Kong Gateway Operations for Kubernetes

Advanced Plugins Review

Course Agenda

1. Kong Gateway Installation on K8s
2. Intro to Kong Ingress Controller
3. Securing Kong Gateway
4. Securing Services on Kong
5. OIDC Plugin
6. Kong Vitals
7. **Advanced Plugins Review**
8. Troubleshooting
9. Test your Knowledge

Learning Objectives

Understand, explain, configure, and administer

- Rate Limiting Advanced Plugin
- Request & Response Transformer
- jq Plugin
- Exit Transformer Plugin
- Prometheus Plugin

Reset Environment

Before we start this lesson, be sure to reset your environment:

```
$ cd /home/labuser
$ git clone https://github.com/Kong/edu-kgac-202.git
$ source ./edu-kgac-202/base/reset-lab.sh
```

Advanced Plugins

In the final lesson of this course we will review a few Kong Plugins you should be aware of, namely

- Prometheus Plugin
- Rate Limiting Advanced Plugin
- Request & Response Transformer Advanced Plugins
- jq Plugin
- Exit Transformer Plugin

Advanced Plugins are available with Kong Enterprise subscription. These plugins are supersets of their open source version, and provide additional features.

For availability of plugins under Free, Plus and Enterprise License Tiers, check out Kong License Tiers at Kong Plugin Hub: https://docs.konghq.com/hub/

Plugins Execution Order

Some plugins might depend on the execution of others to perform some operations. For example, plugins relying on the identity of the consumer have to run after authentication plugins.

Considering this, Kong Gateway defines priorities between plugins execution to ensure that order is respected.

Please consult documentation, to find out more about the current order of execution for bundled plugins:
https://docs.konghq.com/gateway/latest/plugin-development/custom-logic/#plugins-execution-order

Rate Limiting Advanced

What is the Rate Limiting Advanced Plugin?

- The Rate Limiting Advanced plugin is a re-engineered version of the Kong OSS Rate Limiting plugin.
- It is used to limit the number of HTTP(S) requests which could be made in **multiple configurable time periods** to maintain QoS and overall availability.
- The Rate Limiting Advanced plugin provides increased throughput performance with better accuracy over the Kong OSS Rate Limiting plugin.
- Rate limiting is an important part of API security, as DoS attacks and intended/unintended overuse can overwhelm a server with unlimited API requests.

Rate Limiting Advanced Sample Use Case

- Protection against DDoS attacks
- Protection against API abuse
- Protection against inadvertent overuse
- Protection against brute force attacks
- Limit sensitive data exposure
- Protect Computationally and/or financially intensive endpoints served by auto-scaling
- Provide configurable rate time periods. For example, 10 request per 90 seconds

Rate Limiting Advanced Plugin Usage

- The Rate Limiting Advanced Plugin can limit usage based on IP address, Consumer, Consumer Credential, Service, & HTTP Header, and can be scoped to Consumer, Route, Service or globally.
- Provides the ability to apply multiple rate limits in sliding or fixed windows
- Multiple instances of the Rate Limiting Advanced plugin may be applied to services, routes and/or consumers.
- Tweak performance of the plugin by configuring how often the plugin synchronises with backend counter storage data store.

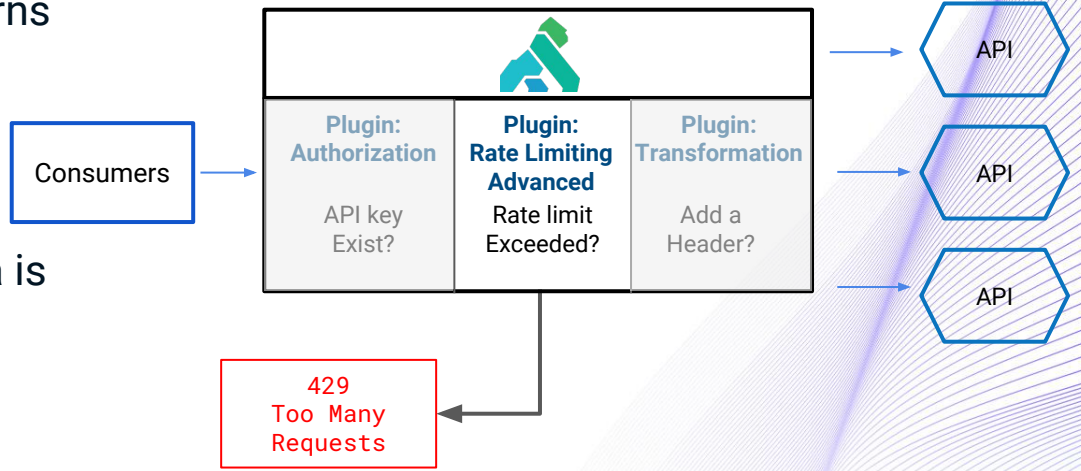
Rate Limiting Advanced Plugin Data Flow

Kong receives client's request, and based on the plugin setup and client criteria, provides the response through normal flow, or returns HTTP 429 status code with this JSON body

```
{"message": "API rate limit exceeded"}
```

When this plugin is enabled, Kong returns `headers` to the client indicating

- the allowed limits
- requests remain available, and
- the time remaining until the quota is reset.



Enabling Rate Limiting Advanced Plugin

The Rate Limiting Advanced Plugin can be configured through Kong Manager, Admin API or Declaratively.

Here is an example of the plugin scoped to a service using the Admin API, with counters stored in the datastore and shared across nodes, plugin configured to keep proxying requests in case of loss of connectivity to the datastore. Limits are set to 5 requests per 30 seconds, on a fixed window, rate limited by IP address and accessed through each node of a Kong cluster:

```
$ http --form POST http://testHost:30001/services/testService/plugins \
    name=rate-limiting-advanced \
    config.window_size=30 \
    config.limit=5 \
    config.window_type=fixed \
    config.identifier=ip \
    config.strategy=cluster
```

Enabling Rate Limiting Advanced Plugin

The Rate Limiting Advanced Plugin can be configured through Kong Manager, Admin API or Declaratively

Here is an example of the plugin that is

- Configured using the Admin API as follows
- Scoped to service 'testService'
- With limits are set to 5 requests per 30 seconds,
- On a fixed window,
- Rate limited by IP address and accessed through each node of a Kong cluster:

```
$ http --form POST
http://testHost:30001/services/testService
/plugins \
    name=rate-limiting-advanced \
    config.window_size=30 \
    config.limit=5 \
    config.window_type=fixed \
    config.identifier=ip \
    config.strategy=cluster
```

In this case counters are stored in the datastore shared across nodes, so will keep proxying requests in case of loss of connectivity to the datastore.

Rate Limiting Advanced Counter Storage Strategies

A central datastore is used for retrieving and incrementing rate limits. Available values are:

- **'cluster'** (default): stores counters in the cluster's datastore. Each requests forces a read/write, so relatively largest impact on performance among these policies. However, it is accurate and no extra components are needed to implement.
- **'redis'**: stores counters in Redis, external to Kong. It is accurate, but needs installation/configuration of Redis as an extra component.

➔ There is no **local** strategy available like the Rate Limiting plugin. However if **local** is required, use **cluster** strategy with a `sync_rate` of `'-1' to` stores counters in-memory on the node.

➔ **local** is the simplest strategy to implement with minimal performance impact, however it's less accurate, and could diverge when scaling, unless a consistent-hashing load balancer is used.

Implementation Considerations

- For general back-end protection , where accuracy is of less importance, ‘local’ would work. You need to fine tune based on number of nodes in Kong cluster and load balancing policy. In cases where accuracy is important, such as financial/military, recommendation is to start with ‘cluster’ and move to ‘redis’ in case of drastic performance impact.
- In a Kong DB-less setup, the cluster strategy is not supported, as it requires to be able to write the counters to to the Kong database. Furthermore, this plugin does not support the cluster strategy in Hybrid mode, so only ‘local’ and ‘redis’ can be used for Data Planes.
- Default entity used when aggregating limits is set to ‘consumer’, but could have other values such as ‘header’, ‘path’, ‘service’, ‘credential’ and ‘ip’. When selected entity can't be retrieved, due to issues such as missing header/service, the plugin will fallback to use IP as identifier.

Task: Configure Service/Route/Plugin/Consumer

Let’s configure a Configure Service/Route/Plugin/Consumer which we will use to generate some traffic:

```
$ cd ~/edu-kgac-202/exercises/adv-plugins
$ kubectl apply -f ./httpbin-ingress-jane.yaml
namespace/httpbin-demo created
kongplugin.configuration.konghq.com/httpbin-auth created
service/httpbin-service created
ingress.networking.k8s.io/httpbin-ingress created
secret/jane-apikey created
kongconsumer.configuration.konghq.com/jane created
$
```

Task: Configure Rate Limiting Advanced Plugin

Now configure rate-limiting-advanced plugin to rate limit 10 requests per minute & 20 per 90 seconds, on a sliding window, with real-time sync with the redis datastore:

```
$ kubectl apply -f ./rate-limiting-adv.yaml
kongclusterplugin.configuration.konghq.com/global-rate-limit-adv created
$
```

Task: Create Traffic with Advanced Rate Limiting Plugin Enabled

Copy and paste the command, and leave it running. This will proxy a request every second for 32 seconds

```
$ for ((i=0;i<32;i++))
  do
    sleep 1
    http get $KONG_PROXY_URI/httpbin?apikey=JanePassword
  done
```

```
HTTP/1.1 429 Too Many Requests
...
RateLimit-Limit: 10
RateLimit-Remaining: 0
RateLimit-Reset: 55
Retry-After: 55
{
  "message":"API rate limit exceeded"
}
```

You'll quickly see the rate limit being exceeded.

Troubleshooting Tips/Tricks

- When this plugin is enabled, Kong sends additional headers back to the client with useful information on the operation of the plugin:

```
RateLimit-Limit: 10 # Allowed limits
RateLimit-Remaining: 0 # Remaining available requests
RateLimit-Reset: 75 # Remaining seconds until quota is reset
X-RateLimit-Limit-Minute: 10 # Allowed limits
X-RateLimit-Remaining-Minute: 0 # Remaining available requests
```

- If more than one limit is set, a combination of time limits will be included in the response headers by Kong:

```
X-RateLimit-Limit-90: 20
X-RateLimit-Limit-minute: 10
X-RateLimit-Remaining-90: 11
X-RateLimit-Remaining-minute: 1
```

Request & Response Transformer Advanced Plugins

What are the Request & Response Transformer Advanced Plugins?

- The Request & Response Transformer Advanced plugins build on the Kong OSS Request and Response Transformer plugins base functionality for transforming requests and responses using regular expressions, templates & variables
- Enhanced capabilities include:
  - Transform nested JSON objects & arrays in JSON payload for transforming response payloads
  - JSON body contents can be restricted to a set of allowed properties
  - Arbitrary transformation functions written in Lua can be applied
  - Responses payloads can be GZip decompressed/recompressed
- Order of execution of operations within the plugin: remove → rename → replace → add → append

Transformer Advanced Use Cases

- ➔ Stripping or obfuscating sensitive data in the header/body
- ➔ Supporting requests with old specification during transition periods
- ➔ Copying/Moving an API Key from the query string to the header
- ➔ Removing query string value
- ➔ Adding a version number to a query string
- ➔ Modifying the header token to Bearer Auth
- ➔ Moving JWT from header to body
- ➔ Simple and complex URL rewriting for request transformations
- ➔ Editing JSON body for request & responses
- ➔ Configuring request & response HTTP headers
- ➔ Configuring request query parameters
- ➔ Configuring request HTTP method
- ➔ Executing response transformations for list of response status codes
- ➔ Modifying complex response JSON payloads with nested objects

Enabling **Request** Transformer Advanced Plugin

Here is an example of the plugin scoped to a service, that removes/renames/replaces/adds HTTP headers, query strings and body properties:

```
http --form POST testHost:30001/testService/plugins \
  name=request-transformer-advanced \
  config.add.headers=x-added:value \
  config.remove.headers=x-to-remove \
  config.rename.headers=x-old-name:x-new-name \
  config.replace.headers=x-to-replace:new-value \
  config.add.querystring=qs-added:value \
  config.remove.querystring=qs-to-remove \
  config.rename.querystring=qs-old-name:qs-new-name \
  config.replace.querystring=qs-to-replace:new-value \
  config.add.body=body-param-added:value \
  config.remove.body=body-param-to-remove \
  config.rename.body=body-param-old-name:body-param-new-name \
  config.replace.body=body-param-to-replace:new-value
```

Enabling **Response** Transformer Advanced Plugin

Here is an example of the plugin scoped to a service, that adds/removes/renames/replaces HTTP headers and body properties:

```
http --form POST testHost:30001/testService/plugins \
  name=response-transformer-advanced \
  config.add.headers=x-added:value \
  config.remove.headers=x-to-remove \
  config.rename.headers=x-old-name:x-new-name \
  config.replace.headers=x-to-replace:new-value \
  config.add.json=json-key-added:value \
  config.remove.json=json-key-to-remove \
  config.replace.json=json-key-to-replace:new-value
```

Request Transformations Lab

Task: Configure Request Transformer Advanced Plugin

Configure request-transformer-advanced plugin to modify the HTTP request:

```
$ kubectl apply -f ./request-transform.yaml
namespace/httpbin-demo unchanged
kongclusterplugin.configuration.konghq.com/global-request-transform-adv
created
```

This will

add the header `X-Kong-Test-Request-Header:MyRequestHeader`

and

`rename header User-Agent to My-User-Agent`

Task: Create Request to See Request Headers

Now submit a request and view the request headers echoed back in the body by the upstream:

```
$ http get $KONG_PROXY_URI/httpbin?apikey=JanePassword
```

```
. . .
"My-User-Agent": "HTTPie/0.9.8",
"X-Kong-Test-Request-Header": "MyRequestHeader",
. . .

HTTP/1.1 200 OK
```

Response Transformations Lab

Task: Configure Response Transformer Advanced Plugin

Configure response-transformer-advanced plugin to modify the JSON response body:

```
$ kubectl apply -f ./response-transform.yaml
namespace/httpbin-demo unchanged
kongclusterplugin.configuration.konghq.com/global-response-transform-adv
created
```

This will add the response header

```
X-Kong-Test-Header:Test-Value
```

And also will add this JSON key to the response body

```
json-key-added:Test-Key
```

Task: Create Request to See Response Headers/Body

Now submit a request and view the response header and JSON key added to the response:

```
$ http -h get $KONG_PROXY_URI/httpbin?apikey=JanePassword
```

```
HTTP/1.1 200 OK
...
X-Kong-Proxy-Latency: 1
X-Kong-Test-Response-Header: MyResponseHeader
X-Kong-Upstream-Latency: 39
X-Numbers-API-Number: 1578
X-Numbers-API-Type: year
. . .
    "json-key-added": "Test-Key",
. . .
```

jq Plugin

What is the jq Plugin?

- jq is a lightweight and flexible CLI JSON processor that allows the user to manipulate and transform JSON documents.
- jq is like awk, grep or sed for JSON data, you can use it to slice and filter and map and transform structured data within a JSON document.
- The jq plugin enables arbitrary jq transformations on JSON objects included in API requests or responses.
- This plugin covers the functionality of both request and response transformer plugins, and adds more capabilities, when working with JSON bodies.
- jq plugin requires an Enterprise license and is available in 2.6+ releases of Kong.

Enabling jq Plugin

The jq Plugin can be configured through Kong Manager, Admin API or Declaratively. You can configure it using admin API, by posting to /plugins endpoint under the desired scope of global/consumer/route/service. To configure your jq filter, you need to provide the request_jq_program and/or response_jq_program options in your plugin config.

The Kong documentation at https://docs.konghq.com/hub/kong-inc/jq has a comprehensive overview of the available options. Here is an example of the plugin scoped to a service, that uses jq to extract the title key from multiple items in a response:

```
http -f POST testHost:30001/testService/plugins \
  name=jq \
  config.response_jq_program=[.[].title]
```

jq Plugin Lab

Task: Configure jq Plugin

Configure jq plugin to modify the HTTP request to remove apikey from the response:

```
$ http -b get kongcluster:30000/httpbin?apikey=JanePassword
```

```
. . .
    "args": {
        "apikey": "JanePassword"
    },
    "startedDateTime": "2022-04-25T22:27:03.947Z",
    "url": "http://kongcluster/anything?apikey=JanePassword"
}
```

```
$ kubectl apply -f ./jq.yaml
namespace/httpbin-demo unchanged
kongplugin.configuration.konghq.com/httpbin-jq created
service/httpbin-service configured
```

Task: Create Request to See Response Body

Now submit a request and view the response body from the upstream filtered by the jq plugin:

```
$ http -b get $KONG_PROXY_URI/httpbin?apikey=JanePassword
```

```
. . .
  "method": "GET",
    "postData": {
        "mimeType": "application/octet-stream",
        "params": [],
        "text": ""
    },
    "args": {},
    "startedDateTime": "2022-04-25T22:41:48.830Z",
    "url": "http://30000-1-6936d79c.labs.konghq.com/anything?apikey=JanePassword"
}
```

Exit Transformer Plugin

What is the Exit Transformer Plugin?

- Exit Transformer Plugin is used to customize and transform Kong response exit messages using Lua functions. This plugin has a range of capabilities from changing body/header/status code to complete transformation of the structure of responses.
- Kong API Gateway is built on OpenResty, which extends the NGINX proxy server to run Lua scripts. Consequently you can use Lua to create custom function or plugins.
- Exit Transformer Plugin requires an Enterprise license.
- This plugin is compatible with DB-less mode, where you configure Kong Gateway declaratively.

Enabling Exit Transformer Plugin

To configure the plugin, you need to provide an array of functions used to transform any Kong proxy exit response.

The Kong documentation at https://docs.konghq.com/hub/kong-inc/exit-transformer has a comprehensive overview of the available options. Here is an example of the plugin scoped to a service, that uses my_function.lia to transform Kong's proxy exit response:

```
http -f POST testHost:30001/testService/plugins \
  name=exit-transformer \
  config.config.functions=@my_function.lua
```

Exit Transformer Plugin Lab

Task: Use Transformer Plugin

Configure Exit Transformer plugin to add a header, append to any message, and add error and status field to the response body, in case of an error.

First we can make a call before applying the plugin to see the exit response:

```
$ http get $KONG_PROXY_URI/httpbin
```

```
HTTP/1.1 401 Unauthorized
Connection: keep-alive
. . .
X-Kong-Response-Latency: 0
X-Kong-Test-Response-Header: MyResponseHeader
{
    "json-key-added": "Test-Key",
    "message": "No API key found in request"
}
```

Task: Configure Exit Transformer Plugin

Next, create an instance of the exit transformer plugin, supplying the lua function from previous slide as its function:

```
$ kubectl apply -f ./exit-transform.yaml
namespace/httpbin-demo unchanged
kongplugin.configuration.konghq.com/httpbin-exit-transform created
service/httpbin-service configured
```

Task: Create Request to See Response Header/Body Transformation

Now submit a request and view the response body from the upstream transformed by the exit transformer plugin:

```
$ http get $KONG_PROXY_URI/httpbin
```

```
HTTP/1.1 401 Unauthorized
...
X-Kong-Test-Response-Header: MyResponseHeader
X-Some-Header: ETP Triggered

{
    "error": true,
    "json-key-added": "Test-Key",
    "message": "No API key found in request, Achtung!",
    "status": 401
}
```

Prometheus Plugin

What is Prometheus?

- Prometheus is an open-source system monitoring and alerting toolkit.
- Prometheus collects and stores real-time metrics in a time series database, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels.
- Prometheus was originally built at SoundCloud, but joined the CNCF framework in 2016 as the second hosted project, after Kubernetes.
- Prometheus is now a standalone open source project and maintained independently of any company.
- Prometheus has widespread support and has become the de facto standard for monitoring in the microservices ecosystem.
- Prometheus has a very active developer and user community.

What is the Prometheus Kong Plugin?

- The Prometheus Plugin exposes metrics in Prometheus Exposition format which can then be scraped by a Prometheus Server. The exposed metrics may relate to Kong Gateway itself, or proxied upstream services.

- These metrics can then be visualised in Prometheus and Grafana. Like Prometheus, Grafana is also an open source product. Grafana enables the user to create dashboards for monitoring systems.

- We saw how to visualise Kong Vitals Metrics using Prometheus and Grafana in the previous lesson. That did not involve using the Prometheus plugin and was aimed at visuasilising metrics provided by Vitals. The configuration process on the Prometheus/Grafana is quite similar, but these are two distinct methods of visualising metrics.

Prometheus Kong Plugin Compatibility

- This plugin is compatible with DB-less mode. In DB-less mode, the Admin API is mostly read-only. The only tasks it can perform are all related to handling the declarative configurations, including:
  - setting a target's health status in the load balancer
  - validating configurations against schemas
  - uploading the declarative configuration using the /config endpoint

- The database will always be reported as reachable in Prometheus with DB-less.
- the DB entity count metric (kong_db_entities_total) is not emitted in DB-less mode.

Enabling the Prometheus Plugin

- To configure it using admin API, post to `/plugins` endpoint under the desired scope of service:

```
http --form post testServer:30001/services/testService/plugins \
  name=prometheus
```

- Or configure it globally:

```
http --form post testServer:30001/plugins \
  name=prometheus
```

Available Metrics

The following metrics are exposed by the Prometheus plugin:

- **Status codes**: HTTP status codes returned by Upstream services, either per service and across all services.
- **Latencies Histograms**: Latency as measured at Kong:
  - **Request**: Total time taken by Kong and Upstream services to serve requests.
  - **Kong**: Time taken for Kong to route a request and run all configured plugins.
  - **Upstream**: Time taken by the Upstream service to respond to requests.
- **Bandwidth**: Total egress/ingress Bandwidth flowing through Kong, per service and as a sum across all services.
- **DB reachability**: A gauge type with a value of 0 or 1, which represents whether DB can be reached by a Kong node.
- **Connections**: Various Nginx connection metrics like active, reading, writing, and number of accepted connections.
- **Target Health**: The health status of Targets for a given Upstream (healthchecks_off, healthy, unhealthy, or dns_error).
- **Dataplane Status**: Last seen timestamp, config hash, and config sync status for data plane nodes that is exported to control plane.
- **Enterprise License Information**: The Kong Gateway license expiration date, features and license signature.

Accessing Metrics

Metrics exposed through the Prometheus plugin can be accessed in two ways:

- Through the Admin API at {HOST}:30001/metrics. Note that this endpoint is unavailable when RBAC is enabled on the Admin API, as Prometheus does not support Key-Auth to pass the token.
- Through the Status API at {node}:port/metrics, if it is enabled. The Status API is a read-only endpoint allowing monitoring tools to retrieve metrics, healthiness, and other non-sensitive information of the current Kong node. This is the preferred method.

This plugin records and exposes metrics at the node level. Your Prometheus server will need to discover all Kong nodes via a service discovery mechanism, and consume data from each node’s configured /metrics endpoint.

Task: Enable the Prometheus Plugin & Generate Traffic

This will enable the Prometheus plugin with a global scope and proxy a request every second for 64 seconds:

```
$ kubectl apply -f ./prometheus-adv.yaml
namespace/httpbin-demo unchanged
kongclusterplugin.configuration.konghq.com/global-prometheus created
```

```
$ for ((i=0;i<32;i++))
    do
      sleep 1
      http -h get $KONG_PROXY_URI/httpbin?apikey=JanePassword > /dev/null 2>&1
      sleep 1
      http -h get $KONG_PROXY_URI/httpbin > /dev/null 2>&1
    done &
```

Task: Get Prometheus Plugin Metrics through API

Expose the services for the status ports on the control plane and data plane pods and get shell in k8s:

```
$ DP_POD=`kubectl get pods --selector=app=kong-dp-kong -n kong-dp -o
jsonpath='{.items[*].metadata.name}'`
$ CP_POD=`kubectl get pods --selector=app=kong-kong -n kong -o
jsonpath='{.items[*].metadata.name}'`
$ kubectl expose pod $DP_POD --name kong-dp-status -n kong-dp --port=8100
--target-port=8100
$ kubectl expose pod $CP_POD --name kong-status -n kong --port=8100
--target-port=8100
$ kubectl run tmp-shell --rm -i --tty --image nicolaka/netshoot
 tmp-shell  ~
```

Task: Get Prometheus Plugin Metrics through API

Run http get command to get the information from inside k8s to the ClusterIP:

```
$ http kong-dp-status.kong-dp.svc.cluster.local:8100/metrics | grep httpbin
```

```
...
kong_http_status{service="mockbin",route="mockbin",code="200"} 11
kong_http_status{service="mockbin",route="mockbin",code="429"} 16
...
kong_latency_bucket{service="mockbin",route="mockbin",type="kong",le="00001.0
"} 27
...
kong_latency_bucket{service="mockbin",route="mockbin",type="upstream",le="300
00.0"} 11
...
kong_latency_count{service="mockbin",route="mockbin",type="upstream"} 11
...
```

Task: Get Prometheus Plugin Metrics through API

Now let's take a look at generated metrics on CP:

```
$ http kong-status.kong.svc.cluster.local:8100/metrics | grep httpbin
```

There are no hits for httpbin, as the plugin records and exposes metrics at the node level, and the generated traffic was for DP.

Let's take a look at generated metrics on CP without filtering for httpbin service:

```
$ http kong-status.kong.svc.cluster.local:8100/metrics
```

You will see a variety of metrics provided, related to DB, license, shared dicts, connections, timers etc.

Exit out of our netshoot container

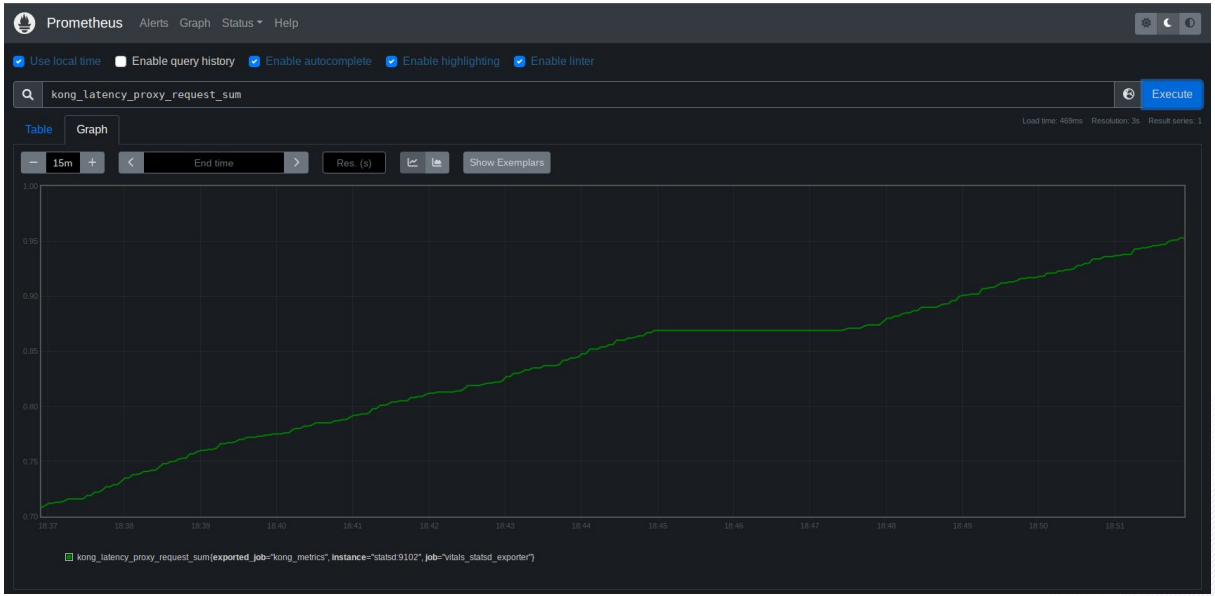
```
$ exit
```

Task: Get Metrics from Prometheus

1. Get the URI for accessing Prometheus from the environment variable and open it in a browser tab:

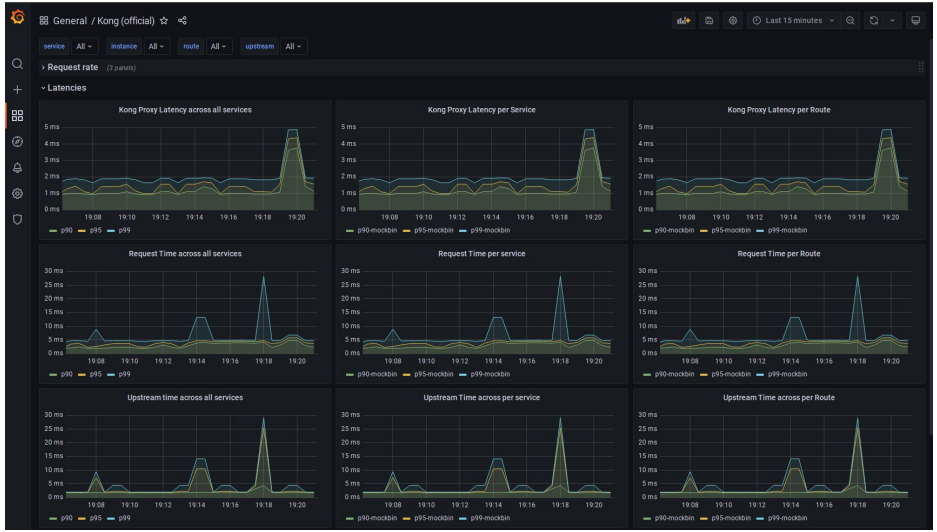
```
$ echo $PROMETHEUS_URL
```

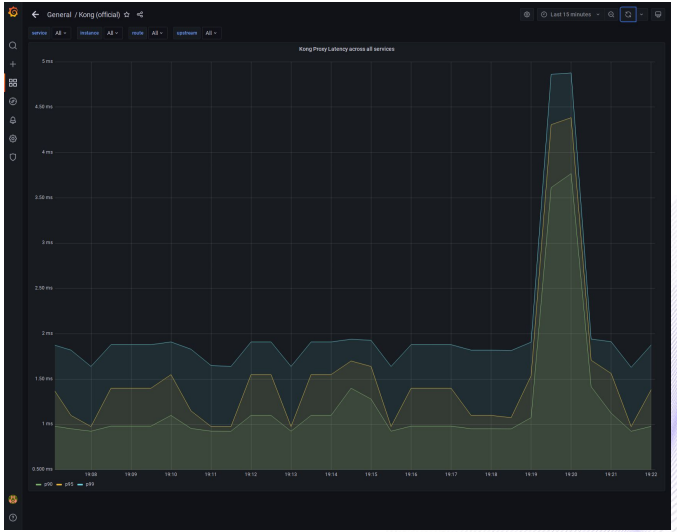
2. Query a data item such as kong_latency_proxy_request_sum
3. Visualise it over a period of time such as 15 minutes as a graph



Task: Get Metrics from Grafana

1. Switch to the browser tab for Grafana - Login with **admin** and **prom-operator**
2. Change the time range on the Kong Dashboard to Last 15 minutes
3. Observe Available graphs
4. Select Latencies > Kong Proxy Latency across all services > View
5. Observe the graph for comparing 90, 95 and 99 percentiles for delay





Summary

In this lesson we saw how to explain, configure, and administer

- Rate Limiting Advanced Plugin
- Request Transformer Advanced Plugin
- Response Transformer Advanced Plugin
- jq Plugin
- Exit Transformer Plugin
- Prometheus Plugin

Questions?

What's next?

In the next lesson we will look at some troubleshooting techniques.