**Microservices & API Gateways, Part 2: How Kong Can Help**



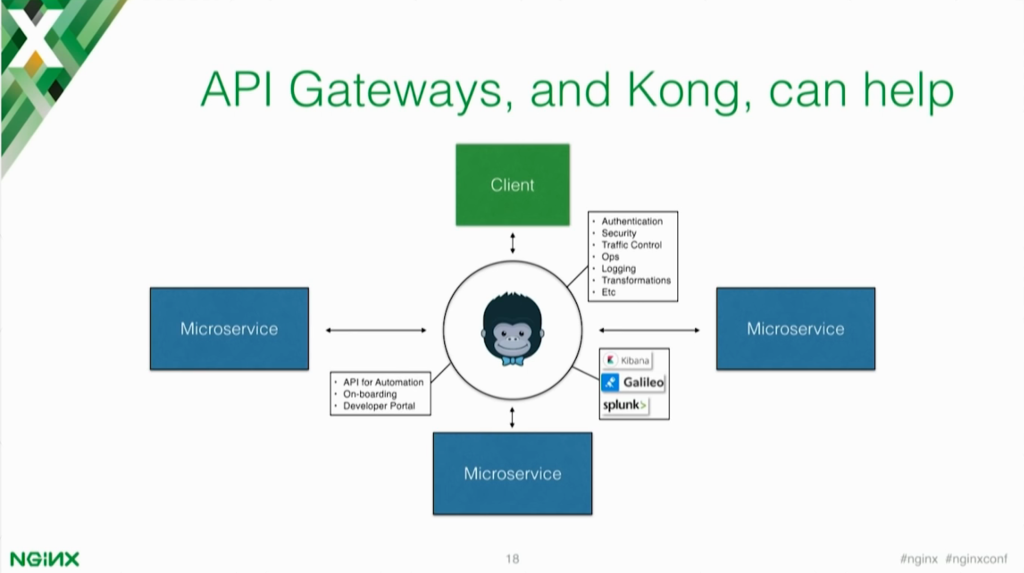
*This post is adapted from a presentation by Marco Palladino of Mashape.com at nginx.conf in September 2016. You can view a recording of the presentation on* [*YouTube*](https://www.youtube.com/watch?v=zs_QyRTIpDM)*.*

*This post the second of two parts, and focuses on how Mashape.com’s API gateway, Kong, can fit your microservices architecture.* [*Part 1*](https://www.nginx.com/blog/microservices-api-gateways-part-1-why-an-api-gateway/) *highlights the core difference between monolithic and microservices architectures and shows to set up an API gateway for additional functionality.*

**Table of Contents**

|  |  |
| --- | --- |
|  | [Part 1](https://www.nginx.com/blog/microservices-api-gateways-part-1-why-an-api-gateway/) |
| 23:52 | [API Gateways and Kong Can Help](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#APIGatewaysandKongCanHelp) |
| 25:49 | [What is Kong?](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#WhatisKong) |
| 26:09 | [What Does Kong Do?](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#WhatDoesKongDo) |
| 26:35 | [Kong Plug‑ins](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#KongPlugins) |
| 27:22 | [Kong = OpenResty + NGINX](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#KongOpenRestyNGINX) |
| 30:26 | [NGINX Configuration](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#NGINXConfiguration) |
| 32:53 | [Kong Entry-Points](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#KongEntryPoints) |
| 33:30 | [Core Entities](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#CoreEntities) |
| 34:34 | [Plugins Configuration Matrix](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#PluginsConfigurationMatrix) |
| 35:18 | [Multi-DC Deployment](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#MultiDCDeployment) |
| 37:27 | [Demo Time](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#DemoTime) |
| 41:14 | [Questions](https://www.nginx.com/blog/microservices-api-gateways-part-2-how-kong-can-help/#Questions) |

**23:52 API Gateways and Kong Can Help**



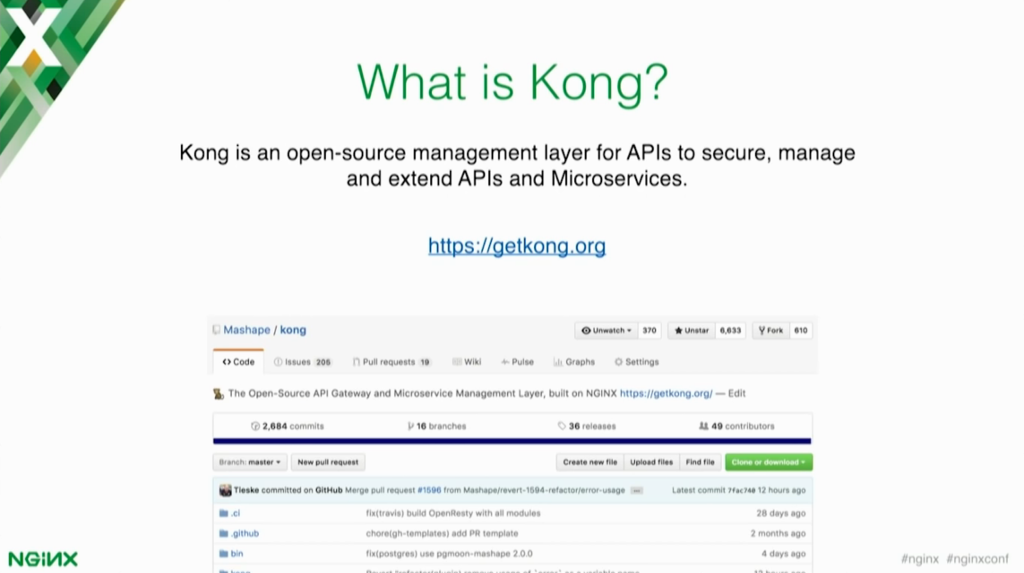
**Palladino:** API gateways can fix some of [these problems](https://www.nginx.com/blog/microservices-api-gateways-part-1-why-an-api-gateway/#DivisionofLabor) by implementing those middle functions that you don’t have to re-implement again in those services. Sometimes different teams implement different microservices in different ways.

If you don’t have a centralized way of doing things, you will end up with teams doing authentication and rate limiting in a different way than another team. You want to avoid having this fragmentation.

An API gateway can also help fix not only the management part of an API, but can also the other two missing things that we have to do.

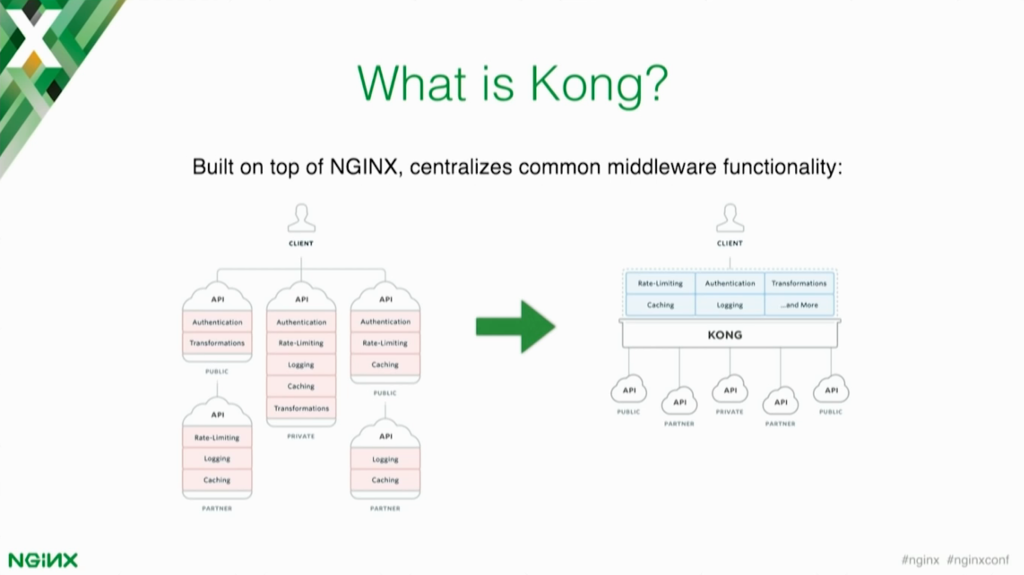
* **Analytics** – A gateway can communicate transparently with your analytics infrastructure because the gateway is the entry point for every request. The gateway sees all of the data and the traffic that’s going through your services, so you have one central place where you can push all of this information to your monitoring or analytics solution like Kibana or Splunk.
* **Automation** – The gateway can also help automate deployments, but also documentation and onboarding. What is onboarding? If you have an API and you have authentication, you will need something that allows users to create credentials for that API and start consuming the API. The developer portal or your documentation can integrate with the gateway to provision these credentials so you don’t have to build something from scratch.

**25:49 What is Kong?**



So what is Kong? It’s an open source API gateway, or management layer for APIs, that you can use for implementing extra features on top of those upstream services. Kong is open source, so it’s available on [GitHub](https://github.com/Mashape/kong). You can [download](https://getkong.org) and use it today.

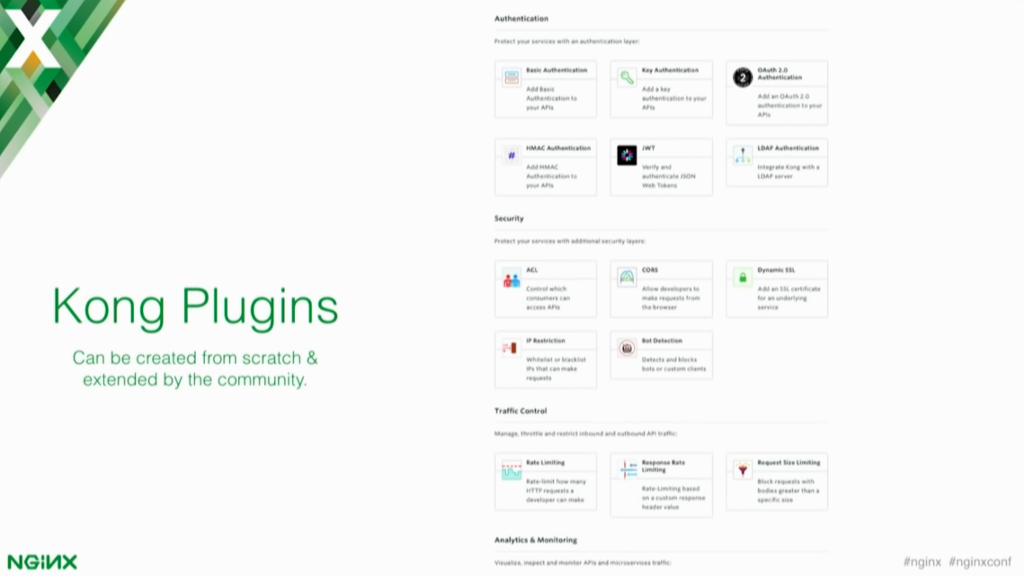
**26:09 What Does Kong Do?**



Kong centralizes all that fragmentation in one place. This shows exactly what I was talking about before – the fragmentation of having multiple services, each one with different implementations for common features.

An API gateway like Kong can centralize all of that in one place, which in turn makes development of those services even easier because you have less code to handle and maintain.

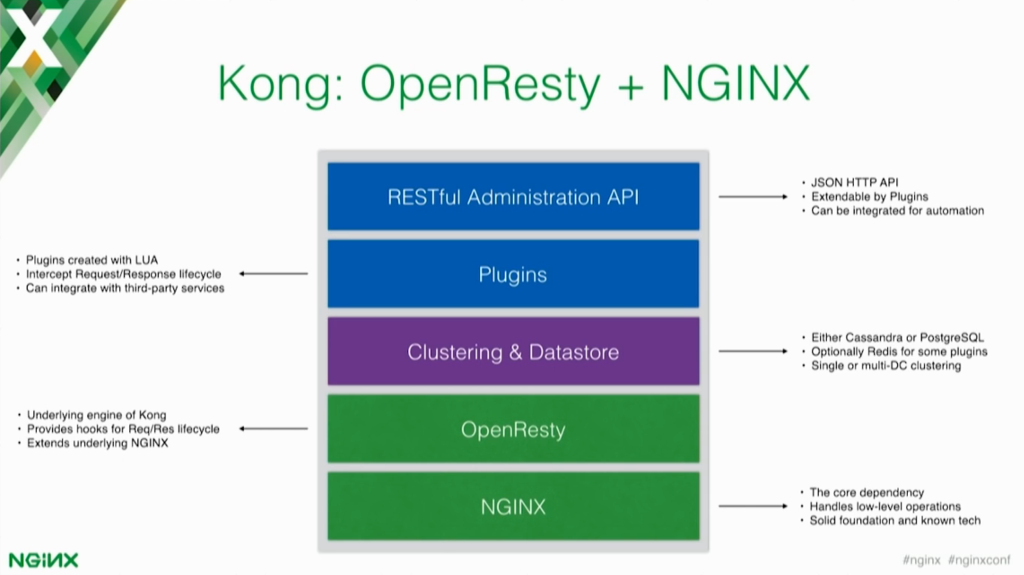
**26:35 Kong Plug‑ins**



So what are plug‑ins? Plug‑ins are the Kong middleware that you can add on top of those upstream services. Middleware plug‑ins can be anything from authentication to security, to traffic control, logging, or transformations.

You may have a SOAP service that you need to make available with a RESTful interface. An API gateway like Kong can implement that transformation layer so you don’t have to ask your team to change the implementation of the API to do the transformation. The API gateway can implement the transformation for you.

**27:22 Kong = OpenResty + NGINX**



Technically speaking, Kong is an OpenResty application. OpenResty runs on top of NGINX and extends NGINX with Lua. Lua is a very easy‑to‑use scripting language that allows you to script the things you can do in NGINX.

OpenResty provides hooks for different events in the API request‑and‑response life cycle, so you can write Lua scripting code that can hook into those events – for example, a new request is coming in, a new request is about to be proxied, a response came back. You can write custom code for each one of these events and you can change requests and responses on the fly.

[In the architecture] we have NGINX at the bottom which handles the low‑level features of Kong. All of the [proxying](https://www.nginx.com/resources/glossary/reverse-proxy-server/) is being done by NGINX – it’s a very solid technology. OpenResty is the underlying core of Kong. it extends NGINX with these new capabilities, and Kong on top of those two technologies implements clustering, plug‑ins, and a RESTful API that you can use for managing the API gateway.

Like Elasticsearch, where you have an API for doing pretty much whatever you need to do with a data store, Kong exposes an API that allows you fully operate the system just by making HTTP calls and parsing a JSON response.

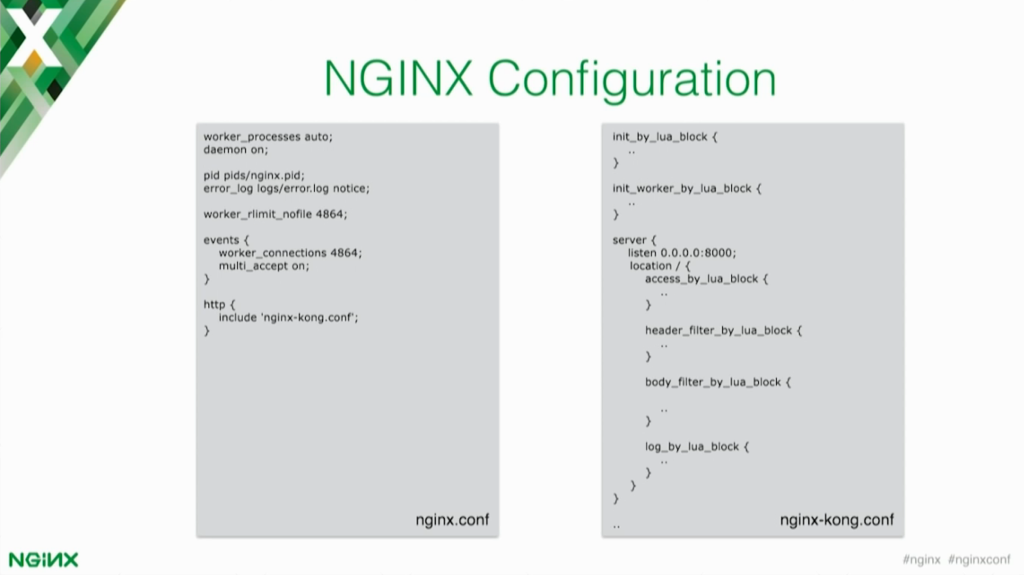
This means you can integrate Kong with your a href=”https://www.nginx.com/resources/glossary/devops/”>DevOps and automation tools. You can also integrate Kong with third‑party services, developer portals, and onboarding tools. Kong stores all information in either PostgreSQL or Cassandra, and depending on the use case you may want to use one or the other.

Cassandra is an eventually consistent data store. This means that if you have a cluster of Cassandra nodes and you’re storing data, eventually that data will be propagated to all other nodes – not at the same time [immediately].

So you can have, let’s say, a Cassandra cluster of three nodes in DC [data center] 1, a Cassandra cluster of another three nodes in DC 2, and then you link them together. Whatever operation you’re doing on one data center, it will be eventually replicated to the other data center.

PostgreSQL is easier to use. It supports master and slave, but it doesn’t support masterless multi‑DC replication; however, there are some tools that allow you to do that.

**30:26 NGINX Configuration**



Since Kong is built on top of NGINX, you are able to access the NGINX configuration and pretty much replace your existing NGINX implementation with Kong and still run the old functionality on top of Kong.

This is exactly how it works – we have an NGINX configuration that you can change, update, and modify. Then Kong comes with its own configuration that you can include inside of the NGINX configuration.

In the Kong configuration, it’s leveraging the OpenResty directives like … access\_by\_lua\_block and header\_filter\_by\_lua. Those are all of the events in the life cycle of a request or response that you can hook into when you are creating a plug‑in in Kong.

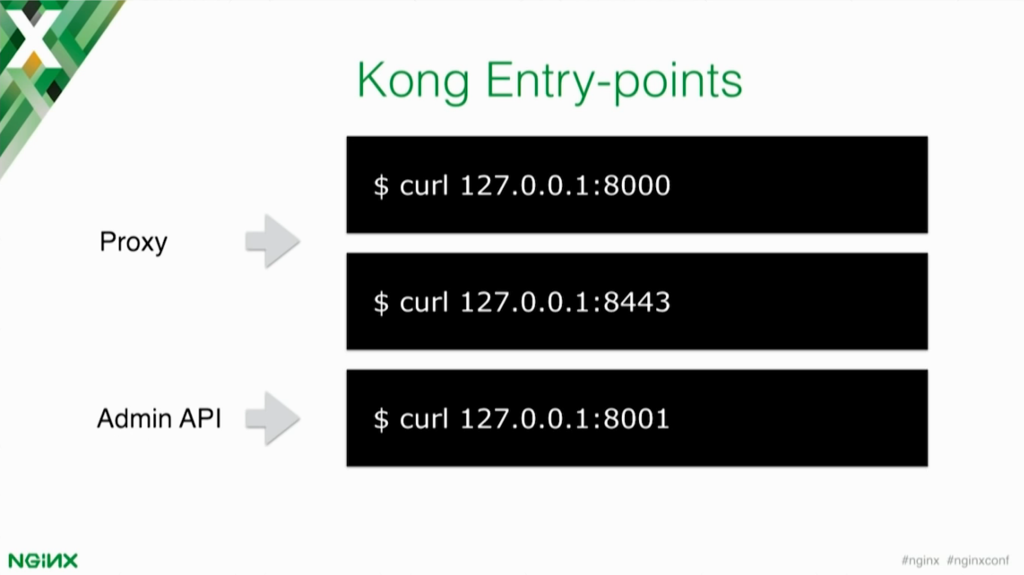
Kong already comes with some plug‑ins that either the community or Mashape has built, and some of them are available on the [getkong.com](https://getkong.org) website or on the GitHub repo. Also, the community has extended the system with their own plug‑ins. If you search on GitHub for “Kong plug‑ins [or plugins]”, you can find a plug‑in for other things that are not featured on the website.

On the website, we generally list just those plug‑ins we feel are stable enough to be used in production, but there are other are ready‑to‑use plug‑ins on GitHub. If you are planning to use Kong, I encourage you to check out what the community has built so that you don’t have to reinvent the wheel.

If you need to do something that’s very custom and very specific for your use case, you can of course extend Kong by creating your own plug‑in that you can shape privately inside your organization. The plug‑in will have access to all of these OpenResty events so you can change requests and responses on the fly.

You can also make requests to third‑party services. Let’s say you have a legacy authentication system that you want to integrate in the API gateway. You can create a plug‑in that makes a request to that authentication system on every request. Then you handle that [authentication] request in the plug‑in and then return a response so that the client only has to make a request to the API.

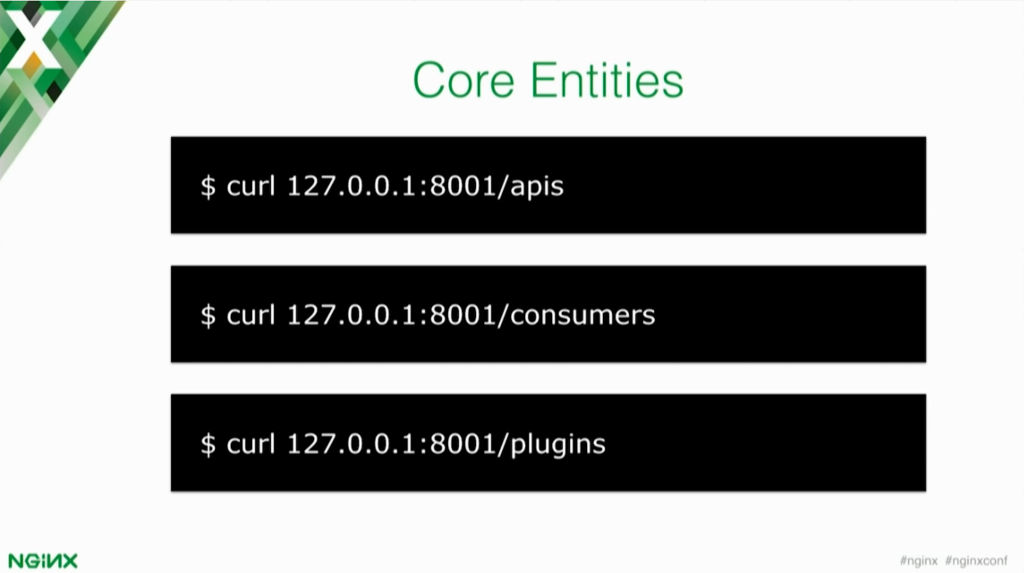
**32:53 Kong Entry Points**



There are two main entry points for Kong. The first is called the **Proxy** entry point, which means that consumers and clients that want to consume the upstream services are able to do that through the default port, 8000, or 8443 for SSL.

Then you have the **Admin API** [entry point], which is available on a different port, 8001. That is the API you can use for doing pretty much everything you have to do on the system.

**33:30 Core Entities**

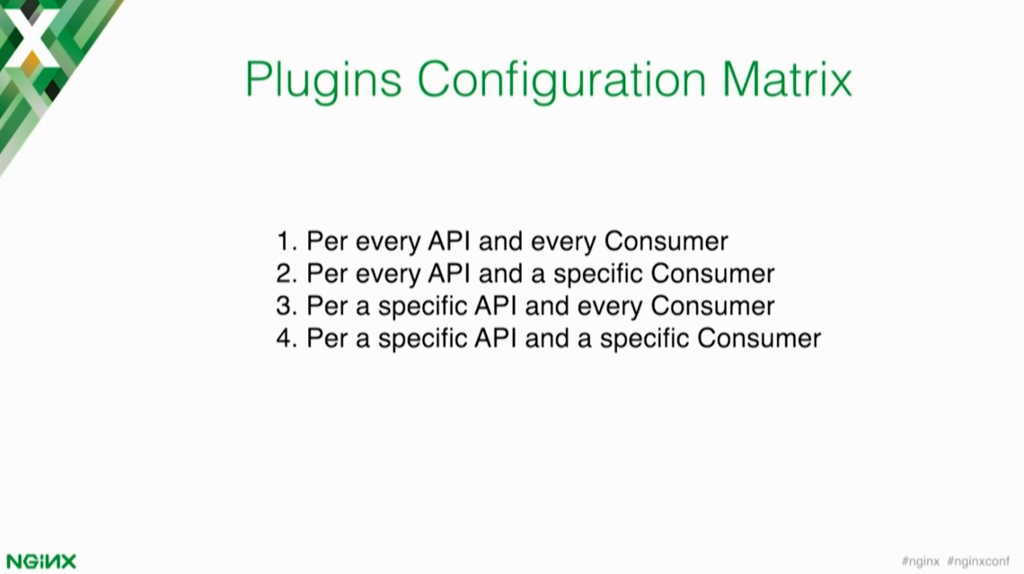


After this presentation, I will quickly show you how to use Kong via the terminal. Before doing that, I want to make you aware that Kong has three main core entities that you will always use in the API gateway. Those entities are APIs, which represent the upstream services that you’re trying to put behind Kong.

So we can have a thousand different services behind Kong and we call them **apis**.

You can have **consumers** – clients or individual developers, depending on your use case – that are going to consume those APIs. The consumer can be a client app, either internal to your organization or public, or a partner. Then we have **plugins** that you can apply on top of APIs and consumers to change how the middleware functionality works.

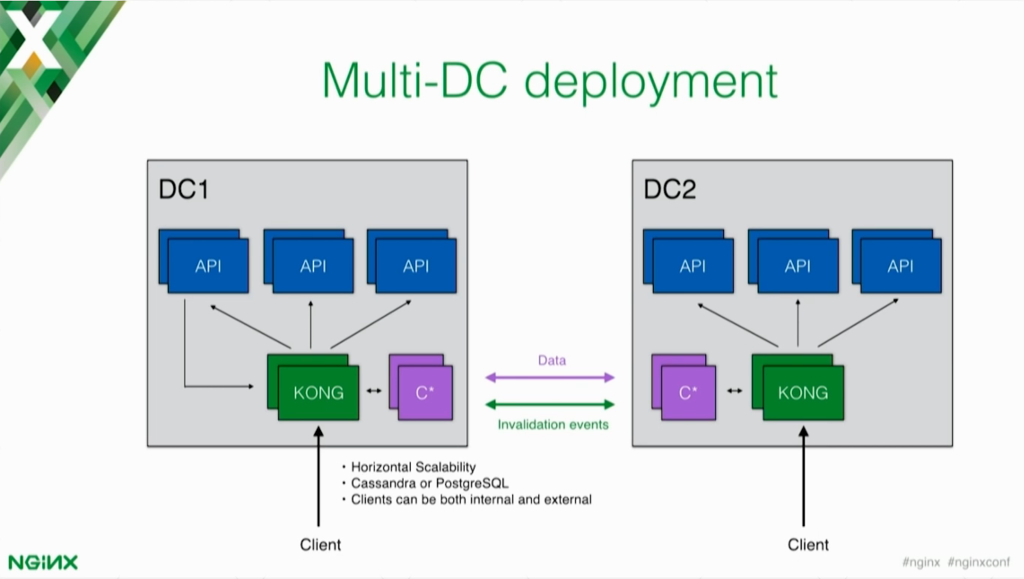
**34:34 Plug‑ins Configuration Matrix**



For example, I can have plug‑ins that are applied for every API and every consumer. Let’s say I want rate limiting for every service and I want to rate limit requests to 200 requests per second for every service.

Then you can have plug‑ins that are for every API, but for a specific consumer. So I want everybody to make 200 requests per second but then an internal app should not have any limit. You can do that for one consumer for example. Or you can do per API and every consumer. You can play with it in a very flexible way.

**35:18 Multi‑DC Deployment**



This is an example use case for a multi‑DC deployment. Kong is the entry point for all of the requests your clients are going to make. Kong accepts the request, figures out what kind of upstream service you’re trying to access, and tries to run the middleware associated to that API.

Some of the middleware can be authentication plug‑ins, which are executed first in the order of execution. Then once Kong knows which consumer is trying to consume the API, it can load dynamically all of this information.

Kong relies on the data store only for the first request. On the first request, Kong parses all of the information from the data store and then caches it in memory. So all of the other requests after the first one are handled in memory, which means that Kong can be very fast, not adding too much latency on the transaction.

If you have two different clusters, you need to somehow connect those data centers together. The information that’s going to be shared is going to either be data between the Cassandra nodes, or invalidation events.

Because Kong caches everything on the first request, what happens if you’ve made some changes on another node? How does the first Kong node know that the data isn’t valid anymore?

There are invalidation events that are being sent across the Kong nodes, so every time you perform an operation on one Kong node, for example changing the address of an upstream service, that node invalidates that one specific entity by sending an invalidation event for it to all the other Kong nodes.

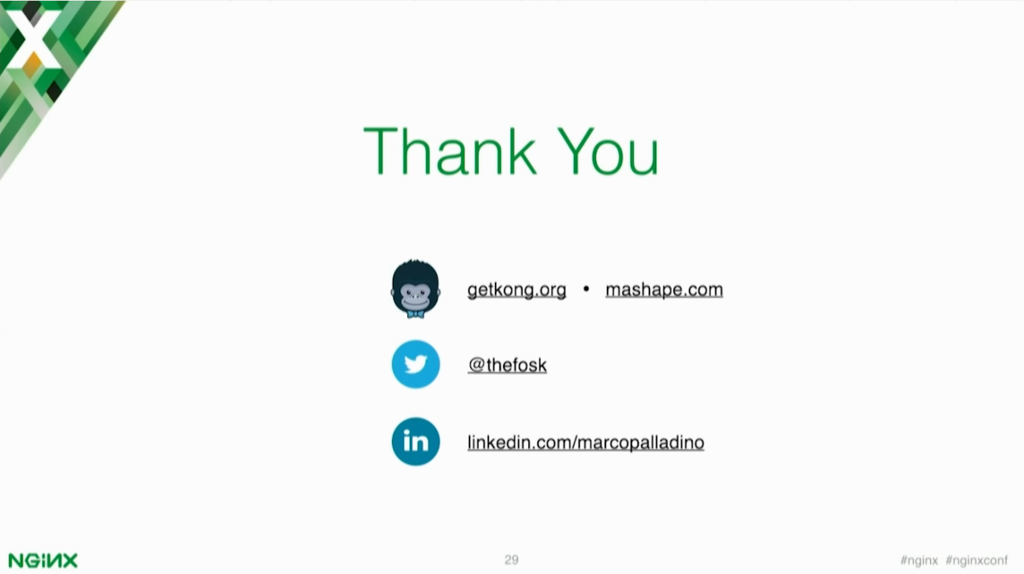
The other Kong nodes receive the invalidation event and delete the entity. When a new request comes in, Kong is forced to acquire the data again from the data store.

**37:27 Demo Time**



*Editor – The video below skips to the beginning of the demo at 37:27.*

**41:14 Questions**



I think have two minutes extra for questions.

*Q: Is it possible for Kong to rate limit upstream requests? For example, it might have 1000 requests coming from down the stream but Kong limits the number of upstream requests to perhaps 30, therefore somehow queuing the downstream requests.*

That’s definitely possible. The rate‑limiting plug‑in that I have used today does not implement that feature, but you can fork this plug‑in (for example) – it’s open source – and then implement the logic in Lua to do exactly that.

The community has built another rate‑limiting plug‑in – it’s called the “quota rate‑limiting plug‑in”. It does not do exactly what you’re trying to implement, but it allows you to configure from the API how requests are rate limited. In the API response you can set a custom header that tells Kong the number of requests that you want to allow for that one consumer. If you set it to zero, Kong will block that consumer from making additional requests.

*Q: Let’s say I have a multitenant system and I have lots of customers sending in request to Kong, but one of them decides to go wild and send 10,000 times as much traffic. Is it possible to somehow queue up that customer’s traffic so that it doesn’t interrupt the traffic from the other customers without necessarily rate limiting it?*

I believe there is a throttling plug‑in available, so you can check that out. Plug‑ins can be applied per API and per consumer, so you can create a plug‑in configuration for one specific consumer of one specific API, and you can have a special treatment just for him.

Microservices Reference Architecture

Learn how to connect, secure, and scale microservices

[Download now](https://www.nginx.com/resources/library/microservices-reference-architecture/)