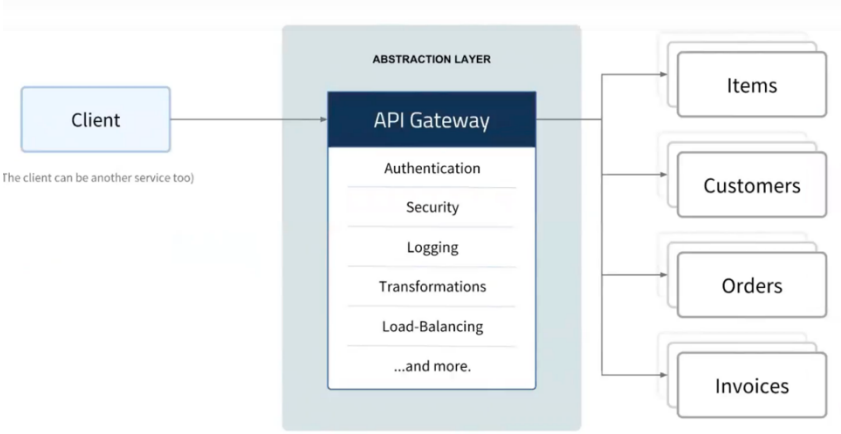
Kong

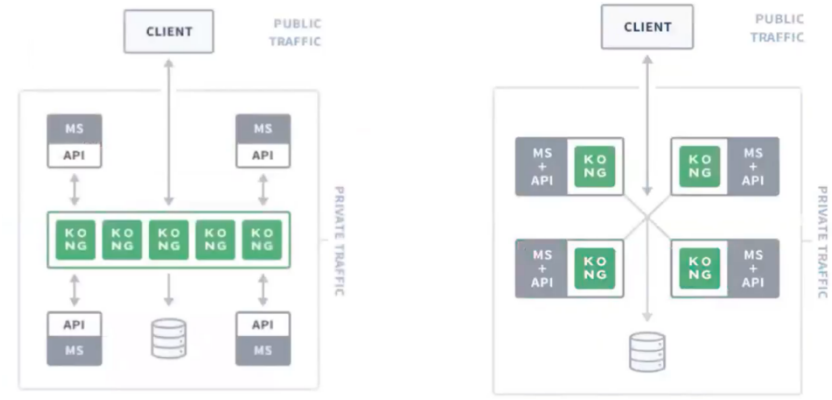
Kong Overview

Kong jest to open source API Gateway podobny do np. Kafki. Podstawową wersję można pobrać za darmo z GitHuba lub z oficjalnej strony. Kong jest stworzony, aby współpracować z najnowszymi trendami - API służące do komunikacji mikroserwisów, konteneryzacja. Kong potrafi współpracować z innymi oprogramowaniami i nie wymaga dedykowanej architektury. Jest zintegrowany ze wszystkimi narzędziami do konteneryzacji np. Docker, Kubernetes i ze wszystkimi chmurami (AWS, Azure itp.). Kong jest zbudowany na podstawie Nginx, Lua i LuaJIT, dzięki temu jest bardzo skalowalny. Z Kongiem można się komunikować przez CLI, ale też przez GUI (Kong Manager).

Gdy przechodzimy z infrastruktury typu monolit na MicroService musimy pamiętać, że każdy z tych serwisów musi mieć osobno skonfigurowane Authentication, Secuirty, Logging do zarządzania requestami i połączeniami. Takich serwisów możemy mieć tysiące i nie chcemy za każdym razem ustawiać tego ręcznie, więc najlepiej jest użyć czegoś, co jest w stanie automatycznie ustawiać to za nas. W tym miejscu należy użyć API Gatewaya takiego jak Kong. Wszystkie cechy, którymi się Kong zajmuje to Plugins (można tworzyć swoje, lub korzystać z tego co inni stworzyli). Domyślnie Kong zarządza requestami na porcie 8000.

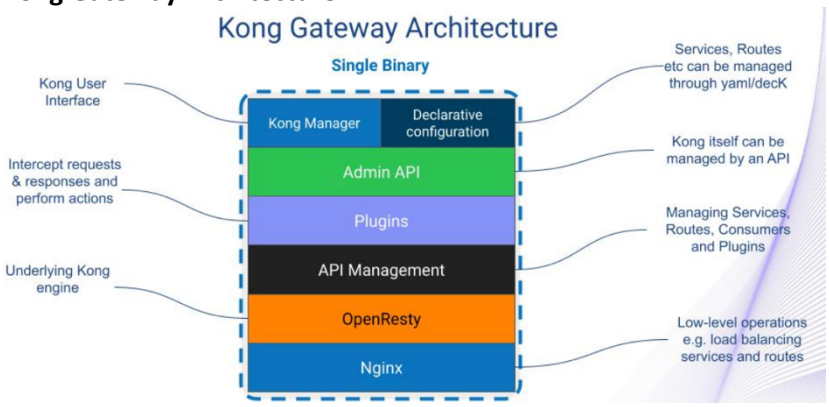


Kong może być zdeployowany Centralnie lub przez Zdecentralizowanie (podobnie jest zdeployowane Istio, że jest w każdym podzie w Kubernetesie).



Dokumentacja Konga: docs.konghq.com

Kong Gateway Architecture

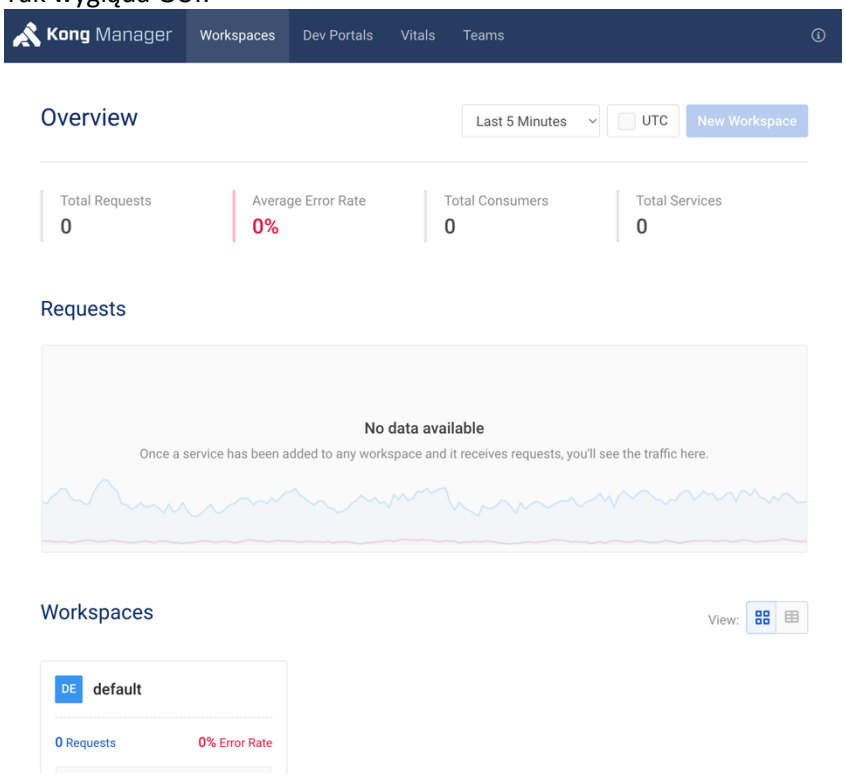


Nginx zajmuje się podstawowymi operacji Konga np. load balancingiem Service i Route.
OpenResty to wewnętrzny silnik Konga i zajmuje się obsługą API requestów i zarządza lifecyclem obiektów.
API Management to warstwa odpowiedzialna za zarządzanie Service, Route, Consumers i Plugins.
Plugins są napisane w LUA, ich zadaniem jest sprawdzanie requestów i ich odpowiednia modyfikacja, gdy jest to wymagane. Są one także integrowane z 3rd-party.
Admin API to JSON HTTP API, które może być użyte do zarządzania Kongiem.
Kong Manager - GUI w przeglądarce dla Admin API.

Fundamentals of Kong Gateway

Kong Manager
Jest to GUI w przeglądarce do monitorowania i konfigurowania Konga. Można tutaj:
-konfigurować nowe Routy i Service
-aktywować lub dezaktywować Pluginy
-monitorować performance przy pomocy dashboardów
-zarządzać grupami użytkowników przy pomocy RBAC
Domyślnie jest on dostępny na porcie 8002 dla http i 8445 dla https.

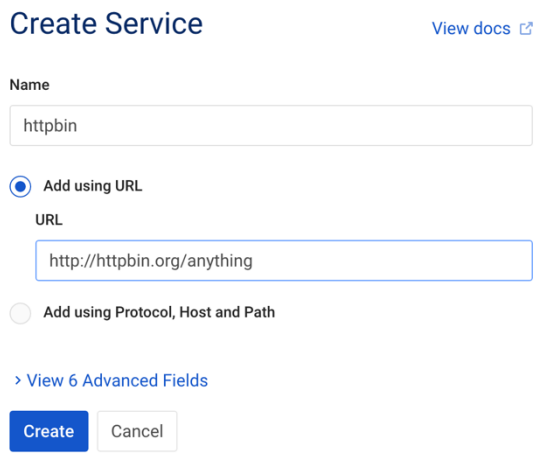
Tak wygląda GUI:



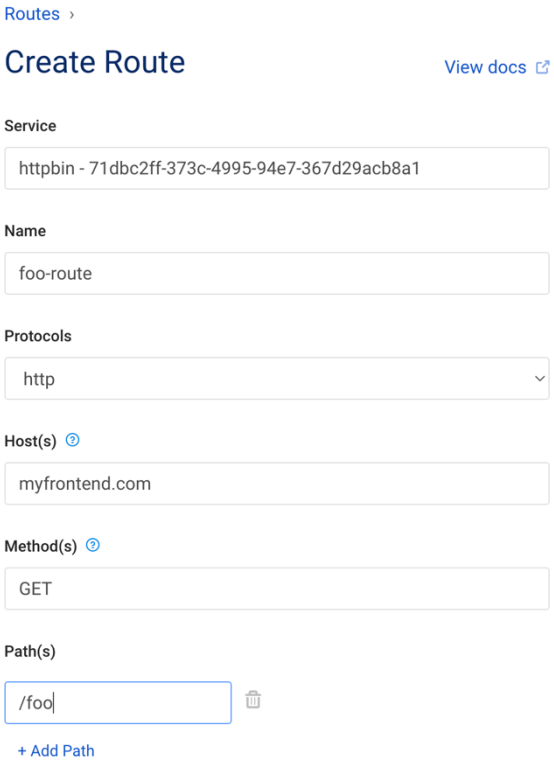
Mamy tutaj Workspace, które pozwalają dzielić resourcy na grupy wewnątrz tego samego clustra, tak aby różne zespoły mogły pracować tylko ze swoimi obiektami. Dostęp jest nadawany przez RBAC.

Głównymi konceptami Konga są Service, Route, Consumer, Plugins i Upstream:

Services and Routes
Service to upstream API lub mikroserwis np. data transformation microservice, billing API itp. Głównym atrybutem Service jest jego URL, określa on gdzie Kong powinien przekazać ruch sieciowy. Można go skonfigurować poprzez zwykły napis, lub precyzując protokół, host, port i ścieżkę oddzielnie. Gdy chcemy stworzyć nowy service to w GUI po prawej wybieramy API Gateway-> Service-> New Service. Tu przykładowo robimy service dla httpbin.org. W sekcji URL ustawiamy, gdzie ruch przychodzący do tego Service będzie przekierowywany:



Route to punkt wejściowy dla klienckich requestów, jest on potrzebny do tego, żeby określić, jaką drogą przychodzące do Kong Gatewaya requesty mają dojść do swoich Service, a potem do endpointów. Każdy Servie może mieć wiele Route połączonych ze sobą. W trakcie tworzenia Routa trzeba wybrać Service, do którego się on będzie łączyć, oraz przynajmniej jedną opcję z methods, hosts, headers lub paths, które określają pożądany endpoint. Route w GUI tworzymy w API Gateway->Routes->New Route:

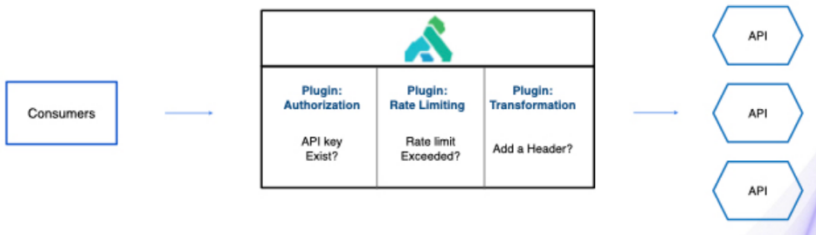


Dzięki tej operacji Kong Gateway będzie akceptował przychodzące GET requesty do *myfrontend.com/foo*. Będą one przypisane do Service *httpbin*, który prześle je do *http://httpbin.org/anything*, tak jak jest to ustawione w Service. Domyślnie Kong będzie operował takimi requestami na porcie 8000.

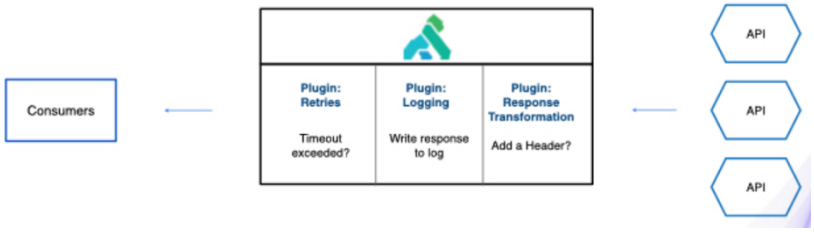
Teraz jeśli chcielibyśmy sprawdzić połączenie, to możemy to zrobić komendą:
http --proxy http://localhost:8000/ http://myfrontend.com/foo
Jako rezultat otrzymamy 200 i w odpowiedzi zobaczymy linijkę *Via:kong/2.5.0.0-beta2-enterprise-edition* mówiącą, że request przeszedł przez Kong Gateway. Gdybyśmy próbowali się połączyć z inną ścieżką, to dostalibyśmy 404 ponieważ, nie ma innej ścieżki ustawionej w Route.

Gdy używamy http z http://localhost można użyć skróconej wersji URL:
http :8000/foo host:myfrontend.com

Plugins
Wszystkie requesty przychodzące z zewnątrz przechodzą przez Kong Konnect Gateway. Używa on ustawionej konfiguracji, aby routować requesty do odpowiedniego backend Service. Dodatkowo Gateway uruchamia on Pluginy, aby zmodyfikować lekko request lub wykonać inne czynności (przykładowe Pluginy to Authentication, Rate Limitting, Logging, Transformations itp).
Należy pamiętać, że lepiej jest ustawić jedynie mniejsze modyfikacje requestów bezpośrednio w Gatewayu, do bardziej zaawansowanych, takich jak business logic transformation lepiej użyć np. Enterprise Service Bus, który zapewnia komunikację end-to-end pomiędzy serwisami.
Przykładowe użycie pluginów w Gatewayu:

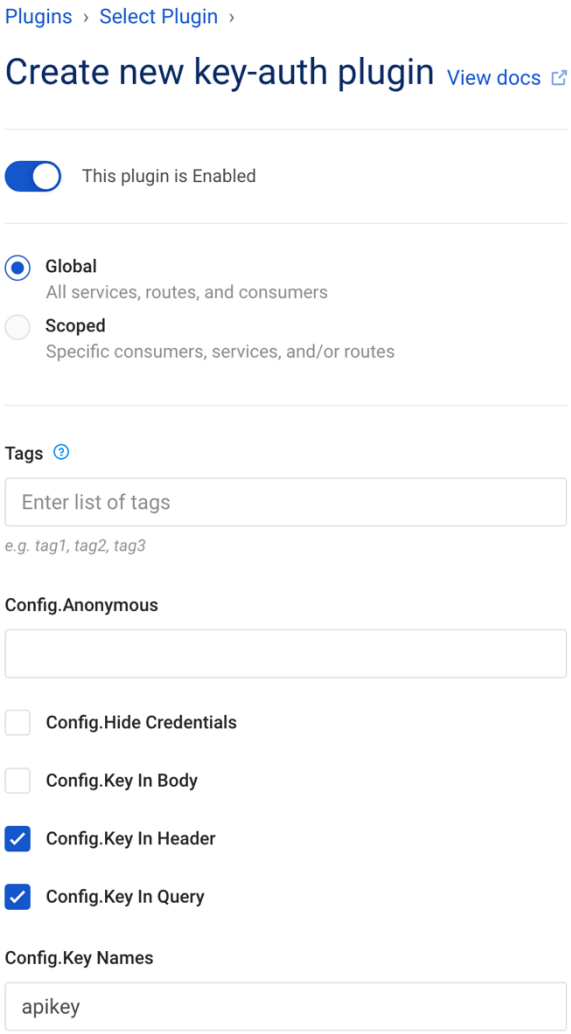


W tym przykładzie, gdy Consumer chce się połączyć z jakimś service, Pluginy sprawdzają, czy request ma odpowiedni API Key (jeśli będzie brakować klucza to dostaniemy 401), czy nie przekracza Rate Limitu (jeśli będzie za dużo requestów to dostaniemy 429 Too Many Requests) oraz modyfikują request dodając header (jeśli w dwóch poprzednich krokach nie ma błędów, to nastąpi dodanie headeru przed wysłaniem requesta do Service).
Pluginy mogą być też zaimplementowane do odpowiedzi wychodzących z Serivce:



Przykładowo ten plugin może dodać kolejny header, zapisać odpowiedź do logów i ustawić retry logic, gdy osiagniemy timeout.
Oficjalne Kongowe Pluginy znajdziemy w https://docs.konghq.com/hub, ale można też pobierać takie, które stworzyli inni oraz pisać je sami przy pomocy Lua.
Po ustawieniu Pluginu dla danego Service, każdy request do niego idący przejdzie przez ten Plugin. Przed instalacją danego pluginu warto sprawdzić, czy jest on kompatybilny z naszą wersją Konga.

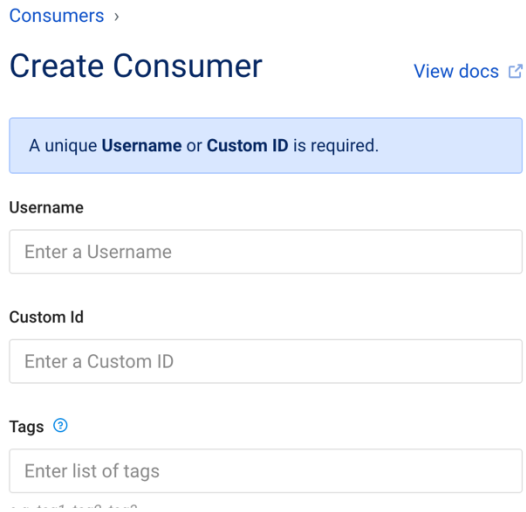
Gdy chcemy dodać Plugin to robimy to w API Gateway->Plugins->New Plugin.



W tym przykładzie dodajemy Key Authentication plugin globalnie, aby dodać autentykację do naszego Service (jakby inne były to też by dostały, bo to globalnie). Teraz tylko requesty z kluczem *apikey* będą zaakceptowane przez Kong Gateway.
Teraz jak byśmy spróbowali się połączyć tą komendą to będzie 401, bo nie podaliśmy żadnych credentiali.

Consumers

Consumer reprezentuje klientów, czyli zewnętrznych użytkowników, serwisy, którzy chcą się połączyć do Service i pozwala na kontrolowanie kto się może połączyć. Consumer pozwala także na logowanie ruchu poprzez Plugin i Kong Vitals. Można dzielić Consumerów na różne grupy, tak aby Kong lepiej obsługiwał i requesty i przygotowywał dla nich odpowiedzi. Jeśli chcemy ustawić specjalnie Plugin pod określonego Consumera to możemy stworzyć oddzielną instancję Plugina i w nim skonfigurować Service i Consumera, którzy będą brali udział w komunikacji.
Domyślnie http requesty przychodzące będą po porcie 8000 (non-secure) i 8443 (secure).
Aby stworzyć Consumera wchodzimy w API Gateway-> Consumer->New Consumer:



Aby nadać nowemu Consumerowi credentiale, gdy np. wymaga tego authorization Plugin wchodzimy w użytkownika->Credentials->New Key Auth Credential.
Teraz w trakcie sprawdzania połączenia jeśli sprecyzujemy apikey to dostaniemy 200 jako odpowiedź:
http --proxy http://localhost:8000/ http://myfrontend.com/foo apikey:JoePassword
W odpowiedzi dostaniemy też więcej szczegółów na temat samego połączenia:
"X-Consumer-Username":"Joe",
"Apikey":"JoePassword",
"X-Forwarded-Host":"myfrontend.com"

Podsumowując, odpowiedź mówi nam, że user Joe połączył się przez Konga do Service *httpbin*. Cały proces zawiera się w 3 krokach: Kong przesyła request do microsrevice, microservice odpowiada, Kong jako reverse proxy przesyła odpowiedź do Joe.

Upstream Targets

Gdy w przypadku połączenia endpoint nie jest pojedynczą instancją, tylko jest ich kilka, to możemy przed nie postawić load balancer, który będzie rozdzielał do nich ruch. Kong ma dwie metody load balancingu: straightforward DNS-based i ring-balancer. Ten drugi wymaga ustawienia upstreamu (Service) z target entities. W przypadku tej metody za dodawanie i usuwanie backend Service jest odpowiedzialny Kong Gateway i nie trzeba nigdzie updatować DNS.

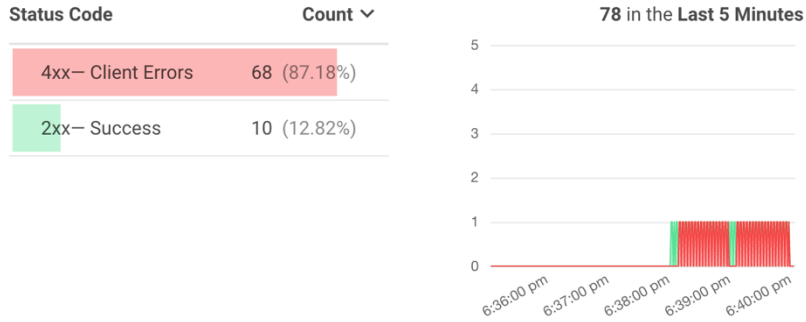
Monitoring with Vitals

Kong Vitals pozwala na:
-monitorowanie stanu i performance Gatewaya
-wizualizację API transactions, które przechodza przez Konga
-dostęp do statystyk
-wyświetlanie wszystkich anomalii w Gatewayu
Wszystkie te dane można wyświetlać w dashboardach i na wykresach, a także tworzyć z nich pliki .csv. Przy pomocy Kong API możemy też oglądać dodatkowe wizualizacje np. z innych systemów.

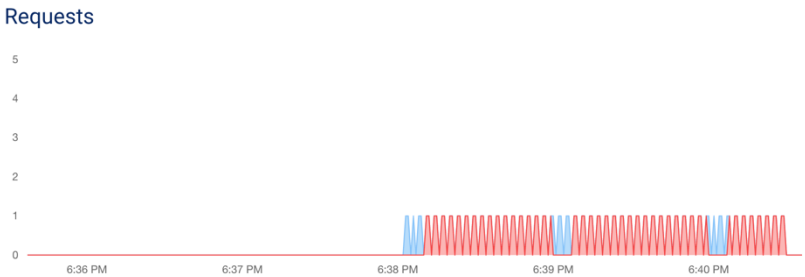
Przykład sprawdzający jak działa Kong Vitals - ustawimy limit requestów na usera Joe i spróbujemy go przekroczyć, aby obejrzeć błędy w Vitals.
Tworzymy nowy Plugin o typie Rate Limiting. Następnie przy pomocy poniższej pętli zwiększamy ruch:

```
for((i=1;i<=600;i++));
do
        sleep1;
        http --proxy http://localhost:8000/ http://myfrontend.com/foo apikey:JoePassword
done
```

Po chwili zobaczymy w konsoli, że dostajemy błąd 429 z wiadomością: *API rate limit exceeded.* Dodatkowo wchodząc w Vitals->Status Codes widzimy bardzo dużo 4xx błędów:



Dodatkowo w sekcji Workspaces mamy odpowiadające informacje:



Klikając w zakładkę Vitals na górze też zobaczymy na wykresie zwiększony ruch w clustrze.

Kong Admin API

Admin API to REST API do konfigurowania Kong Gateway, dostępne na porcie 8001 (http) i 8444 (https). API ma pełny dostęp do clustra, więc trzeba uważać, żeby go nie udostępniać.

Komendą *http GET* http://localhost:8001 *--headers* możemy sprawdzić, czy Kong API działa. Jako rezultat powinniśmy dostać 200. Flaga headers jest dodana po to, żeby widzieć mniejszą ilość informacji, bo tak to wszystko byłby duży output.

Aby stworzyć Service przy pomocy Kong API musimy użyć metody POST, ustawić /services jako ednpoint i w linku podać odpowiednie parametry:

Parametry te można też spiąć razem w URL.

Poniższa komenda pozwoli nam na wyświetlenie wszystkich Service:

http GET http://localhost:8001/services, a dodając *|jq -r .data[].name* na koniec, pofiltrujemy .json output, tak by wyświetlił tylko nazwy.

Aby stworzyć Route przy pomocy Kong API musimy użyć metody POST, ustawić /services/service-name/routes jako endpoint i podać parametry. Parametry do komendy można podać w formacie .json:

```
echo '{"name": "my-route", "paths": [ "/path/one", "/path/two" ]}' | http POST
http://KONG_ADMIN:8001/services/my-service/routes

OR

echo '{"name": "my-route", "paths": [ "/path/one", "/path/two" ]}' > data.json

http POST http://KONG_ADMIN:8001/services/my-service/routes < data.json
```

Ale też normalnie:
http POST :8001/services/mocking_service/routes name=mocking hosts:='["localhost"]'paths:='["/mock"]'
Aby sprawdzić istniejące Route należy uruchomić:
http :8001/routes |jq -r .data[].name

Aby dodać Plugn przy pomocy Kong API używamy metody POST, ustawiamy /services/service-name/plugins jako endpoint i konfigurację pluginu podajemy jako parametry:

```
http POST http://KONG_ADMIN:8001/services/my-service/plugins \
  name=<plugin-name> \
  config.item=<value>
```

Aby wyświetlić Pluginy dla danego Service uruchamiamy:
http GET :8001/services/mocking_service/plugins

Teraz, gdy chcemy dodać Consumera przy pomocy Kong API, to musimy najpierw stworzyć Consumera, a potem dodać mu credentiale na potrzeby naszego Pluginu:
http POST :8001/consumers username=Jane
http POST :8001/consumers/Jane/key-auth key=JanePassword
Poniższa komenda wyświetli Consumerów:
http GET :8001/consumers

Workspaces, Teams and RBAC
Workspace pozwalają na podzielenie obiektów organizacji na namespacy. Pozwala to pogrupować resourcy np. do danych zespołów, które powinny mieć dostęp tylko do swoich obiektów.

Teams pozwala dzielić administratorów i użytkowników na grupy, dzięki czemu będzie można do nich przypisywać oddzielnie dane przywileje przy pomocy RBAC.

Design API with Insomnia

OpenAPI Specification
OpenAPI Specification (OAS) definiuje ustandaryzowany, zgodny z różnymi językami programowania opis interfejsu dla HTTP APIs. Pozwala to ludziom i komputerom rozumieć możliwości serwisu bez dostępu do kodu źródłowego, dokumentacji i inspekcji ruchu sieciowego. Bazuje on na Swagger API Specification od Linux Foundation.
OpenAPI pozwala na lepszy opis HTTP API zawierając informacje o API, dostępnych endpointach, requestach, responsach i sposobach autentykacji. Można to porówna do menu w restauracji, klienci wiedzą jaki mają wybór ruchu sieciowego i czego się mogą spowiedzać.

Zastosowania OpenAPI Specification:
-Description, Validation and Linting - sprawdzanie, czy plik ma dobry syntax według ustalonych standardów.
-Data Validation - sprawdzanie, czy dane przechodzące przez API są poprawne, jest to sprawdzane podczas deploymentu i w trakcie normalnej pracy.
-Documentation Generation - tworzenie dokumentacji zrozumiałej dla człowieka.
-Code Generation - tworzenie kodu serwera i klienta w dowolnym języku programowania, oraz sprawdzanie go.
-Graphical Editors - możliwość tworzenia plików z opisami przy pomocy GUI.
-Mock Servers - tworzenie fakowych serwerów, które będą przydatne w testowaniu.
-Security Analysis - wyłapywanie potencjalnych zagrożeń w trakcie projektowania API.

Plik z informacjami o danym API to OpenAPI Document i jest on w formacie JSON lub YAML:

```
openapi: 3.0.0
protocol: https
tags:
  - description: Creates a random UUDI and returns it in a
JSON structure
    name: Generate UUID
layout: system/spec-renderer.html
info:
  contact:
    email: me@kennethreitz.org
    responsibleDeveloper: Kenneth Reitz
    url: https://kennethreitz.org
  description: >-
    A simple service returning ...
  title: UUID generator based on httpbin.org
  version: 1.0.3
paths:
  /uuid:
...
```

Na ten dokument składa się wiele sekcji:
-openapi - ustawiamy w niej wersję naszego dokumentu,
-info - źródło danych o naszym api, jest tu opis, mogą być dane kontaktowe. Obowiązkowymi polami są Title i Version (ta wersja to wersja API, a nie dokumentu),
-tags - można dodawać tagi do dokumentów, dzięki czemu inne toole, czy biblioteki mogą wykonywać na nich dane operacje,
-paths - tu ustawiane są aktualne endpointy dodane do URLa, supportowane motody HTTP, parametry requestu, responsy. Pole operationId odnosi się do operacji na docelowym serwerze. Wszystkie pathy musza się zaczynać od "/", bo są one dodane do URLa. Można też dodawać jakieś dynamiczne placeholders, żeby budować Dynamic URL zamiast Static URL.

```
paths:
  /pet:
    post:
      tags:
        - pet
      summary: Add a new pet to the store
      description: ""
      operationId: addPet
      requestBody:
        $ref: "#/components/requestBodies/Pet"
      responses:
        "405":
          description: Invalid input
    put:
      tags:
        - pet
```

-externalDocs - opisane są tu przydatne zewnętrzne dokumentacje
-servers- jest tu sprecyzowany API server i podstawowy URL, gdzie API jest przypisane
-components - często wiele API dzieli tę samą część kodu lub parametry. Aby uniknąć powtórzeń, można te definicje ustawić w globalnych komponentach i potem do nich się odnosić przy pomocy $ref. W przykładzie poniżej zduplikowana definicja usera jest podana tylko raz i jest skonfigurowana jako component:

```
components:
  requestBodies:
    UserArray:
      content:
        application/json:
          schema:
            type: array
            items:
              $ref: "#/components/schemas/User"
      description: List of user object
      required: true
    Pet:
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Pet"
```

Using Insomnia
Insomnia to tool pozwalający na edytowanie, testowanie i debugging, w celu wsparcia tworzenia API. Może być użyty samodzielnie lub być zintegrowany np. z GitHubem w celu współpracy. Pozwala na szybszy design Service, debuggowanie ich bez potrzeby używania zewnętrznych tooli jak curl, czy postman, oraz testowanie Service, które można wbudować w pipeline. Pozwala też na automatyczną konwersję API Specification z OpenAPI format na Kubernetes Custom Resource Definition, który jest używany przez Ingress Controller, lub na standardowy plik Kong config.
Insomnia ma dwa tryby pracy, jako zewnętrzna aplikacja i jako CLI.

Po uruchomieniu mamy 3 opcje pracy: Design (edytor do modyfikacji OpenAPI Specification), Debug i Test (testowanie tego co zrobiliśmy w Design i Debug).

Aby pobrać OpenAPI Document wybieramy sekcję Design i tam Create->Import From->URL/File (można np. z Githuba pobrać). W trybie Design, gdy modyfikujemy plik i popełnimy błąd to od razu będzie podświetlony. Mamy tu też możliwość przetestowania od razu naszego API klikając na GET lub Try It Out.

W sekcji Debug, wyświetlone zostaną błędy i np. może nie być ustawiona jakaś wartość zmiennej. Można tu też stworzyć Custom Request, który też pozwoli na przetestowanie połączenia. Możemy też wyeksportować nasze requesty przyciskiem Generate Code.
Ostateczne testy zrobimy w sekcji Test. Po stworzeniu naszego Custom Requestu i przetestowaniu go, możemy go zsynchronizować z GitHubem, aby tam go przechowywać, zapisywać zmiany i różne jego wersje. Zsynchronizowane mogą być zmienne środowiskowe, zmiany w Design, nasz Debugging i Test Suites.
Na koniec procesu możemy nasz przetestowany API wyeksportować jako Ingress Controller do Kuberenetesa lub jako Kong config.

Inso CLI tool
Jest to CLI Insomni do użycia w APIOps pipelines. Można go integrować z GitHubem, GitLabem i Bitbucketem. Przykładowe komendy:
inso run test - uruchamia test skonfigurowany w Insomni
inso export spec - eksportuje OpenAPI Specification
inso generate config - generuje Kong config lub Ingress Controller dla Kubernetesa.

Testowanie Inso CLI. Najpierw kopiujemy repo:
git clone https://github.com/johnfitzpatrick/httpbin-insomnia.git
Inso będzie zawsze szukać folderu .insomnia w swoim katalogu roboczym. Jest on zawsze wygenerowany poprzez synchronizację GitHuba puszczoną z sekcji Design. Jeśli inso nie znajdzie tego katalogu to spróbuje pracować w Designer app directory.
Gdy chcemy wykonać operację na OpenAPI Specification musimy podać jego ID w komendzie. Są one długie, ale tak jak w przypadku Dockera, można użyć skróconej nazwy np. gdy chcemy sprawdzić syntax danej specyfikacji: *inso lint spec spc_f347ab*
Przykładowy output z takiej komendy, gdy znajdziemy jakieś będy to:

```
labuser@d6d27748b3ab:~/httpbin-insomnia$ inso lint spec spc_f347ab
4 lint errors found.

20:7 - Bad indentation of a mapping entry
20:8 - Incomplete explicit mapping pair; a key node is missed
27:2 - Bad indentation of a mapping entry
27:16 - Mapping key must be a string scalar rather than object
```

Gdy chcemy uruchomić testy z CLI to trzeba podać Test Suite i Environment:
inso run testuts_64b --env localdev
Testy będą puszczone na wszystkie pathy i przykładowy output to:
TestSuite01
✓Returns 200with delay (2970ms)
✓Returns 200with uuid
2passing (4s)

Testy inso zwracają regularne kody wyjściowe (exit codes), więc można je łatwo zautomatyzować w CI/CD lub APIOps pipeline. W Linuxie komenda $? Zwraca kody wyjściowy poprzedniej komendy, więc *echo $?* zwróci to do standardowego wyjścia. Kod wyjściowy o wartości 0 znaczy, że poprzednia komenda zakończyła się sukcesem, a jak będą jakieś błędy, to odpowiedź wyniesie 1.

```
labuser@d6d27748b3ab:~/httpbin-insomnia$ inso lint spec spc_f347ab
4 lint errors found.

20:7 - Bad indentation of a mapping entry
20:8 - Incomplete explicit mapping pair; a key node is missed
27:2 - Bad indentation of a mapping entry
27:16 - Mapping key must be a string scalar rather than object
labuser@d6d27748b3ab:~/httpbin-insomnia$ echo $?
1
```

inso run test uts_64b --env localdev

TestSuite01
✓Returns 200with delay (3255ms)
✓Returns 200with uuid
2passing (3s)
echo $?
0

Eksportowanie OpenAPI Specification do pliku yaml:
inso exportspec spc_f347ab --output uuid-openapi.yaml

Generowanie Kong configuration na podstawie pliku yaml:
inso generate config uuid-openapi.yaml -o uuid-kong.yaml (ten drugi plik output to ta konfiguracja). Takiego pliku można teraz użyć jako konfigurację do Kong Gatewaya.

Deploying APIs

Understanding Declarative Configuration

W podejściu deklaratywnym precyzujemy całą konfigurację, którą chcemy otrzymać np. tak jak w pliku yaml. Podejście imperatywne natomiast to, uruchamianie komend jedna po drugiej tak, aby doprowadziło nas to do żądanego rezultatu:

Jeśli chcemy to zastosować w CI/CD to imperatywnie napiszemy skrypt, który będzie łączył się z różnymi API i deployował je na platformę, będzie rejestrował endpoint, dodawał policy itp. W podejściu deklaratywnym nie musimy się o to martwić, mówimy tylko, jak chcemy żeby wyglądała nasza platforma, a sama platforma zajmuje się deploymentem.

Gdy ustawiamy Kong Gateway ręcznie (imperatywnie) nie zawsze będziemy w stanie dokładnie śledzić jednorazowe komendy, dlatego poleca się podejście deklaratywne.

Administering Kong Gateway using decK

Deck to Cli tool używający deklaratywnej konifguracji Konga i pozwalający na eksport, import i zarządzanie Kongiem w deklaratywnym podejściu. Konfiguracja może być wersjowana i zarządzana przez różne zespoły na ich własne potrzeby.

Zarządzanie polega na ustawieniu pożądanego stanu w pliku yaml i pozwolenie decK Sync na utworzenie go, bez ręcznego uruchamiania komend. Konfiguracja może być przechowywana w GitHubie. decK może też odwracać zmiany i tworzyć backupy konfiguracji. Może on pracować razem z inso i być zautomatyzowany w CI/CD lub APIOps pipeline.

Komendą *deck version* sprawdzimy wersję decK i zobaczymy, czy jest w ogóle zainstalowany. Kong domyślnie nasłuchuje na porcie 8001 i jest ustawiony przez *admin_listen* w *'/etc/kong/kong.conf'* np. *admin_listen = 0.0.0.0:8001.* decK używa tej samej wartości żeby pingować Kong Gateway.

deck ping - komenda sprawdzająca, czy decK może się łączyć z Kongiem.

deck dump - komenda robiąca backup konfiguracji Konga do pliku kong.yaml.

deck validate - ta komenda sprawdza, czy kong.yaml nie ma żadnych błędów w składni. Jeśli nie ma żadnego outputu to znaczy, że jest okej. Tak jak inso, decK też używa standardowych kodów wyjściowych, więc można go łatwo zautomatyzować w CI/CD.

deck diff-s kong.yaml - komenda pokazująca różnice pomiędzy lokalnym plikiem kong.yaml, a rzeczywistą konfiguracją Kong Gatewaya. Przykładowy output z różnicami:

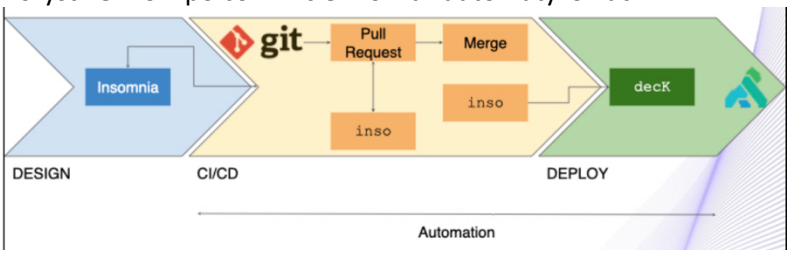
```
labuser@d6d27748b3ab:~/httpbin-insomnia$ deck diff -s kong.yaml
creating consumer Jason
updating key-auth sword for consumer 3cb6910e-c1f8-483c-850d-2897d6d7b98d  {
   "consumer": {
-     "id": "4d7c04ca-cc48-401a-8e8b-fe69ef239acd"
+     "id": "3cb6910e-c1f8-483c-850d-2897d6d7b98d"
   },
   "id": "24a5af80-abee-4552-a9ce-9753b1927c2c",
   "key": "JoePassword"
 }

deleting consumer Joe
Summary:
  Created: 1
  Updated: 1
  Deleted: 1
```

deck sync - komenda kopiująca lokalną konfigurację do rzeczywistego Kong Gatewaya. Nadpisze to istniejącą dotychczas konfigurację i nowa wejdzie w życie. Po uruchomieniu tej komendy *deck diff* nie pokaże już żadnej różnicy.
deck reset - ta komenda czyści bazę danych Konga i stawia ją od nowa, więc trzeba uważać z tą komendą. Jeśli chcemy teraz wszystko odtworzyć, to możemy to zrobić bardzo łatwo z lokalnego pliku kong.yaml używając komendy *deck sync*.

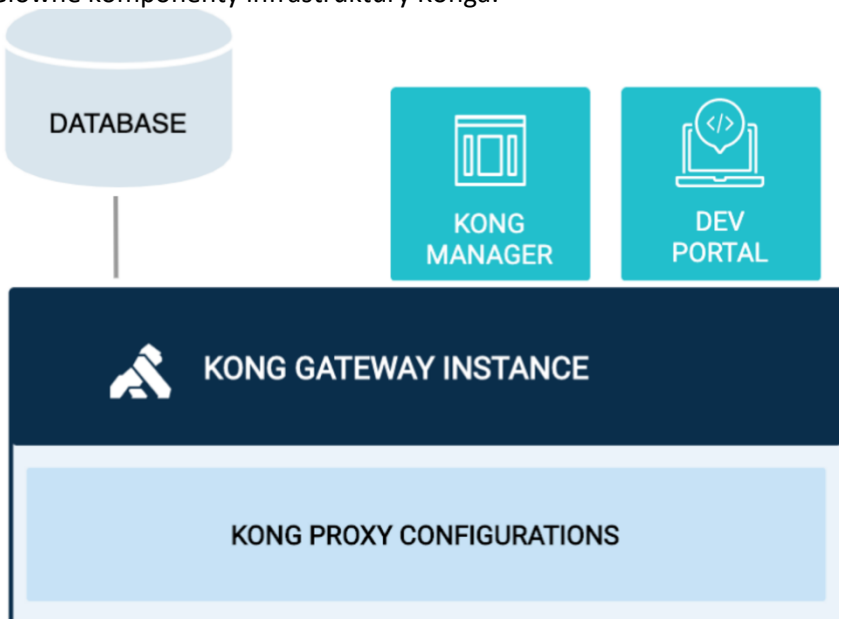
Introduction to APIOps

Przykład wykorzystania tych wszystkich tooli w automatyzacji przy pomocy CD pipeline.
Najpierw zaczynamy od projektowania naszego API w Insomni. Tam też je testujemy, jak już jest przetestowane przy pomocy mock serverów to robimy commit. Wypychamy nasze API np. do GitHuba, tak by inni pracownicy mogli z niego korzystać i byśmy mogli śledzić zmiany. Gdy chcemy wprowadzić jakieś zmiany to robimy to przez pull request.
Następnie przy pomocy inso CLI sprawdzamy syntax, testujemy i dodajemy policy do naszego API i updatujemy pull request. Gdy zmiana jest zaakceptowana, następuje merge do master brancha.
Teraz przy pomocy inso możemy wygenerować plik Kong config lub Ingress Gateway do Kubernetesa. Na koniec zmiana konfiguracji może być zdeployowana bezpośrednio do Konga przy pomocy decK. Wszystkie kroki po commicie można zautomatyzować:



Installation of Kong Gateway

Installation Overview
Główne komponenty infrastruktury Konga:



Kong Gateway - zajmuje się przetwarzaniem requestów do API
Kong Manager - GUI w przeglądarce do monitorowania i zarządzania Gatewayem
Kong Developer Portal - pozwala na operacje na API
Database - przechowuje skonfigurowane obiekty, takie jak Routes, Services i Plugins. Może to być Postgres lub Cassandra.

Są różne rodzaje instalacji:
-embedded (deployment wszystkich serwisów na jednym node - tylko dla developmentu. Występuje tu tylko jedna instancja Konga - Control Plane i Data Plane są razem na node),
-distributed (jest tu wiele instancji Konga na różnych nodach i każdy node jest połączony do bazy dancyh. Dodatki takie jak Kong Manager i Developer Portal są tylko na Control Plane)
-hybrid (Control Plane i Data Plane oddzielone na osobne nody, tylko Control Plane ma dostęp do bazy danych. Control Plane monitoruje i zarządza API, a Data Plane przekazuje przychodzące requesty i je proxuje. Data Plane otrzymuje konfiguracje z Control Plane i może być zainstalowane dowolnie on-prem lub cloud).

Jest wiele technicznych aspektów do wybrania przed instalacją:
-rozmiar resourców: CPU, RAM, bandwith, opóźnienie i przepustowość, bazy danych, rozmiar memory cache itp.
-domyślne porty: trzeba dobrać porty dla Proxy, Admin API, Kong Manager i Dev portal dla http i https.
-rozważania na temat DNS: należy dobrać hostname dla wyżej wymienionych serwisów i zarządzanie cookies.
-network i firewall: ustawienie firewalla, otwieranie portów dla klientów, ustawienia proxy
-bezpieczeństwo i certyfikaty: ustawienie bezpieczeństwa Admin API, enkrypcja danych, zarządzanie certyfikatami i secretami
-licencja

W przykładzie będziemy instalować Konga przy pomocy Docker Compose.

Najpierw w celu zabezpieczenia należy skonfigurować certyfikaty SSL. Użyjemy tu wildcard certificate dla Admin API, Kong Manager GUI, Dev Portal API i Dev Portal GUI. Plik docker-compose będzie szukał certyfikatów w katalogu */srv/shared/ssl-certs*.
Teraz możemy zacząć instalację Kong Gatewaya. Najpierw trzeba uruchomić Dockerowe kontenery, aby deployować w nich Konga. Zrobimy to za pomocą:
kong-gateway-operations/docker-compose.yaml at main · kong-education/kong-gateway-operations · GitHub
Stworzy to 5 kontenerów: kong-dp (Kong Data Plane), kong-cp (Kong Control Plane), db (Database), httpbin (przykładowy Service do testowania) i smtp-server (email service). Wszystkie te obiekty będą podłączone do sieci training-kong-net.

Po zainstalowaniu możemy sprawdzić, czy nasza instalacja przebiegła pomyślnie. W tym ćwiczeniu możemy się połączyć z Kong Admin API w kongcluster:8001, więc komenda: *http --headers GET kongcluster:8001* umożliwi nam połączenie. Jak będzie okej, to dostaniemy 200.
Wyświetlając wartość zmiennej: *echo $KONG_MANAGER_URI* otrzymamy link do sprawdzenia Kong Managera.

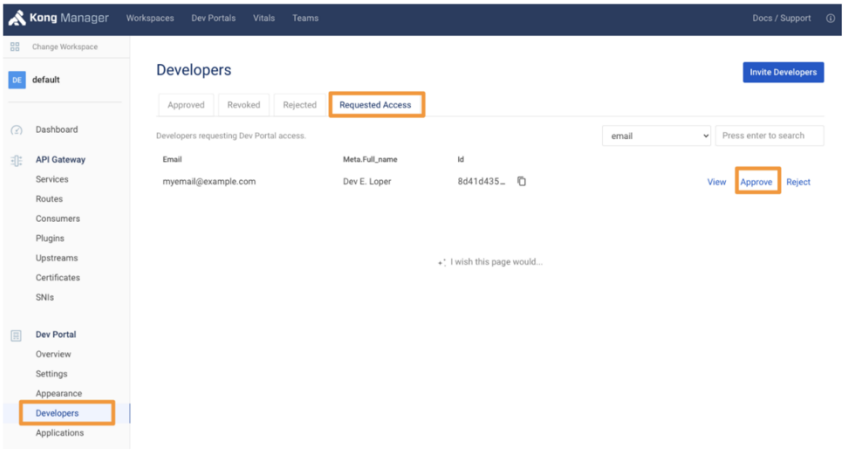
Teraz trzeba wgrać licencję do Konga. Jeśli mamy hybrydowy rodzaj deploymentu, to wystarczy ją zdeployować tylko w Control Plane i będzie ona dystrybuowana do clustrów Data Plane. W tradycyjnym deploymencie trzeba ją wgrać osobno na wszystkie nody. Jest klika sposobów na wgranie licencji, oto jak Kong ją sprawdza:

1. Sprawdzana jest wartość zmiennej KONG_LICENSE_DATA
2. Sprawdzana jest domyślna lokalizacja pliku z licencją: /etc/kong/license.json
3. Sprawdzana jest zawartość pliku ustawionego w KONG_LICENSE_PATH
4. Deployment licencji bezpośrednio przez /licenses Admin API endpoint. Robimy to przy pomocy komendy:
   http POST "kongcluster:8001/licenses" payload=@/etc/kong/license.json

Po tej akcji trzeba zrekreować Control Plane:
docker-compose stop kong-cp;docker-compose rm-f kong-cp;docker-compose up -d kong-cp

Teraz można władować pożądaną konfigurację Konga przy pomocy decK. Najpierw trzeba zupdatować konfigurację decK:
sed-i "s|KONG_ADMIN_API_URI|$KONG_ADMIN_API_URI|g"deck/deck.yaml
Komenda *deck diff --config deck/deck.yaml -s deck/default-entities.yaml --workspace default* pokaże nam co będzie zmienione w trakcie ładowania konfiguracji, a poniższa komenda zapisze nową konfigurację na naszym Kong Gateway:
deck sync --config deck/deck.yaml -s deck/default-entities.yaml --workspace default

Teraz można też skonfigurować Dev Portal. Domyślnie nie jest on uruchomiony, więc trzeba najpierw to zrobić. Po uruchomieniu, domyślnie będzie używać basic-auth do autentykacji, więc trzeba będzie stworzyć i potwierdzić Kong Developera.
Włączanie odbywa się w Kong Managerze, wybieramy zakładkę Dev Portals-> Set Up Dev Portal-> Enable Developer Portal i dostaniemy do niego link. Po wejściu w link możemy tam stworzyć konto dla Developera, zanim jednak będzie mógł się zalogować, musimy go zaakceptowac w Kong Managerze w Dev Portal->Developers:


Kong Manager
Workspaces
Dev Portals
Vitals
Teams
Docs / Support
Change Workspace
DE
default
Dashboard
API Gateway
Services
Routes
Consumers
Plugins
Upstreams
Certificates
SNIs
Dev Portal
Overview
Settings
Appearance
Developers
Applications
Developers
Invite Developers
Approved
Revoked
Rejected
Requested Access
Developers requesting Dev Portal access.
email
Press enter to search
Email
Meta.Full_name
Id
myemail@example.com
Dev E. Loper
8d41d435...
View
Approve
Reject
I wish this page would...