rate limits. If you want to differentiate the errors, please check the response body.