

Kong Gateway Operations for Kubernetes

Troubleshooting

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

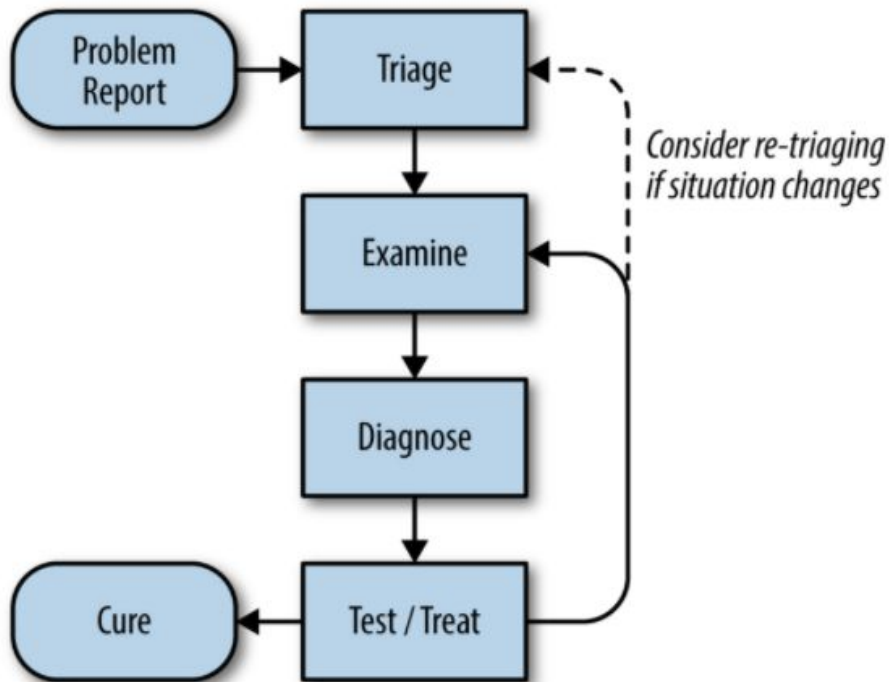
1. Troubleshooting Methodology & Techniques
2. Distinguishing between downstream and upstream networking issues
3. Collecting troubleshooting information
4. Identifying common issues

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
```

Troubleshooting Methodology



"Be warned that being an expert is more than understanding how a system is supposed to work. Expertise is gained by investigating why a system doesn't work."

Brian Redman

Problem Report

Problem Report comes in different forms

- System is Slow
- System is Down
- API connection fails
- Unexpected Behaviour
- Unexpected Return code

Useful details

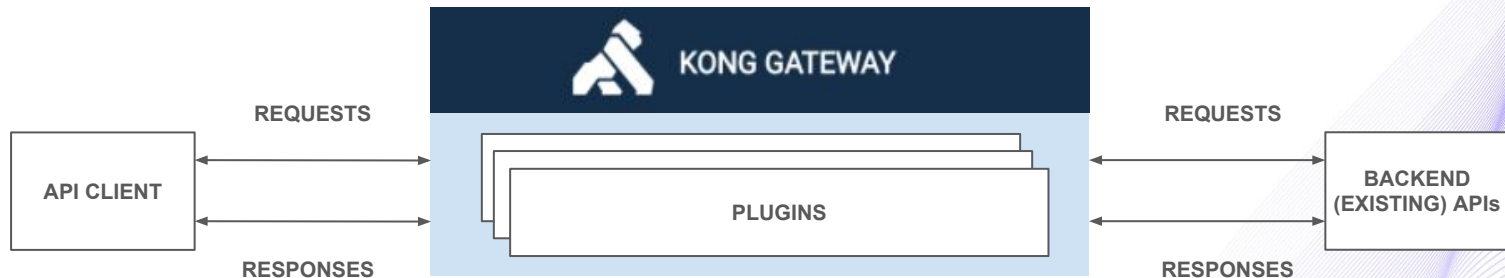
- What is the current behaviour?
- What is the expected behaviour?
- Was this working before?
- What changed?

Points of Failure

There are many points on the Kong request/response path that a transaction could fail, but they can be categorised into three main areas

1. Between the Client and the Kong Gateway (downstream connection)
2. Between the Kong gateway and backend APIs (upstream connection)
3. Kong Gateway in the middle, including its plugins

The first two of these may be due to network reliability, or due to use of wrong certificates



REST API Troubleshooting Checklist

- Is the URL correct?
- Is the API key valid?
- Is the authorization header valid?
- Is the correct HTTP method being used?
- Does the API user have feature access?
- Is the SSL certificate valid and included in your trusted store?
- Are the request parameters valid?
- Is the API code able to process the returned error codes?
- Does the API response coding follow best practices?
- Did the API call return the expected results?
- Did the API call return data with the expected performance?
- Does returned HTTP Status Codes (400x/500x) provide insight?

Diagnosis and Information Gathering

We will invoke different troubleshooting techniques depending upon where things go wrong on the request/response path.

Some fixes are just trial & error, however it's important to consider:

- The thought process on approaching an issue in a Kong deployment, and
- What information you can gather that could help in fixing the problem

There are a number of tools available for gathering information when troubleshooting issues:

- API Testing Tools: Insomnia, HTTPie, cURL, Postman.
- Web Debugging Proxy Apps: Fiddler, Charles, HTTP Toolkit, Proxyman
- Network Traffic Analyzers: tcpdump, Wireshark, Sysdig, CloudShark

In this section we will look at a number of techniques & methodologies you can use to either diagnose and fix the problem, or to gather information for Kong Support.

Test/Treat/Cure

- Possible Causes? Has there been changes in the environment?
- Reproducible? Retrace steps from start to finish & see if/where it breaks.
- Network connectivity? Is it caused by hosts, or connecting medium?
- Performance issues? CPU/Memory/DB etc., load testing.

Ensure to make note of the tests & results which will help to isolate the issue.

- Tests are used to narrow down possible causes to AFAP.
- Once cause(s) is/are defined, Root Cause Analysis & Solution Identification are next.
- Kong Docs and Knowledge Center is first stop for RCA and SI:
 - <https://docs.konghq.com>
 - <https://support.konghq.com/support/s/knowledge>

Triage, Examine, Diagnose

What is the business impact?

- Urgent
- High
- Normal
- Low

How many users are affected?

Where does the issue happen
(Frontend/Backend)?

Is it reproducible on demand? What are
the steps?

Are any error/warning(s) in the logs?

- `/usr/local/kong/logs/access.log`
- `/usr/local/kong/logs/admin_access.log`
- `/usr/local/kong/logs/error.log`
- `/usr/local/kong/logs/admin_gui_access.log`

Save off log files which contain the time of issue happening

Save off the configuration **`http GET <kong_host>:30001`**

Check Status of Kong **`http GET <kong_host>:30001/status`**

Check Metrics **`http GET <kong_host>:30001/metrics`**

Please Note that logs are usually in `/usr/local/kong/logs`, but on our lab system they reside in `/srv/shared/logs`.

Business Impact & Support SLAs

Kong Support Plans				
Initial Response Severity Based SLAs	SLA Description	Pro	Business	Platinum
		7AM-7PM* PST, GMT and SGT	7AM-7PM* PST, GMT and SGT	
Urgent	Kong Software is inoperable or down in Customer's production system / environment or slowed to such a degree that requests do not go through, having critical impact on Customer's business.	4 Business Hours	2 Business Hours	2 Hours, 24x7x365
High	Kong Software is operational but has a severe loss or restricted functionality in Customer's production system / environment, or a total system outage on Customer's staging system / environment, causing a significant impact on Customer's business.	24 Business Hours	4 Business Hours	4 Hours, 24x7x365
Normal	Kong Software is operational but has a minor loss of functionality (with or without a workaround) in Customer's production or staging system / environment, causing low or no impact on Customer's business. General questions regarding Software functionality.	48 Business Hours	8 Business Hours	4 Business Hours
Low	Documentation errors and Software feature requests.	96 Business Hours	16 Business Hours	8 Business Hours

*Business Hours means 7 a.m to 7 p.m. Monday to Friday excluding banking or statutory holidays in San Francisco (PST/PDT), London (GMT/BST), or Singapore (SGT) time zones. Applicable time zone is determined based on the region (North/South America, EMEA, AP/ANZ) of the Customer's address in the applicable Order.

Kong Information Collection

For troubleshooting Kong related issues try to gather the following Information

- **:30001/**
- **:30001/status**
- **error.log**

We'll talk through each of these in more detail.

Also Config dumps (via `kong config db_export`) and `deck` can be very useful if there's a particular config scenario that Kong support is trying to replicate locally.

Reproducible examples & steps (if possible) are the single best information to provide, so Kong engineers can do a reproduction in their environment and do Root Cause Analysis.

Task: Gather :30001/ information

```
$ http GET kongcluster:30001 | jq '.' > 30001.json
```

```
$ jq -C '.' 30001.json | less -R
```

Outputs useful information that will give a basic understanding of customer's deployment, e.g.:

- Kong configuration (kong.conf), sensitive data such as database passwords removed
- Loaded plugins
- Lua version
- Timer statistics

Task: Gather :30001/status information

```
$ http GET kongcluster:30001/status | jq '.' > 30001-status.json
```

```
$ jq -C '.' 30001-status.json | less -R
```

Outputs health information related to this particular Kong instance, including:

- Database reachability
- Lua shared dicts memory capacity/allocation (caution: not direct representation of usage)
- Lua VM memory usage (caution: requires proxy request to be updated)
- Connection statistics

Kong Error Log Files

Log files are essential to troubleshooting issues that arise, specifically the error related ones. Storage path and naming depends on Kong's configuration. The granularity of these logs is adjusted by the `log_level` property in the helm chart. Here is a list of error log files' description and default path.

- **proxy_error_log**: Path for proxy port request error logs
 - **logs/error.log**
- **proxy_stream_error_log**: Path for TCP streams proxy port request error logs
 - **logs/error.log**
- **admin_error_log**: Path for Admin API request error logs
 - **logs/error.log**
- **status_error_log**: Path for Status API request error logs
 - **logs/status_error.log**
- **admin_gui_error_log**: Path for Kong Manager error logs
 - **logs/admin_gui_error.log**
- **portal_gui_error_log**: Path for Developer Portal GUI error logs
 - **logs/portal_gui_error.log**
- **portal_api_error_log**: Path for Developer Portal API error logs
 - **logs/portal_api_error.log**

Kong Error Log Files

Log files are essential to troubleshooting issues that arise, specifically the error related ones. Storage path and naming depends on Kong's configuration. The granularity of these logs is adjusted by the `log_level` property. Here is a list of error log files' description and default path.

- **proxy_error_log**: Path for proxy port request error logs (**logs/error.log**)
- **proxy_stream_error_log**: Path for TCP streams proxy port request error logs (**logs/error.log**)
- **admin_error_log**: Path for Admin API request error logs (**logs/error.log**)
- **status_error_log**: Path for Status API request error logs (**logs/status_error.log**)
- **admin_gui_error_log**: Path for Kong Manager error logs (**logs/admin_gui_error.log**)
- **portal_gui_error_log**: Path for Developer Portal GUI error logs (**logs/portal_gui_error.log**)
- **portal_api_error_log**: Path for Developer Portal API error logs (**logs/portal_api_error.log**)

Kong Access Log Files

Access log files could be essential for troubleshooting issues that arises. Storage path and naming depends on Kong's configuration. The granularity of these logs is adjusted by the `log_level` property.

- **proxy_access_log**: Path for proxy port request access logs
 - **logs/access.log**
- **proxy_stream_access_log**: Path for TCP streams proxy port request access logs
 - **logs/access.log**
- **admin_access_log**: Path for Admin API request access logs
 - **logs/admin_access.log**
- **status_access_log**: Path for Status API request access logs
 - Default: **off**
- **admin_gui_access_log**: Path for Kong Manager access logs
 - **logs/admin_gui_access.log**
- **portal_gui_access_log**: Path for Developer Portal GUI access logs
 - **logs/portal_gui_access.log**
- **portal_api_access_log**: Path for Developer Portal API access logs
 - **logs/portal_api_access.log**

Task: Kong Log Files in Kubernetes

The log files are set to end all output to stdout by default via the helm chart values file. Because of this, all you have to do is run `kubectl logs` to get the information.

```
$ kubectl describe deployment kong-kong -n kong | grep _LOG | sort | uniq
KONG_ADMIN_ACCESS_LOG:          /dev/stdout
KONG_ADMIN_ERROR_LOG:           /dev/stderr
KONG_ADMIN_GUI_ACCESS_LOG:       /dev/stdout
KONG_ADMIN_GUI_ERROR_LOG:        /dev/stderr
KONG_PORTAL_API_ACCESS_LOG:      /dev/stdout
KONG_PORTAL_API_ERROR_LOG:       /dev/stderr
KONG_PORTAL_GUI_ACCESS_LOG:      /dev/stdout
KONG_PORTAL_GUI_ERROR_LOG:       /dev/stderr
KONG_PROXY_ACCESS_LOG:           /dev/stdout
KONG_PROXY_ERROR_LOG:            /dev/stderr
KONG_PROXY_STREAM_ACCESS_LOG:    /dev/stdout basic
KONG_PROXY_STREAM_ERROR_LOG:     /dev/stderr
KONG_STATUS_ACCESS_LOG:          off
KONG_STATUS_ERROR_LOG:           /dev/stderr
```

Kong Log Levels

debug: provides debug information about the plugin's runloop and each individual plugin or other components. Only to be used during debugging since it is too chatty.

info/notice: Both provide information about normal behavior, most of which can be ignored.

warn: To log any abnormal behavior that doesn't result in dropped transactions but requires further investigation, warn level should be used.

error: Used for logging errors that result in a request being dropped, e.g. HTTP 500 errors. The rate of such logs need to be monitored.

crit: This level is used when Kong is working under critical conditions and not working properly thereby affecting several clients.

Task: Explore Error Logs

```
$ ./edu-kgac-202/exercises/troubleshooting/kiclogs.sh  
$ ./edu-kgac-202/exercises/troubleshooting/dplogs.sh
```

```
time="2022-07-26T19:25:29Z" level=info msg="Starting workers" logger=controllers.TCPIngress  
worker count=1  
time="2022-07-26T19:25:38Z" level=info msg="successfully synced configuration to kong."  
time="2022-07-26T19:33:07Z" level=info msg="successfully synced configuration to kong."
```

The logs are provide extremely useful information when troubleshooting Kong related issues.

- Debug level preferred: KONG_LOG_LEVEL=debug
- When enabled, the entire Kong request processing flow can be tracked
- Generates a lot of log lines
- Highly compressible
- **May contain PII** (e.g. request lines)

Kong System Logs

By default, kong system logs are redirected to stdout. You can change the redirection to a file, for a more thorough review and use in troubleshooting. In order to do that, you'll have to redirect the output generated by **kong** to a log file of your choice. You can also add **--vv** to get a high level of verbosity for debugging.

System logs could be quite useful in troubleshooting issues caused by the startup configuration of Kong and/or its execution on the underlying platform.

Reviewing the startup log is a practical way to see details on the configuration that is actually loaded into the memory for execution.

Kong System Logs

By default, kong system logs are redirected to stdout, although you can redirect to a file

You can also add `--vv` to get a high level of verbosity for debugging.

System logs could be quite useful in troubleshooting issues caused by the startup configuration of Kong and/or its execution on the underlying platform.

Reviewing the startup log is a practical way to see details on the configuration that is actually loaded into the memory for execution.

Task: Explore Startup System Log

Use 'kong reload' command to observe startup log in 3 different levels of verbosity. Find out the value of audit_log which controls audit logging discussed later in this lesson.

```
$ CP_POD=`kubectl get pods --selector=app=kong-kong -n kong -o  
jsonpath='{.items[*].metadata.name}'`  
$ kubectl exec $CP_POD -it -n kong -c proxy -- /bin/sh  
$ kong reload --v  
$ kong reload --vv
```

Here is the output for the verbose (--v) level:

```
2022/05/20 21:37:56 [verbose] Kong: 2.8.1.0-enterprise-edition  
2022/05/20 21:37:56 [verbose] prefix in use: /usr/local/kong  
2022/05/20 21:37:56 [verbose] reading config file at /usr/local/kong/.kong_env  
2022/05/20 21:37:56 [verbose] prefix in use: /usr/local/kong  
2022/05/20 21:37:56 [info] Kong reloaded
```

```
$ exit
```

Kong Debug Header & Granular Tracing

You can pass the debug header **Kong-Debug** with the request, and get these headers indicating the service/route used to proxy the request:

- Kong-Route-Id
- Kong-Route-Name
- Kong-Service-Id
- Kong-Service-Name

Granular tracing offers a mechanism to expose metrics and detailed debug data about the lifecycle of Kong in a human, or machine, consumable format.

Details on how to configure granular tracing can be found here:

<https://docs.konghq.com/gateway/2.8.x/reference/configuration/#granular-tracing-section>

Task: Use Debug Header to see used Service/Route

Let us set up a service first:

```
$ kubectl apply -f ~/edu-kgac-202/base/httpbin-ingress.yaml
namespace/httpbin-demo created
service/httpbin-service created
ingress.networking.k8s.io/httpbin-ingress created
```

Now let us use the debug header in request to see used service/route in response headers:

```
$ http -h GET kongcluster:30000/httpbin Kong-Debug:1
```

```
. . .
Kong-Route-Id: c35f1223-7f45-4abb-b8dc-4977bbcef3bc
Kong-Route-Name: httpbin-demo.httpbin-ingress.00
Kong-Service-Id: fe6acef1-4440-497c-98c9-0801af977489
Kong-Service-Name: httpbin-demo.httpbin-service.pnum-80
```

Task: Use Granular Tracing to Log Details

Let us set up key authentication for a consumer on the service:

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

Let us review granular tracing configuration we have included in Kong configuration:

```
$ cat ~/edu-kgac-202/base/dp-values.yaml | grep tracing
tracing: on
tracing_writing_strategy: file
tracing_types: all
tracing_time_threshold: 0
tracing_write_endpoint: /dev/stdout
tracing_debug_header: X-Trace
```

Task: Use Granular Tracing to Log Details

Next we will send requests to the service, including X-Trace header to get details logged:

```
$ http -h GET kongcluster:30000/httpbin apikey:JanePassword X-Trace:1 | head -1
```

```
HTTP/1.1 200 OK
```

```
$ http -h GET kongcluster:30000/httpbin X-Trace:1 | head -1
```

```
HTTP/1.1 401 Unauthorized
```

As expected, first request results in a success, and second one fails. Looking at the generated log file, you will see detailed information on the flow of the request through Kong, which are quite useful for troubleshooting a large number of issues such as time spend in each phase, plugins used to handle the request etc.

```
$ ~/edu-kgac-202/exercises/troubleshooting/dplogs.sh
```

Kong Audit Logs

When Audit Logging is enabled, Kong will store detailed audit data regarding Admin API and DB access in the datastore. Details on storage and maintenance of audit records are available in documentation:

<https://docs.konghq.com/gateway/latest/admin-api/audit-log/#main>

<https://docs.konghq.com/gateway/latest/reference/configuration/#data--admin-audit-section>

Updates to the DB are mostly associated with Admin API requests. Consequently, DB object audit log data is tied to a given HTTP via a unique identifier, providing built-in association of Admin API and DB traffic.

Granular logging facility on Kong Admin API allows cluster administrators to keep detailed track of changes made to the cluster configuration throughout its lifetime, aiding in compliance efforts and providing valuable data points during forensic investigations.

Task: Explore Audit Logs

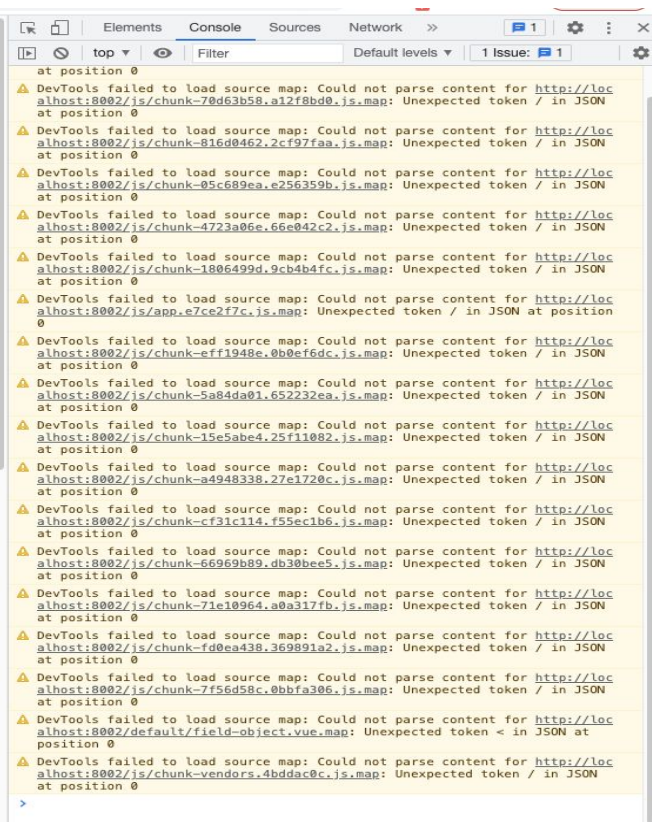
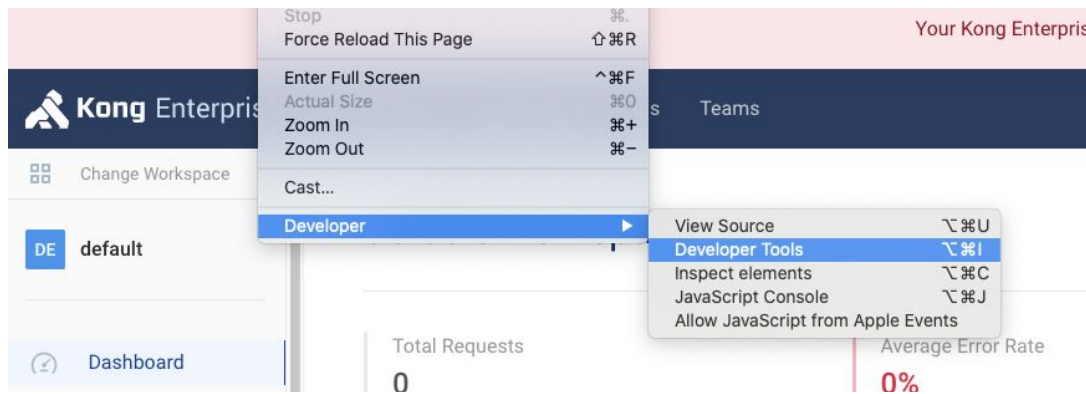
As we saw before. Audit Logging is already enabled. Create a request to admin API and observe the audit log generated for it.

```
$ http GET kongcluster:30001/license/report
```

```
$ http GET kongcluster:30001/audit/requests
```

```
...{
  "client_ip": "172.24.0.47",
  "method": "GET",
  "path": "/license/report",
  "payload": null,
  "rbac_user_id": null,
  "removed_from_payload": null,
  "request_id": "7Z1aPDfssXr0mavaTxAdQd1C7DVzI876",
  "request_timestamp": 1653085837,
  "signature": null,
  "status": 200,
  "ttl": 2591986,
  "workspace": "d99b1ba4-2b84-4f76-a1a6-65d69a0efa27"
}...
```

Using HAR files for Troubleshooting



If the issue is observable in browser, you may wish to generate and supply a HAR file, i.e. HTTP session information as a JSON object. Further information available at:

https://toolbox.googleapps.com/apps/har_analyzer/

<https://support.zendesk.com/hc/en-us/articles/204410413-Generating-a-HAR-file-for-troubleshooting>

Network Troubleshooting

Downstream or Upstream Issue?

Sometimes it's clear where the issue is when you make a request to your Kong instance.

If the issue is on the upstream side, then Kong will return an error message, so if a client reports something is not right, then the issue is generally on the downstream side.

Downstream (Client <-> Kong)	Upstream (Kong <-> Services)
Connection could not be made (Connection reset by peer, timeout etc)	50x errors returned by Kong
Client reports TLS handshake failures	Kong error logs contains lines related to the request
Kernel error messages	Kernel error messages

More detailed analysis may require data packet capture, but narrowing it down early can help identify the port/IP to filter on, reducing the size of the capture file to analyse.

Task: Network troubleshooting with cURL

cURL can also be used to get more information on possible connectivity issues, for example:

```
$ curl -svv --fail --trace-time https://api.github.com/repos/kong/kong | jq
```

Looking at the output, you'll see detailed information about the handshake, certificates, request/response data, timing of each step etc.

```
$ curl -o /dev/null --silent --show-error --write-out '\n\nlookup: %{time_namelookup}\nconnect:
%{time_connect}\nappconnect: %{time_appconnect}\npretransfer: %{time_pretransfer}\nredirect:
%{time_redirect}\nstarttransfer: %{time_starttransfer}\ntotal: %{time_total}\nsize: %{size_download}\n\n'
'https://api.github.com/repos/kong/kong'
```

The above command gives you detailed timing information for a request, such as TTFB, time to first byte, DNS resolution time, TLS handshake time etc. Further details available here:

<https://blog.cloudflare.com/a-question-of-timing>

Common Network Issues

The following are common indications of network issues, but there are more

- Slow proxy performance
- Failed TLS handshake
- Unexpected 500 Errors to upstream
- Client proxy page downloading slowly

Networking issues in general can be extremely difficult to troubleshoot.

You need a clear approach to know what information Kong really needs in order to solve the problem.

Other Issues

What else could possibly go wrong?

Apart from the network there are other areas from which errors may arise

- Misconfigurations
- Bugs in custom plugins
- Bugs in Kong
- Resource starvation
- Performance issues
- ...

By far the most common case is misconfigurations.

Method of exclusion - this can be achieved by good data collection.

Troubleshooting NGINX

Troubleshooting NGINX is beyond the scope of this course, but these YouTube video may help if you're interested

Tracing and Troubleshooting NGINX, OpenResty, and Their Backends

<https://www.youtube.com/watch?v=NR8U1qYHGqk>

Advanced NGINX Config, Performance Improvements, Caching, Clustering

<https://www.youtube.com/watch?v=WMsgw68Dhlq>

Task: Broken Lab Scenario 1 - 502 Error

Lab Setup

Before we go any further we need to set up the environment for the following troubleshooting lab

```
$ cd ~/edu-kgac-202/exercises/troubleshooting
$ kubectl delete namespace httpbin-demo
$ kubectl apply -f ./broken-lab-1.yaml
```

Lab Task

Sending a request to the service results in an error. Troubleshoot and fix the issue.

```
$ http --headers GET $KONG_PROXY_URI/httpbin?apikey=JanePassword | head -1
```

```
$ http --headers GET $KONG_PROXY_URI/httpbin?apikey=JanePassword
```

Broken Lab Scenario 1: Tips

Troubleshooting tips

- What does the 502 say about where the issue resides - upstream or downstream?
- Are there any clues in the logs?

```
$ ./dplogs.sh
```

- Are there any clues in the response body?
- Are there any clues when using curl to get details on the request/response?
- Are there any clues in the upstream application's logs?
- Examine your yamls again

Broken Lab Scenario 1: Solution

Problem

- `httpbin` Service is configured for HTTPS, however the upstream is configured for HTTP. Consequently SSL handshake attempts by Kong are failed by the upstream.

Solution

- Update `httpbin`'s service protocol setting to HTTP, using our beloved CRDs.

```
$ sed -i 's/konghq.com\/protocol: "https"/konghq.com\/protocol: "http"/g' ./broken-lab-1.yaml
$ kubectl apply -f ./broken-lab-1.yaml
```

Services › mockbin ›

Update Service [View docs](#)

☒ This service is Enabled

Name
mockbin

Protocol
http

Host
mockbin

Task: Broken Lab Scenario 2 - Headers Issue

Lab Setup

Before we go any further we need to set up the environment for the following troubleshooting lab

```
$ cd ~/edu-kgac-202/exercises/troubleshooting
$ kubectl delete namespace httpbin-demo
$ kubectl apply -f ./broken-lab-2.yaml
```

Lab Task

When issuing the request with X-with-ID:true header, we expected to X-Correlation-ID:uuid and X-MyCustom-Header:true echoed back from the upstream. Neither should be echoed back otherwise. Check the existing configuration and change it to get the expected result.

```
$ http GET $KONG_PROXY_URI/httpbin X-with-ID:true
```

Broken Lab Scenario 2: Tips

Troubleshooting tips

- Are there any clues in the logs?

```
$ ./dplogs.sh  
$ ./kiclogs.sh  
$ ./cplogs.sh
```

- Are there any clues in the response body?
- Are there any clues when using curl to get details on the request/response?
- Are there any clues in the route/plugin configuration?
- Examine your yamls again

Broken Lab Scenario 2: Solution

Problem: correlation Route has a correlation-id plugin on it which adds X-Correlation-ID set to tracker. nocorrelation Route has a request-transformer plugin on it which adds X-MyCustom-Header.

Solution: Modify correlation-id plugin to set X-Correlation-ID to uuid. Remove request-transformer plugin from nocorrelation Route

```
$ sed -i 's/generator: tracker/generator: uuid/g' ./broken-lab-2.yaml  
  
$ sed -i '/konghq.com\/plugins: "httpbin-request-transform"/d' ./broken-lab-2.yaml  
  
$ kubectl apply -f ./broken-lab-2.yaml
```

Task: Broken Lab Scenario 3 - Plugin Issue

Lab Setup

Before we go any further we need to set up the environment for the following troubleshooting lab

```
$ cd ~/edu-kgac-202/exercises/troubleshooting
$ kubectl delete namespace httpbin-demo
$ kubectl apply -f ./broken-lab-3.yaml
```

Lab Task

When we try to reach the service anonymously, we expect to get a 403. However, we are getting 200s. Jane is currently our only consumer with a key auth so she is essentially the only one that should be able to access the service.

```
$ http GET $KONG_PROXY_URI/httpbin
$ http GET $KONG_PROXY_URI/httpbin apikey:JanePassword
```

Broken Lab Scenario 3: Tips

Troubleshooting tips

- Are there any clues in the logs:

```
$ ./dplogs.sh  
$ ./kiclogs.sh  
$ ./cplogs.sh
```

- Are there any clues in the response body?
- Are there any clues when using curl to get details on the request/response?
- Are there any clues in the route/plugin configuration?
- Examine your yamls again

Broken Lab Scenario 3: Solution

Problem: In the KIC logs we see that the plugin cannot be found. After examining the yaml again in further detail, we see that the plugin was created in the default namespace instead of the httpbin-demo namespace.

Solution: Change the configuration of the yaml to specify the correct namespace so KIC can find it and apply it to the Service and Route.

```
$ sed -i 's/namespace: default/namespace: httpbin-demo/g' ./broken-lab-3.yaml

$ kubectl apply -f ./broken-lab-3.yaml

$ kubectl delete kongplugin httpbin-auth -n default

$ http GET $KONG_PROXY_URI/httpbin

$ http GET $KONG_PROXY_URI/httpbin apikey:JanePassword
```

Broken Lab: Summary

Key Learnings

- 502 error indicates the issue is on the upstream side.
- Clues are in the logs.
- Network / Request tracing is key to stepping into a user's shoes.
- The right toolkit available, and the right knowledge, makes for faster learning.
- KIC logs contain information about incorrect CRD configs and problems pushing to data planes

Summary

In this lesson we learned

- How to troubleshoot
- Different troubleshooting methodology & techniques
- How to distinguish between downstream and upstream networking issues
- How to collecting troubleshooting information
- How to identifying common issues

Questions?

What's next?

In the last part of the course, a knowledge check for the covered material is provided.

