Apusic 金蝶天燕

生成时间：2026-05-19


# 用户手册-敏捷版

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 产品介绍
  - 1.1 产品简介
  - 1.2 核心功能
  - 1.3 产品架构与功能
    - 1.3.1 服务代理
    - 1.3.2 负载均衡
    - 1.3.3 流量管理
    - 1.3.4 安全
  - 1.4 部署架构
- 2 产品安装
  - 2.1 单节点部署
  - 2.2 集群部署
  - 2.3 docker部署
- 3 快速入门
  - 3.1 产品配置
  - 3.2 产品启动、停止和重启
  - 3.3 配置反向代理
  - 3.4 配置代理静态资源
  - 3.5 配置动静分离
  - 3.6 管控台使用
    - 3.6.1 WEB管理控制台运行状态
    - 3.6.2 WEB管理控制台节点配置管理
    - 3.6.3 WEB管理控制台节点日志查看
- 4 产品使用介绍
  - 4.1 产品license使用说明
    - 4.1.1 授权文件类型支持
    - 4.1.2 授权类型配置
    - 4.1.3 不同授权类型及其配置说明
    - 4.1.4 通过环境变量配置统一授权
  - 4.2 产品端口修改
  - 4.3 开机启动项
  - 4.4 ALB服务健康检查

- 4.4.1 ALB健康检查日志
- 4.5 ALB日志自动切割
  - 4.5.1 开启日志切割
  - 4.5.2 日志切割功能取消
- 4.6 负载均衡算法
  - 4.6.1 轮询
  - 4.6.2 权重轮询
  - 4.6.3 IP哈希
  - 4.6.4 最少连接数
- 4.7 健康检查
  - 4.7.1 被动健康检查
  - 4.7.2 主动健康检查
    - 4.7.2.1 不同类型说明
- 4.8 监控
  - 4.8.1 简单流量统计数据
    - 4.8.1.1 配置简单流量统计数据和获取
  - 4.8.2 Prometheus监控
    - 4.8.2.1 启动说明
    - 4.8.2.2 Prometheus监控数据获取
  - 4.8.3 使用vts模块监控
    - 4.8.3.1 HTTP监控指标说明
    - 4.8.3.2 vts模块配置和查看
    - 4.8.3.3 自定义监控指标
    - 4.8.3.4 stream监控
- 4.9 TCP代理
- 4.10 日志配置
  - 4.10.1 访问日志
    - 4.10.1.1 访问日志配置
    - 4.10.1.2 基本配置样例
    - 4.10.1.3 日志格式详解
    - 4.10.1.4 关闭访问日志
  - 4.10.2 错误日志
    - 4.10.2.1 配置错误日志
    - 4.10.2.2 基本配置

- 4.10.2.3 日志级别
- 4.10.3 日志切割
- 4.10.3.1 使用 logrotate
- 4.10.3.2 使用 cron
- 4.11 流量控制
- 4.11.1 使用 limit_req 模块进行限流
- 4.11.1.1 示例配置
- 4.11.2 使用 limit_conn 模块进行连接数限制
- 4.11.2.1 示例配置
- 4.11.3 使用 limit_rate 指令进行限速
- 4.11.3.1 示例配置
- 4.12 正向代理
- 4.12.1 示例配置
- 5 高可用集群搭建
- 5.1 前提准备
- 5.2 原生高可用
- 5.2.1 查看物理网卡名称
- 5.2.2 配置alb-ha
- 5.2.2.1 主节点
- 5.2.2.2 备节点
- 5.2.3 注册系统服务并启动系统服务
- 5.2.4 验证VIP是否自动飘逸测试步骤
- 5.2.5 其他配置说明:
- 5.3 使用keepalived实现高可用
- 5.3.1 安装 keepalived
- 5.3.2 查看物理网卡名称
- 5.3.3 配置 keepalived
- 5.3.4 主备节点均启动keepalived服务
- 5.3.5 验证VIP是否指向主节点
- 5.3.6 停止主节点alb，验证VIP是否漂移到备节点
- 6 产品常见问题
- 6.1 管控台开启运行流量监控
- 6.2 启动失败： getgrnam("nobody") failed

# 1 产品介绍

## 1.1 产品简介

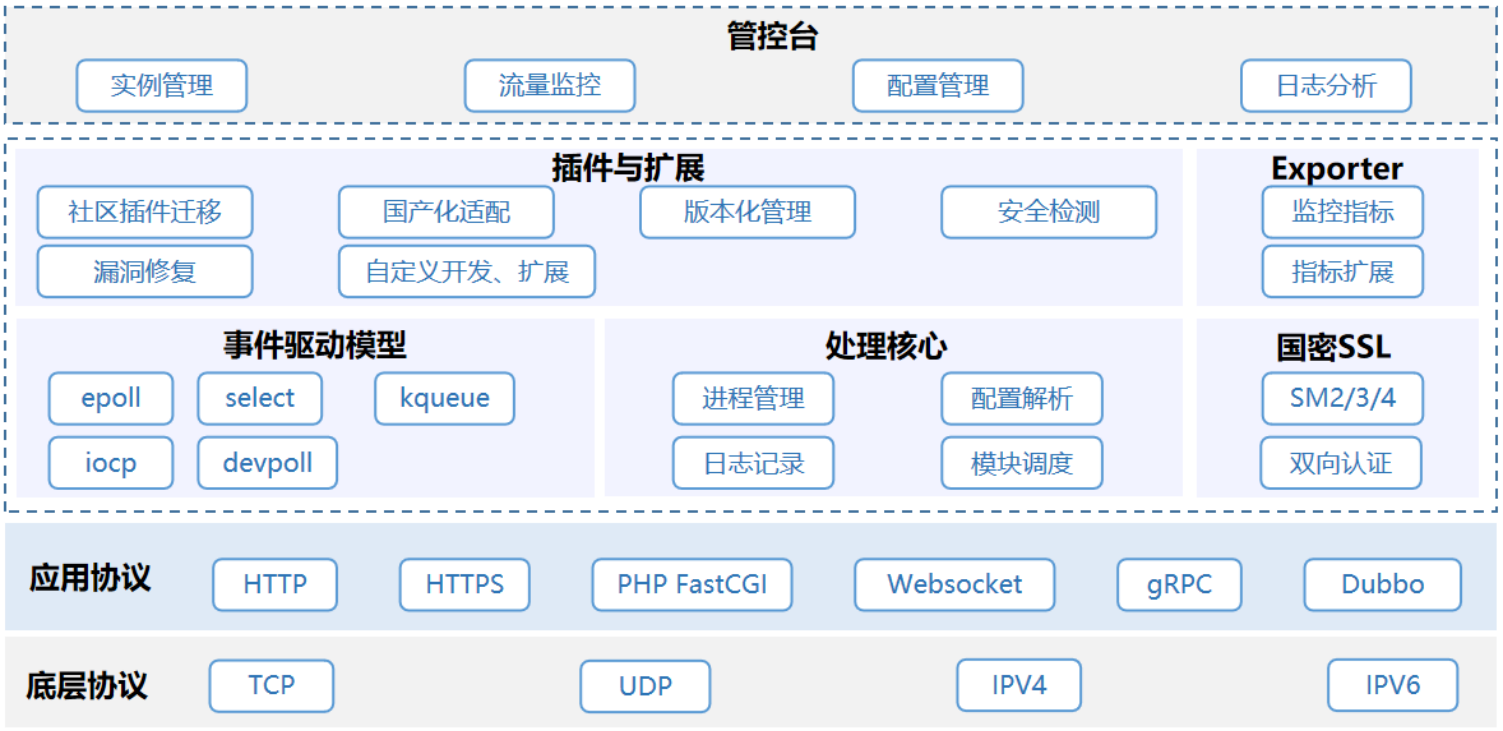
金蝶天燕负载均衡器软件敏捷版（Apusic Load Balancer，ALB）是一款兼具轻量级、高性能及高可用性的负载均衡器软件。ALB能有效应对大规模服务集群在面向客户端时的请求与流量管理挑战，承担起请求安全控制、验证、过滤，以及负载均衡和反向代理的重任。通过这些功能，ALB成功隔离了客户端访问对服务应用系统、平台及其资源的直接影响，实现了对集群访问流量的有效控制、精细化管理和均衡分配，确保服务稳定性和高效性。

## 1.2 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。 |
| 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。 |

## 1.3 产品架构与功能

ALB敏捷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敏捷版能够灵活地添加新功能，满足不同业务场景的需求。ALB敏捷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。




### 1.3.1 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 | 支持IPV4和IPV6协议。 |

### 1.3.2 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。 |

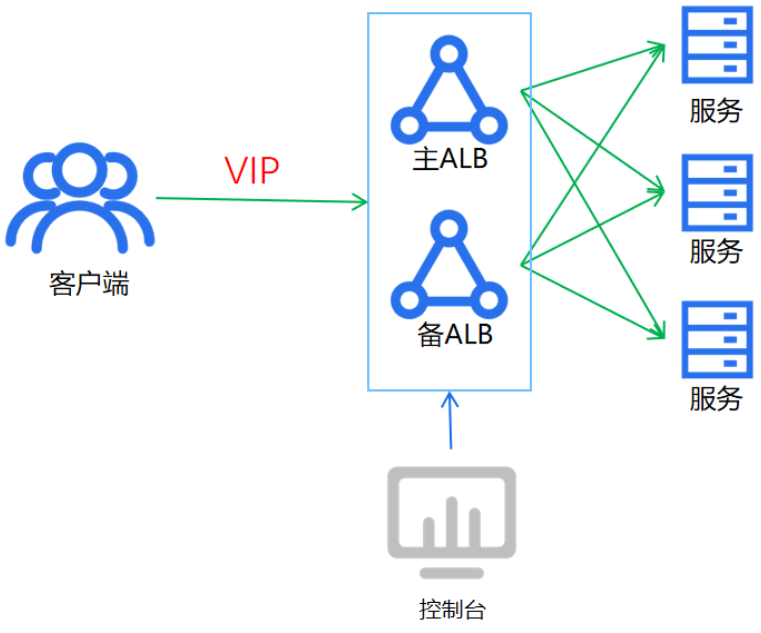
### 1.3.3 流量管理

| 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。 |

### 1.3.4 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。 |

## 1.4 部署架构




ALB敏捷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。
- 控制台：管理ALB节点的配置、监控和日志等。
- 后端服务器集群：承载业务应用。

# 2 产品安装

## 2.1 单节点部署

1. 获取软件安装包 ALB安装介质为:alb-agile-2.0-x86.tar.gz或alb-agile-2.0-arm.tar.gz
2. 复制安装包到目标主机
3. 解压:alb-agile-2.0-x86.tar.gz到目标安装路径（示例为/opt目录）

```
tar -zxvf alb-agile-2.0-x86.tar.gz -C /opt/
```

注：如果没有权限，请使用root账户或sudo

## 2.2 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。

## 2.3 docker部署

alb敏捷版提供arm和x86的系统镜像，如：

- alb-agile-2.0-docker-arm.tar.gz
- alb-agile-2.0-docker-x86.tar.gz

1. docker镜像安装

下面的操作以alb-agile-2.0-docker-arm.tar.gz为例

- 解压 `tar -zxvf alb-agile-2.0-docker-arm.tar.gz`
- 进入解压后的文件夹 `alb-agile-2.0-docker-arm`
- 加载镜像： `docker load < alb-agile-2.0-image-arm.tar.gz`

2. 镜像启动

使用docker-compose管理镜像的启动，其中涉及license授权、alb的配置需要挂载出来。

另外需要映射alb配置文件中对应的端口，以下是参考：

```
version: "3.5"
services:
```

```
  alb:
    image: alb-agile:2.0 # 注意核对镜像版本
    privileged: true
    volumes:
      - "/opt/alb-agile-install-dir/alb-agile-2.0/alb.conf:/opt/alb-agile-2.0/conf/alb.conf" # alb配置文件
      - "/opt/alb-agile-install-dir/alb-agile-2.0/alb_license.conf:/opt/alb-agile-2.0/conf/alb_license.conf" # alb授权配置
      - "/opt/alb-agile-install-dir/alb-agile-2.0/license.lic:/opt/alb-agile-2.0/license.lic" # alb授权文件
    working_dir:  /opt/alb-agile-2.0/
    entrypoint: ./bin/start-all.sh
    ports:
      - 8080:8080       # ALB http对外端口
      - 8887:8887       # ALB 管控台端口
```

**注意**： 配置中的/opt/alb-agile-install-dir需要替换为当前`alb-agile-2.0-docker-arm.tar.gz`后的目录。

- 确定已经修改alb-agile-docker-compose.yaml
- 启动命令： `docker-compose -f alb-agile-docker-compose.yaml up -d`

# 3 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-agile-2.0-x86`，并将申请的ALB敏捷版KBC授权文件`license.lic`放入到安装目录下。

## 3.1 产品配置

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

安装路径`/conf/alb.conf`

**直接使用Nginx的配置方式和内容进行配置即可**

## 3.2 产品启动、停止和重启

- 启动ALB实列和管理控制台

```
cd /opt/alb-agile-2.0-x86                # 进入安装目录
./bin/start-all.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-agile-2.0-x86                # 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-agile-2.0-x86                # 进入安装目录
./bin/stop-all.sh                        # 停止alb和管理控制台
```

- 停止ALB实例

```
cd /opt/alb-agile-2.0-x86                # 进入安装目录
./bin/stop-alb.sh                        # 停止alb
```

- 产品热加载

```
cd /opt/alb-agile-2.0-x86                # 进入安装目录
./bin/reload-alb.sh                      # 停止alb
```

## 3.3 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
```

## 3.4 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
```

## 3.5 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
```

## 3.6 管控台使用

ALB敏捷版管控控制台访问地址为: `http://IP:8887`

- 默认用户名: `admin`
- 默认登录密码: `Apusic@123`

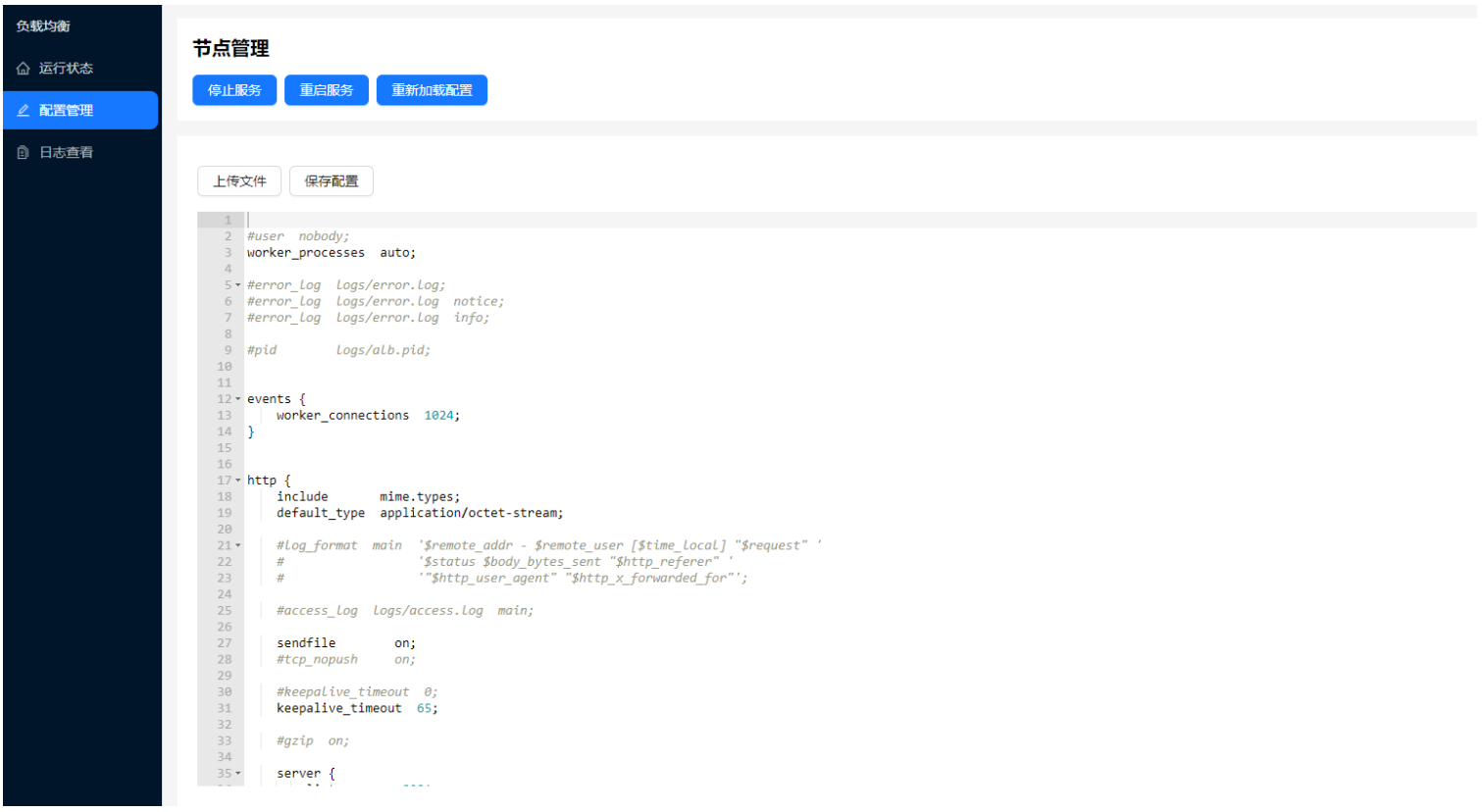
### 3.6.1 WEB管理控制台运行状态

ALB运行状态展示了ALB服务运行情况和流量数据。


- 虚拟服务器数量: 配置文件中的server块
- 位置指令数量: server块中的location块数量

- 监听端口： 全部server的listen端口。
- 请求流量： 依赖node-status模块，展示alb实时请求数据。

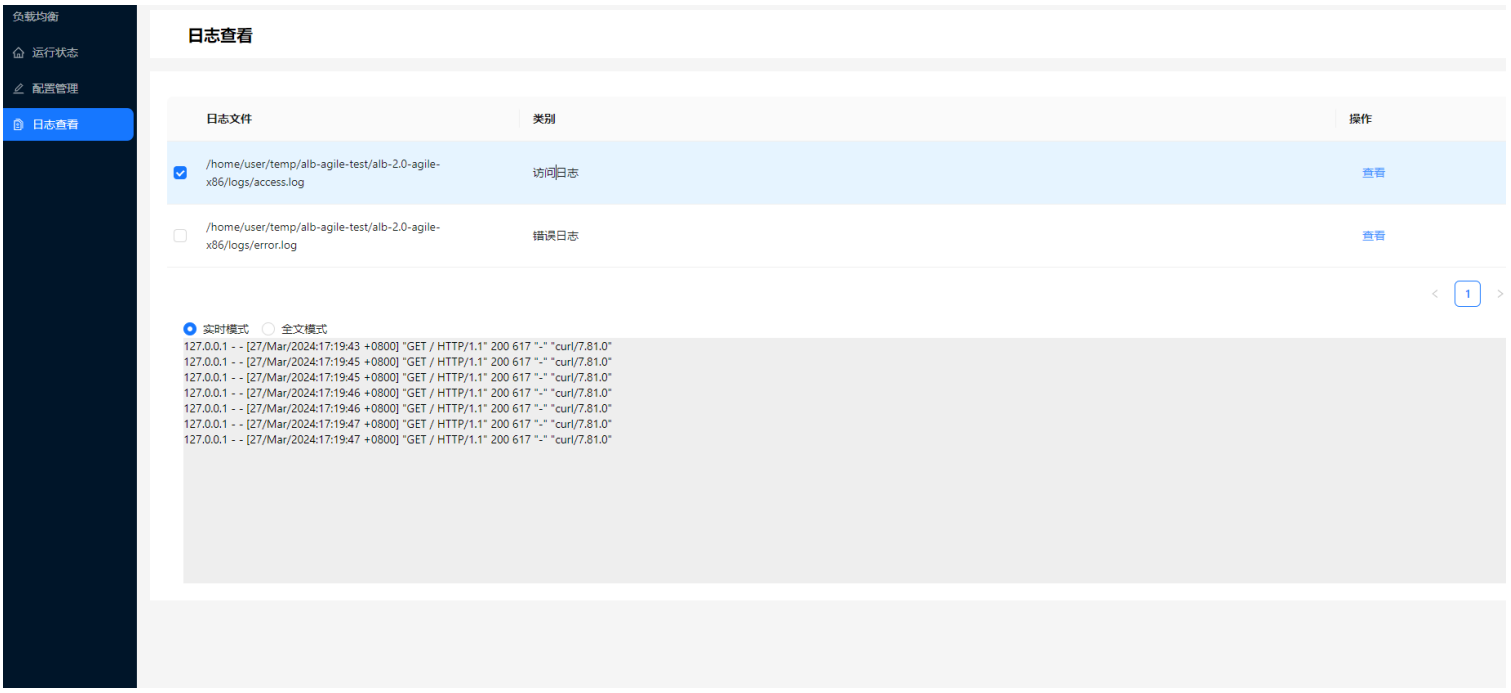
### 3.6.2 WEB管理控制台节点配置管理




- 停止/启动服务： 启停ALB服务
- 重启服务： 快速重启ALB
- 重新加载配置： ALB热更新配置文件内容
- 上传文件： 可上传ALB配置文件，兼容nginx配置文件
- 保存配置： 保存当前编辑内容，但不生效，如需生效，需点击重新加载配置

### 3.6.3 WEB管理控制台节点日志查看

支持查看ALB实时日志，列表展示ALB配置中声明的access日志 和error日志，支持实时模式和全文模式查看日志内容。




- 实时模式： 查看日志文件实时内容
- 全文模式： 获取日志文件全部内容

# 4 产品使用介绍

## 4.1 产品license使用说明

### 4.1.1 授权文件类型支持

alb支持金蝶天燕认证、金蝶KBC认证、金蝶统一授权三种模式，默认为金蝶KBC授权验证。

### 4.1.2 授权类型配置

alb授权文件的配置文件为：`安装目录/conf/alb_license.conf`

```
license local license.xml;  # 本地授权验证，license文件为:安装目录/license.xml
```

**语句格式规范**：`license [授权类型] [授权文件或地址];`

配置语句开头为license，后面跟授权模式和授权文件或地址。

### 4.1.3 不同授权类型及其配置说明

- local: 表示金蝶天燕本地授权，后面需要紧跟授权文件。如：`license local license.xml;`
- kbc: 表示金蝶KBC授权，后面需要紧跟授权文件。如：`license kbc license.lic;`，
  - KBC特征码获取：授权配置文件写：`license kbc;`
  - 执行ALB启动`./bin/start-alb.sh`脚本，获取授权码：`KBC auth: Auth Code is: SZTY2500879438`。
  - 授权码为SZTY开头的内容,如上为：`SZTY2500879438`
  - 使用授权码在KBC系统中申请授权文件。
  - 获取授权文件后，放入安装目录，并更新alb_license.conf配置，重启ALB即可。
  - **注**：在多网卡多ip环境中，可以通过-ac参数指定网卡或ip，如`license kbc license-file.lic -ac eth0`
- center: 表示金蝶天燕统一授权中心，后面需要跟授权服务器地址(IP:端口)、租户名称和命名空间。如：`license center 172.21.33.33:6789 tenant namespace;`
  - 如果租户名称不确定，可以填写为public。

**注意**，更换授权文件后需要重启ALB

### 4.1.4 通过环境变量配置统一授权

alb支持环境变量中设置统一授权的配置，如下关键项：

```
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
```

## 4.2 产品端口修改

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conf/alb.conf文件中的`listen 8080`行。

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

## 4.3 开机启动项

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-agile-2.0-x86              # 进入安装目录
./utils/systemd/enable_startup.sh      # 自动设置开机启动项。
```

## 4.4 ALB服务健康检查

为了保证ALB单节点服务正常，和出现异常后可以自动重启ALB，可以开启ALB的健康检查服务：

```
cd /opt/alb-agile-2.0-x86                              # 进入安装目录
./utils/healthy_check/enable_alb_health_check.sh # 开启自动健康检查
```

### 4.4.1 ALB健康检查日志

健康检查日志文件：`安装目录/utils/healthy_check/alb_health_check.log`

健康检查日志标记了ALB服务重启时间。

## 4.5 ALB日志自动切割

ALB日志默认不自动切割，但当实时流量过大，导致日志文件过大，则需开启自动切割。

ALB提供日志自动切割脚本(utils/log_rotate目录下)，默认保留30天日志，如果需要修改保留日志的天数，请修改alb_logrotate文件中：rotate 30的30为需要保留的天数

#### 4.5.1 开启日志切割

- 进入alb安装目录下utils/log_rotate文件夹
- 执行: `./init_logrotate.sh`
- 出现： `please input alb log backup dir` 时，可以输入日志文件切割备份保存的目标文件夹，如果不改变（默认等待10秒），则默认备份到alb日志文件夹。
- 出现： `register logrotate successed` 的时候，表示注册成功。

#### 4.5.2 日志切割功能取消

- 进入alb安装目录下utils/log_rotate文件夹
- 执行: `./remove_logrotate.sh`

## 4.6 负载均衡算法

#### 4.6.1 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;
}
```

#### 4.6.2 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;
}
```

#### 4.6.3 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;
}
```

#### 4.6.4 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;
}
```

## 4.7 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。

#### 4.7.1 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，Nginx 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，Nginx 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。

### 4.7.2 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特定URL发送HTTP请求来检测其是否健康。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。

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。

#### 4.7.2.1 不同类型说明

1. TCP 健康检查 (type=tcp)

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;

        # 发送空的 TCP 数据包
        check_send "";
        check_expect "";
    }

    server {
        listen 80;
        server_name example.com;

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

```
    }
}
```

2. HTTP 健康检查 (type=http)

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;
        }
    }
}
```

## 4.8 监控

ALB提供丰富的监控指标，包括提供简单流量统计数据、Prometheus监控数据和每一项配置流量数据等。

### 4.8.1 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。

#### 4.8.1.1 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。

### 4.8.2 Prometheus监控

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。

#### 4.8.2.1 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-agile-2.0/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)
```

#### 4.8.2.2 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。

### 4.8.3 使用vts模块监控

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。

#### 4.8.3.1 HTTP监控指标说明

1、基本信息 应支持对服务代理基本信息监控，监控指标包括：

- 主机信息，如代理服务器所在的主机名或IP地址
- 版本号
- 服务器运行时间

2、连接信息

- 当前客户端的连接数
- 读取客户端连接的总数
- 写入客户端连接的总数
- 已处理客户端的连接总数

3、HTTP虚拟主机监控

- 请求总数。
- 每秒请求的数量
- 1xx错误码的统计
- 2xx错误码的统计

- 3xx错误码的统计
- 4xx错误码的统计
- 5xx错误码的统计
- 所有错误码的统计
- 发送的总流量
- 接收的总流量
- 发送的流量速率
- 接受的流量速率

4、HTTP缓存监控

- 缓存未命中数
- 缓存旁路数
- 缓存已过期数
- 缓存失效数
- 缓存更新数
- 重新验证的缓存数
- 缓存命中数
- 缓存总数

5、HTTP上游服务器监控

- 服务状态
- 设置的服务权重值。
- 设置的服务最大失败次数值。
- 设置的服务故障后停止服务的时间值。
- 请求总数
- 每秒请求的数量
- 1xx错误码的统计
- 2xx错误码的统计
- 3xx错误码的统计

- 4xx错误码的统计
- 5xx错误码的统计
- 所有错误码的统计
- 发送的总流量
- 接收的总流量
- 发送的流量速率
- 接受的流量速率

6、Stream虚拟主机监控

- 请求总数。
- 每秒请求的数量。
- 1xx错误码的统计
- 2xx错误码的统计
- 3xx错误码的统计
- 4xx错误码的统计
- 5xx错误码的统计
- 所有错误码的统计
- 发送的总流量
- 接收的总流量
- 发送的流量速率
- 接受的流量速率

7、Stream上游服务器监控

- 服务状态
- 设置的服务权重值。
- 设置的服务最大失败次数值。

- 设置的服务故障后停止服务的时间值。
- 请求总数
- 每秒请求的数量
- 1xx错误码的统计
- 2xx错误码的统计
- 3xx错误码的统计
- 4xx错误码的统计
- 5xx错误码的统计
- 所有错误码的统计
- 发送的总流量
- 接收的总流量
- 发送的流量速率
- 接受的流量速率

#### 4.8.3.2 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`

#### 4.8.3.3 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

#### 4.8.3.4 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`

## 4.9 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：

```
stream {
    upstream backend {
        server 127.0.0.1:8080;
    }

    server {
        listen 443;    # 监听端口
```

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

## 4.10 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。

### 4.10.1 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。

#### 4.10.1.1 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。

#### 4.10.1.2 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/nginx/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。

#### 4.10.1.3 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。

#### 4.10.1.4 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;
```

### 4.10.2 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。

#### 4.10.2.1 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。

#### 4.10.2.2 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。

#### 4.10.2.3 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。

#### 4.10.3 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。你可以使用 logrotate 或 cron 来实现日志切割。

##### 4.10.3.1 使用 logrotate

```
vim /etc/logrotate.d/alb
```

在 /etc/logrotate.d/alb 文件中配置日志切割规则：

```
/path/to/alb-access.log {
    daily
    rotate 14
    compress
    missingok
    notifempty
}
```

- /path/to/alb-access.log: 标识具体的日志文件路径。
- daily: 每天执行一次切割操作。
- rotate 14: 保持最多 14 天的日志文件。
- compress: 压缩旧的日志文件。
- missingok: 如果日志文件不存在，不报错继续执行。

注：

- 根据服务器的具体需求调整日志配置。
- 考虑到性能因素，不要在生产环境中记录过多不必要的信息。
- 定期审查日志文件，以确保它们不占用过多磁盘空间。

配置好后，可以执行 logrotate 命令来立即切割日志：

```
logrotate /etc/logrotate.d/alb
```

#### 4.10.3.2 使用 cron

如果没有安装 logrotate，可以使用 cron和shell脚本来定期执行日志切割。

注： /path/to/alb-logrotate.sh的path/to请根据需要修改。

1、创建一个 shell 脚本，例如 `alb-logrotate.sh`：

```
#!/bin/bash

# 日志文件的位置
LOG_DIR="/path/to/logs"

# 日志文件的名称
LOG_FILE="alb-access.log"

# 新日志文件的名称
NEW_LOG_FILE="${LOG_FILE}.new"

# 老日志文件的名称
OLD_LOG_FILE="${LOG_FILE}.old"

# 日志文件的完整路径
LOG_PATH="${LOG_DIR}/${LOG_FILE}"
NEW_LOG_PATH="${LOG_DIR}/${NEW_LOG_FILE}"
OLD_LOG_PATH="${LOG_DIR}/${OLD_LOG_FILE}"

# 清空新的日志文件
> ${NEW_LOG_PATH}

# 将当前日志文件重命名为老日志文件
mv ${LOG_PATH} ${OLD_LOG_PATH}

# 将新的日志文件重命名为当前日志文件
mv ${NEW_LOG_PATH} ${LOG_PATH}
```

```
# 清理旧日志文件
find ${LOG_DIR} -name "${LOG_FILE}.*" -mtime +14 -exec rm -f {} \;

# 通知 Nginx 重新打开日志文件
killall -USR1 alb
```

这个脚本做了以下几件事：

- 创建一个新的空日志文件。
- 将当前的日志文件重命名为老日志文件。
- 将新的日志文件重命名为当前日志文件。
- 清理超过 14 天的老日志文件。
- 通知 alb 重新打开日志文件。

2、设置脚本权限

```
chmod +x /path/to/alb-logrotate.sh
```

3、添加 cron 任务

```
crontab -e
```

在文件末尾添加一行来指定日志切割脚本的执行时间：

```
0 1 * * * /path/to/alb-logrotate.sh
```

这行 cron 任务表示每天凌晨 1 点执行日志切割脚本。

3、重启 cron 服务

```
service cron restart
```

这样，每天的凌晨 1 点，就会自动执行日志切割脚本。

## 4.11 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。

### 4.11.1 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。

#### 4.11.1.1 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。

### 4.11.2 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。

#### 4.11.2.1 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。

### 4.11.3 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）。

#### 4.11.3.1 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。

## 4.12 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。

### 4.12.1 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。

# 5 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。

## 5.1 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。

## 5.2 原生高可用

ALB自带高可用组件，可一键部署和启动ALB。

### 5.2.1 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。

### 5.2.2 配置alb-ha

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

#### 5.2.2.1 主节点

1. 切换至utils/alb-ha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

alb-checker:
```

```
  # 检查alb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

  # 当前alb状态检查url，用于检查alb是否健康，要求返回200状态码
  health-url: http://localhost:8080/node_status
```

#### 5.2.2.2 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

alb-checker:

  # 检查alb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

  # 当前alb状态检查url，用于检查alb是否健康，要求返回200状态码
  health-url: http://localhost:8080/node_status
```

#### 5.2.3 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. 执行：`./setup-alb-ha-service.sh` 自动注册系统服务
3. 启动：`systemctl start alb_ha.servcie`

#### 5.2.4 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。

#### 5.2.5 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 即可

# 5.3 使用keepalived实现高可用

### 5.3.1 安装 keepalived

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

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

### 5.3.2 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。

### 5.3.3 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" |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" |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
```

#### 5.3.4 主备节点均启动keepalived服务

```
systemctl start keepalived.service
```

#### 5.3.5 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}'
```

#### 5.3.6 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）：

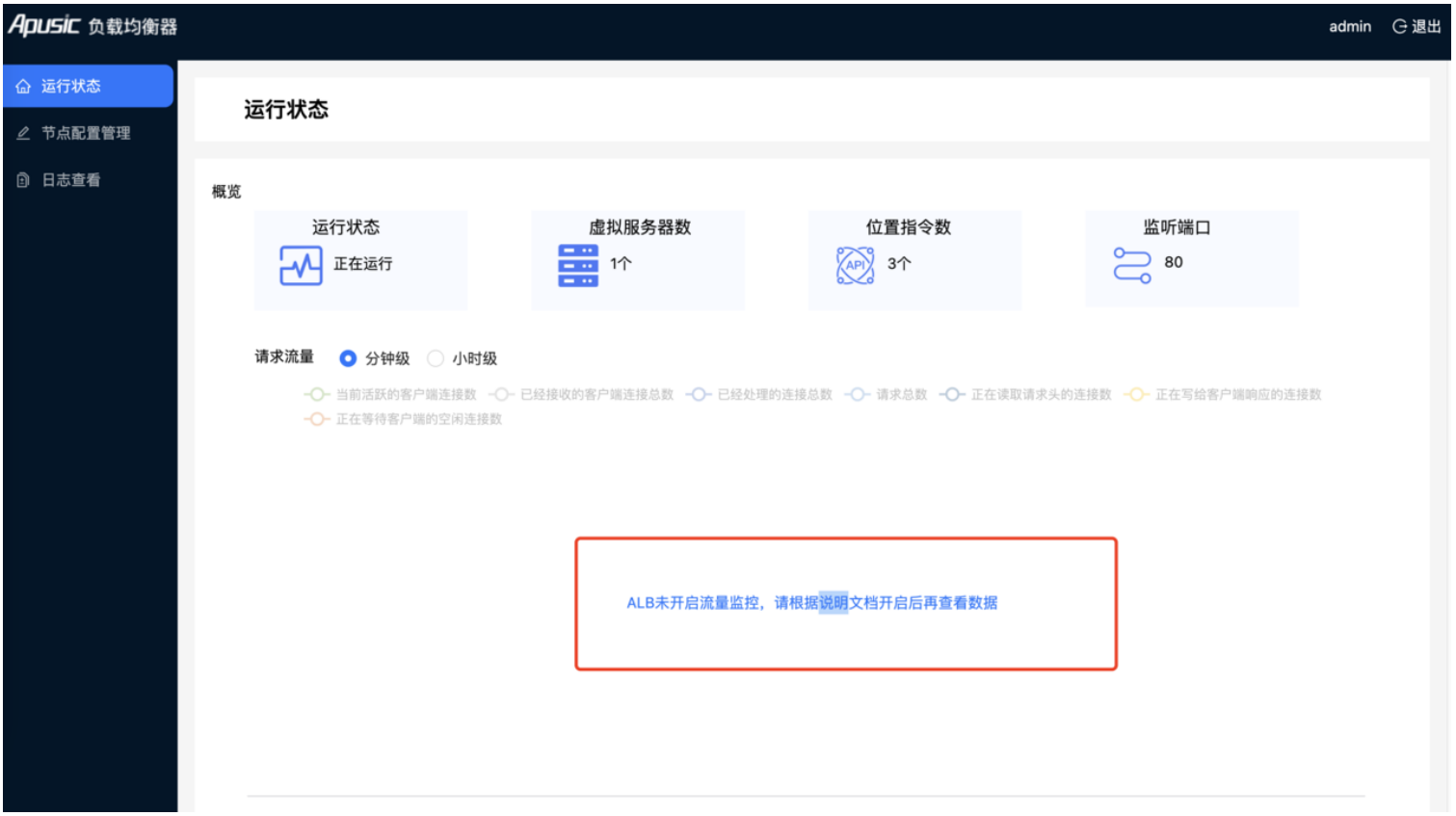
```
ip addr show eth0 |grep inet|grep -v inet6|awk '{print $2}'
```

如果VIP指向主节点，则说明keepalived配置成功。

# 6 产品常见问题

## 6.1 管控台开启运行流量监控

启动ALB敏捷版登录管控台后，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

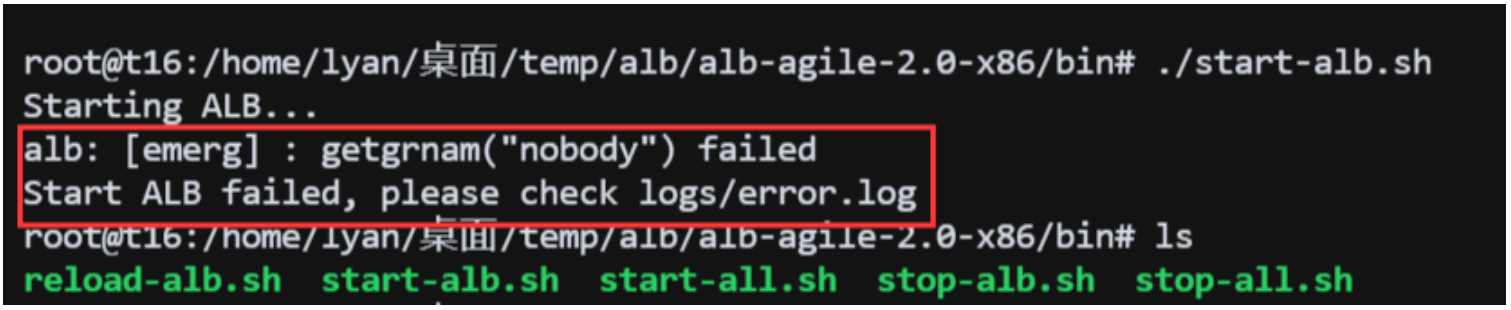
```
server {

    # 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。

## 6.2 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;
```

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