基于Pingora实现k8s的网关，代码实战（一）

前言

反向代理的网关，nginx通常是首选项，但是如果我们有一些业务需求，需要二次开发它，那简直就是噩梦，不管是用c直接改nginx的源代码，还是用lua写脚本，都各有各的问题。

不光是接入上，在安全和资源利用率上也不是很好。

很早之前我们团队就有用rust开发网关的实践，当时主要是做一些流量管理，权限管理，安全模块集成等工作。

最近看到cloudflare开源了pingora，见猎心喜，决定用pingora给k8s做一个网关。

目标

1. 实现基础反向代理功能（http1|2,grpc,websocket）
2. 通过监听ingress，自动，平滑更新路由。
3. 简单，易用的中间层设计。
4. 工程化能力（监控，流控，安全）

pingora 介绍

介绍pingora之前，先说一下他的开发团队`cloudflare`，他是全球最大的网络服务商之一，提供最优质的cdn和ddos的解决方案。

在2022年，cloudflare就开始使用pingora替代nginx。

目前，Pingora 提高性能的同时，每天处理超过 1 万亿条互联网请求，并为 Cloudflare 客户带来了许多新功能，同时只需要以前代理基础架构的三分之一的 CPU 和内存资源。

它是这样介绍自己的：

> Pingora is a Rust framework to [build fast, reliable and programmable networked systems](http://cxyroad.com/ ”https://blog.cloudflare.com/pingora-open-source”).
>
>
> Pingora is battle tested as it has been serving more than 40 million Internet requests per second for [more than a few years](http://cxyroad.com/ ”https://blog.cloudflare.com/how-we-built-pingora-the-proxy-that-connects-cloudflare-to-the-internet”).

k8s & ingress 介绍
---

k8s作为最流行的容器编排管理系统，几乎统治了所有的后端服务，相信没有人不知道。

而ingress是k8s中一个对象，它描述了网关如何对service进行反向代理。

像我们常用的nginx和istio，都会做一个ingress的control，通过监控ingress的变化，动态调整网关的路由。

架构设计

* 整体思路还是洋葱模式，一层套一层的handle
* 监控，事件，安全模块都是通过handle集成到链路中

* 监听ingress，动态调整路由结构
* 提供控制器用于和网关交互

代码实现

> 我们不将架构中的内容全部实现，先做一个MVP版本。

Ingress watcher

* 通过watch机制，监控ingress的变化，[代码传送门](http://cxyroad.com/”https://github.com/woshihaoren4/pingora_ingress/blob/main/src/pkg/ingress.rs”)

```
impl WatchIngress {
    //开始监听
    pub async fn start_watch(&self)->
anyhow::Result<Receiver<IngressEvent>> {
        // 将ingress的变化发送到channel中，这里是一个异构设计
        let (sender,receiver) = async_channel::bounded(8);

        // 创建k8s客户端
        let client = Client::try_default().await?;
        let api:Api<Ingress> = xxx
        ...
        // 创建一个watcher
        ...
        let mut watch = watcher(api, wc)
            .default_backoff().boxed();
        tokio::spawn(async move {
            while let Some(result) = watch.next().await{
            ... //循环发送配置变化
            }
        });
        Ok(receiver)
    }
}
```

Router 控制

* [代码传送门](http://cxyroad.com/”https://github.com/woshihaoren4/pingora_ingress/blob/main/src/service/http_proxy.rs”)

```
impl HttpProxyControl {
    //从channel中接受事件，使用ing_event_to_router函数处理，
    //主要功能就是根据事件更新路由
    fn
ing_event_to_router(ing:IngressEvent,acl:Acl<HashMap<String,Router>>){
        ...
        //处理ingress事件
        match ty {
            1 | 2=>{ //init | update
            ... //创建或者更新路由
            }
            3=>{ //delete
            ... //删除路由
            }
            ...
        }
        // 处理sni
        for (host,i) in sni.sni{
            ...
        }
        //通过acl指针，无锁更新路由
        acl.update(move |_|{
            map
        });
    }
}
```

关于路由的设计

* ingress目前有三种路由策略：固定，前缀，特殊。我们先支持固定和前缀。
* 固定模式，直接map处理，相对简单。
* 前缀，采用经典的压缩字典树的结构，[代码传送门](http://cxyroad.com/”https://github.com/woshihaoren4/pingora_ingress/blob/main/src/infra/url_tree.rs”)

pingora启动

* 我们这里不需要做负载均衡，只需要找到正确的service即可

```
pub fn start_pingora(){
    ...
    let mut my_server = Server::new(Some(Opt::default())).unwrap();
    my_server.bootstrap();

    let mut gateway = http_proxy_service(&my_server.configuration,hpc);
    gateway.add_tcp(format!("0.0.0.0:{}",cfg.port).as_str());

    my_server.add_service(gateway);
    my_server.run_forever(); }
```

使用
--

需要先部署一个k8s集群，

再创建一个验证的namespace：qa

然后随便创建几个http服务，并设置对应的service。我这里启动的是一个回显服务用作测试。

部署服务

需要先创建`ServiceAccount`，pod里面的服务需要这个权限才能访问k8s的资源，命令如下：

```
kubectl apply -f ./deploy/role.yaml -n qa
```

然后创建deployment启动服务 `pingora-ingress-ctl`，我这里做好了镜像，用如下命令创建：

* 程序会自动监听yaml中设置为http的端口

```
kubectl apply -f ./deploy/role.yaml -n qa
```

创建ingress，并将`/api/v1`路由到我们的回显服务，命令如下：

```
kubectl apply -f ./deploy/ingress.yaml -n qa
```

为了能够在集群外访问服务，需要暴露一个主机端口，我们为服务创建一个service，命令如下：

```
kubectl apply -f ./deploy/pingora-ingress-ctl-src.yaml -n qa
```

体验

我们发送一个正确的请求，下面可以看到回显：

```
//请求
curl --location --request GET
'http://test.com:30003/api/v1/greet/hello?content=world'

//回复
{"response": "Get [test-server]---> request=hello query=world"}
```

```

将路由中的v1改成v2，回复就会变成404 not found。

### 自定义配置

我们可以给pingora加一些自定义的配置，如下测试配置：

```
---
version: 1
threads: 2
pid_file: /tmp/load_balancer.pid
error_log: /tmp/load_balancer_err.log
upgrade_sock: /tmp/load_balancer.sock
```

通过`ConfigMap`将配置注入到k8s中。命令如下：

```
kubectl apply -f ./deploy/config_map.yaml -n qa
```

然后就是将配置挂载到pod中，我们需要修改上面deployment的yaml，只需要将注释的部分解开。

我这里也贴一下：

```
...
spec:
    ...
    spec:
      containers:
        # 在启动命中指定配置文件路径
        - args:
            - '-c'
```
```

```
            - /config/config.yaml
        - command:
            - ./pingora-ingress
          image: wdshihaoren/pingora-ingress:14294998
          ...
          #挂盘
          volumeMounts:
            - mountPath: config
              name: config
              readOnly: true
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      # 挂盘
      volumes:
        - configMap:
            defaultMode: 420
            name: pingora-ingress-ctl-cm
          name: config
```

修改完成后，重新部署一下即可

尾语

前两天在群里，看到不少水友吐槽，学完了rust没有实践项目，随即决定将它开源出来。

目前只是有了一个基本结构，小伙伴们可以将任何你觉得有价值的功能集成进来，让我们卷起来。

原文链接: https://juejin.cn/post/7363484574158815269