天翼云CLI工具

目录

产品介绍

产品定义

天翼云CLI工具是一款运行在终端中的交互式 / 脚本化软件客户端。通过使用天翼云CLI工具，您可以在终端中轻松管理您的天翼云资源。此外，您还可将其作为 AI 智能体的调用工具入口，通过指令对接天翼云 API，轻松实现资源全生命周期管理、批量配置调整与自动化运维。

为什么使用CLI

日常管理云资源时，相较于网页控制台、SDK、原生 OpenAPI ，天翼云 CLI 能很好平衡易用性与扩展性，是更适配多场景的选择。

网页控制台在批量运维场景下操作繁琐、效率低下；SDK 需要搭建开发环境、调试与上手成本偏高；直接调用 OpenAPI 还需自行处理鉴权签名、请求封装、异常兼容等底层逻辑。

天翼云 CLI 原生运行于终端环境，交互式调试、批量脚本运维均可一键实现，无需复杂开发成本。同时它具备独特的 AI 集成能力，可作为标准化 Tools 对接 AI 智能体框架，大模型能直接下发命令调用天翼云 API，无需额外封装接口逻辑。

CLI支持多系统调用

各系统执行命令无明显差异。

- Windows
- Linux
- Mac OS

支持CLI的产品

您可在安装CLI后，在命令行中通过以下指令，查询天翼云CLI支持的产品列表：

```
ctyun-cli --help              # ######
```

产品优势

终端轻量化运维，兼顾交互与自动化

无需打开网页控制台反复点击操作，在服务器、本地终端即可执行单条指令快速管理资源；同时支持编写脚本批量执行任务，不用搭建完整开发工程，相比 SDK、可视化控制台，无图形环境也能高效完成批量配置、资源调度。

省去底层 API 封装成本，开箱即用

对比直接调用原生 OpenAPI，工具内置鉴权、签名、请求处理逻辑，无需手动拼装接口参数、处理加密校验；不用像 SDK 那样编写大量代码、维护开发环境，简单命令就能调用全量天翼云开放 API。

原生适配 AI 智能体，实现智能云上调度

可作为标准化 Tools 接入各类大模型 Agent 框架，AI 能直接下发 CLI 指令管控云资源，无需额外封装接口适配层，轻松搭建 AI 驱动的自动化运维流程，是控制台、SDK、原生 OpenAPI 不具备的专属能力。

功能特性

终端多种云资源管控

支持在终端环境执行命令，完成云主机、VPC、存储等多种天翼云产品的创建、配置、查询、销毁操作。

您可在命令行中，通过以下指令，查询天翼云CLI支持的产品列表：

```
ctyun-cli --help              # ######
```

内置鉴权封装，简化 API 调用

无需手动处理接口签名、身份校验、请求报文组装等底层逻辑，一条命令即可调用天翼云开放 API，相比直接对接原生 OpenAPI 大幅降低使用门槛。

多格式结构化数据输出

- table 格式可视化展示资源列表，终端查看信息清晰直观；
- JSON 结构化输出便于程序读取解析，适配后续自动化处理与工具对接。

标准化 AI 智能体调用入口

可作为标准 Tools 对接各类大模型 Agent 框架，Agent 能够直接下发 CLI 命令调用天翼云 API，无需手动封装鉴权、请求逻辑，快速搭建 AI 驱动的智能云上调度流程。

应用场景

智能体自动化管控云资源

场景说明

开发、运维团队搭建基于大模型的 Agent 智能运维体系，交由 Agent 自主完成资源巡检、弹性扩缩容、闲置资源回收等云上管理动作，实现无人值守智能调度。

场景痛点

如果让 Agent 直接对接原生 OpenAPI，需要单独开发适配层，自行封装鉴权、签名、参数校验逻辑，接口调试、异常兼容工作量大；若对接 SDK，还需维护开发运行环境，大幅拉长智能运维方案落地周期。

产品优势

天翼云 CLI 提供标准化 Tools 调用入口，Agent 可直接下发 CLI 指令调用云 API，工具原生内置鉴权、请求封装逻辑，无需额外开发适配代码，快速打通 AI 与天翼云资源管控链路。

企业批量自动化运维云上资产

场景说明

运维人员需要批量完成云主机批量创建、资源定期巡检、闲置资源批量销毁、VPC 批量配置变更等重复、大规模云上操作。

场景痛点

- 使用网页控制台：只能单点点击操作，批量处理需要重复人工操作，耗时且易出错；
- 使用 SDK：需要搭建完整开发环境、编写大量业务代码，代码维护、版本调试成本高；
- 直接调用 OpenAPI：需手动处理签名加密、鉴权、报文组装，底层逻辑重复开发，调试繁琐。

产品优势

对比 SDK 与原生 OpenAPI，天翼云 CLI 无需搭建开发环境、不用编写大量业务代码，仅通过简易脚本即可实现批量操作；工具内置 API 鉴权、请求封装能力，省去底层接口适配工作，同时支持 table/JSON 格式输出，批量数据查看、二次处理更便捷，大幅降低批量自动化运维的上手与落地成本。

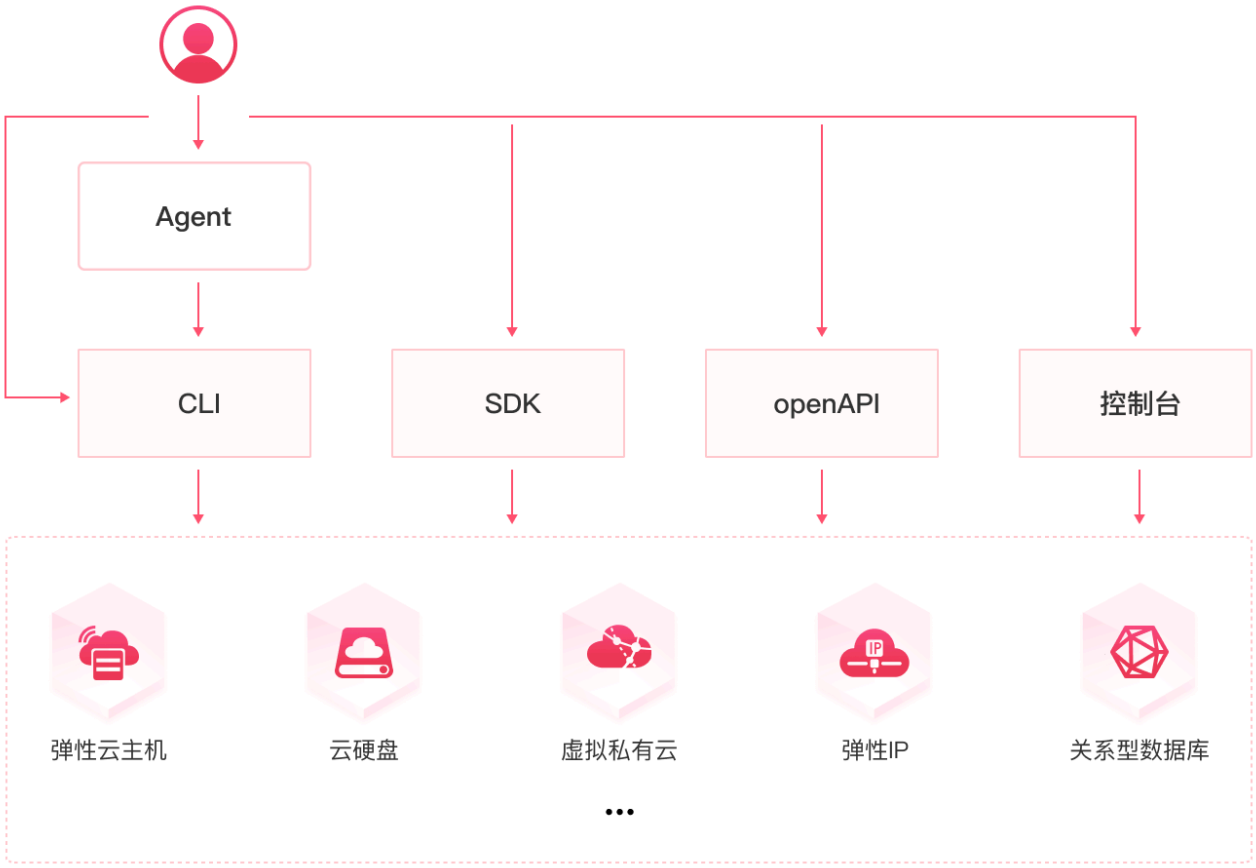
使用限制

当前发布的 CLI 工具为公开预览版（Public Preview），不承诺向后兼容，后续版本可能发生不兼容的命令或参数变更。

预览版不提供正式版的服务等级协议（SLA）保障，建议在日常研发和测试中使用。若在生产环境使用，则后续升级版本时必须做好充分验证。

与其他服务关系

天翼云CLI工具是云资源的统一上层管控工具，本身不产生业务算力与服务能力，仅作用于管控调度，服务于全品类天翼云业务云产品，是天翼云原生配套管控工具。和OpenAPI、开发者SDK、各类IaaS/PaaS/AI云产品互为互补、协同共生，共同构成系统化、代码化、智能化三位一体云上管控体系。



与天翼云各类云产品的关系

CLI工具当前已适配天翼云27款云产品，覆盖IaaS、PaaS、AI服务，具体可在安装CLI后，在命令行中通过以下指令，查询天翼云CLI支持的产品列表：

```
ctyun-cli --help              # ######
```

与天翼云AI智能体产品

CLI原生对接天翼云自研TeleAgent、OpenClaw全系AI智能体产品，是平台标准化调用工具。智能体无需改造接口、无需开发适配层，可直接下发CLI指令调度云资源，打通大模型与底层云产品的联动链路，实现AI自主巡检、智能扩容、故障自愈，是天翼云智能化运维生态的核心枢纽工具。

与天翼云OpenAPI的关系

OpenAPI是天翼云对外开放的原生接口底层，是所有资源操作的能力源头；CLI深度封装OpenAPI能力，内置账号鉴权、接口签名、参数适配、异常处理全套底层逻辑。用户无需对接原生接口、拼装请求报文，输入简易命令即可调用OpenAPI能力，简化原生API繁琐的开发调试流程。

与天翼云SDK的关系

SDK面向深度二次开发，需要搭建专属开发环境、编写业务代码、适配版本依赖，适合定制化深度开发；CLI无需编程环境、无需编译代码，以指令、脚本替代代码开发，轻量化完成资源调度，适合运维日常调度、简易自动化、智能体对接场景，是低代码、零代码替代SDK的便捷管控方式。

计费说明

计费说明

天翼云CLI为天翼云官方免费运维管控工具，工具本身永久免费，无下载安装费、命令调用费，用户可免费下载、安装、使用、接入天翼云AI智能体。

用户通过CLI指令创建、调用、使用的天翼云产品，实际产生的费用为具体云产品费用，收费标准与各云产品官方定价保持一致，CLI仅作为调用管控入口，不改变原有云产品计费规则。

如需了解具体云产品的详细计费信息，请参考天翼云官方产品文档或联系客服咨询。

快速入门

用户指南

安装CLI工具

概述

安装天翼云 CLI

本文面向第一次使用天翼云 CLI 的用户，介绍两种安装方式：

1. 一键部署：推荐方式，脚本自动识别操作系统和 CPU 架构。2. 下载 setup 包自行安装：适合不能直接执行远程安装脚本，或需要离线分发的场景。

安装完成后，用户调用的命令统一为：

```
ctyun-cli
```

Windows 下实际文件名是 `ctyun-cli.exe`，但在 PowerShell 中仍然执行 `ctyun-cli`。

1. 安装前准备

1.1 支持的平台

操作系统	x86_64 / amd64	ARM64
Windows	ctyun-cli-windows-amd64-setup.zip	ctyun-cli-windows-arm64-setup.zip
Linux	ctyun-cli-linux-amd64-setup.tar.gz	ctyun-cli-linux-arm64-setup.tar.gz
macOS	ctyun-cli-darwin-amd64-setup.tar.gz	ctyun-cli-darwin-arm64-setup.tar.gz

macOS / Linux 可以通过下面的命令查看系统和架构：

```
uname -s
uname -m
```

常见结果：

- `Darwin` 表示 macOS，`Linux` 表示 Linux。
- `x86_64` 或 `amd64` 表示 x86 64 位。
- `arm64` 或 `aarch64` 表示 ARM64。

Windows PowerShell 可以执行：

```
$env:PROCESSOR_ARCHITECTURE
```

返回 `AMD64` 或 `ARM64` 均可安装。

1.2 网络和命令要求

- macOS / Linux 需要能够使用 `curl`。
- Windows 建议使用 PowerShell 5.1 或更高版本。
- 当前用户需要能够写入自己的用户目录。
- 安装到默认目录不需要 sudo 或管理员权限。

> 说明
>
> 建议优先使用一键安装。安装脚本会自动选择正确的二进制文件、校验 SHA256，并配置用户 PATH。

2. 选择平台安装

根据你的操作系统，选择对应的安装指南：

- Windows 安装
- Linux 安装
- macOS 安装

3. 验证安装结果

无论使用哪种操作系统，安装后都应执行：

```
ctyun-cli version
ctyun-cli --help
```

看到版本号和产品命令列表，表示二进制文件已经能够正常运行。

也可以直接使用完整路径验证，用来区分"文件没有安装"和"PATH 没有生效"：

macOS / Linux：

```
"$HOME/.ctyun-cli/bin/ctyun-cli" version
```

Windows PowerShell：

```
& "$HOME\.ctyun-cli\bin\ctyun-cli.exe" version
```

验证通过后，请继续阅读 配置 CLI 完成 AK/SK 认证配置， 然后参考命令格式与参数执行云资源操作。

Windows

本文介绍如何在 Windows 上安装天翼云 CLI。用户侧只推荐两种方式：

1. 一键部署：脚本自动识别 amd64 或 arm64 架构并安装。
2. 下载 setup 包自行安装：先确认架构，再下载匹配的 zip 包。

安装完成后，在 PowerShell 中执行：

```
ctyun-cli
```

1. 一键安装

打开 PowerShell，安装最新版本天翼云CLI：

```
[Net.ServicePointManager]::SecurityProtocol = [Net.Security
ProtocolType]::Tls12

Invoke-WebRequest -UseBasicParsing "https://huabei-2.zos.ctyun.cn/ctyun-cli/
install-cli.ps1" -OutFile install-cli.ps1

powershell -NoProfile -ExecutionPolicy Bypass -File .\install-cli.ps1 -BaseUrl
 "https://huabei-2.zos.ctyun.cn/ctyun-cli"
```

下载指定版本天翼云CLI：

```
[Net.ServicePointManager]::SecurityProtocol = [Net.Security
ProtocolType]::Tls12

Invoke-WebRequest -UseBasicParsing "https://huabei-2.zos.ctyun.cn/ctyun-cli/
install-cli.ps1" -OutFile install-cli.ps1

powershell -NoProfile -ExecutionPolicy Bypass -File .\install-cli.ps1 -BaseUrl
 "https://huabei-2.zos.ctyun.cn/ctyun-cli" -Version v0.1.0
```

执行后，CLI会安装到以下目录，并将该目录写入当前用户的 PATH：

```
%USERPROFILE%\.ctyun-cli\bin\ctyun-cli.exe
```

如果 PowerShell 提示不允许执行脚本，可以只为当前窗口临时放开限制：

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

然后重新执行安装命令。该设置只影响当前 PowerShell 窗口。

2. 手动安装

确认系统和架构

```
$env:PROCESSOR_ARCHITECTURE
```

架构对应关系：

输出	包名字段	安装包
AMD64 / X64	amd64	ctyun-cli-windows-amd64-setup.zip
ARM64	arm64	ctyun-cli-windows-arm64-setup.zip

解压后，执行压缩包内的脚本：

```
powershell -NoProfile -ExecutionPolicy Bypass -File .\install-cli.ps1
```

默认会安装到：

```
%USERPROFILE%\.ctyun-cli\bin\ctyun-cli.exe
```

3. 验证安装结果

安装完成后执行：

```
ctyun-cli version
ctyun-cli --help
```

如果当前 PowerShell 还找不到命令，先直接验证文件：

```
& "$HOME\.ctyun-cli\bin\ctyun-cli.exe" version
```

如果可以运行，关闭并重新打开 PowerShell，或在当前窗口临时加入 PATH：

```
$env:Path = "$HOME\.ctyun-cli\bin;$env:Path"
ctyun-cli version
```

Linux

本文介绍如何在 Linux 上安装天翼云 CLI。用户侧只推荐两种方式：

1. 一键部署：脚本自动识别 CPU 架构并安装。
2. 下载 setup 包自行安装：先确认架构，再下载匹配的包。

安装完成后，命令统一为：

```
ctyun-cli
```

1. 一键安装

安装最新版本天翼云CLI

```
curl -fsSL "https://huabei-2.zos.ctyun.cn/ctyun-cli/install-cli.sh" | sh -s --
 --base-url "https://huabei-2.zos.ctyun.cn/ctyun-cli"
```

安装指定版本天翼云CLI

```
curl -fsSL "https://huabei-2.zos.ctyun.cn/ctyun-cli/install-cli.sh" | sh -s --
 --base-url "https://huabei-2.zos.ctyun.cn/ctyun-cli" --version v0.1.0
```

默认安装路径：

```
~/.ctyun-cli/bin/ctyun-cli
```

2.手动安装

先确认架构后，再选择对应包。

架构	latest setup 包
amd64	ctyun-cli-linux-amd64-setup.tar.gz
arm64	ctyun-cli-linux-arm64-setup.tar.gz

解压并安装：

```
mkdir -p ctyun-cli-setup
tar -xzf "$PACKAGE" -C ctyun-cli-setup
```

```
cd ctyun-cli-setup
sh install-cli.sh
```

默认安装路径：

```
~/.ctyun-cli/bin/ctyun-cli
```

3.验证安装结果

安装完成后执行：

```
ctyun-cli version
ctyun-cli --help
```

如果当前终端还找不到命令，先直接检查文件是否存在：

```
ls -l "$HOME/.ctyun-cli/bin/ctyun-cli"
"$HOME/.ctyun-cli/bin/ctyun-cli" version
```

如果文件存在，说明只是当前终端的 PATH 尚未生效：

```
export PATH="$HOME/.ctyun-cli/bin:$PATH"
ctyun-cli version
```

也可以重新登录终端，或按安装脚本提示执行 `. ~/.bashrc`、`. ~/.zshrc`、`. ~/.profile`。

macOS

本文介绍如何在 macOS 上安装天翼云 CLI。用户侧只推荐两种方式：

1. 一键部署：脚本自动识别 Intel 或 Apple Silicon 架构并安装。
2. 下载 setup 包自行安装：先确认架构，再下载匹配的包。

安装完成后，命令统一为：

```
ctyun-cli
```

1. 一键安装

安装最新版本天翼云CLI：

```
curl -fsSL "https://huabei-2.zos.ctyun.cn/ctyun-cli/install-cli.sh" | sh -s --
 --base-url "https://huabei-2.zos.ctyun.cn/ctyun-cli"
```

安装指定版本天翼云CLI：

```
curl -fsSL "https://huabei-2.zos.ctyun.cn/ctyun-cli/install-cli.sh" | sh -s --
 --base-url "https://huabei-2.zos.ctyun.cn/ctyun-cli" --version v0.1.0
```

2. 手动安装

确认架构后，再选择对应包：

架构	latest setup 包
amd64	ctyun-cli-darwin-amd64-setup.tar.gz

架构	latest setup 包
arm64	ctyun-cli-darwin-arm64-setup.tar.gz

解压并安装：

```
mkdir -p ctyun-cli-setup
tar -xzf "$PACKAGE" -C ctyun-cli-setup
cd ctyun-cli-setup
sh install-cli.sh
```

默认安装路径：

```
~/.ctyun-cli/bin/ctyun-cli
```

3.验证安装结果

安装完成后执行：

```
ctyun-cli version
ctyun-cli --help
```

如果当前终端还找不到命令，先直接检查文件是否存在：

```
ls -l "$HOME/.ctyun-cli/bin/ctyun-cli"
"$HOME/.ctyun-cli/bin/ctyun-cli" version
```

如果文件存在，说明只是当前终端的 PATH 尚未生效：

```
export PATH="$HOME/.ctyun-cli/bin:$PATH"
ctyun-cli version
```

macOS 默认使用 zsh。也可以重新打开终端，或按安装脚本提示执行：

```
. ~/.zshrc
```

配置CLI工具

天翼云 CLI 的配置项包括访问凭证、输出格式和日志行为。所有配置项均支持三种配置方式，优先级从高到低为：

命令行参数 > 配置文件 > 环境变量

交互式配置

运行 `configure` 命令，按提示逐项输入：

```
ctyun-cli configure
```

交互过程：

```
[INFO] #######CLI...
Access Key [dd2b****786d3]:           # ## AccessKey##########
```

```
Secret Key [7cf0****120e]:            # ## SecretKey########
#### (json/table) [json]:
#### (true/false) [false]:
###### []:
#### (text/json) [text]:
[OK] ###### ~/.ctyun-cli.yaml
```

- AccessKey / SecretKey 输入时关闭终端回显，防止敏感信息泄露
- 已有配置值时，AccessKey / SecretKey 以掩码形式显示（如 `dd2b****786d3`），其他字段明文显示
- 直接按回车可保留当前值

配置文件

配置文件默认路径：

- Linux / macOS：`~/.ctyun-cli.yaml`
- Windows：`%USERPROFILE%\.ctyun-cli.yaml`

也可通过 `--config` / `-c` 指定其他路径：

```
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 --config /
path/to/config.yaml
```

手动创建或编辑配置文件：

```
# ~/.ctyun-cli.yaml
access_key: "your-access-key"    # ### AccessKey
secret_key: "your-secret-key"    # ### SecretKey
output: "json"                   # #### (json/table)
log: false                       # ######
log_file: ""                     # ############# stderr
log_format: "text"               # #### (text/json)
```

访问凭证

天翼云 CLI 通过 AccessKey / SecretKey 对调用方进行身份认证。调用任何 API 操作前，必须先完成认证信息的配置。AccessKey / SecretKey的获取 点击此处

参数	说明	环境变量	命令行参数
AccessKey	天翼云 AccessKey，标识用户身份	`CTYUN_AK`	`--access-key / -a`
SecretKey	天翼云 SecretKey，用于请求加密	`CTYUN_SK`	`--secret-key / -s`

环境变量

适合 CI/CD 等自动化场景：

```
# Linux / macOS
export CTYUN_AK="your-access-key"
export CTYUN_SK="your-secret-key"

# Windows PowerShell
```

```
$env:CTYUN_AK="your-access-key"
$env:CTYUN_SK="your-secret-key"
```

也可在单次命令中临时指定：

```
CTYUN_AK="your-access-key" CTYUN_SK="your-secret-key" ctyun-cli vpc ListVpc --
regionID bb9fdb42056f11eda1610242ac110002
```

命令行参数

通过 `--access-key` / `-a` 和 `--secret-key` / `-s` 直接传入：

```
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 -a your-
access-key -s your-secret-key
```

警告

命令行参数会出现在进程列表和 Shell 历史记录中，存在泄露风险，建议仅在临时调试时使用。

安全建议

- 不要将 AccessKey / SecretKey 硬编码在脚本中，优先使用环境变量或配置文件
- 不要长期将 AccessKey / SecretKey 写在 Shell 命令中，Shell 历史记录会保存明文密钥
- 配置文件建议设置文件权限为仅当前用户可读
- 生产环境建议使用环境变量，避免密钥落盘
- 命令行参数方式会将密钥暴露在进程列表中，仅用于临时调试
- CLI 日志不会记录 AccessKey 和 SecretKey

输出格式

通过 `--output` / `-o` 控制返回结果的展示格式，支持 `json`（默认）和 `table`：

```
# JSON ######
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002

# Table ##
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 -o table
```

通过 `--cli-query` 可使用 JMESPath 表达式对返回结果进行过滤。

埋点与日志

默认不输出日志。开启 `--log` 后，CLI 会记录命令调用和 HTTP 请求的结构化日志：

```
# ##### stderr
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 --log true

# #########/##
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 --log true
 --log-file ./ctyun-cli.log
```

日志参数说明

参数	说明	默认值
`--log`	是否开启日志	关闭
`--log-file`	日志文件路径；为空时输出到 stderr	空（stderr）
`--log-format`	日志格式：`text` / `json`	`text`

环境变量

也可通过环境变量控制日志行为：

环境变量	对应参数
`CTYUN_LOG`	`--log`
`CTYUN_LOG_FILE`	`--log-file`
`CTYUN_LOG_FORMAT`	`--log-format`

警告

安全提示：日志不会记录 AccessKey、SecretKey 或签名 Header。

优先级规则

当多种方式同时配置时，CLI 按以下优先级取值：

```
##### > #### > ####
```

示例：若配置文件中设置了 `access_key: "key-from-file"`，同时环境变量 `CTYUN_AK="key-from-env"`，则 CLI 使用配置文件中的值；若再通过 `--access-key key-from-cli` 传入，则使用命令行参数的值。

全局参数一览

以下参数可附加在任何命令之后：

```
  -c, --config string       ########### ~/.ctyun-cli.yaml#
  -o, --output string       #### (json|table### json)
      --cli-query string    JMESPath ########## API ####
  -a, --access-key string   ### AccessKey
  -s, --secret-key string   ### SecretKey
      --log                 ####
      --log-file string     ############ stderr
      --log-format string   #### (text|json### text)
```

使用CLI工具

内置命令

configure — 交互式配置

首次使用前，通过交互式命令完成认证配置：

```
ctyun-cli configure
```

按提示输入 AccessKey、SecretKey 等信息，配置保存到 `~/.ctyun-cli.yaml`。

version — 查看版本

```
ctyun-cli version
```

输出当前 CLI 的版本号。

completion — 生成 Shell 自动补全脚本

启用自动补全后，在终端输入命令时按 Tab 键即可自动补全产品名、Action 名和参数名，无需记忆完整拼写。例如输入 `ctyun-cli vpc L` 后按 Tab，可自动补全为 `ctyun-cli vpc ListVpc`。

```
ctyun-cli completion <shell>
```

支持的 shell：`bash`、`zsh`、`fish`、`powershell`。

各 shell 的启用方式：

Shell	持久化	当前会话
Bash	`ctyun-cli completion bash > /etc/bash_completion.d/ctyun-cli`	`source <(ctyun-cli completion bash)`
Zsh	`ctyun-cli completion zsh > "${fpath[1]}/_ctyun-cli"`	`source <(ctyun-cli completion zsh)`
Fish	`ctyun-cli completion fish > ~/.config/fish/completions/ctyun-cli.fish`	—
PowerShell	`ctyun-cli completion powershell >> $PROFILE`	`ctyun-cli completion powershell | Out-String | Invoke-Expression`

help — 查看帮助

```
ctyun-cli --help              # ######
ctyun-cli vpc --help          # #########
```

```
ctyun-cli vpc ListVpc --help  # #########
```

帮助信息包含所有可用参数（必填参数标注 (##)）、复杂类型参数的子字段结构及响应示例。

调用OpenAPI

基本格式

```
ctyun-cli <product> <action> [flags]
```

部分	说明	示例
ctyun-cli	CLI 可执行文件名	—
<product>	产品标识	vpc、ecs、evs
<action>	操作名称	ListVpc、CreateVpc
[flags]	操作参数和全局参数	--regionID xxx、-o table

警告

注意：产品名、Action 名和参数名区分大小写，必须与 --help 输出中的名称完全一致。

操作参数

每个操作的参数通过 --help 查看，使用 --### # 的方式传入：

```
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 --pageNumber
 1 --pageSize 10
ctyun-cli vpc CreateVpc --clientToken "3dbb3c17-cbdc-465d-9e7b-
64c936a211d3" --name "my-vpc" --CIDR "172.31.0.0/16" --regionID bb9fdb42
056f11eda1610242ac110002
```

–body 参数

所有操作都支持 --body / -b 参数，直接传入 JSON 请求体。--body 中的字段优先级最高，会覆盖同名参数的值。

```
ctyun-cli vpc CreateVpc --body '{"clientToken":"3dbb3c17-cbdc-465d-9e7b-
64c936a211d3","name":"my-vpc","CIDR":"172.31.0.0/16","regionID":"bb9fdb42
056f11eda1610242ac110002"}'
```

适用场景：

- 参数较多时，直接传入 JSON 更方便
- 需要覆盖某个参数的值时

说明

建议：使用 --body 前先通过 --help 查看该操作的必填参数，确保 JSON 中包含所有必填字段。

复杂类型参数

Object、Array 等复杂类型参数需要传入 JSON 字符串。复杂类型参数的子字段结构会在 `--help` 的 Complex Parameter 部分展示，包含每个子字段的名称、类型、是否必填和说明。

```
# Object ## — ## JSON ##
ctyun-cli <product> <action> --<objectParam>
 '{"key1":"value1","key2":"value2"}'

# Array ## — ## JSON ##
ctyun-cli <product> <action> --<arrayParam> '["value1","value2"]'
```

警告

注意：通过PowerShell使用时，JSON字符串中的引号需要正确转义。

必填参数

必填参数在 `--help` 中标注为 `(##)`。如果必填参数未通过 flag 或 `--body` 提供，CLI 会报错提示缺失的参数名。

```
# #############
##: #######: regionID#### flag # --body ###
```

查看帮助信息

CLI 支持的产品和操作在安装时已内置，可通过 `--help` 逐级浏览：

```
# ######
ctyun-cli --help

# ############
ctyun-cli vpc --help

# ##########
ctyun-cli vpc ListVpc --help
```

帮助信息包含所有可用参数（必填参数标注 `(##)`）、复杂类型参数的子字段结构及响应示例。

输出与过滤

输出格式

通过 `--output` / `-o` 全局参数控制返回结果的展示格式：

JSON 格式（默认）

```
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002
```

输出示例：

```
{
  "returnObj": {
    "vpcs": [
      {
        "vpcID": "vpc-xxx",
```

```
        "name": "my-vpc",
        "CIDR": "172.31.0.0/16"
      }
    ]
  },
  "statusCode": 800
}
```

Table 格式

以表格形式展示，建议配合JMESPath过滤使用：

```
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 -o table
```

JMESPath 过滤

通过 `--cli-query` 参数使用 JMESPath 表达式对返回结果进行筛选，只输出感兴趣的字段。

提取指定字段后以表格展示

```
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 --cli-query
 'returnObj.vpcs[].{id:vpcID,name:name,cidr:CIDR}' -o tableshu
```

输出示例：

```
cidr            id              name
----            --              ----
192.168.0.0/16  vpc-en0h72cf8y  test
10.0.0.0/8      vpc-w9snphifbt  default_network
192.168.0.0/16  vpc-5c9t641lw7  tf-vpc-gaokeyong
192.168.0.0/16  vpc-dg0cwnoeou  tf-vpc-for-dns-6
192.168.4.0/24  vpc-3uyset3jjs  vpc-test-route-2
192.168.0.0/16  vpc-djkto43khs  tf-vpc-for-redis
192.168.0.0/16  vpc-211azln26d  tf-vpc-for-ebm
192.168.0.0/16  vpc-4qievg3tu9  tf-vpc-for-dns-6
192.168.0.0/16  vpc-eeg90ay37v  tf-vpc-for-kafka
192.168.0.0/16  vpc-sce5od49s5  tf-vpc-for-ec
```

提取单字段列表

```
ctyun-cli vpc ListVpc --regionID bb9fdb42056f11eda1610242ac110002 --cli-query
 'returnObj.vpcs[].vpcID'
```

输出示例：

```
[
 "vpc-en0h72cf8y",
 "vpc-w9snphifbt",
 "vpc-5c9t641lw7",
 "vpc-dg0cwnoeou",
 "vpc-3uyset3jjs",
 "vpc-djkto43khs",
 "vpc-211azln26d",
 "vpc-4qievg3tu9",
 "vpc-eeg90ay37v",
 "vpc-sce5od49s5"
]
```

常用 JMESPath 语法

表达式	说明	示例
`key`	取顶层字段	`statusCode`
`a.b.c`	嵌套取值	`returnObj.vpcs`
`[*].field`	提取数组中所有元素的字段	`returnObj.vpcs[*].vpcID`
`[?field=='value']`	条件过滤	`returnObj.vpcs[?name=='my-vpc']`
`[*].{k1:f1,k2:f2}`	投影（重命名字段）	`returnObj.vpcs[].{id:vpcID,name:name}`
`[0:3]`	数组切片	`returnObj.vpcs[0:3]`
`length()`	计数	`length(returnObj.vpcs)`

说明

提示：JMESPath 表达式基于 API 的原始 JSON 返回结构编写，建议先不带 `--cli-query` 执行一次命令，观察返回结构后再编写过滤表达式。

安装与使用Skill

本文介绍如何获取 `ctyun-cli-guidance` Skill，并把它接入到你正在使用的智能体中。

当前发布方式不提供 skill 安装脚本。Skill 以 zip 包发布。用户下载 zip 后，按所用智能体的要求导入即可。不同智能体的机制不完全一样，常见方式包括：

- 解压到智能体约定的 skills、knowledge、context 或扩展目录。
- 在智能体的会话、项目知识库、系统提示词或自定义指令中上传或引用 SKILL.md。
- 通过智能体自带的插件、扩展、MCP、知识库管理功能导入。

1.Skill 能做什么

ctyun-cli-guidance 会指导智能体：

- 安装当前平台对应的天翼云 CLI。
- 根据用户需求选择正确的天翼云 CLI 产品和 Action。
- 通过 ctyun-cli --help 核对真实命令和参数，避免用错命令名。
- 查询资源池、可用区、规格、镜像、VPC、子网、安全组、EIP 等资源。
- 帮用户创建、删除、退订、绑定、解绑资源等操作，变更操作前说明影响并请求确认。
- 协助排查 AK/SK、权限、参数、签名、区域和资源 ID 等问题。

Skill 是给智能体使用的操作指南，不是天翼云 CLI 本身：

- 只安装 Skill 后，智能体可以解释命令并协助规划操作。
- 需要查询或变更真实云资源时，智能体会触发安装 ctyun-cli，并引导配置有效的 AK/SK。

- Skill 不会内置、上传或代替用户的 AK/SK。

2.通过智能体辅助安装skill

如果你正在使用的智能体具备文件下载、解压和写入本地目录的能力，推荐把 Skill 下载地址交给智能体，让它辅助完成下载和导入。这样通常比手动复制多段命令更省事。

可以直接对智能体说：

```
######## CLI Skill#
Skill ######https://huabei-2.zos.ctyun.cn/ctyun-cli/skill/latest/ctyun-cli-
guidance.zip
############### zip##### ctyun-cli-guidance/SKILL.md ########## skill#
knowledge#context ##########
###################################
```

如果要安装指定版本，可以把地址换成：

```
https://huabei-2.zos.ctyun.cn/ctyun-cli/skill/0.1.0/ctyun-cli-guidance-0.1.0.
zip
```

注意：不同智能体支持的导入机制不同。有的支持本地 skills 目录，有的支持项目知识库、会话上下文、自定义指令或插件配置。让智能体安装时，应要求它先确认当前智能体实际读取的位置，不要盲目使用某个固定目录。

3. 手动下载 Skill zip

macOS / Linux 下载最新版：

```
curl -fL "https://huabei-2.zos.ctyun.cn/ctyun-cli/skill/latest/ctyun-cli-
guidance.zip" -o ctyun-cli-guidance.zip
```

macOS / Linux 下载指定版本：

```
curl -fL "https://huabei-2.zos.ctyun.cn/ctyun-cli/skill/0.1.0/ctyun-cli-
guidance-0.1.0.zip" -o ctyun-cli-guidance.zip
```

Windows PowerShell 下载最新版：

```
[Net.ServicePointManager]::SecurityProtocol = [Net.Security
ProtocolType]::Tls12
Invoke-WebRequest -UseBasicParsing "https://huabei-2.zos.ctyun.cn/ctyun-cli/
skill/latest/ctyun-cli-guidance.zip" -OutFile ctyun-cli-guidance.zip
```

Windows PowerShell 下载指定版本：

```
[Net.ServicePointManager]::SecurityProtocol = [Net.Security
ProtocolType]::Tls12
Invoke-WebRequest -UseBasicParsing "https://huabei-2.zos.ctyun.cn/ctyun-cli/
skill/0.1.0/ctyun-cli-guidance-0.1.0.zip" -OutFile ctyun-cli-guidance.zip
```

zip 包内部结构是：

```
ctyun-cli-guidance/SKILL.md
ctyun-cli-guidance/references/*.md
```

4. 手动导入到智能体

优先查看你所使用智能体的官方说明，确认它支持哪种导入方式。

如果智能体支持本地 skill 目录，请把 zip 解压到该目录，让最终路径类似：

macOS / Linux路径为：

```
<###-skill##>/ctyun-cli-guidance/SKILL.md
<###-skill##>/ctyun-cli-guidance/references/*.md
```

Windows 路径为：

```
<###-skill##>\ctyun-cli-guidance\SKILL.md
<###-skill##>\ctyun-cli-guidance\references\*.md
```

不要只把 `SKILL.md` 单独放进目录根部，因为目录名 `ctyun-cli-guidance` 和 `references` 目录都是识别、组织和按场景加载规则的一部分。

如果智能体不支持本地 skill 目录，但支持会话级或项目级知识配置，优先导入整个 `ctyun-cli-guidance` 目录；如果只能导入单文件，可以先导入 `ctyun-cli-guidance/SKILL.md`，但部分场景化参考规则可能无法自动读取。导入后建议在新会话中使用。

5. 解压示例

下面只是通用示例，不代表所有智能体都使用这些路径。实际路径以你使用的智能体文档为准。

macOS / Linux：

```
# ###### skill ### ~/.example-agent/skills
mkdir -p "$HOME/.example-agent/skills"
unzip -o ctyun-cli-guidance.zip -d "$HOME/.example-agent/skills"
```

Windows PowerShell：

```
# ###### skill ### %USERPROFILE%\.example-agent\skills
New-Item -ItemType Directory -Force "$HOME\.example-agent\skills" | Out-Null
Expand-Archive ".\ctyun-cli-guidance.zip" -DestinationPath "$HOME\.example-
agent\skills" -Force
```

Codex 示例：

```
mkdir -p "${CODEX_HOME:-$HOME/.codex}/skills"
unzip -o ctyun-cli-guidance.zip -d "${CODEX_HOME:-$HOME/.codex}/skills"
```

Claude Code 示例：

```
mkdir -p "$HOME/.claude/skills"
unzip -o ctyun-cli-guidance.zip -d "$HOME/.claude/skills"
```

如果你的智能体不使用上述目录，请不要照抄路径，按该智能体文档选择正确目录或会话配置方式。

6. 确认导入成功

如果是本地目录方式，检查目标目录中是否存在：

```
ctyun-cli-guidance/SKILL.md
ctyun-cli-guidance/references/
```

macOS / Linux 示例：

```
test -f "$HOME/.example-agent/skills/ctyun-cli-guidance/SKILL.md" \
  && test -d "$HOME/.example-agent/skills/ctyun-cli-guidance/references" \
  && echo "skill installed"
```

Windows PowerShell 示例：

```
Test-Path "$HOME\.example-agent\skills\ctyun-cli-guidance\SKILL.md"
Test-Path "$HOME\.example-agent\skills\ctyun-cli-guidance\references"
```

如果是会话或项目知识方式，检查智能体配置中是否能看到 ctyun-cli-guidance 目录，或至少能看到 `SKILL.md` 和 `references` 中的参考内容。

导入或更新 `Skill` 后，建议重新启动智能体客户端，或新建一个会话。已经打开的会话可能不会自动重新扫描 Skill 文件。

可以用下面的问题做验证：

```
#### CTYUN CLI ## VPC ####### product # Action#
######## Action #############
```

预期回答应包含：

```
product: vpc
Action: ListVpc
ctyun-cli vpc ListVpc --help
```

常见问题

安装CLI工具

安装出现 `404`

404 表示安装脚本、setup 包或指定版本在发布地址中不存在。请检查发布根地址、版本号和平台架构是否正确。

安装出现 `checksum mismatch`

这通常是发布侧的 setup 包和 checksums.txt 不匹配。不要跳过校验继续使用，请保留报错信息并联系客户支持、发布负责人或 天翼云CLI 研发处理。

`cannot execute binary file: Exec format error`

这通常表示下载的 setup 包与当前操作系统或 CPU 架构不匹配。请重新执行一键部署，或确认平台后重新下载匹配的 setup 包。

windows系统安装

禁止运行脚本

文档中的命令使用了 `-ExecutionPolicy Bypass`，只对这一次 powershell 子进程生效，不会永久修改系统策略。

如果仍希望在当前 PowerShell 窗口里直接运行 .\install-cli.ps1，可以只为当前窗口临时放开限制：

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

提示无法识别 `ctyun-cli`

通常是 PATH 还没有在当前 PowerShell 窗口生效。先用完整路径验证，再临时加入 PATH：

```
& "$HOME\.ctyun-cli\bin\ctyun-cli.exe" version
$env:Path = "$HOME\.ctyun-cli\bin;$env:Path"
```

在windows powershell中 使用天翼云CLI出现乱码

可以设置当前会话为utf8编码：

```
$OutputEncoding = [System.Text.UTF8Encoding]::new()
[Console]::InputEncoding = [System.Text.UTF8Encoding]::new()
[Console]::OutputEncoding = [System.Text.UTF8Encoding]::new()
chcp 65001
```

如果结合智能体使用的话，可以尝试与智能体说：

```
####powershell######
```

checksum mismatch

这通常是发布侧的 setup 包和 checksums.txt 不匹配。不要跳过校验继续使用，请保留报错信息并联系客户支持、发布负责人或 CLI 研发处理。

下载后提示架构不匹配

请按第 1 节重新确认架构，然后下载 `windows-amd64` 或 `windows-arm64` 对应的 setup 包。

Linux系统安装

`ctyun-cli: command not found`

通常是 PATH 还没有在当前终端生效。先用完整路径验证，再临时加入 PATH：

```
"$HOME/.ctyun-cli/bin/ctyun-cli" version
export PATH="$HOME/.ctyun-cli/bin:$PATH"
```

提示 `cannot execute binary file: Exec format error`

该错误通常表示 setup 包与当前 CPU 架构不匹配。请重新执行一键部署，或确认架构后重新下载匹配的 setup 包。

`checksum mismatch`

这通常是发布侧的 setup 包和 checksums.txt 不匹配。不要跳过校验继续使用，请保留报错信息并联系客户支持、发布负责人或 CLI 研发处理。

macOS系统安装

`ctyun-cli: command not found`

通常是 PATH 还没有在当前终端生效。先用完整路径验证，再临时加入 PATH：

```
"$HOME/.ctyun-cli/bin/ctyun-cli" version
export PATH="$HOME/.ctyun-cli/bin:$PATH"
```

提示 `cannot execute binary file: Exec format error`

该错误通常表示 setup 包与当前 CPU 架构不匹配。请重新执行一键部署，或确认架构后重新下载匹配的 setup 包。

`checksum mismatch`

这通常是发布侧的 setup 包和 checksums.txt 不匹配。不要跳过校验继续使用，请保留报错信息并联系客户支持、发布负责人或 CLI 研发处理。

macOS 阻止运行

请先确认文件确实来自可信发布地址，然后在“系统设置 -> 隐私与安全性”中允许该程序运行。

安装Skill

导入成功，但智能体没有使用 Skill

依次检查：

1. Skill 是否导入到了当前智能体实际读取的位置。
2. 文件名是否为 `SKILL.md`。
3. 目录名是否为 `ctyun-cli-guidance`。
4. 是否已经重新启动智能体客户端或新建会话。
5. 提问内容是否明确涉及 CTYUN、天翼云或 `ctyun-cli`。

我的智能体没有 skills 目录怎么办

使用该智能体支持的知识库、项目上下文、自定义指令或系统提示词功能，优先导入整个 ctyun-cli-guidance 目录。核心目标是让智能体在回答天翼云 CLI 问题时能读取 `SKILL.md` 和 `references` 中的场景化规则，而不是必须使用某个固定目录。

最佳实践

云资源巡检日报

适用对象

- 运维日报。
- 多项目资源盘点。
- 值班交接。
- 资源治理和容量观察。

巡检项

- ECS 总数、运行中、关机、异常
- EIP 总数、已绑定、未绑定
- 云硬盘总数、已挂载、未挂载
- ELB 后端健康状态
- NAT 网关状态
- VPC/Subnet 数量和资源分布
- CPU、内存、磁盘等监控 Top 数据

智能体自动化流程

1. 查询目标区域或全部区域2. 按区域分页查询资源3. 汇总资源数量和状态4. 查询异常状态资源5. 查询监控 Top 数据6. 识别未绑定、未挂载、长期关机等资源7. 生成 Markdown 或 table 格式日报

智能体执行CLI 过程

以华南2资源池为例：

```
# ########2###############

  Thought for 5s (ctrl+o to expand)

# Bash(ctyun-cli common GetCustomerRegionResources --regionID "200000002530"
 2>&1 | python3 -c "
      import json,sys…)
  #  === ##2 #### ===
       CERT: 1
       FIREWALL: 19
     … +10 lines (ctrl+o to expand)

# Bash(ctyun-cli common GetCustomerRegionQuotas --regionID "200000002530" 2>&1
 | python3 -c "
```

```
      import json,sys…)
  #  === #### ===
       vm_limit: 50
       vcpu_limit: 200
     … +10 lines (ctrl+o to expand)

# Bash(ctyun-cli ecs GetEcsInstanceStatistics --regionID "200000002530" 2>&1)
  #  {
       "description": "##",
       "message": "SUCCESS",
     … +14 lines (ctrl+o to expand)

  Thought for 3s (ctrl+o to expand)

# 6 Explore agents finished (ctrl+o to expand)
   # Scan ECS instances in ##2 · 2 tool uses · 13.3k tokens
   # #  Done
   # Scan VPC/subnet/ACL in ##2 · 4 tool uses · 12.4k tokens
   # #  Done
   # Scan security groups in ##2 · 2 tool uses · 13.3k tokens
   # #  Done
   # Scan EIP/NAT in ##2 · 7 tool uses · 17.4k tokens
   # #  Done
   # Scan ELB/listeners in ##2 · 31 tool uses · 29.9k tokens
   # #  Done
   # Scan EVS storage in ##2 · 8 tool uses · 27.1k tokens
     #  Done

  Thought for 2m 22s (ctrl+o to expand)
```

输出示例

华南2 资源巡检日报

日期：2026-07-02 | 区域：华南2（200000002530，cn-huanan-02）| 可用区：3（1A/2A/3A）

一、资源总览

资源分布

ECS：	11台（全运行）
EVS：	15卷（840G+1400G）
VPC：	9个
Subnet:	12个
SG：	17个（136规则）
EIP：	6个（全ACTIVE）
NAT：	2个（SNAT 7规则）
ELB：	5个（8监听器）
EIP：	6个（4绑ELB + 2绑NAT）

配额使用率

vCPU：	152/200	76%
内存：	352/400	88%
VPC：	9/10	90%
ECS：	11/50	22%
存储：	2340G/200T	1%
快照：	0/30	0%

二、计算资源（ECS）

2.1 实例清单（11/11 运行中）

实例	规格	vCPU	RAM	可用区	内网 IP	系统
	c8.8xlarge.2	32	64G	1A		CTyunOS
	c8.8xlarge.2	32	64G	1A		CTyunOS
	c8.8xlarge.2	32	64G	2A		CTyunOS
	p8a.6xlarge.4	24	96G	1A		CTyunOS V4
小计（AI 集群）		**120**	**288G**			
	s8e.2xlarge.2	8	16G	1A		CTyunOS
	s8.xlarge.2	4	8G	1A		CTyunOS
	s8.xlarge.2	4	8G	2A		CTyunOS
	s8e.xlarge.2	4	8G	1A		CTyunOS
	s8e.xlarge.2	4	8G	1A		CTyunOS
	s8e.xlarge.2	4	8G	1A		CTyunOS
	s8e.xlarge.2	4	8G	1A		Win2022
总计		**152**	**352G**			

2.2 关键指标

指标	值	评估
运行率	100% (11/11)	✅ 正常
过期/关机	0	✅ 无资源浪费
计费模式	100% 按需	⚠️ 无包周期，成本偏高
AZ 分布	1A:9 / 2A:2 / 3A:0	🟡 AZ2 仅 2 台，AZ3 为空
系统多样性	CTyunOS:9 / CTyunOS V4:1 / Win2022:1	✅ 正常
GPU 资源	1×A100 40GB (llm-test-01)	ℹ️ 使用中
密钥对	1个 (KeyPair-4193，未绑定)	⚠️ 建议所有实例绑定密钥

2.3 配额预警

资源	已用	上限	使用率
vCPU	152	200	**76%** ⚠️
内存	352 GB	400 GB	**88%** 🔴

> 🔴 ***vCPU 和内存配额接近上限。****当前剩余仅 48 vCPU / 48 GB，无法再启动一台 c8.8xlarge.2（32 vCPU / 64G 规格）的大规格实例。*

三、存储资源 (EVS)

3.1 存储总览

类别	数量	总容量
系统盘	11 卷	840 GB
数据盘	4 卷	1,400 GB
合计	**15 卷**	**2,340 GB**

5.1 EIP 清单（6 个全 ACTIVE）

EIP	IP	带宽	用途
[illegible]	[illegible]	100M cn2	ELB → Apollo 配置中心
[illegible]	[illegible]	10M cn2	ELB → **K8s API (6443)**
[illegible]	[illegible]	20M cn2	ELB → **ArangoDB (30678)**
[illegible]	[illegible]	20M cn2	ELB → IAM 代理 (2080) + starsky
[illegible]	3[illegible]	100M cn2	NAT → 4 个子网 SNAT
[illegible]	[illegible]	10M 163	NAT → 3 个子网 SNAT

5.2 NAT 网关

NAT	EIP	子网 SNAT 覆盖	IP 数
nat-cn2	[illegible]	4 个子网	4,864
nat-internet	[illegible]	3 个子网	4,608

> 🟡 *172.30.128.0/20 (4096 IP) 同时挂在两个 NAT 网关上。*

5.3 ELB 公网暴露矩阵

ELB	公网 IP	端口	后端服务	ACL	风险
[illegible]	[illegible]	TCP 6443	**K8s API**	❌	🔴 严重
[illegible]	[illegible]	TCP 30678	**ArangoDB**	❌	🔴 严重
apc[illegible]	[illegible]	TCP 31910/30571	Apollo Admin	❌	🟠 高
[illegible]	[illegible]	TCP 2080	IAM Proxy	❌	🟠 高
[illegible]	[illegible]	TCP 30223/30224	starsky-proxy	✅ 白名单	✅ 正常
[illegible]	(内网)	TCP 8529	ArangoDB	–	✅ 内网

六、安全组风险矩阵

6.1 严重 🔴

安全组	规则	优先级
[illegible]	0.0.0.0/0 ANY 1-65535	1（最高）
[illegible]	0.0.0.0/0 ANY 1-65535	50
[illegible]	0.0.0.0/0 ANY 1-65535	50

6.2 高危 🟠

风险	影响范围
SSH (22) 全网开放	7 个默认安全组 (IPv4+IPv6)
RDP (3389) 全网开放	7 个默认安全组 (IPv4+IPv6)

安全智能巡检

适用对象

- 安全合规巡检。
- 公网暴露面排查。
- 生产环境上线前检查。
- 定期网络安全治理。

巡检项

- 安全组是否开放 0.0.0.0/0 的 SSH、RDP、数据库端口
- 安全组是否存在过宽端口范围
- 安全组是否存在无用或重复规则
- EIP 是否直接绑定关键 ECS
- SNAT 是否覆盖过大网段
- DNAT 是否把高危端口暴露公网
- ELB 是否对公网开放敏感端口
- 子网 ACL 是否过宽

智能体自动化流程

1. 分页查询安全组和规则2. 查询 EIP 及绑定关系3. 查询 NAT 网关、SNAT、DNAT 规则4. 查询 ELB 监听器和后端服务5. 关联 ECS、VPC、子网和公网入口6. 按风险规则生成风险清单7. 给出最小权限修复建议8. 用户确认后执行修复9. 修复后再次查询验证

智能体执行CLI细节

以华南2为例：

```
# #######2#############################EIP ###SNAT/DNAT ###ELB ############

  Thought for 4s (ctrl+o to expand)
# Bash(echo "=== 1/5 ##### ===" && ctyun-cli vpc QueryVpcSecurityGroupsNew --
regionID "200000002530" --pageNo 1 --pageSize 50 2>&1 | python3 -c "
      import json,sys…)
  #  === 1/5 ##### ===
     Traceback (most recent call last):
       File "<string>", line 8, in <module>
     … +12 lines (ctrl+o to expand)

# Bash(echo "=== 3/5 NAT + SNAT/DNAT ===" && ctyun-cli nat Describe
InternetnatGateways --regionID "200000002530" 2>&1 | python3 -c "
      import json,sys…)
  #  === 3/5 NAT + SNAT/DNAT ===
     NAT####: 2
       nat-cn2              | natgw-3kdkicwibv | spec=None |
 vpc=vpc-4s7wjfnuyo | desc=''
```

```
     … +14 lines (ctrl+o to expand)
  #  (timeout 1m)

# Bash(echo "=== 4/5 ELB ### ===" && ctyun-cli elb ListLoadbalancers --
regionID "200000002530" 2>&1 | python3 -c "
      import json,sys…)
  #  Error: Exit code 1
     (eval):24: unmatched "

# Bash(echo "=== 5/5 ###### ===" && echo "--- ACL ---" && ctyun-cli vpc
 ListAclNew --regionID "200000002530" --pageNo 1 --pageSize 200 2>&1 | python3
 -c "
      import json,…)
  #  === 5/5 ###### ===
     --- ACL ---
     Traceback (most recent call last):
     … +35 lines (ctrl+o to expand)

  Thought for 5s (ctrl+o to expand)


# Bash(echo "=== ####### ==="
      ctyun-cli vpc QueryVpcSecurityGroupsNew --regionID "200000002530" --
pageNo 1 --pageSize 50 2>&1 | python3 -c "…)
  #  === ####### ===
     #SG#: 17
     [Default-Security-Group] sg-ffwqrn428b vpc=web-01-vpc vmNum=0
     … +49 lines (ctrl+o to expand)

# Bash(echo "=== SNAT ## ==="
      for natgw in natgw-3kdkicwibv natgw-hlslgtaye5; do…)
  #  === SNAT ## ===
     --- natgw-3kdkicwibv ---
     SNAT###: 0
     … +2 lines (ctrl+o to expand)

# Bash(echo "=== ELB ### ==="
      for elb in lb-ftgjxzh146 lb-s4pwzmjvhf lb-yvf4o0dcgz lb-0id7hpshei lb-
qk759ms5hz; do…)
  #  === ELB ### ===
     --- lb-ftgjxzh146 ---
       k8s-arangodb-producti-f698a7191b | TCP : 8529 | ACL=Close #
     … +11 lines (ctrl+o to expand)

  Thought for 9s (ctrl+o to expand)

# Bash(echo "=== SNAT###### ==="
      ctyun-cli nat ListInternetnatSnats --regionID "200000002530" --
natGatewayID "natgw-3kdkicwibv" 2>&1 | head -60…)
  #  === SNAT###### ===
     {
       "description": "##",
     … +63 lines (ctrl+o to expand)

# Bash(echo "=== ACL## ==="
      ctyun-cli vpc ListAclNew --regionID "200000002530" --pageNo 1 --pageSize
 200 2>&1 | python3 -c "…)
  #  === ACL## ===
     ACL##: 0
     … +7 lines (ctrl+o to expand)
  #  (timeout 30s)

  Thought for 6s (ctrl+o to expand)

# Bash(echo "=== nat-internet SNAT## ==="
      ctyun-cli nat ListInternetnatSnats --regionID "200000002530" --
natGatewayID "natgw-hlslgtaye5" 2>&1 | python3 -c "…)
  #  === nat-internet SNAT## ===
```

```
    SNAT###: 3
      ngwsr-fuqz74t6mo | subnet=subnet-lfasf93b7p | cidr=172.30.128.0/20 |
eip=14.22.89.70
    … +2 lines (ctrl+o to expand)

 Thought for 7s (ctrl+o to expand)
```

输出示例

🔴 *华南2 网络与访问控制安全审计报告*

区域：华南2（200000002530，cn-huanan-02，3 AZ）
审计时间：2026-07-02
资源概览：9 VPC / 12 子网 / 17 安全组（136 规则）/ 6 EIP / 2 NAT 网关 / 5 ELB（8 监听器）/ 11 ECS

一、关键发现总览

风险等级	数量	说明
🔴 严重	5	K8s API 公网暴露、ArangoDB 公网暴露、3 个安全组全端口全协议对公网开放
🟠 高危	16	7 个默认安全组 SSH/RDP 对公网开放、4 个 EIP 绑公网 ELB 无访问控制、Apollo 配置中心暴露、IAM 代理暴露、3 个子网双 NAT 网关 SNAT
🟡 中危	4	删除保护全关、无健康检查目标组、空 VPC/空安全组、无 ACL

二、严重风险 🔴

2.1 Kubernetes API Server 公网暴露 — 严重

字段	值
ELB	cce-apiserver-jpPfvg7xwF（lb-0id7hpshei）
公网 IP	[redacted]（10 Mbps cn2）
监听端口	TCP 6443（K8s API Server）
访问控制	无（Close）
后端	3 个 K8s master 节点（172.30.144.10/11/12）

影响：任何人均可从公网访问 Kubernetes API。攻击者可扫描 API 版本、探测集群配置、尝试凭证爆破。

2.2 ArangoDB 数据库双重暴露（内网+公网）— 严重

路径	ELB	IP	端口
公网	[redacted]（lb-yvf4o0dcgz）	[redacted]	TCP 30678
内网	[redacted]（lb-ftgjxzh146）	[redacted]	TCP 8529

两个 ELB 后端指向**完全相同**的 3 台 VM（172.30.144.14/15/17:31786），且公网 ELB **无访问控制**。

2.3 三个安全组全端口全协议对公网开放 — 严重

安全组	规则描述	优先级
sg-v6d6rmtvu9（[redacted]）	0.0.0.0/0 ANY 1-65535 ACCEPT	1（最高）
sg-cx7xj8m29t（[redacted]）	0.0.0.0/0 ANY 1-65535 ACCEPT	50
sg-hga9sggpt6（[redacted]）	0.0.0.0/0 ANY 1-65535 ACCEPT	50

> ⚠️ *iam-proxy-sg 的规则描述为 [redacted]，但规则仍处于最高优先级活跃状态。*

三、高危风险 🟠

3.1 7 个默认安全组 SSH (22) + RDP (3389) 对公网全开

每个 VPC 的系统默认安全组 ("Default-Security-Group") 均包含以下规则：

```
Ingress | 0.0.0.0/0 | TCP 22   | ACCEPT | Priority 99
Ingress | ::/0      | TCP 22   | ACCEPT | Priority 99
Ingress | 0.0.0.0/0 | TCP 3389 | ACCEPT | Priority 99
Ingress | ::/0      | TCP 3389 | ACCEPT | Priority 99
```

VPC	默认安全组 ID
web-01-vpc	sg-ffwqrn428b
vpc-for-ai-agent	sg-0fj2mcrv5f
tf-vpc-for-oceanfs1	sg-m9eiqlg3d7
tf-vpc-for-oceanfs	sg-ing6jmggq1
vpc-a204-zhangtao	sg-p4eowx0vq1
vpc-8eb4-zhangtao	sg-1o51rhsinm
auto_nodelvpc	sg-2ariylrvtb

3.2 Apollo 配置中心公网暴露 (HTTP 明文，无 ACL)

ELB	公网 IP	服务	端口	协议
[illegible]	[illegible]	config-service	31910	HTTP (明文)
[illegible]	[illegible]	admin-service	30571	HTTP (明文)

Apollo 配置中心通常存储数据库连接串、密钥等。admin-service 允许管理所有配置。

3.3 IAM 代理测试服务公网暴露

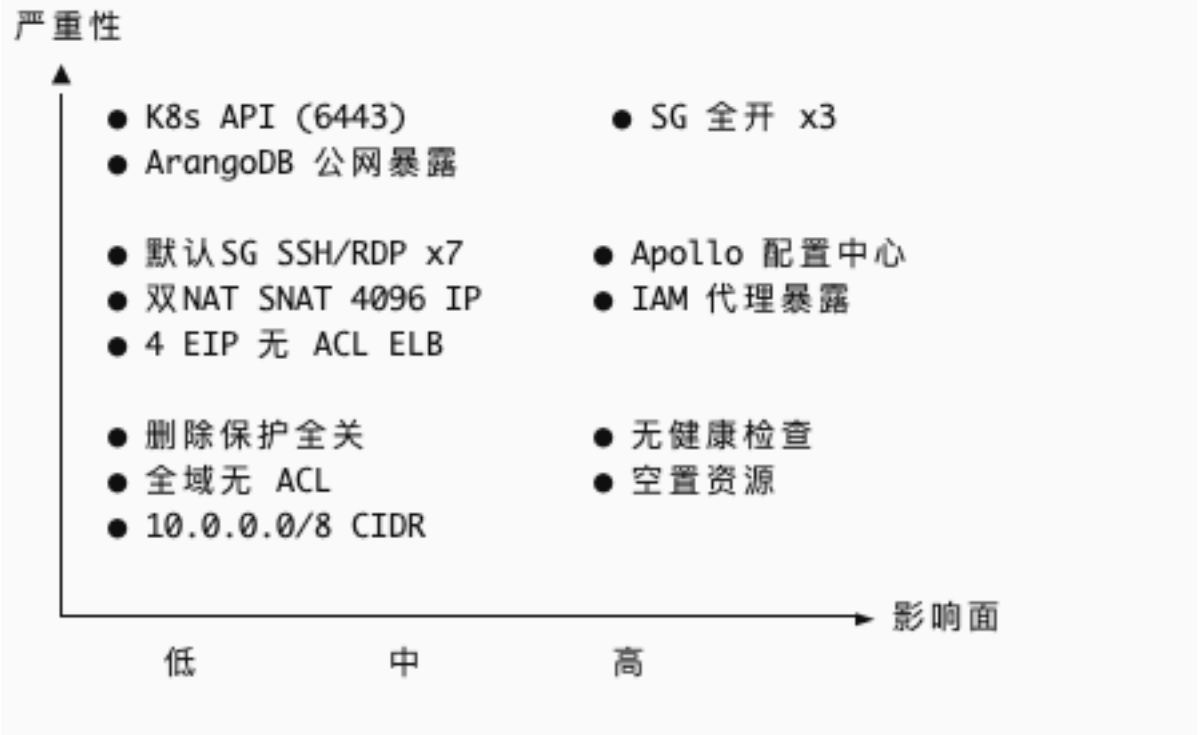
ELB	公网 IP	端口	ACL
[illegible] (lb-qk759ms5hz)	[illegible]	TCP 2080	无

名为 "iam-test"，可能涉及身份认证代理，暴露在公网且无访问控制。

3.4 三个子网双 NAT 网关 SNAT，最大覆盖 4096 IP

子网	CIDR	IP 数	NAT 1 (cn2)	NAT 2 (internet)
subnet-lfasf93b7p	**172.30.128.0/20**	4096	✅	✅
subnet-kbfbpb6c4b	172.30.144.0/24	256	✅	✅
subnet-kn78lbkeaj	172.30.147.0/24	256	✅	✅

这些子网内任何资源可通过**两条出口**不受限制地访问公网。


严重性
● K8s API (6443)
● ArangoDB 公网暴露
● SG 全开 x3
● 默认SG SSH/RDP x7
● 双NAT SNAT 4096 IP
● 4 EIP 无 ACL ELB
● Apollo 配置中心
● IAM 代理暴露
● 删除保护全关
● 全域无 ACL
● 10.0.0.0/8 CIDR
● 无健康检查
● 空置资源
影响面
低
中
高