



Developer Manual

Apusic Load Balancer

Copyright Notice

All intellectual property rights including software copyrights involved in this document have been legally registered and duly owned by Kingdee Apusic Cloud Computing Co., Ltd. Such rights are protected by Copyright Law of the People's Republic of China, Regulations on Protection of Computer Software, Regulations on Intellectual Property Protection, relevant international copyright conventions, laws, regulations and other intellectual property legislations. Unauthorized use is strictly prohibited.

Disclaimer

All copyright information contained herein is legally owned by Kingdee Apusic Cloud Computing Co., Ltd. and protected by applicable laws. The Company shall bear no liability for any information unrelated to Kingdee Apusic contained in this document. You may view, copy and print this document only within the scope permitted by Copyright Law of the People's Republic of China. No entity or individual may use, modify or republish any part or content of this document without written authorization from Kingdee Apusic Cloud Computing Co., Ltd. Any violation shall constitute infringement, and the Company reserves the right to pursue legal liabilities accordingly.

Updates to this document shall not be notified separately. You may consult the Company regarding issues related to this document. All rights not expressly granted by the Company are reserved.

Trademark Notice

 is a registered trademark filed with the National Intellectual Property Administration of PRC by Kingdee Apusic Cloud Computing Co., Ltd. Shenzhen. Exclusive rights to the registered trademark are legally owned by Kingdee Apusic and protected by law. Without written consent of Kingdee Apusic, no entity or individual shall use, copy, modify, disseminate, transcribe, or bundle and sell any part of this trademark with other products in any manner or for any reason. Kingdee Apusic shall pursue legal liabilities against any infringement of its trademark rights. All other trademarks or registered trademarks mentioned in this document belong to their respective owners.

Contents

- 1 Introduction
 - 1.1 Target Audience
 - 1.2 Related Documents
 - 1.3 Technical Support
- 2 Table of Contents
- 3 Overview
 - 3.1 Comparison of the Three Capabilities
 - 3.2 Reader Path
- 4 External API
 - 4.1 Common Conventions
 - 4.1.1 Authentication Method
 - 4.1.2 Request Format
 - 4.1.3 Response Format
 - 4.1.4 HA Cluster Behavior
 - 4.2 HTTP Site API
 - 4.2.1 Create HTTP Site
 - 4.2.2 Update HTTP Site
 - 4.2.3 Delete HTTP Site
 - 4.2.4 Enable HTTP Site
 - 4.2.5 Disable HTTP Site
 - 4.2.6 Get HTTP Site List
 - 4.2.7 Get HTTP Site Details
 - 4.2.8 Bind SSL Certificate to HTTP Site
 - 4.3 Stream Site API
 - 4.3.1 Create Stream Site
 - 4.3.2 Update Stream Site
 - 4.3.3 Delete Stream Site
 - 4.3.4 Enable / Disable Stream Site
 - 4.3.5 Get Stream Site List
 - 4.3.6 Get Stream Site Details
 - 4.4 Error Response Examples
 - 4.4.1 Authentication Failed (401)

- 4.4.2 External API Not Enabled (403)
- 4.4.3 HA Standby Node Write (10001)
- 4.4.4 Business Error (7)
- 4.5 Quick Verification Script
- 4.6 Audit Query
- 5 alb-cli Command-Line Tool
 - 5.1 Installation and Configuration
 - 5.1.1 Authentication Method
 - 5.1.2 Configuration Sources (Priority from High to Low)
 - 5.1.3 Verify Connection
 - 5.2 Global Parameters
 - 5.3 Output Format
 - 5.3.1 JSON (Default)
 - 5.3.2 Table
 - 5.3.3 Error Output
 - 5.4 Command Reference
 - 5.4.1 `site` — Site Management
 - 5.4.2 `status` — ALB Running Status
 - 5.4.3 `metrics` — Monitoring Data
 - 5.4.4 `config reload` — Reload ALB Configuration
 - 5.4.5 `log` — Log Viewing
 - 5.5 Security Policy
 - 5.5.1 Dangerous Operation Confirmation
 - 5.5.2 Non-Interactive Environment
 - 5.6 Typical Workflow
 - 5.6.1 Publish an HTTP Site
 - 5.6.2 Enable HTTPS for a Site
 - 5.6.3 Daily Inspection
 - 5.6.4 Troubleshooting
 - 5.7 Backend API Mapping
- 6 AI Agent Integration
 - 6.1 SKILL Purpose
 - 6.2 SKILL Installation
 - 6.2.1 Install in Claude Code

- 6.2.1.1 Method 1: Copy to the Project Skill Directory (Recommended)
- 6.2.1.2 Method 2: Create Manually
- 6.2.1.3 Verify Installation
- 6.3 Using in Other Agent Tools
 - 6.3.1 OpenCode
 - 6.3.2 General Compatible Tools
- 6.4 Prerequisites
- 6.5 Trigger Method
- 6.6 Quick Reference of Available Commands
- 6.7 Default Output and Parsing
- 6.8 Security Constraints
- 6.9 Standard Workflows
 - 6.9.1 Workflow 1: Publish an HTTP Site
 - 6.9.2 Workflow 2: Enable HTTPS
 - 6.9.3 Workflow 3: Daily Inspection
 - 6.9.4 Workflow 4: Troubleshooting
- 6.10 JSON File Templates
- 6.11 Common Error Handling
- 6.12 Best Practices
- 7 Operations and Monitoring APIs
 - 7.1 ALB Running Status
 - 7.2 Traffic and Node Monitoring
 - 7.3 Configuration Reload
 - 7.4 Log Query
- 8 Typical Integration Scenarios
 - 8.1 Scenario 1: CI/CD Automatic Release of HTTP Sites
 - 8.2 Scenario 2: Integration with ERP / Business Systems
 - 8.3 Scenario 3: Daily Inspection Bot (AI Agent)
 - 8.4 Scenario 4: Automatic Fault Localization
- 9 Integration Recommendations
- 10 Error Code Quick Reference
 - 10.1 External API Error Codes
 - 10.2 Common alb-cli Errors
- 11 Best Practices and Troubleshooting

- [11.1 Best Practices](#)
- [11.2 Troubleshooting](#)
- [12 Version Update Notes](#)

1 Introduction

This document is the Developer Manual for Apusic Load Balancer software V2.0.6 Standard Edition. It is intended for operations and development personnel, and provides a detailed introduction to the **External API**, the **alb-cli command-line tool**, and the integration capabilities for **AI Agents (Claude, OpenCode, etc.)** exposed by the ALB Console. It helps customers implement automated operations, CI/CD releases, platform integration, and AI-assisted operations scenarios based on ALB.

1.1 Target Audience

This document applies to:

- ALB secondary development and integration engineers
- Operations automation platform / DevOps platform development engineers
- SRE engineers introducing AI Agents for daily operations
- Integration parties for business systems such as ERP, operations monitoring, and business release
- System architects and technical leads at customer organizations

1.2 Related Documents

For more information about ALB V2.0.6 products, please refer to the following ALB V2.0.6 product manual document set:

No.	Manual Document	Description
1	ALB V2.0.6 Release Notes	Introduces the version updates of the ALB product.
2	ALB V2.0.6 Quick Start Guide	Briefly introduces how to quickly get started with ALB.
3	ALB V2.0.6 Installation Guide	Details how to install ALB on various operating systems, ALB service start/stop operations, and the product registration process.
4	ALB V2.0.6 Proxy Function Manual	Details the use, configuration, management of ALB-related functions, and the use of supporting tools.
5	ALB V2.0.6 Console User Manual	Details the use and operational instructions for ALB Console-related functions.
6	ALB V2.0.6 Developer Manual	Details the ALB External API, alb-cli command-line tool, and AI Agent integration methods.

7	ALB V2.0.6 Migration Guide	Details instructions for migrating and upgrading from historical ALB versions to V2.0.6.
8	ALB V2.0.6 Operations Manual	Details monitoring, operations, and security hardening instructions for ALB.
9	ALB V2.0.6 Performance Optimization Guide	Details ALB performance tuning instructions.

1.3 Technical Support

ALB products provide comprehensive technical support services. You can obtain technical support through the following channels:

- Website: www.apusic.com
- Phone: 400-855-5800
- Email: support@apusic.com
- Kingdee Cloud Community: <https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN>

When obtaining technical support, please provide the following information:

1. Your name
2. Company information and contact details
3. Operating system and its version
4. Product version number
5. Detailed information such as logs and screenshots of the exception or error

2 Table of Contents

- [Overview](#)
 - [External API](#)
 - [alb-cli Command-Line Tool](#)
 - [AI Agent Integration](#)
 - [Operations and Monitoring APIs](#)
 - [Typical Integration Scenarios](#)
 - [Integration Recommendations](#)
 - [Error Code Quick Reference](#)
 - [Best Practices and Troubleshooting](#)
-

3 Overview

ALB V2.0.6 Standard Edition, on top of the Console, opens up three capabilities for developers and operations personnel:

Capability	Form	Applicable Scenarios
External Site API	HTTPS RESTful API (API Key authentication)	Integration with business systems, operations platforms, and CI/CD pipelines
alb-cli	Single-binary command-line tool	Operations scripts, automation tasks, manual quick operations
AI Agent SKILL	SKILL descriptions for Agents such as Claude and OpenCode	Introducing AI assistants for daily operations and troubleshooting

3.1 Comparison of the Three Capabilities

Dimension	External API	alb-cli	AI Agent SKILL
Invocation Method	HTTP/JSON	Command line	Natural language + tool calling
Authentication	X-API-Key	ALB_API_KEY environment variable	ALB_API_KEY environment variable
Suitable Integrators	Backend services, ERP	Operations, DevOps	AI assistant scenarios
Output Format	JSON	JSON / Table	Parse JSON then provide answers
Dangerous Operations	Requires manual confirmation	Built-in interactive confirmation	Built-in user confirmation + non-TTY rejection

Note: All three capabilities share the same underlying Console API with consistent call semantics. Developers can choose the most appropriate form based on the integration scenario.

3.2 Reader Path

- **Backend Developers:** Start with [External API](#), focusing on common conventions, HTTP sites, Stream sites, and HA cluster behavior.
- **Operations / DevOps:** Start with [alb-cli Command-Line Tool](#) and [Typical Integration Scenarios](#).
- **AI Integrators:** Start with [AI Agent Integration](#) and the troubleshooting flow in [Typical Integration Scenarios](#).

4 External API

Version: v1.0 Base Path: /api/alb/console/external/v1 Applicable Version: ALB V2.0.6+

4.1 Common Conventions

4.1.1 Authentication Method

All External APIs pass the API Key through the HTTP Header:

```
X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

The API Key is configured in [ALB Installation Directory]/console/conf/config.yaml :

```
external:
  enabled: true
  api_keys:
    - name: "erp-system"
      key: "alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

Security Tip: Keep your API Key secure. Access the Console over HTTPS in production environments; it is recommended to assign different API Keys to different business systems for easier auditing and selective revocation.

4.1.2 Request Format

- Request bodies uniformly use application/json
- The path parameter :id is the site ID (number)
- The locations , upstreams , backend_servers fields are JSON strings (the backend stores them as strings). When making a request, serialize the array as a JSON string, such as "[{"path":"/","type":"proxy"}]"

4.1.3 Response Format

```
{
  "code": 0,
  "data": { },
}
```

```
"msg": "ok"
}
```

code	Meaning
0	Success
7	General business error
401	Authentication failed (missing API Key / invalid Key)
403	External API not enabled
10001	The current node is an HA standby node; writes are forbidden

4.1.4 HA Cluster Behavior

Node Status	Read Operations (GET)	Write Operations (POST/PUT/DELETE)
HA not enabled	✓ Normal	✓ Normal
HA enabled + Primary node	✓ Normal	✓ Normal + Automatically synced to standby node
HA enabled + Standby node	✓ Normal	✗ Returns code=10001

Tip: Read operations on the HA standby node are not automatically forwarded to the primary node. To obtain the latest data from the standby node, send the request to the primary node, or implement retries on the client side.

4.2 HTTP Site API

4.2.1 Create HTTP Site

POST `/sites/http`

Request Parameters

Field	Type	Required	Description
name	string	Yes	Site name
domains	string	Yes	Listening domain

listen_port	int	Yes	Listening port
locations	string(JSON)	No	Location configuration list (JSON string)
upstreams	string(JSON)	No	Upstream configuration list (JSON string); required when <code>upstream_name</code> is used in locations
ssl_cert_id	int	No	SSL certificate ID
auto_reload	bool	No	Automatically reload after save, default true
status	string	No	<code>enabled</code> / <code>disabled</code> , default <code>enabled</code>

Location Field Description

Field	Type	Required	Description
path	string	Yes	Match path, e.g. <code>/</code> or <code>/api</code>
type	string	Yes	<code>proxy</code> / <code>static</code> / <code>redirect</code>
proxy_pass	string	No	Direct backend address, e.g. <code>http://192.168.1.10:8080</code>
upstream_name	string	No	Reference to an existing upstream name (must be defined in <code>upstreams</code> simultaneously)

Note: Choose either `proxy_pass` or `upstream_name`. When using `upstream_name`, you must define the upstream in the `upstreams` field at the same time; otherwise the nginx reload will fail.

Request Example 1: Direct Proxy (Simplest, No Upstream Required)

```
curl -X POST "http://alb-master:8887/api/alb/console/external/v1/sites/http" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{
  "name": "test-api",
  "domains": "www.example.com",
  "listen_port": 8088,
  "locations": "
[{"path":"/","type":"proxy","proxy_pass":"http://192.168.1.1
"}]",
  "upstreams": "[]",
}
```

```
"custom_headers": "[]",
"proxy_set_headers": "[]",
"extra_directives": "[]",
"auto_reload": true,
"status": "enabled"
}'
```

Request Example 2: Use Upstream (Supports Load Balancing)

```
curl -X POST "http://alb-
master:8887/api/alb/console/external/v1/sites/http" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{
  "name": "test-upstream",
  "domains": "www.example.com",
  "listen_port": 8090,
  "upstreams": "[{\"name\": \"backend-
web\", \"balance_method\": \"round-robin\", \"servers\":
[\"address\": \"192.168.1.10:8080\", \"weight\": 1},
{ \"address\": \"192.168.1.11:8080\", \"weight\": 1}]]]",
  "locations": "
[\"path\": \"/\", \"type\": \"proxy\", \"upstream_name\": \"backend-
web\", \"proxy_timeout\": 60, \"support_web_socket\": false, \"use_alias\": f
[]]]",
  "custom_headers": "[]",
  "proxy_set_headers": "[]",
  "extra_directives": "[]",
  "auto_reload": true,
  "status": "enabled"
}'
```

Response Example

```
{
  "code": 0,
  "data": {
    "id": 42,
    "name": "test-upstream",
    "domains": "www.example.com",
    "listen_port": 8090,
    "status": "enabled"
  },
  "msg": "Created successfully"
}
```

4.2.2 Update HTTP Site

PUT /sites/http

Request Parameters

Field	Type	Required	Description
id	int	Yes	Site ID
name	string	No	Site name
domains	string	No	Listening domain
listen_port	int	No	Listening port
locations	string(JSON)	No	Location configuration list (JSON string)
upstreams	string(JSON)	No	Upstream configuration list (JSON string)
ssl_cert_id	int	No	SSL certificate ID
auto_reload	bool	No	Automatically reload after save
status	string	No	enabled / disabled

Request Example

```
curl -X PUT "http://alb-master:8887/api/alb/console/external/v1/sites/http" \
```

```
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{
  "id": 42,
  "name": "test-api-update",
  "listen_port": 8080
}'
```

Response Example

```
{
  "code": 0,
  "data": null,
  "msg": "Updated successfully"
}
```

4.2.3 Delete HTTP Site

DELETE /sites/http

Request Parameters

Field	Type	Required	Description
id	int	Yes	Site ID

Request Example

```
curl -X DELETE "http://alb-master:8887/api/alb/console/external/v1/sites/http" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{
  "id": 42
}'
```

Response Example

```
{
  "code": 0,
  "data": null,
  "msg": "Deleted successfully"
}
```

4.2.4 Enable HTTP Site

POST `/sites/http/{id}/enable`

```
curl -X POST "http://alb-master:8887/api/alb/console/external/v1/sites/http/42/enable" \
  -H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

Response Example

```
{
  "code": 0,
  "data": null,
  "msg": "Enabled successfully"
}
```

4.2.5 Disable HTTP Site

POST `/sites/http/{id}/disable`

```
curl -X POST "http://alb-master:8887/api/alb/console/external/v1/sites/http/42/disable" \
  -H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

Response Example

```
{
  "code": 0,
  "data": null,
```

```
"msg": "Disabled successfully"
}
```

4.2.6 Get HTTP Site List

GET `/sites/http`

Query Parameters

Field	Type	Required	Description
page	int	No	Page number, default 1
pageSize	int	No	Items per page, default 10
name	string	No	Fuzzy search by name

Request Example

```
curl "http://alb-master:8887/api/alb/console/external/v1/sites/http?
page=1&pageSize=10" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

Response Example

```
{
  "code": 0,
  "data": [
    {
      "id": 42,
      "name": "Official Website",
      "domains": "www.example.com",
      "listen_port": 80,
      "status": "enabled",
      "created_at": "2024-01-15 10:30:00"
    }
  ],
}
```

```
"msg": "ok"
}
```

4.2.7 Get HTTP Site Details

GET /sites/http/{id}

```
curl "http://alb-master:8887/api/alb/console/external/v1/sites/http/42" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

Response Example

```
{
  "code": 0,
  "data": {
    "id": 42,
    "name": "Official Website",
    "domains": "www.example.com",
    "listen_port": 80,
    "locations": [
      {
        "path": "/",
        "type": "proxy",
        "upstream_name": "backend-web"
      }
    ],
    "status": "enabled",
    "created_at": "2024-01-15 10:30:00"
  },
  "msg": "ok"
}
```

4.2.8 Bind SSL Certificate to HTTP Site

POST /sites/http/bind-ssl

Request Parameters

Field	Type	Required	Description
site_id	int	Yes	Site ID
cert_id	int	Yes	Certificate ID

```
curl -X POST "http://alb-master:8887/api/alb/console/external/v1/sites/http/bind-ssl" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{
  "site_id": 42,
  "cert_id": 2
}'
```

4.3 Stream Site API

Stream sites are used for Layer 4 load balancing (TCP/UDP). The API structure is consistent with HTTP sites.

4.3.1 Create Stream Site

POST /sites/stream

Request Parameters

Field	Type	Required	Description
name	string	Yes	Site name
listen_port	int	Yes	Listening port
protocol	string	Yes	TCP / UDP
backend_servers	string(JSON)	Yes	Backend server list (JSON string)
auto_reload	bool	No	Automatically reload after save, default true
status	string	No	enabled / disabled, default enabled

BackendServer Field Description

Field	Type	Required	Description
address	string	Yes	Backend address, e.g. 192.168.1.10:3306
weight	int	No	Weight, default 1
max_fails	int	No	Maximum failure count
fail_timeout	int	No	Failure timeout (seconds)
backup	bool	No	Whether it is a backup server

Request Example

```
curl -X POST "http://alb-master:8887/api/alb/console/external/v1/sites/stream" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{
  "name": "Database Proxy",
  "listen_port": 3306,
  "protocol": "TCP",
  "backend_servers": "
[{\\"address\\":\\"192.168.1.10:3306\\",\\"weight\\":1}],
  "auto_reload": true,
  "status": "enabled"
}'
```

Response Example

```
{
  "code": 0,
  "data": {
    "id": 15,
    "name": "Database Proxy",
    "listen_port": 3306,
    "protocol": "TCP",
    "status": "enabled"
  },
}
```

```
"msg": "Created successfully"
}
```

4.3.2 Update Stream Site

PUT /sites/stream

```
curl -X PUT "http://alb-master:8887/api/alb/console/external/v1/sites/stream" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{
  "id": 15,
  "backend_servers": "
[{"address":"192.168.1.11:3306","weight":1}]"
}'
```

4.3.3 Delete Stream Site

DELETE /sites/stream

```
curl -X DELETE "http://alb-master:8887/api/alb/console/external/v1/sites/stream" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" \
-H "Content-Type: application/json" \
-d '{"id": 15}'
```

4.3.4 Enable / Disable Stream Site

POST /sites/stream/{id}/enable

POST /sites/stream/{id}/disable

```
curl -X POST "http://alb-master:8887/api/alb/console/external/v1/sites/stream/15/enable" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

```
curl -X POST "http://alb-master:8887/api/alb/console/external/v1/sites/stream/15/disable" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

4.3.5 Get Stream Site List

GET /sites/stream?page=1&pageSize=10

```
curl "http://alb-master:8887/api/alb/console/external/v1/sites/stream?page=1&pageSize=10" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

4.3.6 Get Stream Site Details

GET /sites/stream/{id}

```
curl "http://alb-master:8887/api/alb/console/external/v1/sites/stream/15" \
-H "X-API-Key: alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

4.4 Error Response Examples

4.4.1 Authentication Failed (401)

```
{
  "code": 401,
  "msg": "invalid api key"
}
```

4.4.2 External API Not Enabled (403)

```
{
  "code": 403,
```

```
"msg": "External API not enabled"
}
```

4.4.3 HA Standby Node Write (10001)

```
{
  "code": 10001,
  "msg": "The current node is an HA standby node. Please go to the
primary node to execute."
}
```

4.4.4 Business Error (7)

```
{
  "code": 7,
  "msg": "Site name already exists"
}
```

4.5 Quick Verification Script

```
#!/bin/bash

API_KEY="alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
BASE_URL="http://alb-master:8887/api/alb/console/external/v1"

echo "=== 1. Create HTTP site (direct proxy) ==="
curl -s -X POST "$BASE_URL/sites/http" \
  -H "X-API-Key: $API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Test Site",
    "domains": "test.example.com",
    "listen_port": 80,
```

```

    "locations": "
[{\\"path\\":\\\"/\\\",\\\"type\\\":\\\"proxy\\\",\\\"proxy_pass\\\":\\\"http://192.168.1.1
[]\\\"},
    "upstreams": "[]",
    "custom_headers": "[]",
    "proxy_set_headers": "[]",
    "extra_directives": "[]",
    "auto_reload": true,
    "status": "enabled"
}' | jq .

echo "=== 2. Get list ==="
curl -s "$BASE_URL/sites/http?page=1&pageSize=5" \
-H "X-API-Key: $API_KEY" | jq .

echo "=== 3. Disable site ==="
curl -s -X POST "$BASE_URL/sites/http/1/disable" \
-H "X-API-Key: $API_KEY" | jq .

echo "=== 4. Delete site ==="
curl -s -X DELETE "$BASE_URL/sites/http" \
-H "X-API-Key: $API_KEY" \
-H "Content-Type: application/json" \
-d '{"id": 1}' | jq .

```

4.6 Audit Query

External API call records are stored in the `sys_operation_logs` table and can be queried via the following SQL:

```

SELECT
    created_at,
    operator_name,
    oper_type,
    request_method,
    request_path,

```

```
status
FROM sys_operation_logs
WHERE operator_name LIKE 'external:%'
ORDER BY id DESC
LIMIT 20;
```

Example output:

created_at	operator_name	oper_type	request_method	request_path	s
2024-01-15 10:30:01	external:erp-system	External API: Create HTTP Site	POST	/api/alb/console/external/v1/sites/http	

5 alb-cli Command-Line Tool

Version: V2.0.6 Positioning: A command-line entry point to the ALB Console for AI Agents such as Claude and OpenCode

5.1 Installation and Configuration

The ALB V2.0.6 product package comes with the `alb-cli` executable program, located at `[ALB Installation Path]/utils/alb-cli`.

5.1.1 Authentication Method

`alb-cli` communicates with the Console via **API Key**, using the `x-API-Key` request header.

Configure the API Key: Configure `external.api_keys` in the backend configuration file `[ALB Installation Directory]/console/conf/config.yaml`, or set it via the `ALB_API_KEY` environment variable.

5.1.2 Configuration Sources (Priority from High to Low)

1. Environment variables
2. Configuration file `~/.alb/config.yaml`

Environment Variables

```
export ALB_CONSOLE_URL=https://172.21.61.46:8887
export ALB_API_KEY=your-api-key
```

Configuration File

```
# ~/.alb/config.yaml
url: https://172.21.61.46:8887
api_key: your-api-key
```

5.1.3 Verify Connection

```
alb-cli status
```

5.2 Global Parameters

Parameter	Description	Default
<code>--output</code>	Output format: json OR table	json
<code>-h, --help</code>	Show help	-

Global parameters can be placed before or after subcommands:

```
alb-cli --output table site list --type http
alb-cli site list --type http --output table
```

5.3 Output Format

5.3.1 JSON (Default)

All query commands output JSON by default, which is convenient for Agent parsing.

```
alb-cli site list --type http
```

Example output:

```
[
  {
    "id": 1,
    "name": "example",
    "domains": "example.com",
    "listen_port": 80,
    "enable_ssl": false,
    "status": "enabled"
  }
]
```

5.3.2 Table

Suitable for human reading.

```
alb-cli site list --type http --output table
```

Example output:

```
ID    NAME      PORT  STATUS  DOMAINS
1     example   80    enabled example.com
```

5.3.3 Error Output

All errors are uniformly output to `stderr` :

```
missing API key: set ALB_API_KEY or ~/.alb/config.yaml api_key
```

Errors returned by the server are also printed directly as messages.

5.4 Command Reference

5.4.1 `site` — Site Management

```
alb-cli site [subcommand] --type http|stream [parameters]
```

Global flags:

Flag	Description	Default
<code>--type</code>	Site type: http Or stream	http
<code>--status</code>	Filter by status: enabled / disabled	-

`site list` — List sites

```
alb-cli site list --type http
alb-cli site list --type http --status enabled
alb-cli site list --type stream --output table
```

`site get` — View site details

```
alb-cli site get 1 --type http
```

site create — Create a site from a JSON file

```
alb-cli site create --type http --file site.json
```

site.json example (HTTP):

```
{
  "name": "example",
  "domains": "example.com www.example.com",
  "listen_port": 80,
  "enable_ssl": false,
  "locations": "[{ \"path\": \"/\", \"type\": \"proxy\", \"proxy_pass\": \"http://127.0.0.1:8080\" }]"
}
```

site.json example (Stream):

```
{
  "name": "tcp-example",
  "listen_port": 9090,
  "protocol": "TCP",
  "backend_servers": "[{ \"address\": \"127.0.0.1:8080\", \"weight\": 1 }]"
}
```

site update — Update a site from a JSON file

```
alb-cli site update 1 --type http --file site.json
```

The JSON file does not need to include `id`; the CLI will automatically write the ID passed in from the command line into the request body.

site enable / **site disable** — Enable/disable a site

```
alb-cli site enable 1 --type http
alb-cli site disable 1 --type stream
```

site delete — Delete a site

```
alb-cli site delete 1 --type http
```

site bind-ssl — Bind an SSL certificate to an HTTP site

```
alb-cli site bind-ssl --site-id 1 --cert-id 2
```

5.4.2 **status** — ALB Running Status

```
alb-cli status
```

Returns the current ALB running status, listening ports, number of servers, etc.

5.4.3 **metrics** — Monitoring Data

metrics traffic — Traffic Trend

```
alb-cli metrics traffic --metric-type min    # Past 1 hour
alb-cli metrics traffic --metric-type hour  # Past 24 hours
```

metrics nodes — Multi-node status snapshot

```
alb-cli metrics nodes
```

5.4.4 **config reload** — Reload ALB Configuration

```
alb-cli config reload
```

After execution, ALB will reload the configuration file, making site changes take effect.

5.4.5 log — Log Viewing

`log list` — List of log files

```
alb-cli log list
```

`log show` — View complete log content

```
alb-cli log show access.log
```

`log tail` — View log tail

```
alb-cli log tail access.log -n 100
```

5.5 Security Policy

5.5.1 Dangerous Operation Confirmation

The following commands require terminal interactive confirmation by default:

- `site create`
- `site update`
- `site enable`
- `site disable`
- `site delete`
- `site bind-ssl`
- `config reload`

Example interaction:

```
$ alb-cli config reload
reload ALB configuration? [y/N]: y
```

5.5.2 Non-Interactive Environment

If `stdout` or `stdin` is not a TTY, write operations will be rejected:

```
destructive operation requires an interactive terminal
```

This prevents Agents from accidentally triggering destructive operations in CI / scripts.

5.6 Typical Workflow

5.6.1 Publish an HTTP Site

```
# 1. Create the site
alb-cli site create --type http --file site.json

# 2. Reload the configuration for it to take effect
alb-cli config reload

# 3. Verify site status
alb-cli site list --type http --output table
```

5.6.2 Enable HTTPS for a Site

```
# 1. Bind certificate
alb-cli site bind-ssl --site-id 1 --cert-id 2

# 2. Reload configuration
alb-cli config reload
```

5.6.3 Daily Inspection

```
alb-cli status
alb-cli metrics traffic --metric-type min
alb-cli metrics nodes
```

5.6.4 Troubleshooting

```
alb-cli log list
alb-cli log tail error.log -n 50
```

5.7 Backend API Mapping

CLI Command	Backend API
site list --type http	GET /api/alb/console/external/v1/sites/http
site get 1 --type http	GET /api/alb/console/external/v1/sites/http/1
site create --type http	POST /api/alb/console/external/v1/sites/http
site update 1 --type http	PUT /api/alb/console/external/v1/sites/http
site enable 1 --type http	POST /api/alb/console/external/v1/sites/http/1/enable
site disable 1 --type http	POST /api/alb/console/external/v1/sites/http/1/disable
site delete 1 --type http	DELETE /api/alb/console/external/v1/sites/http?id=1
site bind-ssl	POST /api/alb/console/external/v1/sites/http/bind-ssl
status	GET /api/alb/console/external/v1/alb/status
metrics traffic	GET /api/alb/console/external/v1/alb/flow?type=min
metrics nodes	GET /api/alb/console/external/v1/analytic/nodes
config reload	POST /api/alb/console/external/v1/config/reload
log list	GET /api/alb/console/external/v1/alb/logger/name
log show	GET /api/alb/console/external/v1/alb/logger/text?logger_name=...

6 AI Agent Integration

ALB provides standard SKILL descriptions for **AI Agents** (Claude, OpenCode, etc.), enabling Agents to invoke ALB Console capabilities safely and controllably.

6.1 SKILL Purpose

By interacting with the ALB Console via `alb-cli`, AI Agents can accomplish:

- Site management (create, update, enable/disable, delete, SSL binding)
- Status monitoring (running status, traffic trends, node health)
- Configuration reload (effective immediately after modification)
- Log troubleshooting (list / show / tail)

6.2 SKILL Installation

`[ALB Installation Path]/utils/alb-cli/alb-cli/SKILL.md` is the Skill definition file for AI Agents such as Claude Code and OpenCode. Once installed, Agents can invoke `alb-cli` through natural language to complete ALB operations tasks.

6.2.1 Install in Claude Code

Claude Code automatically loads the Skill at `SKILL.md` under `.claude/skills/<skill-name>/` in the project root directory.

6.2.1.1 Method 1: Copy to the Project Skill Directory (Recommended)

```
# Copy from the source directory to the .claude/skills/ in the
project root directory
cp -r [ALB Installation Path]/utils/alb-cli/alb-cli/alb-cli
.claude/skills/
```

Directory structure after copying:

```
.claude/skills/
└─ alb-cli/
    └─ SKILL.md
```

6.2.1.2 Method 2: Create Manually

Execute in the project root directory:

```
mkdir -p .claude/skills/alb-cli
cp [ALB Installation Path]/utils/alb-cli/alb-cli/alb-cli/SKILL.md
.claude/skills/alb-cli/SKILL.md
```

6.2.1.3 Verify Installation

Restart Claude Code or enter the following in a conversation:

```
/alb-cli view status
```

If Claude Code recognizes the Skill, it will automatically execute `alb-cli status` and return the ALB running status.

6.3 Using in Other Agent Tools

6.3.1 OpenCode

OpenCode follows the same `.claude/skills/<skill-name>/SKILL.md` directory convention. After copying via Method 1, it can be used directly; the trigger words are the same as Claude Code.

6.3.2 General Compatible Tools

Any Agent tool that supports Markdown Skill definitions can call the commands defined in `SKILL.md` by reading the `description` and `allowed-tools`. The installation method is usually:

1. Copy `cli/docs/alb-cli/SKILL.md` to the skill directory specified by that tool
2. Ensure the tool can access the `alb-cli` binary (packaged with the Console under `console/tools/`, or built separately into `cli/alb-cli`)
3. Configure environment variables or `~/.alb/config.yaml`

6.4 Prerequisites

Before using the Skill, ensure the target environment meets:

1. **ALB Console is accessible:** Configure `ALB_CONSOLE_URL`, for example `http://127.0.0.1:8887`
2. **API Key is valid:** Configure `ALB_API_KEY`; this key must match an entry in `external.api_keys` in the backend `config/config.yaml`

Example environment variables:

```
export ALB_CONSOLE_URL=http://127.0.0.1:8887
export
ALB_API_KEY=9489ebf25ac6664c242e0b2f2b406164d4175ae227bd7b09db762d66c58fa32b
```

Or configuration file `~/.alb/config.yaml` :

```
url: http://127.0.0.1:8887
api_key:
9489ebf25ac6664c242e0b2f2b406164d4175ae227bd7b09db762d66c58fa32b
```

6.5 Trigger Method

Triggered when the user inputs `/alb-cli` or similar commands, for example:

- "Help me check the current ALB status"
- "List all HTTP sites"
- "Create a new site"
- "Reload ALB configuration"
- "View the last 100 lines of error.log"

6.6 Quick Reference of Available Commands

Task	Command
View running status	<code>alb-cli status</code>
List HTTP sites	<code>alb-cli site list --type http</code>
List Stream sites	<code>alb-cli site list --type stream</code>
View site details	<code>alb-cli site get <id> --type http</code>
Create a site	<code>alb-cli site create --type http --file site.json</code>
Update a site	<code>alb-cli site update <id> --type http --file site.json</code>
Enable/disable a site	<code>alb-cli site enable <id> --type http / alb-cli site disable <id> --type http</code>
Delete a site	<code>alb-cli site delete <id> --type http</code>

Bind SSL certificate	<code>alb-cli site bind-ssl --site-id <id> --cert-id <id></code>
View traffic	<code>alb-cli metrics traffic --metric-type min</code>
View node status	<code>alb-cli metrics nodes</code>
Reload configuration	<code>alb-cli config reload</code>
Log list	<code>alb-cli log list</code>
View log	<code>alb-cli log show <name></code>
Tail log	<code>alb-cli log tail <name> -n <lines></code>

6.7 Default Output and Parsing

- Default output is `json` ; the Agent should prefer JSON parsing.
- Add `--output table` for human-readable tables.
- Error messages are output to `stderr` .

6.8 Security Constraints

Write operations require terminal confirmation. The following commands will prompt `[Y/N]` when executed:

- `site create`
- `site update`
- `site enable`
- `site disable`
- `site delete`
- `site bind-ssl`
- `config reload`

If the current environment is not an interactive terminal (stdout/stdin is not a TTY), these commands will fail and return:

```
destructive operation requires an interactive terminal
```

Before calling these commands, the Agent must first confirm with the user, then let the user execute them in the terminal, or require the user to explicitly authorize.

6.9 Standard Workflows

6.9.1 Workflow 1: Publish an HTTP Site

```
# 1. Create the site (execute after user confirmation)
alb-cli site create --type http --file site.json

# 2. Reload the configuration (execute after user confirmation)
alb-cli config reload

# 3. Verify
alb-cli site list --type http
```

6.9.2 Workflow 2: Enable HTTPS

```
# Execute after user confirmation
alb-cli site bind-ssl --site-id 1 --cert-id 2
alb-cli config reload
```

6.9.3 Workflow 3: Daily Inspection

```
alb-cli status
alb-cli metrics traffic --metric-type min
alb-cli metrics nodes
```

6.9.4 Workflow 4: Troubleshooting

```
alb-cli status
alb-cli log list
alb-cli log tail error.log -n 50
```

6.10 JSON File Templates

JSON file example when creating/updating an HTTP site:

```
{
  "name": "example",
  "domains": "example.com www.example.com",
  "listen_port": 80,
  "enable_ssl": false,
  "locations": "
[{\\"path\\":\\"/\\",\\"type\\":\\"proxy\\",\\"proxy_pass\\":\\"http://127.0.0.1:8080\\"}]
}
```

Stream site example:

```
{
  "name": "tcp-example",
  "listen_port": 9090,
  "protocol": "TCP",
  "backend_servers": "
[{\\"address\\":\\"127.0.0.1:8080\\",\\"weight\\":1}]
}
```

After the Agent generates the JSON file, it can call:

```
alb-cli site create --type http --file /tmp/site.json
```

6.11 Common Error Handling

Error	Handling
missing console URL	Prompt the user to configure <code>ALB_CONSOLE_URL</code> Or <code>~/.alb/config.yaml</code>
missing API key	Prompt the user to configure <code>ALB_API_KEY</code>
invalid api key	Prompt the user to regenerate the API Key in the Console and enable the External API
destructive operation requires an interactive terminal	Explain that this command requires user confirmation; please execute manually or switch to an interactive terminal

```
alb status module not enabled
```

Prompt the user to enable the status module in the Console before querying traffic

6.12 Best Practices

1. **Query before modify:** Before creating/updating/deleting a site, first run `site list` to confirm the current state.
2. **Reload after modify:** After site change operations (create/update/enable/disable/delete/bind-ssl), it is recommended to execute `config reload`.
3. **Prefer JSON:** Use the default `--output json` when the Agent is parsing.
4. **Avoid batch dangerous operations:** Do not loop through delete/disable/reload without user confirmation.
5. **File cleanup:** Use the `/tmp/` directory for temporary JSON files; they can be deleted after the task is completed.

7 Operations and Monitoring APIs

This section summarizes the operations and monitoring capabilities exposed by ALB for integration with operations platforms and monitoring systems.

7.1 ALB Running Status

API/Command	Description
GET <code>/api/alb/console/external/v1/alb/status</code>	Current ALB node running status, listening ports, number of servers
<code>alb-cli status</code>	CLI entry, equivalent to the above API

7.2 Traffic and Node Monitoring

API/Command	Description
GET <code>/api/alb/console/external/v1/alb/flow?type=min</code>	Traffic trend for the past 1 hour
GET <code>/api/alb/console/external/v1/alb/flow?type=hour</code>	Traffic trend for the past 24 hours
GET <code>/api/alb/console/external/v1/analytic/nodes</code>	Multi-node status snapshot (used in HA clusters)
<code>alb-cli metrics traffic / alb-cli metrics nodes</code>	CLI entry

Tip: Before calling `metrics traffic`, please confirm that the ALB status module (`stub_status`) is enabled.

7.3 Configuration Reload

API/Command	Description
POST <code>/api/alb/console/external/v1/config/reload</code>	Notify ALB to reload configuration
<code>alb-cli config reload</code>	CLI entry, triggers interactive confirmation

Note: In an HA cluster, write operations are automatically synced from the primary node to the standby node; there is no need to call repeatedly on the standby node.

7.4 Log Query

API/Command	Description
-------------	-------------

GET /api/alb/console/external/v1/alb/logger/name	List of log files
GET /api/alb/console/external/v1/alb/logger/text?logger_name=...	Read the specified log content
alb-cli log list / alb-cli log show / alb-cli log tail	CLI entry

8 Typical Integration Scenarios

8.1 Scenario 1: CI/CD Automatic Release of HTTP Sites

In the continuous integration pipeline, publish build artifacts to ALB through alb-cli:

```
# 1. Prepare site JSON
cat > /tmp/site.json <<EOF
{
  "name": "web-${BUILD_ID}",
  "domains": "web.example.com",
  "listen_port": 80,
  "locations": "
[[{\"path\":\"/\", \"type\":\"proxy\", \"proxy_pass\":\"http://${APP_HOST}
  "auto_reload": true,
  "status": "enabled"
}
EOF

# 2. Executed by operations staff in the terminal (to avoid
accidental triggers in CI)
export ALB_CONSOLE_URL=https://alb-master:8887
export ALB_API_KEY=${ALB_API_KEY}
alb-cli site create --type http --file /tmp/site.json
alb-cli config reload
```

8.2 Scenario 2: Integration with ERP / Business Systems

Through the External API, automatically create a reverse proxy in ALB when entering a "business system" in the ERP:

```
import requests

API_KEY = "alb-erp-2024-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
BASE = "http://alb-master:8887/api/alb/console/external/v1"
```

```
def create_proxy(name, domain, port, upstream):
    return requests.post(
        f"{BASE}/sites/http",
        headers={"X-API-Key": API_KEY, "Content-Type":
"application/json"},
        json={
            "name": name,
            "domains": domain,
            "listen_port": port,
            "locations":
f'[{{"path":"/", "type":"proxy", "proxy_pass":"{upstream}"}}]',
            "auto_reload": True,
            "status": "enabled",
        },
    ).json()
```

8.3 Scenario 3: Daily Inspection Bot (AI Agent)

Connect the SKILL to Claude / OpenCode to let the AI perform daily inspection:

User: Help me check the current ALB status and view the last 50 lines of error.log

AI execution:

1. alb-cli status
2. alb-cli log list
3. alb-cli log tail error.log -n 50

AI output (combined with JSON parsing): The ALB primary node is currently running normally, listening on 3 ports, and no ERROR level logs are found in the last 50 lines of error.log.

8.4 Scenario 4: Automatic Fault Localization

User: I cannot access the backend at 192.168.1.100:8080, please help me check

AI execution:

1. alb-cli site list --type http (Find the corresponding site ID)
2. alb-cli site get <id> --type http
3. alb-cli log tail error.log -n 100
4. alb-cli metrics nodes

AI output: The backend 192.168.1.100:8080 of the upstream for site test-api failed the health check,
please check whether the backend service is alive.

9 Integration Recommendations

- Do not write the API Key into the code repository; it is recommended to inject it through external means such as environment variables, K8s Secrets, or Vault.
 - Use HTTPS to access the Console in production environments, to avoid the API Key being transmitted in plaintext over the network.
 - Assign independent API Keys to different business systems for easier revocation, auditing, and quota isolation as needed.
 - Before calling write operations in an HA cluster, it is recommended to first run `alb-cli status` to confirm the primary node identity, then send the request to the primary node.
-

10 Error Code Quick Reference

10.1 External API Error Codes

code	Meaning	Suggested Handling
0	Success	-
7	General business error	Read the <code>msg</code> field and handle according to the business hint
401	Authentication failed	Check whether <code>x-API-Key</code> is correct or expired
403	External API not enabled	Enable it in the Console under System Management > Open API
10001	The current node is an HA standby node	Send the request to the primary node

10.2 Common alb-cli Errors

Error Message	Cause	Suggested Handling
<code>missing console URL</code>	<code>ALB_CONSOLE_URL</code> Or <code>~/.alb/config.yaml</code> not configured	Check environment variables or configuration file
<code>missing API key</code>	<code>ALB_API_KEY</code> not configured	Check environment variables or configuration file
<code>invalid api key</code>	API Key is incorrect or External API is not enabled	Regenerate and enable it in the Console
<code>destructive operation requires an interactive terminal</code>	Write operation executed in a non-TTY environment	Execute manually in the terminal
<code>alb status module not enabled</code>	ALB status module not enabled when calling <code>metrics traffic</code>	Enable <code>stub_status</code> or the status module in the Console

11 Best Practices and Troubleshooting

11.1 Best Practices

1. **Query before modify:** Before write operations, first run `site list` to confirm the target ID and current state.
2. **Reload after modify:** Call `config reload` after all site change operations.
3. **Write operations go through interactive confirmation:** Do not bypass alb-cli's security confirmation in CI / scripts; it is recommended that operations staff trigger them manually in the terminal.
4. **Use HTTPS:** Access the Console over HTTPS in production environments, to avoid transmitting the API Key in plaintext.
5. **Tiered API Keys:** Assign independent API Keys to different business systems, and revoke them in the Console as needed.
6. **Audit trail:** All External API calls are logged in `sys_operation_logs` , and can be queried via SQL with `operator_name LIKE 'external:%'` .
7. **HA cluster:** Send write operations to the primary node; write operations on the standby node will be rejected (code=10001).

11.2 Troubleshooting

Symptom	Possible Cause	Suggested Handling
API call returns 401	API Key is incorrect or expired	Regenerate the API Key in the Console
API call returns 403	External API not enabled	Enable it in the Console under System Management > Open API
Write operation returns 10001	The current node is an HA standby node	Send the request to the primary node
alb-cli prompts missing api key	Environment variable / configuration file not configured	Configure <code>ALB_CONSOLE_URL</code> and <code>ALB_API_KEY</code>
alb-cli prompts requires an interactive terminal	Non-TTY environment	Switch to a terminal or have operations staff execute manually
<code>metrics traffic</code> prompts that the status module is not enabled	ALB stub_status not enabled	Enable the status module in the Console

Access returns 502 after creating a site	Backend service unreachable	Check upstream node connectivity
--	-----------------------------	----------------------------------

12 Version Update Notes

Date	Manual Version	Applicable Product	Update Notes
June 2026	V2.0.6	ALB V2.0.6 Standard Edition	Brand-new release: External API, alb-cli, AI Agent SKILL

UNIFIED SERVICE HOTLINE

4008-555-800

Kingdee Apusic Cloud Computing Co., Ltd. (referred to as “Kingdee Apusic”) was founded in 2000. Its predecessor was “Kingdee Middleware Company.” It is a basic software platform provider under Kingdee Group, a national standards-setting enterprise for cloud computing, and a core software enterprise in the national IT application innovation industry. Kingdee Apusic is an enterprise undertaking the National 863 Key R&D Program and the National Major Science and Technology Project for Core Electronic Components, High-end General-purpose Chips and Fundamental Software, and is also a basic platform provider for the national key projects of “Two Networks, One Website, Four Databases, and Twelve Golden Projects.” Its products are widely used in key industries such as government, military, finance, and energy, and it has cumulatively served more than 100,000 customers.



National standard setter for cloud computing
Basic software enterprise under Kingdee Group
Core enterprise for IT application innovation
Website: www.apusic.com

