零声教育 Mark 老师 QQ : 2548898954

Kong 简介

> Kong 是一款基于 openresty 编写的高可用、易扩展的开源 API Gateway 项目。
>
> Kong 支持两种工作模式：一种是不使用数据库；另一种是使用数据库；可用的数据库为 PostgreSQL、Cassandra（分布式 NoSQL 数据库）

Kong 参考文档

- 官方网站：https://konghq.com
- 官方文档：https://docs.konghq.com
- 项目地址：https://github.com/Kong/kong
- 中文文档：https://github.com/wanglongsxr/kong-docs-cn 基于1.1.x版本

docker 安装（推荐）

创建 Kong 网络

> 创建自定义 Docker 网络以允许容器相互发现和通信

```
sudo docker network create kong-net
```

安装 PostgreSQL

```
# 创建 PostgreSQL 容器
sudo docker run -d --name kong-database --network=kong-net -p 5432:5432 -e "POSTGRES_USER=kong" -e "POSTGRES_DB=kong" -e "POSTGRES_PASSWORD=kong" --restart always postgres:9.6

# 使用 Kong 容器运行进行数据库初始化
sudo docker run --rm --network=kong-net -e "KONG_DATABASE=postgres" -e "KONG_PG_HOST=kong-database" -e "KONG_PG_USER=kong" -e "KONG_PG_PASSWORD=kong" -e "KONG_CASSANDRA_CONTACT_POINTS=kong-database" kong:2.5.0 kong migrations bootstrap
```

创建 Kong 容器

```
# -u -root:  Kong 启动过程中，需要权限创建一些文件和文件夹；
sudo docker run -d --name kong --network=kong-net -u root -e "KONG_DATABASE=postgres" -e "KONG_PG_HOST=kong-database" -e "KONG_PG_USER=kong" -e "KONG_PG_PASSWORD=kong" -e "KONG_CASSANDRA_CONTACT_POINTS=kong-database" -e "KONG_PROXY_ACCESS_LOG=/dev/stdout" -e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" -e "KONG_PROXY_ERROR_LOG=/dev/stderr" -e "KONG_ADMIN_ERROR_LOG=/dev/stderr" -e "KONG_ADMIN_LISTEN=0.0.0.0:8001,0.0.0.0:8444 ssl" -p 8000:8000 -p 8443:8443 -p 8001:8001 -p 8444:8444 --restart always kong:2.5.0
```

搭建 Konga

> Konga 是 Kong 的可视化 API 操作工具；
>
> https://hub.docker.com/r/pantsel/konga/

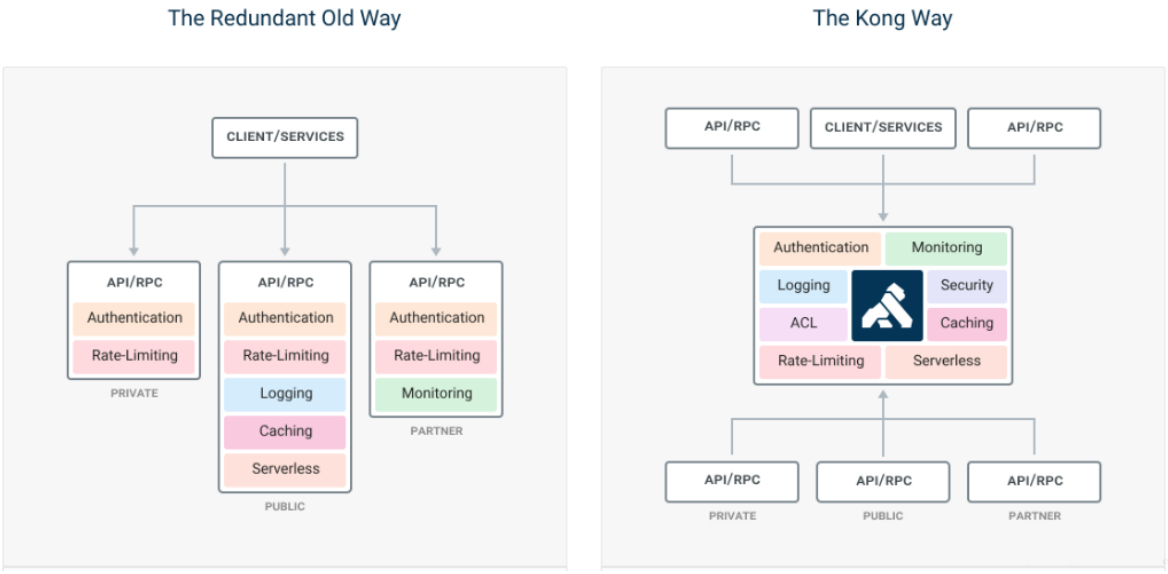
```
# 复用上面的数据库容器
sudo docker run --rm --network=kong-net pantsel/konga:0.14.9 -c prepare -a postgres -u postgresql://kong:kong@kong-database/konga

# 运行 Konga 容器
sudo docker run -d -p 1337:1337 --network=kong-net -e "DB_ADAPTER=postgres" -e "DB_URI=postgresql://kong:kong@kong-database/konga" -e "NODE_ENV=production" --name konga pantsel/konga:0.14.9
```

测试 Kong & Konga

```
# 输入下列命令，若返回一个大的 json 串，说明安装成功
curl http://localhost:8001

# 浏览器运行
http://192.168.31.91:1337
```

为什么需要 API 网关?

> 在微服务架构之下，服务被拆的非常零散，降低了耦合度的同时也给服务的统一管理增加了难度。在旧的服务治理体系之下，**鉴权，限流，日志，监控**等通用功能需要在每个服务中单独实现，这使得系统维护者没有一个全局的视图来统一管理这些功能。API 网关致力于解决的问题便是为微服务纳管这些**通用**的功能，在此基础上**提高系统的可扩展性**。微服务搭配上 API 网关，可以使得**服务**本身更**专注于自己的领域**，很好地对**服务调用者**和**服务提供者**做了**隔离**。



Kong 有哪些特性?

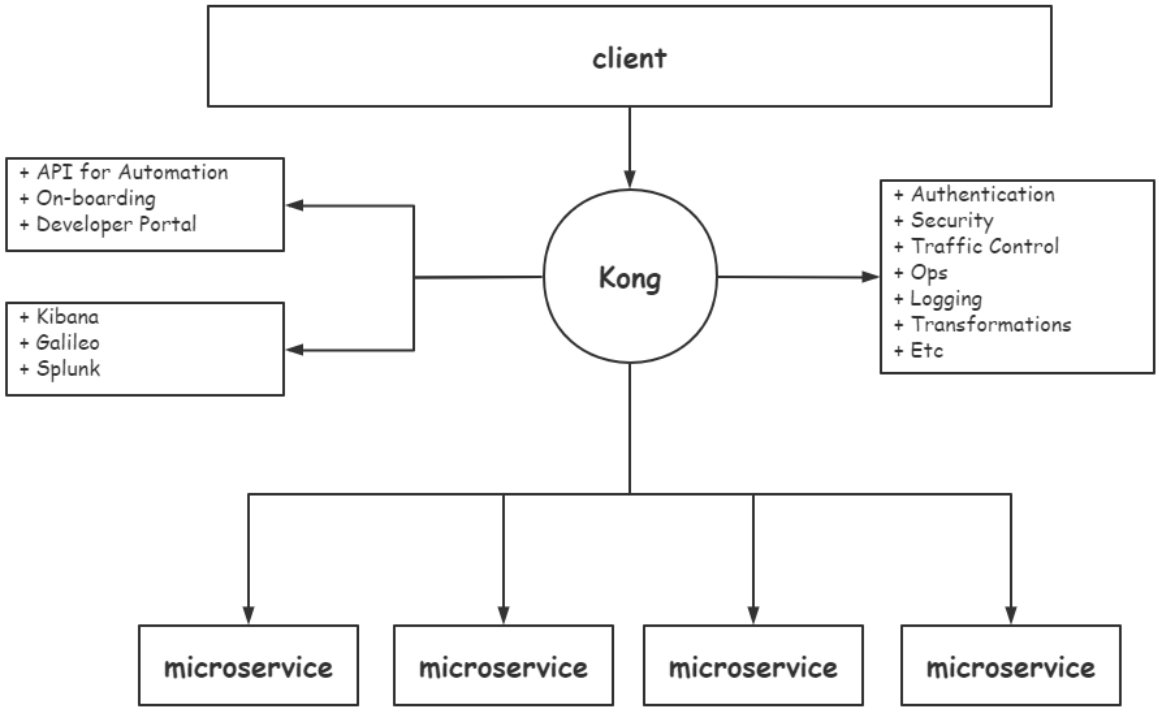
- 云原生：与平台无关，Kong 可以从裸机运行到 Kubernetes；
- 动态路由：Kong 的背后是 OpenResty，所以从 OpenResty 继承了动态路由的特性；
- 熔断
- 健康检查
- 日志：可以记录通过 Kong 的 HTTP，TCP，UDP 请求和响应；

- 鉴权：权限控制，IP 黑白名单，同样是 OpenResty 的特性；
- SSL：Setup a Specific SSL Certificate for an underlying service or API；
- 监控：Kong 提供了实时监控插件；
- 认证：支持 HMAC， JWT，Basic，OAuth2.0 等常用协议；
- 限流
- REST API：通过 Rest API 进行配置管理，从繁琐的配置文件中解放；
- 可用性：天然支持分布式；
- 高性能：利用 nginx 的非阻塞 io 模型；
- 插件机制：提供众多开箱即用的插件，且有易于扩展的自定义插件接口，用户可以使用 Lua 自行开发插件；

kong 体系结构

- Kong 核心基于 Openresty 构建，实现了请求/响应的 lua 处理化；
- Kong 插件拦截请求/响应，等价于拦截器，实现请求/响应的 AOP 处理；
- Kong Restful 管理 API 提供了 API / API 消费者 / 插件的管理；
- 数据中心用于存储 Kong 集群节点信息、API、消费者、插件等信息，目前提供了 PostgreSQL 和 Cassandra支持，如果需要高可用建议使用 Cassandra；
- Kong 集群中的节点通过 gossip 协议(redis cluster)自动发现其他节点，当通过一个 Kong 节点的管理 API 进行一些变更时也会通知其他节点。每个 Kong 节点的配置信息是会缓存的，如插件，那么当在某一个 Kong 节点修改了插件配置时，需要通知其他节点配置的变更。

Kong 工作流程



> API 网关可以通过实现一些中间组件来解决一些问题，这些中间组件的功能就不需要到每个微服务中实现了，这样不同的团队可使用不同的方式来实现了不同的微服务。
>
> API 网关不仅可以帮你解决 API 的管理部分，而且还可以解决下面两件事情：
>
> - 分析（Analytics） – API 网关可以和你的分析基础设施保持透明的交互和通信，因为 API 网关是每个请求（request）的入口，必经地。API 网关可以看到所有的数据，可以知道经过你的 service 的流量。这就相当于你有一个集中的地方，你可以把所有的这些信息 push 到你的监控或分析工具，比如 Kibana 或者 Splunk。
> - 自动化（Automation） –网关有助于自动化部署，还可以实现自动化登入验证。 什么是登入呢？ 如果你有 API，并且你希望有身份验证，你可能需要一些功能可以允许用户为该 API 创建登入凭据（credentials）然后开始使用（消费）API。 开发人员门户网站或你的文档中心等都可以与网关集成来配置这些凭据（credentials），这样就不用从头开始构建一些功能了。
>
> Kong 默认是缺失如 API 级别的超时、重试、fallback 策略、缓存、API 聚合、AB 测试等功能，这些功能插件需要企业开发人员通过 Lua 语言进行定制和扩展。

```
init_by_lua_block {
    kong = require 'kong'
    kong.init() // 完成 Kong 的初始化，路由创建，插件预加载等
}
init_worker_by_lua_block {
    kong.init_worker() // 初始化 Kong 事件，worker 之间的事件，由 worker_events 来处理，cluster 节点之间的事件，由 cluster_events 来处理，缓存机制
}

upstream kong_upstream {
    server 0.0.0.1;

    balancer_by_lua_block {
        kong.balancer() //负载均衡
    }
    keepalive 60;
}
server {
    server_name kong;
    listen 0.0.0.0:8000 reuseport backlog=16384;
    listen 0.0.0.0:8443 ssl http2 reuseport backlog=16384;

    rewrite_by_lua_block {
        kong.rewrite() //插件生效策略的筛选,并执行对应的操作，只能处理全局插件(kong插件级别，全局(作用于所有请求),route(作用于当前路由)，service(作用于匹配到当前service的所有请求))，路由匹配未开始。
    }
    access_by_lua_block {
        kong.access() //1.完成路由匹配，2.确认加载的插件(并加入缓存) 3.进入balancer阶段
    }

    header_filter_by_lua_block {
        kong.header_filter() //遍历在缓存中的插件列表，并执行
    }
    body_filter_by_lua_block {
        kong.body_filter() //遍历在缓存中的插件列表，并执行
    }
    log_by_lua_block {
```

```
        kong.log() //遍历在缓存中的插件列表，并执行
    }
    location / {
        proxy_pass            $upstream_scheme://kong_upstream$upstream_uri;
    }
}
```

从 nginx 配置到 Kong 配置

nginx

```
upstream mark_upstream {
    server 192.168.31.91:3000 weight=100;
}

server {
    listen 80;
    location /hello {
        proxy_pass mark_upstream;
    }
}
```

Kong

> https://docs.konghq.com/gateway-oss/2.5.x/admin-api/

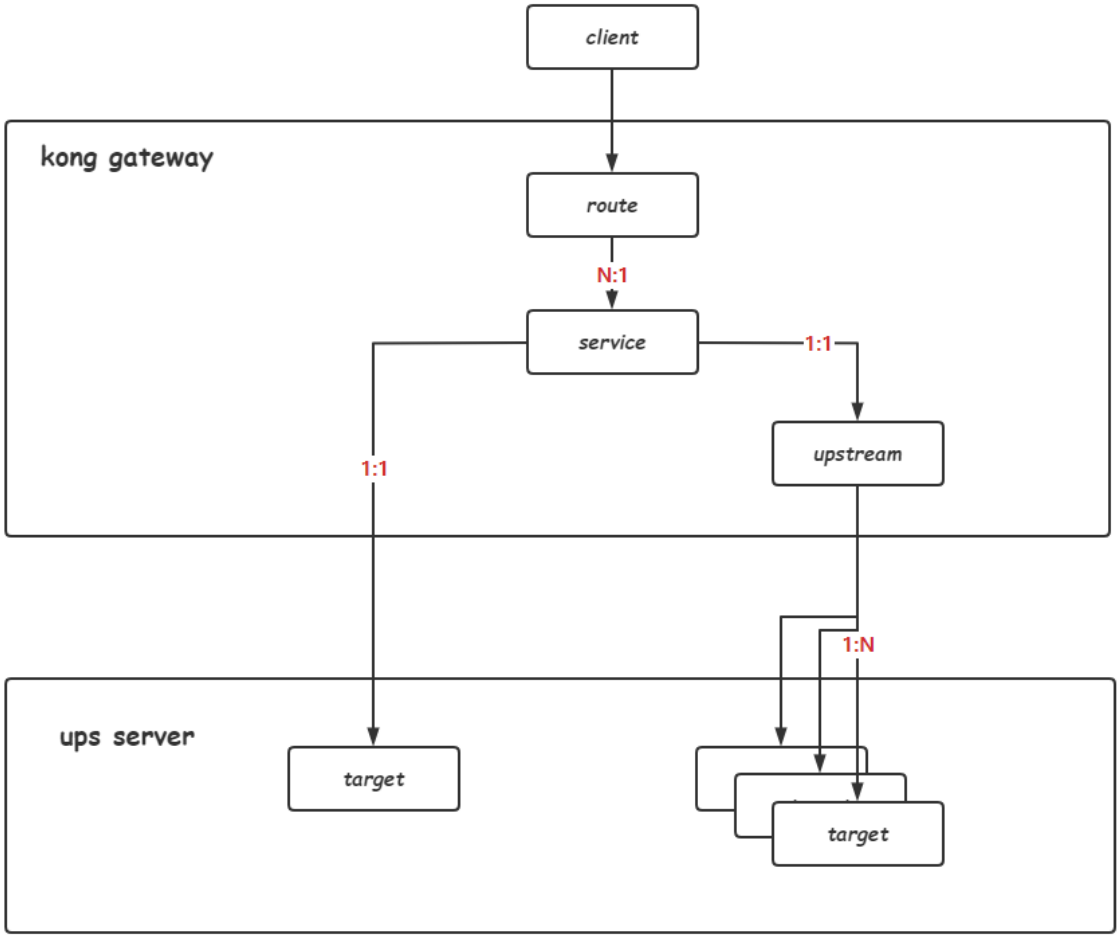
```
# 配置 upstream
curl -X POST http://localhost:8001/upstreams --data "name=mark_upstream"

# 配置 target
curl -X POST http://localhost:8001/upstreams/mark_upstream/targets --data "target=192.168.31.91:8888" --data "weight=100"
curl -X POST http://localhost:8001/upstreams/mark_upstream/targets --data "target=192.168.31.91:9999" --data "weight=100"

# 配置 service
curl -X POST http://localhost:8001/services --data "name=hello" --data "host=mark_upstream"
# 配置 route
curl -X POST http://localhost:8001/routes --data "paths[]=/hello" --data "service.id=15722364-296f-4624-9026-ceff2d2166f2"
```

Kong 核心四对象

> Kong 涉及到 upstream，target，service，route 概念；
>
> upstream 是对上游服务器的抽象；
>
> target 代表了一个物理服务，是 ip + port 的抽象；
>
> service 是抽象层面的服务，他可以直接映射到一个物理服务（host 指向 ip + port），也可以指向一个 upstream 来做到负载均衡；
>
> route 是路由的抽象，他负责将实际的 request 映射到 service；

四对象关系

> upstream 和 target： 1 : n;
>
> service 和 upstream： 1 : 1 或 1 : 0； （service 也可以直接指向具体的 target，相当于不做负载均衡）
>
> service 和 route：1 : n;



插件机制

> Kong 的插件机制是 Kong 最核心的功能；Kong 要解决的问题是承担所有服务共同需要的那部分功能；插件可能是作用全局的，也可能是作用局部的，比如大部分插件是作用在 service 或 route 上；
>
> Kong 的插件列表：https://konghq.com/plugins/

```
# 为 hello 服务添加 50/s 的限流，作用在 service 上
curl -X POST http://localhost:8001/services/hello/plugins \
--data "name=rate-limiting" \
--data "config.second=50"

# 为某个 routeId 添加 50/s 的限流，作用在 routes 上
curl -X POST http://localhost:8001/routes/{routeId}/plugins \
--data "name=rate-limiting" \
--data "config.second=50"
```

Kong 网关插件

- 身份认证插件：Kong提供了 Basic Authentication、Key authentication、**OAuth2.0 Authentication**、HMAC authentication、JWT、LDAP authentication 认证实现。
- 安全控制插件：ACL（访问控制）、CORS（跨域资源共享）、动态 SSL、**IP限制**、爬虫检测实现。
- 流量控制插件：请求限流（基于请求计数限流）、上游响应限流（根据 upstream 响应计数限流）、**请求大小限制。**限流支持本地、Redis 和集群限流模式。
- 分析监控插件：Galileo（记录请求和响应数据，实现 API 分析）、Datadog（记录 API Metric 如请求次数、请求大小、响应状态和延迟，可视化 API Metric）、Runscope（记录请求和响应数据，实现 API 性能测试和监控）。
- 协议转换插件：请求转换（在转发到 upstream 之前修改请求）、响应转换（在 upstream 响应返回给客户端之前修改响应）。
- 日志应用插件：TCP、UDP、HTTP、File、Syslog、StatsD、Loggly 等。

Kong 应用

负载均衡（课上）

黑白名单（课上）

限流（课上）

```
$ curl -i -X POST \
--url http://localhost:8001/services/example-service/plugins/ \
--data 'name=rate-limiting' \
--data 'config.minute=100'
```

Basic Auth

> 基本认证是一种用来允许 Web 浏览器或其他客户端程序在请求时提供用户名和口令形式的身份凭证的一种登录验证方式，通常**用户名和密码**会通过 HTTP 头传递。
>
> 格式：base64(username:userpass)

使用场景

- 内部网络，或者对安全要求不高的网络；
- 结合 https 一起使用，https 保证网络的安全性，然后通过基本认证来做客户端身份识别；