Apusic 金蝶天燕

生成时间：2026-05-19

用户手册

金蝶Apusic负载均衡器

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 前言

本文档为金蝶Apusic负载均衡器软件V2.0.5标准版的使用说明，为用户介绍金蝶Apusic负载均衡器产品，帮助用户快速上手。

2 面向对象

本用户手册主要面向对象为适用金蝶Apusic负载均衡器的开发人员，以及相关的管理人员和运维人员。

3 版本更新说明

本文根据实际情况进行更新，最新版本包含历史修改记录。

日期	手册版本	适用产品	更新说明
2024年12月	V2.0.2	ALB V2.0.2 敏捷版	修改ALB高可用和授权认证使用文档
2025年08月	V2.0.5	ALB V2.0.5 标准版	新版用户手册，新增管控台使用说明
2025年12月	V2.0.5	ALB V2.0.5 标准版	新增用户案例配置和ebpf使用文档

4 产品介绍

4.1 产品简介

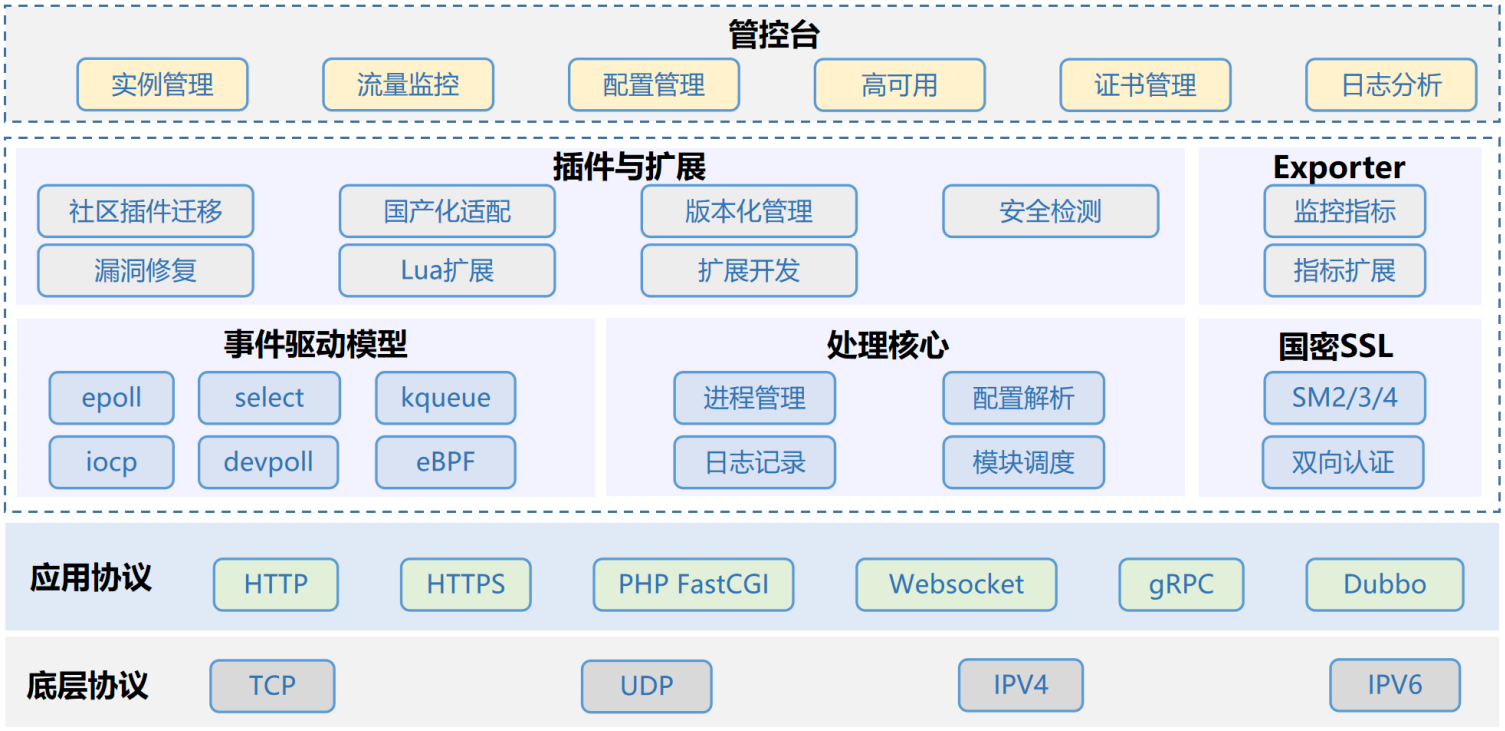
金蝶天燕负载均衡器软件标准版（Apusic Load Balancer，ALB）是一款兼具轻量级、高性能及高可用性的负载均衡器软件。ALB能有效应对大规模服务集群在面向客户端时的请求与流量管理挑战，承担起请求安全控制、验证、过滤，以及负载均衡和反向代理的重任。通过这些功能，ALB成功隔离了客户端访问对服务应用系统、平台及其资源的直接影响，实现了对集群访问流量的有效控制、精细化管理和均衡分配，确保服务稳定性和高效性。

4.2 核心功能

功能	功能说明
静态文件服务	支持静态文件缓存、断点续传等功能，实现快速、低延迟的静态内容分发。
反向代理	支持HTTP、HTTPS、TCP、UDP等多种协议，实现反向代理功能。
邮件代理	支持邮件协议，实现邮件代理功能。
负载均衡	支持四层负载均衡、七层负载均衡，实现负载均衡功能。
安全	支持HTTPS协议，支持国密SSL双向认证通信，可保障客户端和服务端通信安全可信。
Lua扩展	支持Lua语言编写扩展插件，实现自定义流量处理逻辑。
跨平台	支持国产软硬件平台，包括：鲲鹏、龙芯、兆芯、海光等国产软硬件平台。
监控	支持监控采集和展示ALB节点的运行状态、流量、性能、配置、日志等。

4.3 产品架构与功能

ALB标准版的架构设计为高效、稳定且易于扩展，它使用多进程模型来处理网络请求，确保了高性能和高并发能力。通过模块化的设计，ALB标准版能够灵活地添加新功能，满足不同业务场景的需求。ALB标准版架构支持从简单的Web服务器到复杂的负载均衡和反向代理等多种网络服务，同时保证了操作的简便性和服务的可靠性。



4.3.1 服务代理

功能	功能说明
静态文件服务	支持静态文件缓存、断点续传等功能，实现快速、低延迟的静态内容分发。
反向代理	支持HTTP、HTTPS、FastCGI、Websocket、gRPC、TCP、UDP等多种协议代理。
邮件代理	支持邮件协议，实现邮件代理功能。
协议支持	HTTP2/HTTP3，支持IPV4和IPV6协议
高可用	支持高可用集群部署，通过VIP实现主备高可用

4.3.2 负载均衡

功能	功能说明
轮询	每个请求按时间顺序逐一分配到不同的后端服务器。
带权轮询	根据节点权重生成节点流量比例，用于后端服务器性能不均的情况。
最小连接数	选择上游服务节点连接数最少的一个节点作为转发节点。
sticky(会话粘滞)	通过维护session的客户端关联性，确保来自同一客户端的请求在会话期间始终被定向到同一后端服务器，实现 session一致性。
IP地址哈希	根据IP地址进行哈希计算获得目标节点，保证每个访客固定访问一个后端服务器，可解决session一致性问题。

4.3.3 流量管理

功能	功能描述
限制请求速度	通过限制请求速度，达到防止上游服务同时被过多的请求淹没。
限制请求次数	限制客户端单位时间内的请求次数。
限制并发	限制客户端单位时间允许的连接数。
灰度发布	灰度发布，实现对服务的灰度发布，对新功能进行测试，确保新功能正常可用。

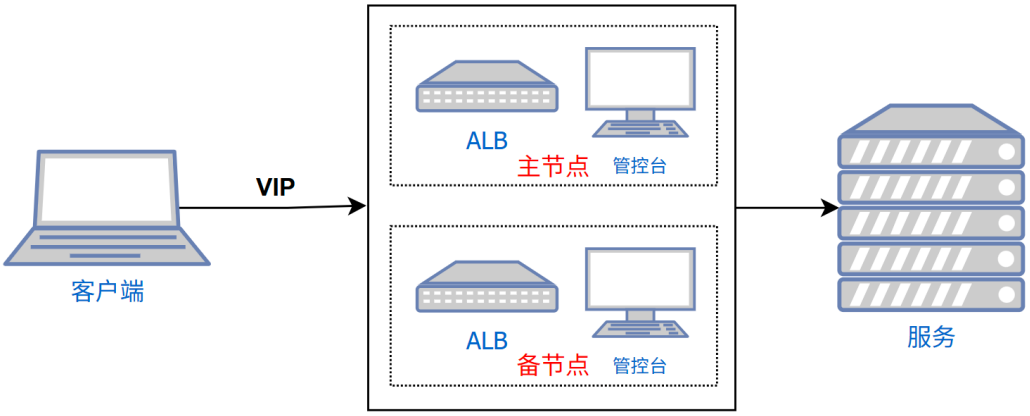
4.3.4 安全

功能	功能说明
HTTP/HTTPS协议	支持HTTP/HTTPS协议，可保障客户端和服务端通信安全可信。
国密SSL	支持国密SSL双向认证通信，可保障客户端和服务端通信安全可信。
访问控制	支持IP访问控制、IP黑白名单、referer黑白名单、静态资源防盗链等功能。

4.3.5 管控

功能	功能说明
三员权限	支持三级权限控制，可对用户、角色、权限进行管理。
审计	支持审计功能，可查看用户操作日志。
流量监控	支持查看实时流量数据，以及支持查看服务器负载等信息
配置管理	支持在线管理配置文件，并自动热更新服务

4.4 部署架构



ALB标准版主备高可用部署架构如上图所示，主要包括以下组件：

- 主备ALB节点：提供负载均衡服务和配置管理。
- 控制台：管理ALB节点的配置、监控、高可用等。
- 服务：反向代理承载业务应用系统。
- VIP：虚拟IP地址，动态指向主备ALB节点。

5 产品安装

5.1 安装包说明

ALB标准版安装包名称结构是：

`ALB-版本号-SE-构建日期-CPU架构.tar.gz` 或 `ALB-版本号-SE-构建日期-CPU架构-ebpf.tar.gz`

目前仅支持linux平台下arm64和amd64架构:

- Linux-arm64架构安装包： `ALB-V2.0.5-SE-构建日期-arm64.tar.gz`
- Linux-amd64(x86_64)架构安装包: `ALB-V2.0.5-SE-构建日期-amd64.tar.gz`
- Linux-arm64架构 `ebpf` 特性安装包： `ALB-V2.0.5-SE-构建日期-arm64-ebpf.tar.gz`
- Linux-amd64(x86_64)架构 `ebpf` 特性安装包: `ALB-V2.0.5-SE-构建日期-amd64-ebpf.tar.gz`

注：构建日期为ALB产品包具体构建日期，同一个版本可能存在多个构建日期，获取安装包以最新日期为准。本手册全部演示内容中的安装包名称移除了构建日期。

注： `ebpf` 特性安装包，Linux内核版本不低于 `4.19` .

5.1.1 安装介质

- tar.gz压缩包： `ALB-版本号-SE-构建日期-CPU架构.tar.gz` 解压到目标机器任意路径即可使用。
- docker镜像： `ALB-版本号-SE-构建日期-CPU架构-docker.tar.gz` 或 `ALB-版本号-SE-Docker-构建日期-CPU架构-基础操作系统.tar.gz` 。
- RPM包： `ALB-版本号-SE-构建日期-CPU架构.rpm` ,使用 `rpm -ivh 安装包名` 安装，安装完成后，其安装路径为 `/opt/ALB-SE-版本号`

5.2 单节点部署

1. 获取软件安装包 ALB安装介质为: ALB-V2.0.5-SE-amd64.tar.gz或ALB-V2.0.5-SE-arm64.tar.gz
2. 复制安装包到目标主机
3. 解压: ALB-V2.0.5-SE-amd64.tar.gz到目标安装路径（示例为/opt目录）

`tar -zxvf ALB-V2.0.5-SE-amd64.tar.gz -C /opt/` 4. ALB使用参考【快速入门】章节内容。

注：如果没有权限，请使用root账户或sudo

5.3 集群部署

集群部署架构如上图所示，在主备节点上分别执行单节点安装步骤安装完毕后，参考本文档中的高可用部署文档进行配置即可。

5.4 Docker部署

alb标准版提供arm和x86的Docker系统安装包，可通过ALB Docker安装包直接部署ALB.

5.4.1 安装包说明

1. ALB压缩安装包名称

- `ALB-V2.0.5-SE-docker-amd64.tar.gz` 或 `ALB-V2.0.5-SE-Docker-20250225-amd64-kylin.tar.gz`
- `ALB-V2.0.5-SE-docker-arm64.tar.gz` 或 `ALB-V2.0.5-SE-Docker-20250225-arm64-kylin.tar.gz`

注： 安装包日期可能不同，请根据实际情况选择。

2. 安装包目录结构

```
tree -l 1 ALB-V2.0.5-SE-Docker-20250225-amd64-kylin
ALB-V2.0.5-SE-Docker-20250225-amd64-kylin            # docker安装包解压后目录
|-- ALB-V2.0.5-SE-Docker-20250225-amd64-kylin.tar.gz # docker镜像文件
|-- alb                                              # ALB启动配置文件、授权文件、日志等挂载目录
|   |-- alb.conf                                     # ALB配置文件
|   |-- license.xml                                  # 授权文件
-- alb-standard-dockercompose.yaml                      # docker compose配置文件
```

5.4.2 安装步骤

下面的操作以ALB-V2.0.5-SE-Docker-20250225-amd64-kylin.tar.gz为例

1. 解压和导入镜像

- 解压 `tar -zxvf ALB-V2.0.5-SE-Docker-20250225-amd64-kylin.tar.gz`
- 进入解压后的文件夹 `cd ALB-V2.0.5-SE-Docker-20250225-amd64-kylin`
- 加载镜像： `docker load < ALB-V2.0.5-SE-Docker-20250225-amd64-kylin.tar.gz`

2. 修改授权文件

ALB授权文件需要挂载到容器内部安装目录，当前样例是`alb/license.xml`，在启动容器前，请更换授权文件。

> *授权文件挂载说明*
>
> - *license.xml支持旧授权*
> - *license.xml支持KBC授权（如果是KBC授权，把kbc授权内容复制写入license.xml即可）*
> - *acls.properties统一授权配置文件需要添加挂载到容器路径：`/opt/ALB-V2.0.5-SE`*

3. 镜像启动与停止

- 启动命令：`docker-compose -f alb-standard-docker-compose.yaml up -d`
- 停止命令：`docker-compose -f alb-standard-docker-compose.yaml down`

4. `alb-standard-docker-compose.yaml`配置说明

```
version: '3'
services:
  alb:
    image: harbor.apusic.com/apusic/alb:V2.0.5-ae.20250225-kylin-
amd64    # ALB 镜像
    ports:
      - 8080:8080           # ALB http对外端口
      - 8887:8887           # ALB 管控台端口
    volumes:
      - ./alb/alb.conf:/opt/ALB-V2.0.5-SE/conf/alb.conf       # alb配
置文件
      - ./alb/license.xml:/opt/ALB-V2.0.5-SE/license.xml      # alb授
权文件
      - ./alb/logs:/opt/ALB-V2.0.5-SE/logs                    # alb访
问和错误日志存储目录
    command: bash /opt/ALB-V2.0.5-SE/bin/start-alb-and-console.sh
```

5.5 K8s部署

alb标准版支持通过Kubernetes进行部署，以下是详细的部署步骤。

5.5.1 准备工作

- 环境要求
  - 已安装 kubectl 并配置好 Kubernetes 集群访问权限。
  - 若使用私有镜像仓库（如Harbor），确保集群有权限拉取镜像。
- 镜像准备
  - 将 Docker 镜像上传至镜像仓库（以 AMD64 镜像为例）

```
# 解压安装包
tar -zxvf ALB-V2.0.5-SE-Docker-20250225-amd64-kylin.tar.gz
# 切换到解压后的安装包
cd ALB-V2.0.5-SE-Docker-20250225-amd64-kylin
# 导入ALB镜像
docker load < ALB-V2.0.5-SE-Docker-20250225-amd64-kylin.tar.gz
# 重新标记镜像，请根据自己镜像仓库地址进行修改
docker tag [镜像ID] harbor.apusic.com/apusic/alb:V2.0.5-
ae.20250225-kylin-amd64
# 推送镜像至镜像仓库
docker push [镜像ID]
```

5.5.2 配置文件与授权管理

- 创建 ConfigMap 将 `alb.conf` 配置文件挂载至容器：

```
kubectl create configmap alb-config --from-file=./alb/alb.conf
```

- 创建 Secret 将 `license.xml` 授权文件以 Secret 形式存储（敏感信息建议使用 Secret）：

```
kubectl create secret generic alb-license --from-
file=./alb/license.xml
```

5.5.3 部署ALB服务

1. 创建 Deployment 编写 alb-deployment.yaml 文件：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: alb
  template:
    metadata:
      labels:
        app: alb
    spec:
      containers:
      - name: alb
        image: harbor.apusic.com/apusic/alb:V2.0.5-se.20250225-kylin-amd64 # 请根据仓库地址进行修改
        command: ["bash", "/opt/ALB-V2.0.5-SE/bin/start-alb-and-console.sh"]
        ports:
        - containerPort: 8080
        - containerPort: 8887
        volumeMounts:
        - name: alb-config
          mountPath: "/opt/ALB-V2.0.5-SE/conf/alb.conf"
          subPath: "alb.conf"
        - name: alb-license
          mountPath: "/opt/ALB-V2.0.5-SE/license.xml"
          subPath: "license.xml"
        - name: alb-logs
          mountPath: "/opt/ALB-V2.0.5-SE/logs"
      volumes:
      - name: alb-config
        configMap:
```

```
          name: alb-config
      - name: alb-license
        secret:
          secretName: alb-license
      - name: alb-logs
        hostPath:
          path: "/var/alb/logs"
          type: DirectoryOrCreate
```

参数说明

- image: 替换为实际镜像地址。
- hostPath: 日志目录挂载路径，可按需改为 PVC（如使用云存储）。

2. 创建 Service

```
apiVersion: v1
kind: Service
metadata:
  name: alb-service
spec:
  type: NodePort
  selector:
    app: alb
  ports:
  - name: http
    port: 8080
    targetPort: 8080
    nodePort: 30080
  - name: console
    port: 8887
    targetPort: 8887
    nodePort: 30887
```

参数说明

- nodePort 默认范围为 30000-32767，若省略端口则由系统自动分配

- 若在云环境，可将 type 改为 LoadBalancer 直接获取外部 IP

3. 应用配置

```
kubectl apply -f alb-deployment.yaml
kubectl apply -f alb-service.yaml
```

5.5.4 部署验证

1. 检查 Pod 状态

```
kubectl get pods -l app=alb
```

2. 访问服务

- HTTP 服务：`http://<节点IP>:30080`
- 管控台：`http://<节点IP>:30887`

5.5.5 停止与清理

1. 删除部署

```
kubectl delete -f alb-deployment.yaml -f alb-service.yaml
```

2. 清理配置

```
kubectl delete configmap alb-config
kubectl delete secret alb-license
```

6 快速入门

产品使用前需切换进入到ALB安装目录，示例为`/opt/ALB-V2.0.5-SE-amd64`，并将申请的ALB标准版KBC授权文件`license.lic`放入到安装目录下。

6.1 产品配置

ALB 核心兼容Nginx配置，具体配置文件conf/alb.conf路径为：

安装路径`/conf/alb.conf`

可使用Nginx的配置方式及其配置内容进行配置ALB

6.2 产品启动、停止、热加载、卸载、验证配置文件和获取授权码

- 启动ALB实列和管理控制台

```
cd /opt/ALB-V2.0.5-SE-amd64                    # 进入安装目录
./bin/start-alb-and-console.sh                          # 启动alb和管理控制台
```

- 启动ALB实例，但不启动管控台

```
cd /opt/ALB-V2.0.5-SE-amd64                    # 进入安装目录
./bin/start-alb.sh                           # 启动alb
```

- 停止ALB实列和管理控制台

```
cd /opt/ALB-V2.0.5-SE-amd64                    # 进入安装目录
./bin/stop-alb-and-console.sh                           # 停止alb和管理控制台
```

- 停止ALB实例

```
cd /opt/ALB-V2.0.5-SE-amd64                    # 进入安装目录
./bin/stop-alb.sh                            # 停止alb
```

- 热加载

```
cd /opt/ALB-V2.0.5-SE-amd64                    # 进入安装目录
./bin/reload-alb.sh                            # 停止alb
```

- 卸载
    - 解压安装包
        - 直接删除安装目录
    - rpm：
        - x86：`rpm -e ALB-SE-2.0.5-1.el7.x86_64`
        - arm64：`rpm -e ALB-SE-2.0.5-1.el7.aarch64`
- 验证配置文件

可以使用-t参数验证配置文件的配置是否存在错误

验证成功提示：`syntax is ok` 和 `test is successful`

```
cd /opt/ALB-V2.0.5-SE-amd64                    # 进入安装目录
./sbin/alb -t                                  # 验证conf/alb.conf配置文件是否
存在语法错误
```

验证指定的配置文件，可以通过 `-c` 参数指定配置文件

```
cd /opt/ALB-V2.0.5-SE-amd64                    # 进入安装目录
./sbin/alb -t -c conf/alb.conf                 # 指定验证conf/alb.conf配置文件
是否存在语法错误

# 输出
alb:the configuration file /opt/ALB-V2.0.5-SE-amd64/conf/alb.conf
syntax is ok
alb:configuration file /opt/ALB-V2.0.5-SE-amd64/conf/alb.conf test
is successful
```

- 产品获取授权码

```
cd /opt/ALB-V2.0.5-SE-amd64                # 进入安装目录
./bin/alb-authcode                         # 获取alb在当前节点的授权码
# 注：  如果出现多个授权码，选择任意一个授权码申请授权文件即可

./bin/alb-authcode -ac eth0                # 获取指定网卡的授权码
./bin/alb-authcode -ac 172.21.61.110       # 获取指定IP的授权码
```

6.3 配置反向代理

步骤如下：

1. 准备目标需要反向代理服务器IP和服务端口。
2. 修改ALB的配置文件，配置监听端口和反向代理到目标服务器。
3. 使用热更新命令更新alb配置。
4. 验证反向代理测试。

> *例子： 现在有个ftp服务器： 172.21.32.42:8080，需要通过alb代理发布内容，在alb节点可以正常访问ftp服务器，则可以通过配置alb的反向代理实现代理ftp服务。*

1. 目标代理服务： 172.21.32.42:8080
2. 修改alb.conf文件，配置监听端口和反向代理到目标服务器。

```
# 在http块中新增代理server块
http{
  # 省略其他配置内容，下面server块为增加内容
  server {
      listen 80;    # 监听80端口

      location / {    # 全部访问的url转发到 172.21.32.42:8080
          proxy_pass http://172.21.32.42:8080/;
      }
  }
}
```

3. 使用热更新命令更新alb配置。

```
./bin/reload-alb.sh
```

4. 验证反向代理测试。

```
curl http://172.21.32.42:80
```

6.4 配置代理静态资源

步骤如下：

1. 准备静态资源，上传到ALB节点(172.21.32.42)的/data/www目录下。
2. 修改ALB的配置文件，新增代理静态资源配置。
3. 使用热更新命令更新alb配置。

> *例子： 先有一个纯HTML等静态资源构成的网址，需要通过alb发布网站。*

1. 解压目标网站静态资源到ALB节点目录。
2. 修改alb.conf文件，新增代理静态资源配置。

```
# 在http块中新增代理server块
http{
  server {
    listen 80;
    location / {
      root /data/www;                    # 通过root指令，指定静态资源路径
      index index.html;                  # 访问首页的默认文件，比如直接访问/，则访问index.html
      try_files $uri $uri/ /index.html; # 尝试访问uri，如果失败则尝试访问uri/，最后访问index.html
    }
  }
}
```

3. 使用热更新命令更新alb配置。

```
./bin/reload-alb.sh
```

4. 验证静态资源测试。

```
curl http://172.21.32.42:80/index.html
```

6.5 配置动静分离

步骤如下：

1. 准备静态资源，上传到ALB节点(172.21.32.42)。
2. 准备待代理的后端服务器节点。
3. 修改ALB的配置文件，新增动静分离配置。
4. 使用热更新命令更新alb配置。

> *例子： 现在有个动静分离的网站，需要通过ALB代理以static开头url的静态资源和以api开头url的API接口。*

1. 准备静态资源，上传到ALB节点的/data/www目录下。
2. 后端API理服务： 172.21.32.42:8080和172.21.32.43.8080。
3. 修改alb.conf文件，新增动静分离配置。

```
http{

  # 定义后端api服务器地址
  upstream api {
    server 172.21.32.42:8080;
    server 172.21.32.43:8080;
  }

  server {
    listen 80;
    location /static/ {    # static开头的url转发到/data/www/static/
      root /data/www;
      index index.html;
    }
    location /api/ {    # api开头的url转发到定义的upstream api中。
```

```
        proxy_pass http://api/;
      }
    }
}
```

4. 使用热更新命令更新alb配置。

```
./bin/reload-alb.sh
```

5. 验证动静分离测试。

```
# 获取静态资源index.html
curl http://172.21.32.42:80/static/index.html
# 请求api
curl http://172.21.32.42:80/api/user/list
```

7 管控台使用说明

ALB管控台是一个基于Web的管理和监控工具，对ALB负载均衡器进行可视化管理，降低了用户直接操作的使用难度，同时提高了管理效率。ALB管控台支持监控ALB运行状态、数据流量、节点负载情况、配置管理、证书管理、配置高可用、操作日志等

7.1 三员权限说明

ALB管控台使用三员管理方式，系统默认提供四种角色：

- admin-security（**安全管理员**），安全管理员主要负责用户管理操作，如：用户的注册、删除、编辑、查看用户列表、查看用户详情。
- admin-audit（**审计管理员**），审计管理员主要负责安全审计操作，如：查看操作日志列表、查看操作日志详情。
- admin-alb（**系统管理员**），系统管理员主要负责所有的ALB管理操作，如：实例流量查看、配置管理、SSL证书管理、日志查看、高可用配置等。
- user-alb（**普通用户**），普通用户只支持所有ALB操作的查看工作，无任何修改权限。

四种角色初始默认用户及其密码：

角色名称	用户名	初始密码
安全管理员	admin-security	Apusic@alb@admin1
审计管理员	admin-audit	Apusic@alb@admin2
系统管理员	admin-alb	Apusic@alb123

7.2 管控台登录

管控台的登录网址分为如下两种：

1. 安全管理员/审计员管理员： http://{ALB_IP}:8887/admin
2. 系统管理员/普通用户： http://{ALB_IP}:8887

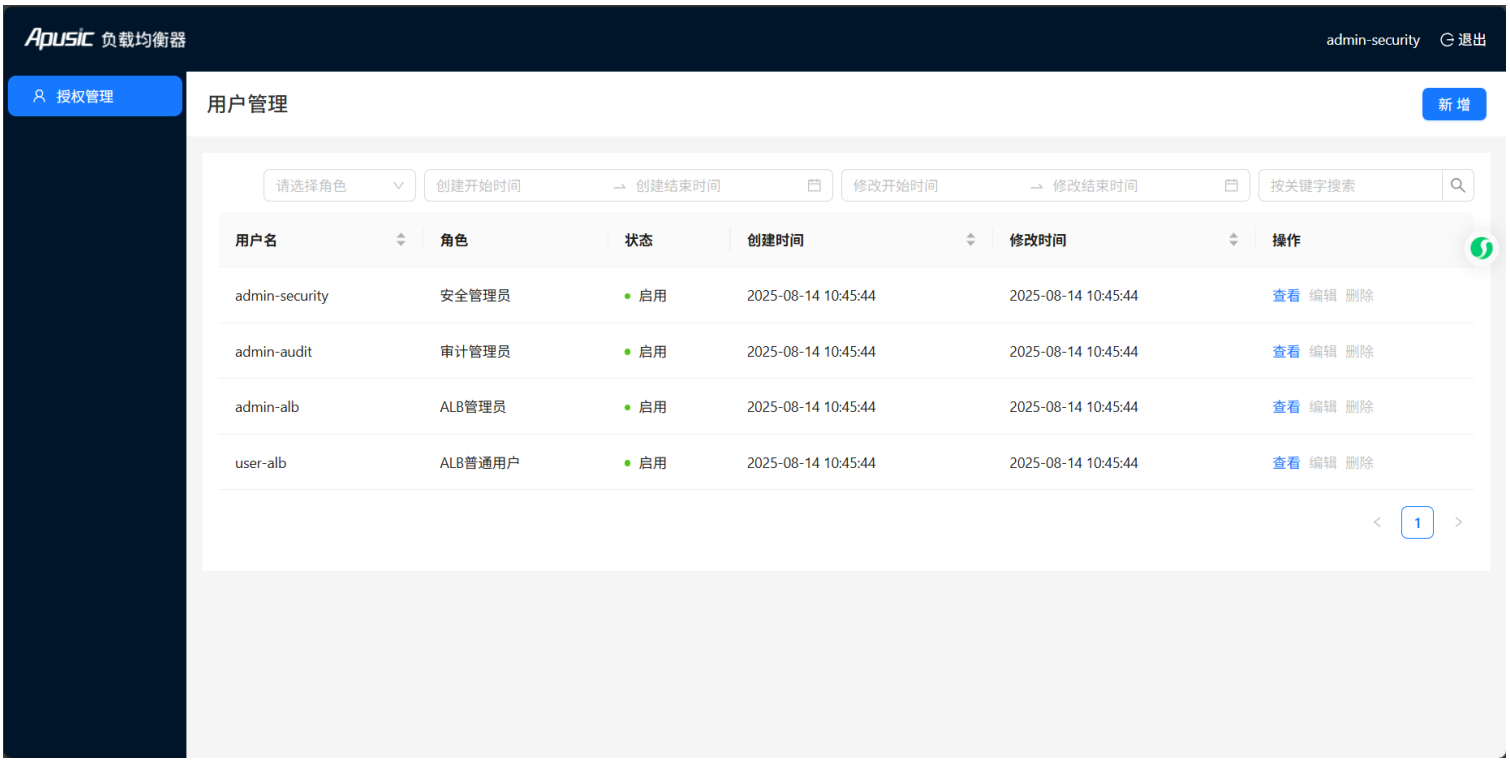
如果要使用https协议，需要打开ALB管控台配置文件`ALB安装目录/console/conf/config.yaml`，修改如下配置启用https协议：

```
......
system:
  ......
  # 使用https
  https: true
  certfile: "./cert/cert.pem" # 默认的证书文件
  keyfile: "./cert/key.key" # 默认的密钥文件
......
```

如果不使用默认证书和密钥文件，直接将自己的证书和密钥文件放到`ALB安装目录/console/cert/`目录下。或者放入到任意目录，然后修改配置项目`certfile`和`keyfile`指向对应文件。

7.3 安全管理员操作

安全管理员登录成功之后，默认页面如下：

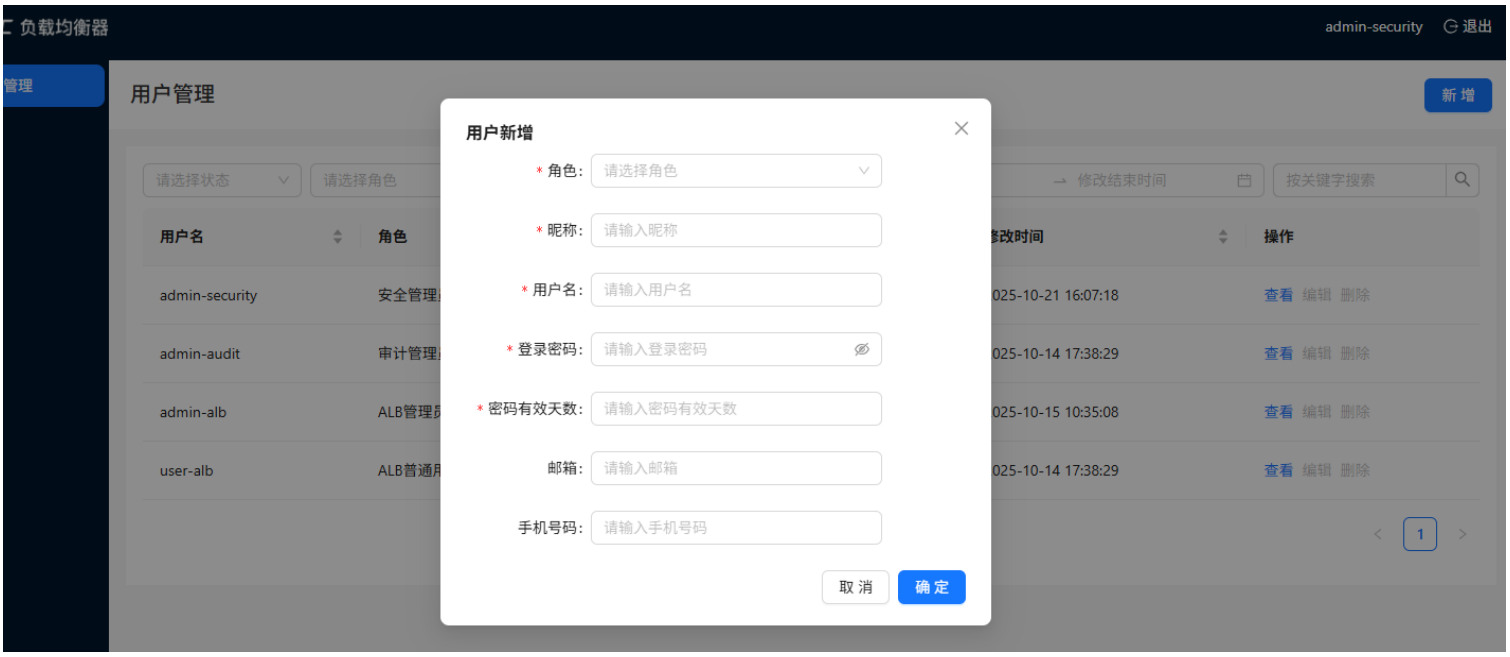


对于每条记录，提供以下信息：

- 用户名：表示用户的名称，用户唯一标识
- 角色：表示用户的角色
- 状态：表示用户当前的状态
- 创建时间：表示用户注册的时间
- 修改时间：表示用户修改的时间

7.3.1 注册用户

点击“新增”，进入注册用户页面

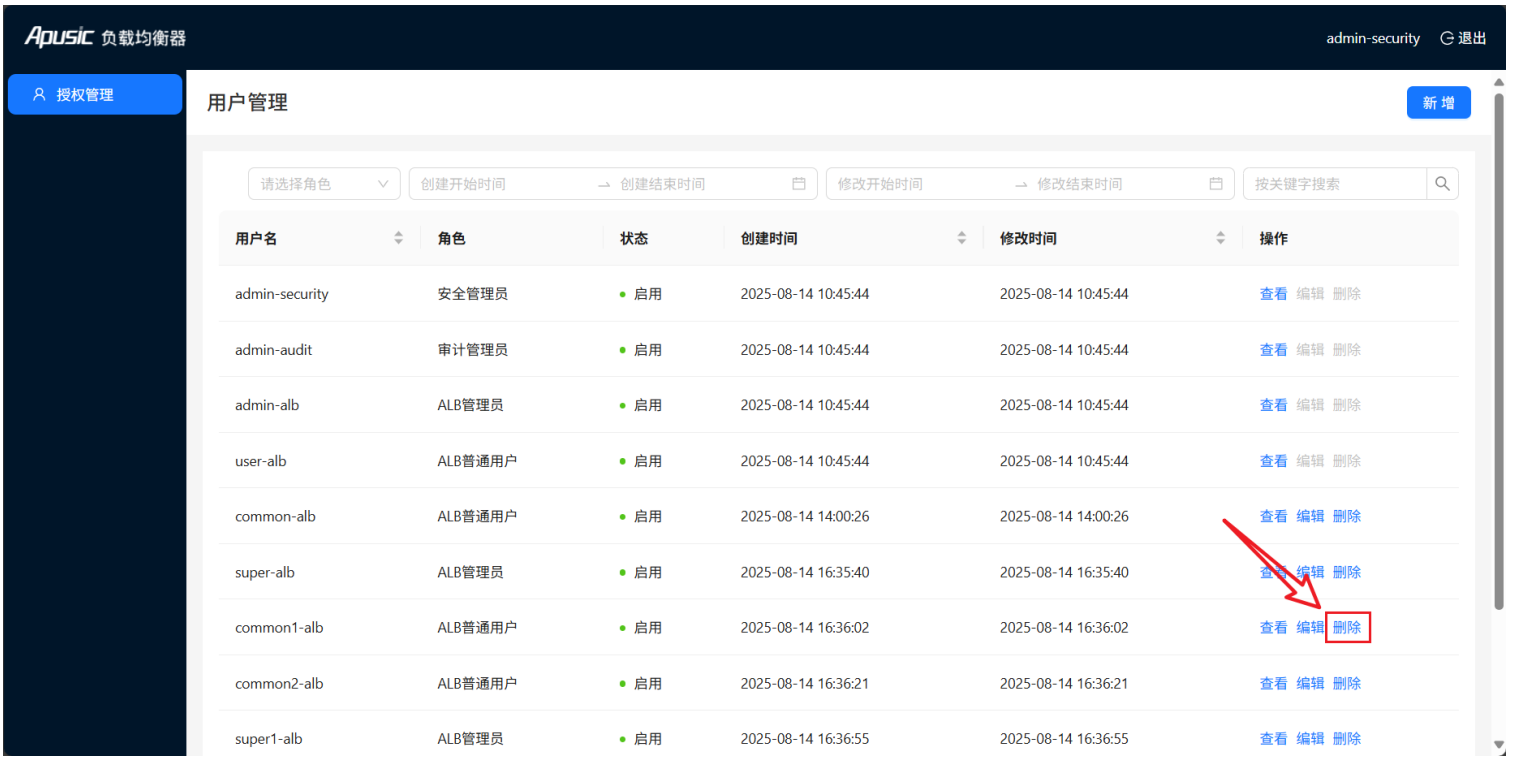


用户信息字段说明：

- 角色：选择系统提供的默认四种角色之一
- 昵称：用户名的别称，不能为空
- 用户名：用户登录系统时的标识，具有唯一性，仅由大小写字母、数字、点、下划线、减号组成
- 登录密码：由大小写字母、数字、特殊字符（如 `!@#$%^&*` ）中的三者组成，并且长度在8到50个字符之间
- 密码有效天数：自创建用户/编辑用户密码/用户自修改密码起密码的有效时长，单位是天，超过时长，用户需修改密码，否则被禁用。密码到期前一星期每次登录会提示密码到期时间。用户首次登录也会提示修改密码。-1表示永久有效
- 邮箱：符合常见邮箱格式（非必须）
- 手机：符合中国手机号格式（非必须）

7.3.2 删除用户

选择用户，点击"删除"可删除该用户，默认用户无法删除



7.3.3 编辑用户

选择用户，点击“编辑”，进入编辑页面，即可编辑用户基本信息，如果编辑前后信息未发生任何变化，则会提示相关信息。默认用户无法编辑

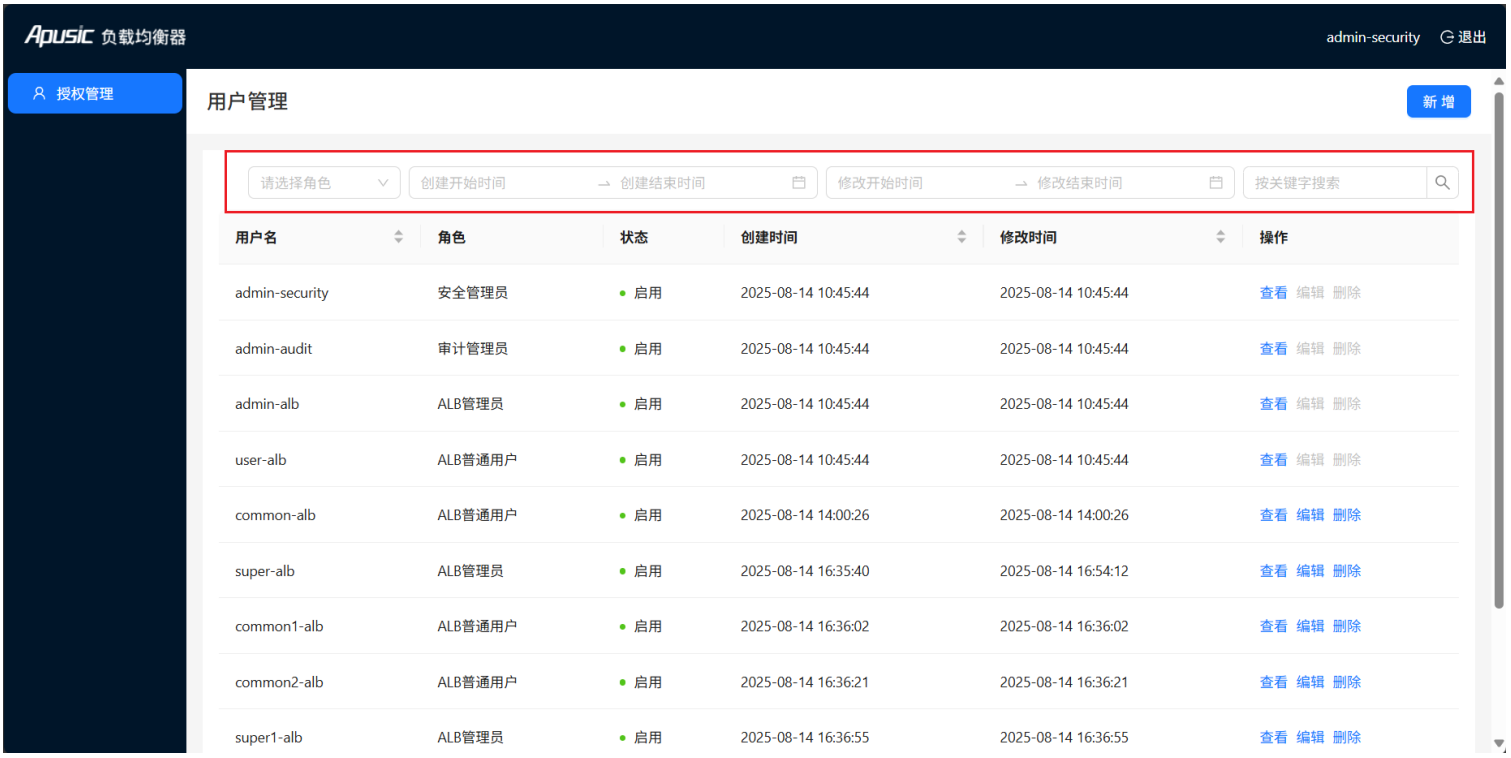
编辑用户和注册用户的区别就是编辑用户可以改变用户的状态

状态说明：

- 正常：用户处于正常状态，可以做任何非越权操作
- 禁用：用户处于禁用状态，无法任何操作，包括登录

7.3.4 查看用户列表

支持按条件进行分页获取用户列表



以下是条件和分页说明，每个条件都是可选的：

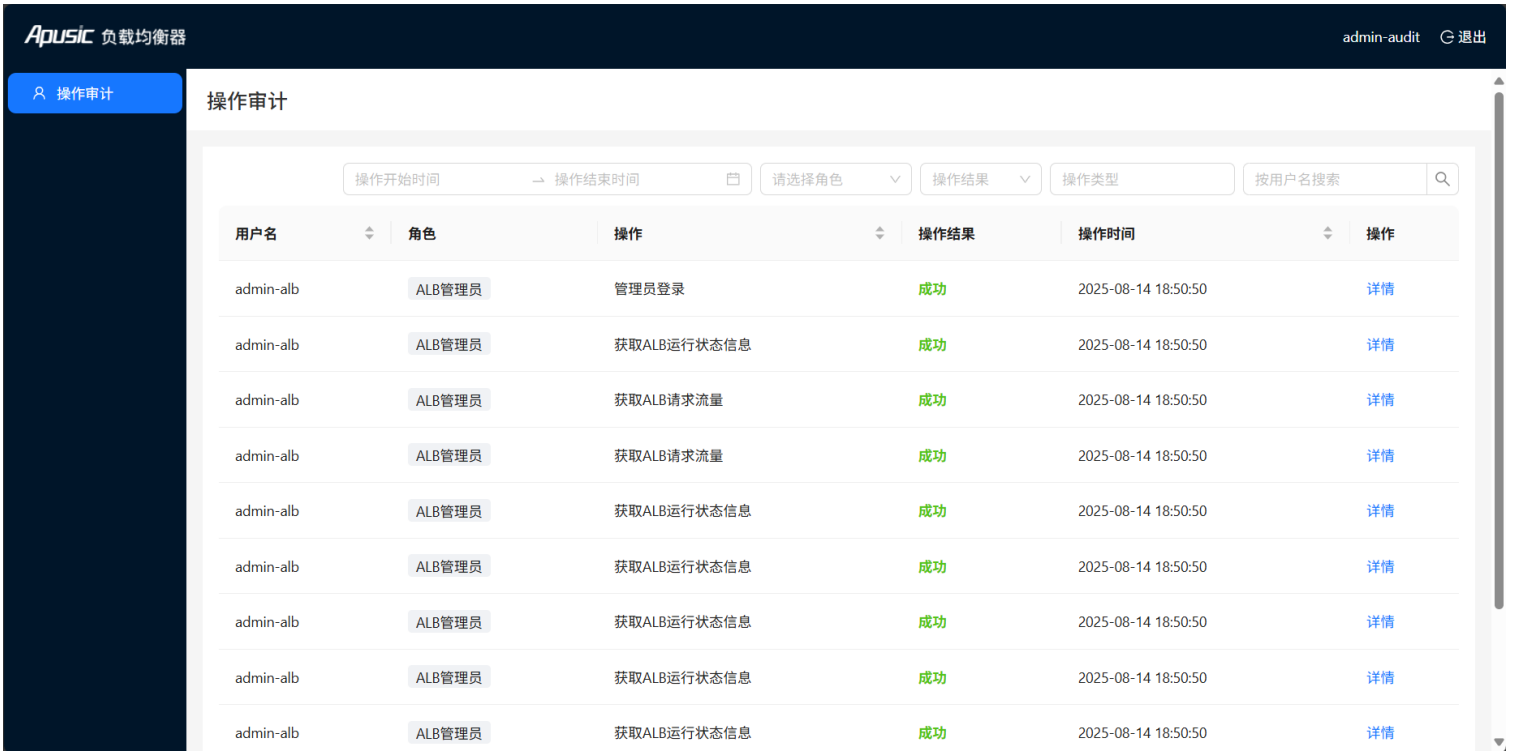
- 角色：可以选择系统提供的四种角色进行过滤
- 时间：可以根据创建时间和修改时间进行过滤
- 用户名：可以在搜索框中输入用户名进行过滤，支持模糊匹配
- 用户状态：可以选择状态进行过滤
- 排序字段：支持按照用户名、创建时间、修改时间排序，默认是按照创建用户的先后顺序排序。点击用户名、创建时间、修改时间字段可以选择降序还是升序
- 排序顺序：支持按照升序或降序排序，默认是升序

7.3.5 查看用户详情

选择用户，点击“查看”可查看用户详情

7.4 审计管理员操作

审计管理员登录成功之后，默认页面如下：

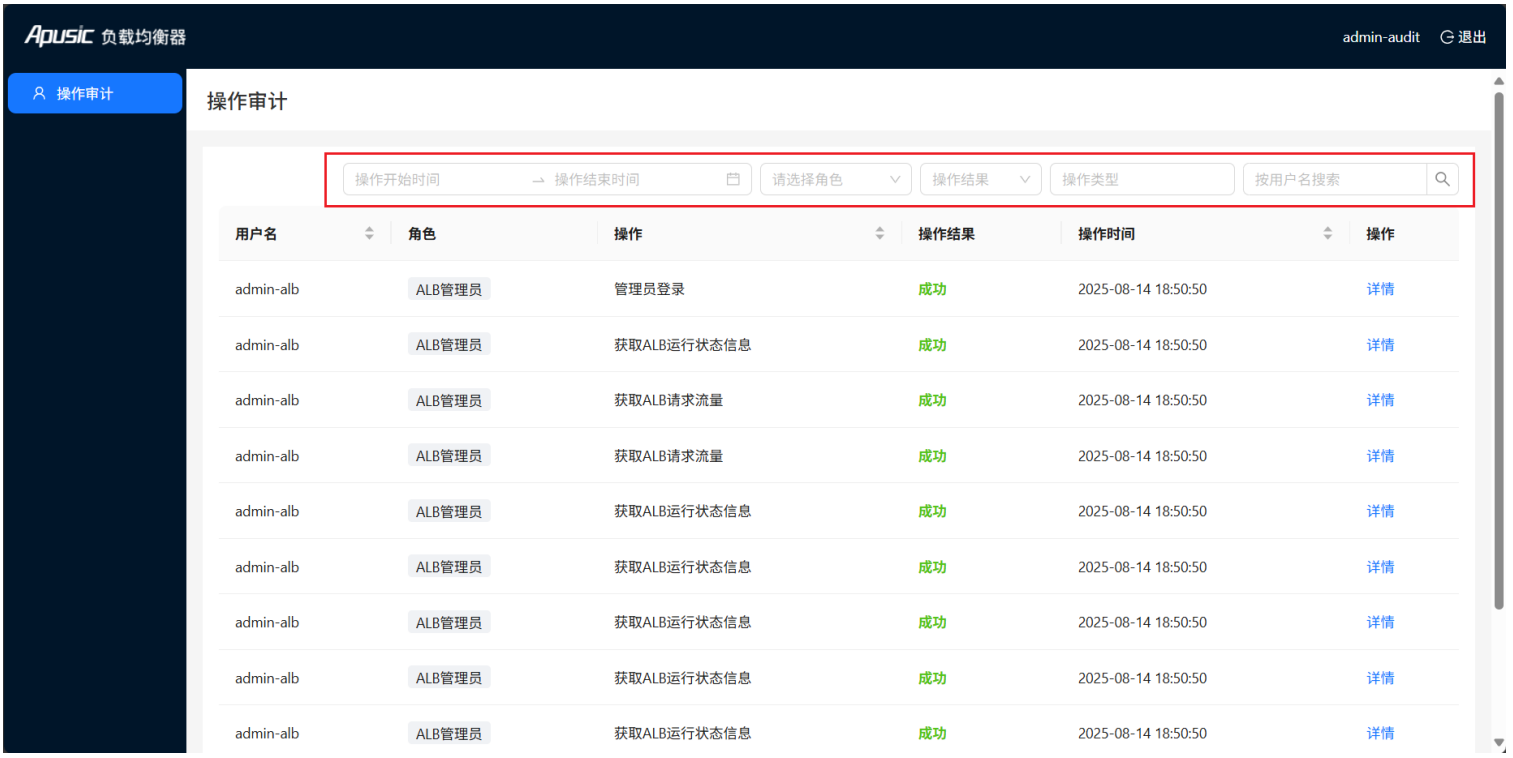


对于每条记录，提供以下信息：

- 用户名：表示此操作的用户是谁
- 角色：表示操作用户的角色
- 操作：表示用户进行了什么操作
- 操作结果：表示用户操作是否成功
- 操作时间：表示用户操作的时间

7.4.1 查看操作日志列表

支持按条件进行分页获取操作日志列表

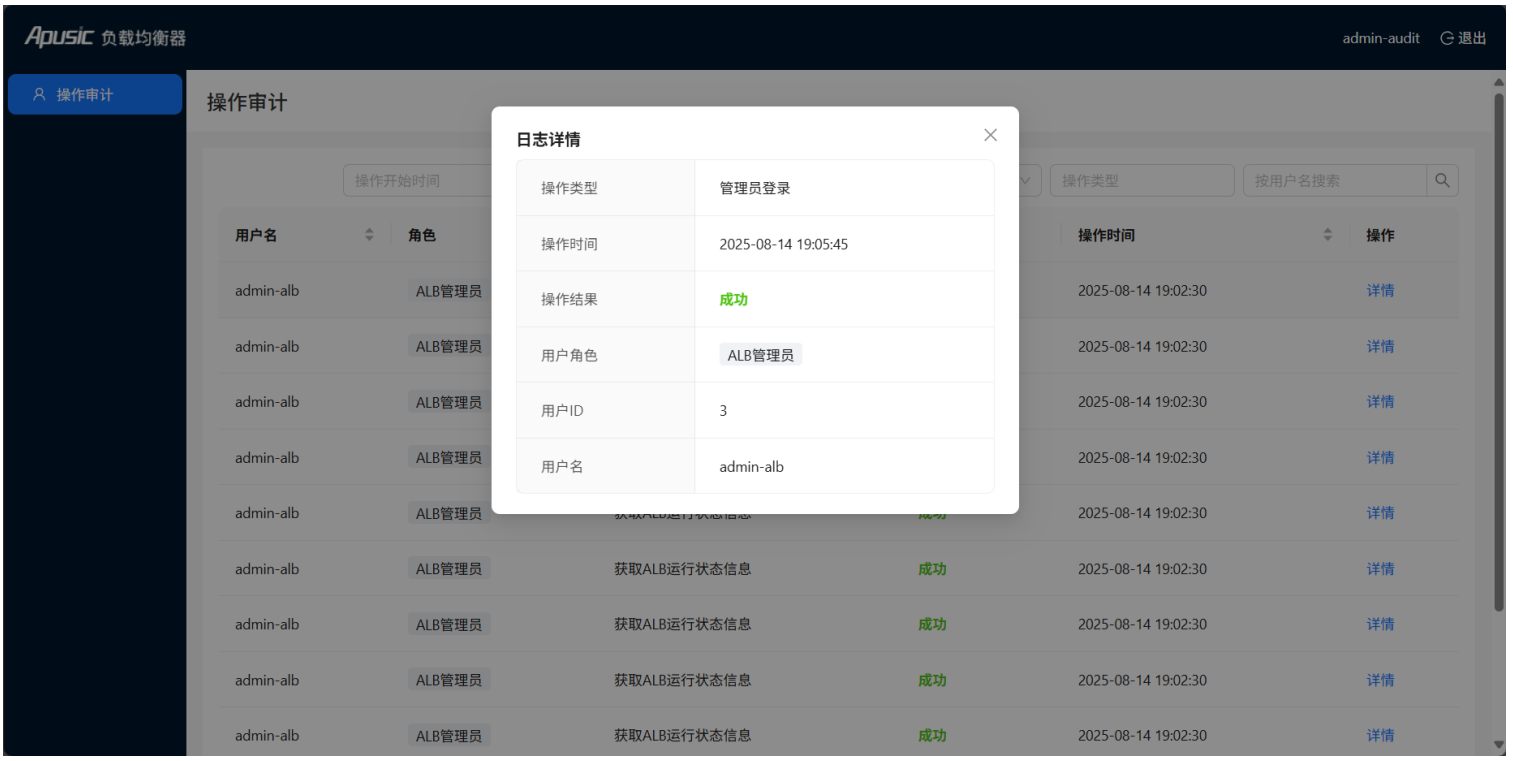


以下是条件和分页说明，每个条件都是可选的：

- 用户名：可以在用户名搜索框中输入用户名进行过滤，支持模糊匹配
- 角色：可以选择系统提供的四种角色进行过滤
- 操作：可以在操作类型搜索框中输入操作关键字进行过滤，支持模糊匹配
- 操作结果：可以选择操作结果进行过滤
- 时间：可以根据操作时间进行过滤
- 排序字段：支持按照用户名、操作、操作时间排序，默认按照操作的时间进行排序
- 排序顺序：支持按照升序或降序排序，默认是升序

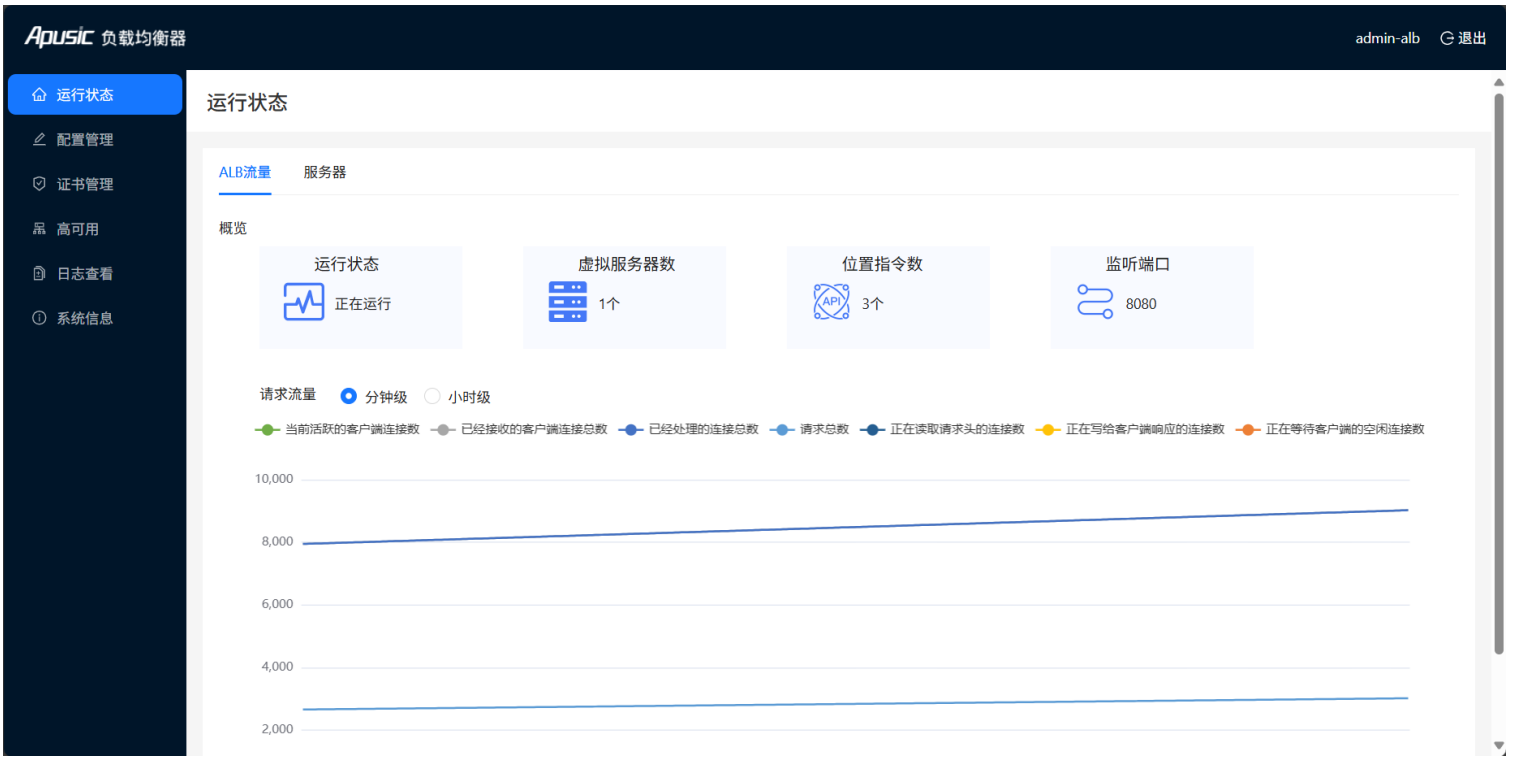
7.4.2 查看操作日志详情

选择日志，点击“查看“可查看日志详情



7.5 系统管理员操作

系统管理员登录成功之后，默认页面如下：



7.5.1 运行状态

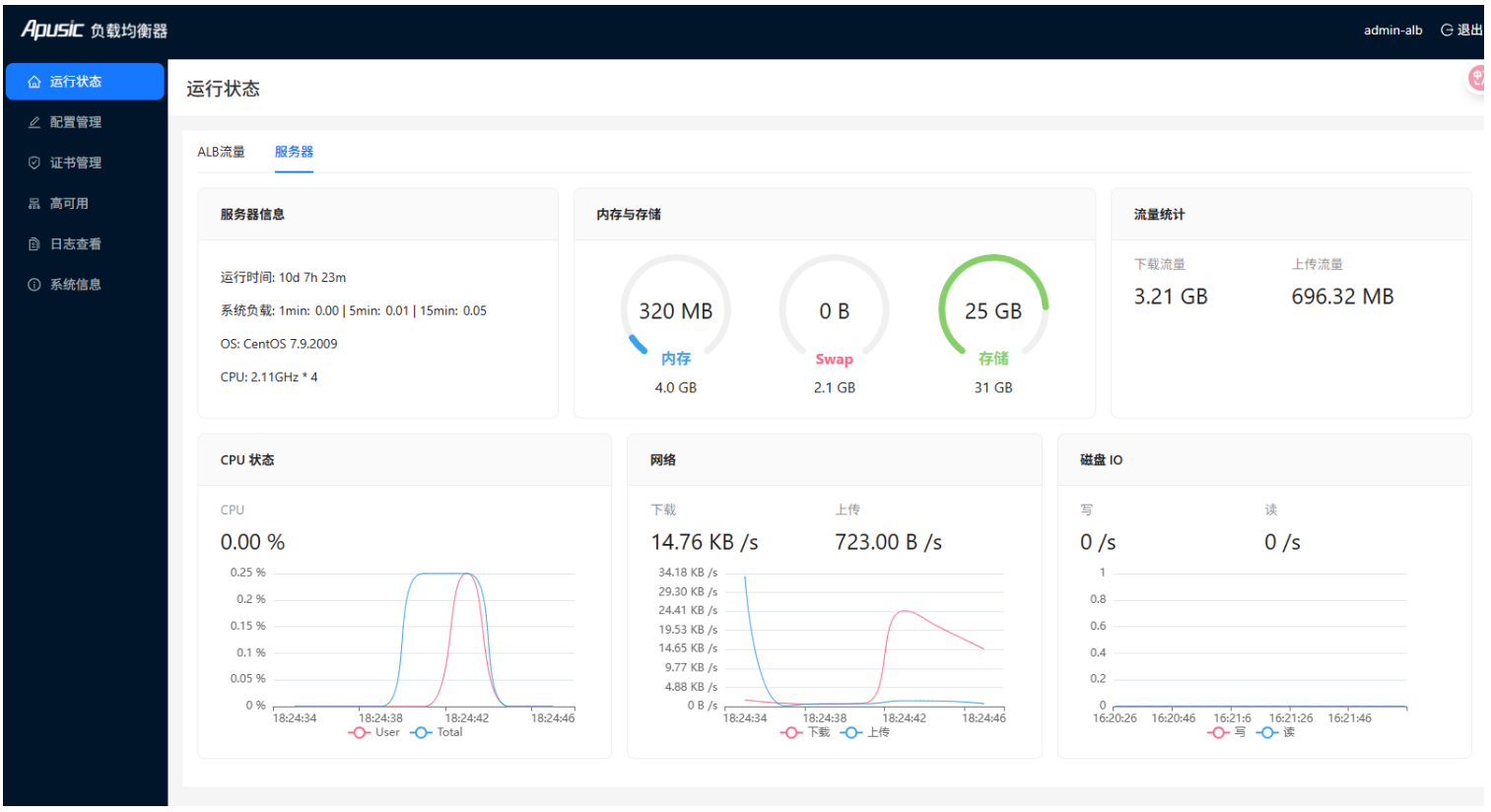
运行状态页面是监控入口，可以监控ALB流量和节点负载情况

其中ALB流量标签下，可以监控ALB节点的请求数据和虚拟服务器信息，如下所示：

页面信息说明：

- 运行状态：ALB是否运行中
- 虚拟服务器数量：ALB的总虚拟服务器数量
- 位置指令数：所有虚拟服务中总的路由块数量
- 监听端口：所有虚拟服务监听的端口
- 请求流量：以折线图的形式展示流量趋势，支持按分钟级和小时级查看ALB流量情况，如当前活跃的客户端连接数、已经处理的连接数、请求总数等

其中服务器标签下，可以监控ALB所部署节点的负载情况：



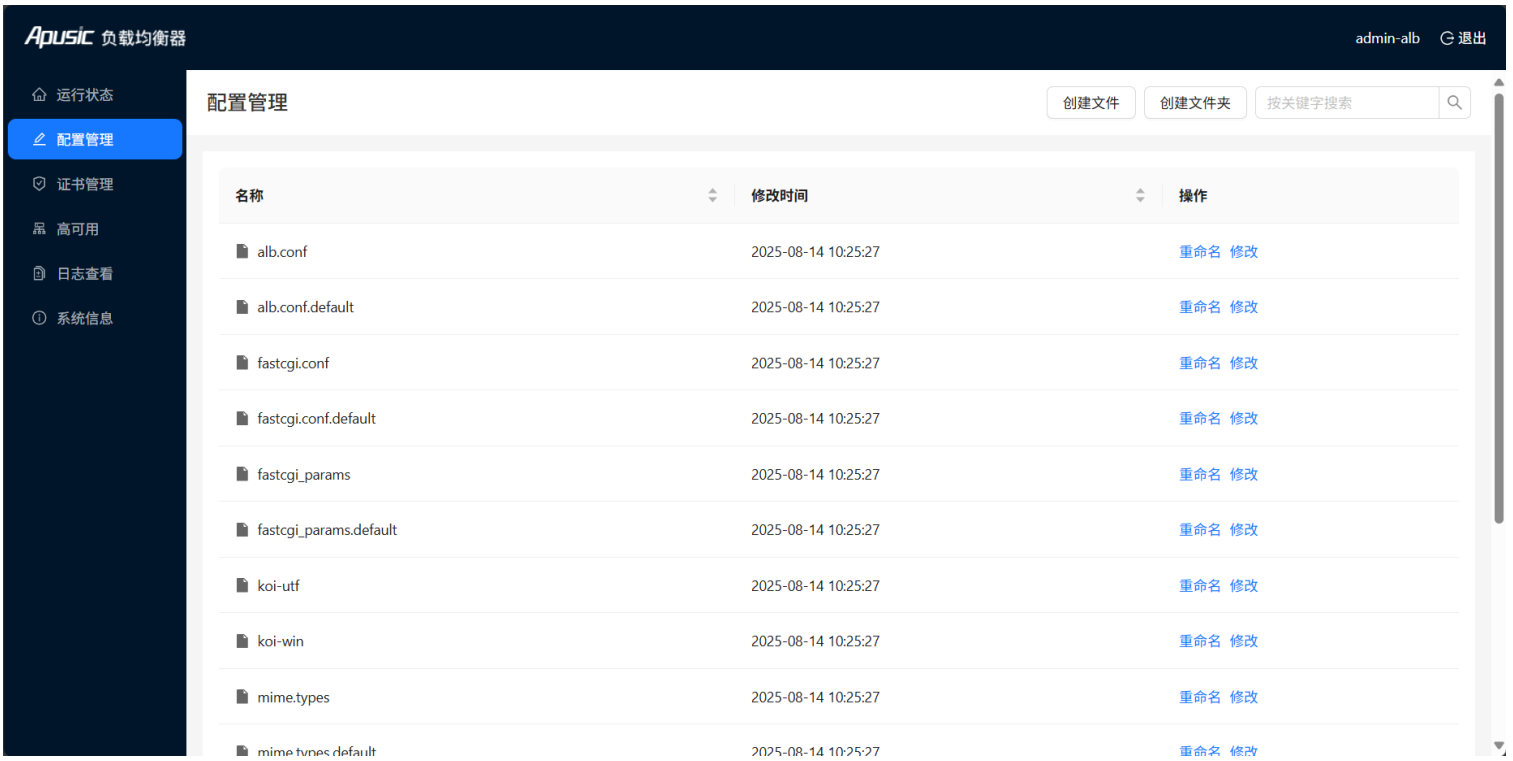
页面信息说明：

- 服务器信息：ALB运行时间，格式为天/时/分；系统的负载情况；操作系统平台标识；CPU的主频
- 内存与存储：内存大小
- 流量统计：客户端向服务器发送的数据量（上传流量）和服务器向客户端发送的数据量（下载流量）
- CPU状态：CPU的使用率
- 网络：网络上传和下载的速率
- 磁盘IO：磁盘IO的速率

7.5.2 配置管理

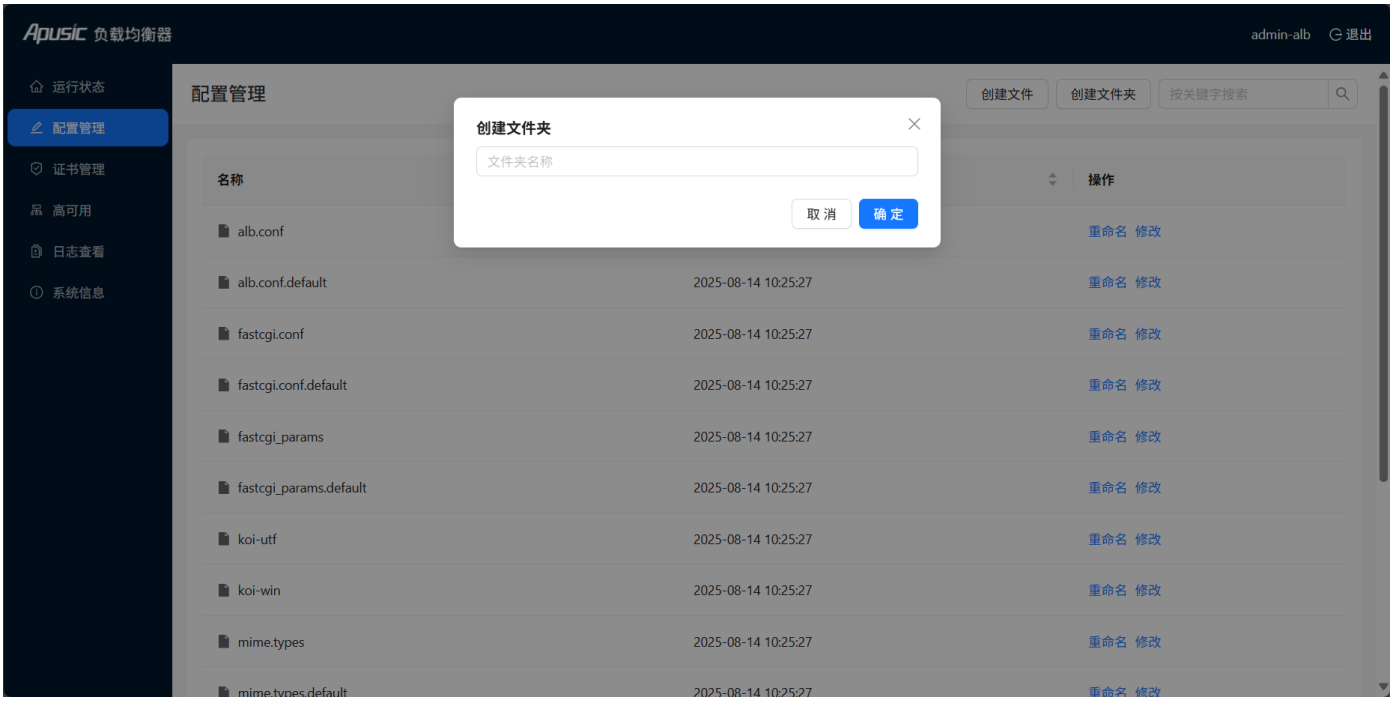
配置页面是ALB全部配置管理的入口，对ALB安装目录下的conf文件夹内全部文件进行管理

如下页面中的所有文件都是默认的配置文件：

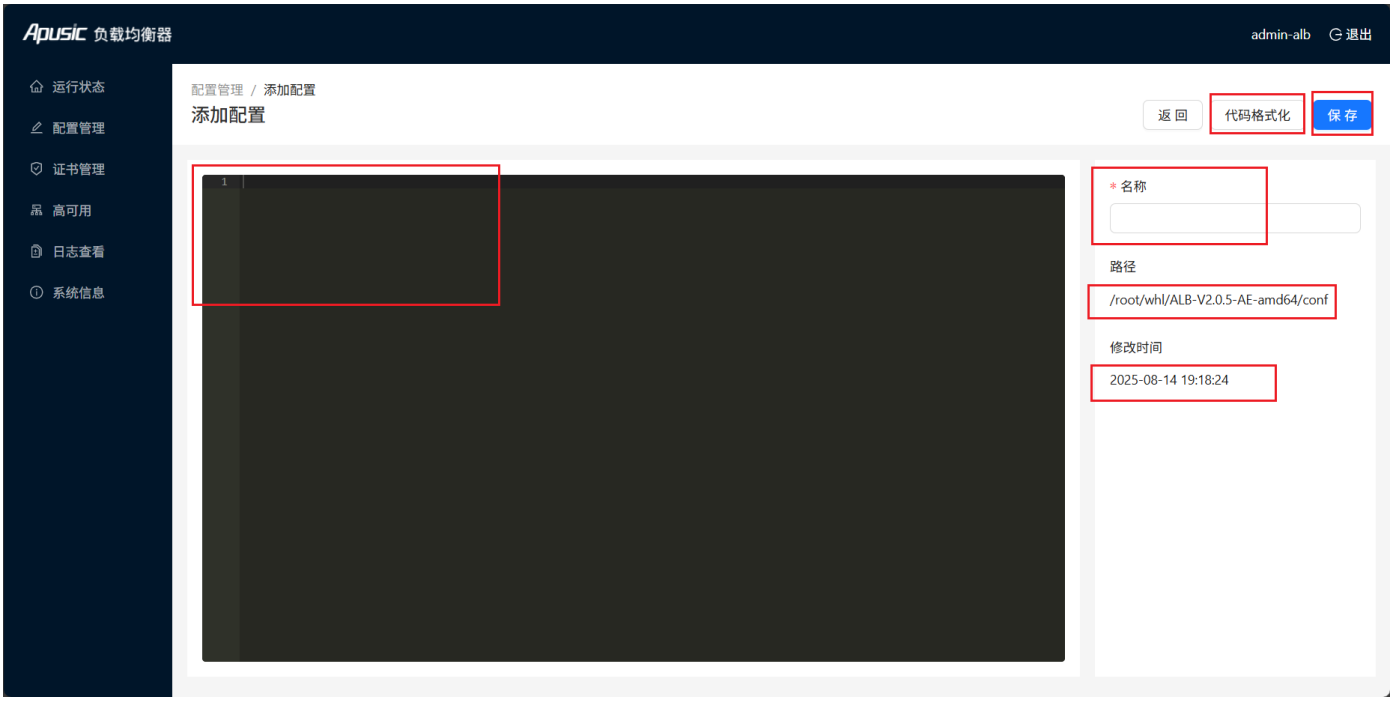


页面信息说明：

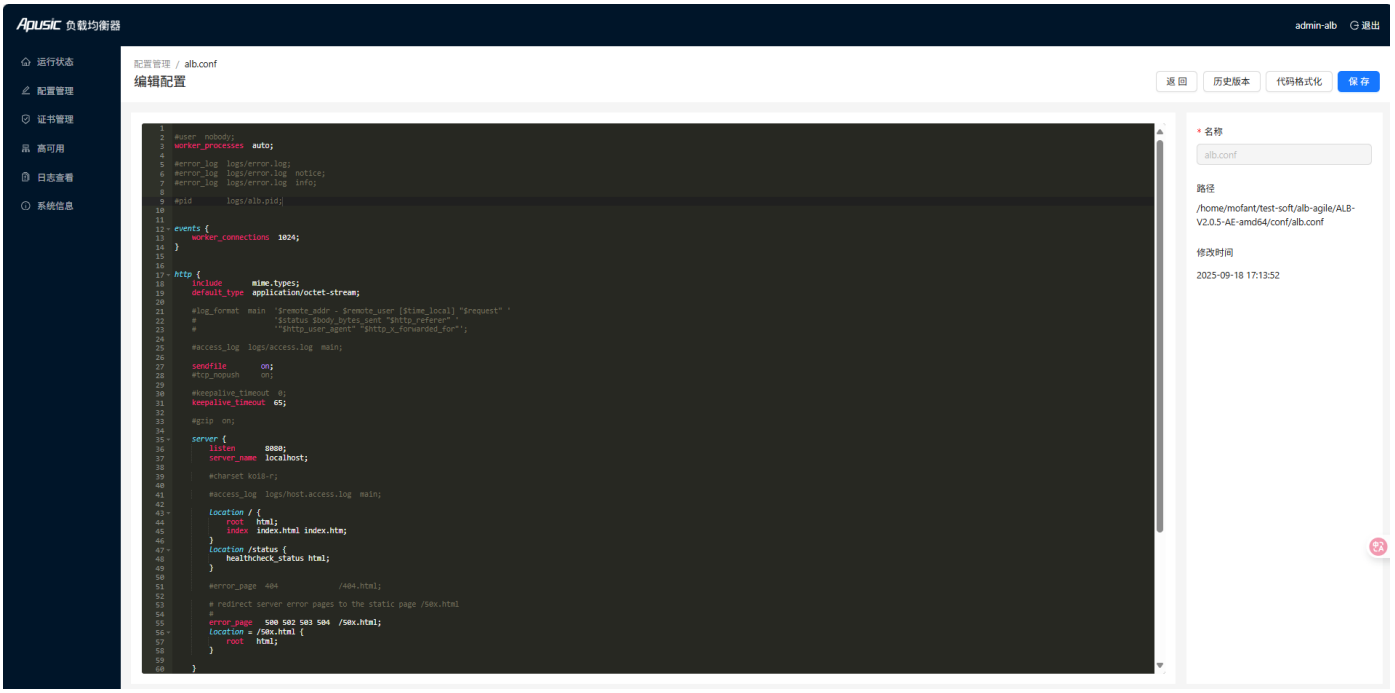
- 名称：配置文件的名称
- 修改时间：文件最近一次修改的时间
- 创建文件夹：点击“创建文件夹”按钮，进入如下页面，即可创建文件夹。文件名命名规则是长度在1~255之间的字符串，并且该字符串不能包含 `\/:*?"<>|` 字符，以及ASCII控制字符



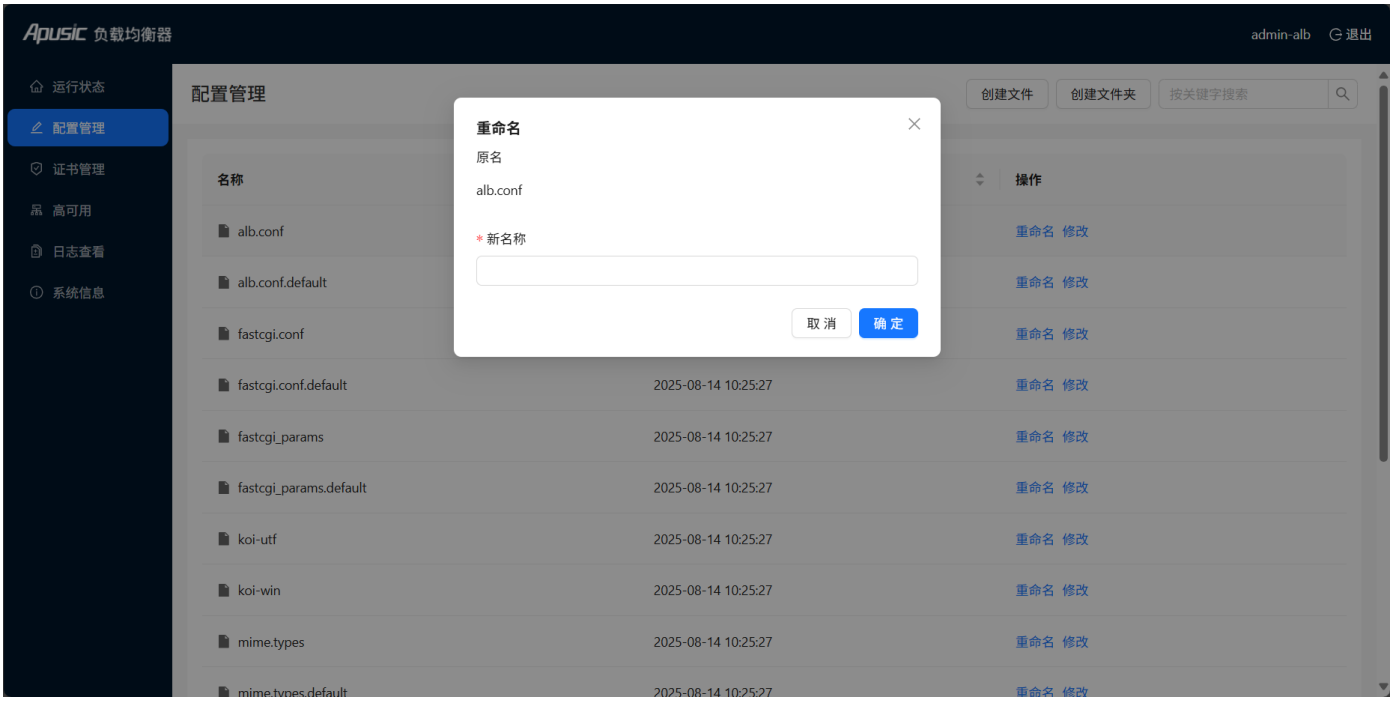
- 创建文件：点击“创建文件”按钮，进入如下页面，自定义配置文件内容、文件名，点击保存即可。文件名命名规则是长度在1~255之间的字符串，并且该字符串不能包含 `\/:*?"<>|` 字符，以及ASCII控制字符。页面中的“代码格式化”按钮，可以对文件内容进行格式调整。页面中还可以看到文件的绝对路径和修改时间。如果编辑的文件内容没有保存就返回，则会进行相关提示



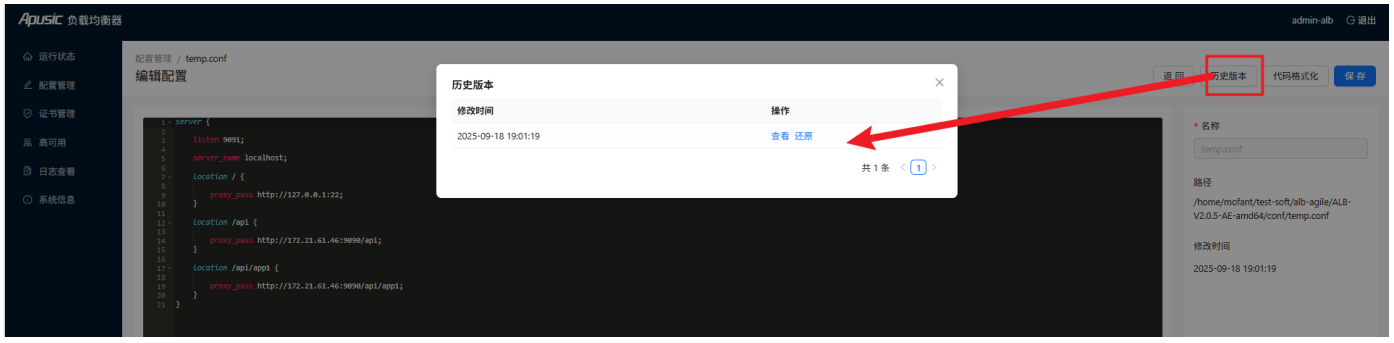
- 修改：选择对应文件点击“修改”按钮，进入如下页面，对配置文件内容进行修改，点击保存即可编辑成功。如果文件没有保存，直接返回，也会进行相关提示。页面中还可以看到文件的名称、路径和修改时间



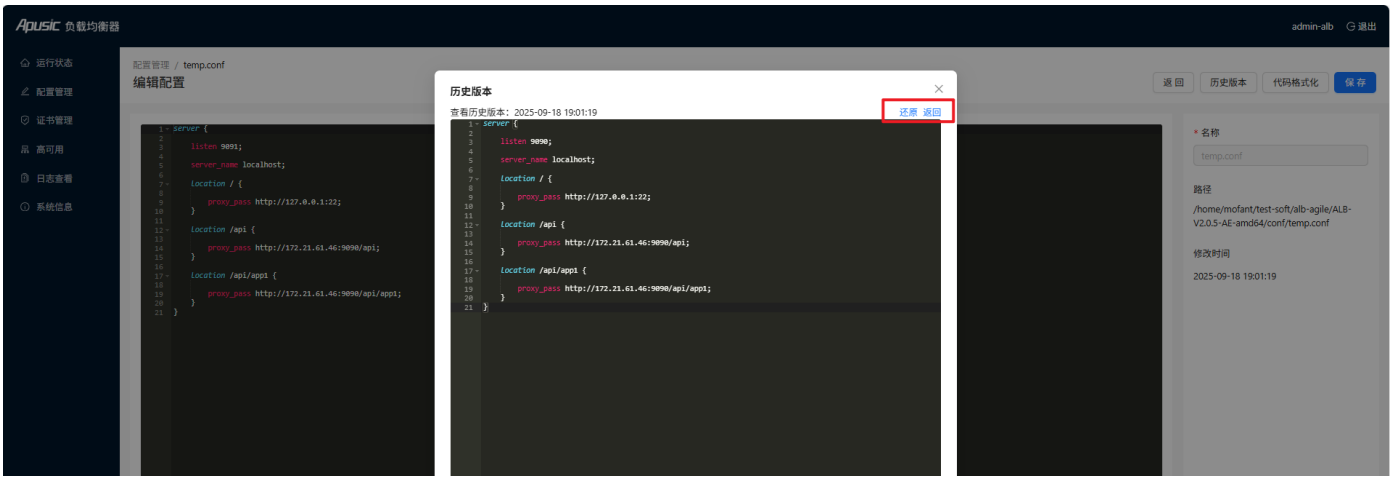
- 重命名：选择对应文件点击“重命名”按钮，进入如下页面，即可对配置文件或文件夹进行重命名操作，重命名也要遵循文件名的命名规则



- 历史版本： 可以查看文件修改的历史版本



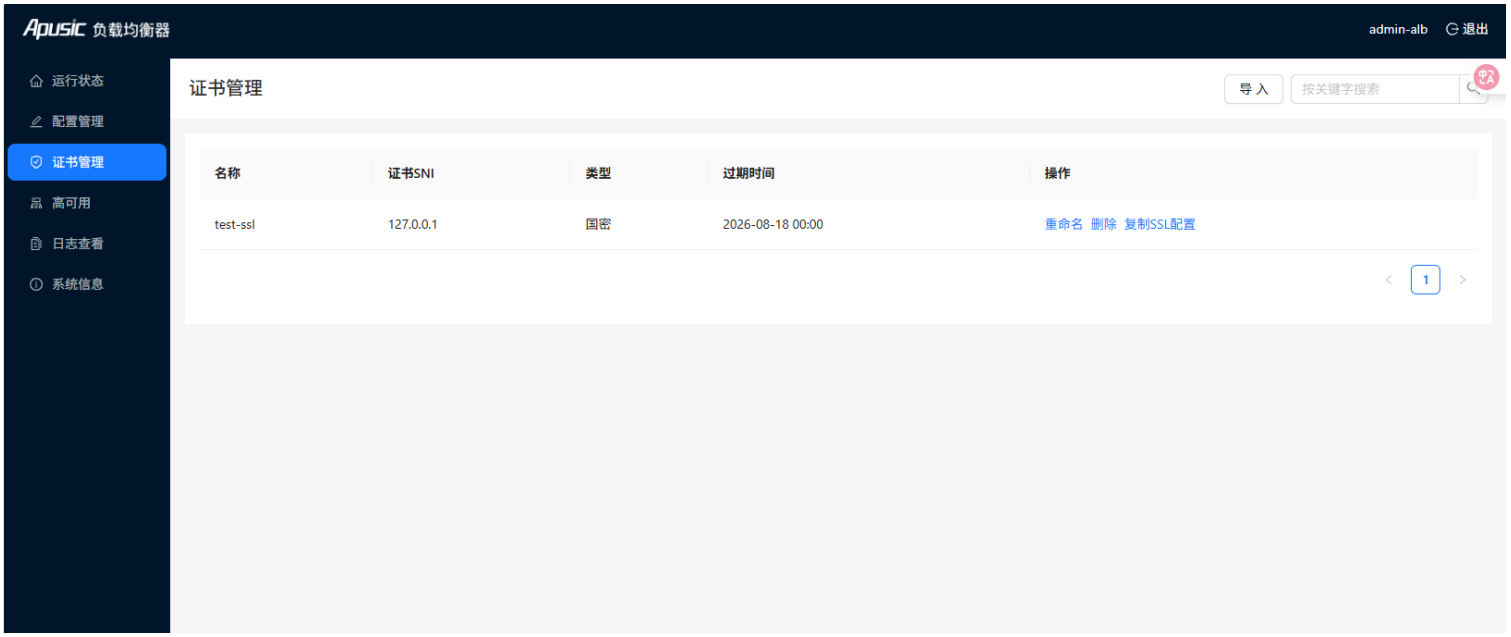
在历史版本中，可以点击查看按钮，展示之前版本的配置内容



点击还原按钮后，旧版配置将回退到编辑区，待确认无误后，点击**保存**按钮，即可还原配置。

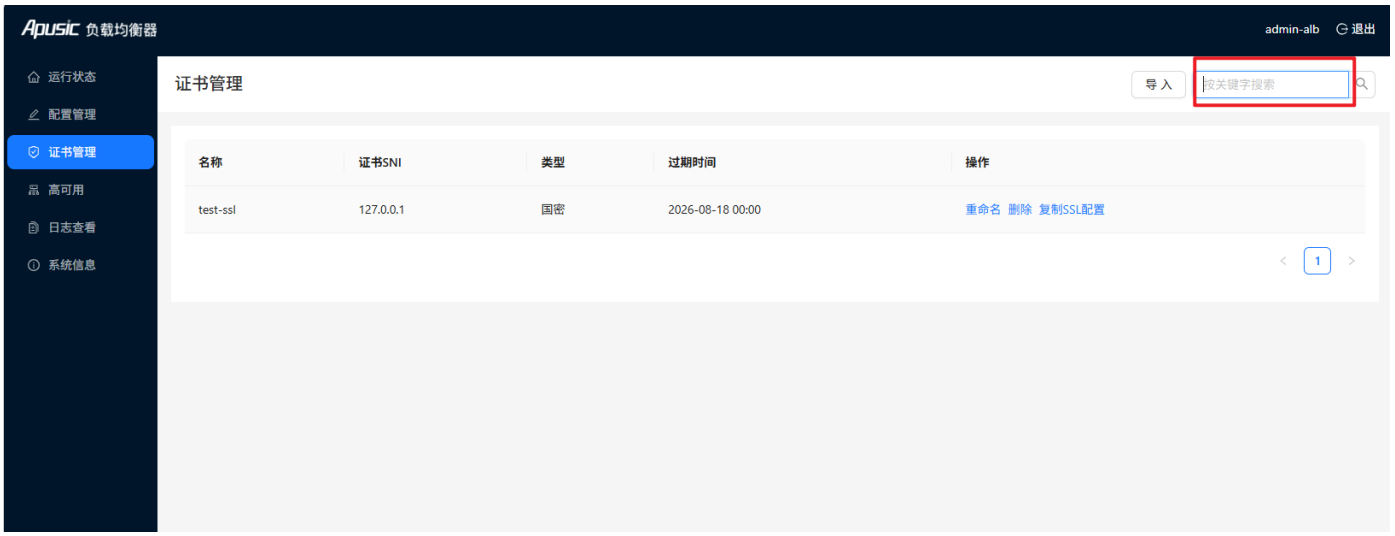
7.5.3 证书管理

证书管理页面是ALB上传和管理SSL证书的入口，对SSL证书文件进行管理

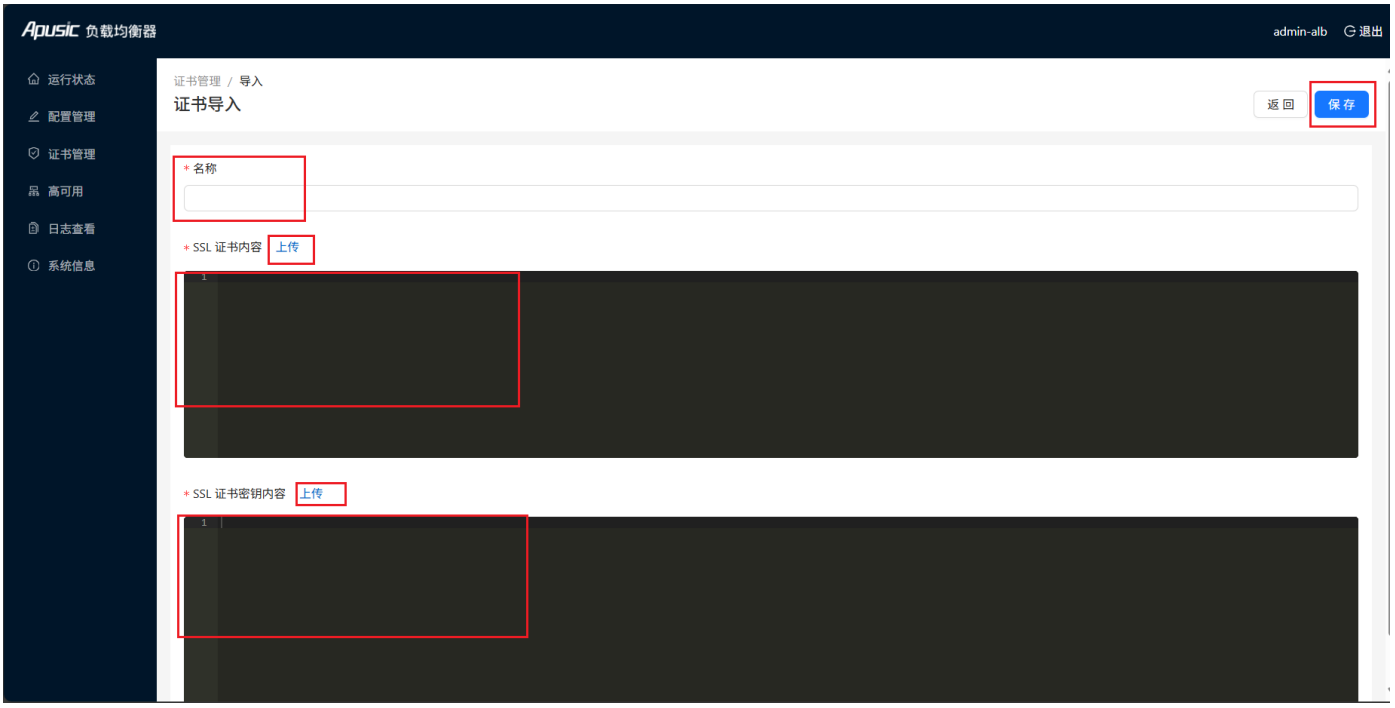


页面信息说明：

- 名称：证书的名字，也是证书的唯一标识
- 证书SNI：标识服务器的域名或 IP 地址
- 类型：证书的类型，包括标准的SSL证书或国密证书
- 过期时间：证书的过期日期
- 查看证书列表：支持按条件进行分页获取证书列表，可以在搜索框中输入证书名称进行过滤，支持模糊匹配

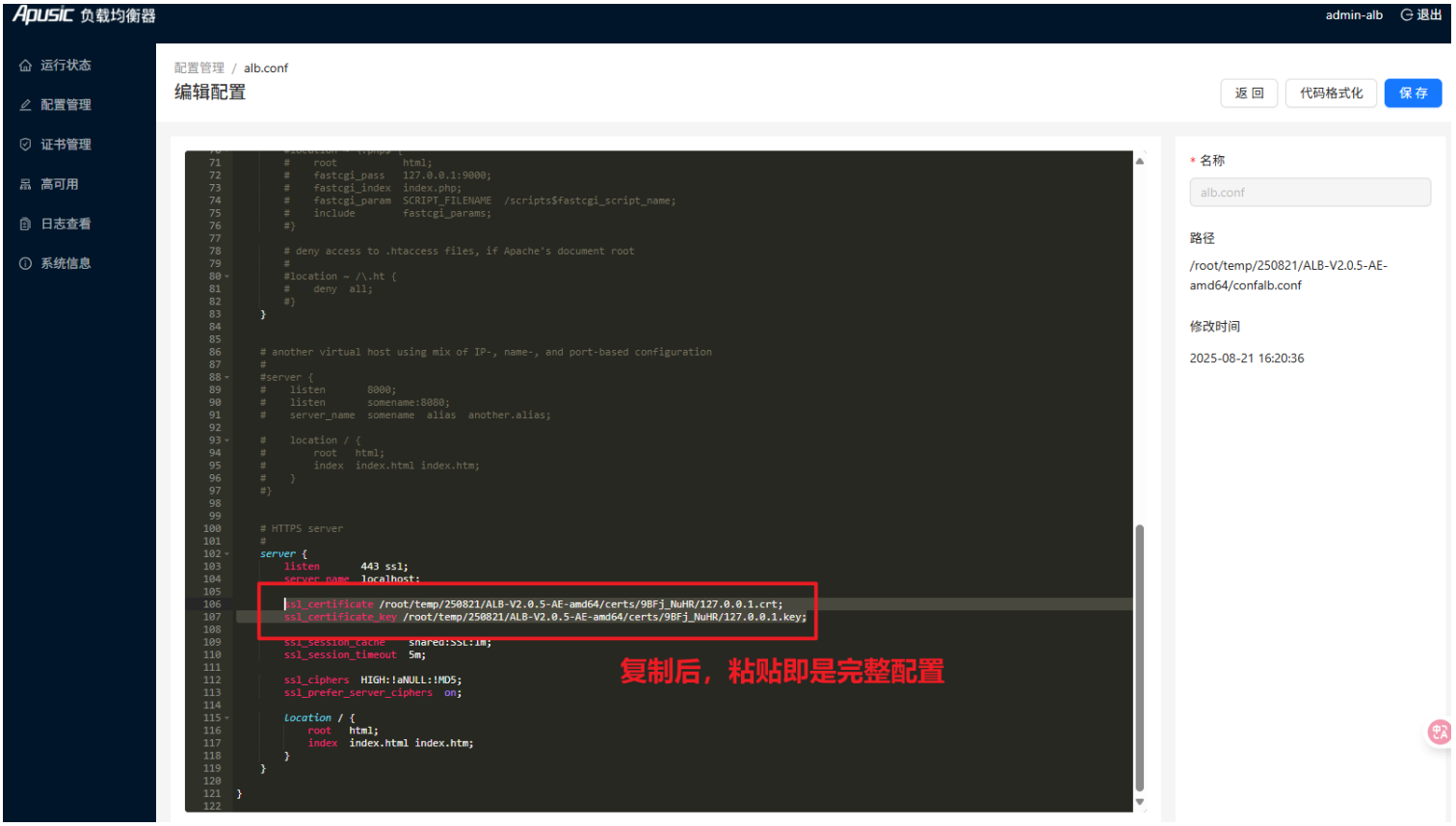


- 导入：点击“导入”按钮，进入如下页面，即可对SSL证书和密钥文件进行操作，支持手动编辑，也支持本地上传



7.5.3.1 证书使用

在已经上传的证书中，点击复制SSL配置，即可在配置中直接粘贴使用。



7.5.3.2 国密证书导入

国密证书需要分两次导入以下证书内容：

- 1、服务端加密证书文件和服务端加密证书私钥文件
- 2、服务端签名证书（**国密**必须）和服务端签名证书私钥文件（**国密**必须）

即在管控台针对国密证书分两次上传后使用，如下图所示：

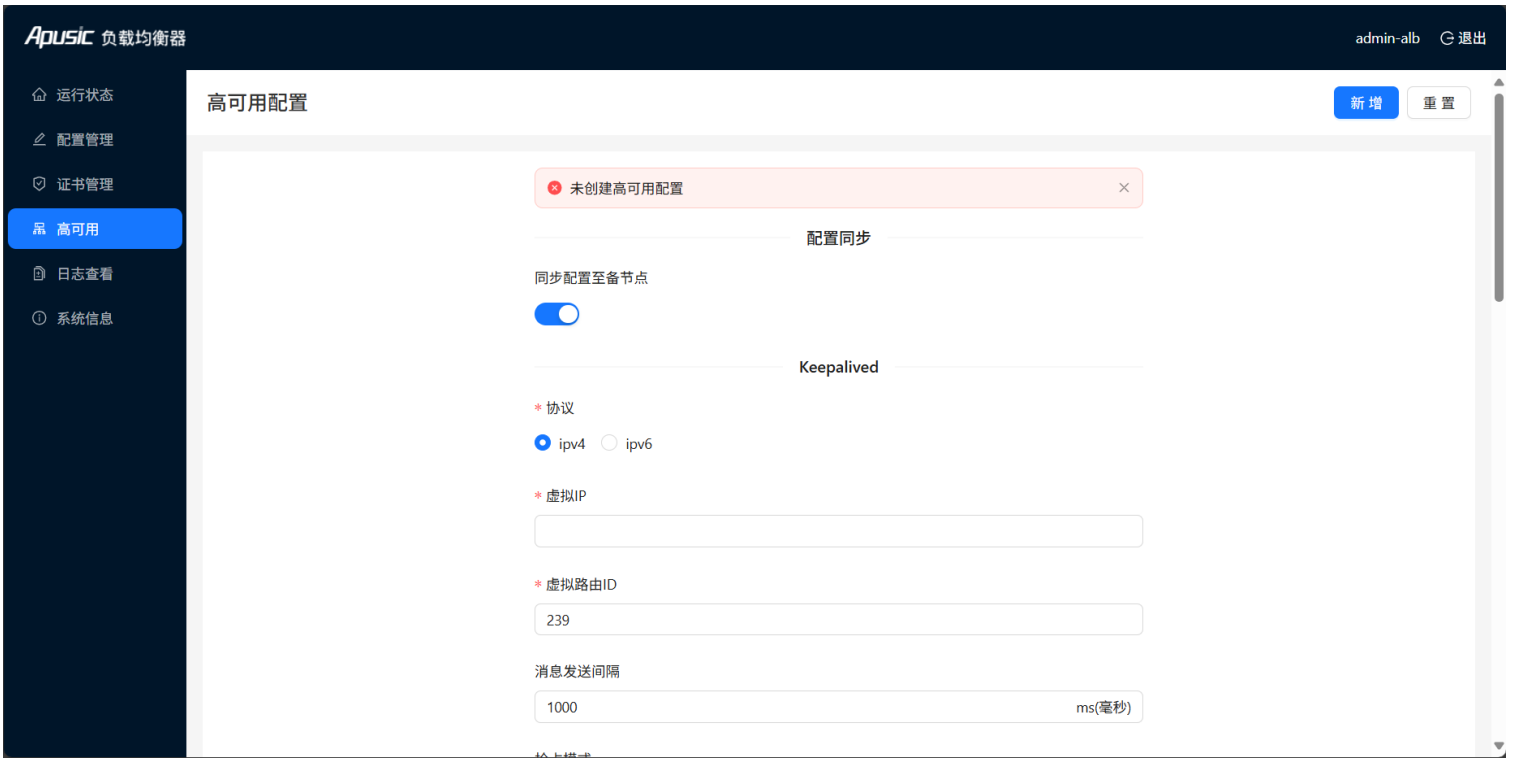
上传完毕后，在对应的server配置块中，需要两次复制SSL配置粘贴至配置项中，如下图所示：

7.5.4 高可用

高可用页面是查看和配置ALB高可用功能的入口，只有主节点才能配置高可用，备节点只能查看，即当你在当前节点配置高可用的时候，配置页面中主节点的信息默认只能选择当前节点IP，但是如果主节点的优先级低于其他备节点，当配置完成，主节点会自动切换，优先级最高的节点变成主节点

注：使用高可用功能管控台必须**使用root用户启动。**

如果没有配置高可用，则页面如下所示：



页面信息说明：

- 配置同步：表示ALB配置文件的变更是否也同步到备节点，即配置管理中对文件的所有操作是否同步到备节点
- Keepalived：用来实现VIP（虚拟IP）的故障转移和负载均衡，如下是各项的具体含义：
    - 协议：表示虚拟IP在IPv4或IPv6网络中运行，同时也决定了虚拟IP的类型
    - 虚拟IP：一个动态IP，主备节点集群的入口IP，用于对外提供服务，它可以在主备份节点之间动态切换
    - 虚拟路由ID：用来标识高可用集群中的主备节点，在同一集群中的节点必须是相同的ID
    - 消息发送间隔：表示主节点向备份节点发送心跳消息的时间间隔
    - 抢占模式：表示是否允许高优先级的节点在恢复后重新接管虚拟IP
- 健康检查：用来检查ALB服务的运行状态是否正常
    - 检查类型：可以选择检查时的通讯协议，支持http或https
    - 检查端口：表示节点的健康检查服务的端口，取值范围：1~65535，默认8080
    - 检查URL：表示节点的健康检查服务的URL，不能以/开头，默认node_status，建议配置为业务入口的URL地址。
    - 健康状态码：表示健康检查返回这种状态码的时候，则认为是健康的状态，支持手动增加和删除，取值范围：100~599，默认 200、404
    - 检查间隔：表示两次健康检查之间的时间间隔，默认为1秒，支持手动调整

- 重试次数：表示健康检查失败时，允许重新尝试检查的最大次数，默认为3，防止网络抖动导致误判，提高容错率，支持手动修改
- 重试间隔：表示健康检查失败时，每次重试的时间间隔，默认为2秒，支持手动修改
- 检查到alb不健康，是否重启alb：当节点被标记为不健康的时候，是否重启ALB

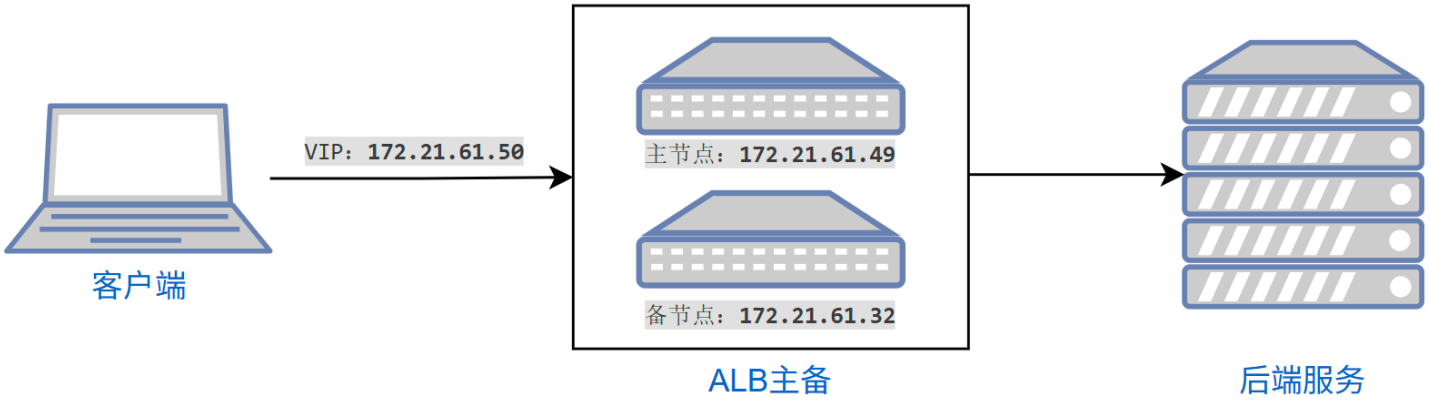
- 主/备节点
  - IP：部署ALB节点的IP地址，部署高可用配置的节点默认是主节点（配置完成后会自动根据优先级发生转移），主节点IP默认是当前节点IP，直接选择即可，备节点需要手动输入
  - 网卡：IP地址所属的网卡，不需要手动填写，当输入IP之后会自动获取，然后选择即可，备节点需要点击“测试联调”自动填写
  - 管控台端口：部署ALB节点上的ALB管控台服务端口，默认8887
  - 优先级：主备节点的优先级，用来决定哪个节点晋升为主节点，数字越大，优先级越高，最大值为255
  - AuthKey：主备节点进行通信时的身份验证，防止未经授权的节点加入集群，同时确保节点间通信的安全性，AuthKey值在节点配置文件：`ALB安装目录/console/conf/config.yaml`的alb：`aha-auth-key`字段中配置。
  - 测试联调：当备节点配置完成时，点击“测试联调”会测试备节点是否可达，并获取IP对应网卡信息

- 新增：在页面中输入相关配置信息后，点击“新增”按钮，即可创建高可用配置（前提是ALB产品是正常启动状态），提示保存成功即表示创建高可用配置成功，创建配置成功后默认会开启高可用功能。当配置成功之后在页面最上边会增加一栏状态信息，如下所示以及信息说明：
  - 当前节点角色：表示当前节点是主节点还是备节点
  - VIP绑定状态：表示VIP（虚拟IP）是否成功绑定，如果高可用服务启动正常，但是VIP绑定状态显示未绑定，请刷新几次页面即可
  - 高可用状态：表示是否开启高可用，可以手动选择开启或关闭

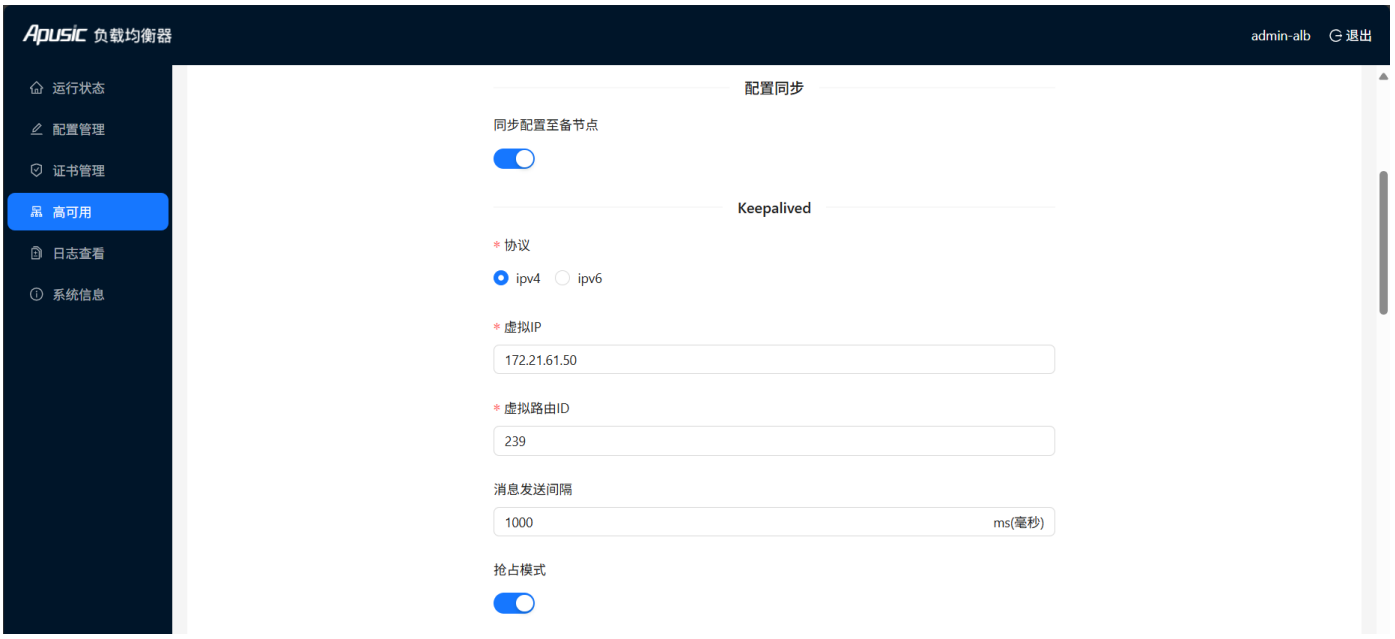
- 修改：点击“修改”按钮可以对页面中高可用配置进行修改，只有当创建高可用配置成功之后才会出现“修改”按钮
- 重置：点击“重置”按钮可以重置高可用配置，相当于直接删除高可用配置、关闭高可用功能

7.5.4.1 创建主备高可用

如下图，创建主备ALB高可用环境：



- 高可用集群信息
  - 主节点： 172.21.61.49
  - 备节点： 172.21.61.32
  - 虚拟IP： 172.21.61.50
- Keepalived配置


Apusic 负载均衡器
admin-alb
退出
运行状态
配置管理
证书管理
高可用
日志查看
系统信息
配置同步
同步配置至备节点
Keepalived
* 协议
ipv4
ipv6
* 虚拟IP
172.21.61.50
* 虚拟路由ID
239
消息发送间隔
1000
ms(毫秒)
抢占模式

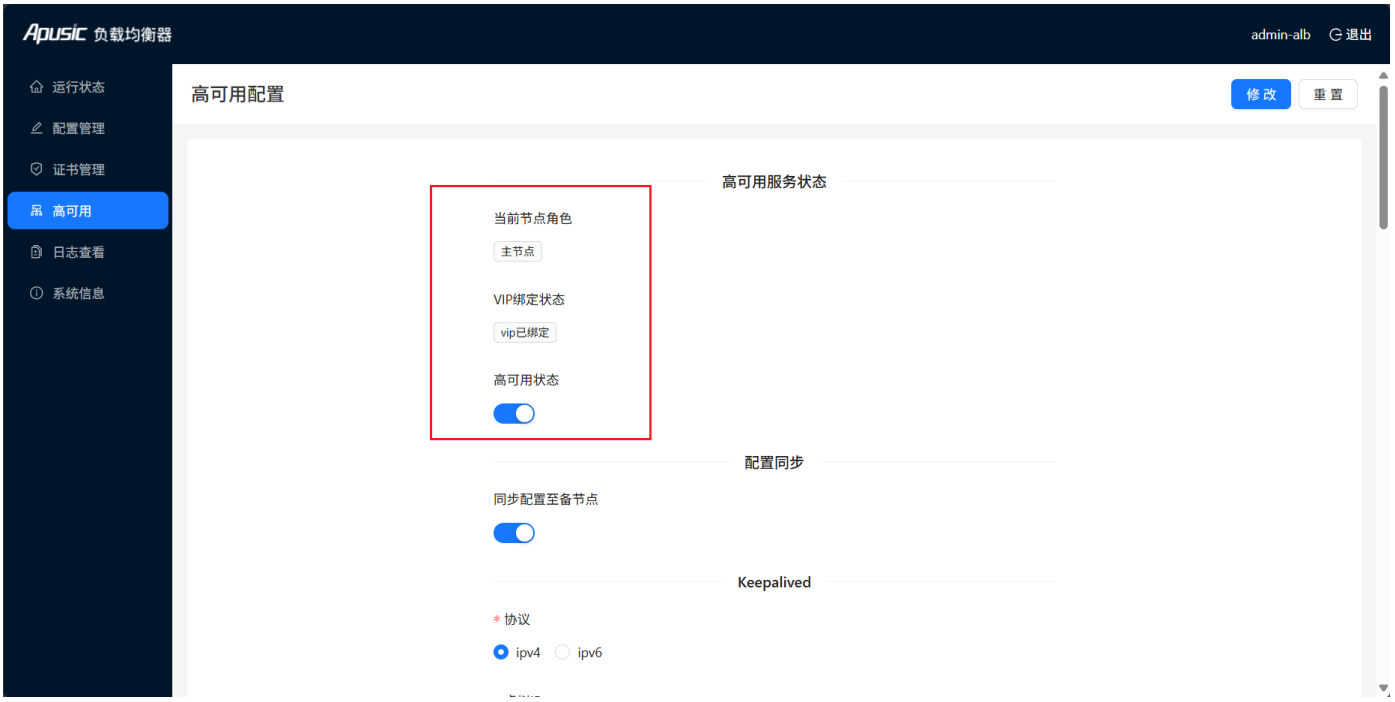
- 主节点配置


Apusic 负载均衡器
admin-alb
退出
运行状态
配置管理
证书管理
高可用
日志查看
系统信息
主节点
* IP
172.21.61.49
* 网卡
eth0
* 管控台端口
8887
* 优先级
255

- 备节点配置

点击“创建”按钮，配置成功之后，主备节点展示如下：

- 主节点



- 备节点

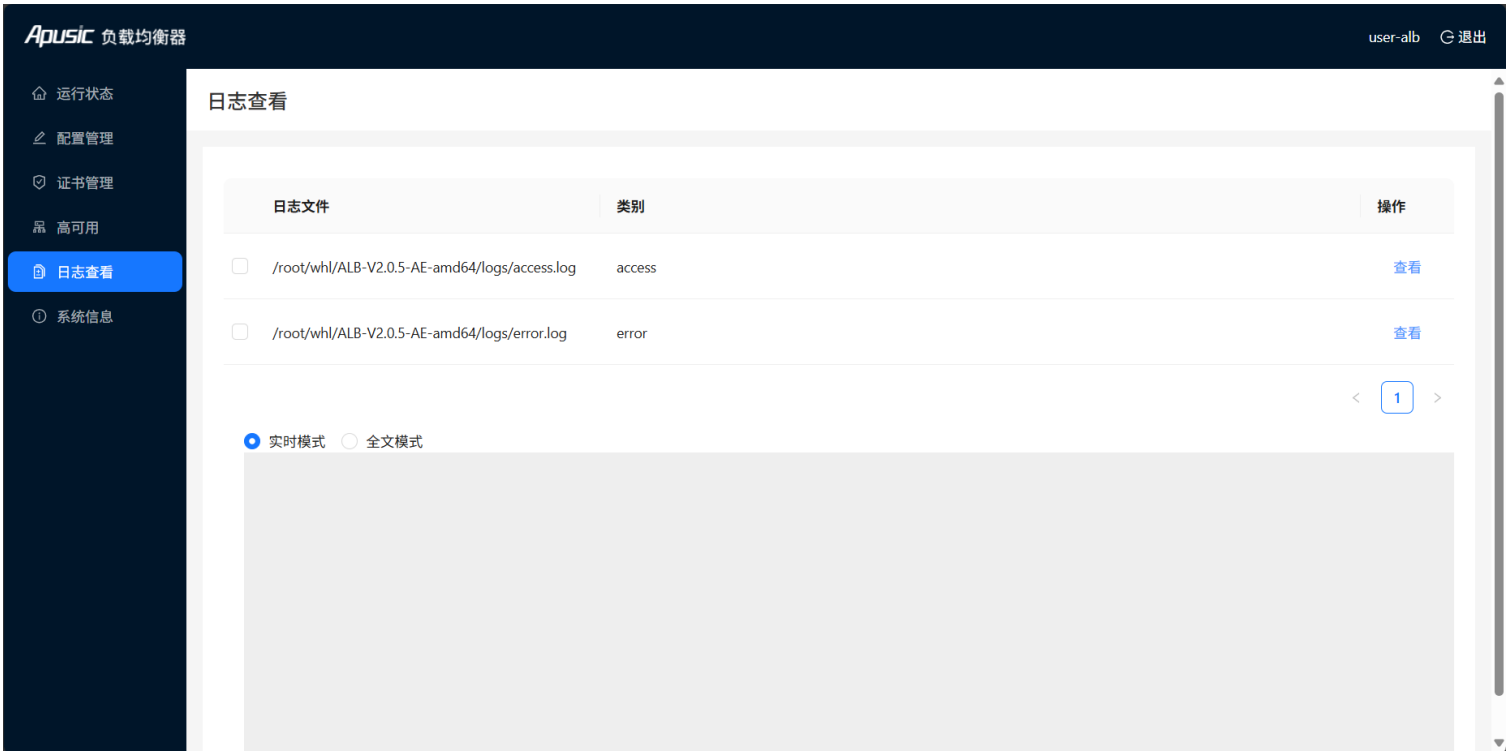
当我把主节点`172.21.61.49`的优先级调整为10，点击“修改”按钮后，主节点变成备节点，备节点`172.21.61.32`变成主节点

- 调整主节点`172.21.61.49`优先级为10

- 主备发生切换

7.5.5 日志查看

日志查看页面是查看ALB访问日志和错误日志的入口



页面信息说明：

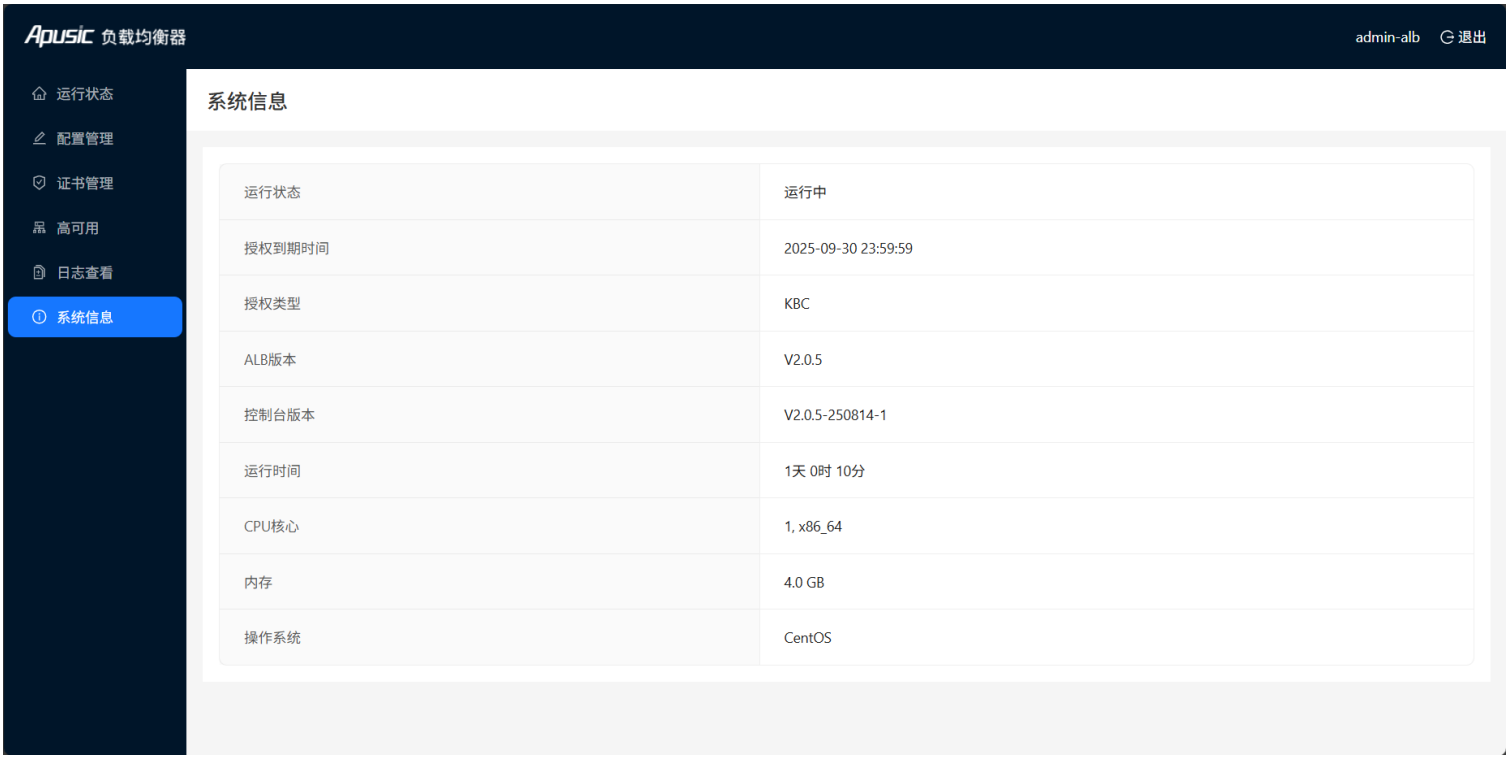
- 日志列表：ALB配置中所有日志文件的集合
  - 日志文件：日志文件的名称

- 类别：日志文件的类型，一种是访问日志，一种是错误日志。访问日志记录所有客户端请求的详细信息，错误日志记录运行过程中发生的错误或警告信息

- 查看：点击查看按钮，可以查看日志文件内容，默认是实时模式。
- 实时模式：实时获取日志选择日志的最新数据。
- 全文模式：获取日志选择日志的完整数据，但最多获取10MB大小日志内容。

7.5.6 系统信息

系统信息页面可以查看ALB运行信息、版本信息和所部署的节点信息

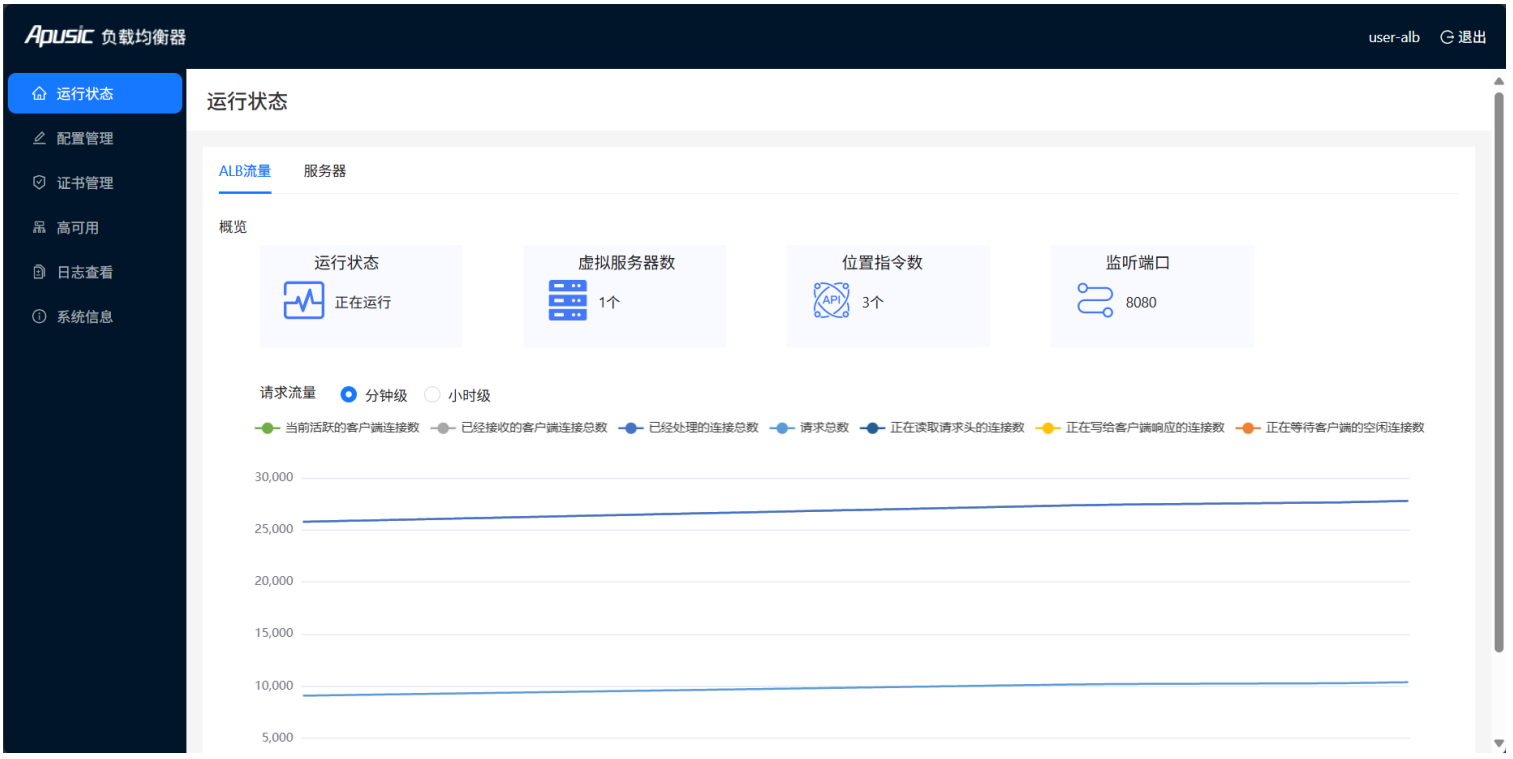


页面信息说明：

- 运行状态：ALB是否正在运行
- 许可证：许可证的过期时间
- 类型：许可证的类型
- ALB版本：ALB版本
- 管控台版本：ALB管控台版本
- 运行时间：ALB运行时间
- CPU核心：ALB管控台服务所处节点的CPU核心数和架构
- 内存大小：ALB管控台服务所处节点的内存大小
- 操作系统：ALB管控台服务所处节点的操作系统标识

7.6 普通用户操作

普通用户登录成功之后，默认页面如下：



普通用户和系统管理员看到的信息是相同的，唯一的区别就是管理员具有操作权限，而普通用户仅有查看权限

7.7 开启/禁用强制用户密码修改

ALB管控台用户第一次登录密码必须要修改，该功能开启/关闭方式在`ALB安装目录/console/conf/config.yaml`文件：

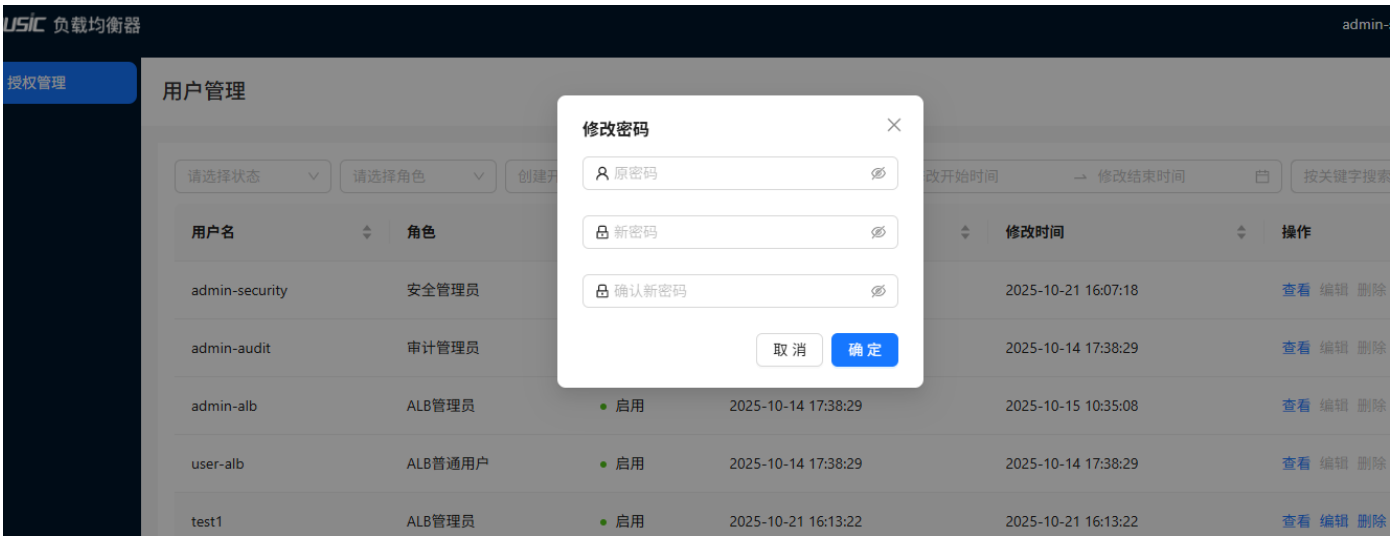
```
system:
  # 是否强制首次登录修改密码，默认强制
  change-pwd-at-first: true
```

第一次登录会提示修改密码，只有修改密码后才能登录。

7.8 用户密码修改

用户密码修改方式有两种：

- 安全管理员登录管理后端，编辑用户信息，可以直接修改密码。
- 用户自己登录系统后，点击自己用户名弹框修改。



7.9 用户密码重置

用户密码重置可以通过登录到ALB安装目录下，执行:

`./alb-standard-V2.0.5-console --reinit 用户名` 其中用户名为需要重置密码的默认用户，如下重置admin-alb用户的密码

```
cd console
./alb-standard-V2.0.5-console --reinit admin-alb
[alb-standard-console]2025/10/09 - 17:16:20.467    info
/home/mofant/work/alb-agile/alb-lightweight-console-
be/initialize/sqlite.go:52  register table success
[alb-standard-console]2025/10/09 - 17:16:20.489    info
/home/mofant/work/alb-agile/alb-lightweight-console-
be/initialize/sqlite.go:138 用户密码重置成功，请使用新密码登录
{"username": "admin-alb"}
```

7.10 重置管控台

若需要重置ALB管控台，可以通过以下操作：

- 使用终端登录系统，切换至alb安装目录
- 停止ALB
- 删除 `console/data/alb.db` 文件
- 启动ALB
- 重新登录

注: 若不想停止ALB服务，可以单独找到进程名为： `alb-standard-V2.0.5-console` 的进程，并kill -9进程ID，然后再删除 `ALB安装目录/console/data/alb.db` 文件，然后再切换至 `console` 目录，然后使用 `nohup ./alb-standard-V2.0.5-console > console.log 2>&1 &` 启动ALB服务

8 用户场景配置案例

8.1 概述

ALB 不仅是一个 Web 服务器，更是现代应用架构中的"流量中枢"。它在高并发、安全防护、服务治理等方面发挥着关键作用。本文通过多个贴近真实业务的场景，详细说明企业在不同发展阶段如何借助ALB解决实际问题，并附上可直接复用的配置样例与效果分析。

8.2 场景一：前端单页应用托管与路由支持/静态资源发布

8.2.1 1. 背景

某公司使用 React 开发管理后台，构建后输出静态资源。初期通过 Node.js Express 提供静态服务，部署在单台云服务器上。

8.2.2 2. 遇到的问题与待解决的问题

- 用户直接访问 /settings 等非根路径时返回 404（因服务器无对应物理文件）；
- 静态资源未缓存，每次刷新都重新下载，带宽成本高、加载慢；
- Express 在并发 >500 时 CPU 占用飙升，响应延迟超过 2 秒。

待解决问题：实现 SPA 路由兼容、静态资源高效分发、降低服务器负载。

8.2.3 3. ALB 功能与配置与问题匹配

- `try_files` 指令：实现前端路由 fallback 到 index.html；
- `expires` 与 `add_header`：设置长期缓存策略；
- ALB 高性能事件驱动模型：高效处理静态文件请求。

8.2.4 4. 使用 ALB 解决该场景下的配置与详解

```
server {
    listen 80;
    server_name test.com;

    root /var/www/admin/dist;
    index index.html;

    # SPA 路由支持：若文件不存在，回退到 index.html
    location / {
```

```
        try_files $uri $uri/ /index.html;
    }

    # 静态资源设置 1 年缓存（immutable）
    location ~* \.(js|css|png|jpg|svg|woff2)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }
}
```

- `try_files $uri $uri/ /index.html`：优先找物理文件，找不到则交由前端路由处理；
- `immutable` 缓存：浏览器在缓存有效期内绝不发起条件请求，极大提升复访速度。

8.2.5 5. 效果，前后对比

指标	改造前（Express）	改造后（ALB）
首屏加载时间	2.1s	0.6s
静态资源 QPS	300	5000+
CPU 占用（500 并发）	85%	12%
路由 404 错误	频繁发生	完全消除

8.3 场景二：微服务 API 统一入口与负载均衡

8.3.1 1. 背景

某电商平台完成微服务拆分，拥有 user、order、payment 三个核心服务，分别部署在不同主机或容器中。

8.3.2 2. 遇到的问题与待解决的问题

- 客户端需维护多个服务地址，调用复杂；
- 某实例宕机时无法自动剔除，导致请求失败；
- 缺乏统一日志，排查问题困难。

待解决问题：统一 API 入口、自动负载均衡、集中可观测性。

8.3.3 3. ALB 功能与配置与问题匹配

- `upstream` + `server`：定义后端集群；
- `proxy_pass`：反向代理到对应服务；

- `max_fails/fail_timeout`：健康检查机制；
- `access_log`：统一记录请求日志。

8.3.4 4. 使用 ALB 解决该场景下的配置与详解

```
upstream user_backend {
    server 10.0.1.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.1.11:8080 max_fails=3 fail_timeout=30s;
}

upstream order_backend {
    server 10.0.1.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.1.11:8080 max_fails=3 fail_timeout=30s;
}

upstream payment_backend {
    server 10.0.1.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.1.11:8080 max_fails=3 fail_timeout=30s;
}

server {
    listen 80;
    server_name api.shop.com;

    access_log /var/log/alb/api.log combined;

    location /api/users/ {
        proxy_pass http://user_backend/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /api/orders/ {
        proxy_pass http://order_backend/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
```

```
    }

    location /api/payment/ {
        proxy_pass http://payment_backend/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

- `max_fails=3`：连续 3 次失败后，30 秒内不再转发请求到该节点；
- `X-Real-IP`：透传真实客户端 IP，便于后端审计。

8.3.5 5. 效果，前后对比

- 客户端调用地址从 3 个减少为 1 个；
- 服务实例故障时，错误率从 15% 降至 0.2%；
- 运维可通过单一日志文件追踪所有 API 调用链。

8.4 场景三：全站 HTTPS 强制跳转与 SSL 终止

8.4.1 1. 背景

某在线教育平台因 GDPR 合规要求，必须启用 HTTPS。当前同时开放 HTTP 和 HTTPS，证书由 Java 应用管理。

8.4.2 2. 遇到的问题与待解决的问题

- 用户可能通过 HTTP 访问，存在数据泄露风险；
- 后端应用处理 SSL 加解密，CPU 开销大；
- 证书更新需重启应用，影响可用性。

待解决问题：强制 HTTPS、SSL 边缘终止、零停机证书更新。

8.4.3 3. ALB 功能与配置与问题匹配

- return 301：HTTP 强制跳转 HTTPS；
- ssl_certificate：加载 TLS 证书；
- 支持热重载（./bin/reload-alb.sh），无需中断连接。

8.4.4 4. 使用 ALB 解决该场景下的配置与详解

```
server {
    listen 80;
```

```
    server_name learn.edu.com;
    return 301 https://$host$request_uri;   # 强制跳转
}

upstream backend {
    server 10.0.1.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.1.11:8080 max_fails=3 fail_timeout=30s;
}
server {
    listen 443 ssl http2;
    server_name learn.edu.com;

    ssl_certificate /opt/ALB-2.0.2-SE-
amd64/conf/learn.edu.com/fullchain.pem;
    ssl_certificate_key /opt/ALB-2.0.2-SE-
amd64/conf/learn.edu.com/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;   # 使用更安全的加密算法

    location / {
        proxy_pass http://backend/;
        proxy_set_header X-Forwarded-Proto $scheme;   # 告知后端原始协议
    }
}
```

8.4.5 5. 效果，前后对比

- HTTP 流量占比从 40% 降至 0%；
- 后端 CPU 负载下降 18%；
- 证书更新只需替换文件并 reload，服务零中断。

8.5 场景四：API 接口防刷与速率限制

8.5.1 1. 背景

某社交平台开放用户信息查询接口 /api/profile/{id}，近期遭遇爬虫高频请求。

8.5.2 2. 遇到的问题与待解决的问题

- 数据库 QPS 从 2000 暴增至 15000，响应超时；
- 正常用户请求被阻塞；
- 临时封 IP 影响合法开发者。

待解决问题：在不修改业务代码前提下，快速实施限流。

8.5.3 3. ALB 功能与配置与问题匹配

- `limit_req_zone`：基于 IP 定义速率限制区域；
- `limit_req`：在 location 中应用限流策略；
- `burst + nodelay`：允许突发流量平滑处理。

8.5.4 4. 使用 ALB 解决该场景下的配置与详解

```
http {
    limit_req_zone $binary_remote_addr zone=profile_limit:10m
rate=5r/s;

    server {
        location = /api/profile {
            limit_req zone=profile_limit burst=10 nodelay;
            limit_req_status 429;
            proxy_pass http://user_service;
        }
    }
}
```

- `rate=5r/s`：每秒最多 5 次请求；
- `burst=10`：允许瞬时突发 10 次，避免正常用户误伤；
- 返回 429 状态码，符合 RESTful 规范。

8.5.5 5. 效果，前后对比

- 恶意请求拦截率 99.6%；
- 数据库 QPS 回落至 2200；
- 正常用户无感知，开发者可依据 429 实现重试逻辑。

8.6 场景五：动静分离与动态内容缓存

8.6.1 1. 背景

视频平台课程详情页包含动态数据（学习进度、评论），但被频繁访问。

8.6.2 2. 遇到的问题与待解决的问题

- 每次请求都穿透到 Java 后端，即使内容未变；
- Redis 缓存策略复杂，命中率低；
- 高峰期数据库连接池耗尽。

待解决问题：对“相对静态”的动态页面做边缘缓存。

8.6.3 3. ALB 功能与配置与问题匹配

- proxy_cache_path：定义缓存存储；
- proxy_cache_valid：设置缓存有效期；
- proxy_cache_use_stale：故障时返回过期缓存，保障可用性。

8.6.4 4. 使用 ALB 解决该场景下的配置与详解

```
proxy_cache_path /cache/alb levels=1:2 keys_zone=course_cache:20m
inactive=10m;

location = /course/detail {
    proxy_cache course_cache;
    proxy_cache_valid 200 10m;
    proxy_cache_use_stale error timeout updating;
    proxy_pass http://course_service;
    add_header X-Cache $upstream_cache_status;
}
```

8.6.5 5. 效果，前后对比

- 课程页缓存命中率 72%；
- 后端 QPS 下降 65%；
- 故障期间用户仍可看到“稍旧但可用”的页面。

8.7 场景六：WebSocket 长连接代理

8.7.1 1. 背景

官网集成在线客服系统，使用 WebSocket 实现实时聊天。

8.7.2 2. 遇到的问题与待解决的问题

- 连接建立后 60 秒自动断开；
- 消息丢失严重，用户体验差；
- 无法与 HTTP 共用 443 端口。

待解决问题：稳定代理 WebSocket 长连接。

8.7.3 3. ALB 功能与配置与问题匹配

- `Upgrade` 和 `Connection` 头透传；
- `proxy_read_timeout`：延长读超时时间。

8.7.4 4. 使用 ALB 解决该场景下的配置与详解

```
location /ws/chat/ {
    proxy_pass http://chat_service;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_read_timeout 86400s;
}
```

8.7.5 5. 效果，前后对比

- WebSocket 连接可稳定维持 24 小时；
- 消息到达率从 82% 提升至 99.9%；
- 与 HTTPS 共享 443 端口，简化网络策略。

8.8 场景七：防止恶意扫描与目录遍历攻击

8.8.1 1. 背景

某政府信息公开网站部署在公网，近期被自动化工具频繁扫描 `/admin/`、`/.git/`、`/backup.zip` 等敏感路径，尝试获取内部配置或源码。

8.8.2 2. 遇到的问题与待解决的问题

- 攻击请求占用服务器资源；
- 日志中大量 404 请求干扰正常监控；
- 存在因误暴露备份文件导致数据泄露的风险。

待解决问题：在入口层拦截高危路径访问，无需修改应用代码。

8.8.3 3. ALB 功能与配置与问题匹配

- `location` 精确匹配 + `deny all`：禁止访问敏感目录；
- 正则匹配隐藏文件（如 `.env`、`.git`）；
- 返回 444（ALB 特有状态码，直接关闭连接，不响应）。

8.8.4 4. 使用 ALB 解决该场景下的配置与详解

```
server {
    listen 80;
    server_name www.govinfo.gov.cn;

    # 禁止访问隐藏文件
    location ~ /\. {
        deny all;
        return 444;
    }

    # 禁止访问常见敏感目录
    location ~* ^/(admin|phpmyadmin|wp-admin|backup|config)/ {
        deny all;
        return 444;
    }

    # 禁止下载特定扩展名文件
    location ~* \.(sql|bak|tar\.gz|zip)$ {
        deny all;
        return 444;
    }

    location / {
        proxy_pass http://backend;
    }
}
```

- `return 444`：不返回任何响应，直接断开 TCP 连接，降低攻击者探测效率；
- 正则 ~* 表示不区分大小写，覆盖更多变种路径。

8.8.5 5. 效果，前后对比

- 每日恶意请求从 12,000+ 降至 <50；
- 安全审计报告中"信息泄露风险"项清零；
- 服务器 CPU 在扫描高峰期下降 30%。

8.9 场景八：基于 Cookie 的灰度发布（A/B 测试）

8.9.1 1. 背景

电商平台计划上线新版商品详情页，希望先对 10% 用户开放新版本，验证转化率后再全量。

8.9.2 2. 遇到的问题与待解决的问题

- 后端无灰度能力，需网关层分流；
- 要求同一用户始终看到同一版本（会话一致性）；
- 不能依赖 IP（多用户共享出口）。

待解决问题：基于用户标识（如 Cookie）实现稳定灰度路由。

8.9.3 3. ALB 功能与配置与问题匹配

- map 指令：根据 $cookie_version 值决定后端；
- 自定义变量控制流量比例；
- sticky 会话保持（通过 Cookie 实现）。

8.9.4 4. 使用 ALB 解决该场景下的配置与详解

```
# 若用户无 version cookie，则随机分配（10% 概率设为 new）
map $cookie_version $backend_group {
    default "old";
    ""      "auto";  # 未设置时走自动分配
    "new"   "new";
}

upstream old_backend {
    server 10.0.1.10:8080;
}
```

```
upstream new_backend {
    server 10.0.1.20:8080;
}

server {
    listen 80;
    server_name shop.com;

    location /product/ {
        # 未分配版本的用户：10% 概率打标为 new
        if ($backend_group = "auto") {
            set $rand "";
            set_by_lua_block $rand { return math.random() < 0.1 and
"new" or "old" }
            set $backend_group $rand;
            add_header Set-Cookie "version=$rand; Path=/; Max-
Age=2592000";
        }

        if ($backend_group = "new") {
            proxy_pass http://new_backend;
        }
        if ($backend_group = "old") {
            proxy_pass http://old_backend;
        }
    }
}
```

8.9.5 5. 效果，前后对比

- 新老版本用户隔离清晰，无交叉污染；
- A/B 测试周期从 2 周缩短至 3 天；
- 无需改造业务系统，灰度策略完全由 ALB 控制。

8.10 场景九：大文件上传超时与分片支持优化

8.10.1 1. 背景

在线教育平台允许教师上传课件（PPT、视频），部分文件超过 500MB。

8.10.2 2. 遇到的问题与待解决的问题

- 默认 60 秒超时导致大文件上传失败；
- 用户网络波动时需重新上传整个文件；
- 后端 Java 应用内存溢出（OOM）。

待解决问题：延长上传超时、支持断点续传、减轻后端压力。

8.10.3 3. ALB 功能与配置与问题匹配

- client_max_body_size：允许大请求体；
- client_body_timeout / proxy_send_timeout：延长超时；
- （可选）结合 alb-upload-module 或前端分片实现断点续传。

8.10.4 4. 使用 ALB 解决该场景下的配置与详解

```
server {
    listen 80;
    server_name teach.edu.com;

    # 允许最大 2GB 文件
    client_max_body_size 2G;

    # 上传相关超时设为 1 小时
    client_body_timeout 3600s;
    proxy_send_timeout 3600s;
    proxy_read_timeout 3600s;

    location /api/upload {
        proxy_pass http://file_service;
        proxy_set_header X-Real-IP $remote_addr;
        # 临时文件存储路径（避免内存缓存大文件）
        client_body_temp_path /var/tmp/alb_upload;
    }
}
```

8.10.5 5. 效果，前后对比

- 500MB+ 文件上传成功率从 40% 提升至 99%；
- 后端 OOM 错误归零；
- 用户可在弱网环境下完成上传。

8.11 场景十：静态资源 Gzip 压缩加速

8.11.1 1. 背景

企业官网包含大量 JS/CSS 文件，海外用户反馈加载缓慢。

8.11.2 2. 遇到的问题与待解决的问题

- 未启用压缩，文本资源体积大；
- 后端动态压缩消耗 CPU；
- 移动端流量成本高。

待解决问题：在边缘层对文本资源进行高效 Gzip 压缩。

8.11.3 3. ALB 功能与配置与问题匹配

- `gzip on`：启用压缩；
- `gzip_types`：指定压缩 MIME 类型；
- `gzip_vary on`：通知代理缓存不同版本。

8.11.4 4. 使用 ALB 解决该场景下的配置与详解

```
http {
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;   # 小于 1KB 不压缩
    gzip_comp_level 6;      # 平衡 CPU 与压缩率
    gzip_types
        text/plain
        text/css
        application/json
        application/javascript
        text/xml
        application/xml
        image/svg+xml;

    server {
```

```
        listen 80;
        server_name www.company.com;
        root /var/www/html;
    }
}
```

8.11.5 5. 效果，前后对比

资源类型	原始大小	Gzip 后	压缩率
main.js	850 KB	210 KB	75% ↓
style.css	320 KB	85 KB	73% ↓

- 页面整体加载时间减少 40%；
- 用户跳出率下降 18%。

9 ALB功能模块使用说明

9.1 license使用说明

ALB标准版支持本地授权和统一授权，其中本地授权采用把授权文件拷贝到产品安装目录进行授权，而统一授权则是通过在产品指定统一授权中心的连接信息进行连接授权。

9.1.1 本地授权

1. 获取本机特征码

```
# 获取本机特征码
./bin/alb-authcode

# 指定网卡获取特征码
./bin/alb-authcode -ac eth0

# 指定IP地址获取特征码
./bin/alb-authcode -ac 192.168.1.1
```

> *注： -ac参数仅支持alb标准版构建日期大于等于202506 2. 获取授权码后，请与金蝶天燕联系，将特征码发回金蝶天燕进行授权文件获取，或者登录金蝶天燕授权平台获取授权文件。 3. 将获取到的授权文件修改为文件名license.xml，然后放到ALB安装目录下。*

9.1.2 统一授权

1. 统一授权中心必须包含ALB标准版的有效授权信息。(请登录授权中心进行查看，若没有有效的产品授权信息，请参考授权中心手册，导入有效的产品授权信息。)
2. 产品配置连接授权中心的连接方式:

- acls.properties配置文件： acls.properties配置文件放置在ALB安装目录下。

```
apusic_acls_enable=true                          # 开启变量统一授权
apusic_acls_authUrls=192.168.101.34:6869         # 统一授权中心地址
apusic_acls_ns=public                            # 命名空间
apusic_acls_tenant=public                        # 租户名称
```

- 系统环境变量配置

```
export apusic_acls_enable=true                    # 开启变量统一授权
export apusic_acls_authUrls=172.24.3.116:6869     # 统一授权中心地址
export apusic_acls_ns=后付费                       # 命名空间
export apusic_acls_tenant=user_env中文             # 租户名称
```

注： 2种授权方式如果都进行了配置，则优先级为系统环境变量配置 > acls.properties文件配置;如果同时配置了本地授权方式和统一授权方式，则使用统一授权方式，若授权验证失败，将不会执行后续的授权验证。

9.2 ALB功能模块清单

模块	功能说明
--with-http_stub_status_module	启用 stub_status 接口，提供运行状态统计信息。
--with-http_ssl_module	启用 HTTPS/SSL 支持。
--with-ipv6	启用 IPv6 监听支持。
--with-http_mp4_module	支持 MP4 流媒体伪流（pseudo-streaming）。
--with-http_auth_request_module	实现子请求认证（auth_request），方便与外部认证服务集成。
--with-http_gzip_static_module	支持直接发送 .gz 预压缩文件，节省 CPU。
--with-http_realip_module	从 X-Forwarded-For/X-Real-IP 等头中获取真实客户端 IP。
--with-http_gunzip_module	对不支持 gzip 的客户端自动解压响应内容。
--with-http_sub_module	响应体字符串替换（sub_filter）。
--with-stream	启用 4 层 TCP/UDP 代理（stream 子系统）。
--with-stream_ssl_module	为 stream 子系统提供 SSL/TLS 支持。
--with-threads	开启线程池支持，提升文件 I/O 性能。
--with-file-aio	启用异步文件 I/O（Linux AIO）。
--with-http_v2_module	启用 HTTP/2 支持。
--with-http_v3_module	启用 HTTP/3（QUIC）支持。
--with-http_addition_module	在响应前后追加内容（add_before_body / add_after_body）。
--with-http_dav_module	支持 WebDAV 协议（PUT、DELETE、MKCOL 等）。

--with-http_random_index_module	目录索引时随机选择默认文件。
--with-http_secure_link_module	生成和校验带时效、防篡改的"安全链接"。
--with-mail	启用邮件代理（IMAP/POP3/SMTP）子系统。
--with-mail_ssl_module	为邮件代理提供 SSL/TLS 支持。
--with-http_slice_module	支持大文件分片下载（Range Slice）。
--with-stream_ssl_preread_module	在 4 层代理里预读 SNI/ALPN，实现按域名路由。
alb_healthcheck_module	ALB主动健康检查模块，支持四层和七层代理节点主动探活。
alb-sticky-module-ng	实现会话保持（sticky cookie/route）。
alb-module-vts	提供详细流量 / 状态统计接口（/status）。
alb-module-stream-sts	为 stream 子系统提供实时状态统计。
alb-module-sts	为 http 子系统提供实时状态统计。
alb_http_proxy_connect_module	支持 HTTP CONNECT 方法，正向代理 HTTPS 流量。
alb-rtmp-module	引入 RTMP/HLS/DASH 直播与流媒体功能。
alb_brotli	支持 Brotli 压缩算法，减少传输体积。
alb-module-lua	引入 Lua 脚本功能，可扩展 ALB 功能。
alb-headers-more	设置和清除-请求和响应头。

9.3 产品端口说明

ALB默认端口如下：

- 8080：http服务端口。
- 8887: ALB管控台端口

如需修改默认http代理端口，可以编辑 `ALB安装目录/conf/alb.conf` 文件中的 `listen 8080` 行。

如需修改管控台的端口，可以编辑 `ALB安装目录/console/conf/config.yaml` 文件中的sysytem中的addr字段端口。

9.4 开机启动项|系统服务

ALB默认不开启开机自动启动，如需开机启动，切换到安装目录，执行

```
cd /opt/ALB-V2.0.5-SE-amd64              # 进入安装目录
./utils/systemd/enable_startup.sh        # 自动设置开机启动项。
systemctl start alb                      # 启动ALB
```

9.4.1 移除系统启动项

```
cd /opt/ALB-V2.0.5-SE-amd64              # 进入安装目录
./utils/systemd/remove_startup.sh        # 移除开机启动项# 自动设置开机启动项。
```

9.5 普通用户启动ALB并且开启监听80端口

ALB支持通过修改程序权限，使得在普通用户下可以监听80端口。具体操作如下：

1. 切换到root用户，并切换到ALB安装目录。
2. 授权监听80端口： `setcap cap_net_bind_service=+eip ./sbin/alb`
3. 切换回普通用户

9.6 负载均衡算法

9.6.1 轮询

轮询是ALB默认的负载均衡算法，按时间顺序将请求轮流分配给后端服务器。如果有服务器宕机，ALB会自动将其剔除。

如下配置多个后端服务器后，默认使用轮询负载均衡算法。

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
}
```

9.6.2 权重轮询

在轮询的基础上，根据指定的权重分配请求，权重越高的服务器处理的请求越多，适用于服务器性能不均的情况。

如下配置多个后端服务器后，通过使用weight参数指定权重。

```
upstream backend {
    server backend1.example.com weight=10;   # 通过weight参数设置权重
    server backend2.example.com weight=20;
}
```

9.6.3 IP哈希

根据客户端的IP地址进行哈希运算，将请求分配给后端服务器。可以保证来自同一IP地址的请求始终被分配到同一台后端服务器。

如下配置多个后端服务器后，通过使用ip_hash指令启用IP Hash负载均衡算法。

```
upstream backend {
    ip_hash;   # 开启IP哈希负载均
    server backend1.example.com;
    server backend2.example.com;
}
```

9.6.4 最少连接数

根据后端服务器的当前活动连接数量进行分配，将请求分配给活动连接最少的后端服务器。适合请求处理时间差异较大的场景

如下配置多个后端服务器后，通过使用least_conn指令启用最少连接数负载均衡算法。

```
upstream backend {
    least_conn;   # 开启最少连接数负载均
    server backend1.example.com;
    server backend2.example.com;
}
```

9.7 健康检查

ALB支持对后端服务器进行健康检查，当后端服务器的响应不符合预期时，ALB会将其剔除。

9.7.1 被动健康检查

被动健康检查由ALB自动执行，根据后端服务器的响应状态码和超时时间来判断其是否健康。

ALB的被动健康检查通过 `fail_timeout` 和 `max_fails` 参数进行配置。

- `fail_timeout`：指定在连续失败后，ALB将后端服务器标记为不健康的时间段。默认值为10秒。
- `max_fails`：指定在 `fail_timeout` 时间段内，连续失败的次数。默认值为3次。

例如，如果你希望在某个后端服务器连续失败 3 次后将其标记为不可用，并在接下来的 30 秒内不向其发送请求，你可以这样配置

```
upstream backend {
    server 192.168.0.1:80 fail_timeout=30s max_fails=3;
    server 192.168.0.2:80 fail_timeout=30s max_fails=3;
}
```

注：

- 如果后端服务器在 fail_timeout 设置的时间结束后仍然不可用，ALB 会再次尝试向该服务器发送请求。
- 当后端服务器恢复正常后，ALB 会自动将其重新加入到轮询队列中。
- 使用这些指令可以帮助减少由于网络波动等原因造成的短暂失败对用户体验的影响。

9.7.2 主动健康检查

主动健康检查由ALB定期执行，通过向后端服务器的发送特定的请求判断是否健康。如果请求成功返回，则认为后端服务器是健康的；否则，将其标记为不健康。

9.7.2.1 七层HTTP代理健康检查

ALB的主动健康检查通过 `check` 指令进行配置。

```
upstream backend {
    server 192.168.0.1:80;
    server 192.168.0.2:80;

    # 启用健康检查
    check interval=3000 rise=2 fall=3 timeout=2000 type=http;

    # 定义健康检查的 URL
    check_http_send "GET /healthcheck HTTP/1.0\r\nHost:
www.example.com\r\n\r\n";
```

```
    check_http_expect_alive http_2xx http_3xx;
}
```

解释配置指令:

- check：开启健康检查，配置健康检查的基本参数。
- interval：定义两次检查之间的间隔时间（单位毫秒）。
- rise：定义连续几次成功响应后认为后端服务器是健康的。
- fall：定义连续几次失败响应后认为后端服务器是不健康的。
- timeout：定义单次健康检查请求的超时时间（单位毫秒）。
- type：定义健康检查请求的类型，如 http 或 tcp。
- check_http_send：定义发送给后端服务器的 HTTP 请求。
- 这里使用了一个 GET请求来检查 /healthcheck 路径。
- check_http_expect_alive：定义期望从后端服务器收到的成功响应的状态码范围。
- http_2xx 和 http_3xx 表示期望收到 2xx 和 3xx 状态码。

9.7.2.2 四层TCP代理健康检查

ALB主动健康检查支持TCP代理，以下是样例:

```
stream {
    upstream tcp-cluster {
        server 127.0.0.1:22;
        server 192.168.0.2:22;
        check interval=3000 rise=2 fall=5 timeout=5000
default_down=true type=tcp;
    }
    server {
        listen 522;
        proxy_pass tcp-cluster;
    }

    upstream udp-cluster {
        server 127.0.0.1:53;
        server 8.8.8.8:53;
        check interval=3000 rise=2 fall=5 timeout=5000
default_down=true type=udp;
    }
```

```
    server {
        listen 53 udp;
        proxy_pass udp-cluster;
    }
}
```

9.7.2.3 不同类型说明

1. TCP 健康检查 (type=tcp)，**支持七层和四层代理**

TCP 健康检查是最简单的类型之一，它仅需要建立一个 TCP 连接到后端服务器，并读取一个字节来判断服务器是否可连通。如下为实例配置

```
http {
    upstream backend {
        server 192.168.0.1:80;
        server 192.168.0.2:80;

        check interval=3000 rise=2 fall=3 timeout=2000 type=tcp;

    }

    server {
        listen 80;
        server_name example.com;

        location / {
            proxy_pass http://backend;
        }
    }
}
```

2. HTTP 健康检查 (type=http)，**支持七层和四层代理**

HTTP 健康检查通过发送一个 HTTP 请求到后端服务器来判断服务器的状态。如下为实例配置

```
http {
    upstream backend {
```

```
        server 192.168.0.1:80;
        server 192.168.0.2:80;

        check interval=3000 rise=2 fall=3 timeout=2000 type=http;

        # 发送一个 GET 请求到 /healthcheck 路径
        check_http_send "GET /healthcheck HTTP/1.0\r\nHost:
www.example.com\r\n\r\n";

        # 期望服务器返回 2xx 或 3xx 状态码表示健康
        check_http_expect_alive http_2xx http_3xx;
    }

    server {
        listen 80;
        server_name example.com;

        location / {
            proxy_pass http://backend;
        }
    }
}
```

3. SSL Hello 健康检查 (type=ssl_hello)

SSL Hello 健康检查通过发送 SSL 客户端 Hello 包，并接收服务器的 SSL Hello 包来判断服务器是否健康。。如下为实例配置

```
http {
    upstream backend {
        server 192.168.0.1:443;
        server 192.168.0.2:443;

        check interval=3000 rise=2 fall=3 timeout=2000
type=ssl_hello;
    }
```

```
    server {
        listen 443 ssl;
        server_name example.com;

        location / {
            proxy_pass https://backend;
        }
    }
}
```

4. MySQL 健康检查 (type=mysql)

MySQL 健康检查通过建立 MySQL 连接并接收问候响应来判断数据库服务器是否健康。

```
http {
    upstream backend {
        server 192.168.0.1:3306;
        server 192.168.0.2:3306;

        check interval=3000 rise=2 fall=3 timeout=2000 type=mysql;
    }

    server {
        listen 80;
        server_name example.com;

        location / {
            proxy_pass http://backend;
        }
    }
}
```

5. FastCGI 健康检查 (type=fastcgi)

FastCGI 健康检查通过发送一个 FastCGI 请求，并接收响应来判断服务器是否健康。

```
http {
    upstream backend {
        server 192.168.0.1:9000;
        server 192.168.0.2:9000;

        check interval=3000 rise=2 fall=3 timeout=2000 type=fastcgi;
    }

    server {
        listen 80;
        server_name example.com;

        location / {
            proxy_pass fastcgi://backend;
        }
    }
}
```

6. UDP 健康检查 (type=udp)，**支持七层和四层代理**

UDP 健康检查通过发送一个 UDP 请求，并接收响应来判断服务器是否健康。

```
stream {
    upstream udp-cluster {
        server 127.0.0.1:53;
        server 8.8.8.8:53;
        check interval=3000 rise=2 fall=5 timeout=5000
default_down=true type=udp;
    }
    server {
        listen 53 udp;
        proxy_pass udp-cluster;
    }
}
```

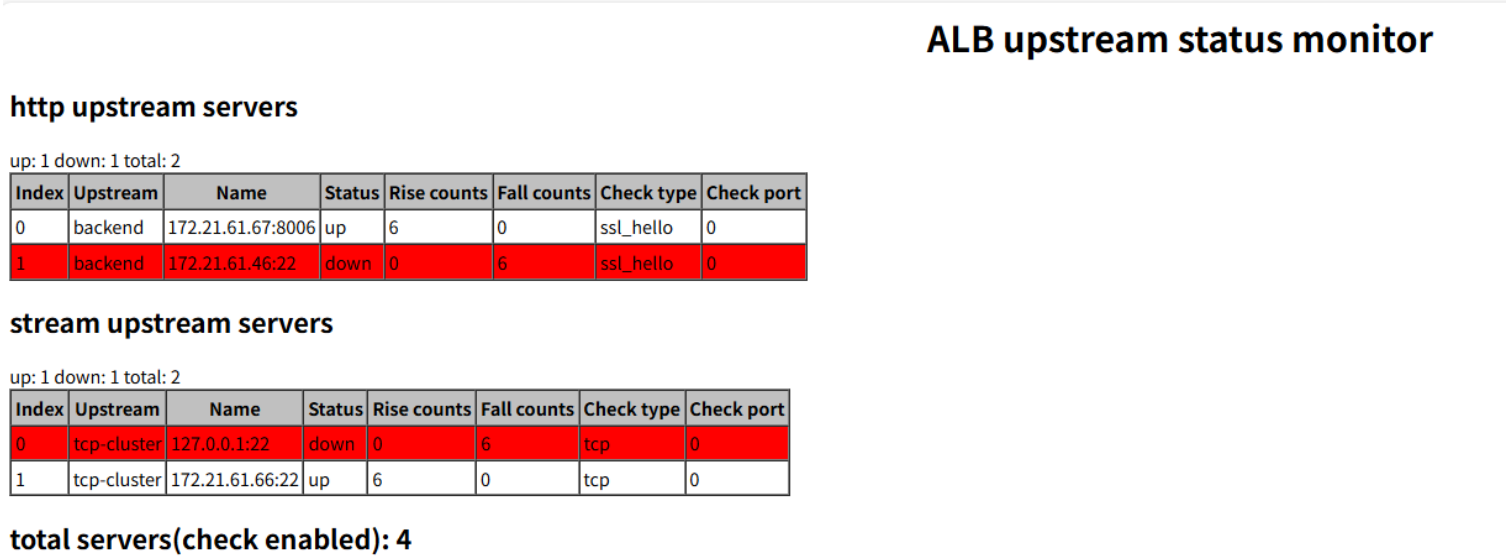
9.7.2.4 主动健康检查状态获取

ALB支持通过指定接口获取主动健康检查状态，如下：

1. 开启主动健康检查状态获取

```
# 省略其他配置

http {
    server {
        listen 8080;

        # 获取主动健康检查状态URL
        # 通过ALB的8080端口的/status接口获取健康状态数据页面
        location /status {
            healthcheck_status html;
        }
    }
    # 省略其他配置
}
```

访问8080端口/status接口，即可获取健康状态数据页面：



ALB upstream status monitor

http upstream servers

up: 1 down: 1 total: 2

Index	Upstream	Name	Status	Rise counts	Fall counts	Check type	Check port
0	backend	172.21.61.67:8006	up	6	0	ssl_hello	0
1	backend	172.21.61.46:22	down	0	6	ssl_hello	0

stream upstream servers

up: 1 down: 1 total: 2

Index	Upstream	Name	Status	Rise counts	Fall counts	Check type	Check port
0	tcp-cluster	127.0.0.1:22	down	0	6	tcp	0
1	tcp-cluster	172.21.61.66:22	up	6	0	tcp	0

total servers(check enabled): 4

2. 支持类型

- html：返回HTML页面
- json：返回JSON数据

- csv：返回CSV数据
- prometheus：返回Prometheus格式数据

9.8 监控

ALB提供丰富的监控指标，包括：

1. 简单流量统计数据，不支持Prometheus采集。
2. Prometheus-expoter，支持简单的流量数据的采集，支持Prometheus。
3. VTS监控数据，ALB精细化监控数据，支持HTML页面展示、JSON数据和Prometheus采集。

注： 上面三种方式可按需选择其中一种。

9.8.1 简单流量统计数据

ALB提供简单的流量统计信息，其监控数据有：

- Active connections: 当前活跃的连接数。
- Server accepts, handled, requests: 自ALB启动以来接受、处理的连接数以及收到的请求数。
- Reading: 当前正在读取客户端请求的数量。
- Writing: 当前正在向客户端写入响应的数量。
- Waiting: 当前处于等待状态的连接数。这些连接已经完成了读取操作，等待下一次写入操作。

9.8.1.1 配置简单流量统计数据和获取

在ALB配置文件中，添加以下内容：

```
server {
    listen 8080;            # node_status的端口

    location /node_status {
        stub_status on;     # 开启node_status功能
    }
}
```

启动ALB后，访问http://alb_ip:8080/node_status即可查看。

9.8.2 Prometheus-expoter（Prometheus监控数据）

ALB提供Prometheus监控功能，默认不启用，如需启用，请切换到utils/exporter目录，

执行：`./start-exporter.sh`，启动ALB的exporter。

9.8.2.1 启动说明

启动ALB的exporter必须要在配置文件中开启stub_status功能，如下：

```
server {
  listen 8080;              # node_status的端口
  location /node_status {  # 必须要使用node_status
        stub_status on;    # 开启stub_status
        access_log off;
        allow 127.0.0.1;
    }
}
```

在执行start-exporter脚本后，要求输入

- node_status对应的server监听端口，默认为8080.
- exporter监听端口，默认为9113

如下为输入过程：

```
root@root:/opt/ALB-V2.0.5-SE-amd64/utils/expoter$ ./start-
exporter.sh
input alb server listening port(请输入node_status监听端口):
alb server listening port(ALB node_status端口):  8080
input exporter listening port(请输入监听端口):
exporter listening port is 9113
exporter are running now!(exporter启动成功)
```

9.8.2.2 Prometheus监控数据获取

通过start-exporter成功后，可以通过浏览器或者prometheus访问: `http://IP:9113/metrics` 获取监控数据。

注： 如果上面修改了exporter的监听端口，上面url中的9113也一并修改。

9.8.3 VTS监控数据

vts模块提供更详细的监控信息，包括连接数、状态码等。与上面两种监控方式相比，vts模块可以提供更加全面和丰富的监控数据，通过vts模块，可以更加深入了解ALB的运行状态。

9.8.3.1 监控指标说明

监控类别	指标名称	说明
1. 基本信息	主机信息	代理服务器所在的主机名或 IP 地址
	版本号	ALB 或监控模块的版本
	服务器运行时间	ALB 进程已运行的时间（通常以秒为单位）
2. 连接信息	当前客户端连接数	当前活跃的客户端连接数量
	读取客户端连接总数	累计从客户端读取请求的连接数
	写入客户端连接总数	累计向客户端写入响应的连接数
	已处理客户端连接总数	累计已处理完成的客户端连接总数
3. HTTP 虚拟主机监控	请求总数	该虚拟主机接收的总请求数
	每秒请求量（RPS）	当前每秒处理的请求数
	1xx 错误码统计	所有 1xx 状态码（如 100, 101）的累计次数
	2xx 错误码统计	所有 2xx 成功响应（如 200, 201）的累计次数
	3xx 错误码统计	所有重定向响应（如 301, 302）的累计次数
	4xx 错误码统计	客户端错误（如 404, 403）的累计次数
	5xx 错误码统计	服务端错误（如 500, 502）的累计次数
	所有错误码统计	所有非 2xx 响应的总和（或全部状态码分类汇总）
	发送的总流量	该虚拟主机向客户端发送的总字节数
	接收的总流量	该虚拟主机从客户端接收的总字节数
	发送流量速率	当前每秒发送的字节数（B/s）
	接收流量速率	当前每秒接收的字节数（B/s）
4. HTTP 缓存监控	缓存命中数	请求命中缓存的次数
	缓存未命中数	请求未命中缓存的次数
	缓存旁路数	因配置 bypass 而跳过缓存的请求次数
	缓存已过期数	缓存项已过期被丢弃的次数

	缓存失效数	因主动失效（如 purge）导致的缓存移除次数
	缓存更新数	缓存被更新（如 stale-while-revalidate）的次数
	重新验证的缓存数	需要向源站重新验证的缓存请求数
	缓存总数	缓存命中 + 未命中 + 旁路等的总和（或当前缓存项总数）
5. HTTP 上游服务器监控	服务状态	upstream server 当前状态（up/down/checking）
	服务权重值	配置的 weight 值
	最大失败次数	max_fails 配置值
	故障后停止服务的时间值	fail_timeout 配置值（单位：秒）
	请求总数	发往该 upstream 的总请求数
	每秒请求量	当前每秒发往该 upstream 的请求数
	1xx 错误码统计	所有 1xx 状态码的累计次数
	2xx 错误码统计	所有 2xx 成功响应的累计次数
	3xx 错误码统计	所有重定向响应的累计次数
	4xx 错误码统计	客户端错误的累计次数
	5xx 错误码统计	服务端错误的累计次数
	所有错误码统计	upstream 返回的所有状态码分类汇总
	发送的总流量	与该 upstream 之间的总发送字节数
	接收的总流量	与该 upstream 之间的总接收字节数
	发送流量速率	与 upstream 通信的当前发送速率（B/s）
	接收流量速率	与 upstream 通信的当前接收速率（B/s）
6. Stream 虚拟主机监控	请求总数	Stream（TCP/UDP）会话总数
	每秒请求数	当前每秒新建会话数
	1xx 错误码统计	⚠ Stream 层无 HTTP 状态码，此项通常不适用（可映射为连接成功/失败）

	2xx 错误码统计	同上
	3xx 错误码统计	同上
	4xx 错误码统计	同上
	5xx 错误码统计	同上
	所有错误码统计	可指连接异常、超时、拒绝等非 HTTP 错误
	发送的总流量	Stream 代理的总发送字节数
	接收的总流量	Stream 代理的总接收字节数
	发送流量速率	当前 Stream 发送速率（B/s）
	接收流量速率	当前 Stream 接收速率（B/s）
7. Stream 上游服务器监控	服务状态	Stream upstream 的健康状态
	服务权重值	配置的 weight 值
	最大失败次数	max_fails 配置值
	故障后停止服务的时间值	fail_timeout 配置值（单位：秒）
	请求总数	发往该 Stream upstream 的总连接数
	每秒请求数	当前每秒新建连接数
	1xx~5xx 错误码统计	同 Stream 虚拟主机，通常指连接结果而非 HTTP 状态
	所有错误码统计	连接失败、超时等统计
	发送的总流量	与该 Stream upstream 的总发送字节数
	接收的总流量	与该 Stream upstream 的总接收字节数
	发送流量速率	当前发送速率（B/s）
	接收流量速率	当前接收速率（B/s）

9.8.3.2 vts模块配置和查看

如下是开启配置vts的样例：

```
http {
    vhost_traffic_status_zone;                  # 启用vts模块

    ...

    server {

        ...

        location /status {                      # 配置vts查看的uri地址
            vhost_traffic_status_display;       # 当前/status的uri展示vts监控指标
            vhost_traffic_status_display_format html;   # 设置默认格式为html页面
        }
    }
}
```

- HTML监控数据查看

通过浏览器访问查看监控页面：`http://IP:Port/status`

ALB Vhost Traffic Status

Server main

Host	Version	Uptime	Connections				Requests				Shared memory			
			active	reading	writing	waiting	accepted	handled	Total	Req/s	name	maxSize	usedSize	usedNode
mofant	2.0.1	27.18s	2	0	1	1	11	11	12	0	ngx_http_vhost_traffic_status	1024.0 KiB	3.4 KiB	1

Server zones

Zone	Requests			Responses					Traffic				Cache								
	Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s	Miss	Bypass	Expired	Stale	Updating	Revalidated	Hit	Scarce	Total
localhost	11	0	0ms	0	10	0	1	0	11	71.3 KiB	9.0 KiB	2.3 KiB	893 B	0	0	0	0	0	0	0	0	0
*	11	0	0ms	0	10	0	1	0	11	71.3 KiB	9.0 KiB	2.3 KiB	893 B	0	0	0	0	0	0	0	0	0

update interval: 1 sec

- JSON数据格式查看

通过浏览器或者curl等获取JSON数据：`http://IP:Port/status/format/json`

- Prometheus监控指标数据接口

可以通过Prometheus直接集成： `http://IP:Port/status/format/prometheus`

9.8.3.3 自定义监控指标

`vhost_traffic_status_filter_by_set_key` 指令允许你指定一个键名（key name），VTS 会根据这个键名来过滤流量统计信息。通常，这个键名对应于 set 指令中定义的一个变量.

```
vhost_traffic_status_filter_by_set_key key_name;
# 配置区域： http, server, location
```

说明： 通过用户定义的变量启用键。键是用于计算流量的键字符串。名称是用于计算流量的组字符串。键和名称可以包含变量，例如 $host、$server_name。如果指定，则名称所属的组。如果没有指定第二个参数名称，则键所属的组。

如下实例：

```
http {
    vhost_traffic_status_zone;                  # 启用vts模块

    ...

    server {

        ...
        vhost_traffic_status_filter_by_set_key  $uri uri::*;   # 过滤uri，统计不同uri的流量情况
        location /status {                      # 配置vts查看的uri地址
            vhost_traffic_status_display;       # 当前/status的uri展示vts监控指标
            vhost_traffic_status_display_format html;   # 设置默认格式为html页面
        }
    }
}
```

配置完后，即可查看不同uri的流量情况：

ALB Vhost Traffic Status

Server main

Host	Version	Uptime	Connections				Requests				Shared memory			
			active	reading	writing	waiting	accepted	handled	Total	Req/s	name	maxSize	usedSize	usedNode
mofant	2.0.1	46.43s	2	0	1	1	123	123	122	1	ngx_http_vhost_traffic_status	1024.0 KiB	20.7 KiB	6

Server zones

Zone	Requests			Responses						Traffic				Cache								
	Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s	Miss	Bypass	Expired	Stale	Updating	Revalidated	Hit	Scarce	Total
localhost	121	1	0ms	0	117	0	4	0	121	547.0 KiB	95.6 KiB	8.2 KiB	899 B	0	0	0	0	0	0	0	0	0
*	121	1	0ms	0	117	0	4	0	121	547.0 KiB	95.6 KiB	8.2 KiB	899 B	0	0	0	0	0	0	0	0	0

Filters

uri::*

Zone	Requests			Responses						Traffic				Cache								
	Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s	Miss	Bypass	Expired	Stale	Updating	Revalidated	Hit	Scarce	Total
/status/	2	0	0ms	0	2	0	0	0	2	105.0 KiB	2.0 KiB	0 B	0 B	0	0	0	0	0	0	0	0	0
/index.html	6	0	0ms	0	6	0	0	0	6	5.0 KiB	6.1 KiB	0 B	0 B	0	0	0	0	0	0	0	0	0
/node_status	1	0	0ms	0	1	0	0	0	1	242 B	125 B	0 B	0 B	0	0	0	0	0	0	0	0	0
/status/format/json	39	1	0ms	0	39	0	0	0	39	238.7 KiB	34.1 KiB	8.2 KiB	899 B	0	0	0	0	0	0	0	0	0
/favicon.ico	3	0	0ms	0	0	0	3	0	3	2.1 KiB	2.9 KiB	0 B	0 B	0	0	0	0	0	0	0	0	0

update interval: 1 sec

JSON

9.8.3.4 stream监控

ALB-VTS模块支持stream模块的监控。默认情况下，该模块不启用对stream模块的支持。其用法如下：

```
http {
    stream_server_traffic_status_zone;   # 启用stream vts模块

    ...

    server {

        ...

        location /stream/status {      # 监控指标暴露uri
            stream_server_traffic_status_display;
            stream_server_traffic_status_display_format html;
        }
    }
}
```

```
stream {
    server_traffic_status_zone;   # 监控的stream

    ...

    server {
        ...
    }
}
```

- HTML监控数据查看

通过浏览器访问查看监控页面：`http://IP:Port/stream/status`

ALB Stream Traffic Status

Server main

Host	Version	Uptime	Connections				Requests				Shared memory			
			active	reading	writing	waiting	accepted	handled	Total	Req/s	name	maxSize	usedSize	usedNode
mofant	2.0.1	26.35s	2	0	1	1	457	457	1597	1	stream_server_traffic_status	1024.0 KiB	0 B	0

Server zones

Zone	Port	Protocol	Requests			Responses						Traffic			
			Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s
*	0	TCP	0	0	0ms	0	0	0	0	0	0	0 B	0 B	0 B	0 B

Upstreams

tcp_backend

Server	State	Response Time			Weight	MaxFails	FailTimeout	Requests			Responses						Traffic			
		Connect	FirstByte	Response				Total	Req/s	Time	1xx	2xx	3xx	4xx	5xx	Total	Sent	Rcvd	Sent/s	Rcvd/s
172.21.32.106:9898	up	0ms	0ms	0ms	1	1	10	0	0	0ms	0	0	0	0	0	0	0 B	0 B	0 B	0 B
172.21.32.107:9898	up	0ms	0ms	0ms	1	1	10	0	0	0ms	0	0	0	0	0	0	0 B	0 B	0 B	0 B

update interval: 1 sec

JSON

- JSON数据格式查看

通过浏览器或者curl等获取JSON数据：`http://IP:Port/stream/status/format/json`

- Prometheus监控指标数据接口

可以通过Prometheus直接集成：`http://IP:Port/stream/status/format/prometheus`

9.8.3.5 VTS监控指标grafana集成

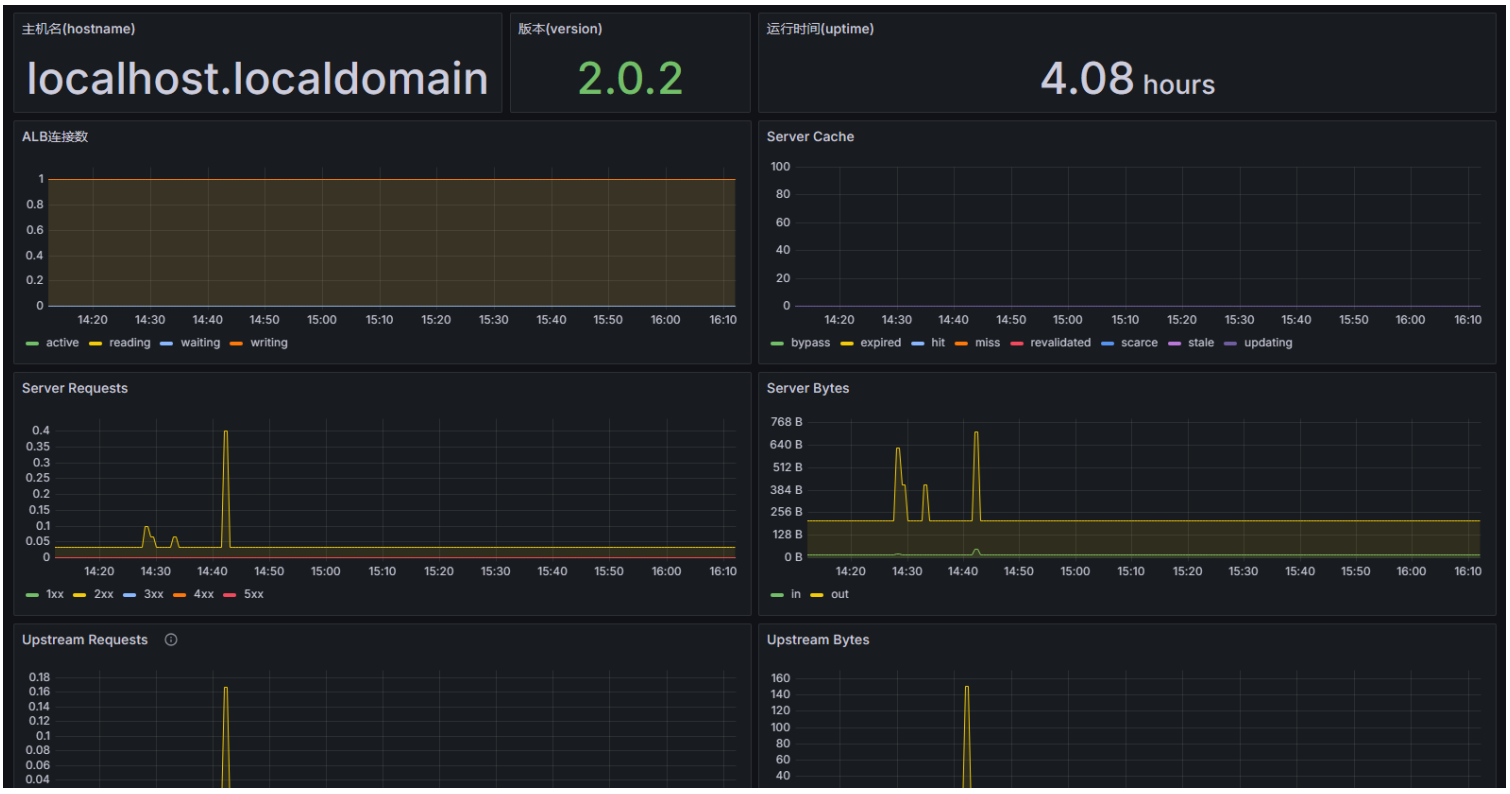
上面VTS监控指标提供prometheus采集格式，在prometheus中，可以配置如下采集：

```
scrape_configs:
- job_name: 'alb-vts-status'
  honor_timestamps: true
  scrape_interval: 1m
  scrape_timeout: 10s
  metrics_path: /status/format/prometheus # ALB vts监控指标采集url
  scheme: http
  enable_compression: true
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
      - 172.21.61.66:8080                 # ALB VTS 监控指标采集地址
    labels:
      group: 'alb'
      instance: 'alb'
```

其中grafana Dashboard的样例配置文件路径:

`ALB安装目录/utils/grafana/alb-grafana.json`

样例配置文件的展示效果如下：



可以在已部署的grafana中，导入该配置文件即可实现上述的监控仪表板。

9.9 TCP代理

ALB的stream模块可以用来做TCP代理，通过在alb配置文件中配置stream模块即可实现tcp代理。

如下的代理tcp示例：

```
# TCP代理
stream {
    upstream backend {
        server 127.0.0.1:8080;
    }

    server {
        listen 443;    # 监听端口

        proxy_pass backend;   # 转发到backend
    }
}
```

```
# 以下为HTTP配置内容
http {
  # 略
}
```

注： TCP代理配置只能写在最外层配置中，不能在写在http块或者在http块中通过include导入。

9.10 TCP代理加速

ALB TCP代理加速基于内核eBPF功能，在数据平面实现了高效的包处理和转发机制，显著提升了TCP代理的性能表现。

9.10.1 内核eBPF加速TCP转发优势

- 零拷贝转发：通过eBPF程序直接在内核态处理网络包，避免了传统方式中用户态和内核态之间的频繁数据拷贝。
- 智能路由决策：利用eBPF的高效匹配能力，快速确定数据包的转发要启用TCP代理的路径。

9.10.2 配置要求

- 用户权限：
  - user 指令必须使用具有特权的用户（如 root、admin）
  - 或者如果内核支持，可将 /proc/sys/kernel/unprivileged_bpf_disabled 设置为 0 来使用非特权用户
- 连接数限制：
  - 连接总数不超过1000000（10万）
- 硬件配置：
  - 内存： >= 16G
- 内核版本：
  - `kernel` 版本 >= 5.15
- ALB产品包要求：
  - 如 `ALB-V2.0.5-SE-20251127-amd64-ebpf.tar.gz` ，后缀携带ebpf特性。

9.10.3 使用示例

```
user root;


stream {
    upstream backend_servers {
        server 127.0.0.1:20001;
    }

    # 全局配置
    ebpf_proxy_timeout 600;   # 上下游之间无数据转发时的空闲超时时间(秒)
    ebpf_enable on;           # 全局启用 eBPF 加速

    server {
        listen 10000;
        ebpf_enable on;           # 在特定 server 中启用 eBPF (可覆盖全局设置)
        proxy_pass backend_servers;
    }

    server {
        listen 10001;
        ebpf_enable off;          # 禁用 eBPF 加速
        proxy_pass backend_servers;
    }
}
```

9.10.4 配置参数说明

9.10.4.1 ebpf_status_return

- 语法: `ebpf_status_return ;`
- 上下文: server
- 作用: 返回简单的状态信息，用于调试目的
- 示例:

```
server {
    listen 30001;
    ebpf_status_return;
}
```

9.10.4.2 ebpf_enable

- 语法: `ebpf_enable on | off;`
- 默认值: off
- 上下文: main, server
- 作用: 启用或禁用 eBPF 加速功能
- 示例:

```
# 全局启用
ebpf_enable on;

server {
    listen 10000;
    # 在特定 server 中启用
    ebpf_enable on;
    proxy_pass backend;
}
```

9.10.4.3 ebpf_proxy_timeout

- 语法: `ebpf_proxy_timeout milliseconds;`
- 默认值: 600000 (单位毫秒，= 600秒)
- 上下文: main, server
- 作用: eBPF连接断开时间，如果长连接，请设置为0
- 示例:

```
# 全局设置
ebpf_proxy_timeout 300000;

server {
```

```
    listen 10000;
    # 在 server 级别覆盖
    ebpf_proxy_timeout 60000;
    proxy_pass backend;
}
```

9.10.4.4 ebpf_proxy_buffer_size

- 语法: `ebpf_proxy_buffer_size size;`
- 默认值: 16384 (16KB)
- 上下文: main, server
- 作用: 设置用于存储从上游接收数据的缓冲区大小
- 示例:

```
# 全局设置
ebpf_proxy_buffer_size 32k;

server {
    listen 10000;
    # 在 server 级别设置
    ebpf_proxy_buffer_size 64k;
    proxy_pass backend;
}
```

9.10.4.5 ebpf_timer_period

- 语法: `ebpf_timer_period milliseconds;`
- 默认值: 2000 (2秒)
- 上下文: main, server
- 作用: 设置定时器周期，用于更新流量统计信息
- 示例:

```
# 全局设置
ebpf_timer_period 1000;

server {
```

```
    listen 10000;
    # 在 server 级别设置
    ebpf_timer_period 5000;
    proxy_pass backend;
}
```

9.11 TCP代理ADMQ

ALB支持使用TCP协议代理ADMQ的服务，下面是使用ALB标准版代理ADMQ服务的样例。

9.11.1 配置过程

1. 已部署ADMQ（略）
2. 部署ALB(略)
3. 修改ALB配置，支持ADMQ代理
4. 验证ADMQ服务代理

9.11.2 ADMQ环境和代理需求

- 假定目标ADMQ集群的三个服务节点为: 172.253.168.90、172.253.168.91、172.253.168.92，MQTT服务端口是5682。
- 假定ALB也同样开放5682端口对外提供ADMQ服务。

9.11.3 修改ALB配置，支持TCP代理ADMQ

编辑 `ALB安装目录/conf/alb.conf` 文件，在 `http` 配置之上最外层添加如下配置：

```
stream {
    upstream mq_server {
        server 172.253.168.90:5682 max_fails=3 fail_timeout=10s;
        server 172.253.168.91:5682 max_fails=3 fail_timeout=10s;
        server 172.253.168.92:5682 max_fails=3 fail_timeout=10s;
    }
    server {
        listen 5682 so_keepalive=on;    # ALB监听5682端口
        proxy_connect_timeout 15s;     # 连接超时时间
        proxy_timeout 300s;
```

```
        proxy_pass mq_server;           # ALB反向代理到三个mq节点
    }
}


# 以下是http配置内容，不需要改动
http {
    .... #略
}
```

9.11.4 验证ALB代理ADMQ服务

- 查看ALB监听端口5682是否存在。
- 若对应端口存在，使用MQ客户端通过ALB的IP和5682端口进行验证。

9.12 流式响应代理

ALB支持流式响应代理，即ALB不缓存响应，而是直接将响应转发给客户端，常用于各种AI/ChatGPT和对话机器人场景，下面是配置样例

```
location /streaming {
    proxy_pass http://backend_server;

    proxy_cache off; # 关闭缓存
    proxy_buffering off; # 关闭代理缓冲
    chunked_transfer_encoding on; # 开启分块传输编码
    tcp_nopush off; # 禁用TCP NOPUSH选项，允许快速发送小数据包，以减少延迟
    tcp_nodelay on; # 开启TCP NODELAY选项，禁止延迟ACK算法
    keepalive_timeout 300; # 设定keep-alive超时时间为300秒
}
```

9.12.1 配置详解

- `proxy_cache off`

> *关闭缓存功能，防止代理服务器缓存流式响应内容，确保客户端能够接收到实时、完整的响应。*

- `proxy_buffering off`

> *关闭代理服务器的响应缓冲，防止其缓冲整个响应后再发送给客户端，从而实现真正的流式传输效果。*

- `chunked_transfer_encoding on`

> *开启分块传输编码，允许将响应分成多个块进行传输。这是实现流式传输的关键。*

- `tcp_nopush off`

> *禁用TCP NOPUSH选项，允许快速发送小数据包。*

- `tcp_nodelay on`

> *开启TCP NODELAY选项，禁用延迟ACK算法。这有助于减少ACK包的延迟，确保数据及时发送。*

- `keepalive_timeout 300`

> *增加keepalive超时时间，防止在流式响应未完成时，代理与源服务器的连接就被关闭。这里设置为300秒，根据实际需求可以调整。*

9.13 日志配置

ALB支持两种类型的日志：访问日志（access logs）和错误日志（error logs）。下面详细介绍这两种日志的配置方法。

9.13.1 访问日志

访问日志记录了每个 HTTP 请求的信息，包括请求方法、请求 URI、响应状态码、响应大小、客户端 IP 地址等。

9.13.1.1 访问日志配置

访问日志可以通过 access_log 指令来配置。该指令可以出现在 http, server 或 location 块中。

9.13.1.2 基本配置样例

```
http {
    log_format main '$remote_addr - $remote_user [$time_local]
"$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

    server {
        listen 80;
        server_name example.com;

        access_log /var/log/alb/example.access.log main;
```

```
    }
}
```

- log_format: 定义日志条目的格式。上面的例子定义了一个名为 main 的格式
- access_log: 指定日志文件的位置和使用的格式。在这个例子中，日志文件被保存在 /var/log/alb/example.access.log，并且使用 main 格式。

9.13.1.3 日志格式详解

- $remote_addr: 客户端 IP 地址。
- $remote_user: 经过认证的用户名。
- $time_local: 本地时间戳。
- $request: 完整的请求行。
- $status: HTTP 状态码。
- $body_bytes_sent: 发送给客户端的响应主体大小。
- $http_referer: 请求的 Referer 请求头。
- $http_user_agent: 客户端的 User-Agent 请求头。
- $http_x_forwarded_for: X-Forwarded-For 请求头，通常用于反向代理环境。

9.13.1.4 关闭访问日志

如果你不想记录访问日志，可以将 access_log 设置为 off：

```
access_log off;
```

9.13.2 错误日志

错误日志记录了ALB运行过程中遇到的错误信息，这对于调试和维护非常重要。

9.13.2.1 配置错误日志

错误日志通过 error_log 指令来配置。该指令可以出现在 http, server 或 location 块中。

9.13.2.2 基本配置

```
http {
    error_log /var/log/alb/error.log warn;

    server {
        listen 80;
        server_name example.com;
```

```
    }
}
```

- error_log: 指定错误日志文件的位置和最小记录级别。在这个例子中，错误日志被保存在 /var/log/alb/error.log，并且只记录 warn 级别及以上的错误。

9.13.2.3 日志级别

错误日志的级别包括：

- debug: 调试信息。
- info: 一般信息。
- notice: 重要信息。
- warn: 警告信息。
- error: 错误信息。
- crit: 临界错误信息。
- alert: 警报信息。
- emerg: 紧急信息。

9.13.3 日志切割

由于日志文件会不断增长，定期清理和分割日志文件是非常重要的。 ALB提供了默认的日志切割工具，根据 JSON 配置文件自动生成 日志切割脚本（Shell 脚本或 logrotate 配置），并自动注册到系统 crontab，实现日志轮转自动化。

支持以下日志：

- ✅ 单个日志文件（如 /var/log/app.log）
- ✅ 通配符日志路径（如 /var/log/alb/*.log）
- ✅ 两种日志轮转模式：shell 或 logrotate
- ✅ 自动压缩、保留天数控制、进程信号通知（USR1）
- ✅ 安装 / 列出 / 卸载任务全生命周期管理
- ✅ 安全的 crontab 自动注册与清理

9.13.3.1 工具结构说明

工具目录： `ALB安装目录/utils/logrotate`

```
.
├── logrotate-config.json        ← 配置文件（默认名）
├── shell_scripts/               ← 生成的 Shell 脚本目录
│   └── <task-name>.sh
├── logrotate_configs/           ← 生成的 logrotate 配置目录
```

```
│   └── <task-name>
├── installed_tasks.json          ← 已安装任务的元数据（用于 list/remove）
```

字段	类型	必填	说明
name	string	✅	任务名称（用于文件名和注释）
log_file	string	✅	日志路径，支持通配符（如 *.log）；若为相对路径，会自动拼接到 albBaseDir
mode	string	✅	"shell" 或 "logrotate"
retain_days	int	❌	保留天数，默认 7 天
compress	bool	❌	是否压缩旧日志，默认 false
cron_schedule	string	❌	cron 表达式（如 "0 2 * * *"），默认 "0 1 * * *"（每天凌晨 1 点）

9.13.3.2 示例配置

```
[
  {
    "name": "alb-access",
    "log_file": "logs/access.log",
    "mode": "shell",
    "retain_days": 14,
    "compress": true,
    "cron_schedule": "0 2 * * *"
  },
  {
    "name": "alb-error",
    "log_file": "logs/error.log",
    "mode": "logrotate",
    "retain_days": 30,
    "compress": true
  },
  {
    "name": "all-app-logs",
    "log_file": "logs/app/*.log",
```

```
    "mode": "logrotate",
    "retain_days": 7,
    "compress": false
  }
]
```

> ✅ *相对路径 logs/access.log 会被自动转为绝对路径：/path/to/alb/logs/access.log* ✅ *通配符 logs/app/*.log 在 logrotate 和 shell 模式下均能正确处理*

9.13.3.3 使用方式

0. **权限要求**

- 需要对日志目录有**读写权限**
- 需要能执行 `crontab -l / crontab -`（通常普通用户即可）
- 若日志由 root 写入，建议以相同用户身份运行本工具
- 若保证可执行，则切换至root用户运行此工具
- 切换至utils/logrotate目录下执行

1. **安装任务（生成脚本 + 注册 crontab）**

```
# 切换至utils/logrotate目录下（必须）
cd utils/logrotate
# 使用默认配置文件安装日志切割任务
./logrotate-gen -install
```

执行后：

- 生成 Shell 脚本到 ./shell_scripts/
- 生成 logrotate 配置到 ./logrotate_configs/
- 自动将任务加入当前用户的 crontab
- 保存安装记录到 installed_tasks.json

2. **预览生成内容（不写文件）**

```
# 预览所有生成内容
./logrotate-gen -dry-run
```

```
# 预览指定配置，前提是配置文件test.json在当前目录
./logrotate-gen -dry-run -config ./test.json
```

示例输出

```
=== SHELL SCRIPT: ./shell_scripts/alb-access.sh ===
#!/bin/bash
set -euo pipefail
shopt -s nullglob
...

=== LOGROTATE CONFIG: ./logrotate_configs/alb-error ===
# Auto-generated for task: alb-error
/path/to/alb/logs/error.log {
    daily
    rotate 30
    compress
    ...
}
```

3. **列出已安装的任务**

```
./logrotate-gen -list
```

输出示例：

```
📅 Installed tasks (installed at: Mon Oct 27 10:00:00 CST 2025):
1. Name: alb-access
   Mode: shell
   Log File: /path/to/alb/logs/access.log
   Schedule: 0 2 * * *
   Retain Days: 14
   Compress: true

2. Name: alb-error
```

```
   Mode: logrotate
   Log File: /path/to/alb/logs/error.log
   Schedule: 0 1 * * * (default)
   Retain Days: 30
   Compress: true

📋 Current crontab entries for logrotate:
0 2 * * * /path/to/tool/shell_scripts/alb-access.sh # alb-access
0 1 * * * logrotate /path/to/tool/logrotate_configs/alb-error # alb-
error
```

4. **卸载所有任务**

```
./logrotate-gen -remove-all
```

执行后：

- 从 crontab 中移除所有本工具添加的条目
- 删除 shell_scripts/ 和 logrotate_configs/ 目录
- 删除 installed_tasks.json

> ⚠ *此操作不可逆，请谨慎使用！*

9.14 流量控制

ALB提供了多种方式来实现流量控制，包括限流、限速和限制连接。

9.14.1 使用 limit_req 模块进行限流

limit_req 模块允许你根据请求频率限制客户端的请求。

1、指令名称：`limit_req_zone`

- 语法：limit_req_zone key zone=name:size rate= number r/s
- 默认值：no
- 区域：http
- 使用示例：limit_req_zone $binary_remote_addr zone=addr:10m rate=1r/s
- 描述：定义一个限流区域，其中 $binary_remote_addr 是请求的客户端 IP 地址，zone=addr:10m 表示该区域的内存大小为 10MB，rate=1r/s 表示限制每个 IP 地址每秒只能发出一个请求。

2、指令名称：`limit_req`

- 语法：limit_req zone=name [burst=number] [nodelay]
- 默认值：no
- 区域：http, server, location
- 使用示例：limit_req zone=mylimit burst=5 nodelay
- 描述：在指定的区域中限制请求速率，其中 burst=5 表示允许的突发请求数量为 5 个，nodelay 表示不延迟处理超过 burst 的请求。

9.14.1.1 示例配置

假设你想限制每个 IP 地址每秒只能发出 1个请求：

```
http {
    limit_req_zone $binary_remote_addr zone=mylimit:10m rate=1r/s;

    server {
        listen 80;
        server_name example.com;

        location / {
            limit_req zone=mylimit burst=5 nodelay;
            proxy_pass http://backend;
        }
    }
}
```

- limit_req_zone: 定义一个限流区域，其中 $binary_remote_addr 是客户端 IP 地址，mylimit 是区域名称，10m 是内存大小（用于存储限流信息），rate=1r/s 表示每秒最多 1 个请求。
- limit_req: 在 location 块中应用限流规则。zone=mylimit 指定使用前面定义的限流区域，burst=5 允许突发请求，即在短时间内可以发送多于 1 个请求，但不能超过 burst 值；nodelay 表示不允许延迟请求。

9.14.2 使用 limit_conn 模块进行连接数限制

limit_conn 模块可以限制每个客户端的连接数。

1、指令名称：`limit_conn_zone`

- 语法：limit_conn_zone key zone=name:size;
- 默认值：no
- 区域：http

- 使用示例：limit_conn_zone $binary_remote_addr zone=addr:10m;
- 描述：定义一个限流区域，其中 key 是用于标识客户端的键（例如 IP 地址），zone=name 是区域名称，size 是内存大小（用于存储连接信息）。

2、指令名称：`limit_conn`

- 语法：limit_conn zone number;
- 默认值：no
- 区域：http, server, location
- 使用示例：limit_conn addr 10;
- 描述：在指定的区域中限制请求速率，其中 zone 是前面定义的区域名称，number 是允许的最大连接数。

9.14.2.1 示例配置

假设你想限制每个 IP 地址最多只能有 10 个并发连接：

```
http {
    limit_conn_zone $binary_remote_addr zone=myconn:10m;

    server {
        listen 80;
        server_name example.com;

        location / {
            limit_conn myconn 10;
            proxy_pass http://backend;
        }
    }
}
```

- limit_conn_zone: 定义一个连接数限制区域，其中 $binary_remote_addr 是客户端 IP 地址，myconn 是区域名称，10m 是内存大小（用于存储连接信息）。
- limit_conn: 在 location 块中应用连接数限制规则。zone=myconn 指定使用前面定义的连接数限制区域，10 表示最多允许 10 个并发连接。

9.14.3 使用 limit_rate 指令进行限速

limit_rate 指令可以限制客户端下载速度。

1、指令名称：`limit_rate`

- 语法：limit_rate rate;
- 默认值：no limit
- 区域：http, server, location
- 使用示例：limit_rate 100k;
- 描述：设置每个连接的下载速度限制，其中 rate 是速率（例如 100k 表示每秒传输 100KB 数据）。

9.14.3.1 示例配置

假设你想将每个客户端的下载速度限制为 100k/s：

```
server {
    listen 80;
    server_name example.com;

    location / {
        limit_rate 100k;
        proxy_pass http://backend;
    }
}
```

- limit_rate: 在 location 块中应用限速规则。100k 表示每秒最多传输 100KB 数据。

9.15 灰度发布和灰度测试

ALB支持灰度发布和灰度测试的功能，可以方便地对新版本进行测试，并平滑升级到生产环境。

9.15.1 场景介绍

场景	需求	典型例子
新功能灰度	只让内部/白名单用户体验	新下单接口只让 `uid=1000~2000` 的用户访问
AB 实验	按流量比例对比两套后端	50% 用户走 `v1` 服务，50% 走 `v2` 服务
地域/渠道灰度	按请求特征分流	广东用户走新机房，其他走旧机房
金丝雀发布	先放 5% 流量，错误率 >1% 自动回滚	监控 `5xx` 比例，超过阈值一键切回
UI/UX 改版测试	A/B 两组用户看到不同界面，评估点击率等指标	
故障回滚演练	快速切回旧版本，保障系统可用性	

9.15.2 ALB关键指令速查

ALB 本身不直接提供"灰度"功能，但可通过以下模块和指令组合实现：

- `map` 指令：根据请求特征（如 Cookie、Header、IP）生成变量
- `split_clients` 指令：基于哈希值按比例分流
- `upstream + server`：定义多个后端服务组

指令/变量	作用	示例
`split_clients`	按变量做**一致性哈希**分流	`split_clients "${remote_addr}AAA" $group { 5% canary; 95% stable; }`
`map`	读 header、cookie、arg 做**条件映射**	`map $cookie_ab_version $backend { default v1; canary v2; }`
`set $var` / `rewrite`	运行时改变量	`set $backend ''; rewrite ^/api/(.*) /$1 break;`
`proxy_pass`	把流量打到对应 upstream	`proxy_pass http://$backend;`
`access_log`	打日志方便回滚	`access_log /var/log/alb/canary.log main if=$log_canary;`

9.15.3 关键指令详解

9.15.3.1 1. split_clients —— 按比例分流（推荐）

```
split_clients ${variable} $backend {
    10%     backend_v2;
    *       backend_v1;
}
```

- `${variable}`：用于哈希的变量，如 $request_id、$remote_addr、$http_user_agent
- `$backend`：输出变量，用于 proxy_pass
- `10%`：流量比例，支持小数（如 0.5%）
- `*`：兜底，匹配剩余流量

✅ **优点**：简单、高效、无状态

❌ **缺点**：无法基于用户身份精准控制（除非变量是用户ID）

9.15.3.2 2. map —— 基于规则路由

```
map $http_x_version_flag $target_backend {
    default           backend_v1;
    "v2"              backend_v2;
    ~*beta            backend_v2;
}
```

- 根据请求头 `X-Version-Flag: v2` 路由到 v2
- 支持正则匹配（`~*` 表示不区分大小写）

✅ **优点**：灵活，可结合 Cookie、Header、IP 等

❌ **缺点**：需客户端配合传标识

3. upstream —— 定义后端服务组

```
upstream backend_v1 {
    server 192.168.1.10:8080;
}

upstream backend_v2 {
    server 192.168.1.11:8081;
}
```

每个 upstream 可包含多个 server，支持负载均衡 可配合 weight、backup 等参数

9.15.4 配置样例： 按IP地址和比例灰度发布

根据客户端 IP（$remote_addr）进行哈希计算，并将 约 5% 的 IP 分配到灰度组。

```
http {
    # 1. 定义 upstream
    upstream stable { server 10.0.0.1:8080; }
    upstream canary { server 10.0.0.2:8080; }

    # 2. 按 IP 做 5% 灰度
    split_clients "${remote_addr}AAA" $is_canary {
        5%      1;      # 变量值为 1 表示进灰度
```

```
        *       0;
    }

    # 3. 选后端
    map $is_canary $backend {
        1       canary;
        0       stable;
    }

    server {
        listen 80;
        server_name api.example.com;

        location / {
            proxy_set_header Host $host;
            proxy_pass http://$backend;
            # 打日志，方便回滚
            access_log /var/log/alb/canary.log combined
if=$is_canary;
        }
    }
}
```

9.15.5 基于Cookie的灰度

用户访问时带上 Cookie：gray=true，即可进入灰度环境。

```
# 若 Cookie 中有 gray=true，则走新版本
map $http_cookie $backend {
    default          old_app;
    ~*gray=true      new_app;
}

upstream old_app { server 10.0.0.10:8000; }
upstream new_app { server 10.0.0.11:8001; }
```

```
server {
    listen 80;
    location / {
        proxy_pass http://$backend;
    }
}
```

9.15.6 基于 IP 白名单强制灰度（内部测试解决方案）

```
upstream old_app { server 10.0.0.10:8000; }
upstream new_app { server 10.0.0.11:8001; }

map $remote_addr $backend {
    default          old_app;
    192.168.1.100    new_app;   # 内部测试 IP
    10.10.0.0/16     new_app;   # 整个内网段
}
```

9.15.7 动态控制灰度比例或基于数据库用户标签分流

- 从请求参数（如 ?user_id=123）或请求头（如 X-User-ID: 123）中读取用户 ID；
- 调用 Lua 脚本判断该用户是否属于灰度用户（例如：用户 ID 为偶数则进入新版本）；
- 根据判断结果，将请求代理到 old_app 或 new_app 后端服务；
- 支持日志记录灰度路由结果，便于监控和排查。

```
http {

  log_format gray_log '$remote_addr - [$time_local] '
                      '"$request" $status $body_bytes_sent '
                      '"$http_referer" "$http_user_agent" '
                      'user_id=$user_id backend=$backend_used';


  upstream old_app { server 10.0.0.10:8000; }
```

```
upstream new_app { server 10.0.0.11:8001; }

server {
  listen 80;
  server_name localhost;

  # 自定义变量，用于传递用户ID和后端选择
  set $user_id "";
  set $backend_used "";

  location / {
    # 1. 从 query string 或 header 获取 user_id
    set_by_lua_block $user_id {
        local user_id = ngx.var.arg_user_id
        if not user_id or user_id == "" then
          user_id = ngx.var.http_x_user_id
        end
        return user_id or ""
    }
    # 2. 判断是否灰度用户（user_id 为偶数），无用户ID，默认走旧版
    set_by_lua_block $target_backend {
        local user_id = ngx.var.user_id
        if user_id == "" then
          return "old_app"
        end

        local num = tonumber(user_id)
        if num and num % 2 == 0 then
          ngx.var.backend_used = "new_app"

          return "new_app"
        else
          ngx.var.backend_used = "old_app"

          return "old_app"
        end
```

```
        }
        # 3. 代理到对应后端
        proxy_pass http://$target_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # 访问日志包含灰度信息
    access_log logs/access.log gray_log;
  }
}
```

9.16 动态DNS解析

ALB 支持动态 DNS 解析，从而实现对域名的实时解析。该功能的核心是通过 resolver 指令配合变量来完成，示例如下：

```
http {
    server {
        listen 80;
        # 声明 DNS 服务器地址及解析结果的有效期
        resolver 8.8.8.8 valid=10s;
        # 定义变量，用于动态解析 DNS
        set  $ test private.server1.com.cn;
        location / {
            # 使用变量实现动态 DNS 解析
            proxy_pass http:// $ test;
        }
    }
}
```

在上述配置中，当访问服务器的根路径 / 时，请求会被转发到 `$test` 变量所定义的后端地址 `http://private.server1.com.cn` 。由于使用了变量 `$test` ，ALB会在每次请求（或缓存过期后）通过 resolver 指定的 DNS 服务器（如 8.8.8.8）对该域名进行重新解析。

解析结果的有效期由 valid=10s 指定，即缓存 10 秒。超过有效期后，下次访问将触发新的 DNS 查询。

重要说明：如果 `proxy_pass` 后直接写的是域名（而非变量），例如 `proxy_pass http://private.server1.com.cn;` ，那么ALB仅在启动或重载配置时解析一次该域名，无法实现动态更新，因此不能达到动态 DNS 解析的效果。

9.16.1 resolver 配置详解

动态域名解析依赖于 `resolver` 指令与变量的结合使用。 `resolver` 可以在 `http` 、 `server` 或 `location` 块中分别配置，作用范围相应不同。

```
http {
    server {
            listen 80;
            set   $test  private.server1.com.cn;
            location / {
                    resolver 8.8.8.8 valid=10s;
                    proxy_pass http://$test;
            }
            location /duplicate/ {
                resolver 114.114.114.114 valid=10s;
                proxy_pass http://$test;
            }
    }
}
```

在此配置中，访问 `/` 和 `/duplicate/` 路径时，虽然代理的目标域名相同（均为 private.server1.com.cn），但它们分别使用不同的 DNS 服务器进行解析：

- `/` 使用 `8.8.8.8`
- `/duplicate/` 使用 `114.114.114.114`

并且，这两个路径的 DNS 解析结果相互独立，不会共享缓存。

参数说明：

- `valid=10s` ：指定 DNS 解析结果的缓存有效期；
- `ipv6=on|off` ：控制是否接受 IPv6 地址。默认为 on，即当域名同时解析出 IPv4 和 IPv6 地址时，两者都会被使用。可通过设置 ipv6=off 禁用 IPv6 解析。

9.17 HTTPS与国密

ALB通过设置SSL证书和私钥来实现安全的HTTP通信。

9.17.1 前置准备

- 获取SSL证书和私钥，支持国密SSL证书。
  - 1、服务端加密证书文件
  - 2、服务端加密证书私钥文件
  - 3、服务端签名证书（**国密**必须）
  - 4、服务端签名证书私钥文件（**国密**必须）

9.17.2 编辑配置ALB的配置文件，开启SSL

- 配置说明: `conf/alb.conf` 文件中的server块内容。
- 核心配置： `listen` 、 `server_name` 、 `ssl_certificate` 、 `ssl_certificate_key`
- 示例配置：

```
server {
    listen 443 ssl;    # 使用SSL标记当前端口开启https
    server_name your_domain.com;    #  SSL证书对应的域名

    ssl_certificate /path/to/your_domain_name.enc.crt.pem;       # 服务端加密证书文件
    ssl_certificate_key /path/to/your_domain_name.enc.key.pem;   # 服务端加密证书私钥文件

    ssl_protocols TLSv1.2 TLSv1.3; # 使用更安全的协议版本
    ssl_prefer_server_ciphers on;   # 优先使用服务器定义的加密套件
    ssl_ciphers HIGH:!aNULL:!MD5;   # 使用更安全的加密算法

    location / {
        root /var/www/html;
        index index.html index.htm;
    }
}
```

- listen：指定监听的端口号，这里设置为 443，表示使用 HTTPS 协议。
- server_name：指定服务器的名称，这里设置为 your_domain.com，表示该服务器对应的域名。
- ssl_certificate：指定 SSL 证书的路径，这里设置为 /path/to/your_domain_name.crt，表示证书文件。

- ssl_certificate_key：指定 SSL 私钥的路径，这里设置为 /path/to/your_domain_name.key，表示私钥文件。
- ssl_protocols：指定支持的 SSL 协议版本，这里设置为 TLSv1.2 和 TLSv1.3，表示使用更安全的协议版本。
- ssl_prefer_server_ciphers：指定使用 stronger ciphers，表示使用更安全的加密算法。
- ssl_ciphers：指定支持的加密算法，这里设置为 HIGH:!aNULL:!MD5，表示使用更安全的加密算法。

9.17.3 配置指令说明

9.17.3.1 listen

- 描述：指定服务器监听的端口和协议。
- 示例： listen 443 ssl;

9.17.3.2 ssl_certificate 和 ssl_certificate_key

- 描述：指定 SSL 证书的路径和私钥的路径。
- 示例： ssl_certificate /path/to/your_domain_name.crt;

9.17.3.3 ssl_protocols

- 描述：指定支持的 SSL 协议版本。
- 示例： ssl_protocols TLSv1.2 TLSv1.3;（建议使用TLS1.2和TLS1.3）

9.17.3.4 ssl_ciphers

- 描述：指定 stronger ciphers 和支持的加密算法。
- 示例： ssl_ciphers HIGH:!aNULL:!MD5;（建议使用 stronger ciphers）

9.17.3.5 ssl_prefer_server_ciphers

- 描述：优先使用服务器定义的加密套件。
- 示例： ssl_prefer_server_ciphers on;

9.17.3.6 ssl_session_cache

- 描述：启用 SSL 会话缓存，以减少 SSL 连接的握手次数。
- 示例： ssl_session_cache shared:SSL:10m;

9.17.3.7 ssl_session_timeout

- 描述：设置 SSL 会话缓存的过期时间。
- 示例： ssl_session_timeout 10m;

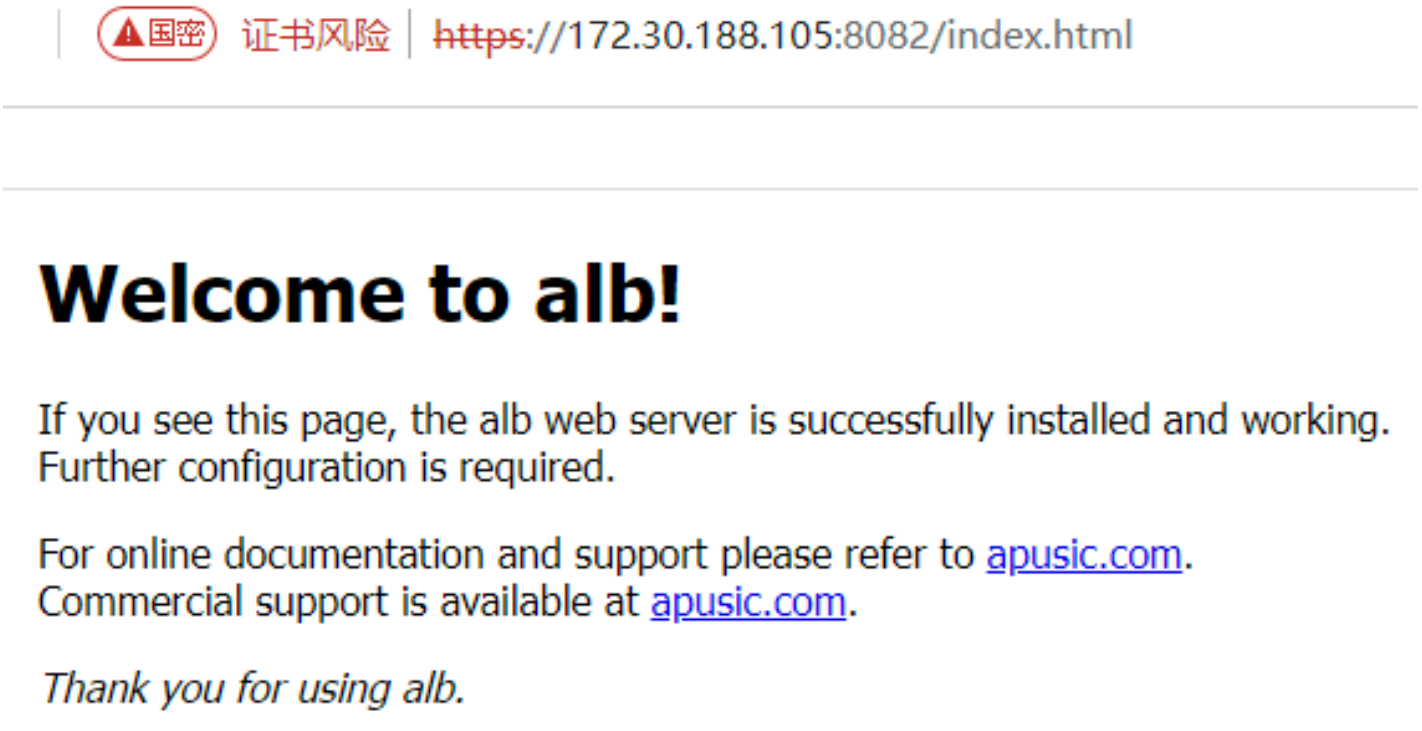
9.17.4 国密配置样例

注： 国密证书和密钥文件命名中必须包含签名证书（sig）和服务端证书（enc），否则将无法识别。同时sig证书必须配置在前面，如下配置：

```
server {
    listen 443 ssl;    # 使用SSL标记当前端口开启https
    server_name your_domain.com;    #  SSL证书对应的域名
```

```
    # 国密服务端签名证书和密钥，一般名称中携带sig字样（sig证书必须先配置）
    ssl_certificate /path/to/your_domain_name.sig.crt.pem;       # 服务端签名证书文件
    ssl_certificate_key /path/to/your_domain_name.sig.key.pem;   # 服务端加密证书私钥文件

    # 国密服务端证书和密钥，一般名字中携带enc（enc证书放在sig证书后面）
    ssl_certificate /path/to/your_domain_name.enc.crt.pem;       # 服务端加密证书文件
    ssl_certificate_key /path/to/your_domain_name.enc.key.pem;   # 服务端加密证书私钥文件

    ssl_protocols TLSv1.2 TLSv1.3; # 使用更安全的协议版本
    ssl_prefer_server_ciphers on;   # 优先使用服务器定义的加密套件
    ssl_ciphers HIGH:!aNULL:!MD5;   # 使用更安全的加密算法

    location / {
        root /var/www/html;
        index index.html index.htm;
    }
}
```

使用国密浏览器访问，在地址栏标记**国密**。



9.17.5 HTTP3

ALB V2.0.5及以上版本支持HTTP3协议，使用HTTP3协议时，请确保客户端（浏览器）支持HTTP3协议。

为了保证和原有协议兼容，可以参考下面的HTTP3配置：

```
server {
    listen 443 ssl;    # 使用SSL标记当前端口开启https
    listen 443 quic reuseport;    # 使用quic标记当前端口开启http3

    server_name your_domain.com;    #  SSL证书对应的域名

    ssl_certificate /path/to/your_domain_name.crt;        # 证书文件
    ssl_certificate_key /path/to/your_domain_name.key;   # 私钥文件

    ssl_protocols TLSv1.3; # 使用更安全的协议版本，HTTP3需要TLSv1.3
    ssl_prefer_server_ciphers on;   # 优先使用服务器定义的加密套件
    ssl_ciphers HIGH:!aNULL:!MD5;   # 使用更安全的加密算法

    # 告诉浏览器：同端口支持 HTTP/3，若不是443端口，请求修改和监听端口一致。
    add_header Alt-Svc 'h3=":443"; ma=86400';
```

```
    location / {
        root /var/www/html;
        index index.html index.htm;
    }
}
```

使用支持http3的curl工具测试：

```
 root@root:~$ /snap/bin/curl --http3-only -I
https://localhost:443/index.html -v -k
* Host localhost:443 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:443...
* error:8000006F:system library::Connection refused
* QUIC connect to ::1 port 443 failed: Could not connect to server
*   Trying 127.0.0.1:443...
* Server certificate:
*  subject: CN=localhost
*  start date: Aug 25 10:23:08 2025 GMT
*  expire date: Aug 25 10:23:08 2026 GMT
*  issuer: CN=localhost
*  SSL certificate verify result: self-signed certificate (18),
continuing anyway.
*   Certificate level 0: Public key type RSA (2048/112
Bits/secBits), signed using sha256WithRSAEncryption
* Connected to localhost (127.0.0.1) port 443
* using HTTP/3
* [HTTP/3] [0] OPENED stream for https://localhost:443/index.html
* [HTTP/3] [0] [:method: HEAD]
* [HTTP/3] [0] [:scheme: https]
* [HTTP/3] [0] [:authority: localhost:443]
* [HTTP/3] [0] [:path: /index.html]
* [HTTP/3] [0] [user-agent: curl/8.15.0]
* [HTTP/3] [0] [accept: */*]
```

```
> HEAD /index.html HTTP/3
> Host: localhost:443
> User-Agent: curl/8.15.0
> Accept: */*
>
* Request completely sent off
< HTTP/3 200
HTTP/3 200
< server: alb/2.0.5
server: alb/2.0.5
< date: Tue, 26 Aug 2025 08:37:04 GMT
date: Tue, 26 Aug 2025 08:37:04 GMT
< content-type: text/html
content-type: text/html
< content-length: 617
content-length: 617
< last-modified: Fri, 22 Aug 2025 08:05:21 GMT
last-modified: Fri, 22 Aug 2025 08:05:21 GMT
< etag: "68a824c1-269"
etag: "68a824c1-269"
< alt-svc: h3=":443"; ma=86400
alt-svc: h3=":443"; ma=86400
< accept-ranges: bytes
accept-ranges: bytes
<


* Connection #0 to host localhost left intact
```

9.18 正向代理

ALB可以用作正向代理服务器，它可以帮助客户端访问其他服务器或服务，同时隐藏客户端的真实 IP 地址。正向代理的主要用途包括缓存、负载均衡、安全过滤等。

9.18.1 示例配置

以下是一个简单的 ALB 配置示例，支持http和https正向代理，用于将请求转发到另一个服务器：

```
server {
    listen                          8080;
    server_name                     localhost;
    resolver                        114.114.114.114 ipv6=off;
    proxy_connect;
    proxy_connect_allow             443 80;
    proxy_connect_connect_timeout   10s;
    proxy_connect_read_timeout      10s;
    proxy_coneect_send_timeout      10s;
    location / {
        proxy_pass $scheme://$http_host$request_uri;
    }
}
```

- resolver: 设置 DNS 解析服务器，这里使用的是 114.114.114.114。
- proxy_connect: 启用代理连接功能。
- proxy_connect_allow: 允许的端口号，这里是 443 和 80。
- proxy_connect_connect_timeout: 连接超时时间，这里是 10 秒。
- proxy_connect_read_timeout: 读超时时间，这里是 10 秒。
- proxy_coneect_send_timeout: 发送超时时间，这里是 10 秒。

在 location 块中，使用 $scheme、$http_host 和 $request_uri 变量来构造目标服务器的地址。

- location /: 配置处理所有请求的路径，将请求转发到目标服务器。
- $scheme: 表示请求使用的协议（http 或 https）。
- $http_host: 表示请求的主机名和端口号。
- $request_uri: 表示请求的 URI。

10 高可用集群搭建

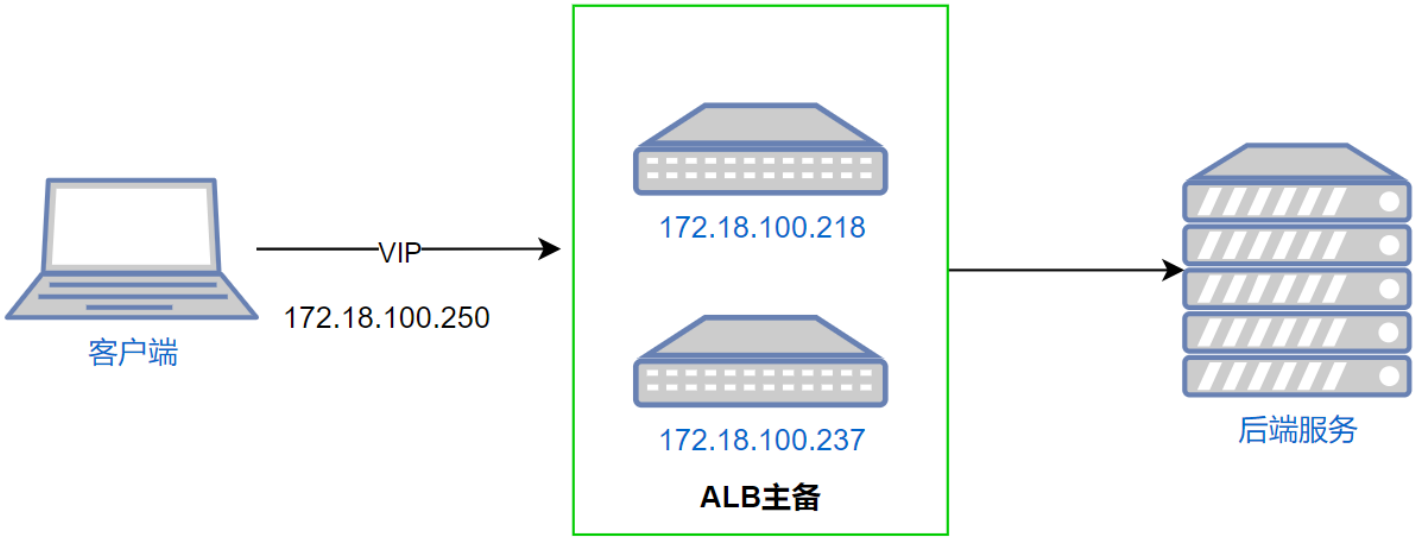
ALB支持原生高可用和keepalived两种高可用方式，搭建高可用集群，当集群节点发生故障时，自动将流量切换至备节点。

在ALB的管控台中一键配置和启动高可用，下面介绍在命令行中的安装配置方式。

10.1 前提准备

1. 两台服务器，一台作为主节点（Master），一台作为备用节点（Backup）。
2. 在两台服务器上部署 ALB 服务。（安装过程忽略）
3. 在两台服务器的局域网中，准备一个未使用的IP地址作为虚拟IP（VIP）。
4. 确保两台服务器之间的网络通信正常。
5. 确保两台服务器上的防火墙允许 keepalived 和 ALB 服务的通信。
6. 安装并配置 keepalived。
7. 验证VIP漂移。

下面以两台服务器（172.18.100.218和172.18.100.237）为例，选择VIP为172.18.100.250。



如上图，通过搭建两套ALB，并配置VIP，实现客户端通过VIP访问ALB，当VIP指向的ALB节点宕机的时候，VIP自动切换至备节点。

10.2 原生高可用

ALB自带高可用组件，可一键部署和启动ALB。在ALB管控台中配置高可用中，默认使用自带高可用组件。

10.2.1 查看物理网卡名称

1. 在主节点和备用节点上执行以下命令，查看当前机器IP的物理网卡名称：

```
ip addr show
```

2. 记录物理网卡名称，例如：eth0。

10.2.2 配置aha

主节点

1. 角色配置为: master
2. 修改VIP地址
3. 修改网卡名称

备节点

1. 角色配置为: backup
2. 修改VIP地址
3. 修改网卡名称

10.2.2.1 主节点

1. 切换至utils/aha目录，编辑config.yaml文件，如下样例

```
keepalived:
  # 虚拟路由ID
  vroute_id: 239

  # 虚拟IP绑定网卡名称
  interface: eth0

  # 虚拟IP地址
  vip: 172.18.100.250

  # 协议类型，可选值为ipv4和ipv6
  protocol: ipv4

  # 是否抢占模式
  #如果启用了抢占模式（Preempt Mode），当一个更高优先级的路由器加入到 VRRP 组中时，它可以取代当前的活动路由器成为新的活动路由器。
```

```
  #如果未启用抢占模式，则即使有更高优先级的路由器加入，当前的活动路由器也不会被取代
  preempt: true

  # 发送消息间隔(毫秒)
  msg-send-interval: 800

  # 优先级，0-255
  priority: 100

  # 角色，可选值为master和backup
  role: master

checker:

  # 检查服务的状态时间间隔(毫秒)
  check-interval: 1000

  # 重试次数
  retry-count: 3

  # 重试间隔(毫秒)
  retry-interval: 2000

  # 健康检查方式，可以url或者shell，url为http或https，shell为shell命令
  health-type: url

  # 当前目标服务状态检查url，用于检查目标服务是否健康
  health-url: http://localhost:8080/node_status

  # 自定义健康检查返回的url状态码
  health-codes:
    - 200
    - 404

  # shell健康检查命令，shell命令执行返回0表示健康，其他值表示异常
```

```
  health-cmd:

  # 当节点故障时候，是否尝试重启目标服务(默认为false)
  restart-on-failure: false
  # 重启目标服务命令
  restart-cmd:

aha:
  # 日志文件路径
  log-file: aha.log

  # 日志级别，可选值为debug, info, warn, error
  log-level: info
```

10.2.2.2 备节点

```
keepalived:
  # 虚拟路由ID
  vroute_id: 239

  # 虚拟IP绑定网卡名称
  interface: eth0

  # 虚拟IP地址
  vip: 172.18.100.250

  # 协议类型，可选值为ipv4和ipv6
  protocol: ipv4

  # 是否抢占模式
  #如果启用了抢占模式（Preempt Mode），当一个更高优先级的路由器加入到 VRRP 组中时，它可以取代当前的活动路由器成为新的活动路由器。
  #如果未启用抢占模式，则即使有更高优先级的路由器加入，当前的活动路由器也不会被取代
  preempt: true
```

```
  # 发送消息间隔(毫秒)
  msg-send-interval: 800

  # 优先级, 0-255
  priority: 100

  # 角色，可选值为master和backup
  role: backup

checker:

  # 检查服务的状态时间间隔(毫秒)
  check-interval: 1000

  # 重试次数
  retry-count: 3

  # 重试间隔(毫秒)
  retry-interval: 2000

  # 健康检查方式，可以url或者shell，url为http或https，shell为shell命令
  health-type: url

  # 当前目标服务状态检查url，用于检查目标服务是否健康
  health-url: http://localhost:8080/node_status

  # 自定义健康检查返回的url状态码
  health-codes:
    - 200
    - 404

  # shell健康检查命令，shell命令执行返回0表示健康，其他值表示异常
  health-cmd:

  # 当节点故障时候，是否尝试重启目标服务(默认为false)
  restart-on-failure: false
```

```
  # 重启目标服务命令
  restart-cmd:

aha:
  # 日志文件路径
  log-file: aha.log

  # 日志级别，可选值为debug, info, warn, error
  log-level: info
```

10.2.3 注册系统服务并启动系统服务

主备节点均需要执行以下操作：

1. 切换到sys-service目录
2. 切换到root用户，或者使用sudo执行下面命令
3. 执行：`./setup-aha-service.sh` 自动注册系统服务
4. 启动：`systemctl start aha.service`

10.2.4 验证VIP是否自动飘逸测试步骤

1. 使用VIP 172.18.100.250 访问ALB正常。
2. 停掉主节点所在的ALB，继续使用VIP访问ALB，验证是否正常。

10.2.5 其他配置说明：

1. 自定义健康检查URL： 修改配置文件中的health-url: http://localhost:8080/node_status
2. 自定义URL的健康HTTP状态码： 修改配置中的：health-codes: 200, 404
3. 配置aha.log日志文件路径： 修改配置文件中的log-file: aha.log
4. 通过自定义shell命令或脚本实现健康检查，要求返回0表示健康，其余值在ALB中标记不健康，如下样例：

```
# 使用shell命令进行健康检查
health-type: shell
# 使用shell命令检查alb进程存在则健康
health-cmd: ps aux | grep alb | grep -v grep
```

10.3 使用keepalived实现高可用

10.3.1 安装 keepalived

```
# kylin、opensuler、centos操作系统
yum install -y keepalived

# uos、ubuntu操作
apt-get install -y keepalived
```

10.3.2 查看物理网卡名称

1. 在主节点和备用节点上执行以下命令，查看物理网卡名称：

```
ip addr show
```

2. 记录物理网卡名称，例如：eth0。

10.3.3 配置 keepalived

1. 编辑主节点（Master）的 keepalived.conf 文件：

```
vi /etc/keepalived/keepalived.conf
```

在主节点/keepalived.conf文件中添加以下内容：

注：

- 把 `interface eth0` 替换为主节点的物理网卡名称。
- 把 `172.18.100.250` 替换为准备使用的虚拟IP地址。

```
! Configuration File for keepalived

global_defs {
   router_id alb
}

vrrp_script chk_http_port {
   script "/etc/keepalived/checkalb.sh"
   interval 2 #（检测脚本执行的间隔）
   weight 2
```

```
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    # 主、备机的 virtual_router_id 必须相同
    priority 100
    # 主、备机取不同的优先级，主机值较大，备份机值较小
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        # 请根据需要，修改这个虚拟IP地址
        172.18.100.250
    }
    track_script {
        chk_http_port
    }
}
```

2. 编辑备用节点（Backup）的 keepalived.conf 文件：

```
vi /etc/keepalived/keepalived.conf
```

在备用节点的keepalibed.conf文件中添加以下内容：

注：

- 把 `interface eth0` 替换为备用节点的物理网卡名称。
- 把 `172.18.100.250` 替换为准备使用的虚拟IP地址。

```
! Configuration File for keepalived
global_defs {
```

```
    router_id alb
}
vrrp_script chk_http_port {
    script "/etc/keepalived/checkalb.sh"
    interval 2 #(检测脚本执行的间隔)
    weight 2
}
vrrp_instance VI_1 {
    state BACKUP
    # state MASTER
    # 备份服务器上将 MASTER 改为 BACKUP
    interface eth0
    virtual_router_id 51
    # 主、备机的 virtual_router_id 必须相同
    priority 90
    # 主、备机取不同的优先级，主机值较大，备份机值较小
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        172.18.100.250 # 请根据需要，修改这个虚拟IP地址
    }
    track_script {
        chk_http_port
    }
}
```

3. 在**两个节点**创建检测脚本checkalb.sh：

```
vi /etc/keepalived/checkalb.sh
```

在checkalb.sh文件中添加以下内容：

```
#!/bin/bash
alb_count=$(ps -ef|grep "alb: main process" | grep -v grep |wc -l)
#1.判断ALB是否存活,如果不存活停止keepalived
if [ $alb_count -eq 0 ];then
    sleep 3
    #2.等待3秒后再次获取一次alb状态
    alb_count=$(ps -ef|grep "alb: main process" | grep -v grep |wc -
l)
    #3.再次进行判断, 如alb还不存活则停止Keepalived,让地址进行漂移,并退出脚本
    if [ $alb_count -eq 0 ];then
        echo "alb is down"
        exit 1
    fi
fi
```

在创建检测脚本后，给checkalb.sh添加执行权限：

```
chmod +x /etc/keepalived/checkalb.sh
```

10.3.4 主备节点均启动keepalived服务

```
systemctl start keepalived.service
```

10.3.5 验证VIP是否指向主节点

在主节点上查看虚拟IP地址（把eth0改为主节点网卡名称）：

```
ip addr show eth0 |grep inet|grep -v inet6|awk '{print $2}'
```

10.3.6 停止主节点alb，验证VIP是否漂移到备节点

```
# 切换到alb安装目录
./bin/stop-alb.sh
```

在备节点上查看虚拟IP地址（把eth0改为备节点网卡名称）：

```
ip addr show eth0 |grep inet|grep -v inet6|awk '{print $2}'
```

如果VIP指向主节点，则说明keepalived配置成功。

11 ALB代理安全配置

11.1 通用安全配置

11.1.1 隐藏ALB版本信息

```
http{
    more_clear_headers 'Server';   # 在http块中添加这行
}
```

11.1.2 限制请求方法

只允许必要的 HTTP 方法（如 GET、POST、HEAD、PUT）：

```
http {
    server {
        if ($request_method !~ ^(GET|HEAD|POST|PUT)$) {
            return 405;
        }
    }
}
```

11.1.3 限制客户端请求大小

防止 DoS 攻击（如大文件上传、超长 URL）：

```
http{
    client_max_body_size 1M;         # 限制请求体
    client_header_buffer_size 1k;    # 请求头缓冲区
    large_client_header_buffers 2 1k;
}
```

11.1.4 设置合理的超时时间

防止慢速攻击（Slowloris 等）

```
http{
    client_body_timeout 10s;
    client_header_timeout 10s;
    keepalive_timeout 5s;
    send_timeout 10s;
}
```

11.1.5 启用 HTTPS 并强制跳转

```
server {
    listen 80;
    server_name example.com;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers
ECDHE+AESGCM:DHE+AESGCM:AES256+EECDH:AES256+EDH:!aNULL:!MD5:!DSS;
    ssl_prefer_server_ciphers off;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
}
```

11.2 反向代理安全配置

11.2.1 清理/重写请求头

防止头注入或信息泄露：

```
http{
    server {
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # 清除客户端可能伪造的内部头
        proxy_set_header X-Original-URI "";
        proxy_set_header X-Internal-Secret "";
    }
}
```

11.2.2 限制后端响应头

避免后端泄露敏感信息（如 Server、X-Powered-By）： 可通过 proxy_hide_header 隐藏：

```
http {
    server {
        proxy_hide_header Server;
        proxy_hide_header X-Powered-By;
        proxy_hide_header X-AspNet-Version;
    }
}
```

11.2.3 限制代理请求速率

防CC攻击

```
http {
    server {
        limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;
        location /api/ {
            limit_req zone=api burst=20 nodelay;
            proxy_pass http://backend;
        }
```

```
    }
}
```

11.3 静态资源服务专用安全配置

11.3.1 禁止目录列表

确保未配置 `autoindex on;`，默认关闭即可。

11.3.2 限制可访问的文件类型

根据静态资源类型设置，只允许特定扩展名（如 .html, .css, .js, .png）：

```
location ~ \.(php|pl|py|jsp|asp|sh|cgi)$ {
    return 403;
}
```

11.3.3 防止敏感文件泄露

```
http{
    server {
        location ~ /\.(env|git|ht|svn|well-known/acme-challenge) {
            deny all;
        }
    }
}
```

11.3.4 设置安全的 Content Security Policy (CSP) 和其他安全头

```
add_header X-Content-Type-Options "nosniff" always;
add_header X-Frame-Options "DENY" always;
add_header X-XSS-Protection "1; mode=block" always;
add_header Referrer-Policy "strict-origin-when-cross-origin" always;
add_header Permissions-Policy "geolocation=(), microphone=(),
camera=()" always;
```

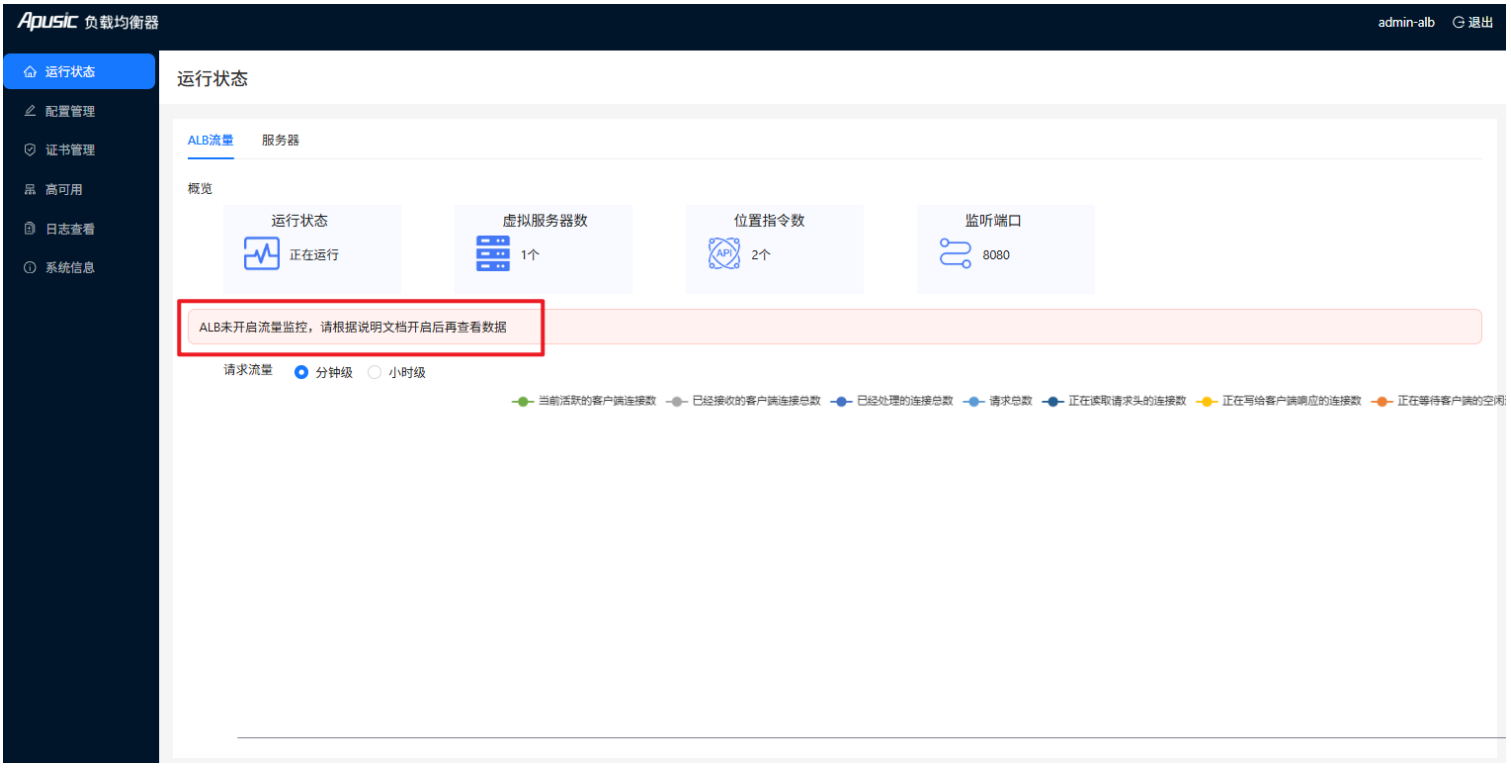
```
# CSP 示例（根据实际调整）
add_header Content-Security-Policy "default-src 'self'; script-src
'self' 'unsafe-inline' https://trusted.cdn.com; style-src 'self'
'unsafe-inline'; img-src 'self' data:; font-src 'self'; object-src
'none'; base-uri 'self'; form-action 'self';" always;
```

12 产品常见问题

注： 性能相关错误参考【性能优化】 手册。

12.1 管控台开启运行流量监控

启动ALB标准版登录管控台后，出现下面的提示：



则需要手动修改conf/alb.conf文件，在任意的server块中添加以下内容

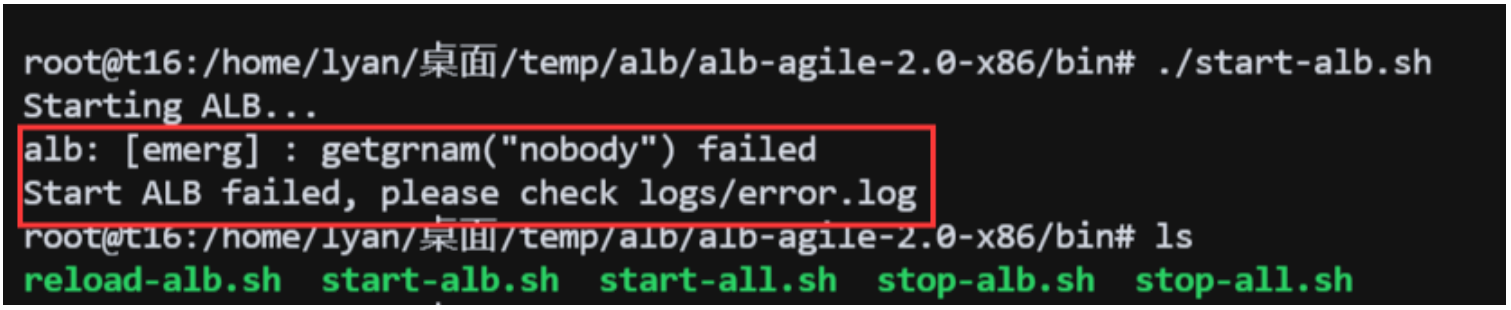
```
server {

    # 添加以下内容即可
    location /node_status {
        stub_status on;
        access_log off;
        allow 127.0.0.1;
    }
}
```

- 在任意server块中，添加上面的location块。

- 添加location块后，保存退出。
- 重新加载alb: `./bin/relaod_alb.sh` 。
- 刷新页面即可。

12.2 启动失败： getgrnam("nobody") failed

启动中出现下面错误：



修复方式：

修改conf/alb.conf文件的第一行注释，把 `#user nobody;` 改为 `user root;`

```
user root;
```

12.3 Docker容器启动失败

启动启动后失败，可以使用docker logs 查看容器日志。

1. 获取ALB容器ID

```
docker ps -a
CONTAINER ID        IMAGE                 COMMAND
CREATED             STATUS
     PORTS               NAMES
c9fcace189a3        alb-agile:V2.0.5     "bash /opt/ALB-V2...."   17
seconds ago       Exited (1) 16 seconds ago
alb-v202-se-docker-amd64-alb-1
```

如上ALB容器的ID是 `c9fcace189a3`

2. 查看容器日志

使用docker logs 查看日志，如下样例：

```
#docker logs 48c60b6fced6
Starting ALB...
alb: [emerg] : open() "/opt/ALB-V2.0.5-SE/conf/alb.conf" failed (13:
Permission denied)
Start ALB failed, please check logs/error.log
```

3. docker启动Permission denied错误

错误原因：容器内部的进程没有足够的权限来访问挂载的卷中的文件和目录。

解决方案：

- 修改`alb-standard-dockercompose.yaml`，标记使用`:z`标记挂载路径

```
version: '3'
services:
  alb:
    image: alb-agile:V2.0.5
    ports:
      - 8080:8080
      - 8887:8887
    volumes:
      - ./alb/alb.conf:/opt/ALB-V2.0.5-SE/conf/alb.conf:z
      - ./alb/license.xml:/opt/ALB-V2.0.5-SE/license.xml:z
      - ./alb/logs:/opt/ALB-V2.0.5-SE/logs:z
    command: bash /opt/ALB-V2.0.5-SE/bin/start-alb-and-console.sh
```

如上，每个挂载路径结尾添加`:z`标记。

- 重启容器

`docker-compose -f alb-standard-dockercompose.yaml up -d`

12.4 启动ALB是，显示授权信息后，停住了

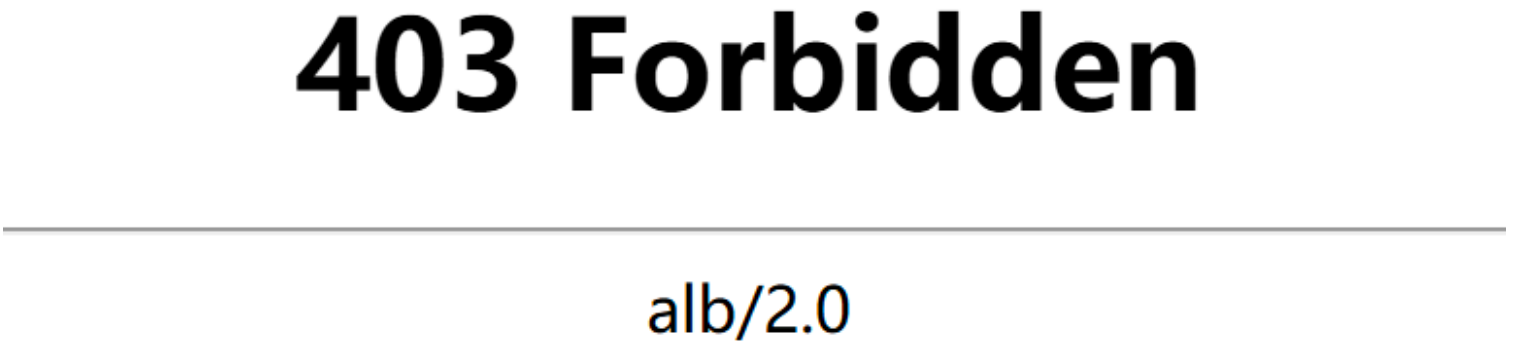
执行ALB启动后，终端显示完授权信息后，停住了，没有自动退出启动脚本，如下所示：

```
./bin/start-alb.sh
Starting ALB...
      ___                          _          Apusic Load Balancer
     /   |  ____  __  _______(_)____    Edition:Agile Edition
    / /| | / __ \/ / / / ___/ / ___/    License start: 2025-02-21
   / ___ |/ /_/ / /_/ (__  ) / /__      License end: 2025-06-31
  /_/  |_/ .___/\__,_/____/_/\___/      IP granted: 255.255.255.255
        /_/                             Licensee: 金蝶天燕研发测试

  version: V2.0.5
```

- 现象： 执行启动脚本后，并没有退出脚本的执行，也没有输出`ALB are running now!`等提示
- 原因分析： ALB前台启动，并没有后台启动
- 问题解决： 在conf/alb.conf配置文件中，检查是否存在`daemon off;`这个配置，如果是，删除即可，重新启动ALB。

12.5 访问页面存现403错误

在配置好ALB后，访问页面出现403错误页面，如下所示：



- 现象： 访问任意的页面（静态资源）均出现403错误。
- 原因分析： ALB启动的用户，没有对应静态资源的读权限，导致403错误。
- 问题解决： 在conf/alb.conf配置文件的第一行，把`user nobody;`改为`user root;`或者修改为对应静态资源有访问权限的用户名，最后重启ALB即可。

12.6 AAS登录失败

- 现象： 使用ALB代理多个AAS服务，在输入用户名和密码等点击登录页面后，然后出现登录失败或者跳转到登录页面。
- 原因分析：
  - 会话不一致： 浏览器访问的时候，不同请求发送到不同的AAS节点上，而登录的会话只在其中一个AAS服务器中。
  - 请求头或cookie丢失： AAS服务器中的服务依赖请求头或者cookie验证客户端是否登录，如果请求头或者cookie丢失，则无法验证客户端是否登录。
  - websocket连接失败： 某些服务在登录后，会简历websocket连接，如果websocket连接失败，则无法验证客户端是否登录。
- 问题解决：
  - 会话不一致： ALB使用ip_hash或者sticky负载均衡算法。
  - 请求头或cookie丢失： 配置转发全部请求头和cookie

```
location / {

  proxy_pass http://backend/;

  # 以下内容为新增的配置
  proxy_cookie_path / /;  # 确保 Cookie 路径正确
  proxy_pass_header Set-Cookie;



  proxy_set_header Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
  proxy_set_header X-Forwarded-Proto $scheme;

}
```

  - websocket连接失败： 单独为websocket连接地址编写location配置代理

```
location /websocket_url {
  proxy_pass http://backend/websocket_url;
```

```
        proxy_pass http://backend/efiles/;

    # WebSocket 必要配置
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";

    # 设置websockt连接读超时时间为10分钟，根据业务修改时间
    proxy_read_timeout 600s;
    # 设置websockt连接写超时时间为10分钟，根据业务修改时间
    keepalive_timeout 600s;

    proxy_cookie_path / /;  # 确保 Cookie 路径正确
    proxy_pass_header Set-Cookie;

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

12.7 后端反向代理服务的一个节点宕机后，服务出现无法访问，但其他节点正常

- 现象： ALB反向代理服务的ABC三个服务节点，其中A宕机或挂了，但BC服务正常，此时访问ALB出现短时间无法访问。
- 原因分析：ALB连接已经宕机的A服务超时导致全部服务不可用。在网络中，A服务宕机，可能导致ALB连接A服务等待时间过长，导致期间服务不可用。
- 问题解决：修改ALB连接上游服务的超时时间，默认为60秒，修改为10秒或者更少，同时启用主动健康检查，让ALB不再把请求转发给宕机服务，下面是参考样例

```
upstream backend {
  sticky; # 使用会话亲和负载均衡算法或者ip_hash
```

```
  server 192.168.1.1:8080 max_fails=3 fail_timeout=10s;
  server 192.168.1.2:8080 max_fails=3 fail_timeout=10s;
  server 192.168.1.3:8080 max_fails=3 fail_timeout=10s;

  # 配置主动健康检查（具体配置参考主动健康检查章节内容）
  check timeout=1000 fall=3 rise=2 type=http interval=3000;
  check_http_expect_alive http_2xx  http_3xx  http_4xx;
}

http {
  server {
    location / {
      proxy_pass http://backend/;
      # 核心配置，缩短服务连接超时（但高并发情况下可能出现错误，根据并发合理设置）
      proxy_connect_timeout 3s;
    }
  }
}
```

12.8 使用统一授权中心授权方式启动失败

- 现象： 使用统一授权中心授权方式启动失败，失败错误提示如下：

```
alb: [emerg]: center auth: request to center fault, err: send auth
auth info to authserver error
```

- 原因分析： 授权中心更新导致ALB解析请求失败
- 问题解决： 更新至2025年发版的ALB即可。

12.9 502错误： no live upstreams while connecting to upstream

- 现象： 前端访问出现502错误，并且ALB错误日志出现大量 `no live upstreams while connecting to upstream`
- 原因分析： 在ALB反向代理中，如果上游节点响应时间超过 `proxy_connect_timeout` 、 `proxy_send_timeout` 、 `proxy_read_timeout` ，那么将会标记节点不可用，同时在上游的配置中添加 `max_fails` 默认值为1，则导致ALB无法访问该节点出现上述错误。

- 问题解决：修改 `proxy_connect_timeout`、`proxy_send_timeout`、`proxy_read_timeout` 为更长的时间（超过60s），或者将 `max_fails` 设置为3，表示不标记节点不可用。
- 配置样例：

```
upstream gateway_api {
    # 增加 max_fails和fail_timeout参数
    server 127.0.0.1:9099 max_fails=3 fail_timeout=15s;
    server 127.0.0.2:9099 max_fails=3 fail_timeout=15s;
}

server {
  location /api/ {
    proxy_pass http://gateway_api/;

    # 增加超时时间参数
    proxy_connect_timeout 15s;
    proxy_send_timeout 60s;
    proxy_read_timeout 60s;
  }
}
```

12.10 管控台访问失败：非法Host请求

- 现象：访问管控台出现 **非法Host请求** 错误：`{"code":7,"data":null,"msg":"非法Host请求"}`
- 原因分析：这是管控台安全策略导致的，只允许访问的IP入口为本机IP地址，如果使用代理/docker等环境会出现这个错误
- 问题解决：修改管控台配置 `【ALB安装目录】/console/conf/config.yaml` 中 `system` 节点的 `docker` 项为 `true`

```
system:
    #系统设置的环境
    env: public
    #系统设置的端口
    addr: 8887
    #   系统路由全局前缀
```

```
routerprefix: "/api/alb/console"
# 使用https
https: false
certfile: "./cert/cert.pem"
keyfile: "./cert/key.key"
# 是否强制首次登录修改密码，默认强制（强制不影响默认账户）
change-pwd-at-first: true
# 是否是docker运行模式
docker: true      # 修改此项为true
```

配置完成后，重新启动管控台服务

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com