Ubuntu 24.04 + Docker，部署 LiteLLM v1.87.0

你当前环境是 **Ubuntu 24.04 + Docker**，部署 LiteLLM v1.87.0 最推荐使用 **Docker Compose**，比官方单行 `docker run` 更适合生产环境（持久化、数据库、配置、密钥管理）。

下面给你一个完整部署方案。

一、环境准备

安装 Docker：

```
#Ubuntu24.04 安装Docker&Compose

sudo apt update && sudo apt install -y ca-certificates curl gnupg lsb-release git

# 重新下载密钥并生成统一签名环文件
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
# 赋予可读权限
sudo chmod 644 /usr/share/keyrings/docker-archive-keyring.gpg

sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

# 开机自启
sudo systemctl enable --now docker
# 当前用户免sudo操作docker
sudo usermod -aG docker $USER
# 刷新用户组，不用重启服务器
newgrp docker

# 验证Docker安装
```

```
docker --version
docker compose version
```

配置国内 docker加速器:

```
mkdir -p /etc/docker
cat > /etc/docker/daemon.json <<EOF
{
  "registry-mirrors": [
    "https://docker.mirrors.ustc.edu.cn",
    "https://hub-mirror.c.163.com"
  ]
}
EOF
systemctl daemon-reload
systemctl restart docke
```

确认 Docker:

```
docker --version
docker compose version
```

建议:

```
Docker >= 24
Docker Compose >= 2.x
```

二、创建目录

例如:

```
mkdir -p /opt/litellm
cd /opt/litellm
```

目录:

```
/opt/litellm
├── docker-compose.yml
├── .env
└── config.yaml
```

三、创建 LiteLLM 配置

config.yaml

```
vi config.yaml
```

内容:

```
model_list:

  # OpenAI 示例
  - model_name: gpt-5
    litellm_params:
      model: openai/gpt-5
      api_key: os.environ/OPENAI_API_KEY



  # Gemini 示例
  - model_name: gemini-3.5-flash
    litellm_params:
      model: vertex_ai/gemini-3.5-flash
      api_key: os.environ/GEMINI_API_KEY



general_settings:

  master_key: os.environ/LITELLM_MASTER_KEY

  # 开启数据库存储
  store_model_in_db: true
```

四、创建环境变量

.env

```
vi .env
```

例如:

```
LITELLM_MASTER_KEY=sk-litellm-admin-123456

OPENAI_API_KEY=sk-xxxx

GEMINI_API_KEY=xxxx
```

五、 Docker Compose 部署（推荐）

创建:

```
vi docker-compose.yml
```

内容:

```
services:

  litellm:
    image: ghcr.io/berriai/litellm:1.87.0

    container_name: litellm

    restart: always
```

```
  ports:
    - "4000:4000"

  environment:

    STORE_MODEL_IN_DB: "True"

    DATABASE_URL: postgresql://litellm:litellm@postgres:5432/litellm

    LITELLM_MASTER_KEY: ${LITELLM_MASTER_KEY}

    OPENAI_API_KEY: ${OPENAI_API_KEY}

    GEMINI_API_KEY: ${GEMINI_API_KEY}


  volumes:

    - ./config.yaml:/app/config.yaml


  command:
    --config /app/config.yaml


  depends_on:
    - postgres



postgres:

  image: postgres:16

  container_name: litellm-postgres

  restart: always
```

```
    environment:
      POSTGRES_DB: litellm
      POSTGRES_USER: litellm
      POSTGRES_PASSWORD: litellm

    volumes:
      - ./postgres:/var/lib/postgresql/data
```

六、启动

执行:

```
# 通过南大ghcr代理拉取镜像
docker pull ghcr.nju.edu.cn/berriai/litellm:1.87.0
# 还原原始镜像标签，适配compose
docker tag ghcr.nju.edu.cn/berriai/litellm:1.87.0 ghcr.io/berriai/litellm:1.87.0
docker rmi ghcr.nju.edu.cn/berriai/litellm:1.87.0

docker compose up -d
```

查看:

```
docker ps
```

应该:

```
litellm
litellm-postgres
```

查看日志:

```
docker logs -f litellm
```

正常:

```
LiteLLM Proxy Server Started
Server running on port 4000
```

七、测试 API

LiteLLM 默认兼容 OpenAI API。

测试:

```
curl http://127.0.0.1:4000/v1/models \
-H "Authorization: Bearer sk-litellm-admin-123456"
```

返回:

```
{
 "data":[
   {
    "id":"gpt-5"
   }
 ]
}
```

测试聊天:

```
curl http://127.0.0.1:4000/v1/chat/completions \
-H "Authorization: Bearer sk-litellm-admin-123456" \
-H "Content-Type: application/json" \
-d '
{
 "model":"gpt-5",
 "messages":[
   {
    "role":"user",
    "content":"hello"
   }
```

```
  ]
}'
```

八、访问管理后台

LiteLLM v1.87 有 Dashboard。

访问:

```
http://服务器IP:4000/ui
```

例如:

```
http://192.168.4.21:4000/ui
```

使用:

```
Master Key:
sk-litellm-admin-123456
```

登录。

九、如果只是测试（官方方式）

不需要 PostgreSQL:

```
docker run \
-d \
--name litellm \
-p 4000:4000 \
-e STORE_MODEL_IN_DB=True \
docker.litellm.ai/berriai/litellm:1.87.0
```

但是:

缺点:

- 配置丢失
- 无数据库
- 无用户管理
- 无 Key 管理

不建议生产。

十、结合你之前的环境（Dify / RAGFlow）

你的环境里已经有：

- Dify
- RAGFlow
- Keycloak
- PostgreSQL
- Docker Compose

LiteLLM 很适合作为统一 AI Gateway：

架构：

```
                  Dify
                   |
                   |
                OpenAI API
                   |
                   v
               LiteLLM 4000
                   |
      +-------------+-------------+
      |             |             |
   OpenAI        Gemini        Ollama
      |
   Azure OpenAI
```

Dify 中配置：

模型类型：

```
OpenAI Compatible
```

Base URL:

```
http://litellm:4000/v1
```

API Key:

```
sk-litellm-admin-123456
```

即可。

十一、生产建议

如果是你的服务器环境，我建议：

```
Ubuntu 24.04
        |
Docker
        |
Nginx HTTPS
        |
LiteLLM 1.87
        |
PostgreSQL 16
        |
Keycloak OAuth
```

再加：

- Redis（限流）
- Prometheus（监控）
- Grafana（Dashboard）

你这里的：

```
LITELLM_MASTER_KEY=sk-litellm-admin-123456
```

就是 LiteLLM Dashboard 登录密码（ Master Key）。

修改方法如下。

---

1. 修改 `.env`

编辑：

```
cd /opt/litellm
vi .env
```

例如改成：

```
LITELLM_MASTER_KEY=sk-litellm-admin-9f83a7d2c1e54b6a

OPENAI_API_KEY=sk-xxxx

GEMINI_API_KEY=xxxx
```

建议生成随机值：

```
openssl rand -hex 32
```

例如：

```
a9d7f3e8b4c1d2e6f7a8b9c0d1e2f3456789abcd
```

然后：

```
LITELLM_MASTER_KEY=sk-a9d7f3e8b4c1d2e6f7a8b9c0d1e2f3456789abcd
```

---

2. 重启 LiteLLM

因为环境变量是在容器启动时读取：

```
docker compose down
```

```
docker compose up -d
```

或者:

```
docker compose restart litellm
```

如果 `.env` 被 compose 读取, 推荐:

```
docker compose up -d --force-recreate litellm
```

确保新环境变量进入容器。

---

3. 验证新 Key

查看容器:

```
docker exec litellm env | grep LITELLM_MASTER_KEY
```

应该看到:

```
LITELLM_MASTER_KEY=sk-litellm-admin-9f83a7d2c1e54b6a
```

---

4. Dashboard重新登录

访问:

```
http://服务器IP:4000/ui
```

输入:

```
Master Key:
sk-litellm-admin-9f83a7d2c1e54b6a
```

即可。

注意：不要只改 `.env` 后 restart

很多人这里容易踩坑。

例如：

```
vi .env
docker compose restart litellm
```

有时不会重新加载环境变量，因为容器已经存在。

确认：

```
docker inspect litellm | grep LITELLM_MASTER_KEY
```

如果还是旧值，需要：

```
docker compose up -d --force-recreate litellm
```

生产环境建议

你当前：

```
LITELLM_MASTER_KEY=sk-litellm-admin-123456
```

属于测试密码。

建议：

```
LITELLM_MASTER_KEY=sk-prod-$(openssl rand -hex 32)
```

并且：

- Master Key 只用于管理 Dashboard
- Dify/RAGFlow 使用单独生成的 API Key
- 不要把 Master Key 放到应用配置里

后续如果接 Dify，可以在 LiteLLM Dashboard 创建：

```
Keys → Generate Key
```

例如：

```
sk-dify-prod-xxxx
```

然后 Dify 使用这个，而不是 Master Key。

你的 LiteLLM 已经部署好了，现在添加阿里百炼（DashScope / Model Studio）的 **qwen3.7-max** 很简单。

LiteLLM 对百炼支持 OpenAI Compatible 接口，配置方式和 OpenAI 类似，只需要指定：

- `model`
- `api_base`
- `api_key`

阿里云百炼 qwen3.7-max 提供 OpenAI 兼容接口，模型 ID 为：

```
qwen3.7-max
```

对应接口地址可以使用百炼 OpenAI Compatible Endpoint。(AlibabaCloud)

---

1. 修改 .env

你现在：

```
cat /opt/litellm/.env
```

类似：

```
LITELLM_MASTER_KEY=sk-litellm-admin-123456


OPENAI_API_KEY=sk-xxxx


GEMINI_API_KEY=xxxx
```

增加：

```
DASHSCOPE_API_KEY=sk-xxxxxxxxxxxxxxxx
```

例如:

```
LITELLM_MASTER_KEY=sk-litellm-admin-123456

OPENAI_API_KEY=sk-xxxx

GEMINI_API_KEY=xxxx

DASHSCOPE_API_KEY=sk-abcd123456789
```

这个 Key 在:

阿里云百炼控制台 → API Key 管理

创建。

---

2. 修改 LiteLLM config.yaml

假设你的配置:

```
/opt/litellm/config.yaml
```

增加:

```
model_list:

  - model_name: gpt-5
    litellm_params:
      model: openai/gpt-5
      api_key: os.environ/OPENAI_API_KEY


  - model_name: gemini-3.5-flash
    litellm_params:
      model: gemini/gemini-3.5-flash
      api_key: os.environ/GEMINI_API_KEY


  # 阿里百炼 qwen3.7-max
```

```
  - model_name: qwen3.7-max
    litellm_params:
      model: openai/qwen3.7-max
      api_base: https://dashscope.aliyuncs.com/compatible-mode/v1
      api_key: os.environ/DASHSCOPE_API_KEY
```

LiteLLM 的 `model_name` 是暴露给 Dify/RAGFlow 的名字，`litellm_params.model` 是实际供应商模型。（docs.litellm.com.cn）

---

3. 重启 LiteLLM

因为 `.env` 修改了：

不要只 restart。

执行：

```
cd /opt/litellm

docker compose up -d --force-recreate litellm
```

查看：

```
docker compose logs -f litellm
```

---

4. 检查模型是否出现

执行：

```
curl http://127.0.0.1:4000/v1/models \
-H "Authorization: Bearer sk-litellm-admin-123456"
```

应该看到：

```
{
 "data":[
   {
```

```
      "id":"gpt-5"
    },
    {
      "id":"gemini-3.5-flash"
    },
    {
      "id":"qwen3.7-max"
    }
  ]
}
```

5. 测试 qwen3.7-max

```
curl http://127.0.0.1:4000/v1/chat/completions \
-H "Authorization: Bearer sk-litellm-admin-123456" \
-H "Content-Type: application/json" \
-d '
{
 "model":"qwen3.7-max",
 "messages":[
    {
     "role":"user",
     "content":"介绍一下阿里百炼"
    }
 ]
}'
```

成功返回:

```
{
 "choices":[
    {
     "message":{
       "content":"..."
     }
    }
 ]
```

```
}
```

---

6. 给 Dify 使用

你现在 Dify 调 LiteLLM，建议暴露统一名字：

例如：

```
- model_name: qwen-max
  litellm_params:
    model: openai/qwen3.7-max
    api_base: https://dashscope.aliyuncs.com/compatible-mode/v1
    api_key: os.environ/DASHSCOPE_API_KEY
```

以后 Dify 只配置：

```
qwen-max
```

以后换：

```
qwen3.7-max
↓
qwen4-max
↓
DeepSeek
```

Dify 不需要改。

---

7. 你的最终架构会变成

```
            Dify
              |
              |
      OpenAI API格式
              |
              |
```

```
            LiteLLM 1.87
                  |
      +-----------+------------+
      |           |            |
   GPT-5     Gemini       Qwen3.7-Max
   OpenAI    Google       阿里百炼
```

这正适合你之前的 Dify + RAGFlow + 企业知识库架构。

另外注意：你之前日志里 `/v1/responses` 是 OpenAI Responses API，如果后续用 Codex/Agent 类客户端调用 qwen3.7-max，需要确认该模型端点对 Responses API 的兼容性；Dify 通常走 `/v1/chat/completions`，问题会少一些。

Revision #5
Created 2026-07-21 09:36:37 UTC by Admin
Updated 2026-07-21 09:53:36 UTC by Admin