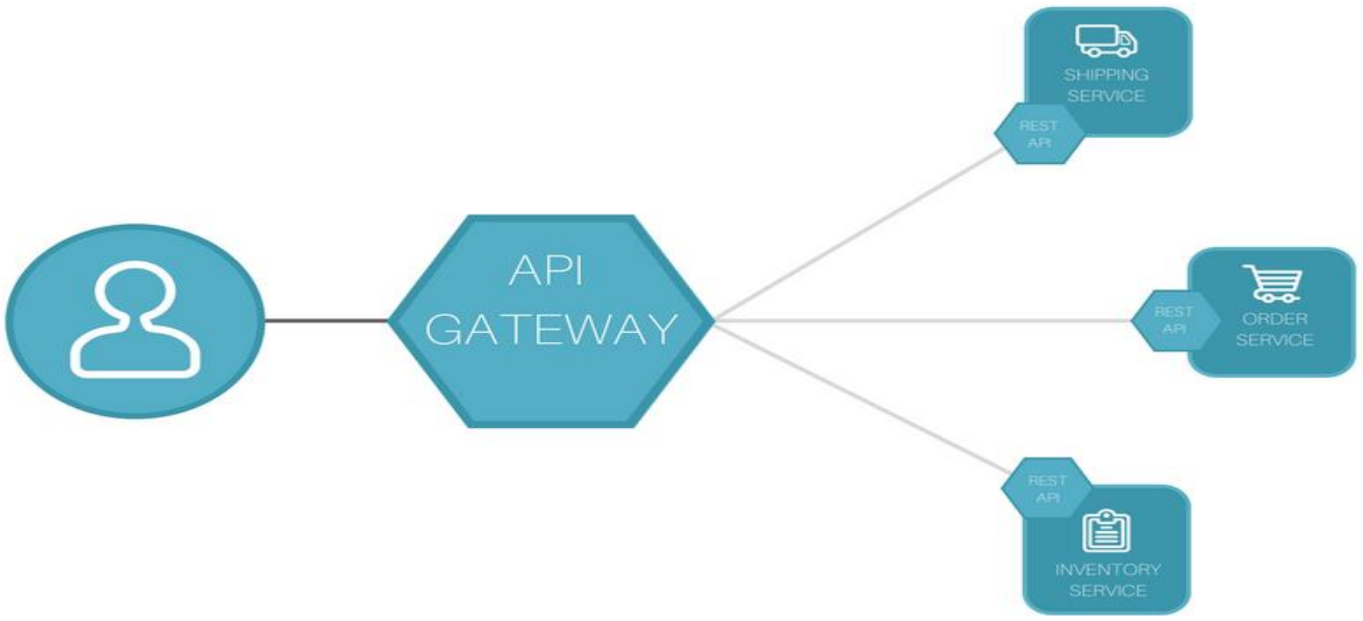
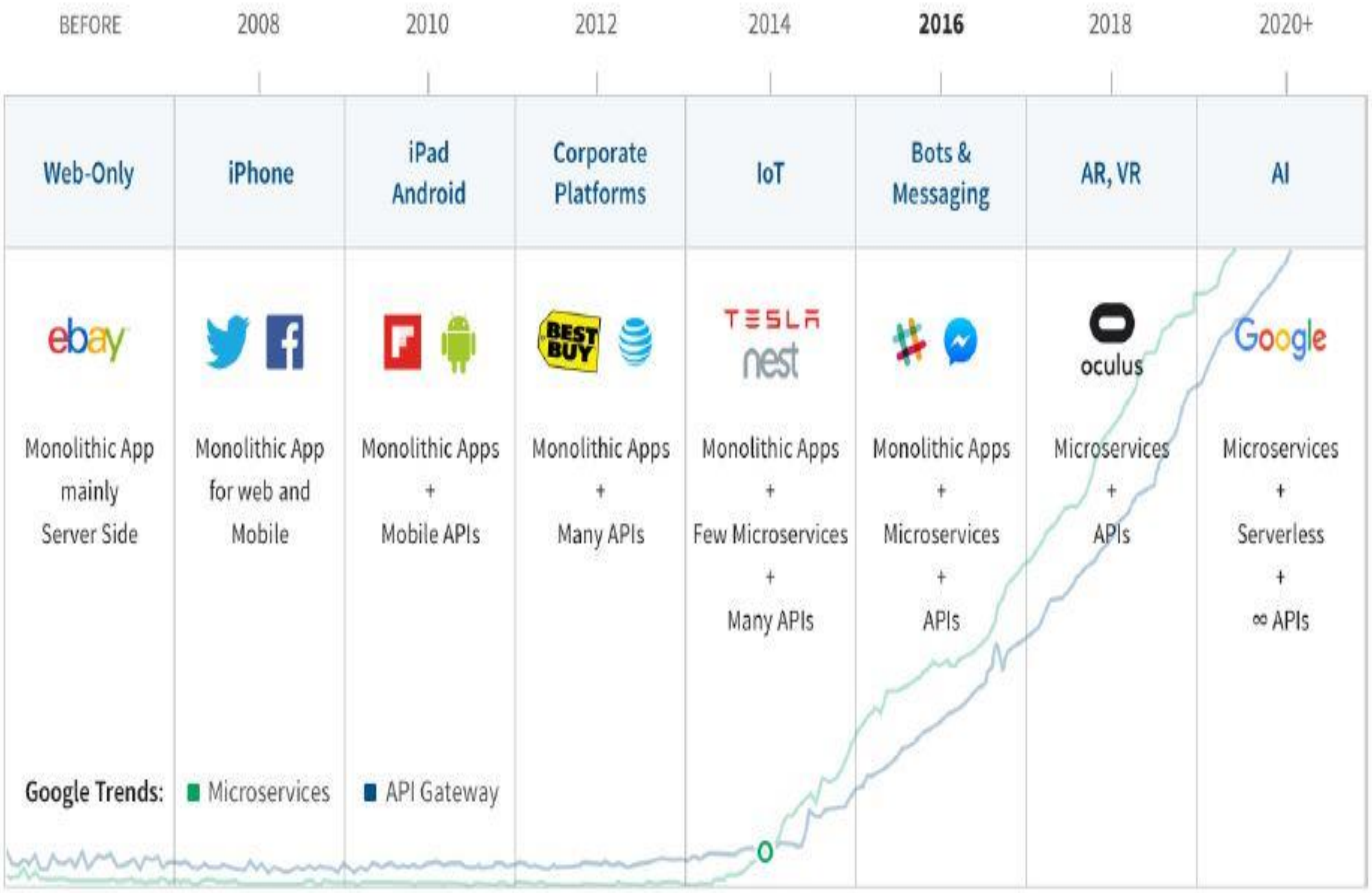
API Gateways in a Microservices World





BEFORE	2008	2010	2012	2014	**2016**	2018	2020+
Web-Only	iPhone	iPad Android	Corporate Platforms	IoT	Bots & Messaging	AR, VR	AI
Monolithic App mainly Server Side	Monolithic App for web and Mobile	Monolithic Apps + Mobile APIs	Monolithic Apps + Many APIs	Monolithic Apps + Few Microservices + Many APIs	Monolithic Apps + Microservices + APIs	Microservices + APIs	Microservices + Serverless + ∞ APIs

Google Trends: 🟩 Microservices 🟦 API Gateway

What is API Gateway and how it works

- An API Gateway is a server that is the single entry point into the system. It is similar to the Facade pattern from object-oriented design.
- The API Gateway is responsible for request routing, composition, and protocol translation.
- It might have other responsibilities such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling

Why use an API Gateway?

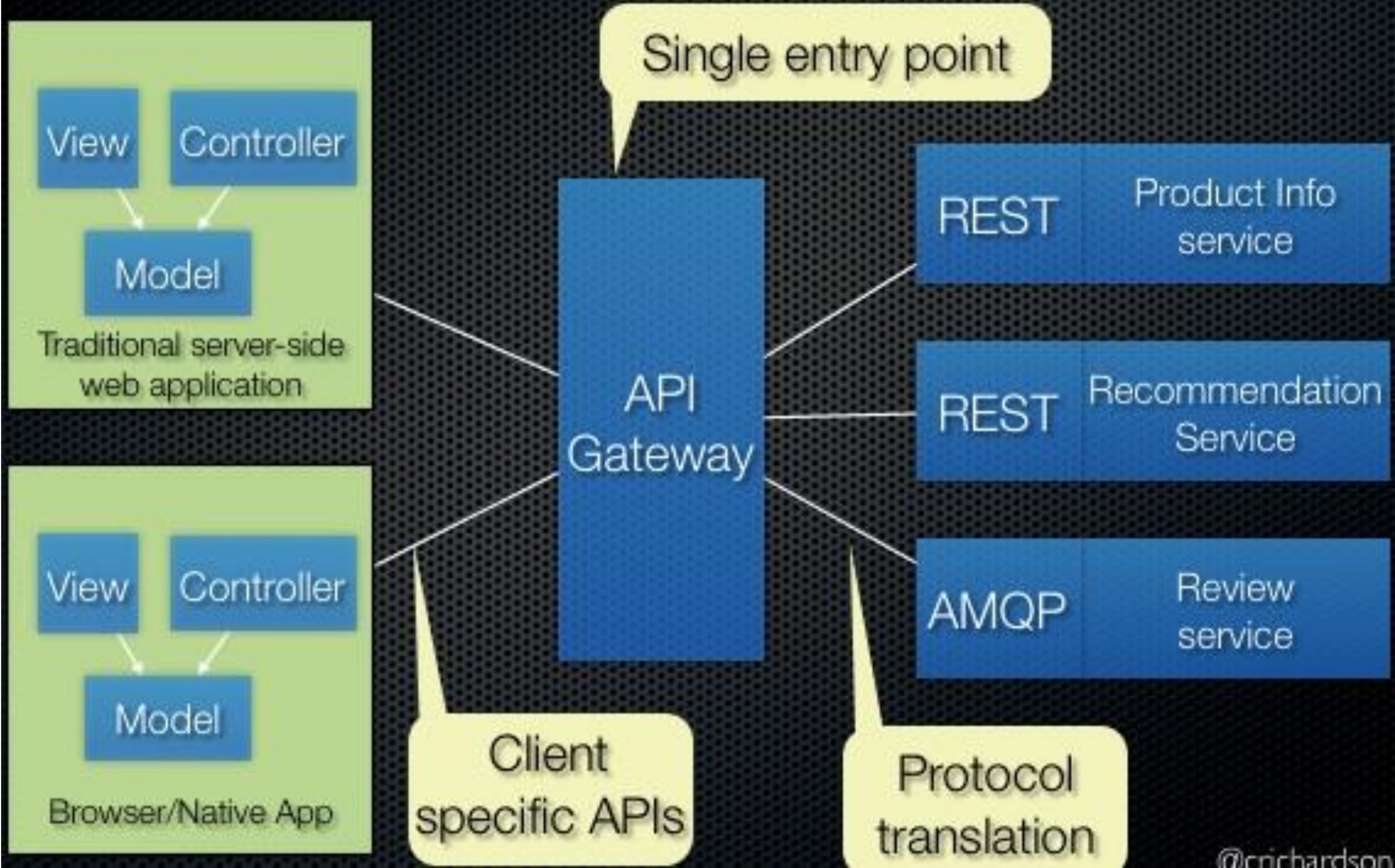
- Gives you centralized control over your APIs and microservices
- Some of the main benefits of the data received via APIs include:

  Accuracy: Data is returned directly from an authoritative source and can be verified to ensure that it is complete and correct.

  Cleanliness: APIs are designed to return data in a usable, accessible format, free from ‘bad’ data and capable of being retrieved by your system in the preferred filetype.

  Speed: Data is returned in real-time, giving much faster updates than manually pulling information from the server.

Use an API gateway



We will take a look at two API Gateways

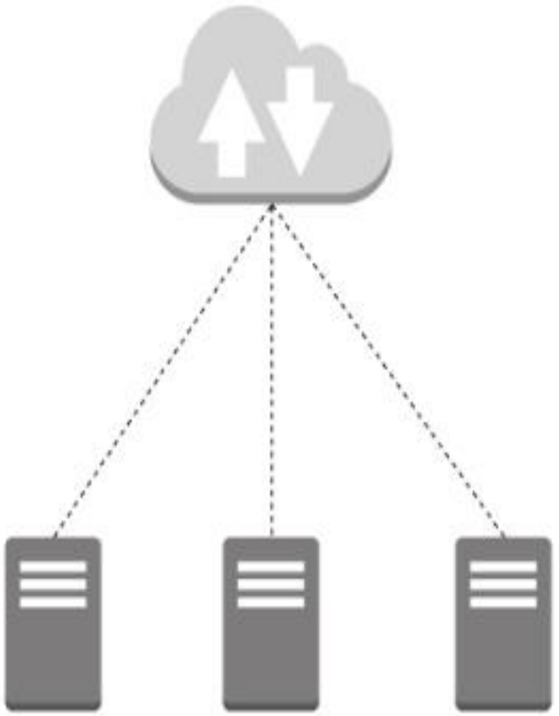
- Fast-gateway and
- Kong

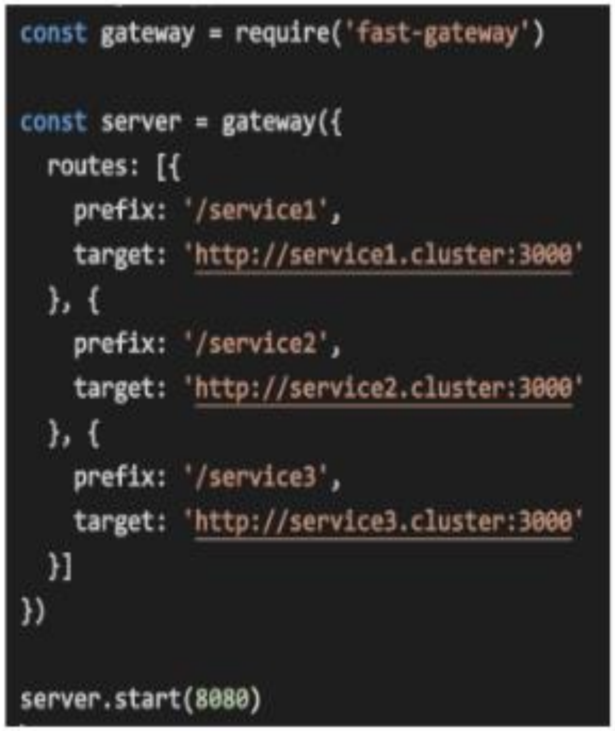
Fast-gateway

- *Fast-gateway* is an HTTP gateway framework that you can program using JavaScript and Node.js. Because it is built using *restana*, connect/express-like middlewares are also compatible and there is no need to re-invent.
- Drastically reducing the latency of your response times to one digit milliseconds.

fast-gateway

fast-proxy is too low level, let's make it a friendly, all in one API Gateway solution for HTTP based distributed applications: *fast-gateway*





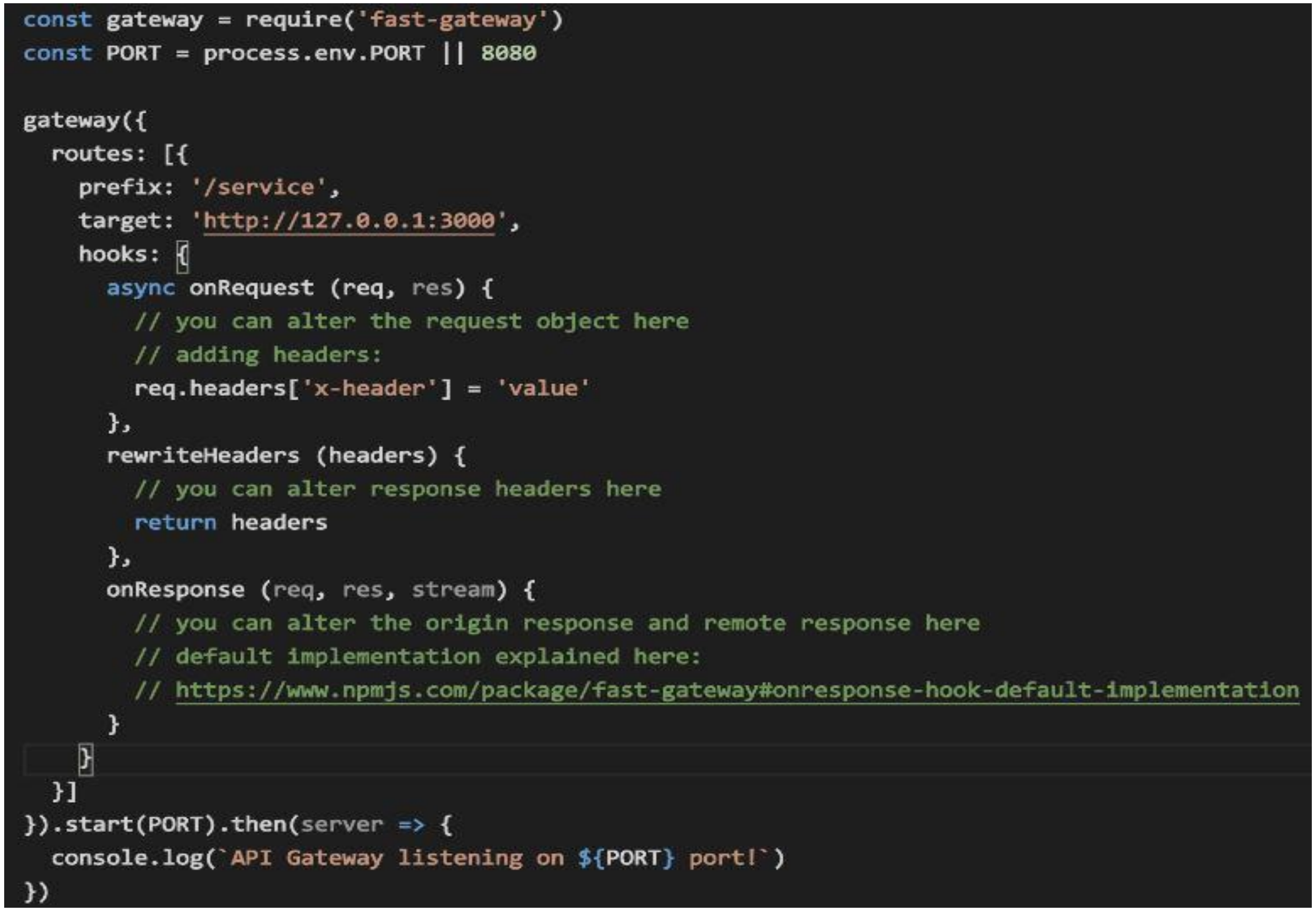
```
const gateway = require('fast-gateway')

const server = gateway({
  routes: [{
    prefix: '/service1',
    target: 'http://service1.cluster:3000'
  }, {
    prefix: '/service2',
    target: 'http://service2.cluster:3000'
  }, {
    prefix: '/service3',
    target: 'http://service3.cluster:3000'
  }]
})

server.start(8080)
```

Hooks with fast-gateway

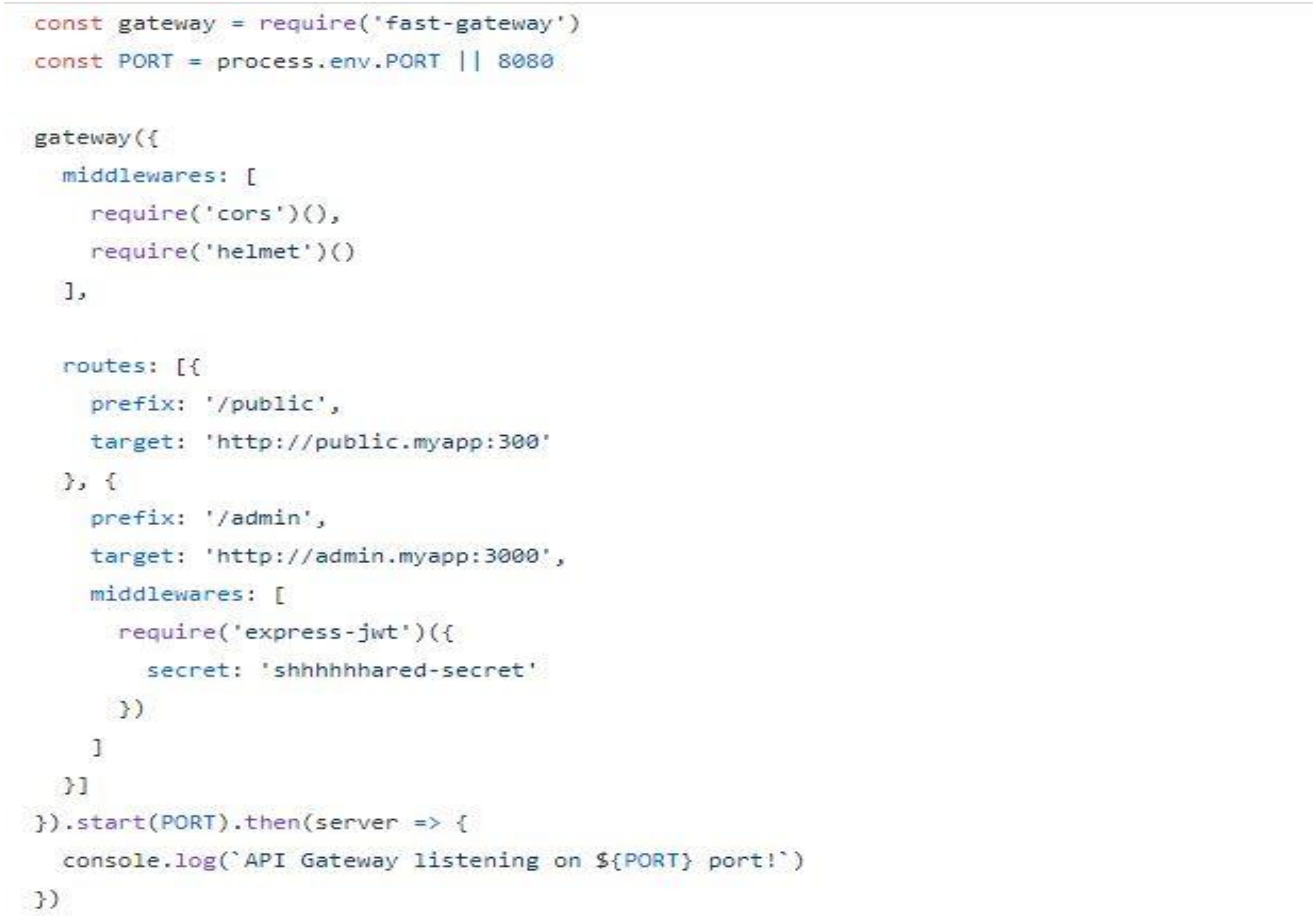
- You also have the ability to optionally intercept requests and responses passing through your routes



```
const gateway = require('fast-gateway')
const PORT = process.env.PORT || 8080

gateway({
  routes: [{
    prefix: '/service',
    target: 'http://127.0.0.1:3000',
    hooks: {
      async onRequest (req, res) {
        // you can alter the request object here
        // adding headers:
        req.headers['x-header'] = 'value'
      },
      rewriteHeaders (headers) {
        // you can alter response headers here
        return headers
      },
      onResponse (req, res, stream) {
        // you can alter the origin response and remote response here
        // default implementation explained here:
        // https://www.npmjs.com/package/fast-gateway#onresponse-hook-default-implementation
      }
    }
  }]
}).start(PORT).then(server => {
  console.log(`API Gateway listening on ${PORT} port!`)
})
```

Example with JTW Authorization

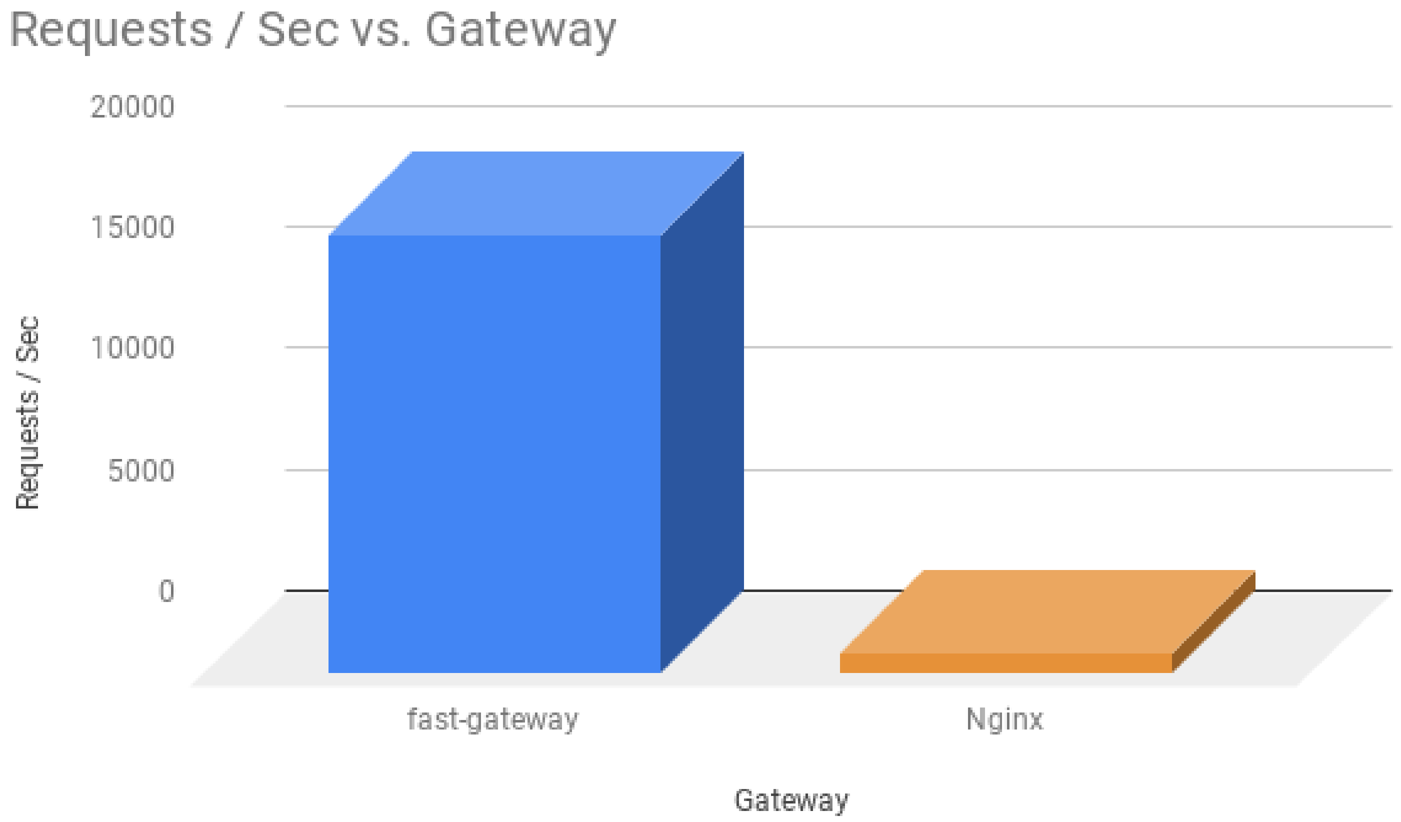


```
const gateway = require('fast-gateway')
const PORT = process.env.PORT || 8080

gateway({
  middlewares: [
    require('cors')(),
    require('helmet')()
  ],

  routes: [{
    prefix: '/public',
    target: 'http://public.myapp:300'
  }, {
    prefix: '/admin',
    target: 'http://admin.myapp:3000',
    middlewares: [
      require('express-jwt')({
        secret: 'shhhhhhared-secret'
      })
    ]
  }]
}).start(PORT).then(server => {
  console.log(`API Gateway listening on ${PORT} port!`)
})
```

Is *fast-gateway* actually fast?



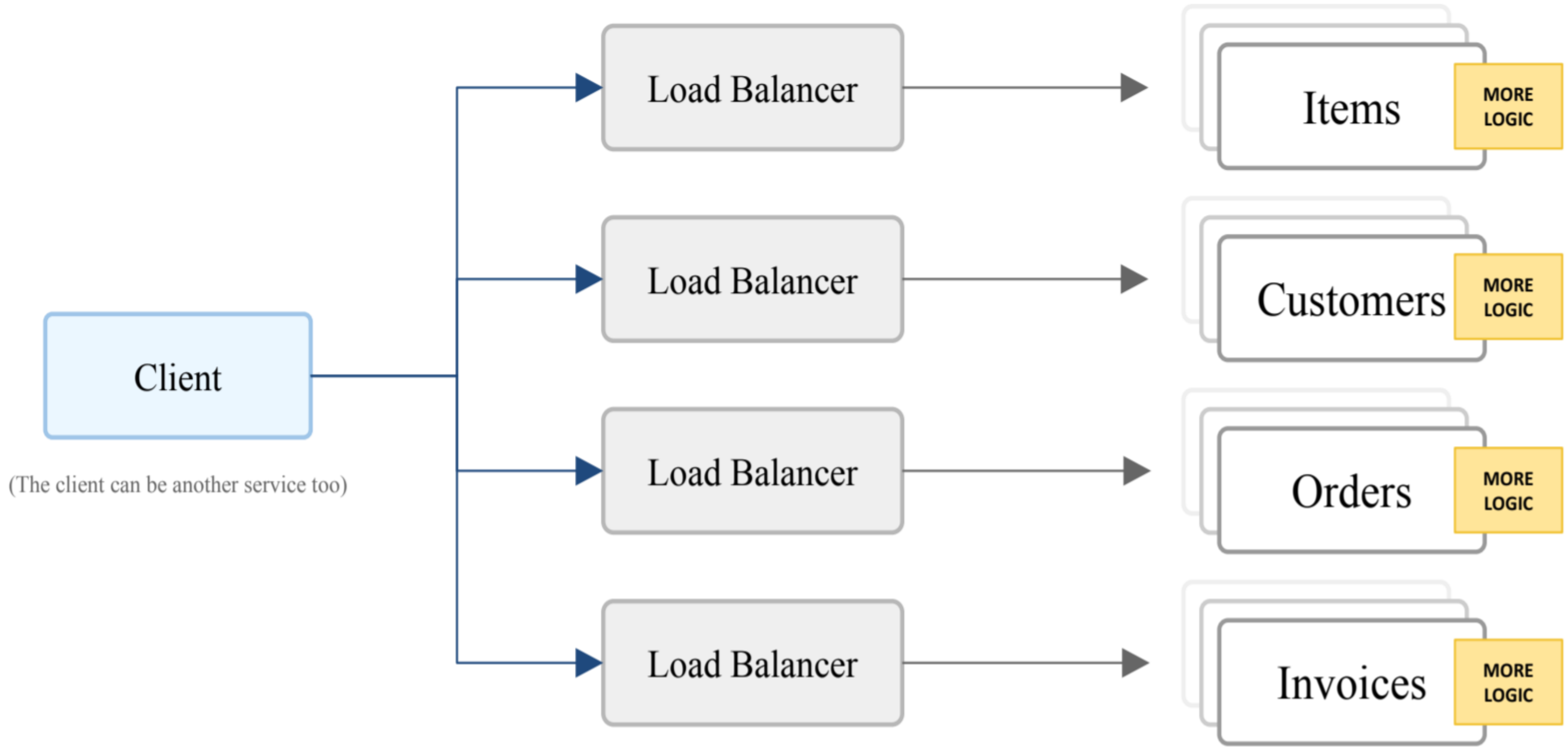
Kong

Features of the Kong API Gateway

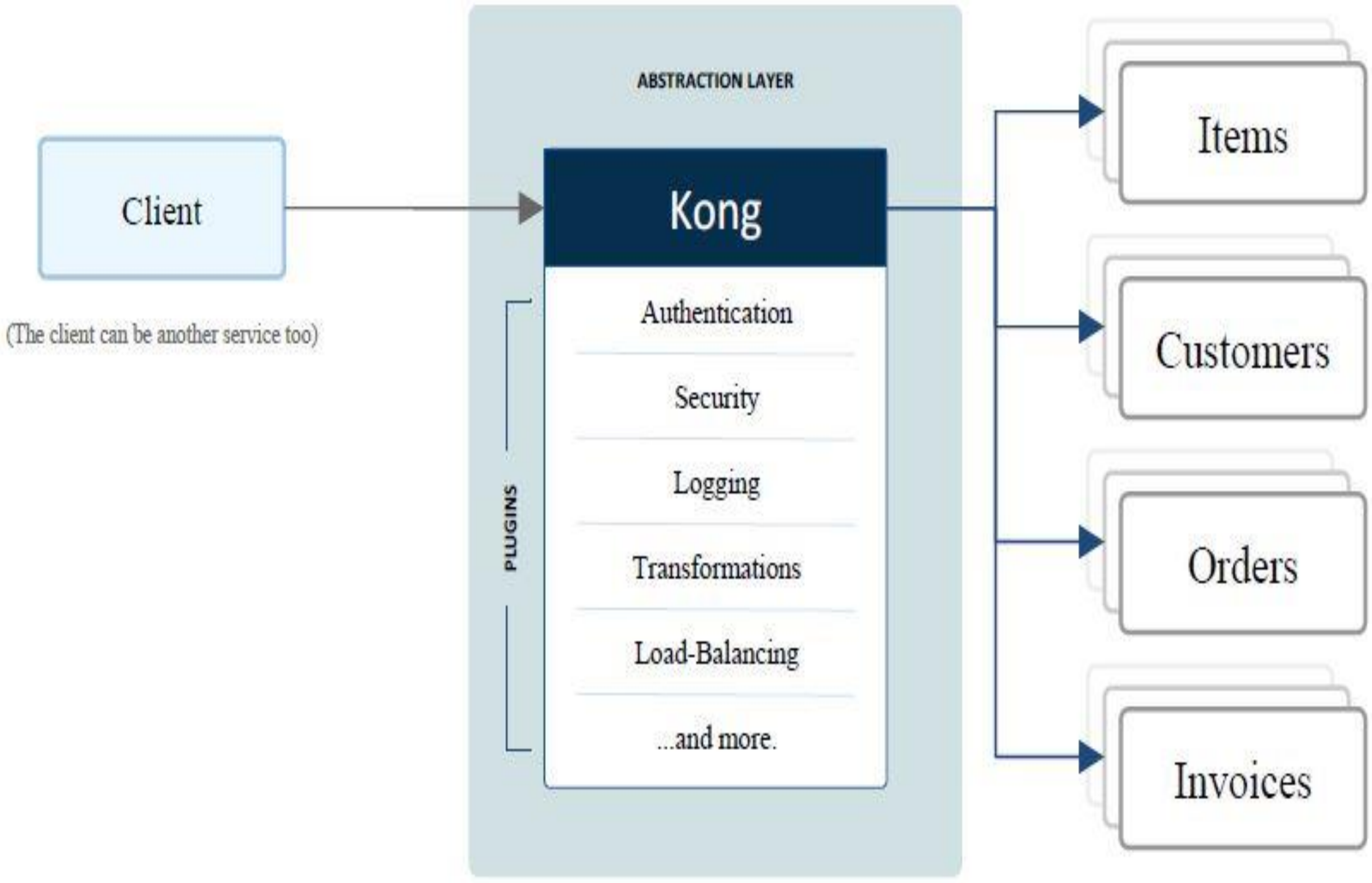
- Open-source
- Standard Kong API Gateway deployment creates an environment in which all of your APIs can work together.
- Option to aggregate API data into a single datastore.
- The Kong API Gateway give robust control over security, traffic and visualization of performance data.
- Lightweight proxy that gives huge scalability, end-to-end encryption and running smoothly

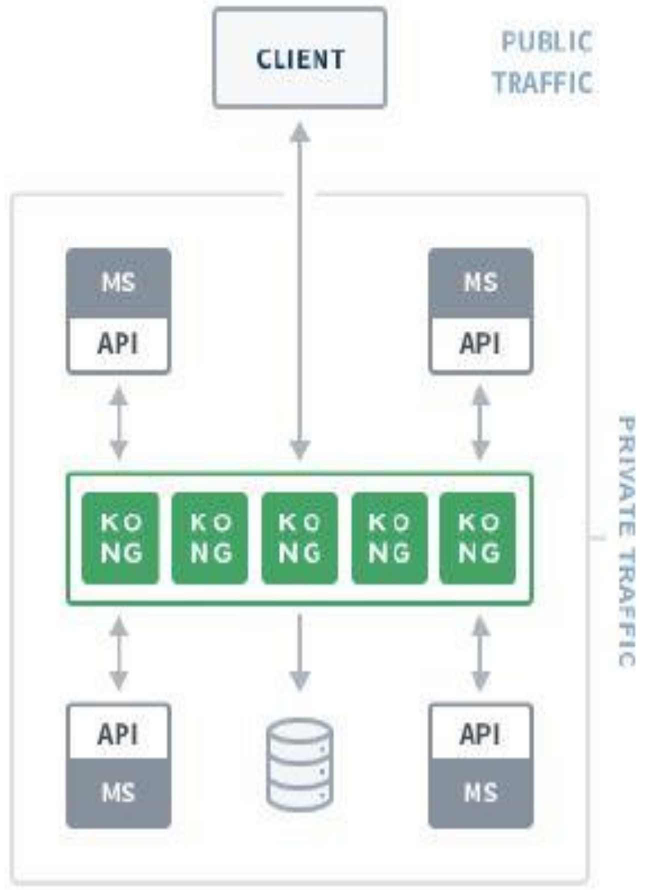
- Some of features include:
- **Analytics**: Detailed data analytics to inspect, monitor and visualize your microservices traffic.
- **Authentication**: An authentication layer to protect your services and ensure only authorized access.
- **Logging**: Real-time and continuous logging of API request and response data.
- **Serverless**: Deploy serverless functions without the need to restart or redeploy Kong.
- **Traffic Control**: Granular control over inbound and outbound API traffic, including the ability to throttle and restrict traffic as required.
- **Transformations**: Handle request and response data transformations on the fly.

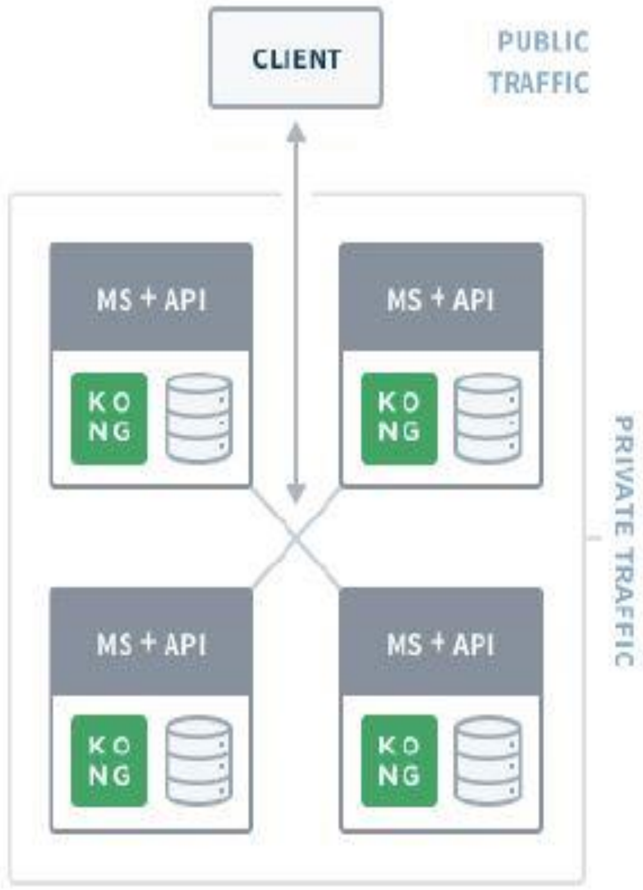
Separating services business logic from cross-cutting concerns using Kong 1/2

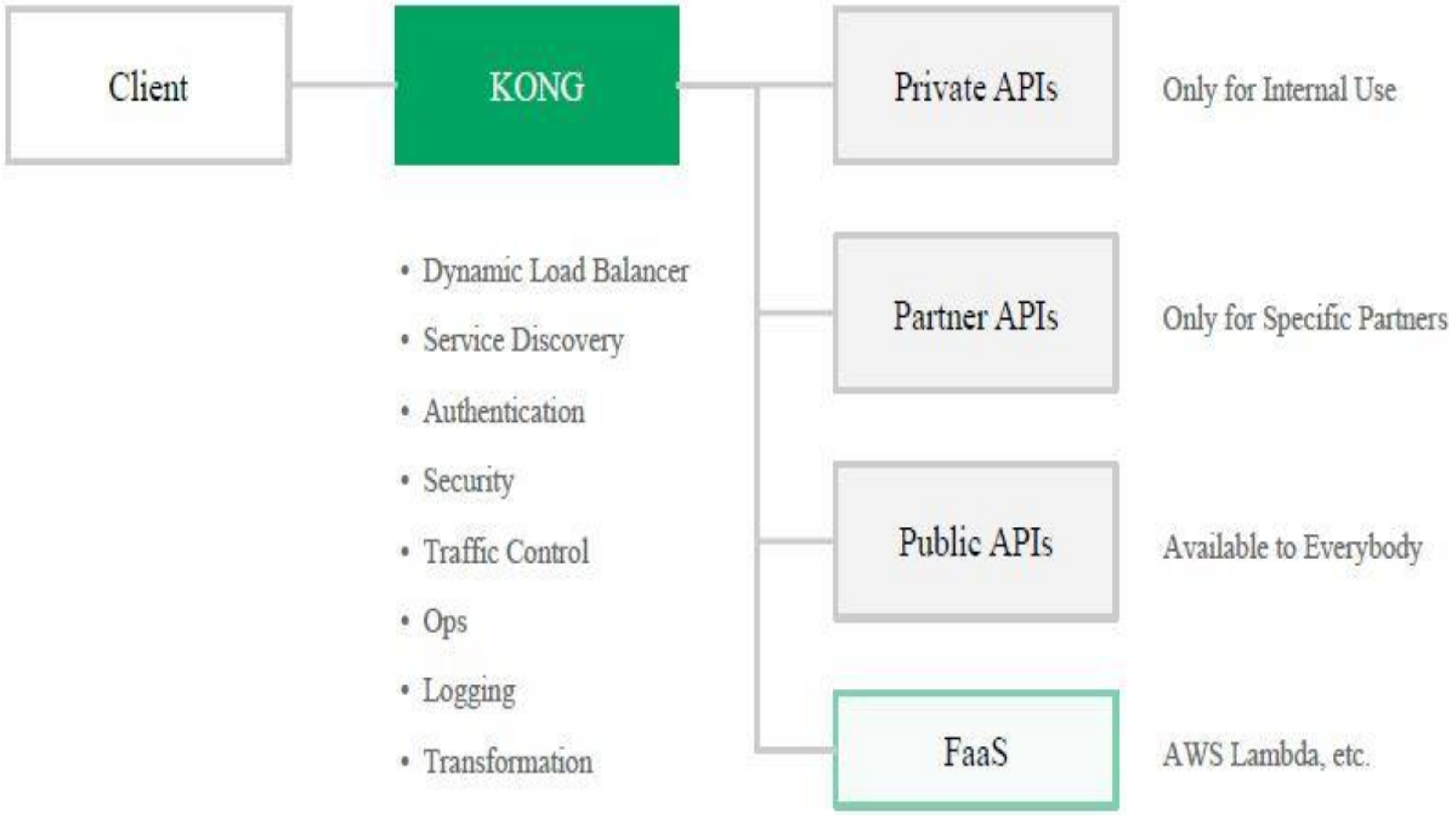


Separating services business logic from cross-cutting concerns using Kong 2/2



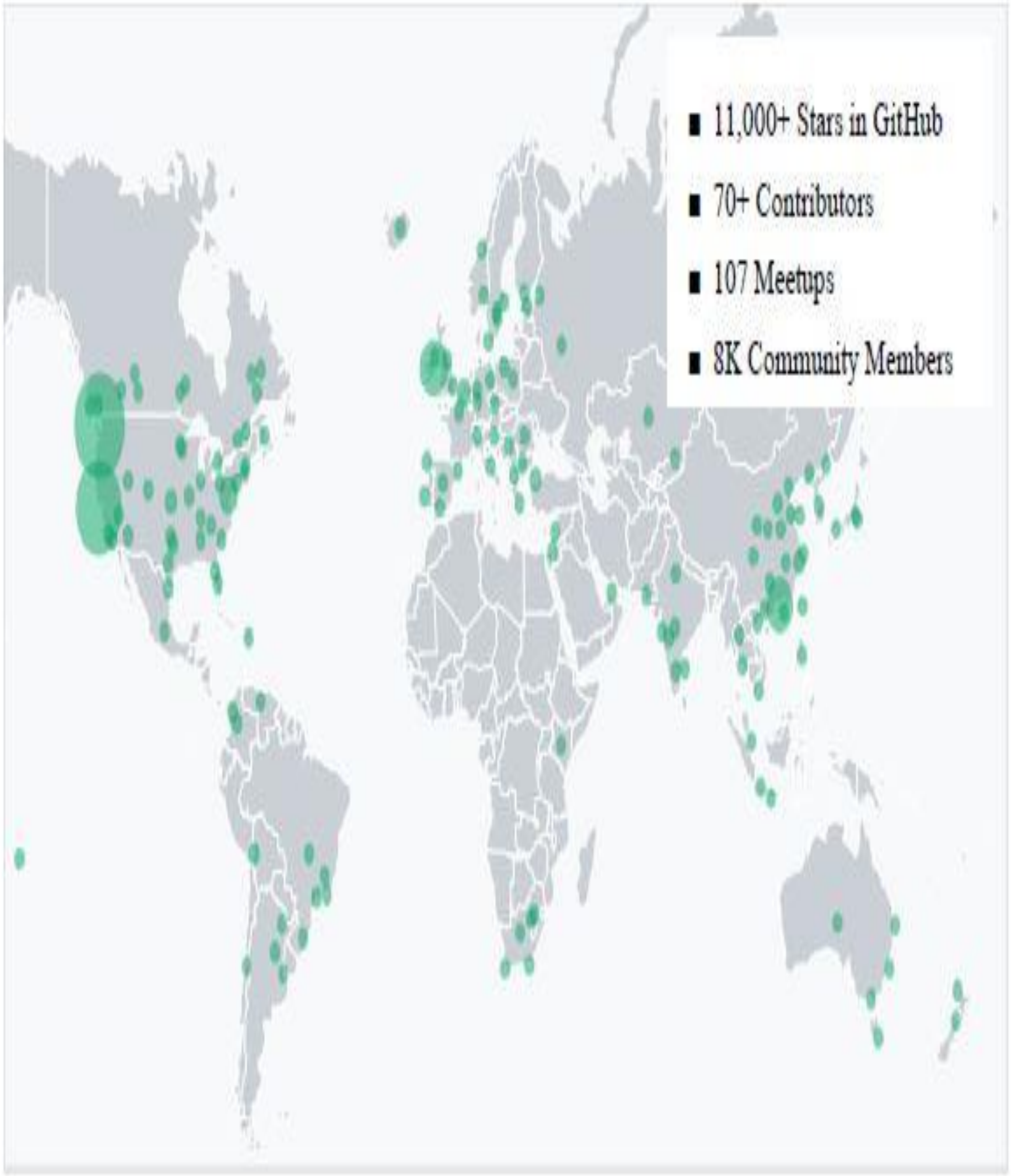

CLIENT
PUBLIC TRAFFIC
MS
API
MS
API
KONG
KONG
KONG
KONG
KONG
API
MS
API
MS
PRIVATE TRAFFIC


CLIENT
PUBLIC TRAFFIC
MS + API
KONG
MS + API
KONG
MS + API
KONG
MS + API
KONG
PRIVATE TRAFFIC


Client
KONG
• Dynamic Load Balancer
• Service Discovery
• Authentication
• Security
• Traffic Control
• Ops
• Logging
• Transformation
Private APIs
Only for Internal Use
Partner APIs
Only for Specific Partners
Public APIs
Available to Everybody
FaaS
AWS Lambda, etc.

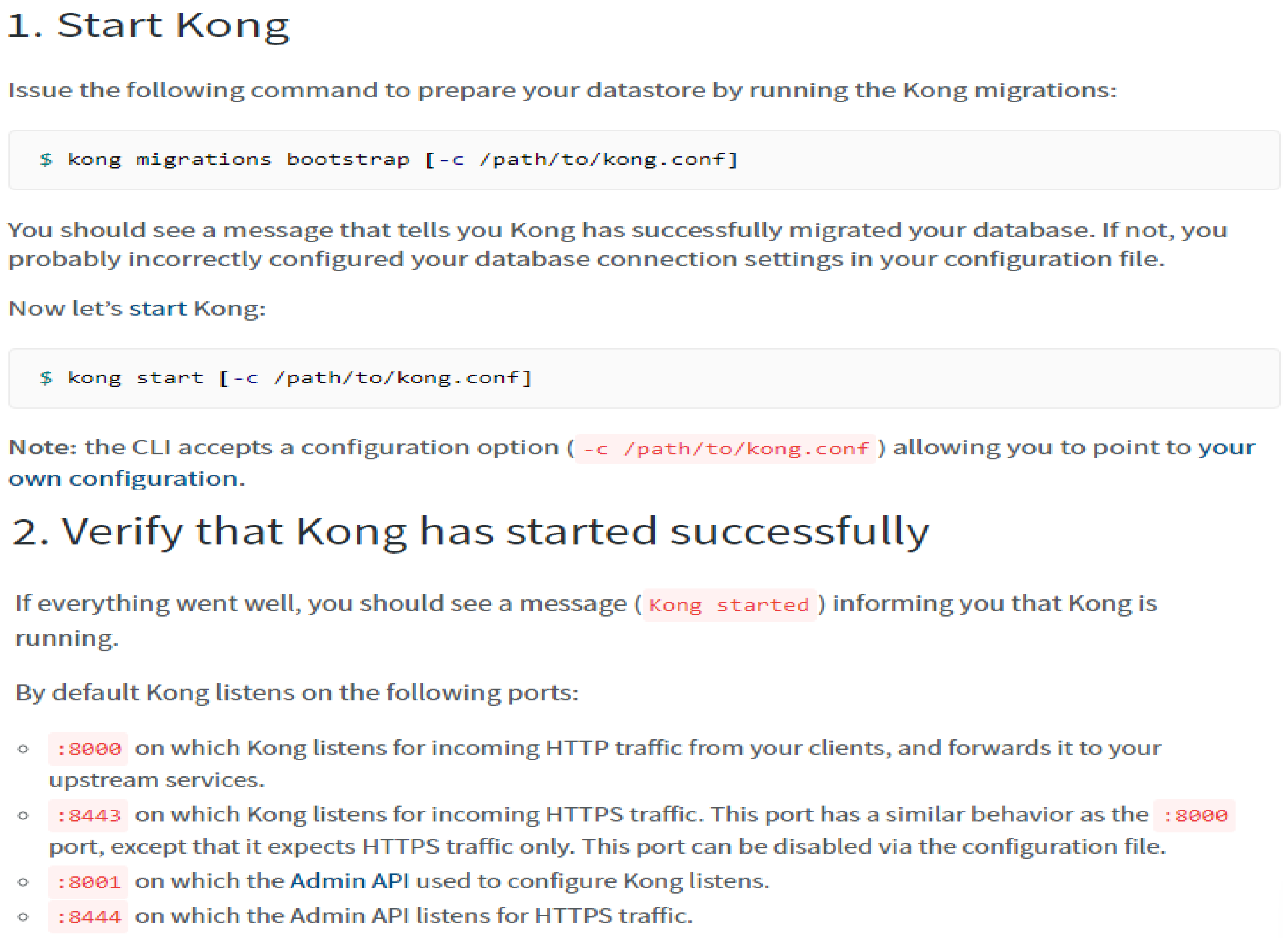
Millions of Community (CE) **downloads** across multiple platforms.

- Open-Source
- 3M+ Downloads
- Built on top of NGINX
- Extensible with Plugins (60+ available)
- **Sub-millisecond** latency on most use-cases
- Cloud-Native & Platform Agnostic
- Fast and Scalable. Up and running in minutes



- 11,000+ Stars in GitHub
- 70+ Contributors
- 107 Meetups
- 8K Community Members

Installation Step-by-step



1. Start Kong

Issue the following command to prepare your datastore by running the Kong migrations:

```
$ kong migrations bootstrap [-c /path/to/kong.conf]
```

You should see a message that tells you Kong has successfully migrated your database. If not, you probably incorrectly configured your database connection settings in your configuration file.

Now let's start Kong:

```
$ kong start [-c /path/to/kong.conf]
```

Note: the CLI accepts a configuration option (`-c /path/to/kong.conf`) allowing you to point to your own configuration.

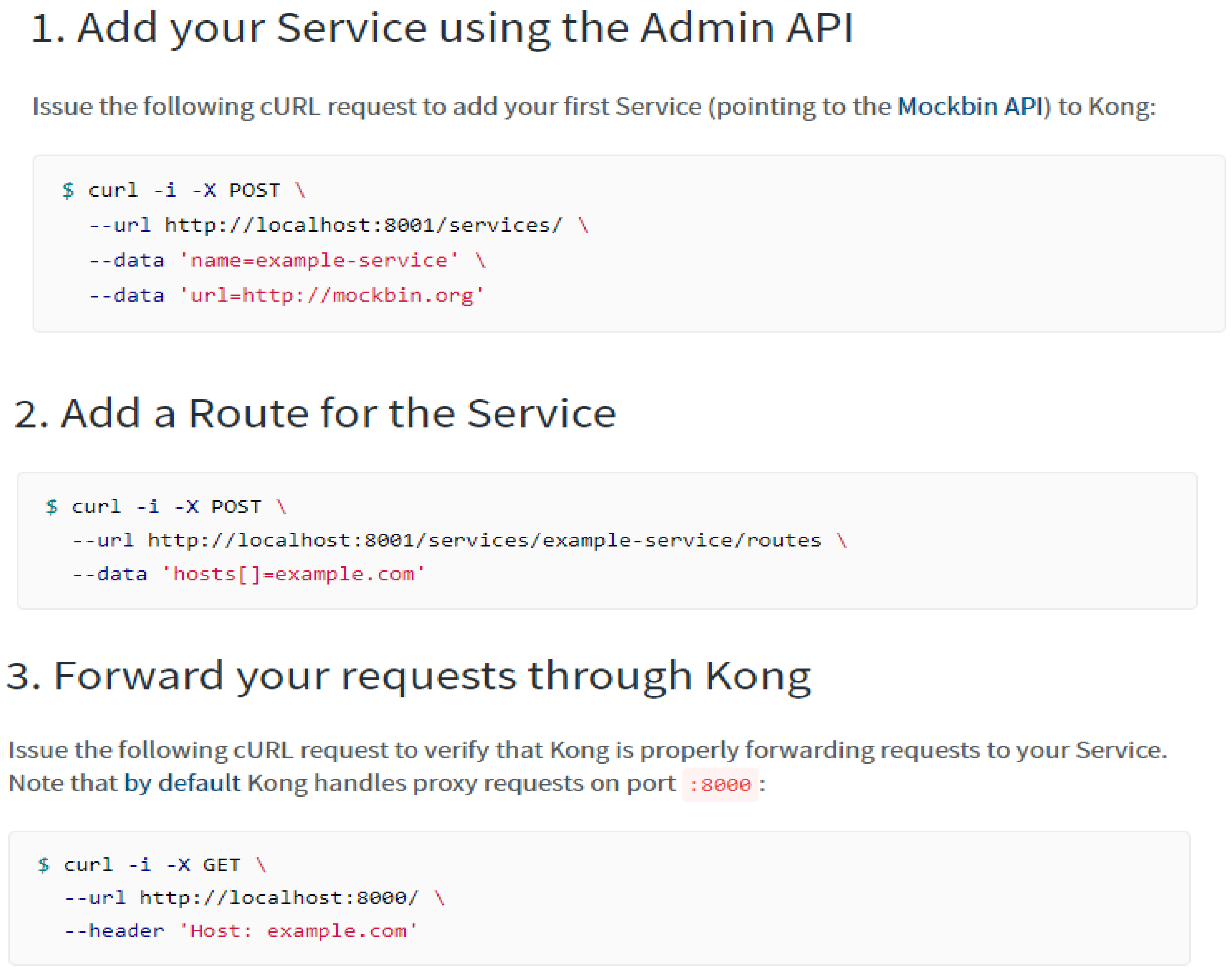
2. Verify that Kong has started successfully

If everything went well, you should see a message (`Kong started`) informing you that Kong is running.

By default Kong listens on the following ports:

- `:8000` on which Kong listens for incoming HTTP traffic from your clients, and forwards it to your upstream services.
- `:8443` on which Kong listens for incoming HTTPS traffic. This port has a similar behavior as the `:8000` port, except that it expects HTTPS traffic only. This port can be disabled via the configuration file.
- `:8001` on which the Admin API used to configure Kong listens.
- `:8444` on which the Admin API listens for HTTPS traffic.

Configuring a Service…



1. Add your Service using the Admin API

Issue the following cURL request to add your first Service (pointing to the Mockbin API) to Kong:

```
$ curl -i -X POST \
  --url http://localhost:8001/services/ \
  --data 'name=example-service' \
  --data 'url=http://mockbin.org'
```

2. Add a Route for the Service

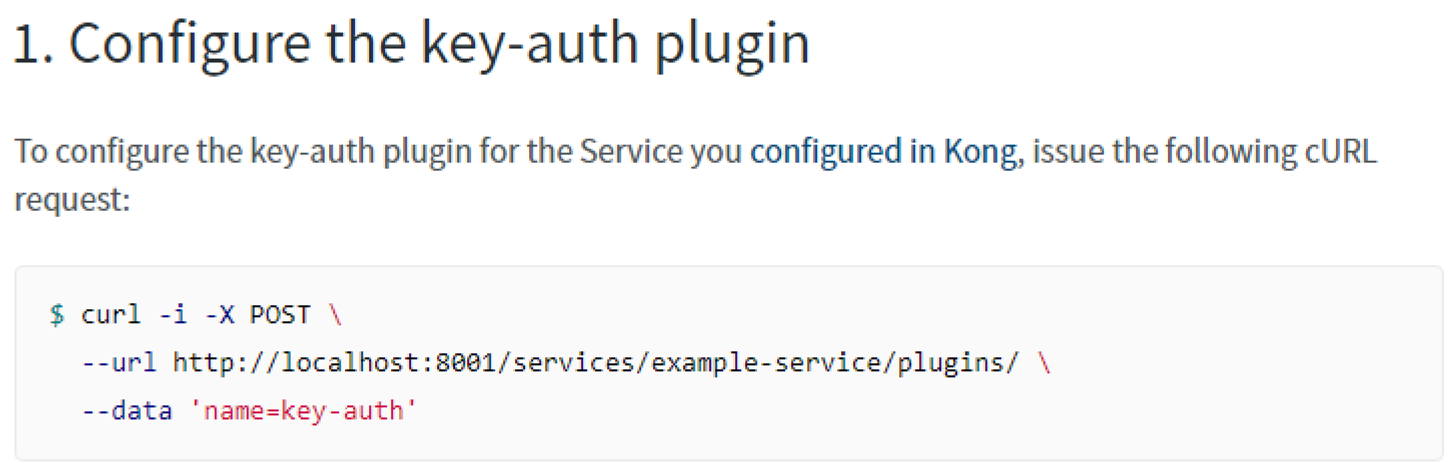
```
$ curl -i -X POST \
  --url http://localhost:8001/services/example-service/routes \
  --data 'hosts[]=example.com'
```

3. Forward your requests through Kong

Issue the following cURL request to verify that Kong is properly forwarding requests to your Service. Note that by default Kong handles proxy requests on port `:8000`:

```
$ curl -i -X GET \
  --url http://localhost:8000/ \
  --header 'Host: example.com'
```

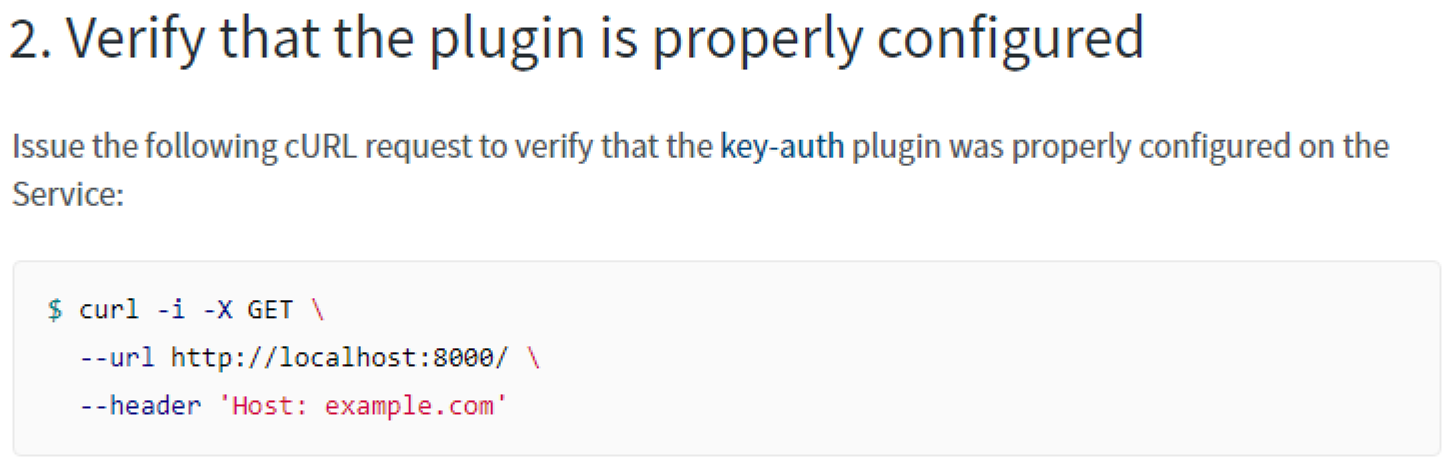
Adding plugins



1. Configure the key-auth plugin

To configure the key-auth plugin for the Service you configured in Kong, issue the following cURL request:

```
$ curl -i -X POST \
  --url http://localhost:8001/services/example-service/plugins/ \
  --data 'name=key-auth'
```

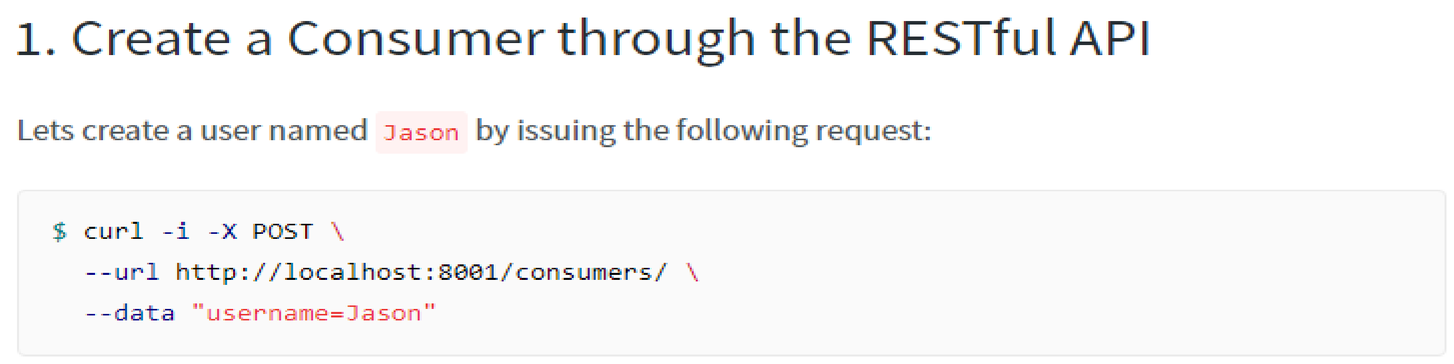


2. Verify that the plugin is properly configured

Issue the following cURL request to verify that the key-auth plugin was properly configured on the Service:

```
$ curl -i -X GET \
  --url http://localhost:8000/ \
  --header 'Host: example.com'
```

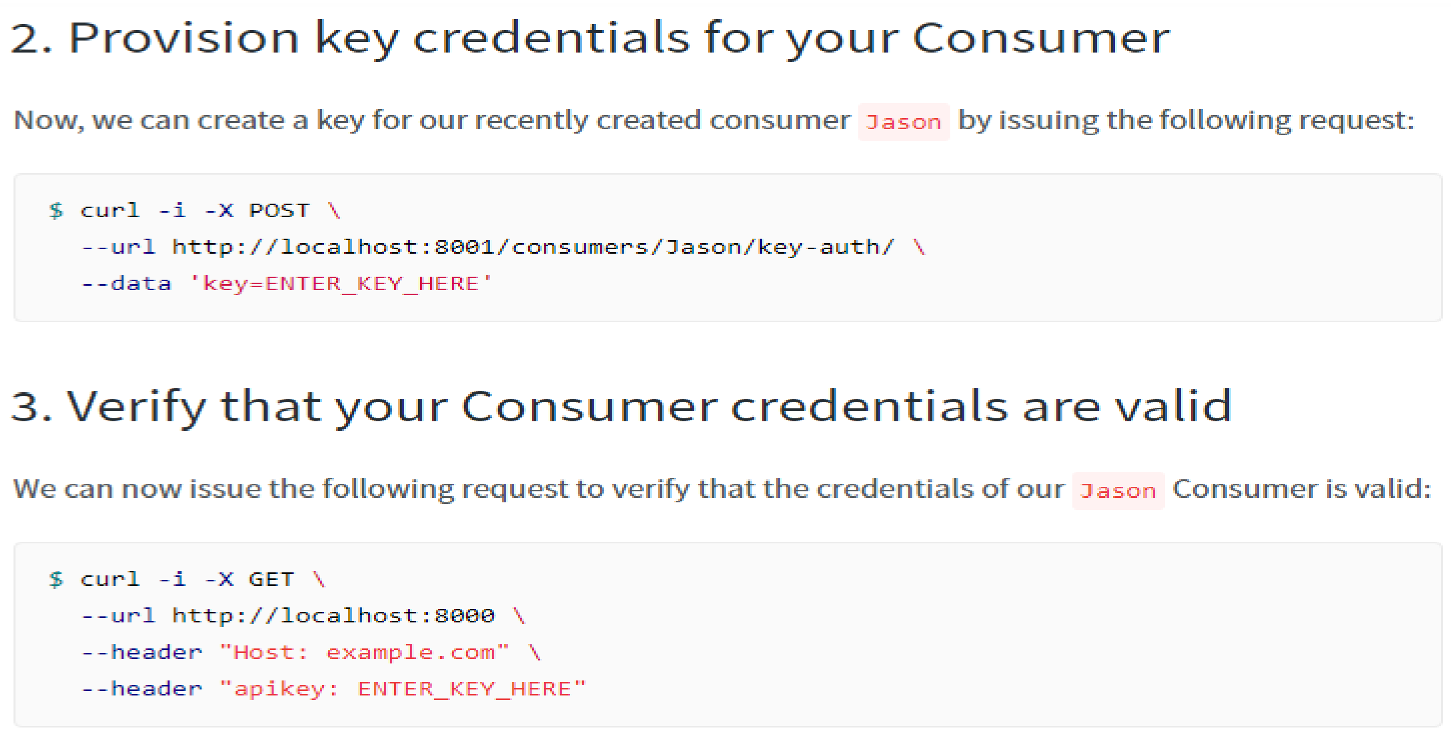
Adding Consumers



1. Create a Consumer through the RESTful API

Lets create a user named `Jason` by issuing the following request:

```
$ curl -i -X POST \
  --url http://localhost:8001/consumers/ \
  --data "username=Jason"
```



2. Provision key credentials for your Consumer

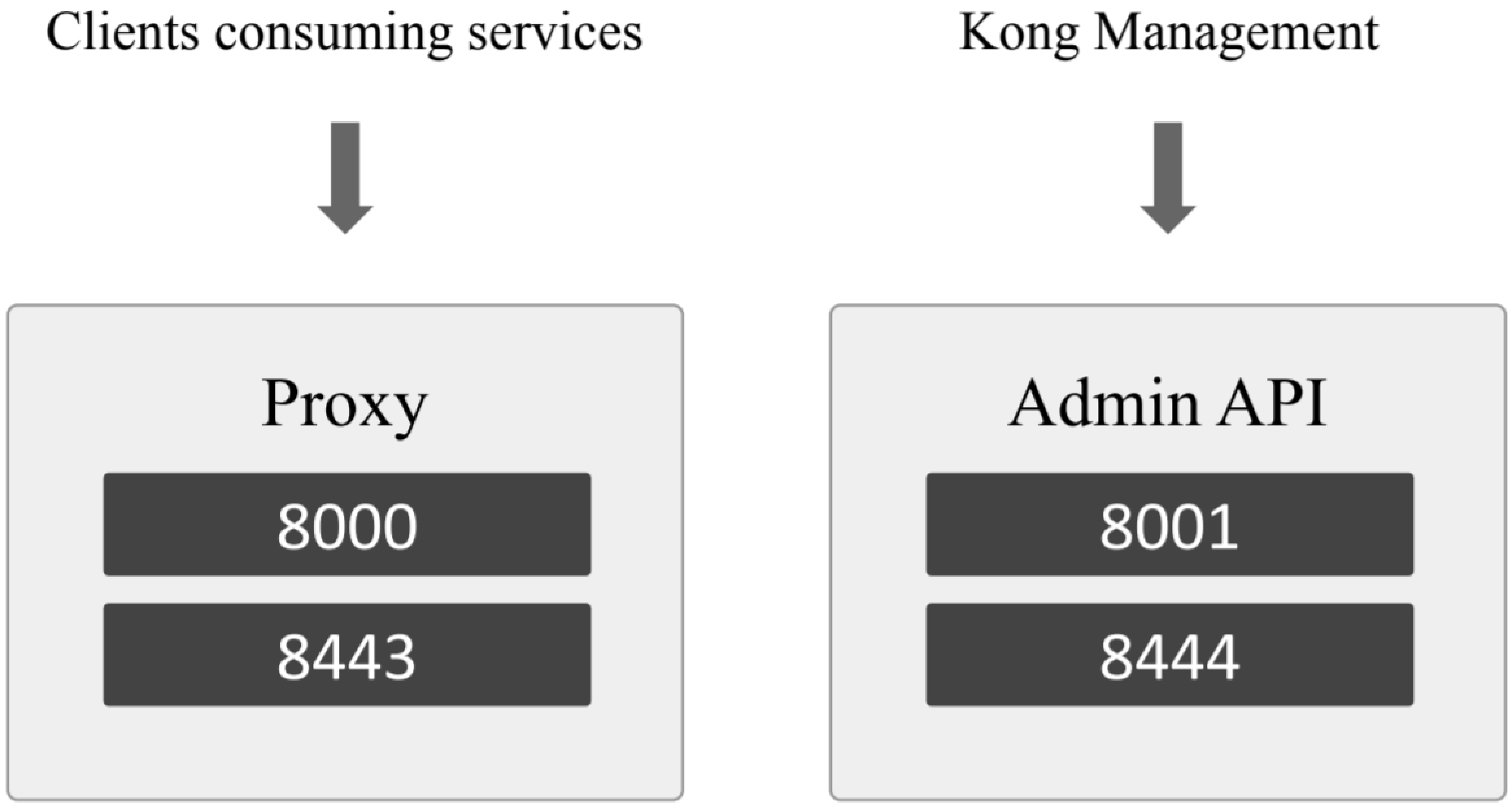
Now, we can create a key for our recently created consumer `Jason` by issuing the following request:

```
$ curl -i -X POST \
  --url http://localhost:8001/consumers/Jason/key-auth/ \
  --data 'key=ENTER_KEY_HERE'
```

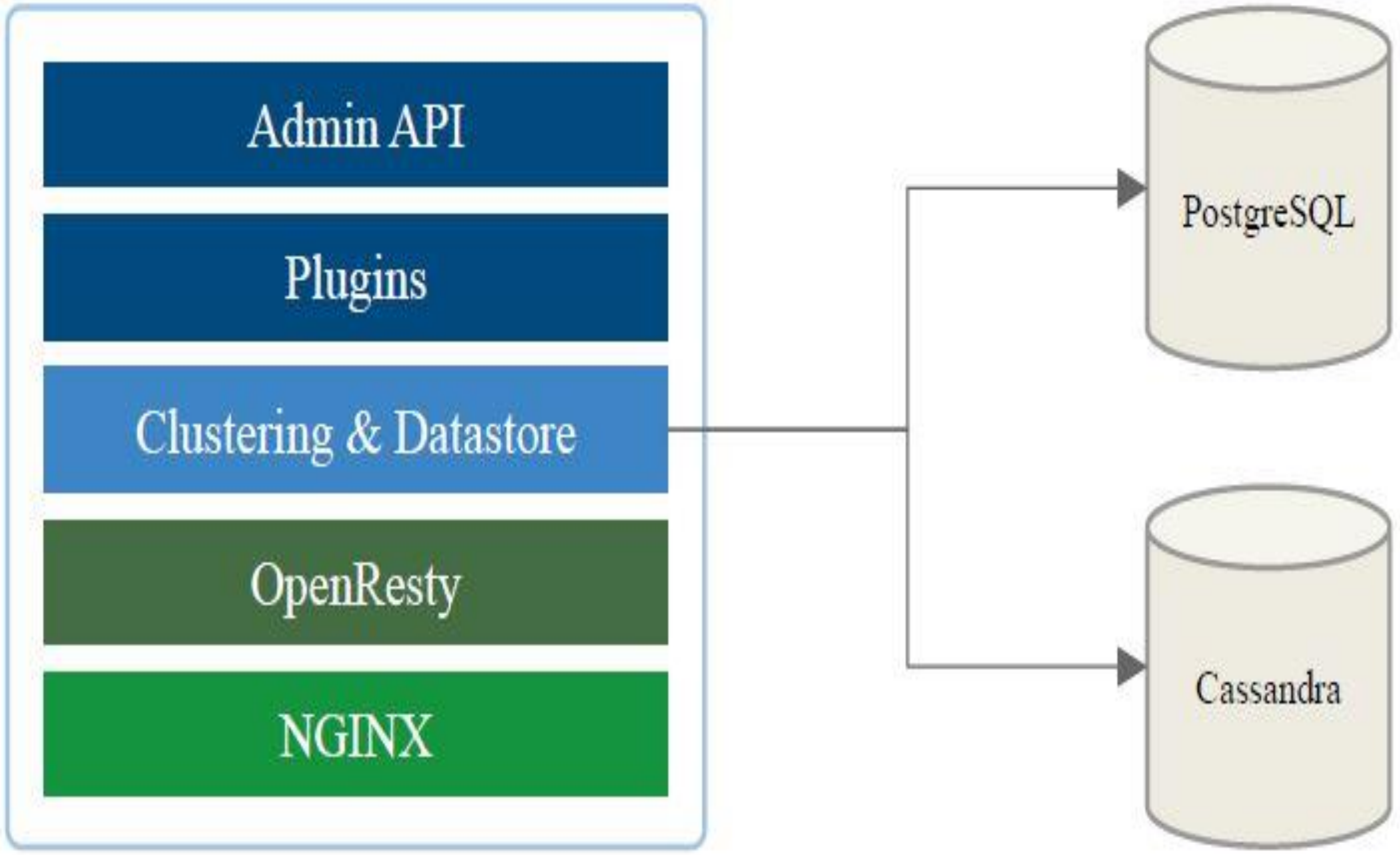
3. Verify that your Consumer credentials are valid

We can now issue the following request to verify that the credentials of our `Jason` Consumer is valid:

```
$ curl -i -X GET \
  --url http://localhost:8000 \
  --header "Host: example.com" \
  --header "apikey: ENTER_KEY_HERE"
```


Clients consuming services
Kong Management
Proxy
8000
8443
Admin API
8001
8444

The Kong stack



Kong DEMO

Questions?