Contents

[Installing 3Scale on Azure OpenShift 2](#_Toc35950600)

[2.1 System requirements: 2](#_Toc35950601)

[2.2 Configuring nodes: 3](#_Toc35950602)

[2.3 Deploying 3Scale 3](#_Toc35950603)

[PVC 4](#_Toc35950604)

[Install APICast on OpenShift 6](#_Toc35950605)

[Managing 7](#_Toc35950606)

[Getting started 7](#_Toc35950607)

[Adding you app 8](#_Toc35950608)

[Useful Parameters: 10](#_Toc35950609)

[Testing 10](#_Toc35950610)

[Scenarios performed 10](#_Toc35950611)

[Support environment 10](#_Toc35950612)

# Installing 3Scale on Azure OpenShift

The document is based on the following guide with some notes. It should be read while performing the corresponding steps.

<https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.7/html/installing_3scale/install-threescale-on-openshift-guide#system-requirements-for-installing-threescale-on-openshift>

Tested with 3Scale 2.7 and OpenShift 4.3.

## 2.1 System requirements:

As noted in the guide, 3Scale needed 4 PVs: 3 ReadWriteOnce and 1 ReadWriteMany. When creating OpenShift on Azure, a default Storage Class by the name “Manage Premium” is created. This storage class is based on azure-disk, which provides only ReadWriteOnce ability. In addition, the class has the following property:

volumeBindingMode: WaitForFirstConsumer.

From Kubernetes documentation:

*The volumeBindingMode field controls when*[*volume binding and dynamic provisioning*](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#provisioning)*should occur.*

*By default, the Immediate mode indicates that volume binding and dynamic provisioning occurs once the PersistentVolumeClaim is created. For storage backends that are topology-constrained and not globally accessible from all Nodes in the cluster, PersistentVolumes will be bound or provisioned without knowledge of the Pod’s scheduling requirements. This may result in unschedulable Pods.*

*A cluster administrator can address this issue by specifying the WaitForFirstConsumer mode which will delay the binding and provisioning of a PersistentVolume until a Pod using the PersistentVolumeClaim is created. PersistentVolumes will be selected or provisioned conforming to the topology that is specified by the Pod’s scheduling constraints. These include, but are not limited to,*[*resource requirements*](https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container)*,*[*node selectors*](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#nodeselector)*,*[*pod affinity and anti-affinity*](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity)*, and*[*taints and tolerations*](https://kubernetes.io/docs/concepts/configuration/taint-and-toleration)*.*

All of the PVCs created during 3Scale’s deployment are using the default storage class. In my case, I encountered a problem when deploying 3Scale. All the of PVC were stuck at pending state.

The PVC for RWX (ReadWriteMany) was understandably stuck, because the storage class couldn’t provide that capability. The 3 RWO (ReadWriteOnce) PVC were what puzzled me.

As far as I understand, this happened because of a kind of dead-lock – the PVC was waiting for the pods to be created (because of WaitForFirstConsumer binding mode) and the pod was waiting for the PVC.

To change that, I create a new StorageClass, named RWOclass, and changed the following parameter:

**volumeBindingMode: Immediate**

In addition, I created another class called RWXclass, to address the issues of the RWX PVC (called system-storage). To do that, follow those guides:

Create a file share in azure: <https://docs.microsoft.com/en-us/azure/storage/files/storage-how-to-create-file-share?tabs=azure-portal>

Create a PV that references the share: <https://docs.openshift.com/container-platform/4.3/storage/persistent_storage/persistent-storage-azure-file.html>

Don’t create the PVC yet, stop after create the PV. You can use 100Mi as your size (that what the System-Storage PVC needs).

## 2.2 Configuring nodes:

The information in the guide (as of March 20) is not relevant for Openshift 4, where the installation is different. To get the amp.yml file, just download it from here: <https://github.com/3scale/3scale-amp-openshift-templates/blob/master/amp/amp.yml>.

Place is someplace with oc available.

## 2.3 Deploying 3Scale

Notice that step 2.3.2 should be performed before 2.3.1.

After creating the needed service account and 3Scale project, you are ready to deploy 3Scale:

*oc new-app --file amp.yml --param WILDCARD\_DOMAIN=apps.OPENSHIFT-DOMAIN.cloudlet-dev.com.*

Take notice that the wildcard domain is not your OpenShift domain, but starts with “apps.”. Otherwise, the routes the 3Scale deployment will create will not be exposed through the DNS rules the OpenShift installation created.

The first thing I encountered was that the pods were waiting for the images to be pulled. I couldn’t figure out why that didn’t happen automatically, so I pushed them manually:

*oc import-image istag/amp-apicast:latest*

*oc import-image istag/amp-backend:latest*

*oc import-image istag/amp-zync:latest*

*oc import-image istag/zync-database-postgresql*

*oc import-image istag/amp-system:latest*

*oc import-image istag/system-redis:latest*

*oc import-image istag/system-mysql:latest*

*oc import-image istag/system-memcached:latest*

*oc import-image istag/backend-redis:latest*

Then the Pods started to run successfully, and half of the 14 needed ones deployed.

### PVC

Now we are back to the PVC issues. If the 3 RWO PVCs are in state “Bound” and not “Pending”, than everything’s alright. If not, you need to edit the PVCs to use the new storage class (RWOclass) that we created previously.

To do that, use:

oc edit pvc PVC-NAME

Copy the yaml file and create a new file with the same content (lets call it PVC-NAME.yml). Then, edit the file and change the StorageClass to RWOclass.

Delete the old pvc using:

oc delete pvc PVC-NAME

(If the pvc is stuck at “terminating”, I used the following command to solve it:

*kubectl patch pvc system-storage -p '{"metadata":{"finalizers": []}}' --type=merge*)

than recreate it:

oc apply -f PVC-NAME.yml

An example of one the PVC I edited (mysql-storage.yml)

*apiVersion: v1*

*kind: PersistentVolumeClaim*

*metadata:*

*annotations:*

*openshift.io/generated-by: OpenShiftNewApp*

*creationTimestamp: "2020-03-16T06:55:55Z"*

*finalizers:*

*- kubernetes.io/pvc-protection*

*labels:*

*app: 3scale-api-management*

*threescale\_component: system*

*threescale\_component\_element: mysql*

*name: mysql-storage*

*namespace: 3scale*

*resourceVersion: "151349"*

*selfLink: /api/v1/namespaces/3scale/persistentvolumeclaims/mysql-storage*

*uid: cdd25149-b9b6-45fb-bd44-ea453f2e62ce*

*spec:*

*accessModes:*

*- ReadWriteOnce*

*resources:*

*requests:*

*storage: 1Gi*

***storageClassName: RWOclass***

*volumeMode: Filesystem*

If you followed the steps successfully, 3 of the 4 PVCs should be now in “Bound” state.

To fix the 4th PVC, follow the same steps:

*oc edit pvc system-storage*

*vi system-storage.yml*

edit the content to use the PV and the RWXclass we created earlier. It should look like this:

*apiVersion: v1*

*kind: PersistentVolumeClaim*

*metadata:*

*….*

*spec:*

*accessModes:*

*- ReadWriteMany*

*resources:*

*requests:*

*storage: 1Gi*

***storageClassName: RWXclass***

***volumeName: "pv-rwx"***

Then run:

*oc delete pvc system-storage*

*oc apply -f system-storage.yml*

Wait around 10 minutes, and 3Scale should be running!

# Install APICast on OpenShift

Follow this guide:

<https://access.redhat.com/documentation/en-us/red_hat_3scale_api_management/2.7/html/installing_3scale/install-threescale-on-openshift-guide#using-apicast-with-threescale-on-openshift>

The section “Connecting APIcast from a different OpenShift cluster” is the relevant one, with some corrections.

1. Creating the token: After creating the token as shown in the steps, first check if it works using the following command:

*curl -v -k "https://3scale-admin.apps.anton-openshift.cloudlet-dev.com/admin/api/services.json?access\_token=TOKEN*

If you get *“{"error":"Your access token does not have the correct permissions"}”,* it means your token is incorrect.

First, check here of the token exists:

<https://3scale-admin.apps.anton-openshift.cloudlet-dev.com/p/admin/user/access_tokens> (replace the address and the user name with the relevant ones).

If it doesn’t it may be that you created the token in a different places (for example in the master api, for managing tenants).

1. After you verified the token works correctly, create a secret called **apicast-configuration-url-secret**. The apicast will use this secret to authenticate to the main 3Scale.

oc create secret generic apicast-configuration-url-secret --from-literal=password=https://TOKEN@3scale-admin.apps.anton-openshift.cloudlet-dev.com \

--type=kubernetes.io/basic-auth

1. Download the API-cast yml from here, or install directly:

*oc new-app -f https://raw.githubusercontent.com/3scale/apicast/master/openshift/apicast-template.yml*

1. In the Workloads->Deployment configs, change the following parameters of the apicast config:

APICAST\_CONFIGURATION\_CACHE: 60

APICAST\_PATH\_ROUTING: true

Read [here](#_Useful_Parameters:) for explanation

# Managing

## Getting started

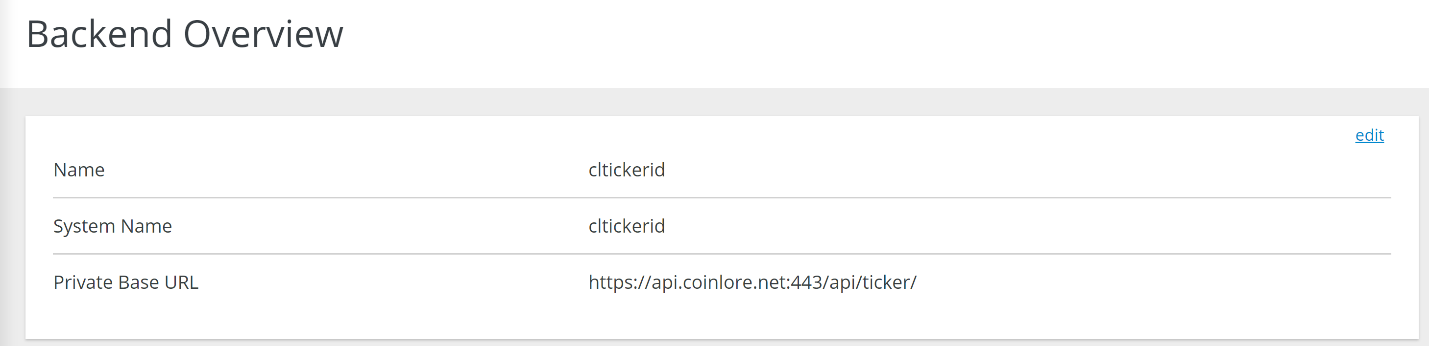
A simple setup of an api can be performed by the wizard here:

<https://3scale-admin.apps.anton-openshift.cloudlet-dev.com/p/admin/onboarding/wizard/intro>

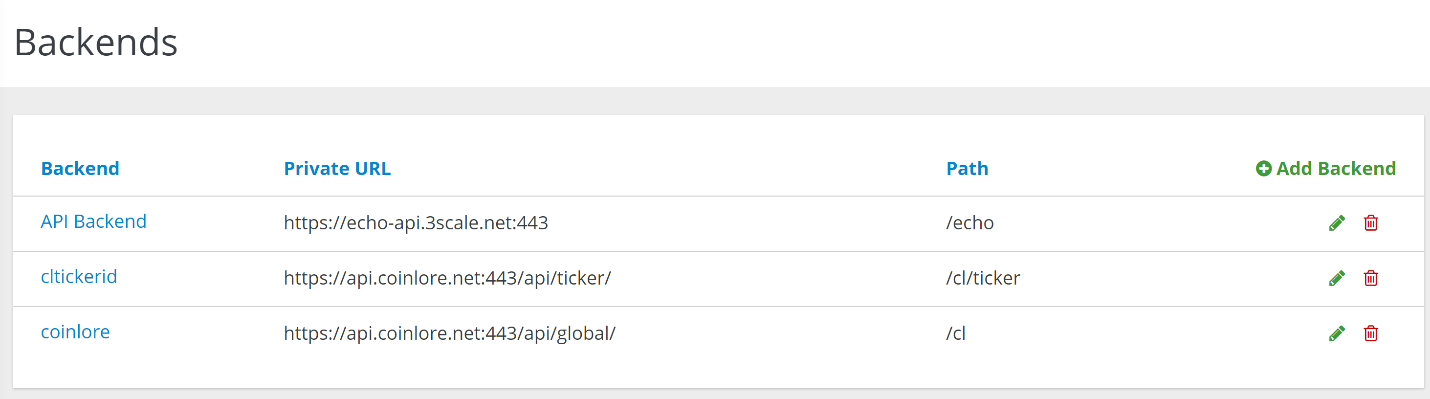
Backend: a collection of internal APIs from your application. containing at least the URL of your API. The backend can optionally have mapping rules, methods and metrics.

Products: Customer facing APIs, that expose your backends. It’s where you define the application plans, and configure APIcast (GW). Can have one or more backends.

For example, if we create a backend like this one:



And add the backend to our product, with the path: /cl/ticker as shown:



After publishing the API in the staging and prod environments (in the configuration tab, in integration menu of your product), we will be able to access the API using the following link:

<https://api-3scale-apicast-staging.apps.anton-openshift.cloudlet-dev.com/cl/ticker/?app_id=b68664e6&app_key=e99a268a04b216aed27da00de0f86071&id=91>

as you can see, I access the general path of the staging API-GW, then the path for the backend, and in the end I can add parameters. The API-GW will pass the query to the private base URL, with the additional parameters.

## Adding you app

The following steps outline how to add your own app, after you already have an APICast (GW) deployed in your OpenShift.

1. Create a project for the app, and deploy it

*oc new-project demo*

*oc new-app --name=demo https://github.com/wicksy/openshift-demo-app*

1. Get the address of the service you want to expose.

You can use:

*oc get svc*

For example: 172.30.186.220:8080 (the port is the one you expose the app on)

1. Create a network policy that allows only the 3scale GW to access your project. As a prerequisite, you need to assign the 3scale project (if not assigned before) a label that you can filter by.

*oc label namespace 3scale-app project=3scale-app*

1. Create a file with the following network policy:

*kind: NetworkPolicy*

*apiVersion: networking.k8s.io/v1*

*metadata:*

*name: allow-from-apicast*

*spec:*

*podSelector: {}*

*ingress:*

*- from:*

*- namespaceSelector:*

*matchLabels:*

***project: 3scale-app***

*podSelector:*

*matchLabels:*

***deploymentconfig: apicast***

And apply the network policy.

*oc apply -f FILENAME*

After applying, only the API GW will be able to talk to your application. If you want your pods to be able to talk with each other, you need to add an additional policy.

For example:

*kind: NetworkPolicy*

*apiVersion: networking.k8s.io/v1*

*metadata:*

*name: allow-same-namespace*

*spec:*

*podSelector:*

*ingress:*

*- from:*

*- podSelector: {}*

1. Now you are finished with OpenShift, and you can move to 3Scale UI.

In your main dashboard, in the “Backends” tab, create a backend. Name it as you want, and in the Private Endpoint enter the service address you got earlier.

1. Go back you the dashboard, and from the product tab create a product.
2. In the integration->backends menu, add the backend you created with the desired path.
3. In the Application->applications plan menu, create an application plan to allow applications to subscribe to your product.
4. In the Audience, click the Group you want to create an application for. Click on “Applications”, and create a new one. Make sure to subscribe to the application plan in the product you created earlier.
5. Go to integration->settings, and change the deployment option to “APICast self-managed”.

Enter the url of your API GW.

1. Now you need to add a mapping rule. To understand why:

We set the parameter “APICAST\_PATH\_ROUTING” (See [here](#_Useful_Parameters:)) in our API GW to true, to allow using the same address for different products (because different apps share the same API).

To know which product (or service, sometimes called) will answer the request, 3Scale go over all the mapping rules and find one that matches. That means the if you leave the default “/” mapping, all the request will go to the first product.

To successfully enable the path routing, you need to match a mapping rule for each backend you have, or follow a convention (for example - /ztube/\* for ztube product, /startrack/\* for startrack product). Make sure there are no conflicts between mapping rules, otherwise you’ll have chaos.

In your case, we will create a mapping rule for /demo, to count hits (doesn’t really matter which metric the mapping rule is applied to).

1. Now go to integration->configuration, and promote your configuration.

<https://developers.redhat.com/blog/2019/07/29/3scale-toolbox-deploy-an-api-from-the-cli/>

# Useful Parameters:

APICAST\_CONFIGURATION\_CACHE

Specifies the interval (in seconds) that the configuration will be stored for. The value should be set to 0 (not compatible with boot value of APICAST\_CONFIGURATION\_LOADER) or more than 60. For example, if APICAST\_CONFIGURATION\_CACHE is set to 120, the gateway will reload the configuration from the API manager every 2 minutes (120 seconds). A value < 0 disables reloading.

APICAST\_PATH\_ROUTING

When this parameter is set to *true*, the gateway will use path-based routing in addition to the default host-based routing. The API request will be routed to the first service that has a matching mapping rule, from the list of services for which the value of the Host header of the request matches the *Public Base URL*

# Testing

## Scenarios performed

1. Create a self-managed APICast and expose an API from a private project using Network Policy.
2. Shutdown the main 3Scale and see that the GW works.

## Support environment

Per RedHat’s response:

*“3scale API Management Hosted with APIcast API Gateway self-managed is being tested and supported with the latest two versions of APIcast API Gateway made available as part of the latest 3scale releases. Bug fixes and Security fixes will be provided on the latest release of APIcast, and only security fixes will be provided on the previous release.”*

**Take note** that 3Scale 2.7 is officially supported only up to OpenShift 4.2.

Sources:

<https://access.redhat.com/articles/2798521>

<https://access.redhat.com/articles/2798521#3scale-api-management-27-2>