

TC-AICC可信计算 快速入门



腾讯云

【 版权声明 】

©2013–2026 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

快速入门

Proxy 集成快速入门

Node.js SDK 接入快速入门

快速入门

Proxy 集成快速入门

最近更新时间：2026-10-28 15:51:39

本文介绍在业务侧部署安全代理（AICC-Proxy）并通过 Proxy 方式接入 TC-AICC可信计算（Artificial Intelligence Confidential Computing, AICC）机密推理服务的方法。

操作场景

当您在公有云上使用大模型处理私有数据，可以在业务侧部署 AICC-Proxy。部署后，业务系统向安全代理发送标准的 OpenAI 或 Anthropic 协议请求，由安全代理完成端到端加密后转发至 AICC 服务端，推理结果加密返回并解密后交回业务系统。

前提条件

- 已提交 [AICC 试用申请](#) 并经审核通过，服务已开通。
- 已获取调用地址与调用凭据。调用地址可在 [高性能应用服务（Hyper Application Inventor, HAI）控制台](#) 获取。
- 已准备一台可运行容器引擎（Docker）的机器，用于部署安全代理，建议部署在企业信任域，网络可达高性能应用服务 HAI 可信集群。建议规格 4C8G（4 核 CPU、8 GB 内存），最低规格 2C4G（2 核 CPU、4 GB 内存），Linux 操作系统。

步骤1：部署 AICC-Proxy

1. 拉取镜像

在部署机上执行以下命令，拉取安全代理镜像（公开镜像仓库）。

```
docker pull ccr.ccs.tencentyun.com/aicc/aicc-proxy-hybrid:v1.2.0
```

2. 启动容器

执行以下命令，以容器方式启动安全代理。请将命令中的 `<调用地址>` 替换为 HAI 控制台提供的实际地址。

```
docker run -d \  
  --name aicc-proxy-hybrid \  
  --restart unless-stopped \  
  -p 8083:8083 \  
  -e AICC_INFER_OPENAI_BASE_URL=<调用地址> \  
  -e AICC_INFER_ANTHROPIC_BASE_URL=<调用地址> \  
  -e AICC_INFER_API_KEY=<调用凭据>
```

```
-e AICC_INFER_RESPONSES_BASE_URL=<调用地址> \
-e AICC_SECURITY_ENDPOINT=<调用地址>/aicc \
-e AICC_POOL_MODE=exclusive \
ccr.ccs.tencentyun.com/aicc/aicc-proxy-hybrid:v1.2.0
```

各参数说明如下。

参数	说明
<code>--name aicc-proxy-hybrid</code>	容器名称。
<code>--restart unless-stopped</code>	容器退出时自动重启（手动停止除外）。
<code>-p 8083:8083</code>	端口映射。将容器内的 <code>8083</code> 端口映射到部署机的 <code>8083</code> 端口。
<code>-e AICC_INFER_OPENAI_BASE_URL</code>	OpenAI 协议推理请求的调用地址。
<code>-e AICC_INFER_ANTHROPIC_BASE_URL</code>	Anthropic 协议推理请求的调用地址。
<code>-e AICC_INFER_RESPONSES_BASE_URL</code>	Responses 协议推理请求的调用地址。
<code>-e AICC_SECURITY_ENDPOINT</code>	安全能力接口地址，格式为 <code><调用地址>/aicc</code> 。
<code>-e AICC_POOL_MODE</code>	资源池模式，示例值为 <code>exclusive</code> 。

步骤2：确认容器运行状态

执行以下命令，查看容器是否已正常运行。

```
docker ps --filter "name=aicc-proxy-hybrid"
```

输出结果中出现 `aicc-proxy-hybrid` 容器，且状态为 `Up`，表示安全代理已启动。
如需查看启动日志以排查异常，可执行以下命令。

```
docker logs aicc-proxy-hybrid
```

步骤3：发起推理调用

安全代理启动后，在部署机的 8083 端口提供推理服务。业务系统向 `http://<部署机 IP>:8083` 发送标准的 OpenAI 或 Anthropic 协议请求即可调用机密模型。以下命令中的<调用凭据>、<模型名称> 和 <协议版本> 均为示例占位符，请根据实际部署环境及控制台提供的调用信息替换后再执行。

以下为 OpenAI 协议的调用示例。

```
curl http://<部署机 IP>:8083/v1/chat/completions \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <调用凭据>" \
-d '{
  "model": "<模型名称>",
  "messages": [{"role": "user", "content": "你好"}]
}'
```

以下为 Anthropic 协议的调用示例。

```
curl http://<部署机 IP>:8083/v1/messages \
-H "Content-Type: application/json" \
-H "x-api-key: <调用凭据>" \
-H "anthropic-version: <协议版本>" \
-d '{
  "model": "<模型名称>",
  "max_tokens": 256,
  "messages": [{"role": "user", "content": "你好"}]
}'
```

① 说明：

上述示例中的请求路径、请求头与模型名称以控制台提供的调用信息和对应的协议规范为准。业务系统原有的 OpenAI 或 Anthropic 协议调用只需将请求地址改为安全代理地址，其余调用方式保持不变。

安全风险提示

- 访问控制：8083 端口只对内网业务网段开放，禁止暴露到公网；跨网络访问请走 VPN 或专线。
- 凭据管理：API Key 由调用方保管，安全代理不保存。

调用模式说明

模式	说明	适用场景
直连	不经过安全代理 Proxy，直接访问调用地址。该模式不启用端到端	仅建议在测试阶段使

模式	加密，安全等级较低。	用。
加密模式	请求经安全代理 Proxy 或 AICC-SDK 完成端到端加密后调用。	生产环境建议使用该模式。

Node.js SDK 接入快速入门

最近更新时间：2026-10-28 15:51:54

本文介绍使用 AICC Node.js SDK 在业务代码中接入 AICC 机密推理服务的方法。AICC Node.js SDK 用于对发往大语言模型（Large Language Model，LLM）的请求与响应做端到端加密：请求在您的程序中加密后才离开本机，响应在您的程序中解密。SDK 只负责会话握手与加解密，超文本传输协议（HyperText Transfer Protocol，HTTP）请求由您的程序自行发送。当前 SDK 版本为 1.3.0。

操作场景

业务系统需要直接调用 AICC 机密推理服务，且具备代码改造条件时，可在业务代码中集成 AICC Node.js SDK。集成后由业务侧自行发起加密请求并解密响应，无需在业务侧额外部署安全代理。若业务系统不便修改代码，可改用安全代理方式接入，请参见 [AICC-Proxy 集成快速入门](#)。

前提条件

- 运行环境为 Node.js 18 及以上版本。
- SDK 同时提供 ECMAScript 模块（ES Modules，ESM）与 CommonJS 两种入口，并带有 TypeScript 类型声明。
- 已提交 AICC 试用申请并经审核通过，服务已开通。
- 已获取 API 密钥（Application Programming Interface Key，API Key）、安全会话服务地址、网关地址与模型名称。
- 运行环境可访问安全会话服务与网关地址。

① 说明：

请妥善保管 API 密钥，不要将 API 密钥写入代码仓库。

概念说明

- **安全会话服务**：负责握手并下发会话密钥的服务，地址对应代码中的 `<AICC_BASE_URL>`。
- **网关**：接收加密请求并返回加密响应的服务，地址对应代码中的 `<AICC_GATEWAY_URL>`。
- **端点**：服务端集群的路由标识。SDK 按模型名称自动发现端点，您无需处理。
- **ModelPool**：按模型名称发现端点，并管理各端点的会话池。
- **Request**：一轮 HTTP 请求的加解密上下文。每轮请求需新建一个。

步骤1：安装 SDK

① 说明：

SDK 暂不直接提供下载。[AICC 试用申请](#)通过后，请您[提交工单](#)联系我们获取。

执行以下命令，安装 SDK。

```
npm install @tencent/tcaicc
```

步骤2：发起非流式请求

以下示例使用 OpenAI Chat Completions 协议。请先将代码中尖括号包围的占位符替换为实际值。

```
import { Protocol, SessionFactory, SessionPool, defaultPolicy,
newModelPool } from '@tencent/tcaicc'

const BASE_URL = '<AICC_BASE_URL>' // 安全会话服务地址
const GATEWAY_URL = '<AICC_GATEWAY_URL>' // 网关地址
const API_KEY = '<API_KEY>'
const MODEL = '<MODEL_NAME>' // 模型名称
const TTL = 8 * 60 * 60 // 会话寿命与模型/端点绑定时长, 单位为秒

async function main(): Promise<void> {
  // 1. 创建模型池。回调必须为每个端点新建一个独立的工厂。
  const models = newModelPool(BASE_URL, API_KEY, TTL, () =>
    new SessionPool(new SessionFactory(BASE_URL, API_KEY, { ttl: TTL }),
4), // 4 为池中最多保留的会话数
  )
  try {
    // 2. 每轮 HTTP 请求取一个 Request。
    const req = await models.newRequest(MODEL, [])
    const policy = defaultPolicy(Protocol.OPENAI)

    // 3. 加密请求体。
    const body = { model: MODEL, messages: [{ role: 'user', content: '你好' }] }
    const encBody =
req.encryptRequest(Buffer.from(JSON.stringify(body)), policy)

    // 4. 发送请求。
    const r = await fetch(GATEWAY_URL, {
      method: 'POST',
      body: encBody.toString(),
```

```
headers: {
  'Content-Type': 'application/json',
  'X-AICC-Mode': '1', // 标识这是密文请求
  Authorization: `Bearer ${API_KEY}`,
},
}))

const raw = Buffer.from(await r.arrayBuffer())
if (!r.ok) throw new Error(`gateway ${r.status}: ${raw.toString()}`)

// 5. 解密响应体。
const plain = req.decryptResponse(raw, policy)
console.log(JSON.parse(plain.toString()).choices[0].message.content)
req.close()
} finally {
  models.close()
}
}

main().catch((e) => {
  console.error(e)
  process.exit(1)
})
```

上述代码用到的三个接口如下。

new SessionFactory(baseUrl, apiKey, options)

参数	类型	必填	说明
baseUrl	string	是	安全会话服务的完整地址。SDK 直接向该地址发送 POST 请求。
apiKey	string	是	API 密钥。
options.ttl	number	是	会话寿命，单位为秒。取值为 0 表示使用默认值 86400。

new SessionPool(factory, num)

参数	类型	必填	说明
----	----	----	----

<code>factory</code>	SessionFactory	是	会话工厂。
<code>num</code>	number	是	池中最多保留的会话数，取值需大于等于 1。

newModelPool(baseUrl, apiKey, ttl, build)

参数	类型	必填	说明
<code>baseURL</code>	string	是	安全会话服务的完整地址。
<code>apiKey</code>	string	是	API 密钥。
<code>ttl</code>	number	是	模型与端点的绑定时长，单位为秒。取值为 -1 表示永不过期。
<code>build</code>	function	是	为指定端点创建会话池的回调。必须为每个端点新建独立的 SessionFactory。

步骤3：发起流式请求

流式请求的步骤与非流式请求相同，区别有以下三处。

1. 请求体设置 `stream: true`。
2. 请求头增加 `Accept: text/event-stream`。
3. 使用 StreamDecryptor 逐个解密服务器发送事件（Server-Sent Events, SSE）。SDK 不解析 SSE，您的程序需自行读取每一行。

```
import type { Policy, Request } from '@tencent/tcaicc'

async function streamChat(req: Request, policy: Policy, encBody: Buffer): Promise<void> {
  const r = await fetch(GATEWAY_URL, {
    method: 'POST',
    body: encBody.toString(),
    headers: {
      'Content-Type': 'application/json',
      Accept: 'text/event-stream',
      'X-AICC-Mode': '1',
      Authorization: `Bearer ${API_KEY}`,
    },
  },
```

```
    })

    if (!r.ok || !r.body) throw new Error(`gateway ${r.status}: ${await
r.text()}`)

    const sd = req.newStreamDecryptor()
    const decoder = new TextDecoder()
    let buf = ''
    let seenLast = false

    for await (const part of r.body) {
        buf += decoder.decode(part, { stream: true })
        let nl: number
        while ((nl = buf.indexOf('\n')) >= 0) {
            const line = buf.slice(0, nl).replace(/\r$/, '')
            buf = buf.slice(nl + 1)
            if (!line.startsWith('data: ')) continue
            const data = line.slice('data: '.length)
            if (data === '[DONE]') continue

            const [out, isLast] = sd.decryptEvent(Buffer.from(data), policy)
            if (out !== null) { // out 为 null 表示合成尾片, 请勿转发
                const choices = JSON.parse(out.toString()).choices ?? []
                if (choices.length > 0)
                    process.stdout.write(choices[0].delta?.content ?? '')
            }
            if (isLast) seenLast = true
        }
        if (seenLast) break
    }
    if (!seenLast) throw new Error('stream truncated: final chunk
missing')
    process.stdout.write('\n')
}
```

① 说明:

- 流式响应必须收到尾片 (`isLast` 为 `true`)。未收到尾片表示流被截断。
- 不要在同一个 Request 上同时读取多个流。

- 加密的流式请求不支持 `n > 1`。

相关配置

选择协议

`defaultPolicy` 的参数决定加密哪些字段，取值需与您所调用的接口一致。

接口	协议常量
<code>/v1/chat/completions</code>	<code>Protocol.OPENAI</code>
<code>/v1/messages</code>	<code>Protocol.ANTHROPIC</code>
<code>/v1/responses</code>	<code>Protocol.RESPONSES</code>

使用 OpenAI Chat Completions 以外的协议时，还需在建池回调中为工厂设置密文绑定的协议。

```
import { SessionFactory, Protocol } from '@tencent/tcaicc'

const factory = new SessionFactory(BASE_URL, API_KEY, { ttl: TTL })
factory.setProtocol(Protocol.ANTHROPIC) // 或 Protocol.RESPONSES
```

使用须知

- 每轮 HTTP 请求需新建一个 Request。同一轮请求的所有密文必须来自同一个 Request。
- 加密后请尽快发送请求。加密时间与服务端当前时间的差值超过 1 小时，请求会被拒绝。
- 程序退出前调用 `models.close()`。

常见问题

网关返回 400 且错误码为 `decrypt_failed`，如何处理？

通常是因为同一轮请求中混用了不同 Request 加密的密文，或加密后等待时间过长才发送。请为每轮请求新建 Request，并在加密后立即发送。

抛出 `SecuritySessionError` 时如何处理？

表示握手失败，或服务端拒绝了请求。请读取 `e.code` 判断原因。如果 `e.code` 为 `AttestationFailed` 或 `CertVerifyFailed`，表示 SDK 判定服务端不可信，请停止使用该服务并联系技术支持，请勿绕过验证。

抛出 `TransportError` 时如何处理？

表示网络失败，请稍后重试。

抛出 DecryptError 时如何处理？

表示密钥不匹配或密文已损坏。请丢弃当前响应或流，重新发起请求。

抛出 ClosedError 时如何处理？

表示对象已关闭。请重新调用 `models.newRequest` 取一个新的 Request。

抛出 ErrPolicy、ErrBodyFormat 或 ErrUnsupported 时如何处理？

表示请求体或加密策略不符合要求。请确认请求体是合法的 JavaScript 对象表示法（JavaScript Object Notation, JSON），并且协议常量与所调用的接口一致。