

容器服务

AI 模型部署



腾讯云

【 版权声明 】

©2013–2025 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

AI 模型部署

Embedding 模型部署

AI 网关部署

AI 模型部署

Embedding 模型部署

最近更新时间：2025-10-13 14:56:12

本文档介绍如何在 TKE 上部署 Embedding 模型。

简介

Embedding 模型是将文本、图像等非结构化数据转化为低维稠密向量的 AI 模型，这些向量能精准捕捉数据的语义或特征关联，让机器可通过向量计算理解数据间的相似性，是语义检索、推荐系统、聚类分析等场景的核心基础，在文档检索中，可通过对比查询与文档的 Embedding 向量快速匹配相关内容。

tke-ai-playbook 是 TKE 团队开源的 AI 大模型相关脚本，包含模型下载、部署推理服务、性能测试等模块，可在 TKE 一站式体验 AI 相关功能，点此获取 [开源链接](#)。

环境准备

步骤1：创建 TKE 标准集群

1. 登录 [腾讯云容器服务控制台](#)，选择左侧导航栏中的**集群**。
2. 单击集群列表上方的**新建**。
3. 在集群类型中，选择**标准集群**。
4. 在**创建集群 > 网络配置**中，选择 **VPC-CNI**，其余参数保持默认即可。



5. 在**创建集群 > 组件配置**中，存储组件勾选 **CFS**（用于持久化存储大模型权重文件；若创建时未勾选，后续可在“集群 - 组件管理”中安装，操作详情请参见 [通过组件管理页安装](#)）。

存储组件 ☒ CBS ⓘ ☐ COS ⓘ ☐ CFSTurbo ⓘ ☒ CFS ⓘ

▶ 参数配置

TKE 提供CSI组件支持多种类型的存储，点击了解 [如何选择集群存储](#)

监控组件 ☒ monitoragent 免费使用

该组件负责采集容器、Pod、节点维度的监控数据，对接基础监控提供容器运维所需的基础监控大盘和告警功能

网络组件 ☒ ip-masq-agent

该组件负责管理容器出站流量的源 IP 伪装规则，通过动态配置 iptables 确保 Pod 访问集群外目标时使用节点 IP 作为源地址。

6. 创建下载节点池。

选择 3 个节点（用于后续 3 个并发下载任务），推荐机型 SA5.LARGE8（性价比高，满足下载需求），系统盘 $\geq 300\text{GB}$ ，并分配“免费公网 IP”及带宽 $\geq 100\text{Mbps}$ （保障单节点下载速度，缩短整体耗时）。

操作系统 TencentOS Server 推荐

[查看TencentOS Server的优势](#)

计费模式

支持网络 ⓘ CIDR: 10.0.0.0/16

机型配置

系统盘

范围：20~2048，步长：1
当前镜像大小为20GB,系统盘最小需要购买20GB

数据盘 ☐ 购买数据盘

公网带宽 ☒ 分配免费公网IP, [查看详情](#)

7. 创建推理节点：需 1 个 GPU 节点（大模型推理依赖高显存），推荐选择 PNV5b.8XLARGE96机型（单卡 48GB 显存，满足模型需求）。

步骤2：创建存储组件

需创建 StorageClass（CFS 存储）和 PVC，确保模型下载后能被推理服务直接调用：

1. 进入集群的存储 > StorageClass 页面，单击**新建**，存储类型选“文件存储 CFS”，可用区与推理节点一致（减少跨区访问延迟），其余参数默认；

名称: cfs-ai
最长63个字符，只能包含小写字母、数字及分隔符("-",)，且必须以小写字母开头，数字或小写字母结尾

地域: 华南地区(广州)

Provisioner: 云硬盘CBS(CSI) 文件存储CFS 文件存储CFS turbo ①

实例创建模式: 创建新实例 共享实例
使用该模式创建的PVC，在挂载时每个PVC将共享同一CFS实例的不同子目录，共享的CFS实例及子目录由系统自动创建

可用区: 广州三区 广州四区 广州六区 广州七区

2. 进入存储 > PVC 页面，单击**新建**，名称为 ai-model（与后续下载脚本参数一致，避免手动修改配置），存储类选择上一步创建的 StorageClass 对象。

名称: ai-model
最长63个字符，只能包含小写字母、数字及分隔符("-",)，且必须以小写字母开头，数字或小写字母结尾

命名空间: default

Provisioner: 云硬盘CBS(CSI) 文件存储CFS 文件存储CFS turbo ① 对象存储COS ① 数据加速GooseFS ①
CFS按照存储量计费，具体请查看CFS计费说明

读写权限: 单机读写 ① 多机只读 ① 多机读写

是否指定StorageClass: 不指定 指定
静态创建的PersistentVolume中,StorageClass类型为所选类型

StorageClass: cfs-ai (按量计费)

是否指定PersistentVolume: 不指定 指定

部署模型

本实践使用 vllm 推理框架，该框架支持的 embedding 模型主要有：**BGE 系列模型**和 **E5-Mistral 系列模型**，这两个系列的模型均可使用 vllm 框架进行模型推理，**本文采用 E5-mistral-7b-instruct 模型完成验证。**

E5-mistral-7b-instruct: 基于 Mistral-7B 架构微调的指令遵循模型，融合了 E5 系列在检索增强领域的优势与 Mistral 的高效推理能力；在中文与多语言理解、长文本处理和检索增强生成（RAG）场景中表现突出，并且拥有较长的上下文长度，支持 vllm 框架下的推理服务。

步骤1：下载依赖

登录集群中的节点，节点镜像为 TencentOS 可以使用以下命令进行安装：

```
yum install -y jq
```

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
| bash
```

步骤2：下载模型

进入集群中，拉取 tke-ai-playbook，机器若提示安装 git，请先安装 git 后再拉取代码：

```
git clone https://github.com/tkestack/tke-ai-playbook.git
```

执行 playbook 中的下载脚本 tke-llm-downloader.sh，该脚本可以使用多个节点并发拉取模型，加快模型下载速度，PVC 名称需与前述一致，模型名称参数则选择目标模型名称，可前往 [ModelScope](#) 获取模型名称。

tke-llm-downloader.sh 是一个在 Kubernetes 集群中自动化下载大语言模型的工具脚本，主要功能包括：

- **模型下载管理**：支持从 ModelScope 或 Hugging Face 下载大语言模型到 Kubernetes PVC 存储；
- **并发下载优化**：通过 Kubernetes Job 的 completions 和 parallelism 参数实现多 Pod 并发下载，提升下载速度；
- **节点调度控制**：避免多个下载 Pod 调度到同一节点。

```
bash scripts/tke-llm-downloader.sh --pvc ai-model --completions 3 --
parallelism 3 --model intfloat/e5-mistral-7b-instruct
```

预期结果：三个 Pod 状态为 Completed 后，模型下载工作则执行完成。

tke-llm-downloader-6bcnr-0-dq2mv	0/1	Completed	0
tke-llm-downloader-6bcnr-1-7nlfs	0/1	Completed	0
tke-llm-downloader-6bcnr-2-5kmq7	0/1	Completed	0

步骤3：模型部署

vllm-inference-tke 是一个部署 vllm 推理框架并暴露服务的 Helm Chart 包，可支持单机单卡、单机多卡、多机多卡等多种部署模式，修改自定义参数即可部署自建大模型。

进入 vllm-inference-tke 目录，运行 vllm-inference-tke 的 Chart 包，该 Chart 可以在 TKE 上部署基于 vLLM 的 OpenAI 兼容 API 服务，APIKey 需要自行设置，默认无需 APIKey，修改 model name 为自己想要部署的模型，并且 PVC 修改为自己集群的 PVC 名称，具体参数修改参考下述，目前默认参数支持单机单卡部署。

```
cd tke-ai-playbook/helm-charts/vllm-inference-tke
vim values.yaml
helm install vllm-service .
```

本文提供部署 E5-mistral-7b-instruct 模型的具体参数，具体的 values.yaml 文件内容如下：

```
model:
  name: "intfloat/e5-mistral-7b-instruct"
  pvc:
    enabled: true
    name: "ai-model"
    path: "intfloat/e5-mistral-7b-instruct"
  local:
    enabled: false
    path: "/data0/intfloat/e5-mistral-7b-instruct"

server:
  replicas: 1
  lwsGroupSize: 1
  image: "ccr.ccs.tencentyun.com/tke-ai-playbook/vllm-openai:v0.10.1-20250801"
  imagePullPolicy: IfNotPresent
  apiKey: ""
  resources:
    requests:
      nvidia.com/gpu: 1
    limits:
      nvidia.com/gpu: 1
  args:
    tpSize: 1
    ppSize: 1
    epEnabled: false
    maxModelLen: 2048
    maxBatchSize: 32
  extraArgs:
    - --disable-log-requests
    - --cuda-graph-sizes 1 2 4 8 16 24 32
  env:
    - name: VLLM_WORKER_MULTIPROC_METHOD
      value: "spawn"
  service:
    enabled: true
    type: LoadBalancer
    port: 60000
```

```
labels: {}  
podAnnotations: {}
```

部署的模型会通过 service 暴露端点和端口，查看 Pod 日志看模型状态是否健康。

```
kubectl get svc | grep vllm-service  
kubectl logs -f pod-name
```

预期结果：

```
(APIServer pid=1) INFO 09-25 19:54:39 [api_server.py:1847] Starting vLLM API server 0 on http://0.0.0.0:60000  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:29] Available routes are:  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /openapi.json, Methods: HEAD, GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /docs, Methods: HEAD, GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /docs/oauth2-redirect, Methods: HEAD, GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /redoc, Methods: HEAD, GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /health, Methods: GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /load, Methods: GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /ping, Methods: POST  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /ping, Methods: GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /tokenize, Methods: POST  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /detokenize, Methods: POST  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /v1/models, Methods: GET  
(APIServer pid=1) INFO 09-25 19:54:39 [launcher.py:37] Route: /version, Methods: GET
```

步骤4：测试向量嵌入

```
curl -X POST http://xx.xx.xx.xx:60000/v1/embeddings \  
-H 'Content-Type: application/json' \  
-H 'Authorization: Bearer 3bxxxxxxxxxxxxxxxx1a1d' \  
-d '{  
    "model": "intfloat/e5-mistral-7b-instruct",  
    "input": "用Python实现快速排序算法"  
}'
```

预期结果：

```
{  
  "id": "embd-f3eda39c42dc4ab5a264964760ba370c",  
  "object": "list",  
  "created": 1758856987,  
  "model": "intfloat/e5-mistral-7b-instruct",  
  "data": [  
    {  
      "index": 0,  
      "object": "embedding",  
      "embedding": [0.022728852927684784,-0.012755637988448143,-0.005278590135276318,-0.007603001315146685,0.014450282789766788,-0.014072422869503498,0.0020195473916828632,0.015618214383721352,0.03764859959483147,0.00032651255605742335,0.0187444146451354027,-0.00916596781462431,0.012629684992134571,0.011759461835026741,0.007545750122517347,-0.006068660877645016,0.010992291383445263,-0.003555319271981716,-0.0031488335225731134,-0.008186967112123966,-0.014770891517400742,-0.018125830218195915,-0.0167403444647789,-0.011456028558313847,0.011249923147261143,0.011965567246079445,0.02628989703953266,-0.013866317458450794,0.0009203181252814829,-0.0017590528586879373,0.015721267089247704,0.005450344644486904,0.03471732139587402,0.008845359086990356,-0.016705993562936783,-0.002171263797208667,0.01946551539003849,-0.002624982036650181,-0.021423518657684326,0.025213567540049553,-0.0038015006575733423,-0.008106814697384834,-0.006143087986856699,-0.0032547488808631897,0.0013389699161052704,0.01039115060120821,-0.0013003250351175666,0.015217453241348267,-0.0012144477805122733,0.004903592634946108,-0.01469073910266161,-0.009057190269231796,-0.016156379133462906,0.15709815919399261,0.004811990074813366,-0.005178399849683046,0.0220990851521492,-0.006246140692383051,-0.0095438277348876,-0.02022123523056507,-8.945549780037254e-05,0.00525282695889473,-0.01725560799241066,-0.02862575836479664,0.00829574465751648,0.0007295990362763405,-0.008061014115810394,-0.012309076264500618,0.02285480499267578,-0.02393113449215889,-0.01504569873213768,-0.01777087152004242,-0.008026662282645702,0.005833929404616356,0.005178399849683046,0.001452041557058692,-0.007832007482647896,-0.007110638543963432,-0.004686036612838507,-0.02759523130953312,0.005095385015010834,-0.01163923367857933,0.015125851146876812,-0.0038931032177060843,-0.019488416612148285,0.008473224937915802,0.001419837586581707,-0.010562905110418797,0.014793791808187962,-0.010757559910416603,-0.007528574671596289,-0.01053427904844284,-0.002437483286485076,0.052075982093811035,9.446500189369544e-05,-0.009183143265545368,-0.022225039079785347,-0.0021941643208265305,0.00021022041619289666,-0.012893041595816612,0.019946428015828133,-0.00878810789436102,-0.010225120931863785,0.01740446127951145,0.001958001870661974,-0.0132480012252926
```

常见问题

部署 vllm-service 后日志显示未找到模型？

检查模型是否已下载完成，Pod 状态应为 Completed。如果模型下载的 Pod 还是 Running 状态，请等待下载结束后重启服务。

部署 vllm-service 后日志显示 CudaOutofMemory？

说明 GPU 显存不足，请选用更大显存的 GPU，部署本文模型需单卡 L20 及以上机型。

AI 网关部署

最近更新时间：2025-10-10 10:36:21

本文档介绍在 TKE 上如何部署 Envoy AI Gateway 并接入大模型。

简介

AI 网关是面向大模型服务的专用流量治理组件，基于传统 API 网关扩展而来，核心能实现多模型按需切换、租户鉴权与配额管控、Token 级限流、内容安全合规校验，还可在模型异常时自动切换保障稳定性。

Envoy AI Gateway 是一个开源的、基于 Kubernetes 原生架构的 AI 网关，专门用于管理和路由大模型服务流量。该项目构建在成熟的 Envoy Proxy 之上，为应用客户端与各种 AI 服务提供商之间提供了一个统一、安全且可扩展的接入层。

环境准备

1. 已创建并部署好 TKE 集群，建议选择[香港地域](#)集群，如果您还没有集群，请参见 [创建集群](#)。
2. 已创建节点池，并且节点池内有至少3个节点，推荐机型为 SA5.LARGE8。

环境验证

验证 kubectl:

```
kubectl version --client
```

预期结果:

```
Client Version: v1.32.2-tke.6  
Kustomize Version: v5.5.0
```

验证 helm 安装:

```
helm version
```

预期结果:

```
version.BuildInfo{Version:"v3.19.0",  
GitCommit:"3d8990f0836691f0229297773f3524598f46bda6",  
GitTreeState:"clean", GoVersion:"go1.24.7"}
```

如果显示未找到 helm 命令，TencentOS 系统可以用下列命令一键安装：

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
| bash
```

验证 curl 安装：

```
curl --version
```

```
curl 8.4.0 (x86_64-koji-linux-gnu) libcurl/8.4.0 OpenSSL/3.0.12
zlib/1.2.13 brotli/1.1.0 libidn2/2.3.4 libpsl/0.21.2 (+libidn2/2.3.4)
libssh/0.10.5/openssl/zlib nghttp2/1.58.0 OpenLDAP/2.6.5
Release-Date: 2023-10-11
Protocols: dict file ftp ftps gopher gophers http https imap imaps ldap
ldaps mqtt pop3 pop3s rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTPS-proxy IDN
IPv6 Kerberos Largefile libz NTLM PSL SPNEGO SSL threadsafe TLS-SRP
UnixSocket
```

部署 AI 网关

第一步：部署 Envoy Gateway

Envoy AI Gateway 建立在 Envoy Gateway 之上，使用 Helm 安装并等待部署准备就绪：

```
helm upgrade -i eg oci://docker.io/envoyproxy/gateway-helm \
  --version v0.0.0-latest \
  --namespace envoy-gateway-system \
  --create-namespace

kubectl wait --timeout=2m -n envoy-gateway-system deployment/envoy-
gateway --for=condition=Available
```

第二步：部署 Envoy AI Gateway

安装 AI Gateway 的 Helm chart，完成后等待部署准备就绪：

```
helm upgrade -i aieg oci://docker.io/envoyproxy/ai-gateway-helm \
  --version v0.0.0-latest \
```

```
--namespace envoy-ai-gateway-system \  
--create-namespace  
  
kubectl wait --timeout=2m -n envoy-ai-gateway-system deployment/ai-  
gateway-controller --for=condition=Available
```

第三步：配置 Envoy AI Gateway

安装 Envoy AI Gateway 后，将特定于 AI Gateway 的配置应用于 Envoy Gateway，重新启动部署，然后等待其准备就绪：

```
kubectl apply -f https://raw.githubusercontent.com/envoyproxy/ai-  
gateway/main/manifests/envoy-gateway-config/redis.yaml  
kubectl apply -f https://raw.githubusercontent.com/envoyproxy/ai-  
gateway/main/manifests/envoy-gateway-config/config.yaml  
kubectl apply -f https://raw.githubusercontent.com/envoyproxy/ai-  
gateway/main/manifests/envoy-gateway-config/rbac.yaml  
  
kubectl rollout restart -n envoy-gateway-system deployment/envoy-gateway  
  
kubectl wait --timeout=2m -n envoy-gateway-system deployment/envoy-  
gateway --for=condition=Available
```

第四步：检查健康状态

检查 AI 网关 Pod：

```
kubectl get pods -n envoy-ai-gateway-system
```

检查 Envoy Gateway 容器：

```
kubectl get pods -n envoy-gateway-system
```

第五步：部署测试后端

首先部署包含测试后端的基本 AI 网关设置：

```
kubectl apply -f https://raw.githubusercontent.com/envoyproxy/ai-  
gateway/main/examples/basic/basic.yaml
```

等待网关容器准备就绪：

```
kubectl wait pods --timeout=2m \      -l gateway.envoyproxy.io/owning-  
gateway-name=envoy-ai-gateway-basic \      -n envoy-gateway-system \      --  
for=condition=Ready
```

设置网关 URL：

```
export GATEWAY_URL=$(kubectl get gateway/envoy-ai-gateway-basic -o  
jsonpath='{.status.addresses[0].value}')
```

验证 URL 是否可用：

```
echo $GATEWAY_URL
```

第六步：测试 AI 网关

打开一个新终端，并使用以下命令向 AI 网关发送测试请求：

```
curl -H "Content-Type: application/json" \  
-d '{ "model": "some-cool-self-hosted-model", "messages":  
[{"role": "system", "content": "Hi."}]}' \  
$GATEWAY_URL/v1/chat/completions
```

预期结果：

```
{"choices": [{"message": {"role": "assistant", "content": "I am  
inevitable."}}]}
```

使用 AI 网关接入大模型

第一步：下载配置模板

本文实践以 Kimi 大模型为例，先获取 AI 网关的默认模型配置模板，后续第二步修改模板中的 hostname 和 API_KEY，需要具备相关运营商的 API_KEY。

```
curl -O https://raw.githubusercontent.com/envoyproxy/ai-  
gateway/main/examples/basic/openai.yaml
```

第二步：修改配置模板

编辑文件 `openai.yaml` 替换其中的 `hostname` 和 `OPENAI_API_KEY`，下面以 Kimi 为例，yaml 文件配置参考如下：

```
apiVersion: aigateway.envoyproxy.io/v1alpha1
kind: AIGatewayRoute
metadata:
  name: envoy-ai-gateway-basic-openai
  namespace: default
spec:
  parentRefs:
    - name: envoy-ai-gateway-basic
      kind: Gateway
      group: gateway.networking.k8s.io
  rules:
    - matches:
        - headers:
            - type: Exact
              name: x-ai-eg-model
              value: kimi-k2-0905-preview
      backendRefs:
        - name: envoy-ai-gateway-basic-openai
---
apiVersion: aigateway.envoyproxy.io/v1alpha1
kind: AIServiceBackend
metadata:
  name: envoy-ai-gateway-basic-openai
  namespace: default
spec:
  schema:
    name: OpenAI
  backendRef:
    name: envoy-ai-gateway-basic-openai
    kind: Backend
    group: gateway.envoyproxy.io
---
apiVersion: aigateway.envoyproxy.io/v1alpha1
kind: BackendSecurityPolicy
metadata:
```

```
name: envoy-ai-gateway-basic-openai-apikey
namespace: default
spec:
  targetRefs:
    - group: aigateway.envoyproxy.io
      kind: AIServiceBackend
      name: envoy-ai-gateway-basic-openai
  type: APIKey
  apiKey:
    secretRef:
      name: envoy-ai-gateway-basic-openai-apikey
      namespace: default
---
apiVersion: gateway.envoyproxy.io/v1alpha1
kind: Backend
metadata:
  name: envoy-ai-gateway-basic-openai
  namespace: default
spec:
  endpoints:
    - fqdn:
        hostname: api.moonshot.cn
        port: 443
---
apiVersion: gateway.networking.k8s.io/v1alpha3
kind: BackendTLSPolicy
metadata:
  name: envoy-ai-gateway-basic-openai-tls
  namespace: default
spec:
  targetRefs:
    - group: 'gateway.envoyproxy.io'
      kind: Backend
      name: envoy-ai-gateway-basic-openai
  validation:
    wellKnownCACertificates: "System"
    hostname: api.moonshot.cn
---
apiVersion: v1
kind: Secret
```

```
metadata:
  name: envoy-ai-gateway-basic-openai-apikey
  namespace: default
type: Opaque
stringData:
  apiKey: sk-xxxxxxxxxxxxxxxx # Replace with your kimi API key.
```

第三步：应用配置

应用更新的配置并等待网关容器准备就绪。

```
kubectl apply -f openai.yaml

kubectl wait pods --timeout=2m \
  -l gateway.envoyproxy.io/owning-gateway-name=envoy-ai-gateway-basic \
  -n envoy-gateway-system \
  --for=condition=Ready
```

第四步：测试配置

```
curl -H "Content-Type: application/json" -d '{
  "model": "kimi-k2-0905-preview",
  "messages": [
    {
      "role": "user",
      "content": "Hi."
    }
  ]
}' $GATEWAY_URL/v1/chat/completions
```

预期结果：

```
{"id":"chatcmpl-
68d64bc7a5422d1970e278da","object":"chat.completion","created":175887456
7,"model":"kimi-k2-0905-preview","choices":[{"index":0,"message":
{"role":"assistant","content":"Hi there! How can I help you
```

```
today?"},"finish_reason":"stop"}]], "usage":  
{"prompt_tokens":9, "completion_tokens":11, "total_tokens":20}}
```

常见问题

1. 在执行 helm 之后，安装 chart 命令超时，无法安装 chart?

推荐集群选择香港地域，防止由于网络延迟等问题导致安装超时。

2. curl 命令测试之后，返回报错：No matching route found. It is likely that the model specified your request is not configured in the Gateway

检查 yaml 文件中的模型文件是否与 curl 命令中的一致。