



APUSIC
固若长城
睿比世界

代理功能手冊

金蝶Apusic負載均衡器

版權所有 © 深圳市金蝶天燕雲計算股份有限公司2026。保留所有權利。

版權聲明

本文件所涉及的軟件著作權、版權等知識產權已依法進行了註冊，由金蝶天燕雲計算股份有限公司合法擁有。受《中華人民共和國著作權法》《計算機軟件保護條例》《知識產權保護條例》和相關國際版權條約、法律、法規以及其它知識產權法律和條約的保護。未經授權許可，不得非法使用。

免責聲明

本文件包含的版權資訊由金蝶天燕雲計算股份有限公司合法擁有，受法律的保護，金蝶天燕雲計算股份有限公司對本文件可能涉及到的非金蝶天燕雲計算股份有限公司的資訊不承擔任何責任。在法律允許的範圍內，您可以查閱並僅能夠在《中華人民共和國著作權法》規定的合法範圍內複製和打印本文件。任何單位和個人未經金蝶天燕雲計算股份有限公司書面授權許可，不得使用、修改、再發佈本文件的任何部分和內容，否則將被視為侵權，金蝶天燕雲計算股份有限公司有依法追究其責任的權利。

本文件如有更新，不另行通知。對本文件中的問題您可向金蝶天燕雲計算股份有限公司告知或查詢。未經本公司明確授予的任何權利均予保留。

商標聲明

 是深圳市金蝶天燕雲計算股份有限公司向中華人民共和國國家商標局申請註冊的註冊商標，註冊商標專用權由金蝶天燕合法擁有，受法律保護。未經金蝶天燕的書面許可，任何單位及個人不得以任何方式或理由對該商標的任何部分進行使用、複製、修改、傳播、抄錄或與其它產品捆綁使用銷售。凡侵犯金蝶天燕商標權的，金蝶天燕將依法追究其法律責任。本文件提及的其他所有商標或註冊商標，由各自的所有人擁有。

目錄

- 1 前言
 - 1.1 適用對象
 - 1.2 相關文件
 - 1.3 技術支援
- 2 功能模組清單
 - 2.1 ALB功能模組清單
- 3 負載均衡演算法
 - 3.1 輪詢
 - 3.2 加權輪詢
 - 3.3 IP雜湊
 - 3.4 最少連線數
- 4 健康檢查
 - 4.0.1 被動健康檢查
 - 4.0.2 主動健康檢查
 - 4.0.2.1 七層HTTP代理健康檢查
 - 4.0.2.2 四層TCP代理健康檢查
 - 4.0.2.3 不同類型說明
 - 4.0.2.4 主動健康檢查狀態獲取
- 5 監控
 - 5.0.1 簡單流量統計資料
 - 5.0.1.1 設定簡單流量統計資料和獲取
 - 5.0.2 Prometheus-exporter (Prometheus監控資料)
 - 5.0.2.1 啟動說明
 - 5.0.2.2 Prometheus監控資料獲取
 - 5.0.3 VTS監控資料
 - 5.0.3.1 監控指標說明
 - 5.0.3.2 vts模組設定和查看
 - 5.0.3.3 自訂監控指標
 - 5.0.3.4 stream監控
 - 5.0.3.5 VTS監控指標grafana整合
- 6 TCP代理
- 7 TCP代理加速

- 7.1 核心eBPF加速TCP轉發優勢
 - 7.1.1 設定要求
 - 7.1.2 使用範例
 - 7.1.3 設定參數說明
 - 7.1.3.1 ebpf_status_return
 - 7.1.3.2 ebpf_enable
 - 7.1.3.3 ebpf_proxy_timeout
 - 7.1.3.4 ebpf_proxy_buffer_size
 - 7.1.3.5 ebpf_timer_period
- 8 TCP代理ADMQ
 - 8.1 設定過程
 - 8.2 ADMQ環境和代理需求
 - 8.3 修改ALB設定，支援TCP代理ADMQ
 - 8.4 驗證ALB代理ADMQ服務
- 9 串流回應代理
 - 9.0.1 設定詳解
- 10 日誌設定
 - 10.0.1 存取日誌
 - 10.0.1.1 存取日誌設定
 - 10.0.1.2 基本設定範例
 - 10.0.1.3 日誌格式詳解
 - 10.0.1.4 關閉存取日誌
 - 10.0.2 錯誤日誌
 - 10.0.2.1 設定錯誤日誌
 - 10.0.2.2 基本設定
 - 10.0.2.3 日誌層級
- 11 流量控制
 - 11.0.1 使用 limit_req 模組進行限流
 - 11.0.1.1 範例設定
 - 11.0.2 使用 limit_conn 模組進行連線數限制
 - 11.0.2.1 範例設定
 - 11.0.3 使用 limit_rate 指令進行限速
 - 11.0.3.1 範例設定
- 12 灰度發佈和灰度測試

- 12.1 場景介紹
- 12.2 ALB關鍵指令速查
- 12.3 關鍵指令詳解
 - 12.3.1 1. split_clients —— 按比例分流（推薦）
 - 12.3.2 2. map —— 基於規則路由
 - 12.3.3 3. upstream —— 定義後端服務群組
- 12.4 設定範例：按IP位址和比例灰度發佈
- 12.5 基於Cookie的灰度
- 12.6 基於 IP 白名單強制灰度（內部測試解決方案）
- 12.7 動態控制灰度比例或基於資料庫使用者標籤分流
- 13 動態 DNS 解析
 - 13.1 resolver 設定詳解
- 14 HTTPS 與國密
 - 14.1 前置準備
 - 14.2 編輯 ALB 設定檔，開啟 SSL
 - 14.2.1 設定指令說明
 - 14.2.1.1 listen
 - 14.2.1.2 ssl_certificate 和 ssl_certificate_key
 - 14.2.1.3 ssl_protocols
 - 14.2.1.4 ssl_ciphers
 - 14.2.1.5 ssl_prefer_server_ciphers
 - 14.2.1.6 ssl_session_cache
 - 14.2.1.7 ssl_session_timeout
 - 14.2.2 國密設定範例
 - 14.3 HTTP/3
- 15 正向代理
 - 15.1 範例設定
- 16 快取設定
 - 16.1 基礎設定
 - 16.2 關鍵指令說明
 - 16.3 真實使用場景
 - 16.3.1 場景一：商品詳情頁快取
 - 16.3.2 場景二：API 開道回應快取 + 後端降級
 - 16.4 注意事項

- 17 工作階段保持
 - 17.1 方式一：基於 IP 雜湊
 - 17.2 方式二：基於 Cookie 一致性雜湊（推薦）
 - 17.3 方式三：基於 Lua + Redis 自訂工作階段保持
 - 17.4 方式四：基於 sticky 模組（推薦）
 - 17.4.1 1. cookie 模式
 - 17.4.2 2. route 模式
 - 17.4.3 3. learn 模式
 - 17.5 關鍵指令說明
 - 17.6 真實使用場景
 - 17.6.1 場景一：電商登入狀態保持
 - 17.6.2 場景二：WebSocket 長連線保持
 - 17.6.3 場景三：Java 後端 sticky 模式無縫接入
 - 17.7 注意事項
- 18 壓縮
 - 18.1 Gzip 壓縮
 - 18.2 關鍵指令說明
 - 18.3 Brotli 壓縮（需 ALB 編譯 alb_brotli 模組）
 - 18.4 預壓縮檔案（gzip_static / brotli_static）
 - 18.5 真實使用場景
 - 18.5.1 場景一：API 回應壓縮
 - 18.5.2 場景二：靜態資源雙壓縮（Gzip + Brotli）
 - 18.6 注意事項
- 19 gRPC 代理
 - 19.1 基礎設定
 - 19.1.1 HTTPS 加密代理
 - 19.2 關鍵指令說明
 - 19.3 真實使用場景
 - 19.3.1 場景一：微服務 gRPC 叢集統一入口
 - 19.3.2 場景二：gRPC 反射與健康檢查
 - 19.4 注意事項
- 20 WebSocket 代理
 - 20.1 基礎設定
 - 20.2 與 HTTPS 共用 443 連接埠

- 20.3 關鍵指令說明
- 20.4 真實使用場景
 - 20.4.1 場景一：線上客服系統 WebSocket 代理
 - 20.4.2 場景二：即時資料推送（股票行情、K 線）
- 20.5 注意事項
- 21 URL 重寫與重新導向
 - 21.1 基礎設定
 - 21.2 關鍵指令說明
 - 21.2.0.1 rewrite 旗標位
 - 21.2.0.2 return 常用狀態碼
 - 21.3 真實使用場景
 - 21.3.1 場景一：舊 API 路徑遷移
 - 21.3.2 場景二：HTTP 強制跳轉 HTTPS
 - 21.3.2.1 場景三：SPA 前端路由兜底
 - 21.3.2.2 場景四：API 版本標頭透傳與路徑重寫
 - 21.4 注意事項
- 22 跨域設定（CORS）
 - 22.0.1 基礎設定
 - 22.1 關鍵指令說明
 - 22.2 真實使用場景
 - 22.2.0.1 場景一：多來源白名單
 - 22.2.0.2 場景二：全開放（僅適用於內部系統/開發環境）
 - 22.3 注意事項
- 23 故障轉移與重試
 - 23.0.1 基礎設定
 - 23.1 關鍵指令說明
 - 23.2 真實使用場景
 - 23.2.0.1 場景一：主備災備
 - 23.2.0.2 場景二：節點維護摘除
 - 23.2.0.3 場景三：可重試與不可重試請求區分
 - 23.3 注意事項
- 24 使用者場景設定案例
 - 24.1 概述
 - 24.2 場景一：前端單頁應用託管與路由支援/靜態資源發佈

- 24.2.1 1. 背景
- 24.2.2 2. 遇到的問題與待解決的問題
- 24.2.3 3. ALB 功能與設定與問題比對
- 24.2.4 4. 使用 ALB 解決該場景下的設定與詳解
- 24.2.5 5. 效果，前後對比
- 24.3 場景二：微服務 API 統一入口與負載平衡
 - 24.3.1 1. 背景
 - 24.3.2 2. 遇到的問題與待解決的問題
 - 24.3.3 3. ALB 功能與設定與問題比對
 - 24.3.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.3.5 5. 效果，前後對比
- 24.4 場景三：全站 HTTPS 強制跳轉與 SSL 終止
 - 24.4.1 1. 背景
 - 24.4.2 2. 遇到的問題與待解決的問題
 - 24.4.3 3. ALB 功能與設定與問題比對
 - 24.4.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.4.5 5. 效果，前後對比
- 24.5 場景四：API 介面防刷與速率限制
 - 24.5.1 1. 背景
 - 24.5.2 2. 遇到的問題與待解決的問題
 - 24.5.3 3. ALB 功能與設定與問題比對
 - 24.5.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.5.5 5. 效果，前後對比
- 24.6 場景五：動靜分離與動態內容快取
 - 24.6.1 1. 背景
 - 24.6.2 2. 遇到的問題與待解決的問題
 - 24.6.3 3. ALB 功能與設定與問題比對
 - 24.6.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.6.5 5. 效果，前後對比
- 24.7 場景六：WebSocket 長連線代理
 - 24.7.1 1. 背景
 - 24.7.2 2. 遇到的問題與待解決的問題
 - 24.7.3 3. ALB 功能與設定與問題比對
 - 24.7.4 4. 使用 ALB 解決該場景下的設定與詳解

- 24.7.5 5. 效果，前後對比
- 24.8 場景七：防止惡意掃描與目錄遍歷攻擊
 - 24.8.1 1. 背景
 - 24.8.2 2. 遇到的問題與待解決的問題
 - 24.8.3 3. ALB 功能與設定與問題比對
 - 24.8.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.8.5 5. 效果，前後對比
- 24.9 場景八：基於 Cookie 的灰度發佈（A/B 測試）
 - 24.9.1 1. 背景
 - 24.9.2 2. 遇到的問題與待解決的問題
 - 24.9.3 3. ALB 功能與設定與問題比對
 - 24.9.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.9.5 5. 效果，前後對比
- 24.10 場景九：大檔案上傳逾時與分片支援最佳化
 - 24.10.1 1. 背景
 - 24.10.2 2. 遇到的問題與待解決的問題
 - 24.10.3 3. ALB 功能與設定與問題比對
 - 24.10.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.10.5 5. 效果，前後對比
- 24.11 場景十：靜態資源 Gzip 壓縮加速
 - 24.11.1 1. 背景
 - 24.11.2 2. 遇到的問題與待解決的問題
 - 24.11.3 3. ALB 功能與設定與問題比對
 - 24.11.4 4. 使用 ALB 解決該場景下的設定與詳解
 - 24.11.5 5. 效果，前後對比

1 前言

本文件為金蝶Apusic負載均衡器軟體V2.0.6標準版的代理功能手冊，詳細介紹ALB代理相關功能的使用、設定及管理方法，包括負載均衡、健康檢查、流量控制、灰度發布、TCP/UDP代理、HTTPS與國密、HTTP/3、正向代理等。

1.1 適用對象

本文件適用於IT資訊化業務負責人、研發經理、軟體專案經理、軟體架構師、運維工程師。

1.2 相關文件

了解更多ALB V2.0.6產品相關的資訊，請參閱以下ALB V2.0.6產品手冊文件集：

序號	手冊文件	說明
1	金蝶Apusic負載均衡器軟體 V2.0.6 發版說明	介紹了ALB 產品版本更新內容。
2	金蝶Apusic負載均衡器軟體 V2.0.6 快速使用手冊	簡單介紹了如何快速上手使用ALB。
3	金蝶Apusic負載均衡器軟體 V2.0.6 安裝手冊	詳細介紹如何在各作業系統上安裝ALB，以及ALB服務啟停操作，產品的註冊過程。
4	金蝶Apusic負載均衡器軟體 V2.0.6 代理功能手冊	詳細介紹 ALB 相關功能的使用、設定、管理及配套工具的使用方法。
5	金蝶Apusic負載均衡器軟體 V2.0.6 管控台使用者手冊	詳細介紹ALB管控台相關功能的使用和操作說明。
6	金蝶Apusic負載均衡器軟體 V2.0.6 開發手冊	詳細介紹ALB外部API、alb-cli命令列工具及AI Agent整合方法。
7	金蝶Apusic負載均衡器軟體 V2.0.6 遷移手冊	詳細介紹ALB歷史版本遷移升級到V2.0.6版本的說明。
8	金蝶Apusic負載均衡器軟體 V2.0.6 維運手冊	詳細介紹ALB的監控、維運、安全加固等維運說明。
9	金蝶Apusic負載均衡器軟體 V2.0.6 效能最佳化手冊	詳細介紹ALB效能調校的說明。

1.3 技術支援

ALB產品提供全面的技術支援服務，您可以透過以下方式獲得技術支援：

- 網址：www.apusic.com
- 電話：400-855-5800
- 電子郵件：support@apusic.com
- 金蝶雲社群：<https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN>

您在取得技術支援時，請提供如下資訊：

1. 您的姓名
2. 公司資訊與聯絡方式
3. 作業系統及其版本
4. 產品版本號
5. 出現異常及錯誤的日誌、截圖等詳細資訊

2 功能模組清單

2.1 ALB功能模組清單

模組	功能說明
--with-http_stub_status_module	啟用 stub_status 介面，提供運行狀態統計資訊。
--with-http_ssl_module	啟用 HTTPS/SSL 支援。
--with-ipv6	啟用 IPv6 監聽支援。
--with-http_mp4_module	支援 MP4 串流媒體偽串流（pseudo-streaming）。
--with-http_auth_request_module	實作子請求認證（auth_request），方便與外部認證服務整合。
--with-http_gzip_static_module	支援直接傳送 .gz 預壓縮檔案，節省 CPU。
--with-http_realip_module	從 X-Forwarded-For/X-Real-IP 等標頭中取得真實用戶端 IP。
--with-http_gunzip_module	對不支援 gzip 的用戶端自動解壓回應內容。
--with-http_sub_module	回應主體字串替換（sub_filter）。
--with-stream	啟用 4 層 TCP/UDP 代理（stream 子系統）。
--with-stream_ssl_module	為 stream 子系統提供 SSL/TLS 支援。
--with-threads	開啟執行緒池支援，提升檔案 I/O 效能。
--with-file-aio	啟用非同步檔案 I/O（Linux AIO）。
--with-http_v2_module	啟用 HTTP/2 支援。
--with-http_v3_module	啟用 HTTP/3（QUIC）支援。
--with-http_addition_module	在回應前後追加內容（add_before_body / add_after_body）。
--with-http_dav_module	支援 WebDAV 協定（PUT、DELETE、MKCOL 等）。
--with-http_random_index_module	目錄索引時隨機選擇預設檔案。
--with-http_secure_link_module	產生和校驗帶時效、防篡改的「安全連結」。
--with-mail	啟用郵件代理（IMAP/POP3/SMTP）子系統。
--with-mail_ssl_module	為郵件代理提供 SSL/TLS 支援。

<code>--with-http_slice_module</code>	支援大檔案分片下載 (Range Slice) 。
<code>--with-stream_ssl_preread_module</code>	在 4 層代理裡預讀 SNI/ALPN，實現按域名路由。
<code>alb_healthcheck_module</code>	ALB主動健康檢查模組，支援四層和七層代理節點主動探活。
<code>alb-module-vts</code>	提供詳細流量 / 狀態統計介面 (<code>/status</code>) 。
<code>alb-module-stream-sts</code>	為 stream 子系統提供即時狀態統計。
<code>alb-module-sts</code>	為 http 子系統提供即時狀態統計。
<code>alb_http_proxy_connect_module</code>	支援 HTTP CONNECT 方法，正向代理 HTTPS 流量。
<code>alb-rtmp-module</code>	引入 RTMP/HLS/DASH 直播與串流媒體功能。
<code>alb_brotli</code>	支援 Brotli 壓縮演算法，減少傳輸體積。
<code>alb-module-lua</code>	引入 Lua 腳本功能，可擴充 ALB 功能。
<code>alb-headers-more</code>	設定和清除-請求和回應標頭。

3 負載均衡演算法

3.1 輪詢

輪詢是ALB預設的負載均衡演算法，按時間順序將請求輪流分配給後端伺服器。如果有伺服器宕機，ALB會自動將其剔除。

如下設定多個後端伺服器後，預設使用輪詢負載均衡演算法。

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

3.2 加權輪詢

在輪詢的基礎上，根據指定的權重分配請求，權重越高的伺服器處理的請求越多，適用於伺服器效能不均的情況。

如下設定多個後端伺服器後，透過使用weight參數指定權重。

```
upstream backend {  
    server backend1.example.com weight=10; # 透過weight參數設定權重  
    server backend2.example.com weight=20;  
}
```

3.3 IP雜湊

根據用戶端的IP位址進行雜湊運算，將請求分配給後端伺服器。可以保證來自同一IP位址的請求始終被分配到同一台後端伺服器。

如下設定多個後端伺服器後，透過使用ip_hash指令啟用IP Hash負載均衡演算法。

```
upstream backend {  
    ip_hash; # 開啟IP雜湊負載均衡  
    server backend1.example.com;
```

```
server backend2.example.com;  
}
```

3.4 最少連線數

根據後端伺服器的目前活動連線數量進行分配，將請求分配給活動連線最少的後端伺服器。適合請求處理時間差異較大的場景

如下設定多個後端伺服器後，透過使用least_conn指令啟用最少連線數負載均衡演算法。

```
upstream backend {  
    least_conn; # 開啟最少連線數負載均衡  
    server backend1.example.com;  
    server backend2.example.com;  
}
```

4 健康檢查

ALB支援對後端伺服器進行健康檢查，當後端伺服器的回應不符合預期時，ALB會將其剔除。

4.0.1 被動健康檢查

被動健康檢查由ALB自動執行，根據後端伺服器的回應狀態碼和逾時時間來判斷其是否健康。

ALB的被動健康檢查透過 `fail_timeout` 和 `max_fails` 參數進行設定。

- `fail_timeout`：指定在連續失敗後，ALB將後端伺服器標記為不健康的時間段。預設值為10秒。
- `max_fails`：指定在 `fail_timeout` 時間段內，連續失敗的次數。預設值為3次。

例如，如果你希望在某個後端伺服器連續失敗 3 次後將其標記為不可用，並在接下來的 30 秒內不向其發送請求，你可以這樣設定

```
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;
}
```

註：

- 如果後端伺服器在 `fail_timeout` 設定的時間結束後仍然不可用，ALB 會再次嘗試向該伺服器發送請求。
- 當後端伺服器恢復正常後，ALB 會自動將其重新加入到輪詢佇列中。
- 使用這些指令可以幫助減少由於網路波動等原因造成的短暫失敗對使用者體驗的影響。

4.0.2 主動健康檢查

主動健康檢查由ALB定期執行，透過向後端伺服器發送特定的請求判斷是否健康。如果請求成功返回，則認為後端伺服器是健康的；否則，將其標記為不健康。

4.0.2.1 七層HTTP代理健康檢查

ALB的主動健康檢查透過 `check` 指令進行設定。

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

    # 啟用健康檢查
```

```

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

# 定義健康檢查的 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;
}

```

解釋設定指令：

- check：開啟健康檢查，設定健康檢查的基本參數。
- interval：定義兩次檢查之間的時間（單位毫秒）。
- rise：定義連續幾次成功回應後認為後端伺服器是健康的。
- fall：定義連續幾次失敗回應後認為後端伺服器是不健康的。
- timeout：定義單次健康檢查請求的逾時時間（單位毫秒）。
- type：定義健康檢查請求的類型，如 http 或 tcp。
- check_http_send：定義發送給後端伺服器的 HTTP 請求。
- 這裡使用了一個 GET 請求來檢查 /healthcheck 路徑。
- check_http_expect_alive：定義期望從後端伺服器收到的成功回應的狀態碼範圍。
- http_2xx 和 http_3xx 表示期望收到 2xx 和 3xx 狀態碼。

4.0.2.2 四層TCP代理健康檢查

ALB主動健康檢查支援TCP代理，以下是範例：

```

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; # 範例DNS伺服器
check interval=3000 rise=2 fall=5 timeout=5000
default_down=true type=udp;
}
server {
    listen 53 udp;
    proxy_pass udp-cluster;
}
}

```

4.0.2.3 不同類型說明

1. TCP 健康檢查 (type=tcp), 適用於所有 TCP 後端服務

TCP 健康檢查是最簡單的類型之一，它僅需要建立一個 TCP 連線到後端伺服器，並讀取一個位元組來判斷伺服器是否可連通。如下為實例設定

```

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 健康檢查 (type=http), 主要用於七層 HTTP 代理

HTTP 健康檢查透過發送一個 HTTP 請求到後端伺服器來判斷伺服器的狀態。如下為實例設定

```
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;

        # 發送一個 GET 請求到 /healthcheck 路徑
        check_http_send "GET /healthcheck HTTP/1.0\r\nHost:
www.example.com\r\n\r\n";

        # 期望伺服器返回 2xx 或 3xx 狀態碼表示健康
        check_http_expect_alive http_2xx http_3xx;
    }

    server {
        listen 80;
        server_name example.com;

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

3. SSL Hello 健康檢查 (type=ssl_hello)

SSL Hello 健康檢查透過發送 SSL 用戶端 Hello 封包，並接收伺服器的 SSL Hello 封包來判斷伺服器是否健康。如下為實例設定

```
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 健康檢查 (type=mysql)

MySQL 健康檢查透過建立 MySQL 連線並接收問候回應來判斷資料庫伺服器是否健康。

```

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 健康檢查 (type=fastcgi)

FastCGI 健康檢查透過發送一個 FastCGI 請求，並接收回應來判斷伺服器是否健康。

```

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 健康檢查 (type=udp)，僅支援四層代理

UDP 健康檢查透過發送一個 UDP 請求，並接收回應來判斷伺服器是否健康。

```

stream {
    upstream udp-cluster {
        server 127.0.0.1:53;
        server 8.8.8.8:53; # 範例DNS伺服器
        check interval=3000 rise=2 fall=5 timeout=5000
default_down=true type=udp;
    }

    server {

```

```

        listen 53 udp;
        proxy_pass udp-cluster;
    }
}

```

4.0.2.4 主動健康檢查狀態獲取

ALB支援透過指定介面獲取主動健康檢查狀態，如下：

1. 開啟主動健康檢查狀態獲取

```

# 省略其他設定

http {
    server {
        listen 8080;

        # 獲取主動健康檢查狀態URL
        # 透過ALB的8080埠的/status介面獲取健康狀態資料頁面
        location /status {
            healthcheck_status html;
        }
    }
    # 省略其他設定
}

```

存取8080埠/status介面，即可獲取健康狀態資料頁面：

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. 支援類型

- html：返回HTML頁面
- json：返回JSON資料
- csv：返回CSV資料
- prometheus：返回Prometheus格式資料

5 監控

ALB提供豐富的監控指標，包括：

1. 簡單流量統計資料：提供基礎連線和請求統計，不支援Prometheus採集格式。
2. Prometheus-exporter：支援將流量資料以Prometheus格式暴露，便於與Prometheus監控系統整合。
3. VTS監控資料：提供ALB精細化監控資料，支援HTML頁面展示、JSON資料格式和Prometheus採集格式，功能最為全面。

註：上面三種方式可按需選擇其中一種。

5.0.1 簡單流量統計資料

ALB提供簡單的流量統計資訊，其監控資料有：

- Active connections: 目前活躍的連線數。
- Server accepts, handled, requests: 自ALB啟動以來接受、處理的連線數以及收到的請求數。
- Reading: 目前正在讀取用戶端請求的數量。
- Writing: 目前正在向用戶端寫入回應的數量。
- Waiting: 目前處於等待狀態的連線數。這些連線已經完成了讀取操作，等待下一次寫入操作。

5.0.1.1 設定簡單流量統計資料和獲取

在ALB設定檔案中，新增以下內容：

```
server {
    listen 8080;                # node_status的埠

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

啟動ALB後，存取http://alb_ip:8080/node_status即可查看。

5.0.2 Prometheus-exporter (Prometheus監控資料)

ALB提供Prometheus監控功能，預設不啟用，如需啟用，請切換到utils/exporter目錄，

執行：`./start-exporter.sh`，啟動ALB的exporter。

5.0.2.1 啟動說明

啟動ALB的exporter必須要在設定檔案中開啟stub_status功能，如下：

```
server {
    listen 8080;                # node_status的埠
    location /node_status {    # 必須要使用node_status
        stub_status on;       # 開啟stub_status
        access_log off;
        allow 127.0.0.1;
    }
}
```

在執行start-exporter腳本後，要求輸入

- node_status對應的server監聽埠，預設為8080.
- exporter監聽埠，預設為9113

如下為輸入過程：

```
root@root:/opt/ALB-V2.0.6-SE-amd64/utils/exporter$ ./start-
exporter.sh
input alb server listening port(請輸入node_status監聽埠):
alb server listening port(ALB node_status埠): 8080
input exporter listening port(請輸入監聽埠):
exporter listening port is 9113
exporter are running now!(exporter啟動成功)
```

5.0.2.2 Prometheus監控資料獲取

透過start-exporter成功後，可以透過瀏覽器或者prometheus存取：`http://IP:9113/metrics` 獲取監控資料。

註：如果上面修改了exporter的監聽埠，上面url中的9113也一併修改。

5.0.3 VTS監控資料

vts模組提供更詳細的監控資訊，包括連線數、狀態碼等。與上面兩種監控方式相比，vts模組可以提供更加全面和豐富的監控資料，透過vts模組，可以更加深入了解ALB的運行狀態。

5.0.3.1 監控指標說明

監控類別	指標名稱	說明
------	------	----

1. 基本資訊	主機資訊	代理伺服器所在的主機名稱或 IP 位址
	版本號	ALB 或監控模組的版本
	伺服器運行時間	ALB 進程已運行的時間（通常以秒為單位）
2. 連線資訊	目前用戶端連線數	目前活躍的用戶端連線數量
	讀取用戶端連線總數	累計從用戶端讀取請求的連線數
	寫入用戶端連線總數	累計向用戶端寫入回應的連線數
	已處理用戶端連線總數	累計已處理完成的用戶端連線總數
3. HTTP 虛擬主機監控	請求總數	該虛擬主機接收的總請求數
	每秒請求量（RPS）	目前每秒處理的請求數
	1xx 錯誤碼統計	所有 1xx 狀態碼（如 100, 101）的累計次數
	2xx 錯誤碼統計	所有 2xx 成功回應（如 200, 201）的累計次數
	3xx 錯誤碼統計	所有重新導向回應（如 301, 302）的累計次數
	4xx 錯誤碼統計	用戶端錯誤（如 404, 403）的累計次數
	5xx 錯誤碼統計	伺服器端錯誤（如 500, 502）的累計次數
	所有錯誤碼統計	所有非 2xx 回應的總和（或全部狀態碼分類彙總）
	發送的總流量	該虛擬主機向用戶端發送的總位元組數
	接收的總流量	該虛擬主機從用戶端接收的總位元組數
	發送流量速率	目前每秒發送的位元組數（B/s）
	接收流量速率	目前每秒接收的位元組數（B/s）
4. HTTP 快取監控	快取命中數	請求命中快取的次數
	快取未命中數	請求未命中快取的次數
	快取略過數	因設定 bypass 而跳過快取的請求次數
	快取已過期數	快取項已過期被捨棄的次數
	快取失效數	因主動失效（如 purge）導致的快取移除次數
	快取更新數	快取被更新（如 stale-while-revalidate）的次數

	重新驗證的快取數	需要向源站重新驗證的快取請求數
	快取總數	快取命中 + 未命中 + 略過等的總和（或目前快取項總數）
5. HTTP 上游伺服器監控	服務狀態	upstream server 目前狀態（up/down/checking）
	服務權重值	設定的 weight 值
	最大失敗次數	max_fails 設定值
	故障後停止服務的時間值	fail_timeout 設定值（單位：秒）
	請求總數	發往該 upstream 的總請求數
	每秒請求量	目前每秒發往該 upstream 的請求數
	1xx 錯誤碼統計	所有 1xx 狀態碼的累計次數
	2xx 錯誤碼統計	所有 2xx 成功回應的累計次數
	3xx 錯誤碼統計	所有重新導向回應的累計次數
	4xx 錯誤碼統計	用戶端錯誤的累計次數
	5xx 錯誤碼統計	伺服器端錯誤的累計次數
	所有錯誤碼統計	upstream 返回的所有狀態碼分類彙總
	發送的總流量	與該 upstream 之間的總發送位元組數
	接收的總流量	與該 upstream 之間的總接收位元組數
	發送流量速率	與 upstream 通訊的目前發送速率（B/s）
	接收流量速率	與 upstream 通訊的目前接收速率（B/s）
6. Stream 虛擬主機監控	請求總數	Stream（TCP/UDP）工作階段總數
	每秒請求數	目前每秒新建工作階段數
	連線成功/失敗統計	⚠ Stream 層無 HTTP 狀態碼，此項統計連線建立成功或失敗次數
	連線成功/失敗統計	同上
	連線重新導向統計	同上

	連線拒絕統計	同上
	伺服器端錯誤統計	同上
	所有錯誤碼統計	可指連線異常、逾時、拒絕等非 HTTP 錯誤
	發送的總流量	Stream 代理的總發送位元組數
	接收的總流量	Stream 代理的總接收位元組數
	發送流量速率	目前 Stream 發送速率 (B/s)
	接收流量速率	目前 Stream 接收速率 (B/s)
7. Stream 上游伺服器 監控	服務狀態	Stream upstream 的健康狀態
	服務權重值	設定的 weight 值
	最大失敗次數	max_fails 設定值
	故障後停止服務的時間值	fail_timeout 設定值 (單位: 秒)
	請求總數	發往該 Stream upstream 的總連線數
	每秒請求數	目前每秒新建連線數
	連線異常統計	同 Stream 虛擬主機, 通常指連線結果而非 HTTP 狀態
	所有錯誤碼統計	連線失敗、逾時等統計
	發送的總流量	與該 Stream upstream 的總發送位元組數
	接收的總流量	與該 Stream upstream 的總接收位元組數
	發送流量速率	目前發送速率 (B/s)
	接收流量速率	目前接收速率 (B/s)

5.0.3.2 vts模組設定和查看

如下是開啟設定vts的範例：

```
http {
    vhost_traffic_status_zone;           # 啟用vts模組
    ...
}
```

```

server {
    ...

    location /status {                                # 設定vts查看的uri地址
        vhost_traffic_status_display;                # 目前/status的uri展示vts
                                                    # 設定預設格式為
                                                    # 監控指標
                                                    # html頁面
    }
}

```

- HTML監控資料查看

透過瀏覽器存取查看監控頁面：`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: sec

- JSON資料格式查看

透過瀏覽器或者curl等獲取JSON資料：`http://IP:Port/status/format/json`

- Prometheus監控指標資料介面

可以透過Prometheus直接整合：`http://IP:Port/status/format/prometheus`

5.0.3.3 自訂監控指標

`vhost_traffic_status_filter_by_set_key` 指令允許你指定一個鍵名 (key name)，VTS 會根據這個鍵名來過濾流量統計資訊。通常，這個鍵名對應於 set 指令中定義的一個變數。

```
vhost_traffic_status_filter_by_set_key key_name;
# 設定區域: http, server, location
```

說明：透過使用者定義的變數啟用鍵。鍵是用於計算流量的鍵字串。名稱是用於計算流量的組字串。鍵和名稱可以包含變數，例如 \$host、\$server_name。如果指定，則名稱所屬的組。如果沒有指定第二個參數名稱，則鍵所屬的組。

如下實例：

```
http {
    vhost_traffic_status_zone;          # 啟用vts模組

    ...

    server {

        ...
        vhost_traffic_status_filter_by_set_key $uri uri::*; # 過濾
uri, 統計不同uri的流量情況
        location /status {              # 設定vts查看的uri地址
            vhost_traffic_status_display; # 目前/status的uri展示vts
            # 設定預設格式為
            vhost_traffic_status_display_format html; # 設定預設格式為
            # 監控指標
            # html頁面
        }
    }
}
```

設定完後，即可查看不同uri的流量情況：

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: sec

[JSON](#)

5.0.3.4 stream監控

ALB-VTS模組支援stream模組的監控，但預設情況下不啟用，需手動設定開啟。其用法如下：

```
http {
    ...

    server {

        ...

        location /stream/status { # 監控指標暴露uri
            stream_server_traffic_status_display;
            stream_server_traffic_status_display_format html;
        }
    }
}

stream {
    stream_server_traffic_status_zone; # 在stream塊中啟用stream vts模
```

組

...

```
server {
    ...
}
```

- [HTML監控資料查看](#)

透過瀏覽器存取查看監控頁面：`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: sec

[JSON](#)

- [JSON資料格式查看](#)

透過瀏覽器或者curl等獲取JSON資料：`http://IP:Port/stream/status/format/json`

- [Prometheus監控指標資料介面](#)

可以透過Prometheus直接整合：`http://IP:Port/stream/status/format/prometheus`

5.0.3.5 VTS監控指標grafana整合

上面VTS監控指標提供prometheus採集格式，在prometheus中，可以設定如下採集：

```

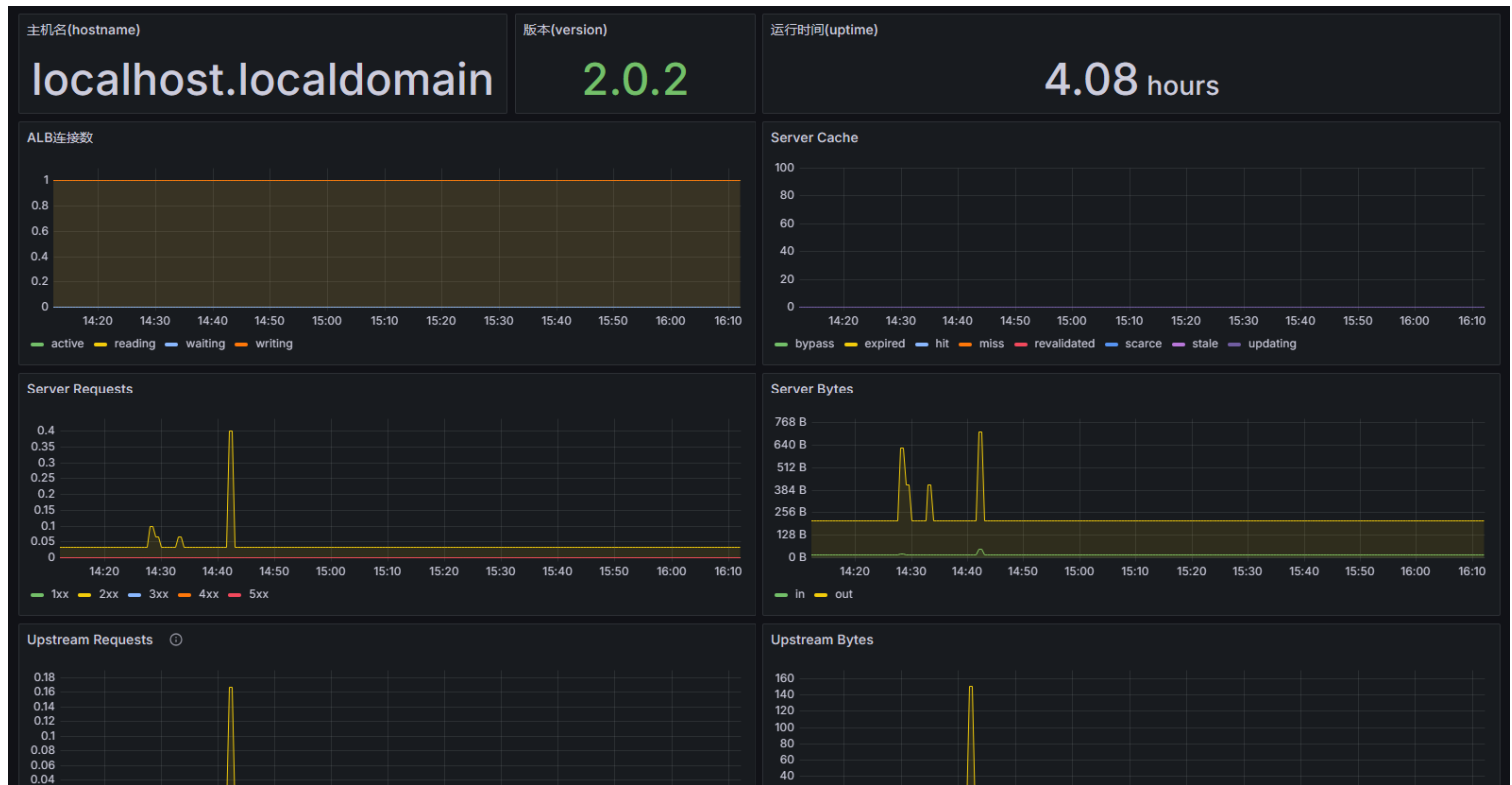
scrape_configs:
- job_name: 'alb-vts-status'
  honor_timestamps: true
  scrape_interval: 1m
  scrape_timeout: 10s
  metrics_path: /status/format/prometheus # ALB vts監控指標採集url
  scheme: http
  enable_compression: true
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
      - 192.168.1.40:8080 # ALB VTS 監控指標採集地址
    labels:
      group: 'alb'
      instance: 'alb'

```

其中grafana Dashboard的範例設定檔案路徑:

ALB安裝目錄/utils/grafana/alb-grafana.json

範例設定檔案的展示效果如下：



可以在已部署的grafana中，匯入該設定檔案即可實現上述的監控儀表板。

6 TCP代理

ALB的stream模組可以用來做TCP代理，透過在alb設定檔案中設定stream模組即可實現tcp代理。

如下的代理tcp範例：

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

    server {
        listen 443;    # 監聽埠

        proxy_pass backend;    # 轉發到backend
    }
}

# 以下為HTTP設定內容
http {
    # 略
}
```

註： TCP代理設定只能寫在最外層設定中，不能在寫在http塊或者在http塊中透過include匯入。

7 TCP代理加速

ALB TCP代理加速基於核心eBPF功能，在資料平面實現了高效的封包處理和轉發機制，顯著提升了TCP代理的效能表現。

7.1 核心eBPF加速TCP轉發優勢

- 零拷貝轉發：透過eBPF程式直接在核心態處理網路封包，避免了傳統方式中使用者態和核心態之間的頻繁資料拷貝。
- 智慧路由決策：利用eBPF的高效匹配能力，快速確定資料封包的轉發路徑，啟用TCP代理加速。

7.1.1 設定要求

- 使用者權限：
 - user 指令必須使用具有特殊權限的使用者（如 root、admin）
 - 或者如果核心支援，可將 `/proc/sys/kernel/unprivileged_bpf_disabled` 設定為 0 來使用非特權使用者
 - 生產環境建議使用具有CAP_BPF能力的非特權使用者，或設定核心參數允許非特權eBPF。
- 連線數限制：
 - 連線總數不超過100000（10萬）
- 硬體設定：
 - 記憶體：>= 16G
- 核心版本：
 - `kernel` 版本 >= 5.15
- ALB產品套件要求：
 - 如 `ALB-V2.0.6-SE-20260421-amd64-ebpf.tar.gz`，後綴攜帶ebpf特性。

7.1.2 使用範例

```
user root; # 範例使用root，生產環境建議使用具有CAP_BPF能力的非特權使用者

stream {
```

```

upstream backend_servers {
    server 127.0.0.1:20001;
}

# 全域設定
ebpf_proxy_timeout 600; # 上下游之間無資料轉發時的閒置逾時時間 (秒)
ebpf_enable on;          # 全域啟用 eBPF 加速

server {
    listen 10000;
    ebpf_enable on;          # 在特定 server 中啟用 eBPF (可覆寫全
    域設定)
    proxy_pass backend_servers;
}

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

```

7.1.3 設定參數說明

7.1.3.1 ebpf_status_return

- 語法: `ebpf_status_return ;`
- 上下文: server
- 作用: 返回簡單的狀態資訊，用於除錯目的
- 範例:

```

server {
    listen 30001;
    ebpf_status_return;
}

```

7.1.3.2 ebpf_enable

- 語法: `ebpf_enable on | off ;`
- 預設值: off
- 上下文: main, server
- 作用: 啟用或停用 eBPF 加速功能
- 範例:

```
# 全域啟用
ebpf_enable on;

server {
    listen 10000;
    # 在特定 server 中啟用
    ebpf_enable on;
    proxy_pass backend;
}
```

7.1.3.3 ebpf_proxy_timeout

- 語法: `ebpf_proxy_timeout milliseconds ;`
- 預設值: 600000 (單位毫秒, = 600秒)
- 上下文: main, server
- 作用: eBPF連線閒置逾時時間, 超過此時間無資料轉發則斷開連線。長連線場景請設定為0 (永不逾時)。
- 範例:

```
# 全域設定
ebpf_proxy_timeout 300000;

server {
    listen 10000;
    # 在 server 層級覆寫
    ebpf_proxy_timeout 60000;
    proxy_pass backend;
}
```

7.1.3.4 ebpf_proxy_buffer_size

- 語法: `ebpf_proxy_buffer_size size ;`

- 預設值: 16384 (16KB)
- 上下文: main, server
- 作用: 設定用於儲存從上游接收資料的緩衝區大小
- 範例:

```
# 全域設定
ebpf_proxy_buffer_size 32k;

server {
    listen 10000;
    # 在 server 層級設定
    ebpf_proxy_buffer_size 64k;
    proxy_pass backend;
}
```

7.1.3.5 ebpf_timer_period

- 語法: `ebpf_timer_period milliseconds ;`
- 預設值: 2000 (2秒)
- 上下文: main, server
- 作用: 設定計時器週期，用於更新流量統計資訊
- 範例:

```
# 全域設定
ebpf_timer_period 1000;

server {
    listen 10000;
    # 在 server 層級設定
    ebpf_timer_period 5000;
    proxy_pass backend;
}
```

8 TCP代理ADMQ

ALB支援使用TCP協定代理ADMQ的服務，下面是使用ALB標準版代理ADMQ服務的範例。

8.1 設定過程

1. 已部署ADMQ（略）
2. 部署ALB(略)
3. 修改ALB設定，支援ADMQ代理
4. 驗證ADMQ服務代理

8.2 ADMQ環境和代理需求

- 假定目標ADMQ叢集的四個服務節點為: 192.168.3.10、192.168.3.11、192.168.3.12，MQTT服務埠是5682。
- 假定ALB也同樣開放5682埠對外提供ADMQ服務。

8.3 修改ALB設定，支援TCP代理ADMQ

編輯 `ALB安裝目錄/conf/alb.conf` 檔案，在 `http` 設定之上最外層新增如下設定：

```
stream {
    upstream mq_server {
        server 192.168.3.10:5682 max_fails=3 fail_timeout=10s;
        server 192.168.3.11:5682 max_fails=3 fail_timeout=10s;
        server 192.168.3.12:5682 max_fails=3 fail_timeout=10s;
    }
    server {
        listen 5682 so_keepalive=on;      # ALB監聽5682埠
        proxy_connect_timeout 15s;        # 連線逾時時間
        proxy_timeout 300s;
        proxy_pass mq_server;             # ALB反向代理到三個mq節點
    }
}
```

```
# 以下是http設定內容，不需要改動
http {
    .... #略
}
```

8.4 驗證ALB代理ADMQ服務

- 查看ALB監聽埠5682是否存在。
- 若對應埠存在，使用MQ用戶端透過ALB的IP和5682埠進行驗證。

9 串流回應代理

ALB支援串流回應代理，即ALB不緩存回應，而是直接將回應轉發給用戶端，常用於各種AI/ChatGPT和對話機器人場景，下面是設定範例

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

    proxy_cache off; # 關閉快取
    proxy_buffering off; # 關閉代理緩衝
    chunked_transfer_encoding on; # 開啟分塊傳輸編碼
    tcp_nopush off; # 停用TCP NOPUSH選項，允許快速傳送小資料封包，以減少延遲
    tcp_nodelay on; # 開啟TCP NODELAY選項，禁止延遲ACK演算法
    keepalive_timeout 300; # 設定keep-alive逾時時間為300秒
}
```

9.0.1 設定詳解

- `proxy_cache off`

關閉快取功能，防止代理伺服器快取串流回應內容，確保用戶端能夠接收到即時、完整的回應。

- `proxy_buffering off`

關閉代理伺服器的回應緩衝，防止其緩衝整個回應後再傳送給用戶端，從而實現真正的串流傳輸效果。

- `chunked_transfer_encoding on`

開啟分塊傳輸編碼，允許將回應分成多個區塊進行傳輸。這是實現串流傳輸的關鍵。

- `tcp_nopush off`

停用TCP NOPUSH選項，允許快速傳送小資料封包。

- `tcp_nodelay on`

開啟TCP NODELAY選項，停用延遲ACK演算法。這有助於減少ACK封包的延遲，確保資料及時傳送。

- `keepalive_timeout 300`

增加keepalive逾時時間，防止在串流回應未完成時，代理與來源伺服器的連線就被關閉。這裡設定為300秒，根據實際需求可以調整。

10 日誌設定

ALB支援兩種類型的日誌：存取日誌（access logs）和錯誤日誌（error logs）。下面詳細介紹這兩種日誌的設定方法。

10.0.1 存取日誌

存取日誌記錄了每個 HTTP 請求的資訊，包括請求方法、請求 URI、回應狀態碼、回應大小、用戶端 IP 位址等。

10.0.1.1 存取日誌設定

存取日誌可以透過 `access_log` 指令來設定。該指令可以出現在 `http`, `server` 或 `location` 區塊中。

10.0.1.2 基本設定範例

```
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`: 定義日誌條目的格式。上面的例子定義了一個名為 `main` 的格式
- `access_log`: 指定日誌檔案的位置和使用的格式。在這個例子中，日誌檔案被保存在 `/var/log/alb/example.access.log`，並且使用 `main` 格式。

10.0.1.3 日誌格式詳解

- `$remote_addr`: 用戶端 IP 位址。
- `$remote_user`: 經過認證的使用者名稱。
- `$time_local`: 本地時間戳記。
- `$request`: 完整的請求行。
- `$status`: HTTP 狀態碼。
- `$body_bytes_sent`: 傳送給用戶端的回應主體大小。

- \$http_referer: 請求的 Referer 請求標頭。
- \$http_user_agent: 用戶端的 User-Agent 請求標頭。
- \$http_x_forwarded_for: X-Forwarded-For 請求標頭，通常用於反向代理環境。

10.0.1.4 關閉存取日誌

如果你不想記錄存取日誌，可以將 access_log 設定為 off:

```
access_log off;
```

10.0.2 錯誤日誌

錯誤日誌記錄了ALB運行過程中遇到的錯誤資訊，這對於除錯和維護非常重要。

10.0.2.1 設定錯誤日誌

錯誤日誌透過 error_log 指令來設定。該指令可以出現在 http, server 或 location 區塊中。

10.0.2.2 基本設定

```
http {  
    error_log /var/log/alb/error.log warn;  
  
    server {  
        listen 80;  
        server_name example.com;  
    }  
}
```

- error_log: 指定錯誤日誌檔案的位置和最小記錄層級。在這個例子中，錯誤日誌被保存在 /var/log/alb/error.log，並且只記錄 warn 層級及以上的錯誤。

10.0.2.3 日誌層級

錯誤日誌的層級包括：

- debug: 除錯資訊。
- info: 一般資訊。
- notice: 重要資訊。
- warn: 警告資訊。
- error: 錯誤資訊。
- crit: 危急錯誤資訊。

- alert: 警報資訊。
- emerg: 緊急資訊。

11 流量控制

ALB提供了多種方式來實現流量控制，包括限流、限速和限制連線。

11.0.1 使用 limit_req 模組進行限流

limit_req 模組允許你根據請求頻率限制用戶端的請求。

1、指令名稱： `limit_req_zone`

- 語法： `limit_req_zone key zone=name:size rate= number r/s`
- 預設值： no
- 區域： http
- 使用範例： `limit_req_zone $binary_remote_addr zone=addr:10m rate=1r/s`
- 描述： 定義一個限流區域，其中 `$binary_remote_addr` 是請求的用戶端 IP 位址，`zone=addr:10m` 表示該區域的記憶體大小為 10MB，`rate=1r/s` 表示限制每個 IP 位址每秒只能發出一個請求。

2、指令名稱： `limit_req`

- 語法： `limit_req zone=name [burst=number] [nodelay]`
- 預設值： no
- 區域： http, server, location
- 使用範例： `limit_req zone=mylimit burst=5 nodelay`
- 描述： 在指定的區域中限制請求速率，其中 `burst=5` 表示允許的突發請求數量為 5 個，`nodelay` 表示不延遲處理超過 `burst` 的請求。

11.0.1.1 範例設定

假設你想限制每個 IP 位址每秒只能發出 1個請求：

```
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`: 定義一個限流區域，其中 `$binary_remote_addr` 為用戶端 IP 位址，`mylimit` 是區域名稱，`10m` 是記憶體大小（用於儲存限流資訊），`rate=1r/s` 表示每秒最多 1 個請求。
- `limit_req`: 在 `location` 區塊中套用限流規則。`zone=mylimit` 指定使用前面定義的限流區域，`burst=5` 允許突發請求，即在短時間內可以傳送多於 1 個請求，但不能超過 `burst` 值；`nodelay` 表示不允許延遲請求。

11.0.2 使用 `limit_conn` 模組進行連線數限制

`limit_conn` 模組可以限制每個用戶端的連線數。

1、指令名稱: `limit_conn_zone`

- 語法: `limit_conn_zone key zone=name:size;`
- 預設值: `no`
- 區域: `http`
- 使用範例: `limit_conn_zone $binary_remote_addr zone=addr:10m;`
- 描述: 定義一個限流區域，其中 `key` 是用於識別用戶端的鍵（例如 IP 位址），`zone=name` 是區域名稱，`size` 是記憶體大小（用於儲存連線資訊）。

2、指令名稱: `limit_conn`

- 語法: `limit_conn zone number;`
- 預設值: `no`
- 區域: `http`, `server`, `location`
- 使用範例: `limit_conn addr 10;`
- 描述: 在指定的區域中限制請求速率，其中 `zone` 是前面定義的區域名稱，`number` 是允許的最大連線數。

11.0.2.1 範例設定

假設你想限制每個 IP 位址最多只能有 10 個並行連線：

```
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`: 定義一個連線數限制區域，其中 `$binary_remote_addr` 為用戶端 IP 位址，`myconn` 是區域名稱，10m 是記憶體大小（用於儲存連線資訊）。
- `limit_conn`: 在 `location` 區塊中套用連線數限制規則。zone=`myconn` 指定使用前面定義的連線數限制區域，10 表示最多允許 10 個並行連線。

11.0.3 使用 `limit_rate` 指令進行限速

`limit_rate` 指令可以限制用戶端下載速度。

1、指令名稱: `limit_rate`

- 語法: `limit_rate rate;`
- 預設值: `no limit`
- 區域: `http`, `server`, `location`
- 使用範例: `limit_rate 100k;`
- 描述: 設定每個連線的下載速度限制，其中 `rate` 是速率（例如 100k 表示每秒傳輸 100KB 資料）。

11.0.3.1 範例設定

假設你想將每個用戶端的下載速度限制為 100k/s:

```

server {
    listen 80;
    server_name example.com;

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

```

- `limit_rate`: 在 `location` 區塊中套用限速規則。100k 表示每秒最多傳輸 100KB 資料。

12 灰度發佈和灰度測試

ALB支援灰度發佈和灰度測試的功能，可以方便地對新版本進行測試，並平滑升級到生產環境。

12.1 場景介紹

場景	需求	典型例子
新功能灰度	只讓內部/白名單使用者體驗	新下單介面只讓 uid=1000~2000 的使用者存取
AB 實驗	按流量比例對比兩套後端	50% 使用者走 v1 服務，50% 走 v2 服務
地域/管道灰度	按請求特徵分流	廣東使用者走新機房，其他走舊機房
金絲雀發佈	先放 5% 流量，錯誤率 >1% 自動回滾	監控 5xx 比例，超過閾值一鍵切回
UI/UX 改版測試	A/B 兩組使用者看到不同介面，評估點擊率等指標	
故障回滾演練	快速切回舊版本，保障系統可用性	

12.2 ALB關鍵指令速查

ALB 本身不直接提供「灰度」功能，但可透過以下模組和指令組合實現：

- map 指令：根據請求特徵（如 Cookie、Header、IP）產生變數
- split_clients 指令：基於雜湊值按比例分流
- upstream + server：定義多個後端服務群組

指令/變數	作用	範例
split_clients	按變數做 一致性雜湊 分流	<pre>split_clients "\${remote_addr}AAA" \$group { 5% canary; 95% stable; }</pre>
map	讀 header、cookie、arg 做 條件對應	<pre>map \$cookie_ab_version \$backend { default v1; canary v2; }</pre>
set \$var / rewrite	執行時改變變數	<pre>set \$backend ''; rewrite ^/api/(.*) /\$1 break;</pre>
proxy_pass	把流量打到對應 upstream	<pre>proxy_pass http://\$backend;</pre>

access_log	打日誌方便回滾	access_log /var/log/alb/canary.log main if=\$log_canary;
------------	---------	---

12.3 關鍵指令詳解

12.3.1 1. split_clients —— 按比例分流（推薦）

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

- `${variable}`：用於雜湊的變數，如 `$request_id`、`$remote_addr`、`$http_user_agent`
- `$backend`：輸出變數，用於 `proxy_pass`
- `10%`：流量比例，支援小數（如 0.5%）
- `*`：兜底，匹配剩餘流量

✅ 優點：簡單、高效、無狀態

❌ 缺點：無法基於使用者身份精準控制（除非變數是使用者ID）

12.3.2 2. map —— 基於規則路由

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

- 根據請求標頭 `X-Version-Flag: v2` 路由到 v2
- 支援正規表示式匹配（`~*` 表示不區分大小寫）

✅ 優點：靈活，可結合 Cookie、Header、IP 等

❌ 缺點：需用戶端配合傳標識

12.3.3 3. upstream —— 定義後端服務群組

```

upstream backend_v1 {
    server 192.168.1.100:8080;
}

upstream backend_v2 {
    server 192.168.1.101:8081;
}

```

每個 upstream 可包含多個 server，支援負載平衡 可配合 weight、backup 等參數

12.4 設定範例： 按IP位址和比例灰度發佈

根據用戶端 IP (\$remote_addr) 進行雜湊計算，並將約 5% 的 IP 分配到灰度群組。

```

http {
    # 1. 定義 upstream
    upstream stable { server 10.0.0.100:8080; }
    upstream canary { server 10.0.0.200:8080; }

    # 2. 按 IP 做 5% 灰度
    split_clients "${remote_addr}AAA" $is_canary {
        5%      1;      # 變數值為 1 表示進灰度
        *      0;
    }

    # 3. 選後端
    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;
        # 打日誌，方便回滾
        access_log /var/log/alb/canary.log combined
    }
    if=$is_canary;
}
}
}

```

12.5 基於Cookie的灰度

使用者存取時帶上 Cookie: gray=true, 即可進入灰度環境。

```

# 若 Cookie 中有 gray=true, 則走新版本
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;
    }
}

```

12.6 基於 IP 白名單強制灰度（內部測試解決方案）

```

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;  # 內部測試 IP
    10.0.0.0/16       new_app;  # 整個內網段
}

```

12.7 動態控制灰度比例或基於資料庫使用者標籤分流

- 從請求參數（如 ?user_id=123）或請求標頭（如 X-User-ID: 123）中讀取使用者 ID；
- 呼叫 Lua 腳本判斷該使用者是否屬於灰度使用者（例如：使用者 ID 為偶數則進入新版本）；
- 根據判斷結果，將請求代理到 old_app 或 new_app 後端服務；
- 支援日誌記錄灰度路由結果，便於監控和排查。

```

http {

    # 使用 map 在 http 區塊層級定義變數，確保 log_format 可存取
    map $arg_user_id $user_id_from_arg {
        default $arg_user_id;
        "" "";
    }

    map $http_x_user_id $user_id_from_header {
        default $http_x_user_id;
        "" "";
    }

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

    upstream old_app { server 10.0.0.10:8000; }
}

```

```

upstream new_app { server 10.0.0.11:8001; }

server {
    listen 80;
    server_name localhost;

    # 初始化變數, 供 Lua 和日誌使用
    set $gray_user_id "";
    set $gray_backend "";

    location / {
        # 使用 access_by_lua_block 在 access 階段執行 Lua 邏輯
        access_by_lua_block {
            -- 1. 從 query string 或 header 取得 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

            -- 2. 設定日誌變數
            ngx.var.gray_user_id = user_id or ""

            -- 3. 判斷是否灰度使用者 (user_id 為偶數) , 無使用者ID預設走舊版
            local backend = "old_app"
            if user_id and user_id ~= "" then
                local num = tonumber(user_id)
                if num and num % 2 == 0 then
                    backend = "new_app"
                end
            end

            ngx.var.gray_backend = backend
        end

        # 4. 代理到對應後端
        proxy_pass http://$gray_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;
}

# 存取日誌包含灰度資訊
access_log logs/access.log gray_log;
}
}
```

13 動態 DNS 解析

ALB 支援動態 DNS 解析，從而實現對域名的即時解析。該功能的核心是透過 resolver 指令配合變數來完成，範例如下：

```
http {
    server {
        listen 80;
        # 宣告 DNS 伺服器位址及解析結果的有效期（建議在 server 區塊層級設定，
        共享 DNS 快取）
        resolver 8.8.8.8 valid=10s; # 範例DNS伺服器
        # 定義變數，用於動態解析 DNS
        set $test private.server1.com.cn;
        location / {
            # 使用變數實現動態 DNS 解析
            proxy_pass http://$test;
        }
    }
}
```

在上述設定中，當存取伺服器的根路徑 / 時，請求會被轉發到 \$test 變數所定義的後端位址

http://private.server1.com.cn。由於使用了變數 \$test，ALB 會在每次請求（或快取過期後）透過 resolver 指定的 DNS 伺服器（如 8.8.8.8）對該域名進行重新解析。

解析結果的有效期由 valid=10s 指定，即快取 10 秒。超過有效期後，下次存取將觸發新的 DNS 查詢。

重要說明：如果 proxy_pass 後直接寫的是域名（而非變數），例如 proxy_pass

http://private.server1.com.cn;，那麼 ALB 僅在啟動或重載設定時解析一次該域名，無法實現動態更新，因此不能達到動態 DNS 解析的效果。

13.1 resolver 設定詳解

動態域名解析依賴於 resolver 指令與變數的結合使用。resolver 可以在 http、server 區塊中設定，作用範圍相應不同。建議將 resolver 放在 server 區塊層級，共享 DNS 快取，避免每個 location 單獨維護。

```
http {
    server {
```

```

listen 80;
# 建議在 server 區塊層級設定 resolver, 共享 DNS 快取
resolver 8.8.8.8 valid=10s; # 範例DNS伺服器
set $test private.server1.com.cn;
location / {
    proxy_pass http://$test;
}
location /duplicate/ {
    # 複用 server 區塊的 resolver 設定, 無需重複宣告
    proxy_pass http://$test;
}
}
}

```

在此設定中，存取 `/` 和 `/duplicate/` 路徑時，雖然代理的目標域名相同（均為 `private.server1.com.cn`），但它們分別使用不同的 DNS 伺服器進行解析：

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

並且，這兩個路徑的 DNS 解析結果相互獨立，不會共享快取。

參數說明：

- `valid=10s`：指定 DNS 解析結果的快取有效期；
- `ipv6=on|off`：控制是否接受 IPv6 位址。預設為 `on`，即當域名同時解析出 IPv4 和 IPv6 位址時，兩者都會被使用。可透過設定 `ipv6=off` 停用 IPv6 解析。

14 HTTPS 與國密

ALB 透過設定 SSL 憑證和私鑰來實現安全的 HTTP 通訊。

14.1 前置準備

- 取得 SSL 憑證和私鑰，支援國密 SSL 憑證。
 - 1、伺服器端加密憑證檔案
 - 2、伺服器端加密憑證私鑰檔案
 - 3、伺服器端簽章憑證（國密必須）
 - 4、伺服器端簽章憑證私鑰檔案（國密必須）

14.2 編輯 ALB 設定檔，開啟 SSL

- 設定說明：`conf/alb.conf` 檔案中的 `server` 區塊內容。
- 核心設定：`listen`、`server_name`、`ssl_certificate`、`ssl_certificate_key`
- 範例設定：

```
server {
    listen 443 ssl;      # 使用SSL標記當前埠開啟https
    server_name your_domain.com;    # SSL憑證對應的域名

    ssl_certificate /path/to/your_domain_name.enc.crt.pem;      # 伺
    服器端加密憑證檔案
    ssl_certificate_key /path/to/your_domain_name.enc.key.pem;  # 伺
    服器端加密憑證私鑰檔案

    ssl_protocols TLSv1.2 TLSv1.3; # 使用更安全的協定版本
    ssl_prefer_server_ciphers on;   # 優先使用伺服器定義的加密套件
    ssl_ciphers HIGH:!aNULL:!MD5;  # 使用更安全的加密演算法

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

- listen: 指定監聽的埠號，這裡設定為 443，表示使用 HTTPS 協定。
- server_name: 指定伺服器名稱，這裡設定為 your_domain.com，表示該伺服器對應的域名。
- ssl_certificate: 指定 SSL 憑證的路徑，這裡設定為 /path/to/your_domain_name.crt，表示憑證檔案。
- ssl_certificate_key: 指定 SSL 私鑰的路徑，這裡設定為 /path/to/your_domain_name.key，表示私鑰檔案。
- ssl_protocols: 指定支援的 SSL 協定版本，這裡設定為 TLSv1.2 和 TLSv1.3，表示使用更安全的協定版本。
- ssl_prefer_server_ciphers: 指定使用 stronger ciphers，表示使用更安全的加密演算法。
- ssl_ciphers: 指定支援的加密演算法，這裡設定為 HIGH:!aNULL:!MD5，表示使用更安全的加密演算法。

14.2.1 設定指令說明

14.2.1.1 listen

- 描述: 指定伺服器監聽的埠和協定。
- 範例: listen 443 ssl;

14.2.1.2 ssl_certificate 和 ssl_certificate_key

- 描述: 指定 SSL 憑證的路徑和私鑰的路徑。
- 範例: ssl_certificate /path/to/your_domain_name.crt;

14.2.1.3 ssl_protocols

- 描述: 指定支援的 SSL 協定版本。
- 範例: ssl_protocols TLSv1.2 TLSv1.3; (建議使用 TLS1.2 和 TLS1.3)

14.2.1.4 ssl_ciphers

- 描述: 指定 stronger ciphers 和支援的加密演算法。
- 範例: ssl_ciphers HIGH:!aNULL:!MD5; (建議使用 stronger ciphers)

14.2.1.5 ssl_prefer_server_ciphers

- 描述: 優先使用伺服器定義的加密套件。
- 範例: ssl_prefer_server_ciphers on;

14.2.1.6 ssl_session_cache

- 描述: 啟用 SSL 工作階段快取，以減少 SSL 連線的交握次數。
- 範例: ssl_session_cache shared:SSL:10m;

14.2.1.7 ssl_session_timeout

- 描述: 設定 SSL 工作階段快取的過期時間。
- 範例: ssl_session_timeout 10m;

14.2.2 國密設定範例

註: 國密憑證和金鑰檔案命名中必須包含簽章憑證 (sig) 和伺服器端憑證 (enc)，否則將無法識別。同時 sig 憑證必須設定在前面，如下設定:

```

server {
    listen 443 ssl;      # 使用SSL標記當前埠開啟https
    server_name your_domain.com;      # SSL憑證對應的域名

    # 國密伺服器端簽章憑證和金鑰，一般名稱中攜帶sig字樣（sig憑證必須先設定）
    ssl_certificate /path/to/your_domain_name.sig.crt.pem;      # 伺服器端簽章憑證檔案
    ssl_certificate_key /path/to/your_domain_name.sig.key.pem;  # 伺服器端簽章憑證私鑰檔案

    # 國密伺服器端憑證和金鑰，一般名字中攜帶enc（enc憑證放在sig憑證後面）
    ssl_certificate /path/to/your_domain_name.enc.crt.pem;      # 伺服器端加密憑證檔案
    ssl_certificate_key /path/to/your_domain_name.enc.key.pem;  # 伺服器端加密憑證私鑰檔案

    ssl_protocols TLSv1.2 TLSv1.3; # 使用更安全的協定版本
    ssl_prefer_server_ciphers on;   # 優先使用伺服器定義的加密套件
    ssl_ciphers HIGH:!aNULL:!MD5;  # 使用更安全的加密演算法

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

```

使用國密瀏覽器存取，在網址列標記國密。



证书风险

<https://172.30.188.105:8082/index.html>

Welcome to alb!

If you see this page, the alb web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to apusic.com. Commercial support is available at apusic.com.

Thank you for using alb.

14.3 HTTP/3

ALB V2.0.6 及以上版本支援 HTTP/3 協定，使用 HTTP/3 協定時，請確保用戶端（瀏覽器）支援 HTTP/3 協定。

為了保證和原有協定相容，可以參考下面的 HTTP/3 設定：

```
server {  
    listen 443 ssl;      # 使用SSL標記當前埠開啟https  
    listen 443 quic reuseport;  # 同一埠可同時監聽SSL和QUIC (HTTP/3) ,  
    需要ALB 1.25+支援  
  
    server_name your_domain.com;  # SSL憑證對應的域名  
  
    ssl_certificate /path/to/your_domain_name.crt;      # 憑證檔案  
    ssl_certificate_key /path/to/your_domain_name.key;  # 私鑰檔案  
  
    ssl_protocols TLSv1.3; # 使用更安全的協定版本，HTTP/3需要TLSv1.3  
    ssl_prefer_server_ciphers on;  # 優先使用伺服器定義的加密套件  
    ssl_ciphers HIGH:!aNULL:!MD5;  # 使用更安全的加密演算法
```

告訴瀏覽器：同埠支援 HTTP/3，若不是443埠，請求修改和監聽埠一致。

```
add_header Alt-Svc 'h3=":443"; ma=86400';
```

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

使用支援 http3 的 curl 工具測試：

```
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/2.0.6
server: alb/2.0.6
< 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
```

15 正向代理

ALB 可以用作正向代理伺服器，它可以幫助用戶端存取其他伺服器或服務，同時隱藏用戶端的真實 IP 位址。正向代理的主要用途包括快取、負載均衡、安全過濾等。

15.1 範例設定

以下是一個簡單的 ALB 設定範例，支援 http 和 https 正向代理，用於將請求轉發到另一個伺服器：

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

- resolver：設定 DNS 解析伺服器，這裡使用的是 114.114.114.114。
- proxy_connect：啟用代理連線功能。
- proxy_connect_allow：允許的埠號，這裡是 443 和 80。

在 location 區塊中，使用 \$scheme、\$http_host 和 \$request_uri 變數來構造目標伺服器的位址。

- location /：設定處理所有請求的路徑，將請求轉發到目標伺服器。
- \$scheme：表示請求使用的協定（http 或 https）。
- \$http_host：表示請求的主機名稱和埠號。
- \$request_uri：表示請求的 URI。

16 快取設定

ALB 支援代理快取 (Proxy Cache) 功能，將後端回應內容快取在本地磁碟，快取有效期內由 ALB 直接回傳回應，可顯著降低後端壓力、縮短回應延遲。常用於商品詳情頁、新聞文章、API 查詢結果等「讀多寫少」的場景。

16.1 基礎設定

```
http {
    # 1. 定義快取儲存路徑與共享記憶體區 (必須位於 http 上下文)
    proxy_cache_path /var/cache/alb/proxy levels=1:2
    keys_zone=my_cache:10m
                                max_size=10240m inactive=60m use_temp_path=off;

    upstream backend {
        server 10.0.0.10:8080;
    }

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

        location / {
            # 2. 在 location 中啟用快取
            proxy_cache my_cache;
            proxy_cache_valid 200 302 10m;
            proxy_cache_valid 404 1m;
            proxy_cache_key "$scheme$proxy_host$request_uri";
            proxy_cache_bypass $http_cache_control;
            proxy_cache_use_stale error timeout http_500 http_502
            http_503 http_504;
            add_header X-Cache-Status $upstream_cache_status;

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

```
}
}
```

16.2 關鍵指令說明

指令	上下文	作用
proxy_cache_path	http	定義快取目錄、目錄層級、共享記憶體區名稱與大小
levels=1:2	proxy_cache_path	目錄層級，影響單目錄檔案數
keys_zone=name:size	proxy_cache_path	共享記憶體區名稱與大小，1m 可儲存約 8000 個 key
max_size=10g	proxy_cache_path	快取總容量上限，超出時按 LRU 淘汰
inactive=60m	proxy_cache_path	快取項在指定時間內未被存取則刪除
proxy_cache	http/server/location	啟用快取並指定 keys_zone 名稱
proxy_cache_valid	http/server/location	針對不同回應狀態碼設定快取時間
proxy_cache_key	http/server/location	快取鍵，預設 \$scheme\$proxy_host\$request_uri
proxy_cache_bypass	http/server/location	指定條件下略過讀快取
proxy_cache_use_stale	http/server/location	後端出錯時使用過期快取
proxy_cache_min_uses	http/server/location	同一資源被請求多少次後才快取
proxy_cache_lock	http/server/location	並行請求同一資源時只允許一個請求回源
proxy_cache_revalidate	http/server/location	啟用 If-Modified-Since 條件請求更新快取
\$upstream_cache_status	—	快取狀態變數： HIT/MISS/EXPIRED/STALE/BYPASS/UPDATING/REVALIDATED

16.3 真實使用場景

16.3.1 場景一：商品詳情頁快取

電商商品詳情頁 80% 的存取集中在「商品已上架」狀態，內容變動頻率低：

```
proxy_cache_path /cache/goods levels=1:2 keys_zone=GOODS:50m
max_size=20g inactive=30m;
```

```
location /goods/detail {
    proxy_cache GOODS;
    proxy_cache_valid 200 5m;
    proxy_cache_key "$scheme$proxy_host$request_uri"; # 建議包含多維度
    資訊，避免僅按商品ID快取導致衝突
    proxy_cache_min_uses 2;
    proxy_cache_bypass $arg_nocache;
    proxy_cache_use_stale error timeout;
    add_header X-Cache $upstream_cache_status;
    proxy_pass http://goods_service;
}
```

16.3.2 場景二：API 闖道回應快取 + 後端降級

對外提供唯讀查詢介面，後端故障時仍可回傳過期資料：

```
proxy_cache_path /cache/api levels=1:2 keys_zone=API:20m max_size=5g
inactive=10m;

location /api/v1/products {
    proxy_cache API;
    proxy_cache_valid 200 1m;
    proxy_cache_valid any 5s;
    proxy_cache_use_stale error timeout invalid_header updating
    http_500 http_502 http_503 http_504;
    proxy_cache_lock on;
    proxy_cache_revalidate on;
    proxy_pass http://api_backend;
}
```

16.4 注意事項

- 快取目錄建議使用 SSD，避免 IO 瓶頸；
- 開啟 `use_temp_path=off` 可減少檔案複製開銷；
- 含 `Set-Cookie` 的回應不會被快取；
- 涉及使用者個人資料（購物車、訂單、登入狀態）的介面**禁止**使用代理快取。

17 工作階段保持

工作階段保持 (Session Persistence) 指將同一用戶端的請求始終路由到同一台後端伺服器，常用於購物車、登入狀態、WebSocket 長連線等有狀態場景。ALB 提供多種工作階段保持方式，可按業務需求選擇。

17.1 方式一：基於 IP 雜湊

基於用戶端 IP 做雜湊計算，相同 IP 始終分配到同一後端。設定最簡單，但無法應對 NAT/代理環境（多使用者共享同一 IP）。

```
upstream backend {  
    ip_hash;  
    server 10.0.0.10:8080;  
    server 10.0.0.11:8080;  
}
```

17.2 方式二：基於 Cookie 一致性雜湊（推薦）

使用 `hash $cookie_xxx consistent` 指令，按使用者 Cookie 做一致性雜湊。即使用戶端 IP 變化（如切換網路），只要 Cookie 不變，工作階段一致性就得以保持。

```
upstream backend {  
    hash $cookie_session_id consistent;  
    server 10.0.0.10:8080;  
    server 10.0.0.11:8080;  
    server 10.0.0.12:8080;  
}
```

17.3 方式三：基於 Lua + Redis 自訂工作階段保持

透過 Lua 腳本存取外部儲存 (Redis) 查詢使用者的工作階段綁定關係，適合需要跨節點共享工作階段狀態、或需要結合業務規則的場景。

```
upstream backend_pool {  
    server 10.0.0.10:8080;
```

```

server 10.0.0.11:8080;
server 10.0.0.12:8080;
}

server {
    listen 80;
    location / {
        set $backend "";
        access_by_lua_block {
            local redis = require "resty.redis"
            local red = redis:new()
            red:set_timeout(1000)
            local ok, err = red:connect("127.0.0.1", 6379)
            if not ok then
                local hash = ngx.var.remote_addr
                local servers = {"10.0.0.10:8080", "10.0.0.11:8080",
"10.0.0.12:8080"}
                ngx.var.backend = servers[(string.byte(hash, 1) %
#servers) + 1]
                return
            end
            local sid = ngx.var.cookie_session_id
            if not sid or sid == "" then
                red:close()
                ngx.var.backend = "10.0.0.10:8080"
                return
            end
            local backend = red:get("session:" .. sid)
            if not backend or backend == ngx.null then
                local servers = {"10.0.0.10:8080", "10.0.0.11:8080",
"10.0.0.12:8080"}
                backend = servers[(ngx.var.remote_addr:byte() %
#servers) + 1]
                red:set("session:" .. sid, backend, "EX", 3600)
            end
            red:close()
        }
    }
}

```

```

        ngx.var.backend = backend
    }
    proxy_pass http://$backend;
}
}

```

17.4 方式四：基於 sticky 模組（推薦）

sticky 是 ALB 內建的工作階段保持模組，隨產品一起發布，無需額外部署即可使用。sticky 支援 cookie、route、learn 三種工作模式，在 `upstream` 區塊內透過 `sticky` 指令啟用。

17.4.1 1. cookie 模式

由 ALB 主動建立 Cookie 記錄後端節點，用戶端首次存取時分配後端並下發 Cookie，後續請求攜帶 Cookie 即可路由到相同節點。

```

upstream backend {
    sticky cookie srv_id expires=1h domain=.example.com path=/
    httponly secure;
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
}

```

參數說明：

參數	必選	說明
srv_id	是	Cookie 名稱，可自訂為業務相關名稱（如 JSESSIONID、route_id）
expires=1h	否	Cookie 有效期，瀏覽器關閉後失效（工作階段級）
domain	否	Cookie 生效域名（如 .example.com 涵蓋子網域）
path	否	Cookie 生效路徑，預設 /
secure	否	僅在 HTTPS 連線下傳輸，防止明文洩漏
httponly	否	禁止 JavaScript 讀取，緩解 XSS 攻擊

17.4.2 2. route 模式

按順序讀取請求中的變數（如 Cookie、Header、參數），使用第一個非空變數作為路由鍵。適合由上游應用程式在登入時下發業務識別 Cookie 的場景。

```
upstream backend {
    sticky route $cookie_session_id $cookie_user_id $http_x_session;
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
}
```

按 `$cookie_session_id` → `$cookie_user_id` → `$http_x_session` 順序查找，找到第一個非空值作為路由鍵。

17.4.3 3. learn 模式

從上游回應的 `Set-Cookie` 中學習工作階段識別。ALB 不主動建立 Cookie，而是透傳上游下發的 Session Cookie。適合已經使用 Session 機制的 Java/PHP 等後端應用程式。

```
upstream backend {
    sticky learn create=$upstream_cookie_JSESSIONID
lookup=$cookie_JSESSIONID
        expires=1h;
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
}
```

- `create=$upstream_cookie_JSESSIONID`：從上游回應的 `Set-Cookie` 標頭中提取 `JSESSIONID` 寫入變數
- `lookup=$cookie_JSESSIONID`：從用戶端請求 Cookie 中讀取 `JSESSIONID` 作為路由鍵
- 上游應用程式負責登入時寫入 Session Cookie（如 Java 的 `JSESSIONID`），ALB 自動識別並保持工作階段

17.5 關鍵指令說明

指令/變數	作用
<code>ip_hash</code>	啟用基於用戶端 IP 的雜湊分配
<code>hash \$key consistent</code>	基於任意變數（Cookie、Header、參數）的一致性雜湊
<code>sticky cookie</code>	由 ALB 建立工作階段 Cookie 進行黏性工作階段

sticky route	按請求變數（Cookie、Header）作為路由鍵
sticky learn	從上游 Set-Cookie 中學習工作階段識別
\$cookie_xxx	讀取指定名稱的 Cookie 值
access_by_lua_block	在請求存取階段執行 Lua 程式碼，常用於動態選擇後端
lua-resty-redis	ALB 內建的 Redis 用戶端函式庫（Lua 模組），可用於工作階段儲存

17.6 真實使用場景

17.6.1 場景一：電商登入狀態保持

使用者登入後寫入 `session_id` Cookie，後續請求根據該 Cookie 路由到同一後端，避免購物車遺失：

```
upstream user_backend {
    hash $cookie_session_id consistent;
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
}

server {
    listen 80;
    server_name shop.example.com;
    location / {
        proxy_pass http://user_backend;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

17.6.2 場景二：WebSocket 長連線保持

WebSocket 連線建立後會被綁定到特定後端，必須保證同一使用者始終連線到同一後端，否則需要重建連線：

```
upstream ws_backend {
    ip_hash;
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
}
```

```

}

server {
    listen 80;
    location /ws/ {
        proxy_pass http://ws_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}

```

17.6.3 場景三：Java 後端 sticky 模式無縫接入

後端為 Java 應用程式（Spring Boot 等）已使用 `JSESSIONID`，ALB 啟用 sticky learn 模式即可自動識別上游 Session Cookie，無需改造業務：

```

upstream java_backend {
    sticky learn create=$upstream_cookie_JSESSIONID
lookup=$cookie_JSESSIONID
        expires=1h;

    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
    server 10.0.0.12:8080;
}

server {
    listen 80;
    server_name java.example.com;
    location / {
        proxy_pass http://java_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}

```

17.7 注意事項

- 啟用 `consistent` 後端擴容/縮容時，僅影響部分使用者的工作階段，遷移成本低；
- 基於 Cookie 的方式要求後端應用程式在登入成功時設定統一的 `session_id` Cookie；
- Lua + Redis 方案需要額外部署 Redis 服務並保證其高可用性；
- sticky cookie 模式下，Cookie 名稱應避免與後端應用程式 Session Cookie 衝突（例如 Java 應用程式同時使用 `JSESSIONID` 時不要將 sticky Cookie 也命名為 `JSESSIONID`）；
- sticky learn 模式要求上游回應中 `Set-Cookie` 標頭必須攜帶工作階段識別，常見 Java 框架預設即符合；
- sticky 模組透過 Cookie 運作，需保證用戶端啟用 Cookie。

18 壓縮

ALB 支援對回應內容進行壓縮，減小傳輸體積、加快頁面載入速度。支援 Gzip 與 Brotli 兩種演算法：Gzip 相容性最好（所有瀏覽器都支援），Brotli 壓縮率更高但需要 HTTPS 協定且僅 Chrome/Firefox/Edge 等現代瀏覽器支援。

18.1 Gzip 壓縮

```
http {
    gzip on;
    gzip_min_length 1024;
    gzip_comp_level 6;
    gzip_vary on;
    gzip_proxied any;
    gzip_types
        text/plain
        text/css
        text/xml
        application/json
        application/javascript
        application/xml
        application/xml+rss
        image/svg+xml
        font/ttf
        font/otf;

    server {
        listen 80;
        server_name www.example.com;
        root /var/www/html;
    }
}
```

18.2 關鍵指令說明

指令	預設值	作用
gzip	off	是否啟用 Gzip 壓縮
gzip_min_length	20	回應體小於該值不壓縮（位元組）
gzip_comp_level	1	壓縮級別 1-9，建議 4-6
gzip_vary	off	是否在回應標頭新增 Vary: Accept-Encoding
gzip_proxied	off	對上游回應啟用壓縮的條件
gzip_types	text/html	需要壓縮的 MIME 類型（text/html 始終壓縮）
gzip_disable	—	透過正規表示式比對停用壓縮
gzip_buffers	32 4k	壓縮緩衝區數量與大小
gzip_http_version	1.1	啟用壓縮的最低 HTTP 版本

18.3 Brotli 壓縮（需 ALB 編譯 alb_brotli 模組）

Brotli 演算法相比 Gzip 壓縮率可提升 15-25%，特別適合 JS/CSS/HTML 等文字資源。

```
http {
    brotli on;
    brotli_comp_level 6;
    brotli_min_length 1024;
    brotli_types
        text/plain
        text/css
        text/xml
        application/json
        application/javascript
        application/xml
        image/svg+xml;

    server {
        listen 443 ssl;
```

```
}
}
```

18.4 預壓縮檔案 (gzip_static / brotli_static)

如果後端在建置階段已經產生了 `.gz` / `.br` 預壓縮檔案，可使用 `gzip_static on` 讓 ALB 直接傳送預壓縮檔案，略過即時壓縮過程，可顯著降低 CPU 佔用。

```
http {
    gzip_static on;
    brotli_static on;
    gzip_proxied any;
    gzip on;
}
```

預壓縮檔案命名規則（以 `index.html` 為例）：

- 原始檔案： `index.html`
- Gzip 預壓縮： `index.html.gz`
- Brotli 預壓縮： `index.html.br`

18.5 真實使用場景

18.5.1 場景一：API 回應壓縮

JSON 介面在行動端網路下體積較大，開啟 Gzip 後通常可減少 60-80% 體積：

```
http {
    gzip on;
    gzip_min_length 256;
    gzip_types application/json application/javascript text/css;

    server {
        location /api/ {
            gzip on;
            gzip_types application/json;
            proxy_pass http://api_backend;
        }
    }
}
```

```

    }
}
}

```

18.5.2 場景二：靜態資源雙壓縮（Gzip + Brotli）

同時啟用 Gzip 與 Brotli，透過 `Vary: Accept-Encoding` 標頭讓瀏覽器和 CDN 快取按壓縮演算法分別儲存：

```

http {
    gzip on;
    gzip_static on;
    brotli on;
    brotli_static on;
    gzip_vary on;

    server {
        listen 80;
        root /var/www/static;
        location / {
            add_header Vary "Accept-Encoding";
        }
    }
}

```

18.6 注意事項

- 已被壓縮的資源（jpg、png、mp4、zip 等）開啟壓縮會浪費 CPU，應避免在 `gzip_types` 中列出；
- CPU 資源緊張時優先使用 `gzip_static` 跳過即時壓縮；
- 啟用 Brotli 必須使用 HTTPS（瀏覽器僅在 TLS 下傳送 `br` 標記）；
- 高並發場景下建議將 `gzip_comp_level` 設定為 4-5，避免 CPU 成為瓶頸。

19 gRPC 代理

ALB 支援 gRPC 代理，可作為微服務架構中 gRPC 服務叢集的統一入口。gRPC 基於 HTTP/2 協定傳輸，ALB 透過 `grpc_pass` 指令實現 gRPC 請求的反向代理。

19.1 基礎設定

```

upstream grpc_backend {
    server 10.0.0.10:50051;
    server 10.0.0.11:50051;
}

server {
    listen 80 http2;           # 必須啟用 http2 (生產環境建議使用 443 連接埠
    # 並啟用 TLS)
    server_name grpc.example.com;

    location / {
        grpc_pass grpc://grpc_backend;

        # 將上游錯誤轉換為 gRPC 狀態碼
        error_page 502 = /grpc_error;
        error_page 503 = /grpc_error;
        error_page 504 = /grpc_error;

        # gRPC 健康檢查見 "健康檢查 -> 七層HTTP代理健康檢查" 章節
    }

    location = /grpc_error {
        internal;
        default_type application/grpc;
        add_header grpc-status 14;           # 14 = UNAVAILABLE
        add_header content-length 0;
        return 204;
    }

```

```
}
}
```

19.1.1 HTTPS 加密代理

生產環境推薦使用 HTTPS 加密傳輸 gRPC 流量：

```
server {
    listen 443 ssl http2;
    server_name grpc.example.com;

    ssl_certificate      /path/to/server.crt;
    ssl_certificate_key  /path/to/server.key;
    ssl_protocols        TLSv1.2 TLSv1.3;

    location / {
        grpc_pass grpc://grpc_backend;      # grpc 表示 HTTPS 加密

        # 透傳用戶端中繼資料
        grpc_set_header X-Request-Id $request_id;
        grpc_set_header X-Real-IP     $remote_addr;
    }
}

upstream grpc_backend {
    server 10.0.0.10:50051;
    server 10.0.0.11:50051;
}
```

19.2 關鍵指令說明

指令	上下文	作用
grpc_pass grpc://upstream	location	代理 gRPC 流量到上游（明文）
grpc_pass grpcs://upstream	location	代理 gRPC 流量到上游（TLS 加密）

grpc_set_header	location	設定/重寫傳遞給上游的 gRPC 中繼資料標頭
error_page 502 = /grpc_error	http/server/location	將 HTTP 錯誤轉換為 gRPC 狀態碼
grpc-status 14	—	gRPC 狀態碼：14 = UNAVAILABLE，常用於後端不可達
listen ... http2	server	啟用 HTTP/2, gRPC 必需

19.3 真實使用場景

19.3.1 場景一：微服務 gRPC 叢集統一入口

微服務架構中，多個 gRPC 服務由 ALB 統一對外暴露入口：

```

upstream user_service {
    server 10.0.0.10:50051;
    server 10.0.0.11:50051;
}

upstream order_service {
    server 10.0.0.20:50051;
    server 10.0.0.21:50051;
}

server {
    listen 443 ssl http2;
    server_name api.example.com;

    ssl_certificate      /etc/alb/certs/api.crt;
    ssl_certificate_key  /etc/alb/certs/api.key;

    # 使用者服務
    location /user.UserService/ {
        grpc_pass grpc://user_service;
    }

```

```
# 訂單服務
location /order.OrderService/ {
    grpc_pass grpc://order_service;
}
}
```

19.3.2 場景二：gRPC 反射與健康檢查

上游 gRPC 服務啟用 gRPC Reflection 與 Health Checking 協定後，可結合 `check` 指令做主動健康檢查：

```
upstream grpc_backend {
    server 10.0.0.10:50051;
    server 10.0.0.11:50051;

    # type=http 即可，gRPC 複用 HTTP 健康檢查機制
    check interval=3000 rise=2 fall=3 timeout=2000 type=http;

    # gRPC 健康檢查路徑（使用 HTTP/1.1，ALB 健康檢查模組目前不支援 HTTP/2.0 請求）
    check_http_send "POST /grpc.health.v1.Health/Check
HTTP/1.1\r\nHost: grpc.example.com\r\nContent-Type:
application/grpc\r\n\r\n";
    check_http_expect_alive http_2xx;
}
```

19.4 注意事項

- gRPC 代理必須啟用 HTTP/2 (`listen 443 ssl http2;` 或 `listen 80 http2;`) ；
- HTTP 錯誤必須透過 `error_page` 轉換為 gRPC 狀態碼，否則用戶端將得到無意義的 HTTP 錯誤；
- gRPC 用戶端對代理逾時非常敏感，需結合上游 RTT 合理設定 `grpc_connect_timeout` 、 `grpc_send_timeout` 、 `grpc_read_timeout` ；
- ALB 與上游 gRPC 服務之間建議使用 TLS (`grpc://`) 加密傳輸。

20 WebSocket 代理

ALB 支援 WebSocket 代理，可將用戶端的 WebSocket 連線反向代理到後端 WebSocket 服務。WebSocket 在用戶端（瀏覽器）與 ALB、ALB 與上游服務之間分別建立 HTTP 升級交握，交握成功後切換為雙向長連線。

20.1 基礎設定

```
upstream ws_backend {
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
    ip_hash;
    # WebSocket 長連線建議保持工作階段黏性，但生產環境
    建議使用 sticky cookie 或基於 cookie 的一致性雜湊
}

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

    location /ws/ {
        proxy_pass http://ws_backend;

        # WebSocket 升級必備標頭
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # 長連線逾時 (預設 60s 過短)
        proxy_read_timeout 3600s;          # 1 小時無資料斷開
        proxy_send_timeout 3600s;

        # 透傳用戶端真實資訊
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
}
}
```

20.2 與 HTTPS 共用 443 連接埠

WebSocket 協定可透過 `Upgrade: websocket` 在 HTTPS 上承載，無需單獨開放連接埠：

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

    ssl_certificate      /etc/alb/certs/ws.crt;
    ssl_certificate_key  /etc/alb/certs/ws.key;

    # 同一 server 區塊下, location 可同時處理 HTTPS 和 WebSocket
    location /ws/ {
        proxy_pass http://ws_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 3600s;
    }

    location / {
        # 一般 HTTPS 請求
        proxy_pass http://backend;
    }
}
```

20.3 關鍵指令說明

指令	作用
<code>proxy_http_version 1.1</code>	必須設定，WebSocket 升級基於 HTTP/1.1
<code>proxy_set_header Upgrade \$http_upgrade</code>	透傳用戶端 Upgrade 標頭（\$http_upgrade 預設值即為空）

proxy_set_header Connection "upgrade"	固定設定為 upgrade，通知上游建立長連線
proxy_read_timeout	讀取閒置逾時（WebSocket 靜默期），預設 60s 極易斷開
proxy_send_timeout	寫入閒置逾時
proxy_buffering off	關閉回應緩衝，適合即時雙向通訊
ip_hash / sticky	WebSocket 長連線需配合工作階段保持，否則節點變更會強制重連

20.4 真實使用場景

20.4.1 場景一：線上客服系統 WebSocket 代理

客服系統 WebSocket 推送使用者訊息、客服回覆、工單狀態變更：

```

upstream chat_backend {
    ip_hash;                # 同一使用者始終連線同一後端
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
    server 10.0.0.12:8080;
}

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

    ssl_certificate         /etc/alb/certs/support.crt;
    ssl_certificate_key     /etc/alb/certs/support.key;

    location /ws/chat {
        proxy_pass http://chat_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 86400s;        # 24 小時靜默逾時
        proxy_send_timeout 86400s;
        proxy_buffering off;
    }
}

```

```

    }
}

```

20.4.2 場景二：即時資料推送（股票行情、K 線）

金融行情推送要求毫秒級延遲，關閉所有緩衝並擴大逾時：

```

upstream market_backend {
    server 10.0.0.10:8080;
    server 10.0.0.11:8080;
}

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

    location /ws/quote {
        proxy_pass http://market_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # 即時性最佳化
        proxy_buffering off;
        proxy_cache off;
        tcp_nodelay on;
        tcp_nopush off;
        keepalive_timeout 300s;

        # 長連線逾時
        proxy_read_timeout 7d;
        proxy_send_timeout 7d;
    }
}

```

一週，避免長時間靜默斷開

20.5 注意事項

- 必須使用 `ip_hash` 或 `sticky` 實現工作階段保持，否則 WebSocket 節點切換將導致連線強制重連；
- `proxy_read_timeout` 是連線靜默逾時，並非總存活時間，應設定為業務允許的最大間隔；
- 反向代理時關閉 `proxy_buffering` 與 `proxy_cache` 避免訊息延遲；
- 當用戶端使用 HTTPS 時，ALB 與上游可使用 HTTP（如 `proxy_pass http://ws_backend;`），無需雙層 TLS；
- WebSocket over HTTP/2 (RFC 8441) 目前相容性較差，ALB 目前建議使用 HTTP/1.1 升級方式建立 WebSocket 連線。

21 URL 重寫與重新導向

ALB 提供靈活的 URL 重寫與重新導向能力，可用於路徑美化、舊連結遷移、API 版本路由、HTTPS 強制跳轉、路由兜底等場景。

21.1 基礎設定

```
# 1. return 強制 HTTPS
server {
    listen 80;
    server_name example.com;

    location / {
        return 301 https://$host$request_uri;
    }
}

# 2. rewrite 路徑重寫
server {
    listen 80;
    server_name example.com;

    location /old-api/ {
        rewrite ^/old-api/(.*)$ /new-api/v2/$1 permanent;
    }
}

# 3. try_files 路由兜底
server {
    listen 80;
    server_name example.com;

    location / {
        try_files $uri $uri/ /index.html;
```

```
}
}
```

21.2 關鍵指令說明

指令	上下文	作用
<code>return code [text url]</code>	server/location	直接回傳狀態碼/重新導向，停止後續處理
<code>rewrite regex replacement [flag]</code>	server/location/if	按正規表示式重寫 URL，可選 last/break/redirect/permanent 旗標
<code>try_files file ... =code</code>	server/location	依次檢查檔案是否存在，最後一個參數為兜底（URI 或狀態碼）

21.2.0.1 rewrite 旗標位

旗標	狀態碼	是否重新比對 location	典型用途
(無)	—	否	內部重寫，瀏覽器無感知
last	—	是	重寫後停止目前 location，重新比對
break	—	否	重寫後停止目前規則，繼續處理目前 location
redirect	302	否	暫時重新導向，瀏覽器網址列更新
permanent	301	否	永久重新導向，瀏覽器與 SEO 友善

21.2.0.2 return 常用狀態碼

狀態碼	含義
301	永久重新導向，瀏覽器會快取
302	暫時重新導向，瀏覽器每次重新請求
307	暫時重新導向，保留請求方法（POST → POST）
308	永久重新導向，保留請求方法
444	ALB 特有，不回傳回應直接關閉連線（防掃描）
403 / 404 / 500 等	錯誤回應

21.3 真實使用場景

21.3.1 場景一：舊 API 路徑遷移

舊介面 `/api/v1/users` 遷移至新介面 `/api/v3/members`，使用 301 永久重新導向保留 SEO 權重：

```
server {  
    listen 80;  
    server_name api.example.com;  
  
    location /api/v1/ {  
        rewrite ^/api/v1/(.*)$ /api/v3/members/$1 permanent;  
    }  
  
    location /api/v3/ {  
        proxy_pass http://api_backend;  
    }  
}
```

21.3.2 場景二：HTTP 強制跳轉 HTTPS

所有 HTTP 存取必須重新導向至 HTTPS：

```
server {  
    listen 80;  
    server_name example.com;  
  
    # 全站強制 HTTPS  
    return 301 https://$host$request_uri;  
}  
  
server {  
    listen 443 ssl;  
    server_name example.com;  
    # ... HTTPS 設定  
}
```

21.3.2.1 場景三：SPA 前端路由兜底

React/Vue 單頁應用，實體路徑不存在時回退到 `index.html`，由前端路由處理：

```
server {
    listen 80;
    server_name app.example.com;
    root /var/www/app;

    location / {
        # 優先找實體檔案 → 找目錄 → 兜底回傳 index.html
        try_files $uri $uri/ /index.html;
    }
}
```

21.3.2.2 場景四：API 版本標頭透傳與路徑重寫

呼叫方統一存取 `/api/v1/`，但後端服務只暴露 `/internal/v3/`：

```
location /api/v1/ {
    # 去掉 /api/v1 前綴，替換為 /internal/v3/
    rewrite ^/api/v1/(.*)$ /internal/v3/$1 break;
    proxy_pass http://api_backend;
}
```

21.4 注意事項

- `rewrite` 與 `return` 同處一個 `location` 時，`return` 優先；
- `permanent` (301) 一旦下發，瀏覽器會長期快取，遷移前確認 URL 永久變更；
- `last` 旗標會重新比對 `location`，可能導致循環重寫（應避免 `last` 後又命中同一規則）；
- 複雜重寫建議使用 `map` 指令 + 變數，避免 `if` 中巢狀 `rewrite`（`if` 指令有諸多限制）。

22 跨域設定 (CORS)

ALB 可在閘道層統一處理 CORS (Cross-Origin Resource Sharing, 跨來源資源共用), 避免在每個後端服務中重複實作跨域邏輯。常見場景: 前後端分離專案中, 前端透過 Ajax 呼叫不同網域的 API 介面。

22.0.1 基礎設定

```
server {  
    listen 80;  
    server_name api.example.com;  
  
    # 簡單請求處理  
    location /api/ {  
        # 允許所有來源 (生產環境應明確指定)  
        add_header Access-Control-Allow-Origin $http_origin always;  
        add_header Access-Control-Allow-Credentials true always;  
        add_header Access-Control-Allow-Methods "GET, POST, PUT,  
DELETE, OPTIONS" always;  
        add_header Access-Control-Allow-Headers "Authorization,  
Content-Type, X-Requested-With" always;  
        add_header Access-Control-Max-Age 3600;  
  
        # 處理 OPTIONS 預檢請求  
        if ($request_method = 'OPTIONS') {  
            add_header Access-Control-Allow-Origin $http_origin;  
            add_header Access-Control-Allow-Credentials true;  
            add_header Access-Control-Allow-Methods "GET, POST, PUT,  
DELETE, OPTIONS";  
            add_header Access-Control-Allow-Headers "Authorization,  
Content-Type, X-Requested-With";  
            add_header Access-Control-Max-Age 3600;  
            add_header Content-Type "text/plain; charset=utf-8";  
            add_header Content-Length 0;  
            return 204;  
        }  
    }  
}
```

```

    proxy_pass http://api_backend;
    proxy_set_header Host $host;
}
}

```

22.1 關鍵指令說明

回應標頭	作用
Access-Control-Allow-Origin	允許的來源（* 或具體網域名稱，多網域名稱需配合 \$http_origin）
Access-Control-Allow-Credentials	是否允許攜帶 Cookie（設為 true 時 Allow-Origin 不可為 *）
Access-Control-Allow-Methods	允許的 HTTP 方法（OPTIONS 預檢回應必備）
Access-Control-Allow-Headers	允許的請求標頭（前端自訂標頭如 Authorization）
Access-Control-Expose-Headers	暴露給 JS 的回應標頭（預設 JS 只能讀取 6 個 CORS 安全標頭）
Access-Control-Max-Age	預檢結果快取時間（秒），減少 OPTIONS 請求頻率
\$http_origin	用戶端請求的 Origin 標頭值

22.2 真實使用場景

22.2.0.1 場景一：多來源白名單

企業開道需要同時支援 `https://admin.example.com` 與 `https://ops.example.com` 呼叫 API：

```

# 透過 map 維護白名單
map $http_origin $cors_origin {
    default "";
    "https://admin.example.com" "https://admin.example.com";
    "https://ops.example.com"   "https://ops.example.com";
    "https://test.example.com"  "https://test.example.com";
}

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

```

```

location /api/ {
    add_header Access-Control-Allow-Origin $cors_origin always;
    add_header Access-Control-Allow-Credentials true always;
    add_header Access-Control-Allow-Methods "GET, POST, PUT,
DELETE, OPTIONS" always;
    add_header Access-Control-Allow-Headers "Authorization,
Content-Type" always;

    if ($request_method = 'OPTIONS') {
        add_header Access-Control-Allow-Origin $cors_origin;
        add_header Access-Control-Allow-Credentials true;
        add_header Access-Control-Allow-Methods "GET, POST, PUT,
DELETE, OPTIONS";
        add_header Access-Control-Allow-Headers "Authorization,
Content-Type";
        add_header Access-Control-Max-Age 86400;
        return 204;
    }

    proxy_pass http://api_backend;
}

```

22.2.0.2 場景二：全開放（僅適用於內部系統/開發環境）

```

location /api/ {
    add_header Access-Control-Allow-Origin * always;
    add_header Access-Control-Allow-Methods "GET, POST, OPTIONS"
always;
    add_header Access-Control-Max-Age 3600;

    if ($request_method = 'OPTIONS') {
        return 204;
    }
}

```

```
proxy_pass http://api_backend;  
}
```

22.3 注意事項

- `add_header` 在 `4xx / 5xx` 回應中預設不生效，需新增 `always` 參數強制下發；
- 使用 Cookie 時 `Access-Control-Allow-Origin` 不能為 `*`，必須為具體網域名稱；
- OPTIONS 預檢請求必須直接回傳 204，**不要**代理到上游（避免上游重複處理）；
- CORS 標頭應統一在 ALB 處理，避免後端服務與 ALB 同時下發導致重複標頭衝突；
- 涉及敏感介面時，建議在 CORS 之外額外校驗 `Origin` 標頭或 `Referer` 標頭。

23 故障轉移與重試

ALB 在上游節點故障時具備自動故障轉移與重試能力，可顯著提升服務可用性。

23.0.1 基礎設定

```
upstream backend {
    server 10.0.0.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.11:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.12:8080 max_fails=3 fail_timeout=30s;

    # 備機節點：所有主機不可用時啟用
    server 10.0.0.100:8080 backup;
}

server {
    listen 80;
    server_name example.com;

    location / {
        proxy_pass http://backend;

        # 失敗重試：僅在指定錯誤時重試下一個上游
        proxy_next_upstream error timeout invalid_header http_502
http_503 http_504;
        proxy_next_upstream_tries 3;           # 最多嘗試 3 個節點
        proxy_next_upstream_timeout 10s;      # 整體重試時間不超過 10s

        # 重試時停用請求主體重傳（POST 失敗時不重試）
        proxy_next_upstream_non_idempotent off;
    }
}
```

23.1 關鍵指令說明

指令	預設值	作用
<code>max_fails=N</code>	1	失敗次數達到 N 後，節點被標記為不可用
<code>fail_timeout=time</code>	10s	失敗次數計算視窗；標記為不可用後，節點被排除的時間
<code>backup</code>	—	備機節點，僅當所有主機都不可用時才使用
<code>down</code>	—	永久標記節點不可用（用於維護時摘除節點）
<code>proxy_next_upstream</code>	error timeout	觸發重試的條件（error / timeout / invalid_header / http_502 / http_503 / http_504 / http_403 / http_404 / http_500 / off）
<code>proxy_next_upstream_tries</code>	0	最多重試次數（0 表示無限制，受 <code>proxy_next_upstream_timeout</code> 約束）
<code>proxy_next_upstream_timeout</code>	0	整體重試逾時（0 表示無限制）
<code>proxy_next_upstream_non_idempotent</code>	off	是否對非冪等請求（POST）啟用重試（預設 off）

23.2 真實使用場景

23.2.0.1 場景一：主備災備

主機房故障時自動切換到備機房：

```

upstream backend {
    # 主機房
    server 10.0.0.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.11:8080 max_fails=3 fail_timeout=30s;
    # 備機房
    server 10.1.0.10:8080 max_fails=3 fail_timeout=30s backup;
    server 10.1.0.11:8080 max_fails=3 fail_timeout=30s backup;
}

server {
    listen 80;
    location / {
        proxy_pass http://backend;
        proxy_next_upstream error timeout http_502 http_503
http_504;
    }
}

```

```

        proxy_next_upstream_tries 3;
    }
}

```

23.2.0.2 場景二：節點維護摘除

需要對某節點進行維護時，可暫時將其標記為 `down`：

```

upstream backend {
    server 10.0.0.10:8080;
    server 10.0.0.11:8080 down;          # 維護中
    server 10.0.0.12:8080;
}

```

或透過 Lua 動態控制（無需重啟）：

```

http {
    init_by_lua_block {
        -- 從 Redis 載入節點狀態
        local redis = require "resty.redis"
        local red = redis:new()
        red:set_timeout(1000)
        local ok = red:connect("127.0.0.1", 6379)
        if ok then
            ngx.shared.nodes:set("10.0.0.11:8080", "down")
        end
    }
}

```

23.2.0.3 場景三：可重試與不可重試請求區分

GET 等幂等請求可重試，POST 涉及訂單/支付等業務應避免重試：

```

map $request_method $retry_enable {
    default 1;
    POST    0;
    PUT     0;
}

```

```

    DELETE    0;
}

server {
    location /api/ {
        proxy_pass http://backend;
        proxy_next_upstream error timeout http_502 http_503
http_504;
        # 僅幂等請求啟用重試
        proxy_next_upstream $retry_enable;
    }
}

```

23.3 注意事項

- `proxy_next_upstream` 在重試時會**完整重傳**請求主體（除非上游已收到部分回應），POST 等非幂等請求需謹慎開啟；
- `max_fails` 與主動健康檢查獨立運作，建議同時啟用主動健康檢查以更快摘除故障節點（參見「健康檢查」章節）；
- `fail_timeout` 時間設定過短會導致節點在恢復後仍被排除，過長則故障切換緩慢，建議 10-30s；
- `proxy_next_upstream_non_idempotent on` 僅在業務確保上游介面**完全幂等**時開啟，否則可能產生重複訂單/支付。

24 使用者場景設定案例

24.1 概述

ALB 不僅是一個 Web 伺服器，更是現代應用架構中的「流量中樞」。它在高並發、安全防護、服務治理等方面發揮著關鍵作用。本文透過多個貼近真實業務的場景，詳細說明企業在不同發展階段如何借助 ALB 解決實際問題，並附上可直接複用的設定範例與效果分析。

24.2 場景一：前端單頁應用託管與路由支援/靜態資源發佈

24.2.1 1. 背景

某公司使用 React 開發管理後台，建置後輸出靜態資源。初期透過 Node.js Express 提供靜態服務，部署在單台雲伺服器上。

24.2.2 2. 遇到的問題與待解決的問題

- 使用者直接存取 /settings 等非根路徑時回傳 404（因伺服器無對應實體檔案）；
- 靜態資源未快取，每次重新整理都重新下載，頻寬成本高、載入慢；
- Express 在並發 >500 時 CPU 佔用飆升，回應延遲超過 2 秒。

待解決問題：實現 SPA 路由相容、靜態資源高效分發、降低伺服器負載。

24.2.3 3. ALB 功能與設定與問題比對

- `try_files` 指令：實現前端路由 fallback 到 index.html；
- `expires` 與 `add_header`：設定長期快取策略；
- ALB 高效能事件驅動模型：高效處理靜態檔案請求。

24.2.4 4. 使用 ALB 解決該場景下的設定與詳解

```
server {  
    listen 80;  
    server_name test.com;  
  
    root /var/www/admin/dist;  
    index index.html;  
  
    # SPA 路由支援：若檔案不存在，回退到 index.html  
    location / {
```

```

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

# 靜態資源設定 1 年快取 (immutable)
location ~* \.(js|css|png|jpg|svg|woff2)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
}
}

```

- `try_files $uri $uri/ /index.html`：優先找實體檔案，找不到則交由前端路由處理；
- `immutable` 快取：瀏覽器在快取有效期內絕不發起條件請求，極大提升重複造訪速度。

24.2.5 5. 效果，前後對比

指標	改造前 (Express)	改造後 (ALB)
首屏載入時間	2.1s	0.6s
靜態資源 QPS	300	5000+
CPU 佔用 (500 並發)	85%	12%
路由 404 錯誤	頻繁發生	完全消除

24.3 場景二：微服務 API 統一入口與負載平衡

24.3.1 1. 背景

某電商平台完成微服務拆分，擁有 user、order、payment 三個核心服務，分別部署在不同主機或容器中。

24.3.2 2. 遇到的問題與待解決的問題

- 用戶端需維護多個服務位址，呼叫複雜；
- 某執行個體當機時無法自動剔除，導致請求失敗；
- 缺乏統一日誌，排查問題困難。

待解決問題：統一 API 入口、自動負載平衡、集中可觀測性。

24.3.3 3. ALB 功能與設定與問題比對

- `upstream` + `server`：定義後端叢集；
- `proxy_pass`：反向代理到對應服務；

- `max_fails/fail_timeout` : 健康檢查機制;
- `access_log` : 統一記錄請求日誌。

24.3.4 4. 使用 ALB 解決該場景下的設定與詳解

```
upstream user_backend {
    server 10.0.0.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.11:8080 max_fails=3 fail_timeout=30s;
}

upstream order_backend {
    server 10.0.0.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.11:8080 max_fails=3 fail_timeout=30s;
}

upstream payment_backend {
    server 10.0.0.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.11:8080 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`：連續 3 次失敗後，30 秒內不再轉發請求到該節點；
- `X-Real-IP`：透傳真實用戶端 IP，便於後端稽核。

24.3.5 5. 效果，前後對比

- 用戶端呼叫位址從 3 個減少為 1 個；
- 服務執行個體故障時，錯誤率從 15% 降至 0.2%；
- 維運可透過單一日誌檔案追蹤所有 API 呼叫鏈。

24.4 場景三：全站 HTTPS 強制跳轉與 SSL 終止

24.4.1 1. 背景

某線上教育平台因 GDPR 合規要求，必須啟用 HTTPS。目前同時開放 HTTP 和 HTTPS，憑證由 Java 應用管理。

24.4.2 2. 遇到的問題與待解決的問題

- 使用者可能透過 HTTP 存取，存在資料外洩風險；
- 後端應用處理 SSL 加解密，CPU 開銷大；
- 憑證更新需重啟應用，影響可用性。

待解決問題：強制 HTTPS、SSL 邊緣終止、零停機憑證更新。

24.4.3 3. ALB 功能與設定與問題比對

- `return 301`：HTTP 強制跳轉 HTTPS；
- `ssl_certificate`：載入 TLS 憑證；
- 支援熱重載（`./bin/reload-alb.sh`），無需中斷連線。

24.4.4 4. 使用 ALB 解決該場景下的設定與詳解

```

server {
    listen 80;

```

```

server_name learn.edu.com;
return 301 https://$host$request_uri; # 強制跳轉
}

upstream backend {
    server 10.0.0.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.0.11:8080 max_fails=3 fail_timeout=30s;
}

server {
    listen 443 ssl http2;
    server_name learn.edu.com;

    ssl_certificate /opt/ALB-V2.0.6-SE-
amd64/conf/learn.edu.com/fullchain.pem;
    ssl_certificate_key /opt/ALB-V2.0.6-SE-
amd64/conf/learn.edu.com/privkey.pem;

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

    location / {
        proxy_pass http://backend/;
        proxy_set_header X-Forwarded-Proto $scheme; # 告知後端原始協定
    }
}

```

24.4.5 5. 效果，前後對比

- HTTP 流量佔比從 40% 降至 0%;
- 後端 CPU 負載下降 18%;
- 憑證更新只需替換檔案並 reload，服務零中斷。

24.5 場景四：API 介面防刷與速率限制

24.5.1 1. 背景

某社交平台開放使用者資訊查詢介面 /api/profile/{id}，近期遭遇爬蟲高頻請求。

24.5.2 2. 遇到的問題與待解決的問題

- 資料庫 QPS 從 2000 暴增至 15000，回應逾時；
- 正常使用者請求被阻塞；
- 暫時封鎖 IP 影響合法開發者。

待解決問題：在不修改業務程式碼前提下，快速實施限流。

24.5.3 3. ALB 功能與設定與問題比對

- `limit_req_zone`：基於 IP 定義速率限制區域；
- `limit_req`：在 location 中套用限流策略；
- `burst + nodelay`：允許突發流量平滑處理。

24.5.4 4. 使用 ALB 解決該場景下的設定與詳解

```
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 次請求；
- `burst=10`：允許瞬間突發 10 次，避免正常使用者誤傷；
- 回傳 429 狀態碼，符合 RESTful 規範。

24.5.5 5. 效果，前後對比

- 惡意請求攔截率 99.6%；
- 資料庫 QPS 回落至 2200；
- 正常使用者無感知，開發者可依據 429 實現重試邏輯。

24.6 場景五：動靜分離與動態內容快取

24.6.1 1. 背景

影片平台課程詳情頁包含動態資料（學習進度、評論），但被頻繁存取。

24.6.2 2. 遇到的問題與待解決的問題

- 每次請求都穿透到 Java 後端，即使內容未變；
- Redis 快取策略複雜，命中率低；
- 尖峰期資料庫連線池耗盡。

待解決問題：對「相對靜態」的動態頁面做邊緣快取。

24.6.3 3. ALB 功能與設定與問題比對

- proxy_cache_path：定義快取儲存；
- proxy_cache_valid：設定快取有效期；
- proxy_cache_use_stale：故障時回傳過期快取，保障可用性。

24.6.4 4. 使用 ALB 解決該場景下的設定與詳解

```
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;
}
```

24.6.5 5. 效果，前後對比

- 課程頁快取命中率 72%；
- 後端 QPS 下降 65%；
- 故障期間使用者仍可看到「稍舊但可用」的頁面。

24.7 場景六：WebSocket 長連線代理

24.7.1 1. 背景

官網整合線上客服系統，使用 WebSocket 實現即時聊天。

24.7.2 2. 遇到的問題與待解決的問題

- 連線建立後 60 秒自動斷開；
- 訊息遺失嚴重，使用者體驗差；
- 無法與 HTTP 共用 443 連接埠。

待解決問題：穩定代理 WebSocket 長連線。

24.7.3 3. ALB 功能與設定與問題比對

- Upgrade 和 Connection 標頭透傳；
- proxy_read_timeout：延長讀取逾時時間。

24.7.4 4. 使用 ALB 解決該場景下的設定與詳解

```
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;
}
```

24.7.5 5. 效果，前後對比

- WebSocket 連線可穩定維持 24 小時；
- 訊息到達率從 82% 提升至 99.9%；
- 與 HTTPS 共用 443 連接埠，簡化網路策略。

24.8 場景七：防止惡意掃描與目錄遍歷攻擊

24.8.1 1. 背景

某政府資訊公開網站部署在公網，近期被自動化工具頻繁掃描 /admin/、/.git/、/backup.zip 等敏感路徑，嘗試取得內部設定或原始碼。

24.8.2 2. 遇到的問題與待解決的問題

- 攻擊請求佔用伺服器資源；
- 日誌中大量 404 請求干擾正常監控；
- 存在因誤暴露備份檔案導致資料外洩的風險。

待解決問題：在入口層攔截高危路徑存取，無需修改應用程式碼。

24.8.3 3. ALB 功能與設定與問題比對

- `location` 精確比對 + `deny all`：禁止存取敏感目錄；
- 正規表示式比對隱藏檔案（如 `.env`、`.git`）；
- 回傳 444（ALB 特有狀態碼，直接關閉連線，不回應）。

24.8.4 4. 使用 ALB 解決該場景下的設定與詳解

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

    # 禁止存取隱藏檔案
    location ~ /\. {
        deny all;
        return 444;
    }

    # 禁止存取常見敏感目錄
    location ~* ^/(admin|phpmyadmin|wp-admin|backup|config)/ {
        deny all;
        return 444;
    }

    # 禁止下載特定副檔名檔案
    location ~* \.(sql|bak|tar\.gz|zip)$ {
        deny all;
        return 444;
    }

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

- `return 444`：不回傳任何回應，直接斷開 TCP 連線，降低攻擊者探測效率；
- 正規表示式 `~*` 表示不區分大小寫，涵蓋更多變種路徑。

24.8.5 5. 效果，前後對比

- 每日惡意請求從 12,000+ 降至 <50；
- 安全稽核報告中「資訊外洩風險」項清零；
- 伺服器 CPU 在掃描尖峰期下降 30%。

24.9 場景八：基於 Cookie 的灰度發佈（A/B 測試）

24.9.1 1. 背景

電商平台計劃上線新版商品詳情頁，希望先對 10% 使用者開放新版本，驗證轉換率後再全量。

24.9.2 2. 遇到的問題與待解決的問題

- 後端無灰度能力，需閘道層分流；
- 要求同一使用者始終看到同一版本（工作階段一致性）；
- 不能依賴 IP（多使用者共用出口）。

待解決問題：基於使用者識別（如 Cookie）實現穩定灰度路由。

24.9.3 3. ALB 功能與設定與問題比對

- `map` 指令：根據 `$cookie_version` 值決定後端；
- 自訂變數控制流量比例；
- `sticky` 工作階段保持（透過 Cookie 實現）。

24.9.4 4. 使用 ALB 解決該場景下的設定與詳解

```
# 若使用者無 version cookie, 則隨機分配 (10% 機率設為 new)
map $cookie_version $backend_group {
    default "old";
    ""      "auto"; # 未設定時走自動分配
    "new"    "new";
}

upstream old_backend {
    server 10.0.0.10:8080;
}
```

```

upstream new_backend {
    server 10.0.0.20:8080;
}

server {
    listen 80;
    server_name shop.com;

    location /product/ {
        # 未分配版本的使用者：10% 機率打標為 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;
        }
    }
}

```

24.9.5 5. 效果，前後對比

- 新舊版本使用者隔離清晰，無交叉污染；
- A/B 測試週期從 2 週縮短至 3 天；
- 無需改造業務系統，灰度策略完全由 ALB 控制。

24.10 場景九：大檔案上傳逾時與分片支援最佳化

24.10.1 1. 背景

線上教育平台允許教師上傳課件（PPT、影片），部分檔案超過 500MB。

24.10.2 2. 遇到的問題與待解決的問題

- 預設 60 秒逾時導致大檔案上傳失敗；
- 使用者網路波動時需重新上傳整個檔案；
- 後端 Java 應用記憶體溢位（OOM）。

待解決問題：延長上傳逾時、支援中斷續傳、減輕後端壓力。

24.10.3 3. ALB 功能與設定與問題比對

- client_max_body_size：允許大請求主體；
- client_body_timeout / proxy_send_timeout：延長逾時；
- （可選）結合 alb-upload-module 或前端分片實現中斷續傳。

24.10.4 4. 使用 ALB 解決該場景下的設定與詳解

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

    # 允許最大 2GB 檔案
    client_max_body_size 2G;

    # 上傳相關逾時設為 1 小時
    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;
        # 暫存檔案儲存路徑（避免記憶體快取大檔案）
        client_body_temp_path /var/tmp/alb_upload;
    }
}
```

24.10.5 5. 效果，前後對比

- 500MB+ 檔案上傳成功率從 40% 提升至 99%;
- 後端 OOM 錯誤歸零;
- 使用者可在弱網環境下完成上傳。

24.11 場景十：靜態資源 Gzip 壓縮加速

24.11.1 1. 背景

企業官網包含大量 JS/CSS 檔案，海外使用者回饋載入緩慢。

24.11.2 2. 遇到的問題與待解決的問題

- 未啟用壓縮，文字資源體積大；
- 後端動態壓縮消耗 CPU；
- 行動端流量成本高。

待解決問題：在邊緣層對文字資源進行高效 Gzip 壓縮。

24.11.3 3. ALB 功能與設定與問題比對

- `gzip on`：啟用壓縮；
- `gzip_types`：指定壓縮 MIME 類型；
- `gzip_vary on`：通知代理快取不同版本。

24.11.4 4. 使用 ALB 解決該場景下的設定與詳解

```
http {
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;    # 小於 1KB 不壓縮
    gzip_comp_level 6;       # 平衡 CPU 與壓縮率
    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;

}

}
```

24.11.5 5. 效果，前後對比

資源類型	原始大小	Gzip 後	壓縮率
main.js	850 KB	210 KB	75% ↓
style.css	320 KB	85 KB	73% ↓

- 頁面整體載入時間減少 40%;
- 使用者跳出率下降 18%。

全國統一服務熱線
4008-555-800

金蝶天燕雲計算股份有限公司（簡稱“金蝶天燕”）成立於2000年，前身為“金蝶中間件公司”，是金蝶集團旗下基礎軟件平台提供商，雲計算國家標準制定企業，國家信創產業核心軟件企業。金蝶天燕是國家863重點研發計劃與核高基重大專項承接企業，也是“兩網一站四庫十二金”國家重點工程的基礎平台提供商，產品廣泛應用於政府、軍工、金融、能源等關鍵行業，累計服務客戶總數超過10萬家。



雲計算國家標準制定企業
金蝶集團旗下基礎軟件企業
資訊技術應用創新核心企業
官網：www.apusic.com

