主讲老师：Fox

Apache APISIX版本： v3.0.0

1. Apache APISIX介绍

1.1 什么是Apache APISIX

Apache APISIX 是一个动态、实时、高性能的云原生 API 网关，提供了负载均衡、动态上游、灰度发布、服务熔断、身份认证、可观测性等丰富的流量管理功能。可以使用 Apache APISIX 处理传统的南北向流量，也可以处理服务间的东西向流量。同时，它也支持作为 K8s Ingress Controller 来使用。

Apache APISIX 官网：https://apisix.apache.org/

APISIX主要特性

- 多平台支持：APISIX 提供了多平台解决方案，它不但支持裸机运行，也支持在 Kubernetes 中使用，还支持与 AWS Lambda、Azure Function、Lua 函数和 Apache OpenWhisk 等云服务集成。
- 全动态能力：APISIX 支持热加载，这意味着你不需要重启服务就可以更新 APISIX 的配置。
- 精细化路由：APISIX 支持使用 NGINX 内置变量做为路由的匹配条件，你可以自定义匹配函数来过滤请求，匹配路由。
- 运维友好：APISIX 支持与以下工具和平台集成：HashiCorp Vault、Zipkin、Apache SkyWalking、Consul、Nacos、Eureka。通过 APISIX Dashboard，运维人员可以通过友好且直观的 UI 配置 APISIX。
- 多语言插件支持：APISIX 支持多种开发语言进行插件开发，开发人员可以选择擅长语言的 SDK 开发自定义插件。

1.2 APISIX架构

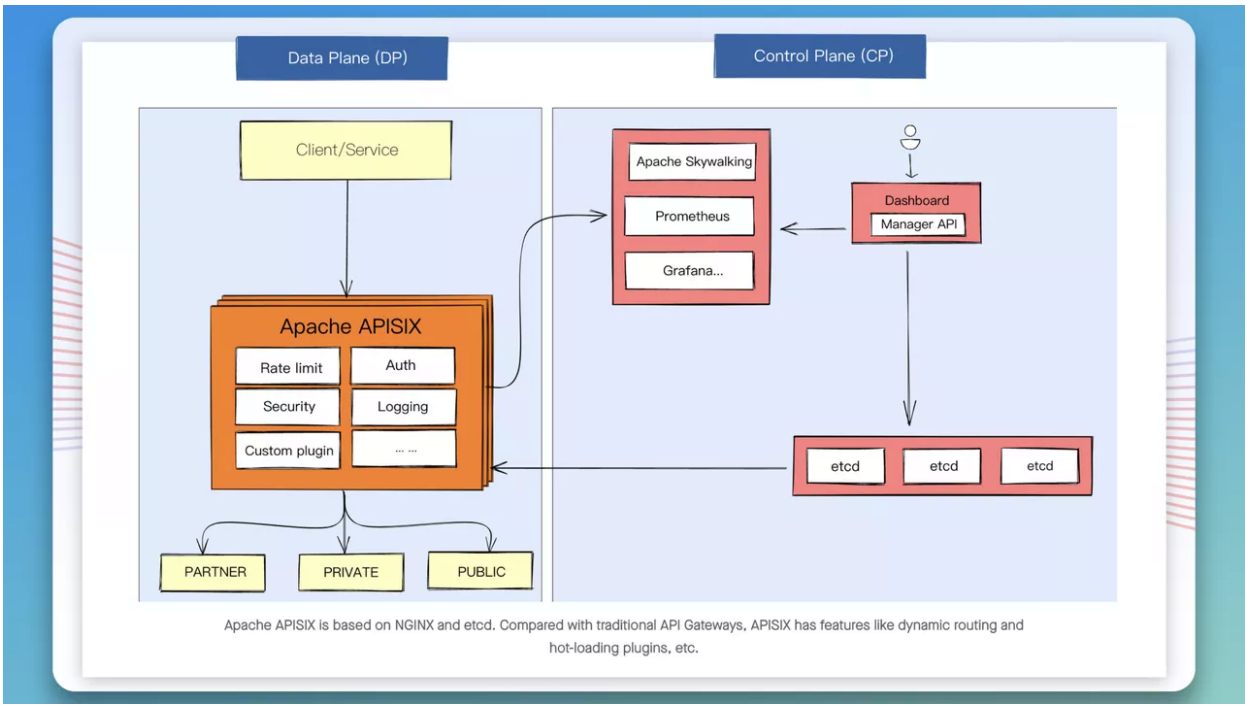
APISIX 的架构主要分成两部分：

第一部分叫做数据面，它是真正去处理来自客户端请求的一个组件，去处理用户的真实流量，包括像身份验证、证书卸载、日志分析和可观测性等功能。数据面本身并不会存储任何数据，所以它是一个无状态结构。

第二部分叫做控制面。APISIX 在底层架构上和其它 API 网关的一个很大不同就在于控制面。APISIX 在控制面上并没有使用传统的类似于像 MySQL 去做配置存储，而是选择使用 etcd。这样做的好处主要有以下几点：

- 与产品架构的云原生技术体系更统一
- 更贴合 API 网关存放的数据类型
- 能更好地体现高可用特性
- 拥有低于毫秒级别的变化通知

使用 etcd 后，对于数据面而言只需监听 etcd 的变化即可。如果轮询数据库的话，可能需要 5-10 秒才能获取取到最新的配置；但如果监听 etcd 的配置变更，就可以将时间控制在毫秒级别之内，达到实时生效的效果。



APISIX 的主要概念和组件

概念/组件	描述
Route	通过路由定义规则来匹配客户端请求，根据匹配结果加载并执行相应的插件，最后把请求转发给到指定的上游应用。
Upstream	上游的作用是按照配置规则对服务节点进行负载均衡，它的地址信息可以直接配置到路由或服务上。
Admin API	用户可以通过 Admin API 控制 APISIX 实例。

1.3 Apache APISIX 的技术优势

NGINX 与 Kong 的痛点

在单体服务时代，使用 NGINX 可以应对大多数的场景，而到了云原生时代，NGINX 因为其自身架构的原因则会出现两个问题：

- 首先是 NGINX 不支持集群管理。几乎每家互联网厂商都有自己的 NGINX 配置管理系统，系统虽然大同小异但是一直没有统一的方案。
- 其次是 NGINX 不支持配置的热加载。很多公司一旦修改了配置，重新加载 NGINX 的时间可能需要半个小时以上。并且在 Kubernetes 体系下，上游会经常发生变化，如果使用 NGINX 来处理就需要频繁重启服务，这对于企业是不可接受的。

而 Kong 的出现则解决了 NGINX 的痛点，但是又带来了新的问题：

- Kong 需要依赖于 PostgreSQL 或 Cassandra 数据库，这使 Kong 的整个架构非常臃肿，并且会给企业带来高可用的问题。如果数据库故障了，那么整个 API 网关都会出现故障。
- Kong 的路由使用的是遍历查找，当网关内有超过上千个路由时，它的性能就会出现比较急剧的下降。

Apache APISIX 和 Kong 相比在技术方面的主要优势，大部分都是在底层模块上的优化和创新。

无数据库依赖

APISIX 在设计之初，就从底层架构上避免了宕机、丢失数据等情况的发生。因为在控制面上，APISIX 使用了 etcd 存储配置信息，而不是使用关系型数据库，这样做的好处主要有以下几点：

- 与产品架构的云原生技术体系更统一；
- 更贴合 API 网关存放的数据类型；
- 能更好地体现高可用特性；
- 拥有低于毫秒级别的变化通知。

使用 etcd 存储配置信息后，对于数据面而言只需监听 etcd 的变化即可。如果采用轮询数据库的方式，可能需要 5-10 秒才能获取到最新的配置信息；如果监听 etcd 的配置信息变更，APISIX 就可以将获取最新配置的时间控制在毫秒级别之内，达到实时生效。

插件热加载

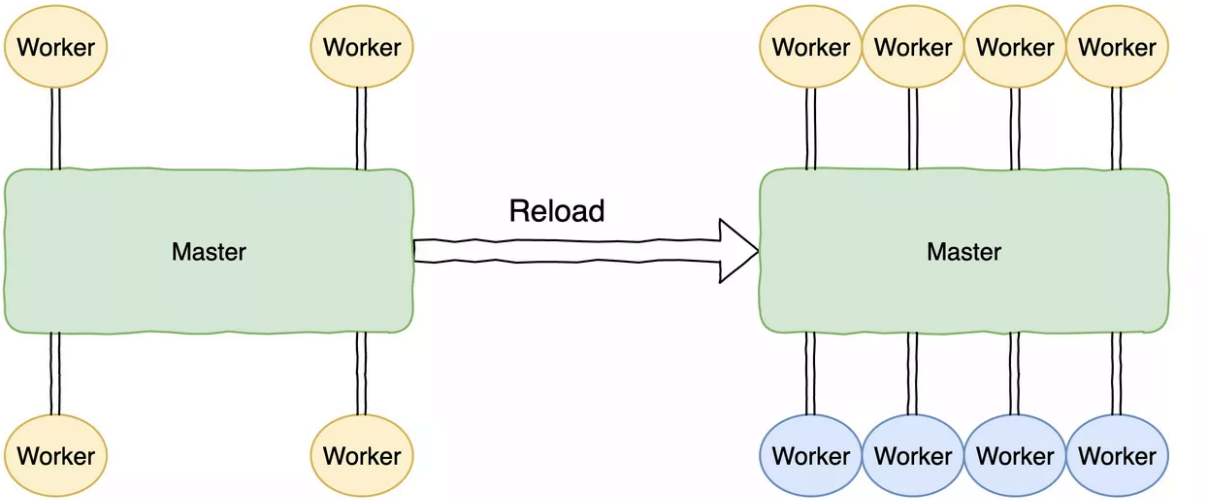
APISIX 和 NGINX 相比，有两处非常大的变化：APISIX 支持集群管理和动态加载。

思考： APISIX是如何实现热加载的

NGINX 热加载的原理

执行 nginx -s reload 热加载命令，就等同于向 NGINX 的 master 进程发送 HUP 信号。在 master 进程收到 HUP 信号后，会依次打开新的监听端口，然后启动新的 worker 进程。此时会存在新旧两套 worker 进程，在新的 worker 进程起来后，master 会向老的 worker 进程发送 QUIT 信号进行优雅关闭。老的

worker 进程收到 QUIT 信号后，会首先关闭监听句柄，此时新的连接就只会流进到新的 worker 进程中，老的 worker 进程处理完当前连接后就会结束进程。



NGINX 热加载的缺陷

- 首先，NGINX 频繁热加载会造成连接不稳定，增加丢失业务的可能性。

NGINX 在执行 reload 指令时，会在旧的 worker 进程上处理已经存在的连接，处理完连接上的当前请求后，会主动断开连接。此时如果客户端没处理好，就可能会丢失业务，这对于客户端来说明显就不是无感知的了。

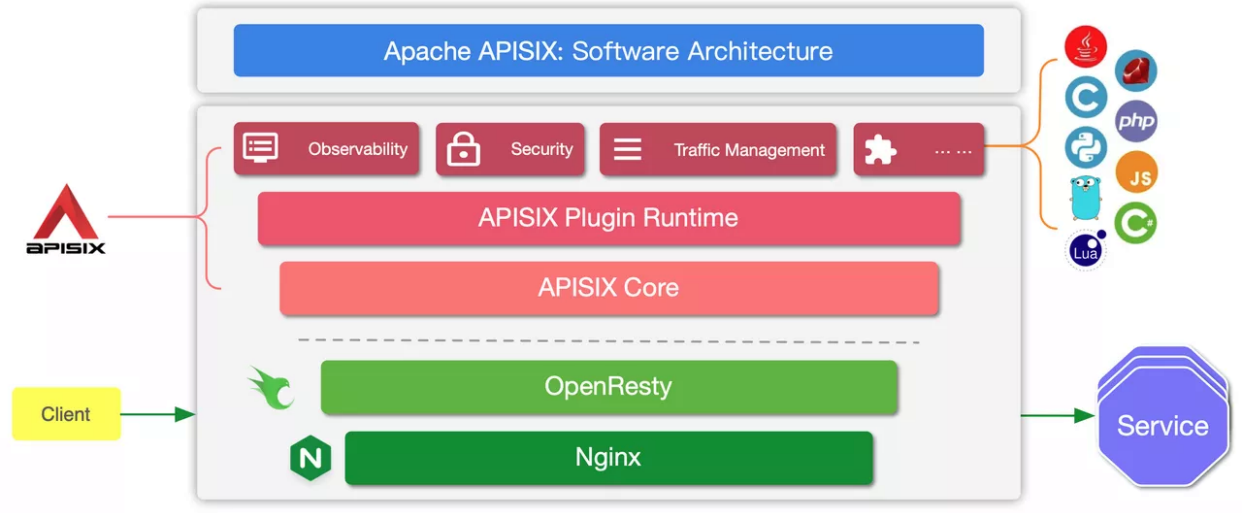
- 其次，在某些场景下，旧进程回收时间长，进而影响正常业务。

比如代理 WebSocket 协议时，由于 NGINX 不解析通讯帧，所以无法知道该请求是否为已处理完毕状态。即使 worker 进程收到来自 master 的退出指令，它也无法立刻退出，而是需要等到这些连接出现异常、超时或者某一端主动断开后，才能正常退出。再比如 NGINX 做 TCP 层和 UDP 层的反向代理时，它也没法知道一个请求究竟要经过多少次请求才算真正地结束。

这就导致旧 worker 进程的回收时间特别长，尤其是在直播、新闻媒体活语音识别等行业。旧 worker 进程的回收时间通常能达到半小时甚至更长，这时如果再频繁 reload，将会导致 shutting down 进程持续增加，最终甚至会导致 NGINX OOM，严重影响业务。

APISIX 在内存中直接生效的热加载方案

在 Apache APISIX 诞生之初，就是希望来解决 NGINX 热加载这个问题的。



通过上述架构图可以看到，之所以 APISIX 能摆脱 NGINX 的限制是因为它把上游等配置全部放到 APISIX Core 和 Plugin Runtime 中动态指定。

以路由为例，NGINX 需要在配置文件内进行配置，每次更改都需要 reload 之后才能生效。而为了实现路由动态配置，Apache APISIX 在 NGINX 配置文件内配置了单个 server，这个 server 中只有一个 location。我们把这个 location 作为主入口，所有的请求都会经过这个 location，再由 APISIX Core 动态指定具体上游。因此 Apache APISIX 的路由模块支持在运行时增减、修改和删除路由，实现了动态加载。所有的这些变化，对客户端都零感知，没有任何影响。

比如增加某个新域名的反向代理，在 APISIX 中只需创建上游，并添加新的路由即可，整个过程中不需要 NGINX 进程有任何重启。再比如插件系统，APISIX 可以通过 `ip-restriction` 插件实现 IP 黑白名单功能，这些能力的更新也是动态方式，同样不需要重启服务。借助架构内的 etcd，配置策略以增量方式实时推送，最终让所有规则实时、动态的生效，为用户带来极致体验。

高性能路由匹配算法

API 网关需要从每个请求的 Host、URI、HTTP 方法等特征中匹配到目标规则，以决定如何对该请求进行处理，因此一个优秀的匹配算法是必不可少的。

Apache APISIX 使用的是 RadixTree，它提供了 KV 存储查找的数据结构并对只有一个子节点的中间节点进行了压缩，因此它又被称为压缩前缀树。此外，在已知 API 网关产品中 Apache APISIX 首次将 RadixTree 应用到了路由匹配中，支持一个前缀下有多个不同路由的场景。

当对某个请求进行匹配时，RadixTree 将采用层层递进的方式进行匹配，其复杂度为 O(K)（K 是路由中 URI 的长度，与 API 数量多少无关），该算法非常适合公有云、CDN 以及路由数量比较多的场景，可以很好地满足路由数量快速增长的需求。

高性能 IP 匹配算法

假设现在有一个包含 500 条 IPv4 记录的 IP 库，APISIX 会将 500 条 IPv4 的记录缓存在 Hash 表中，当进行 IP 匹配时使用 Hash 的方式进行查找，时间复杂度为 O(1)。

而其他 API 网关则是通过遍历的方式完成 IP 匹配，发送到网关每个请求将逐个遍历最多 500 次是否相等后才能知道计算结果。所以 APISIX 的高精度 IP 匹配算法大大提高了需要进行海量 IP 黑白名单匹配场景的效率。

精细化路由

API 网关通过请求中的流量特征完成预设规则的匹配，常见特征包含了请求中的 Host、URI 路径、URI 查询参数、URI 路径参数、HTTP 请求方法、请求头等，这些特征是大部分 API 网关产品所支持的。相较于其它产品，Apache APISIX 支持了更多特征以解决复杂多变的使用场景。

首先，Apache APISIX 支持 NGINX 内置变量，意味着我们可以将诸如 `uri`、`server_name`、`server_addr`、`request_uri`、`remote_port`、`remote_addr`、`query_string`、`host`、`hostname`、`hostname`、`arg_name` 等数十种 NGINX 内置变量作为匹配参数，以支持更复杂多变的匹配场景。

其次，Apache APISIX 支持将条件表达式作为匹配规则，其结构是 `[var, operator, val], ...]]`，其中：

- `var` 值可使用 NGINX 内置变量；
- `operator` 支持相等、不等、大于、小于、正则、包含等操作符。

假设表达式为 `["arg_name", "==", "fox"]`，它意味着当前请求的 URI 查询参数中，是否有一个为 `name` 的参数值等于 `fox`。

此外，Apache APISIX 支持设置路由 ttl 存活时间：

```
curl http://127.0.0.1:9080/apisix/admin/routes/2?ttl=60 \
-H 'X-API-KEY: edd1c9f034335f136f87ad84b625c8f1' -X PUT -i -d '
{
 "uri": "/aa/index.html",
 "upstream": {
 "type": "roundrobin",
 "nodes": {
 "39.97.63.215:80": 1
 }
  }
}'
```

以上配置表示，在 60s 后 APISIX 会自动删除该路由配置，非常适合一些临时验证的场景，比如金丝雀发布、监控输出等。对于线上的流量分流非常方便，是其它网关产品所不具备的能力。

最后一点是 Apache APISIX 支持自定义过滤函数，你可以通过在 filter_func 参数中编写自定义 Lua 函数，例如：

```
curl http://127.0.0.1:9080/apisix/admin/routes/1 \
-H 'X-API-KEY: edd1c9f034335f136f87ad84b625c8f1' -X PUT -i -d '
{
 "uri": "/index.html",
 "hosts": ["foo.com", "*.bar.com"],
 "filter_func": "function(vars)
 return vars['host'] == 'api7.ai'
 end",
 "upstream": {
  "type": "roundrobin",
  "nodes": {
  "127.0.0.1:1980": 1
  }
  }
}'
```

其中 filter_func 入参是 vars，可从 vars 获取 NGINX 变量，然后实现自定义过滤逻辑。

支持多语言插件

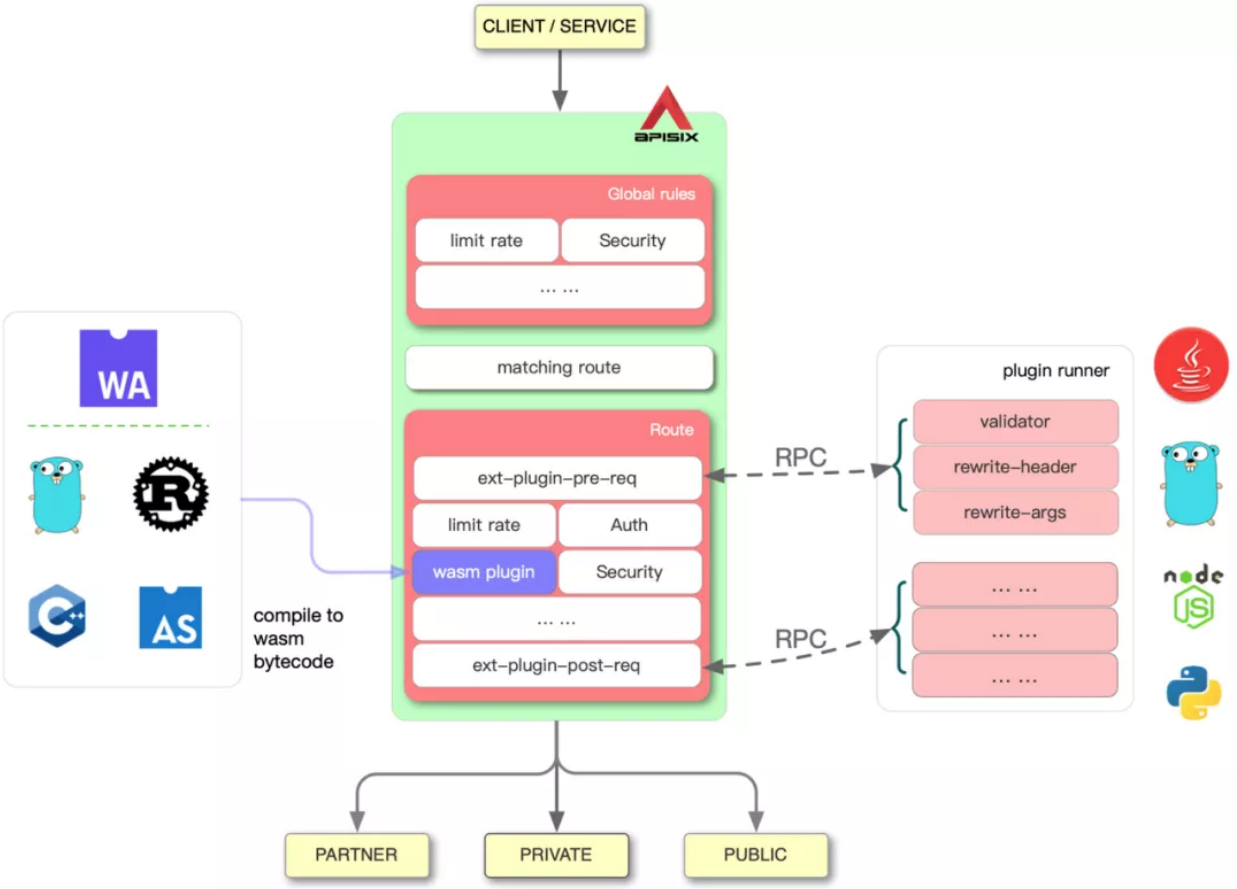
APISIX 目前已经支持了 80 多种插件，但仍然难以涵盖用户的所有使用场景。在实际使用场景中，很多企业都会针对具体业务进行定制化的插件开发，通过网关去集成更多的协议或者系统，最终在网关层实现统一管理。

在 APISIX 早期版本中，开发者仅能使用 Lua 语言开发插件。虽然通过原生计算语言开发的插件具备非常高的性能，但是学习 Lua 这门新的开发语言是需要时间和理解成本的。

针对这种情况，APISIX 提供了两种方式来解决：

第一种方式是通过 Plugin Runner 来支持更多的主流编程语言（比如 Java、Python、Go 等等）。通过这样的方式，可以让后端工程师通过本地 RPC 通信，使用熟悉的编程语言开发 APISIX 的插件。

这样做的好处是减少了开发成本，提高了开发效率，但是在性能上会有一些损失。那么，有没有一种既能达到 Lua 原生性能，同时又兼顾高级编程语言的开发效率方案呢？



第二种方式是使用 Wasm 开发插件，也就是上图左侧部分。

Wasm（WebAssembly）最早是用在前端和浏览器上的一个技术，但是现在在服务端它也逐渐展示出来它的优势。我们将 Wasm 嵌入到了 APISIX 中，用户

就可以使用 Wasm 去编译成 Wasm 的字节码在 APISIX 中运行。最终达到的效果就是利用高效率，开发出了一个既有高性能又使用高级计算语言编写的 APISIX 插件。

因此，在当前 APISIX 版本中，用户可以使用 Lua、Go、Python 和 Wasm 等多种方式，基于 APISIX 编写自定义插件。通过这样的方式，降低了开发者的使用门槛，也为 APISIX 的功能提供了更多的可能性。

1.4 APISIX 的应用场景

APISIX 的核心是高性能代理服务，自身不绑定任何环境属性。当它演变为 Ingress、服务网格等产品时，都是外部服务与 APISIX 配合，变化的是外部程序而不是 APISIX 自身，下面将逐步为大家介绍 APISIX 是如何支持这些场景的。

Load Balancer 和 API 网关

首先是针对传统的 LB 和 API 网关场景，因为 APISIX 基于 NGINX + LuaJIT 实现，所以天然具备高性能、安全等特性，并且原生支持了动态 SSL 证书卸载、SSL 握手优化等功能，在负载均衡的服务能力上也更优秀。从 NGINX 切换到 APISIX 不仅性能不会下降，而且可以享受到动态、统一管理等特性带来的管理效率的提升。

微服务网关

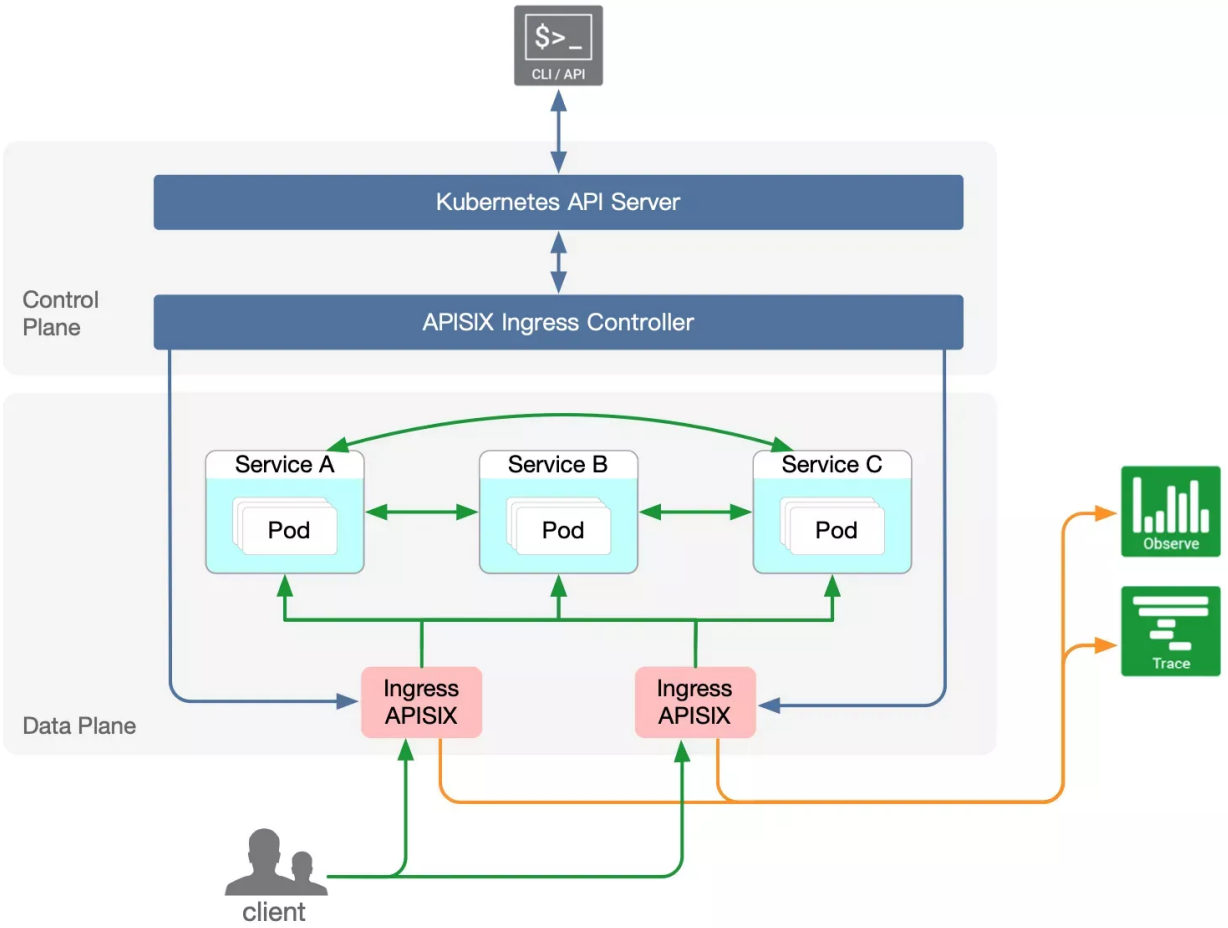
APISIX 目前支持多种语言编写扩展插件，可以解决东西向微服务 API 网关面临的主要问题——异构多语言和通用问题。内置支持的服务注册中心有 Nacos、

etcd、Eureka 等，还有标准的 DNS 方式，可以平滑替代 Zuul、Spring Cloud Gateway、Dubbo 等微服务 API 网关。

Kubernetes Ingress

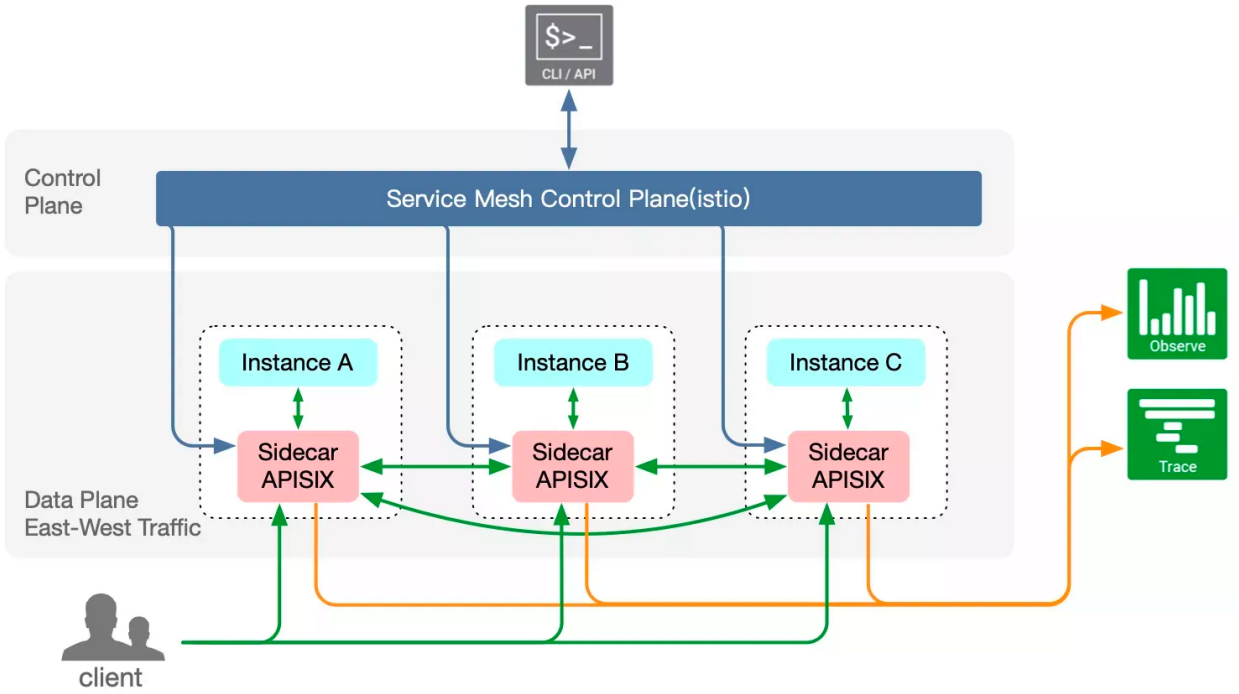
目前 K8s 官方 Kubernetes Ingress Controller 项目主要基于 NGINX 配置文件的方式，所以在路由能力和加载模式上稍显不足，并且存在一些明显劣势。比如添加、修改任何 API 时，需要重启服务才能完成新 NGINX 配置的更新，但重启服务，对线上流量的影响是非常大的。

而 APISIX Ingress Controller 则完美解决了上面提到的所有问题：支持全动态，无需重启加载。同时继承了 APISIX 的所有优势，还支持原生 Kubernetes CRD，方便用户迁移。



服务网格

未来五到十年，基于云原生模式架构下的服务网格架构开始崭露头角。APISIX 也提前开始锁定赛道，通过调研和技术分析后，APISIX 已经支持了 xDS 协议，APISIX Mesh 就此诞生，在服务网格领域 APISIX 也拥有了一席之地。



2. APISIX快速开始

2.1 centos7/centos8安装APISIX

安装etcd

APISIX 使用 etcd 作为配置中心进行保存和同步配置。在安装 APISIX 之前，需要在你的主机上安装 etcd。

```
ETCD_VERSION='3.5.4'
wget https://github.com/etcd-io/etcd/releases/download/v${ETCD_VERSION}/etcd-v${ETCD_VERSION}-linux-amd64.tar.gz
tar -xvf etcd-v${ETCD_VERSION}-linux-amd64.tar.gz && \
cd etcd-v${ETCD_VERSION}-linux-amd64 && \
sudo cp -a etcd etcdctl /usr/bin/
nohup etcd >/tmp/etcd.log 2>&1 &
```

安装APISIX

通过 RPM 仓库安装

如果当前系统没有安装 OpenResty，请使用以下命令来安装 OpenResty 和 APISIX 仓库：

```
sudo yum install -y https://repos.apiseven.com/packages/centos/apache-apisix-repo-1.0-1.noarch.rpm
```

如果已安装 OpenResty 的官方 RPM 仓库，请使用以下命令安装 APISIX 的 RPM 仓库：

```
sudo yum-config-manager --add-repo https://repos.apiseven.com/packages/centos/apache-apisix.repo
```

完成上述操作后使用以下命令安装 APISIX：

```
sudo yum install apisix
# 安装指定版本的 APISIX
sudo yum install apisix-3.0.0
```

通过 RPM 包离线安装

将 APISIX 离线 RPM 包下载到 `apisix` 文件夹

```
sudo mkdir -p apisix
sudo yum install -y https://repos.apiseven.com/packages/centos/apache-apisix-repo-1.0-1.noarch.rpm
sudo yum clean all && yum makecache
sudo yum install -y --downloadonly --downloaddir=./apisix apisix
```

然后将 `apisix` 文件夹复制到目标主机并运行以下命令：

```
sudo yum install ./apisix/*.rpm
```

管理 APISIX 服务

APISIX 安装完成后，你可以运行以下命令初始化 NGINX 配置文件和 etcd：

```
apisix init
```

运行以下命令测试配置文件，APISIX 将根据 `config.yaml` 生成 `nginx.conf`，并检查 `nginx.conf` 的语法是否正确。

```
apisix test
```

使用以下命令启动 APISIX：

```
apisix start
```

> 你可以运行 apisix help 命令，通过查看返回结果，获取其他操作的命令及描述。

如果需要停止 APISIX，你可以使用 `apisix quit` 或者 `apisix stop` 命令。`apisix quit` 将正常关闭 APISIX，该指令确保在停止之前完成所有收到的请求。

```
apisix quit
```

`apisix stop` 命令会强制关闭 APISIX 并丢弃所有请求。

```
apisix stop
```

配置 APISIX

通过修改本地 /usr/local/apisix/conf/config.yaml 文件，或者在启动 APISIX 时使用 -c 或 --config 添加文件路径参数 apisix start -c <path string>，完成对 APISIX 服务本身的基本配置。

- 比如将 APISIX 默认监听端口修改为 8000，其他配置保持默认，在 /usr/local/apisix/conf/config.yaml 中只需这样配置：

```yaml
apisix:
  node_listen: 8000 # APISIX listening port
```

- 比如指定 APISIX 默认监听端口为 8000，并且设置 etcd 地址为 http://192.168.65.200:2379，其他配置保持默认。

在 ./conf/config.yaml 中只需这样配置：

```yaml
apisix:
  node_listen: 8000 # APISIX listening port

deployment:
  role: traditional
  role_traditional:
  config_provider: etcd
  etcd:
  host:
   - "http://192.168.65.200:2379"
```

注意：

- APISIX 的默认配置可以在 ./conf/config-default.yaml 文件中看到，该文件与 APISIX 源码强绑定，请不要手动修改 ./conf/config-default.yaml 文件。如果需要自定义任何配置，都应在 ./conf/config.yaml 文件中完成。
- 请不要手动修改 APISIX 安装目录下的 ./conf/nginx.conf 文件。当 APISIX 启动时，会根据 config.yaml 的配置自动生成新的 nginx.conf 并自动启动服务。

更新 Admin API key

建议修改 Admin API 的 key，保护 APISIX 的安全。

```yaml
deployment:
  admin:
  admin_key
```

```
  -
  name: "admin"
  key: newsupersecurekey # 请修改 key 的值
  role: admin
```

更新完成后，你可以使用新的 key 访问 Admin API：

```
curl http://127.0.0.1:9180/apisix/admin/routes?api_key=newsupersecurekey -i
```

```
[root@redis ~]# curl http://127.0.0.1:9180/apisix/admin/routes?api_key=edd1c9f034335f136f87ad84b625c8f1 -i
HTTP/1.1 200 OK
Date: Tue, 06 Dec 2022 12:05:01 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: keep-alive
Server: APISIX/3.0.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: *
Access-Control-Max-Age: 3600
X-API-VERSION: v3

{"list":[],"total":0}
```

2.2 Docker安装APISIX

环境准备

- 安装Docker 和 Docker Compose。

```
curl -L https://github.com/docker/compose/releases/download/2.14.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

使用 Docker 安装 APISIX 并启用 Admin API。

1) 通过 `git` 命令克隆 apisix-docker 仓库：

```
git clone git@github.com:apache/apisix-docker.git
cd apisix-docker/example
```

2) 通过 `docker-compose` 启动 APISIX

```
docker-compose -p docker-apisix up -d
```

- 请确保其他系统进程没有占用 9080、9180、9443 和 2379 端口。
- 如果 Docker 容器不能正常运行，你可以通过以下命令检查日志进行问题诊断：

```
docker logs -f --tail 100 $<container_id>
```

3）安装完成后，你可以在运行 Docker 的宿主机上执行 `curl` 命令访问 Admin API，根据返回数据判断 APISIX 是否成功启动。

```
# 注意：请在运行 Docker 的宿主机上执行 curl 命令。
```

```
curl "http://127.0.0.1:9180/apisix/admin/services/" -H 'X-API-KEY: edd1c9f034335f136f87ad84b625c8f1'
```

如果返回数据如下所示，则表示 APISIX 成功启动：

```
{
 "count":0,
 "node":{
 "key":"/apisix/services",
 "nodes":[],
 "dir":true
 }
}
```

2.3 创建路由

APISIX 提供了强大的 Admin API 和 Dashboard 供用户使用。在下述示例中，我们将使用 Admin API 创建一个 Route 并与 Upstream 绑定，当一个请求到达 APISIX 时，APISIX 会将请求转发到指定的上游服务中。

以下示例代码中，我们将为路由配置匹配规则，以便 APISIX 可以将请求转发到对应的上游服务：

```
curl "http://127.0.0.1:9180/apisix/admin/routes/1" -H "X-API-KEY: edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
 "methods": ["GET"],
 "host": "example.com",
 "uri": "/anything/*",
 "upstream": {
 "type": "roundrobin",
 "nodes": {
 "httpbin.org:80": 1
  }
  }
}'

```

该配置意味着，当请求满足下述的所有规则时，请求将被转发到上游服务 (httpbin.org:80)：

- 请求的 HTTP 方法为 GET。

- 请求头包含 host 字段，且它的值为 example.com。
- 请求路径匹配 /anything/*，* 意味着任意的子路径，例如 /anything/foo?arg=10。

当路由创建完成后，可以通过以下命令访问上游服务：

```
curl -i -X GET "http://127.0.0.1:9080/anything/foo?arg=10" -H "Host: example.com"
```

该请求将被 APISIX 转发到 http://httpbin.org/anything/foo?arg=10

```
[root@redis ~]# curl -i -X GET "http://127.0.0.1:9080/anything/foo?arg=10" -H "Host: example.com"
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 422
Connection: keep-alive
Date: Tue, 06 Dec 2022 12:20:01 GMT
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Server: APISIX/3.0.0

{
  "args": {
    "arg": "10"
  },
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Host": "example.com",
    "User-Agent": "curl/7.29.0",
    "X-Amzn-Trace-Id": "Root=1-638f3371-5705d5b921080dde4d58af5a",
    "X-Forwarded-Host": "example.com"
  },
  "json": null,
  "method": "GET",
  "origin": "127.0.0.1, 175.0.201.99",
  "url": "http://example.com/anything/foo?arg=10"
}
```

2.4 使用上游服务创建路由

你可以通过以下命令创建一个上游，并在路由中使用它，而不是直接将其配置在路由中：

```
curl "http://127.0.0.1:9180/apisix/admin/upstreams/1" -H "X-API-KEY: edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
  "type": "roundrobin",
  "nodes": {
  "httpbin.org:80": 1
  }
}'
```

该上游配置与上一节配置在路由中的上游相同。同样使用了 roundrobin 作为负载均衡机制，并设置了 httpbin.org:80 为上游服务。为了将该上游绑定到路由，此处需要把 upstream_id 设置为 "1"。

上游服务创建完成后，可以通过以下命令绑定到指定路由：

```
curl "http://127.0.0.1:9180/apisix/admin/routes/1" -H "X-API-KEY: edd1c9f034335f136f87ad84b625c8f1" -X PUT -d '
{
  "uri": "/get",
  "host": "httpbin.org",
  "upstream_id": "1"
}'
```

我们已经创建了路由与上游服务，现在可以通过以下命令访问上游服务：

```
curl -i -X GET "http://127.0.0.1:9080/get?foo1=bar1&foo2=bar2" -H "Host: httpbin.org"
```

该请求将被 APISIX 转发到http://httpbin.org/anything/foo?arg=10

2.5 APISIX Dashboard使用

Apache APISIX Dashboard 是基于浏览器的可视化平台，用于监控、管理 Apache APISIX。Apache APISIX Dashboard 可以让用户尽可能直观、便捷地通过可视化界面操作 Apache APISIX。

通过 Dashboard，我们能够：

- 创建并管理 服务、上游、路由、应用、证书、全局插件等模块；
- 配合内置的80多种插件，可精细化控制流量。
- 支持配置自定义插件；
- 围绕 API 全生命周期管理提供解决方案，通过持续地更新、升级，更好地为用户赋能；
- 兼容 OpenAPI 3.0，支持路由的导入、导出。

安装Apache APISIX Dashboard

基于centos7安装

1）安装

```
# 1. install RPM package
sudo yum install -y https://github.com/apache/apisix-dashboard/releases/download/v2.14.0/apisix-dashboard-2.14.0-0.el7.x86_64.rpm
```

2)启动

```
# run dashboard in the shell
sudo manager-api -p /usr/local/apisix/dashboard/

# or run dashboard as a service
systemctl start apisix-dashboard
```

基于docker安装

1）拉取镜像

```
docker pull apache/apisix-dashboard
```

2）创建/usr/local/apisix-dashboard/conf/conf.yaml配置文件

```
conf:
  listen:
  host: 0.0.0.0 # the address on which the `Manager API` should listen.
  # The default value is 0.0.0.0, if want to specify, please enable it.
  # This value accepts IPv4, IPv6, and hostname.
  port: 9000 # The port on which the `Manager API` should listen.

  allow_list: # If we don't set any IP list, then any IP access is allowed by default.
  etcd:
```

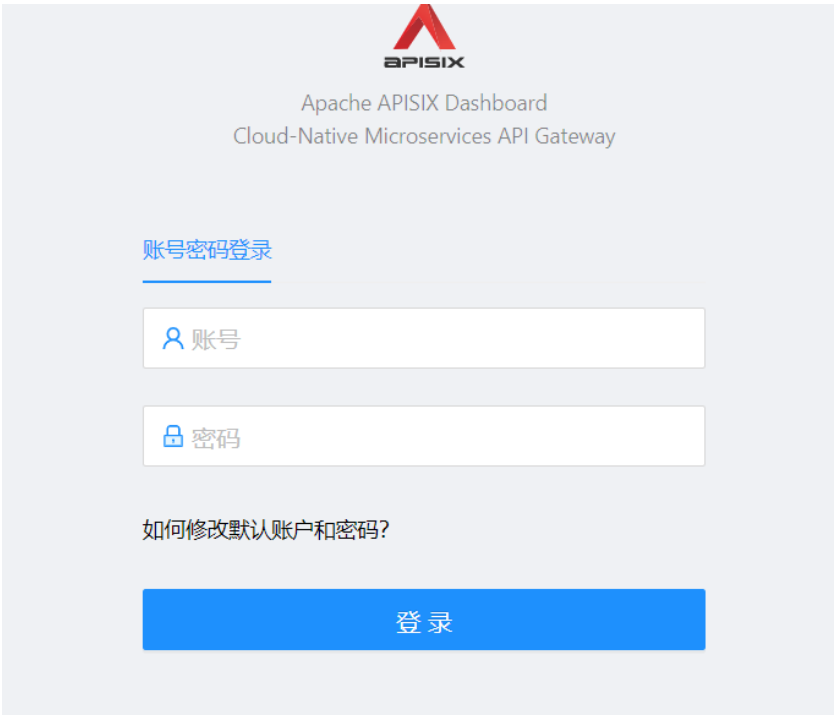
```
  endpoints: # supports defining multiple etcd host addresses for an etcd cluster
    - 192.168.3.100:2379
    #username: "root" #如果没开启授权，可以注掉
    #password: "123456" #如果没开启授权，可以注掉
  authentication:
    secret:
    zQ5w5jkLDh3jZpywJ3sskrw6Yv633ruq
    expire_time: 3600 # jwt token expire time, in second
    users: # yamllint enable rule:comments-indentation
    - username: admin
    password: admin
    - username: fox
    password: 123456

```

> 注意：可能出现连不上etcd的情况，启动etcd服务需要配置监听指定ip:port的客户端请求
> nohup etcd --listen-client-urls http://127.0.0.1:2379,http://192.168.3.100:2379 --advertise-client-urls http://192.168.3.100:2379 >/tmp/etcd.log 2>&1 &

3）启动dashboard

```
docker run -d --name dashboard \
  -p 9000:9000 \
  -v /usr/local/apisix-dashboard/conf/conf.yaml:/usr/local/apisix-dashboard/conf/conf.yaml \
  apache/apisix-dashboard
```

访问http://192.168.3.100:9000/，登录用户密码admin/admin



3. Apache APISIX实战

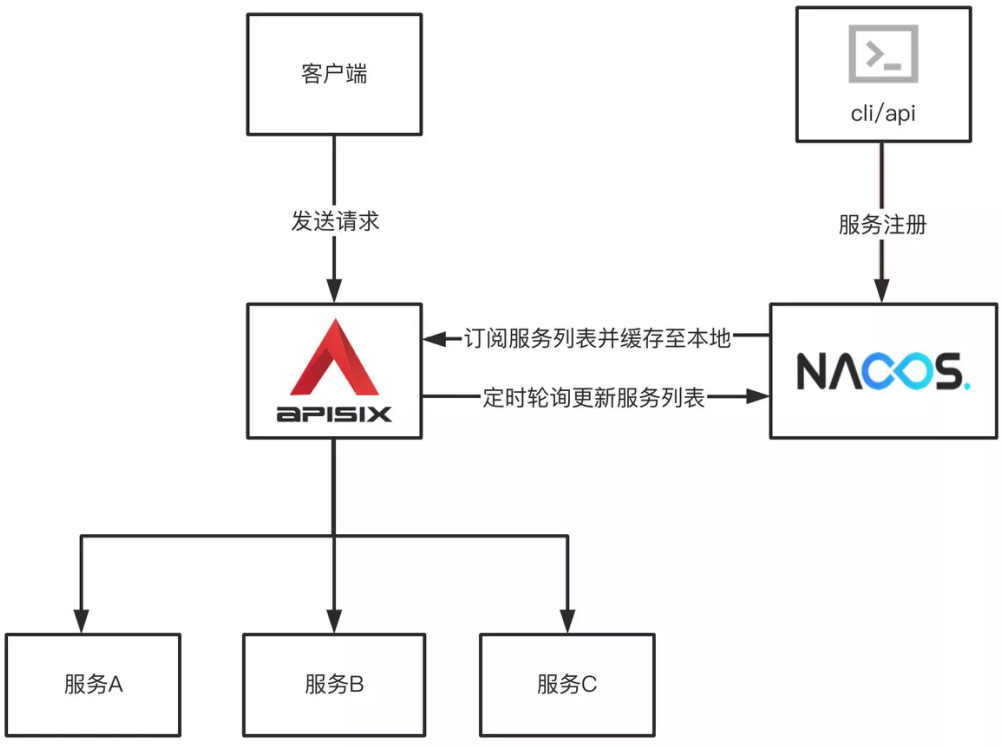
3.1 Apache APISIX基于Nacos实现服务发现

注册中心本质上是为了解耦服务提供者和服务消费者，在微服务体系中，各个业务服务之间会频繁互相调用，并且需要对各个服务的 IP、port 等路由信息进行统一的管理。

注册中心的核心功能为以下三点：

- 服务注册：服务提供方向注册中心进行注册。
- 服务发现：服务消费方可以通过注册中心寻找到服务提供方的调用路由信息。
- 健康检测：确保注册到注册中心的服务节点是可以被正常调用的，避免无效节点导致的调用资源浪费等问题。

Apache APISIX + Nacos 可以将各个微服务节点中与业务无关的各项控制，集中在 Apache APISIX 中进行统一管理，即通过 Apache APISIX 实现接口服务的代理和路由转发的能力。在 Nacos 上注册各个微服务后，Apache APISIX 可以通过 Nacos 的服务发现功能获取服务列表，查找对应的服务地址从而实现动态代理。



1) 配置nacos注册中心

在apisix的配置文件 conf/config.yaml 中添加以下配置：

```
discovery:
 nacos:
 host:
 - "http://${username}:${password}@${host1}:${port1}"
```

```
  prefix: "/nacos/v1/"
  fetch_interval: 30 # default 30 sec
  weight: 100 # default 100
  timeout:
  connect: 2000 # default 2000 ms
   send: 2000 # default 2000 ms
   read: 5000 # default 5000 ms
```

也可以这样简洁配置（未配置项使用默认值）：

```
discovery:
 nacos:
 host:
 - "http://192.168.3.100:8848"
```

2）Upstream 设置

使用 Apache APISIX 提供的 Admin API 创建一个新的路由（Route），APISIX 通过 upstream.discovery_type 字段选择使用的服务发现类型，upstream.service_name 需要与注册中心的对应服务名进行关联。

例如，转发 URI 匹配 "/nacos/*" 的请求到一个上游服务， 该服务在 Nacos 中的服务名是 APISIX-NACOS， 查询地址是 http://192.168.3.100:8848/nacos/v1/ns/instance/list?serviceName=APISIX-NACOS， 创建路由时指定服务发现类型为 nacos 。

```
$ curl http://127.0.0.1:9180/apisix/admin/routes/1 -H 'X-API-KEY: edd1c9f034335f136f87ad84b625c8f1' -X PUT -i -d '
{
 "uri": "/nacos/*",
 "upstream": {
 "service_name": "APISIX-NACOS",
 "type": "roundrobin",
 "discovery_type": "nacos"
 }
}'
```

指定namespace和group

```
$ curl http://127.0.0.1:9180/apisix/admin/routes/4 -H 'X-API-KEY: edd1c9f034335f136f87ad84b625c8f1' -X PUT -i -d '
{
 "uri": "/nacosWithNamespaceIdAndGroupName/*",
 "upstream": {
 "service_name": "APISIX-NACOS",
 "type": "roundrobin",
```

```
  "discovery_type": "nacos",
  "discovery_args": {
  "namespace_id": "test_ns",
   "group_name": "test_group"
   }
   }
}'
```

也可以通过apisix dashboard进行配置

3) 验证配置结果

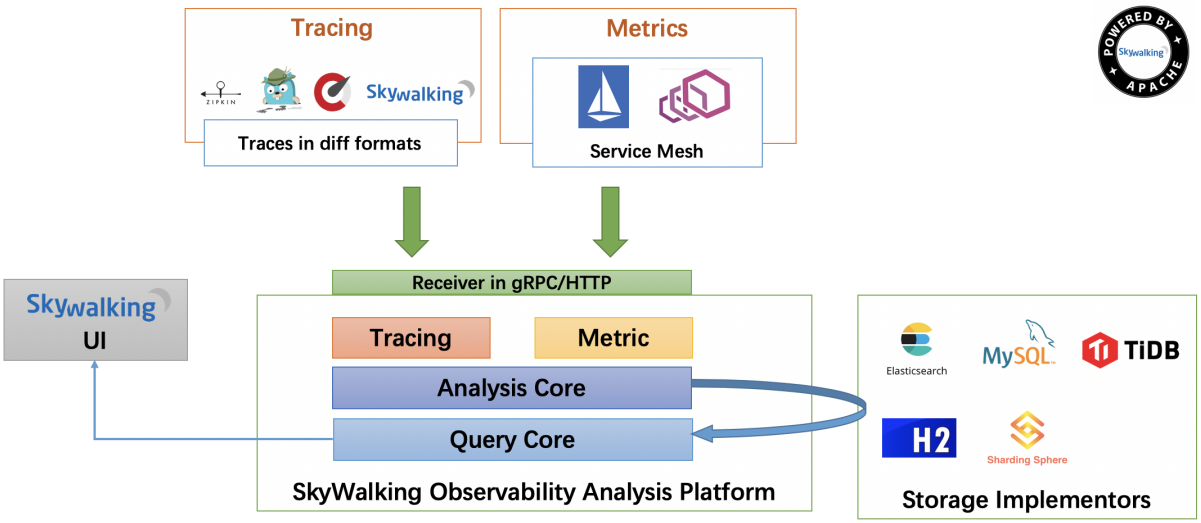
使用以下命令发送请求至需要配置的路由。

```
curl -i http://127.0.0.1:9080/order/findOrderByUserId/1
```

3.2 Apache APISIX 集成 SkyWalking实战

skywalking 插件用于与 Apache SkyWalking 集成。

skywalking环境准备



1) 搭建skywalking服务端环境

下载地址：https://skywalking.apache.org/downloads/

```
#启动oap服务和ui服务
bin/startup.sh
```

2) 微服务端接入skywalking agent

增加jvm参数

```
-javaagent:D:\apache\apache-skywalking-apm-es7-8.4.0\apache-skywalking-apm-bin-es7\agent\skywalking-agent.jar
```

```
-DSW_AGENT_NAME=mall-order
-DSW_AGENT_COLLECTOR_BACKEND_SERVICES=192.168.3.100:11800
```

启用skywalking插件

该插件默认是禁用状态，你需要将其添加到配置文件（./conf/config.yaml）中才可以启用它：

```
plugins:
  - ...
  - skywalking
```

配置完成后，重新加载 APISIX，此时 APISIX 会创建一个后台定时器，向 SkyWalking OAP 服务定期上报数据。

在指定路由中启用 skywalking 插件：

```
curl http://127.0.0.1:9180/apisix/admin/routes/1 -H 'X-API-KEY: edd1c9f034335f136f87ad84b625c8f1' -X PUT -d '
{
  "methods": ["GET"],
  "uris": [
  "/order/*"
  ],
  "plugins": {
  "skywalking": {
  "sample_ratio": 1
   }
   },
   "upstream": {
  "type": "roundrobin",
  "nodes": {
  "192.168.65.103:8020": 1,
  "192.168.65.103:8021": 1
  }
  }
}'
```

- sample_ratio： 采样的比例，默认1。设置为 1 时，将对所有请求进行采样。

设置 Endpoint

在配置文件（./conf/config.yaml）中配置以下属性：

名称	类型	默认值	描述

service_name	string	"APISIX"	SkyWalking 上报的服务名称。
service_instance_name	string	"APISIX Instance Name"	SkyWalking 上报的服务实例名。设置为 `$hostname` 时，将获取本机主机名。
endpoint_addr	string	"http://127.0.0.1:12800"	SkyWalking 的 HTTP endpoint 地址，例如：`http://127.0.0.1:12800`。
report_interval	integer	SkyWalking 客户端内置的值	上报间隔时间，单位为秒。

```
plugin_attr:
  skywalking:
  service_name: APISIX
  service_instance_name: "APISIX Instance Name"
  endpoint_addr: http://127.0.0.1:12800
```

测试

访问接口：http://192.168.3.100:9080/order/findOrderByUserId/1

访问 SkyWalking 的 UI 页面，可以看到如下服务拓扑图：