Red Hat 3Scale 2.0

API Devops

For Use with Red Hat 3Scale 2.0

Last Updated: 2018-04-08

Red Hat 3Scale 2.0 API Devops

For Use with Red Hat 3Scale 2.0

Legal Notice

Abstract

This guide documents development operations with Red Hat 3Scale 2.0.

Table of Contents

CHAPTER 1. DIFFERENTIATE API ENVIRONMENTS

By the end of this tutorial, you will be able to set up 3scale to differentiate between your production and staging environments.

In order to support a formal development lifecycle for your developer community, you may want to provide separate staging and production environments to access your API. So during development and testing, the API may be used without the adverse consequences of operating in a production environment. This is different from managing the dev, test, stage, deploy cycle of the API itself.

1.1. OVERVIEW

There are several options to provide differentiated staging and production environments for your developer community. 3scale supports a lot of flexibility on how to implement this. Once you decide on which approach is right for you, you can implement this within the NGINX gateway as a custom extension within the config files.

This tutorial describes two ways of providing differentiated environments:

- Staging restricts access based on rate limits or methods with single endpoint
- Application plan determines API call routing to staging or production environment

1.2. OPTION 1: STAGING RESTRICTS ACCESS BASED ON RATE LIMITS OR METHODS

This option is simple to set up and simple to use. The main limitations are that both environments share the same backend, and in reporting views, the production and staging traffic is mixed together.

For this option, you would create one Application plan for each environment and set the rate limits and availability of methods/metrics depending on the environment (see setting rate limits). For example in the staging mode to provide lower rate limits and optionally restrict access to any methods that are not desirable for the staging, for example expensive resources, or write/delete methods.

On your integration side, you would have to implement the mapping to the corresponding 3scale methods. Remember that this is only simulating environments and not hitting two different endpoints.

For example assuming there is a developer app under the staging plan which is restricted from "write" calls, this is the auth call to 3scale representing the mapping for POST to /words…..:

```
curl -v -X GET "http://su1.3scale.net/transactions/authrep.xml?
provider_key=PROVIDER_KEY&app_id=APP_ID&app_key=APP_KEY&usage[words-
write]=1"
```

The response will be 409 with the body:

```
<status>
  <authorized>false</authorized>
  ..
```

Whenever desired the plan can be upgraded from staging to production without coding changes by the developer:

1. A self-service plan change in the developer portal.

2. Request to the API provider to make the plan change.

3. Plan change determinded unilaterally by the API provider.

This will depend on your service settings.

1.3. OPTION 2: APPLICATION PLAN DETERMINES API CALL ROUTING TO STAGING OR PRODUCTION ENVIRONMENT

This Option only refers for the downloadable Nginx configuration files.

This option allows differentiation of the API backend for each environment. The operational use is just as simple as option 1. The main difference is that the implementation is slightly more complicated (requiring custom modifications to the NGINX config files). Due to the need for NGINX to parse the response bodies, there will also be a performance hit.

In this scenario, the backend provides different response logic for the two modes, but the developer should not have to make any coding changes to switch between the two environments. 3scale achieves this by using the NGINX gateway to route calls based on the authorization response from 3scale indicating whether production calls are enabled or disabled in the respective application plan. For example, when an app under the staging plan makes a call, the gateway does the auth request to 3scale not knowing if this is a call to staging or production. The call might look like this:

```
curl -v -X GET "http://su1.3scale.net/transactions/authrep.xml?
provider_key=PROVIDER_KEY&app_id=APP_ID&app_key=APP_KEY&usage[words]=1"
```

and the response

```
<authorized>true</authorized>
  <plan>Sandbox</plan>
...
```

And the response is parsed for 'plan' to determine whether to route the call to the staging or production backend:

The next steps assume you have two environments called exactly "Staging" and "Production". If you want to use different names, define them appropriately in the **API** section of your Admin Portal and verify the returned value from the authorization call.

On the NGINX side, you'll have to apply some modifications to the configuration files generated by 3scale. First, define a new upstream.

```
upstream production_upstream {
  server production-environment-url-with-port max_fails=5 fail_timeout=30;
}
upstream sandbox_upstream {
  server sandbox-environment-url-with-port max_fails=5 fail_timeout=30;
}
```

Next, assign the server name for your services (not obligatory if you have only one server).

```
listen 80;
    ## CHANGE YOUR SERVER_NAME TO YOUR CUSTOM DOMAIN OR LEAVE IT BLANK IF
ONLY HAVE ONE
```

```
    server_name YOUR-SERVICE-DOMAIN-FOR-NGINX;
```

Where YOUR-SERVICE-DOMAIN-FOR-NGINX will be the domain(s) assigned to the server where NGINX is hosted. Having this we will have to specify the .lua file path on your server in the 'location: /' part of the config file:

```
## CHANGE THE PATH TO POINT TO THE RIGHT FILE ON YOUR FILESYSTEM
access_by_lua_file LUA-SYSTEM-PATH;
```

With the .conf file customization finished, you can pass to the .lua file, where the logic responsible for conditional proxy pass resides. Find the line starting with **`function authrep(params, service)`** and inside that function definition apply the following changes:

- Comment out lines like here:

```
-- if is_known ~= 200 then
  local res = ngx.location.capture("/threescale_authrep", { share_all_vars
= true })

      --    -- IN HERE YOU DEFINE THE ERROR IF CREDENTIALS ARE PASSED, BUT
THEY ARE NOT VALID
  --    if res.status ~= 200 then
  --       -- remove the key, if it's not 200 let's go the slow route, to
3scale's backend
  --       api_keys:delete(ngx.var.cached_key)
  --       ngx.status = res.status
  --       ngx.header.content_type = "application/json"
  --       error_authorization_failed(service)
  --    else
  --               api_keys:set(ngx.var.cached_key,200)
  --    end

      --         ngx.var.cached_key = nil
  -- end
```

Note that the second line is NOT COMMENTED.

- Just after these commented lines, add the following code:

```
local s = ngx.re.match(res.body,[=[<plan>Sandbox]=])
local p = ngx.re.match(res.body,[=[<plan>Production]=])

    if p then
  ngx.var.proxy_pass = "http://production_upstream"
elseif s then
  ngx.var.proxy_pass = "http://sandbox_upstream"
end
```

This code uses a regular expression to match the response with the plan's name. If you want to experiment more with regular expressions try Rubular.

Now the calls registering usage in certain environments will automatically hit these environments without any additional changes on the developer's side. As soon as the application plan is switched from staging to production or vice versa, the API calls will automatically be re-routed to the correct backend.

1.3.1. Note

If any of your environments reside on a hosted server instance (e.g. Heroku, Google app, etc.) you will have to do a host name rewrite to send a proper host name in the headers. To do this, add the following lines:

- In .conf file under the location: / part add **`set $host null;`**
- In .lua file in the code added by you add: **`ngx.var.host = "STAGING-HOSTNAME"`** in the **`if plan == "Staging"`** condition (STAGING-HOSTNAME can be e.g. 'application.herokuapp.com')**`ngx.var.host = "PRODUCTION-HOSTNAME"`** in the **`if plan == "Production"`** condition.

CHAPTER 2. PRODUCTION TIPS

This document refers to the downloadable Nginx configuration files. As such, most of these tips will not apply to the latest version of APIcast.

If you are running our NGINX-based API gateway in a production environment, we've collected these best practice for commonly asked questions.

2.1. LOAD BALANCING WITH THE 3SCALE API GATEWAY

The 3scale API gateway is based on the high performance NGINX gateway. A single instance is able to handle high enough API traffic volumes to meet most customers' needs. However, we recommend that production environments have multiple instances of the gateway in parallel. This will avoid having a single point of failure on the API gateway layer, and it will also provision extra capacity to handle potential traffic spikes.

The 3scale API gateway is designed to make it really easy to set up a load-balanced environment. It is completely stateless, reaching to the 3scale backend service to perform all authorization tasks.

If you're currently operating with a single API gateway and you're looking to set up multiple instances in parallel, all you need to do is:

- Deploy as many instances as you want following the instructions here
- Download your NGINX configuration files from 3scale.
- Use the same set of files for all your gateways.
- Make sure that the **`server_name`** is the same for all of them (this will be the public domain of your API, which will also be the domain that resolves to your load balancer in front of the gateways).

2.2. CORRECTLY CONFIGURE DNS RESOLUTION IN NGINX

This is useful to know if you are using DNS resolution for load balancing on your API backend. A typical example of this is if you're using AWS Elastic Load Balancing, which will return multiple different IP addresses when clients perform a DNS resolution.

By default, NGINX resolves the domain names of your backend servers only when it is started. It caches the IP and uses that when proxying incoming API requests. This will be a problem in the scenario discussed above for two reasons:

- It will send all the traffic to a single IP, effectively disabling the DNS load balancing.
- The cached IP might not exist anymore because your backend might have scaled down automatically removing some IP addresses from the pool.

There is an easy fix for this: forcing NGINX to resolve the domain of the backend servers at runtime. This solution requires using the **`resolver`** directive to specify a DNS server:

```
resolver 8.8.8.8;
```

That line uses Google's Public DNS servers, but you can configure any other DNS servers, including your own in case you want to resolve private domain names.

The **`resolver`** directive has many useful options, which you can learn about in the official documentation.

2.3. BYPASSING THE AUTHORIZATION STEP IN CASE OF NETWORK FAILURE

The 3scale Service Management API is the service that responds to authorization requests sent by the API gateway. The availability of this service is our top priority, and it has a very good track record of uptime. Very rarely, there may be external circumstances that can cause your API gateway to be unable to reach the 3scale Service Management API (such as a problem in the network or in a corporate firewall). The default behavior of the API gateway in case the authorization request fails is to deny the incoming API call in order to prevent a potential security breach. However, this behavior can be customized to fit your requirements.

For example, you can deny incoming API calls from all users except those that come from a whitelist of mission critical applications. You can implement this behavior in your configuration by changing the **`authrep`** function in your Lua file to match the one in this code snippet. You should also create a **`whitelist.lua`** file with the list of **`app_id`** whose calls should be allowed through. This file should be placed in the same directory as the other NGINX configuration files.

2.3.1. Status updates

We notify of any problems in our service as soon as they happen. In order to get timely status updates, please follow @3scalestatus on Twitter.

CHAPTER 3. USING CHEF WITH YOUR API GATEWAY

This document refers to the Chef Integration for the downloadable Nginx configuration files. This is currently not supported in the latest version of APIcast.

This tutorial describes how to use the official 3scale Chef Cookbook to automate the deployment of your API gateway.

The 3scale Chef Cookbook allows any Chef user to automate the deployment of the 3scale API gateway. Running the Cookbook on one or multiple target nodes will install OpenResty, plus all the necessary system dependencies required to run it. After the execution completes, the nodes will have an up-and-running gateway listening for incoming API requests.

The Cookbook not only installs the API gateway, but it will also deploy your 3scale NGINX configuration files, specifically tailored for your API configuration, to the exact location they are needed.

Chef is a configuration management tool that automates and simplifies software installation by using reusable configuration scripts called Cookbooks. The 3scale API gateway is one of the integration methods that our customers use to integrate their APIs with the 3scale API Management Platform. It's based on OpenResty, a bundle that includes NGINX and some very useful third-party modules that complement it with features such as support for Lua scripting.

3.1. PREREQUISITES

This tutorial assumes familiarity with how Chef works and a ready-to-use Chef environment. Here are some resources that will help you get to that point:

- Get started with Chef
- An introduction to Chef solo

You'll also need to have previously configured your API in your 3scale Admin Portal. If you haven't gone through that step yet, you should do it now. You can follow the instructions here (stop at the part about running your gateway self-managed).

3.2. HOW TO USE IT

3.2.1. 1. Add the Cookbook to your run list

The first step is to add the default recipe of the Cookbook to your node or role run list.

3.2.2. 2. Configure the required Cookbook attributes

There are four attributes to set to configure the Cookbook. All of them are under the 3scale namespace: `3scale` namespace.

- **['3scale']['config-source']** – Where your NGINX configuration files will be taken from. Two options: "local" or "3scale". Read the "Applying your own 3scale configuration" section of this tutorial before setting this attribute.
- **['3scale']['provider-key']** – The key that identifies you as a 3scale customer. It can be found in the "Account" menu of your 3scale admin portal.

- **['3scale']['admin-domain']** – If your 3scale Admin Portal domain is mycompany-admin.3scale.net, then the value of this attribute should be "mycompany".
- **['3scale']['config-version']** – Version ID. If not included, the current configuration from your 3scale account will be used. If included, the value must be a timestamp of one deployment, formatted as: 2015-09-15-041532. See the "Rollback process" section for more information on this.

Here you can see the default value each one of this attributes has. This Cookbook uses and depends on the OpenResty Cookbook, so attributes of that Cookbook are also available to you. You can see a full list here.

Since you'll be using the NGINX configuration files that 3scale generates for you, you won't be able to use the attributes of the OpenResty Cookbook related to configuration parameters that go in the nginx.conf file.

Here is an example of a JSON node description that's ready to be used. In this case, the configuration files will be downloaded from 3scale – as you can see in the **`config-source`** attribute :

3.3. APPLYING YOUR OWN 3SCALE CONFIGURATION

For the API gateway to be configured for your own API endpoints, you need to deploy it using your own set of NGINX configuration files. There are two ways to apply your own configuration files to the Cookbook:

3.3.1. Option 1: Local configuration files

This is the option you should use if:

- Your NGINX configuration has any customization on top of the default files generated by 3scale.
- You have more than one service in 3scale (since you will need to set the domains for each of them in the configuration).

Configure your API in 3scale using the self-managed gateway option. Click on Download the Nginx Config files at the bottom of the screen. Then drop these files on the **`/files/default/config/`** directory of the Cookbook.

To use this option you'll need to set the **`['3scale']['config-source']`** attribute to local in your node or role description.

3.3.2. Option 2: Fetch configuration files from your 3scale account

With this option, the Cookbook will automatically fetch the NGINX configuration files from your 3scale account when running the deployment. You'll need to set the following attributes in your node or role description:

- **`['3scale']['config-source']`** = "3scale"
- **`['3scale']['provider-key']`** = *(see attributes section)*
- **`['3scale']['admin-domain']`** = *(see attributes section)*

In both cases, the NGINX configuration files will be copied to a subdirectory in **`/var/chef/cache`** and symlinked to the NGINX working directory (**`/etc/nginx/`**).

3.4. ROLLING BACK TO A PREVIOUS VERSION OF THE CONFIGURATION

The 3scale Cookbook allows rolling back to a previously deployed version of the configuration. This can be used in cases where you have a node where the API gateway had already been deployed one or multiple times, and you want to deploy it again but using the configuration files from one of the previous deployments instead of the latest version.

The built-in way to roll back is by using the **`['3scale']['config-version']`** attribute. Here's an example of a full node description using the rollback attribute:

3.4.1. Troubleshooting

If you're having problems deploying your API gateway when running the Cookbook, the best first step is to look at Chef's own logs. You can find some useful debugging tips here.

If the deployment completed successfully but the API gateway is not running as expected, the problem is probably in the NGINX configuration files you deployed. The best place to start troubleshooting is the NGINX error log, located at **`/var/log/nginx/error.log`**

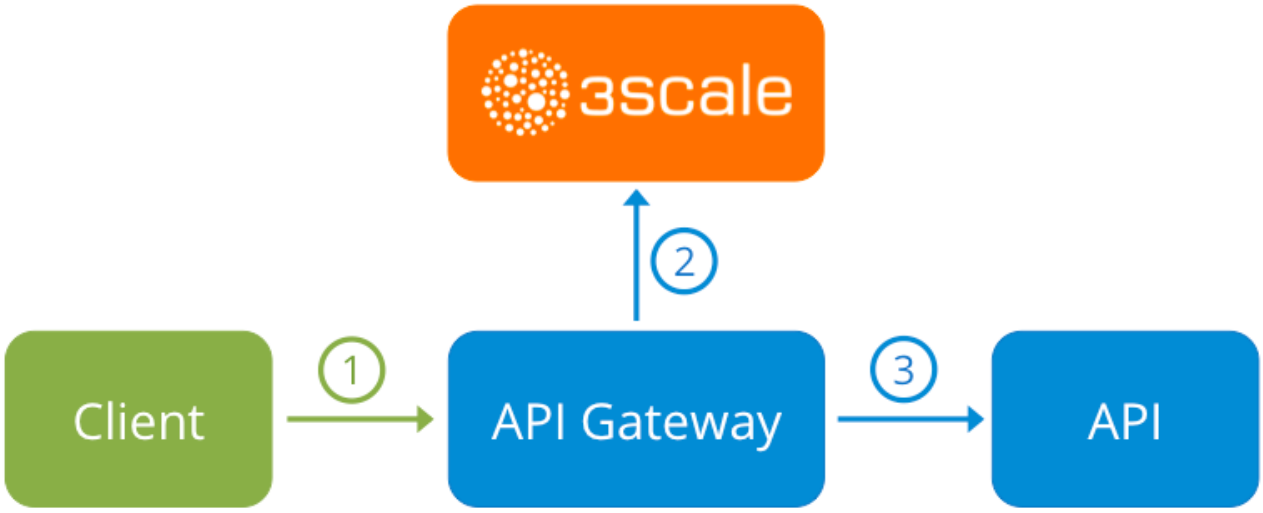
If there are no errors in the NGINX log, you might want to double-check how you configured your API in 3scale. Check out this debugging guide.

CHAPTER 4. TROUBLESHOOTING

When things go wrong with your API, it can sometimes be difficult to identify where a problem lies. The following guide aims to help you identify and fix the cause of issues with your API infrastructure.

API Infrastructure can be a lengthy and complex topic. However, at a minimum you will have 3 moving parts in your Infrastructure:

1. The API gateway
2. 3scale
3. Your API



Errors between any of those three elements will result in your clients being unable to access your API. However, it won't always be clear exactly which component caused the failure. This guide aims to give you some tips to troubleshoot your infrastructure in order to identify where things may be going wrong.

We will start with some common scenarios first, before moving on to more sophisticated troubleshooting for when it's not clear exactly where the problem lies.

4.1. COMMON ISSUES

There are a number of symptoms that can point to some very common issues with your integration with 3scale. These will vary depending on whether you are at the beginning of your API project, setting up your infrastructure, or are already live in production.

4.1.1. Integration issues

The following sections attempt to outline some common issues you may encounter during the initial phases of your integration with 3scale: at the beginning using APIcast and prior to go live running the self-managed API Gateway.

4.1.1.1. APIcast

When you're first integrating your API with APIcast on the Service Integration screen, you might get some of the following errors shown on the page or returned by the test call you make to check for a successful integration.

- **`Test request failed: execution expired`**
  Check that your API is reachable from the public internet. APIcast cannot be used with private APIs. If you're not comfortable about making your API publicly available in order to integrate with APIcast, you can always set up a private secret between APIcast and your API to reject any calls not coming from the API gateway.

- **`The accepted format is 'protocol://address(:port)'`**
  Remove any paths from the end of your API's private base URL. You can add these in the "mapping rules" pattern or at the beginning of the "API test GET request".

- **`Test request failed with HTTP code XXX`**

  - **`405`**: Check that the endpoint accepts GET requests. APIcast only supports GET requests to test the integration.

  - **`403: Authentication parameters missing`**: If your API already has some authentication in place, APIcast will not be able to make a test request.

  - **`403: Authentication failed`**: If this is not the first service you've created with 3scale, check that you have created an application under the service with credentials in order to make the test request. If it is the first service you're integrating, check that you haven't deleted the test account/application that is created on signup.

4.1.1.2. Self-managed gateway

Once you have successfully tested the integration with APIcast, you might want to host the API gateway yourself.

These are some of the errors you might encounter when you first install your self-managed gateway and call your API through it.

- **`500 Internal Server Error`** - Check nginx error.log:

- **`failed to load external Lua file X: cannot open X: No such file or directory`** - Check:

  - Lua file in correct directory

  - Specified location correctly

  - File exists

- **`lua entry thread aborted: runtime error: <path_to_library>: module 'X' not found:`** - Check:

  - Ensure you have installed all required dependencies

- **`lua entry thread aborted: runtime error: access_by_lua:1: module 'X' not found:`**

  - Check that the Lua files are in the correct location

- **`upstream timed out (110: Connection timed out) while connecting to upstream`**

  - Check that there are no firewalls between the API Gateway and the public internet that would prevent it from reaching 3scale.

Some other symptoms that may point to an incorrect self-managed gateway integration are as follows:

- **API calls routed incorrectly**: If you have multiple services pointing to different API backends, you will need to ensure you update the **`server_name`** directive in each server block of your nginx.conf. By default, this will be set to something like **`*.staging.apicast.io;`**

- **Mapping rules not matched** / **Double counting of API calls** Depending on the way you have defined the mapping between methods and actual URL endpoints on your API, you might find that sometimes methods either don't get matched or get incremented more than once per request. A good way to troubleshoot this is to make a test call to your API with the 3scale debug header. This will return a list of all the methods that have been matched by the API call.

- **Authentication parameters not found**: Ensure your are sending the parameters in the correct location as specified in the Service Integration screen. Please note that if you don't choose for credentials to be sent as headers, they should be sent as query parameters for GET requests and body parameters for all other HTTP methods. You can use the 3scale debug header to double-check the credentials that are being read from the request by the API gateway.

Finally, you might see the following alerts on nginx startup:

```
2016/03/18 10:31:25 [alert] 9790#0: lua_code_cache is off; this will hurt
performance in /path/to/nginx.conf:30
```

This setting is a good idea during integration as it allows you to make on-the-fly changes to your Lua code without restarting nginx. However, it should be turned on before going live. This directive is available on every server block and should be changed for all producion server blocks.

4.1.2. Production issues

It is unlikely that you'll have many problems with your API gateway once you've fully tested your setup and have been live with your API for a while. However, here are some of the sorts of issues you might encounter in a live production environment.

4.1.2.1. Availability issues

Availability issues are normally characterised by seeing **`upstream timed out`** errors in your nginx error.log, e.g.

```
upstream timed out (110: Connection timed out) while connecting to
upstream, client: X.X.X.X, server: api.example.com, request: "GET
/RESOURCE?CREDENTIALS HTTP/1.1", upstream: "http://Y.Y.Y.Y:80/RESOURCE?
CREDENTIALS", host: "api.example.com"
```

If you are experiencing intermittent 3scale availability issues, there could be a number of reasons for this:

- You are resolving to an old 3scale IP that is no longer in use.

The latest version of the API gateway configuration files defines 3scale as a variable to force IP resolution each time. For a quick fix, reload your NGINX instance. For a longer term fix, ensure that instead of defining the 3scale backend in an upstream block, you are defining it as a variable within each server block e.g.

```
server {
  # Enabling the Lua code cache is strongly encouraged for production use.
Here it is enabled
```

```
  .
  .
  .
  set $threescale_backend "https://su1.3scale.net:443";
```

and when you refer to it

```
location = /threescale_authrep {
  internal;
  set $provider_key "YOUR_PROVIDER_KEY";

  proxy_pass $threescale_backend/transactions/authrep.xml?
provider_key=$provider_key&service_id=$service_id&$usage&$credentials&log%
5Bcode%5D=$arg_code&log%5Brequest%5D=$arg_req&log%5Bresponse%5D=$arg_resp;
}
```

- You are missing some 3scale IPs from your whitelist. This is the current list of IPs that 3scale resolves to:
- 75.101.142.93
- 174.129.235.69
- 184.73.197.122
- 50.16.225.117
- 54.83.62.94
- 54.83.62.186
- 54.83.63.187
- 54.235.143.255

The above issues refer to problems with perceived 3scale availability. However, you might encounter similar issues with your API availability from the API gateway if your API is behind an AWS ELB. This is due to the fact that NGINX by default does DNS resolution at start-up time and then caches the IP addresses. However, ELBs do not ensure static IP addresses, and these might change from time to time. Whenever the ELB changes to a different IP, NGINX stops being able to reach it.

The solution for this is similar to the above fix for forcing runtime DNS resolution.

1. Set a specific DNS resolver such as Google's DNS. This is done by adding the following line near the top of the **`http`** section: **`resolver 8.8.8.8 8.8.4.4;`**
2. Set your API base URL as a variable, anywhere near the top of the **`server`** section. **`set $api_base "http://api.example.com:80";`**
3. Inside the **`location /`** section, find the line that says **`proxy_pass`** and replace it with the following: **`proxy_pass $api_base;`**

4.1.3. Post-deploy issues

If you make any changes to your API such as adding a new endpoint, you need to make sure you go through the steps to add a new method and URL mapping before downloading a new set of configuration files for your API gateway.

The most common problem when you've made modifications to the configuration downloaded from 3scale will be code errors in the Lua, which will result in a **`500 - Internal server error`** such as:

```
curl -v -X GET "http://localhost/"
* About to connect() to localhost port 80 (#0)
*   Trying 127.0.0.1... connected
> GET / HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0
OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host: localhost
> Accept: */*
>
< HTTP/1.1 500 Internal Server Error
< Server: openresty/1.5.12.1
< Date: Thu, 04 Feb 2016 10:22:25 GMT
< Content-Type: text/html
< Content-Length: 199
< Connection: close
<

<head><title>500 Internal Server Error</title></head>

<center><h1>500 Internal Server Error</h1></center>
<hr><center>openresty/1.5.12.1</center>


* Closing connection #0
```

You can then look into the nginx error.log to dig down into the cause, such as:

```
2016/02/04 11:22:25 [error] 8980#0: *1 lua entry thread aborted: runtime
error: /home/pili/NGINX/troubleshooting/nginx.lua:66: bad argument #3 to
'_newindex' (number expected, got nil)
stack traceback:
coroutine 0:
  [C]: in function '_newindex'
  /home/pili/NGINX/troubleshooting/nginx.lua:66: in function
'error_authorization_failed'
  /home/pili/NGINX/troubleshooting/nginx.lua:330: in function 'authrep'
  /home/pili/NGINX/troubleshooting/nginx.lua:283: in function 'authorize'
  /home/pili/NGINX/troubleshooting/nginx.lua:392: in function  while
sending to client, client: 127.0.0.1, server:
api-2445581381726.staging.apicast.io, request: "GET / HTTP/1.1", host:
"localhost"
```

In the access.log this will look something like:

```
127.0.0.1 - - [04/Feb/2016:11:22:25 +0100] "GET / HTTP/1.1" 500 199 "-"
"curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1
zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
```

The above sections should give you a good overview of the most common, well-known issues that you might encounter at any stage of your 3scale journey.

If all of these have been checked and you're still unable to find the cause and solution for your issue, you should proceed to the more detailed [operational troubleshooting](#troubleshooting-checklists) sections below. Start at your API and work your way back to the client in order to try to identify the point of failure.

4.2. TROUBLESHOOTING 101

If you're experiencing failures when connecting to a server, whether that is the API gateway, 3scale, or your API, the following troubleshooting steps should be your first port of call:

4.2.1. 1. Can we connect?

Use telnet to check basic TCP/IP connectivity **`telnet api.example.com 443`**

- Success

```
telnet echo-api.3scale.net 80
Trying 52.21.167.109...
Connected to tf-lb-i2t5pgt2cfdnbdfh2c6qqoartm-829217110.us-
east-1.elb.amazonaws.com.
Escape character is '^]'.
Connection closed by foreign host.
```

- Failure

```
telnet su1.3scale.net 443
Trying 174.129.235.69...
telnet: Unable to connect to remote host: Connection timed out
```

4.2.2. 2. Is it me or is it them?

Try to connect to the same server from different network locations, devices, and directions. For example, if your client is unable to reach your API, try to connect to your API from a machine that should have access such as the API gateway.

If any of the attempted connections succeed, you can rule out any problems with the actual server and concentrate your troubleshooting on the network between them, as this is where the problem will most likely lie.

4.2.3. 3. Is it a DNS issue?

Try to connect to the server by using its IP address instead of its hostname e.g. **`telnet 94.125.104.17 80`** instead of **`telnet apis.io 80`**

This will rule out any problems with the DNS.

You can get the IP address for a server using **`dig`** for example for 3scale **`dig su1.3scale.net`** or **`dig any su1.3scale.net`** if you suspect there may be multiple IPs a host may resolve to.

NB: Some hosts block `dig any`

4.2.4. 4. Is it an SSL issue?

You can use OpenSSL to test:

- Secure connections to a host or IP, such as from the shell prompt **`openssl s_client -connect su1.3scale.net:443`**

Output:

```
CONNECTED(00000003)
depth=1 C = US, O = GeoTrust Inc., CN = GeoTrust SSL CA - G3
verify error:num=20:unable to get local issuer certificate
---
Certificate chain
 0 s:/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks,
S.L./OU=IT/CN=*.3scale.net
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
 1 s:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIE8zCCA9ugAwIBAgIQcz2Y9JNxH7f2zpOT0DajUjANBgkqhkiG9w0BAQsFADBE
...
TRUNCATED
...
3FZigX+OpWLVRjYsr0kZzX+HCerYMwc=
-----END CERTIFICATE-----
subject=/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks,
S.L./OU=IT/CN=*.3scale.net
issuer=/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
---
Acceptable client certificate CA names
/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks,
S.L./OU=IT/CN=*.3scale.net
/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
Client Certificate Types: RSA sign, DSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+
SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1
:DSA+SHA1:ECDSA+SHA1:RSA+MD5
Shared Requested Signature Algorithms:
RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384:RSA+
SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA224:RSA+SHA1
:DSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA512
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 3281 bytes and written 499 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
```

```
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID:
A85EFD61D3BFD6C27A979E95E66DA3EC8F2E7B3007C0166A9BCBDA5DCA5477B8
    Session-ID-ctx:
    Master-Key:
F7E898F1D996B91D13090AE9D5624FF19DFE645D5DEEE2D595D1B6F79B1875CF935B3A4F6E
CCA7A6D5EF852AE3D4108B
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - a8 8b 6c ac 9c 3c 60 78-2c 5c 8a de 22 88 06 15   ..l..
<`x,\.."...
    0010 - eb be 26 6c e6 7b 43 cc-ae 9b c0 27 6c b7 d9 13   ..&l.
{C....'l...
    0020 - 84 e4 0d d5 f1 ff 4c 08-7a 09 10 17 f3 00 45 2c
......L.z.....E,
    0030 - 1b e7 47 0c de dc 32 eb-ca d7 e9 26 33 26 8b 8e
..G...2....&3&..
    0040 - 0a 86 ee f0 a9 f7 ad 8a-f7 b8 7b bc 8c c2 77 7b   ..........
{...w{
    0050 - ae b7 57 a8 40 1b 75 c8-25 4f eb df b0 2b f6 b7
..W.@.u.%O...+..
    0060 - 8b 8e fc 93 e4 be d6 60-0f 0f 20 f1 0a f2 cf 46   .......`..
....F
    0070 - b0 e6 a1 e5 31 73 c2 f5-d4 2f 57 d1 b0 8e 51 cc
....1s.../W...Q.
    0080 - ff dd 6e 4f 35 e4 2c 12-6c a2 34 26 84 b3 0c 19
..nO5.,.l.4&....
    0090 - 8a eb 80 e0 4d 45 f8 4a-75 8e a2 06 70 84 de 10
....ME.Ju...p...

    Start Time: 1454932598
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
---
```

- SSLv3 support (NOT supported by 3scale)

`openssl s_client -ssl3 -connect su.3scale.net:443`

Output

```
CONNECTED(00000003)
140735196860496:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert
handshake failure:s3_pkt.c:1456:SSL alert number 40
140735196860496:error:1409E0E5:SSL routines:ssl3_write_bytes:ssl handshake
failure:s3_pkt.c:644:
---
no peer certificate available
---
No client certificate CA names sent
```

```
---
SSL handshake has read 7 bytes and written 0 bytes
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : SSLv3
    Cipher    : 0000
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1454932872
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
---
```

These are just some examples. You can find some more details on usage in the OpenSSL man pages.

4.3. TROUBLESHOOTING CHECKLISTS

You should go through the following checks in order to identify where an issue with requests to your API might lie.

4.3.1. API

To confirm that the API is up and responding to requests, try to make the same request directly to your API (not going through the API gateway). You should ensure that you're sending all of the same parameters and headers as the request that goes through the API gateway. If you're unsure of the exact request that is failing, capture the traffic between the API gateway and your API.

If the call succeeds, you can rule out any problems with the API, otherwise you should troubleshoot your API further.

4.3.2. API Gateway > API

To rule out any network issues between the API gateway and the API, try to make the same call as before — directly to your API — from your API gateway server.

If the call succeeds, you can move on to troubleshooting the API gateway itself.

4.3.3. API gateway

There are a number of steps to go through in order to check that the API gateway is working correctly.

4.3.3.1. 1. Is the API gateway up and running?

Try to log in to the machine where the gateway is running. If this fails, your gateway server might be down.

Once you have logged in, check that the NGINX process is running. You can do this by running **`ps ax | grep nginx`** or **`htop`**

If you see **`nginx master process`** and **`nginx worker process`** in the list, NGINX is running.

4.3.3.2. 2. Are there any errors in the gateway logs?

Here are some common errors you might see in the gateway logs, for example in error.log

- API gateway can't connect to API

```
upstream timed out (110: Connection timed out) while connecting to
upstream, client: X.X.X.X, server: api.example.com, request: "GET
/RESOURCE?CREDENTIALS HTTP/1.1", upstream: "http://Y.Y.Y.Y:80/RESOURCE?
CREDENTIALS", host: "api.example.com"
```

- API gateway can't connect to 3scale

```
2015/11/20 11:33:51 [error] 3578#0: *1 upstream timed out (110:
Connection timed out) while connecting to upstream, client: 127.0.0.1,
server: , request: "GET /api/activities.json?user_key=USER_KEY HTTP/1.1",
subrequest: "/threescale_authrep", upstream:
"https://54.83.62.186:443/transactions/authrep.xml?
provider_key=YOUR_PROVIDER_KEY&service_id=SERVICE_ID&usage[hits]=1&user_ke
y=USER_KEY&log%5Bcode%5D=", host: "localhost"
```

4.3.4. API gateway > 3scale

Once you are sure the API gateway is running correctly, the next step is troubleshooting the connection between the API gateway and 3scale.

4.3.4.1. 1. Can the API gateway reach 3scale?

If you're using NGINX as your API gateway, you should see the following in the nginx error logs when the gateway is unable to contact 3scale.

```
2015/11/20 11:33:51 [error] 3578#0: *1 upstream timed out (110:
Connection timed out) while connecting to upstream, client: 127.0.0.1,
server: , request: "GET /api/activities.json?user_key=USER_KEY HTTP/1.1",
subrequest: "/threescale_authrep", upstream:
"https://54.83.62.186:443/transactions/authrep.xml?
provider_key=YOUR_PROVIDER_KEY&service_id=SERVICE_ID&usage[hits]=1&user_ke
y=USER_KEY&log%5Bcode%5D=", host: "localhost"
```

The main thing to note here is the upstream value. This IP corresponds to one of the IPs that the 3scale API Management service resolves to, so you know there is some problem reaching 3scale. You can do a reverse DNS lookup to check the domain for an IP by calling nslookup e.g.

Just because the API gateway is unable to reach 3scale, it does not necessarily mean that 3scale is down. One of the most common reasons for this would be firewall rules preventing the API gateway from connecting to 3scale.

As such, there may be some network issues between the gateway and 3scale that could cause connections to timeout. If that is happening, you should go through the steps in "troubleshooting generic connectivity issues" to identify where the problem lies.

In order to rule out networking issues, you can use traceroute or MTR to check the routing and packet transmission. It might also be useful to run the same command from a machine that is able to connect to 3scale and your API gateway and compare the output.

Additionally, to see the traffic that is being sent between your API gateway and 3scale, you can use tcpdump as long as you temporarily switch to using the HTTP endpoint for the 3scale API Management service (**`su1.3scale.net`**.)

4.3.4.2. 2. Is the API gateway resolving 3scale addresses correctly?

Ensure you have the resolver directive added to your nginx.conf.

For example, in nginx.conf:

```
http {
  lua_shared_dict api_keys 10m;
  server_names_hash_bucket_size 128;
  lua_package_path ";;$prefix/?.lua;";
  init_by_lua 'math.randomseed(ngx.time()) ; cjson = require("cjson")';

  resolver 8.8.8.8 8.8.4.4;
```

If you'd rather not use Google's DNS (8.8.8.8 and 8.8.4.4) you can substitute those values with your preferred DNS.

To check DNS resolution from your API gateway, you can call nslookup as follows with the specified resolver IP:

```
nslookup su1.3scale.net 8.8.8.8
;; connection timed out; no servers could be reached
```

The above example shows the response returned if Google DNS cannot be reached. If this is the case, you will need to update the resolver IPs. You might also see the following alert in your nginx error.log:

```
2016/05/09 14:15:15 [alert] 9391#0: send() failed (1: Operation not
permitted) while resolving, resolver: 8.8.8.8:53
```

Finally, you can run **`dig any su1.3scale.net`** to see the IP addresses currently in operation for the 3scale Service Management API. Please note that this is not the entire range of IP addresses that might be used by 3scale, as some might be swapped in and out for capacity reasons. Additionally, we may add more domain names for the 3scale API Management service in the future, so you should always test against the specific address which will have been supplied to you during integration, if applicable.

4.3.4.3. 3. Is the API gateway calling 3scale correctly?

If you want to check the request your API gateway is making to 3scale — for troubleshooting purposes only — you can add the following snippet to the 3scale authrep location in **`nginx.conf`** (**`/threescale_authrep`** for API Key and App_id authentication modes and **`/threescale_oauth_authrep`** for OAuth)

```
body_filter_by_lua_block{
  if ngx.req.get_headers()["X-3scale-debug"] == ngx.var.provider_key then
    local resp = ""
    ngx.ctx.buffered = (ngx.ctx.buffered or "") .. string.sub(ngx.arg[1],
1, 1000)
    if ngx.arg[2] then
      resp = ngx.ctx.buffered
    end

    ngx.log(0, ngx.req.raw_header())
    ngx.log(0, resp)
  end
}
```

You can also find the above snippet in our codehub.

This snippet will add the following extra logging to the nginx error.log when the **`X-3scale-debug header`** is sent, e.g. **`curl -v -H 'X-3scale-debug: YOUR_PROVIDER_KEY' -X GET "https://726e3b99.ngrok.com/api/contacts.json?access_token=7c6f24f5"`**

Will produce the following log entries:

```
2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:7: GET
/api/contacts.json?access_token=7c6f24f5 HTTP/1.1
Host: 726e3b99.ngrok.io
User-Agent: curl/7.43.0
Accept: */*
X-Forwarded-Proto: https
X-Forwarded-For: 2.139.235.79

 while sending to client, client: 127.0.0.1, server: pili-virtualbox,
request: "GET /api/contacts.json?access_token=7c6f24f5 HTTP/1.1",
subrequest: "/threescale_oauth_authrep", upstream:
"https://54.83.62.94:443/transactions/oauth_authrep.xml?
provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f2
4f5", host: "726e3b99.ngrok.io"
2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:8: <?xml
version="1.0" encoding="UTF-8"?><error
code="access_token_invalid">access_token "7c6f24f5" is invalid: expired or
never defined</error> while sending to client, client: 127.0.0.1, server:
pili-virtualbox, request: "GET /api/contacts.json?access_token=7c6f24f5
HTTP/1.1", subrequest: "/threescale_oauth_authrep", upstream:
"https://54.83.62.94:443/transactions/oauth_authrep.xml?
provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f2
4f5", host: "726e3b99.ngrok.io"
```

The first entry (**`2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:7:`**) prints out the request headers sent to 3scale, in this case: Host, User-Agent, Accept, X-Forwarded-Proto and X-Forwarded-For.

The second entry (**`2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:8:`**) prints out the response from 3scale, in this case: **`<error code="access_token_invalid">access_token "7c6f24f5" is invalid: expired or never defined</error>`**

Both will print out the original request (**`GET /api/contacts.json?access_token=7c6f24f5`**) and subrequest location (**`/threescale_oauth_authrep`**) as well as the upstream request (**`upstream: "https://54.83.62.94:443/transactions/oauth_authrep.xml?provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f24f5"`**.) This last one is the most important, as it allows you not only to see which of the 3scale IPs have been resolved, but also the exact request made to 3scale.

4.3.5. 3scale

4.3.5.1. 1. Is 3scale available?

Check @3scalestatus on Twitter.

4.3.5.2. 2. Is 3scale returning an error?

It's also possible that 3scale is available but is returning an error to your API gateway which would prevent calls going through to your API. Try to make the authorization call directly in 3scale and check the response. If you get an error, check the #troubleshooting-api-error-codes[Error Codes] section to see what the issue could be.

4.3.5.3. 3. Use the 3scale debug headers

Another option is to turn on the 3scale debug headers by making a call to your API with the **`X-3scale-debug`** header, e.g.

```
curl -v -X GET "https://api.example.com/endpoint?user_key" X-3scale-debug: YOUR_PROVIDER_KEY
```

This will return the following headers with the API response:

```
X-3scale-matched-rules: /, /api/contacts.json
< X-3scale-credentials: access_token=TOKEN_VALUE
< X-3scale-usage: usage[hits]=2
< X-3scale-hostname: HOSTNAME_VALUE
```

4.3.5.4. 4. Check the integration errors

You can also check the integration errors on your Admin Portal to check for any issues reporting traffic to 3scale. You can find this at https://YOUR_DOMAIN-admin.3scale.net/apiconfig/errors

Some common reasons for integration errors:

- Sending credentials in the headers with **`underscores_in_headers`** directive not enabled in server block.

4.3.6. Client API gateway

4.3.6.1. 1. Is the API gateway reachable from the public internet?

Try directing a browser to the IP address (or domain name) of your gateway server. If this fails, make sure that you have opened the firewall on the relevant ports.

4.3.6.2. 2. Is the API gateway reachable by the client?

If possible, try to connect to the API gateway from the client using one of the methods outlined earlier (telnet, curl, etc…) If the connection fails, the problem lies in the network between the two.

Otherwise, you should move on to troubleshooting the client making the calls to the API.

4.3.7. Client

4.3.7.1. 1. Test the same call using a different client

If a request is not returning the expected result, test with a different HTTP client. For example, if you are calling an API with a Java HTTP client and you see something weird, cross-check with cURL.

It might also help to call the API through a proxy between the client and the gateway to capture the exact parameters and headers being sent by the client.

4.3.7.2. 2. Inspect the traffic sent by client

You can use a tool like Wireshark to see the requests being made by the client. This will allow you to identify whether the client is actually making calls out to the API and the details of the request.

4.4. OTHER ISSUES

4.4.1. ActiveDocs issues

You might find that sometimes calls that work when you call the API from the command line fail when going through ActiveDocs.

In order to enable ActiveDocs calls to work, we send these out through a proxy on our side. This proxy will add certain headers that can sometimes cause issues on the API if they are not expected. To identify if this is the case, you can try the following steps:

4.4.1.1. 1. Use petstore.swagger.io

Swagger provides a hosted swagger-ui at petstore.swagger.io which you can use to test your Swagger spec and API going through the latest version of swagger-ui. If both swagger-ui and ActiveDocs fail in the same way, you can rule out any issues with ActiveDocs or the ActiveDocs proxy and focus the troubleshooting on your own spec. Alternatively, you can check the swagger-ui GitHub repo for any known issues with the current version of swagger-ui.

4.4.1.2. 2. Check that firewall allows connections from ActiveDocs proxy

Our recommendation here would be not to whitelist IP address for clients using your API. The ActiveDocs proxy uses floating IP addresses for high availability, and there is currently no mechanism to notify of any changes to these IPs.

4.4.1.3. 3. Call the API with incorrect credentials

One way to identify whether the ActiveDocs proxy is working correctly is to call your API with invalid credentials. This will help you to confirm or rule out any problems with both the ActiveDocs proxy and your API gateway.

If you get a 403 code back from the API call (or whichever code you have configured on your gateway for invalid credentials), the problem lies with your API, as the calls are reaching your gateway at the very least.

4.4.1.4. 4. Compare calls

To identify any differences in headers and parameters between calls made from ActiveDocs versus outside of ActiveDocs, it can sometimes be helpful to run your calls through some sort of service (APItools on premise, Runscope, etc.) that allows you to inspect and compare your HTTP calls before sending them on to your API. This will allow you to identify any potential headers and/or parameters in the request that could be causing issues on your side.

4.5. APPENDIX

4.5.1. Logging in NGINX

For a comprehensive guide on this, check out the NGINX Logging and Monitoring docs.

4.5.1.1. Enabling debugging log

To find out more about this, we encourage you to check out the NGINX debugging log documentation.

4.5.2. 3scale error codes

You can double-check the error codes that are returned by the 3scale Service Management endpoints by taking a look at our Service Management API ActiveDocs.

Here is a list of the most important codes returned by 3scale and the conditions under which they would be returned:

- **400:** Bad request. This can be because of:
- Invalid encoding.
- Payload too large.
- Content type is invalid (for POST calls). Valid values for the **`Content-Type`** header are: **`application/x-www-form-urlencoded`**, **`multipart/form-data`**, or empty header.
- **403:**
- Credentials are not valid.
- Sending body data to 3scale for a GET request.
- **404:** Non-existent entity referenced, such as applications, metrics, etc.
- **409:**
- Usage limits exceeded.
- Application is not active.
- Application key is invalid or missing (for app_id/app_key authentication method).

- Referrer is not allowed or missing (when referrer filters are enabled and required).
- **422:** Missing required parameters.
- **428:** Too many requests — sent when you exceed 2000 reqs/s.

Most of these error responses will also contain an XML body with a machine readable error category as well as a human readable explanation.

Please note that, if using the standard API gateway configuration, any non-200 return code from 3scale will result in a 403 being returned to the client. As such, you will need to double-check the real code returned by 3scale in your API gateway logs.