

AI Agent 101 环境搭建完整指南

Ubuntu 22.04.4
Python 3.10.18
Conda Latest
JupyterLab Latest

🚀 从零开始搭建 AI Agent 开发环境的完整指南

[快速开始](#) · [详细步骤](#) · [故障排除](#)

目录

- [快速开始](#)
- [系统要求](#)
- [详细安装步骤](#)
 - [步骤1：虚拟机环境搭建](#)
 - [步骤2：Ubuntu 22.04.4 系统安装](#)
 - [步骤3：Python 和 Conda 环境安装](#)
 - [步骤4：v2raya 科学上网工具](#)
 - [步骤5：JupyterLab 安装与配置](#)
 - [步骤6：项目代码下载](#)
 - [步骤7：环境变量配置](#)
 - [步骤8：JupyterLab 服务管理](#)
 - [步骤9：验证环境](#)
- [故障排除](#)
- [常见问题](#)

🎯 快速开始

方式一：使用预配置虚拟机（推荐新手）

最快速的方式！已配置好所有环境，开箱即用！

```
# 1. 下载虚拟机镜像（OVF 格式）【待修改】  
链接: https://pan.baidu.com/s/1\_uipki1CBEes3xrVb29wzQ 提取码: 1234
```

```
# 2. 导入虚拟机  
- VMware: 文件 → 打开 → 选择 .ovf 文件  
- VirtualBox: 管理 → 导入虚拟设备 → 选择 .ovf 文件  
导入后，就可以启动虚拟机
```

```
# 3. 连接虚拟机
```

```

SSH工具: https://www.xshell.com/zh/free-for-home-school/
地址: 192.168.172.128
端口: 22
用户名: root
密码: fly123

# 4. 配置环境变量
## 全局环境变量配置
vim ~/.bashrc
### 修改AI Agent 101 环境变量配置
## 应用配置:
source ~/.bashrc

## JupyterLab 环境变量
vim /Agent101/app/jupyter/start_jupyter.sh
### 修改AI Agent 101 环境变量配置
## 应用配置:
sudo systemctl daemon-reload
sudo systemctl restart jupyter.service

# 5. 配置科学上网
科学上网订阅地址: https://yundong.xn--xhq8sm16c51s.com/#/register?code=RQKCnEwf
浏览器打开, 注册后, 复制订阅地址。
里面有文档哈, 不能微信多说哈
## 登录V2RayA服务, 添加订阅地址
地址: http://192.168.172.128:2017/
用户名: root
密码: fly123

# 6. 验证环境
conda activate agent101
jupyter lab --version

# 7. 更新代码
cd /Agent101/code/Agent_In_Action
git pull https://github.com/FlyAIBox/Agent_In_Action

# 8. 访问JupyterLab
地址: http://192.168.172.128:8000/
备注: 确保JupyterLab服务正常, 可以通过systemctl status jupyter.service确认

```

预配置虚拟机包含:

- Ubuntu 22.04.4 系统
- Python 3.10.18 + Conda 环境
- JupyterLab (已配置 systemd 服务)
- v2rayA 科学上网工具
- Docker + Docker Compose
- Node.js 22.14.0 (通过 NVM)
- 所有常用开发工具

方式二：手动安装（推荐进阶用户）

如果你想深入理解每个安装步骤，或者需要自定义配置，请按照下面的[详细安装步骤](#)进行操作。

系统要求

基础配置（项目1-5：API调用模式）

| 组件 | 推荐配置 | 说明 |
|--------|------------------|------------|
| 操作系统 | Ubuntu 22.04 LTS | 长期支持版本 |
| Python | 3.10.x | 项目推荐版本 |
| CPU | 4 核心 | 最低 2 核心 |
| 内存 | 8 GB+ | 最低 4 GB |
| 存储 | 100 GB+ | 建议 SSD |
| GPU | 非必需 | 如没有，可使用云厂商 |

高级配置（项目6：模型微调）

| 组件 | 要求 | 说明 |
|-----|------------|---------------|
| GPU | NVIDIA GPU | 支持 CUDA 12.8+ |
| 显存 | 80GB+ | LoRA 微调最低要求 |
| 内存 | 32GB+ | 推荐配置 |
| 存储 | 200GB+ | 模型和数据存储 |

详细安装步骤

步骤1：虚拟机环境搭建（可选）

如果你已有 Ubuntu 22.04 系统，可以跳过此步骤。

1.1 选择虚拟机软件

方案A：VMware Workstation Pro（推荐）

优势：性能好、稳定性高、功能强大

获取方式：

- 官方下载：[Broadcom 支持门户](#)（需要注册）
- 百度网盘：<https://pan.baidu.com/s/12YvAKoVq9Fpz5oUZKKBKbW> 提取码: 1234

安装步骤：

1. 下载安装包

- 文件名: `VMware-workstation-full-17.x.x-xxxxx.exe` (Windows)
- 文件名: `VMware-Workstation-Full-17.x.x-xxxxx.x86_64.bundle` (Linux)

2. 安装 VMware

```
# Windows: 双击安装包, 按提示操作  
# Linux:  
sudo chmod +x VMware-Workstation-Full-*.bundle  
sudo ./VMware-Workstation-Full-*.bundle
```

3. 激活许可证

- 个人使用: 选择"个人用途" (免费)
- 商业使用: 需要购买许可证

方案B：VirtualBox (免费替代方案)

优势: 开源免费、跨平台

重要提示: 请使用 VirtualBox 7.0 版本 (非 7.1) , 7.1 版本在 Windows 下存在网络性能问题。

安装步骤：

1. 下载 VirtualBox 7.0

- 官网: https://www.virtualbox.org/wiki/Download_Old_Builds_7.0
- 下载: VirtualBox 7.0.x platform packages
- 下载: VirtualBox 7.0.x Extension Pack

2. 安装 VirtualBox

- 双击安装包, 按提示完成安装
- 双击 Extension Pack, 在 VirtualBox 中安装增强功能

3. 修改默认安装路径 (可选)

- ! 不要选择已存在的目录
- 直接在路径栏修改, 如: `D:\Oracle\VirtualBox`

1.2 下载 Ubuntu 22.04.4 系统镜像

推荐镜像站: 清华大学开源软件镜像站

1. 访问镜像站

```
https://mirrors.tuna.tsinghua.edu.cn/
```

2. 下载镜像

- 点击"获取下载链接"
- 选择: Ubuntu

- 版本: 22.04.5 (amd64, Desktop LiveDVD)
- 文件名: `ubuntu-22.04.5-desktop-amd64.iso`
- 大小: 约 5.7 GB

备用镜像站:

- 阿里云: <https://mirrors.aliyun.com/ubuntu-releases/>
- 网易: <https://mirrors.163.com/ubuntu-releases/>
- 中科大: <https://mirrors.ustc.edu.cn/ubuntu-releases/>

1.3 创建虚拟机

VMware 创建虚拟机

1. 新建虚拟机

- 打开 VMware Workstation
- 文件 → 新建虚拟机
- 选择"典型 (推荐) "

2. 选择安装源

- 选择"安装程序光盘映像文件(iso)"
- 浏览并选择下载的 Ubuntu iso 文件

3. 配置虚拟机信息

- 虚拟机名称: `Ubuntu_Agent101`
- 位置: 选择存储路径 (建议使用 SSD)

4. 配置硬件

- 处理器: 4 核心 (最低 2 核心)
- 内存: 8 GB (最低 4 GB)
- 硬盘: 100 GB (建议 150 GB)
- 网络: NAT 模式

5. 完成创建

- 点击"完成"
- **⚠** 暂不启动虚拟机, 先调整配置

6. 调整虚拟机设置 (可选)

- 编辑虚拟机设置
- 显示器: 分配更多显存
- USB: USB 3.1
- 声音: 根据需要启用/禁用

VirtualBox 创建虚拟机

1. 新建虚拟机

- 打开 VirtualBox
- 工具 → 新建(N)

2. 基本信息

- 名称: Ubuntu_Agent101
- 文件夹: 选择存储路径
- 类型: Linux
- 版本: Ubuntu (64-bit)
- ISO 映像: 选择下载的 Ubuntu iso 文件
- 勾选"跳过自动安装"

3. 硬件配置

- 内存: 8192 MB (8 GB)
- 处理器: 4 CPU

4. 虚拟硬盘

- 虚拟硬盘大小: 100 GB (建议 150 GB)
- 硬盘文件类型: VDI (VirtualBox 磁盘映像)
- 存储在物理硬盘上: 动态分配

5. 完成创建

- 点击"完成"

6. 调整虚拟机设置

- 选择虚拟机 → 设置
- 常规 → 高级:
 - 共享粘贴板: 双向
 - 拖放: 双向
- 显示:
 - 显存: 128 MB
 - 缩放因子: 100%
- 网络 → 网卡 1:
 - 连接方式: 网络地址转换(NAT)
 - 控制芯片: 准虚拟化网络(virtio-net)
 - 点击"端口转发"
- 端口转发规则:

名称: SSH
协议: TCP
主机端口: 9090
子系统端口: 22

步骤2: Ubuntu 22.04.4 系统安装

2.1 启动虚拟机安装系统

1. 启动虚拟机

- 选择创建的虚拟机
- 点击"启动"或"开启此虚拟机"

2. 进入安装引导

- 等待启动画面
- 选择第一项: Try or Install Ubuntu
- 按 Enter 键

3. 鼠标控制提示

- 如果鼠标被虚拟机捕获, 按右 `ctrl` 键 (VirtualBox) 或 `ctrl+Alt` (VMware) 释放

2.2 安装 Ubuntu 系统

1. 选择语言

- 建议选择 English (安装后可切换中文)
- 原因: 避免中文路径导致的兼容性问题

2. 开始安装

- 点击 Install Ubuntu

3. 键盘布局

- Layout: English (US)
- 点击 Continue

4. 更新和其他软件

- 安装类型: Normal installation
- Download updates while installing Ubuntu
- Install third-party software for graphics and Wi-Fi hardware
- 点击 Continue

5. 安装类型

- 选择 Erase disk and install Ubuntu
-  这不会影响宿主机, 只会格式化虚拟硬盘
- 点击 Install Now
- 确认 → Continue

6. 选择时区

- 地图上点击中国
- 或选择: Shanghai
- 点击 Continue

7. 创建用户

- Your name: AI-Agent101 (或自定义)
- Your computer's name: ubuntu-agent (或自定义)
- Pick a username: agent (或自定义)
- Choose a password: 设置强密码并记住
- Confirm your password: 再次输入密码
- Log in automatically (可选, 方便但不够安全)
- 点击 Continue

8. 等待安装

- 安装过程约 10-20 分钟
- 可以查看 Ubuntu 介绍幻灯片

9. 安装完成

- 出现"Installation Complete"提示
- 点击 Restart Now
- 如提示"Please remove the installation medium, then press ENTER"
- 直接按 Enter 键

10. 首次启动

- 输入密码登录
- **⚠ 忽略所有系统弹窗** (如软件更新提示)
- 点击 Skip 跳过所有向导

2.3 安装增强功能 (VirtualBox)

VMware 通常会自动安装 VMware Tools, 如未自动安装, 选择"虚拟机 → 安装 VMware Tools"

VirtualBox 增强功能安装:

1. 插入增强功能光盘

- 虚拟机菜单: 设备 → 安装增强功能
- 桌面会出现 VBOX_GAS 光盘图标

2. 运行安装脚本

- 打开文件管理器
- 点击左侧 VBOX_GAS
- 找到 autorun.sh 文件
- 右键 → Run as a Program

- 等待安装完成

3. 重启系统

```
sudo reboot
```

2.4 系统初始化

登录系统后，打开终端 (`Ctrl+Alt+T`)：

```
# 更新系统包  
sudo apt update && sudo apt upgrade -y  
  
# 安装必要的系统工具  
sudo apt install -y wget curl git vim net-tools \  
    build-essential software-properties-common \  
    htop tree ssh openssh-server  
  
# 启用 SSH 服务  
sudo systemctl enable ssh  
sudo systemctl start ssh
```

2.5 系统语言和输入法配置（可选）

如需中文环境：

1. 设置中文语言

- 右上角 → Settings
- Region & Language
- Language: 点击 `+` → 选择 `汉语 (中国)`
- 点击 `Manage Installed Languages`
- 点击 `Install`
- 重启系统

2. 保持英文目录名

- 重启后会提示更新文件夹名称
- 选择： `Keep old Names` (保持英文目录)

3. 安装中文输入法（可选）

```
sudo apt install ibus-pinyin  
# 或安装搜狗输入法
```

2.6 创建系统快照（重要）

VirtualBox：

1. 关闭虚拟机
2. 选择虚拟机 → 备份[系统快照]

3. 点击"生成" → 命名为"初始系统"

VMware:

1. 虚拟机 → 快照 → 拍摄快照

2. 命名为"初始系统"

步骤3：Python 和 Conda 环境安装

3.1 安装 Miniconda

Miniconda 是一个轻量级的 Conda 发行版，包含 Conda、Python 和一些基本工具。

1. 下载 Miniconda 安装包

```
# 方式1: 官方地址 (国外服务器, 可能较慢)
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

# 方式2: 清华镜像 (国内推荐)
wget https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/Miniconda3-latest-
Linux-x86_64.sh
```

2. 安装 Miniconda

```
# 静默安装到默认位置 $HOME/miniconda3
bash Miniconda3-latest-Linux-x86_64.sh -b -p $HOME/miniconda3
```

参数说明：

- `-b`: 批处理模式，无需手动确认
- `-p`: 指定安装路径

3. 初始化 Conda

```
# 初始化 bash shell
$HOME/miniconda3/bin/conda init bash

# 重新加载 shell 配置
source ~/.bashrc
```

4. 验证 Conda 安装

```
# 查看 conda 版本
conda --version
# 输出示例: conda 24.1.2

# 查看 conda 信息
conda info
```

3.2 配置 Conda 镜像源（可选但推荐）

使用国内镜像可以大幅提升下载速度：

```
# 添加清华镜像源
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/pro
conda config --add channels
https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2

# 设置搜索时显示通道地址
conda config --set show_channel_urls yes

# 查看配置
conda config --show channels
```

配置文件位置：`~/.condarc`

3.3 创建项目专用环境

```
# 创建名为 agent101 的环境，指定 Python 3.10.18
conda create -n agent101 python=3.10.18 -y

# 激活环境
conda activate agent101

# 验证 Python 版本
python --version
# 输出：Python 3.10.18

# 验证 pip 版本
pip --version
```

环境管理常用命令：

```
# 查看所有环境
conda env list

# 激活环境
conda activate agent101

# 退出环境
conda deactivate

# 删除环境（慎用）
conda remove -n agent101 --all

# 导出环境配置
conda env export > environment.yml
```

```
# 从配置文件创建环境  
conda env create -f environment.yml
```

3.4 配置 pip 镜像源

```
# 配置清华镜像  
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple  
  
# 验证配置  
pip config list  
  
# 查看配置文件位置  
pip config list -v
```

其他国内镜像源：

```
# 阿里云  
pip config set global.index-url https://mirrors.aliyun.com/pypi/simple/  
  
# 中科大  
pip config set global.index-url https://pypi.mirrors.ustc.edu.cn/simple/  
  
# 豆瓣  
pip config set global.index-url https://pypi.douban.com/simple/
```

3.5 Conda 环境优化配置

编辑 `~/.bashrc`，添加以下配置：

```
# Conda 自动激活配置  
# 取消默认激活 base 环境（可选）  
# conda config --set auto_activate_base false  
  
# 添加 conda 初始化脚本（如果还没有）  
# >>> conda initialize >>>  
# !! Contents within this block are managed by 'conda init' !!  
__conda_setup="$(('/root/miniconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"  
if [ $? -eq 0 ]; then  
    eval "$__conda_setup"  
else  
    if [ -f "/root/miniconda3/etc/profile.d/conda.sh" ]; then  
        . "/root/miniconda3/etc/profile.d/conda.sh"  
    else  
        export PATH="/root/miniconda3/bin:$PATH"  
    fi  
fi  
unset __conda_setup  
# <<< conda initialize <<<
```

应用配置：

```
source ~/.bashrc
```

步骤4：v2raya 科学上网工具（可选）

⚠ 法律提醒：请遵守当地法律法规，仅用于学习和技术研究。

4.1 安装 v2raya

1. 添加 v2raya 软件源

```
# 添加公钥
wget -qO - https://apt.v2raya.org/key/public-key.asc | sudo tee
/etc/apt/keyrings/v2raya.asc

# 添加软件源
echo "deb [signed-by=/etc/apt/keyrings/v2raya.asc] https://apt.v2raya.org/
v2raya main" | \
sudo tee /etc/apt/sources.list.d/v2raya.list

# 更新软件包列表
sudo apt update
```

2. 安装 v2raya 和 v2ray 内核

```
sudo apt install -y v2raya v2ray
```

3. 启动 v2raya 服务

```
# 启动服务
sudo systemctl start v2raya.service

# 设置开机自启
sudo systemctl enable v2raya.service

# 查看服务状态
sudo systemctl status v2raya.service
```

4.2 配置 v2raya

1. 访问 Web 界面

```
http://localhost:2017
# 或使用虚拟机 IP
http://虚拟机IP:2017
```

2. 首次配置

- 创建管理员账号和密码
- 设置并记住登录凭据

3. 导入节点

Pay Science 上网订阅服务

<https://yundong.xn--xhq8sm16c5ls.com/#/register?code=RQKCnEWf>

支持多种导入方式：

方式1：导入订阅链接

- 点击"导入" → "从订阅链接导入"
- 粘贴订阅地址
- 点击"导入"

方式2：手动添加节点

- 点击"创建"
- 输入节点信息（服务器地址、端口、加密方式等）

方式3：扫描二维码

- 使用手机端 v2rayN 等工具生成二维码
- 在 v2raya 界面扫描

4. 启动代理

- 选择一个节点
- 点击"启动"
- 选择代理模式：
 - **规则代理**: 按规则自动分流（推荐）
 - **全局代理**: 所有流量走代理
 - **直连模式**: 不使用代理

5. 测试连接

```
# 测试国外网站连接
curl -I https://www.google.com

# 测试延迟
ping 8.8.8.8
```

4.3 配置系统代理（可选）

如果需要在终端使用代理：

```
# 编辑 ~/.bashrc
vim ~/.bashrc

# 添加代理配置
export http_proxy="http://127.0.0.1:20171"
export https_proxy="http://127.0.0.1:20171"
export no_proxy="localhost,127.0.0.1,192.168.0.0/16,10.0.0.0/8"

# 应用配置
source ~/.bashrc
```

临时使用代理：

```
# 仅当前终端会话有效
export http_proxy="http://127.0.0.1:20171"
export https_proxy="http://127.0.0.1:20171"

# 取消代理
unset http_proxy
unset https_proxy
```

4.4 v2raya 常用操作

```
# 查看服务状态
sudo systemctl status v2raya

# 启动服务
sudo systemctl start v2raya

# 停止服务
sudo systemctl stop v2raya

# 重启服务
sudo systemctl restart v2raya

# 查看日志
sudo journalctl -u v2raya -f
```

步骤5：JupyterLab 安装与配置

5.1 安装 JupyterLab

确保已激活 agent101 环境：

```
# 激活环境  
conda activate agent101  
  
# 使用 conda 安装 JupyterLab  
conda install -c conda-forge jupyterlab -y  
  
# 验证安装  
jupyter lab --version  
# 输出示例: 4.0.9
```

5.2 生成配置文件

```
# 生成 JupyterLab 配置文件  
jupyter lab --generate-config  
  
# 输出示例:  
# writing default config to: /root/.jupyter/jupyter_lab_config.py
```

配置文件位置: `~/.jupyter/jupyter_lab_config.py`

5.3 配置 JupyterLab

编辑配置文件:

```
vim ~/.jupyter/jupyter_lab_config.py
```

添加或修改以下配置:

```
# 允许 root 用户启动 (如果使用 root 用户)  
c.ServerApp.allow_root = True  
  
# 设置工作目录  
c.ServerApp.root_dir = '/Agent101/code'  
  
# 修改默认端口  
c.ServerApp.port = 8000  
  
# 设置可访问的 IP (0.0.0.0 表示允许所有 IP 访问)  
c.ServerApp.ip = '0.0.0.0'  
  
# 设置访问令牌 (token) 作为密码  
# 方式1: 自定义 token  
c.ServerApp.token = 'fly123'  
  
# 方式2: 禁用 token (不推荐, 仅本地使用)  
# c.ServerApp.token = ''  
# c.ServerApp.password = ''  
  
# 禁用浏览器自动打开  
c.ServerApp.open_browser = False
```

```
# 启用扩展  
c.ServerApp.jpserver_extensions = {}
```

重要配置说明：

| 配置项 | 说明 | 推荐值 |
|--------------|-----------------|------------------|
| allow_root | 是否允许 root 用户运行 | True (如使用 root) |
| root_dir | JupyterLab 工作目录 | /Agent101/code |
| port | 访问端口 | 8000 |
| ip | 监听 IP | 0.0.0.0 (允许远程访问) |
| token | 访问令牌 | 自定义强密码 |
| open_browser | 自动打开浏览器 | False (服务器模式) |

5.4 安装 ipykernel (重要)

为了在 JupyterLab 中使用 conda 环境，必须安装 ipykernel：

```
# 确保在 agent101 环境中  
conda activate agent101  
  
# 安装 ipykernel  
pip install ipykernel  
  
# 将环境注册为 Jupyter 内核  
python -m ipykernel install --user --name=agent101 --display-name="Python  
(agent101)"  
  
# 验证内核安装  
jupyter kernelspec list
```

输出示例：

```
Available kernels:  
agent101    /root/.local/share/jupyter/kernels/agent101  
python3      /root/miniconda3/envs/agent101/share/jupyter/kernels/python3
```

管理内核：

```
# 查看已安装的内核
jupyter kernelspec list

# 删除内核
jupyter kernelspec remove agent101

# 重新安装内核
python -m ipykernel install --user --name=agent101 --display-name="Python
(agent101)"
```

5.5 测试 JupyterLab

```
# 启动 JupyterLab (前台运行)
jupyter lab --config=/root/.jupyter/jupyter_lab_config.py

# 输出示例:
# [I 2025-01-01 10:00:00.000 ServerApp] Jupyter Server 2.x.x is running at:
# [I 2025-01-01 10:00:00.000 ServerApp] http://0.0.0.0:8000/lab?token=fly123
```

访问 JupyterLab:

```
http://localhost:8000
# 或使用虚拟机 IP
http://虚拟机IP:8000
```

输入 token: fly123

停止 JupyterLab: 按 `Ctrl+C` 两次

步骤6：项目代码下载

6.1 创建代码目录

```
# 创建代码根目录
sudo mkdir -p /Agent101/code

# 修改目录所有者为当前用户
sudo chown -R $USER:$USER /Agent101/code

# 进入代码目录
cd /Agent101/code
```

6.2 克隆项目代码

```
# 克隆 Agent_In_Action 项目
git clone https://github.com/FlyAIBox/Agent_In_Action.git

# 进入项目目录
cd Agent_In_Action

# 查看项目结构
tree -L 2

# 或使用 ls
ls -lah
```

项目结构：

```
Agent_In_Action/
├── 01-agent-tool-mcp/          # 项目1-2：智能体基础与 MCP 集成
├── 02-agent-multi-role/       # 项目3：深度研究助手
├── 03-agent-build-docker-deploy/ # 项目4：旅行规划系统
├── 04-agent-evaluation/        # 项目5：监控与评估
├── 05-agent-model-finetuning/   # 项目6：模型微调
├── docs/                      # 文档目录
└── README.md                  # 项目说明
```

6.3 安装项目依赖

6.3.1 按项目安装

```
# 确保在 agent101 环境中
conda activate agent101

# 项目1-2：智能体基础与 MCP
cd 01-agent-tool-mcp/mcp-demo
pip install -r requirements.txt

# 项目3：深度研究助手
cd ../../02-agent-multi-role/deepresearch/deployment
pip install -r requirements.txt

# 项目4：旅行规划系统
cd ../../..../03-agent-build-docker-deploy
pip install -r backend/requirements.txt
pip install -r frontend/requirements.txt

# 项目5：评估监控
pip install langfuse langchain langgraph

# 返回项目根目录
cd /Agent101/code/Agent_In_Action
```

6.3.2 常用依赖包

项目所需的主要依赖包：

```
# 核心依赖
pip install langchain langgraph langchain-openai langchain-community
pip install openai anthropic
pip install fastapi uvicorn streamlit
pip install python-dotenv

# 工具包
pip install tavily-python serpapi
pip install requests httpx

# 监控评估
pip install langfuse langsmith

# 数据处理
pip install pandas numpy
```

6.4 验证安装

```
# 验证 Python 包
python -c "import langchain; print(f'LangChain: {langchain.__version__}')"
python -c "import langgraph; print('LangGraph installed')"
python -c "import openai; print(f'OpenAI: {openai.__version__}')"

# 查看已安装的包
pip list | grep -E "langchain|openai|fastapi"
```

6.5 备注：Ubuntu22.04 系统配置Git 用户名和密钥

1. 配置 Git 用户名和邮箱

这是最基本的配置，当你提交代码时，Git 会记录下你的用户名和邮箱。

打开终端并执行以下命令：

```
git config --global user.name "你的用户名"
git config --global user.email "你的邮箱地址"
```

将 "你的用户名" 替换为你希望在 Git 提交中显示的名字，将 "你的邮箱地址" 替换为你的常用邮箱。

你可以通过以下命令查看是否配置成功：

```
git config --global user.name
git config --global user.email
```

2. 配置 SSH 密钥（推荐用于与远程仓库交互）

使用 SSH 密钥可以让你在与 GitHub、GitLab、Gitee 等远程仓库交互时，无需每次都输入密码，更加方便和安全。

步骤 2.1: 检查是否已存在 SSH 密钥

在终端中输入以下命令，查看 `~/.ssh` 目录下是否有 `id_rsa` 或 `id_ed25519` 等文件：

```
ls -al ~/.ssh
```

如果看到类似 `id_rsa` (或 `id_ed25519`) 和 `id_rsa.pub` (或 `id_ed25519.pub`) 的文件，说明你可能已经生成过 SSH 密钥，可以直接跳到 **步骤 2.3: 将 SSH 公钥**添加到你的 Git 账户****。

步骤 2.2: 生成新的 SSH 密钥

如果 `~/.ssh` 目录下没有密钥文件，你需要生成一个新的。推荐使用 `ed25519` 算法，如果不支持，可以使用 `rsa`。

- **生成 Ed25519 密钥 (推荐):**

```
ssh-keygen -t ed25519 -c "你的邮箱地址"
```

- 将 `"你的邮箱地址"` 替换为你之前配置的 Git 邮箱地址。
- 当提示 "Enter file in which to save the key" 时，可以直接按回车键，使用默认文件名 `id_ed25519`。
- 然后会提示你输入密码 (passphrase)，这是为了增加密钥的安全性。你可以选择设置密码，也可以直接按两次回车键留空。
- **生成 RSA 密钥 (如果不支持 Ed25519):**

```
ssh-keygen -t rsa -b 4096 -c "你的邮箱地址"
```

- 同样，将 `"你的邮箱地址"` 替换为你的 Git 邮箱地址。
- 按照提示操作，可以选择文件名和密码。

步骤 2.3: 将 SSH 公钥**添加到你的 Git 账户**

你需要将生成的公钥（以 `.pub` 结尾的文件内容）添加到你的远程 Git 仓库账户中（例如 GitHub、GitLab、Gitee 等）。

- **复制 SSH 公钥**到剪贴板:****
- 根据你生成的密钥类型，执行以下命令：

- **Ed25519:**

- `cat ~/.ssh/id_ed25519.pub | xclip -selection clipboard`

- (如果你的系统没有 `xclip`，可以使用 `sudo apt install xclip` 安装)

- **RSA**:**

- `cat ~/.ssh/id_rsa.pub | xclip -selection clipboard`

- (同样, 如果没有 `xclip`, 请安装)
 - 或者, 你也可以使用文本编辑器打开 `.pub` 文件, 手动复制里面的内容。
- 将公钥添加到你的 Git 账户:
- **GitHub:** 登录 GitHub 网站, 点击右上角的头像, 选择 "Settings"。在左侧菜单中选择 "SSH and GPG keys", 然后点击 "New SSH key" 或 "Add SSH key"。将复制的公钥粘贴到 "Key" 文本框中, 并为这个密钥添加一个描述, 然后点击 "Add SSH key"。
 - **GitLab:** 登录 GitLab 网站, 点击右上角的头像, 选择 "Edit profile"。在左侧菜单中选择 "SSH Keys"。将复制的公钥粘贴到 "Key" 文本框中, 可以添加一个 "Title" 来描述这个密钥, 然后点击 "Add key"。
 - **Gitee (码云):** 登录 Gitee 网站, 点击右上角的头像, 选择 "设置"。在左侧菜单中选择 "SSH 公钥", 然后点击 "添加公钥"。将复制的公钥粘贴到 "公钥" 文本框中, 可以设置一个 "标题", 然后点击 "确定"。
- 不同的 Git 服务提供商的操作可能略有不同, 但大致步骤类似。

步骤 2.4: 测试 SSH 连接

添加公钥后, 你可以测试一下 SSH 连接是否成功。

- **GitHub**:**

```
ssh -T git@github.com
```

- **GitLab:**

```
ssh -T git@gitlab.com
```

- **Gitee:**

```
ssh -T git@gitee.com
```

如果看到类似 "Hi [Your Username]! You've successfully authenticated..." 的消息, 说明 SSH 连接配置成功。

完成以上步骤后, 你就成功地在 Ubuntu 22.04 上配置了 Git 的用户名、邮箱和 SSH 密钥。以后在进行 Git 操作时, 你将能够方便地提交代码并与远程仓库进行交互。

步骤7: 环境变量配置

7.1 配置方式选择

有两种配置方式:

1. **全局配置:** 添加到 `~/.bashrc` (推荐, 所有终端会话生效)
2. **项目配置:** 在项目目录创建 `.env` 文件 (项目独立配置)

7.2 全局环境变量配置

编辑 `~/.bashrc` 文件:

```
vim ~/.bashrc
```

在文件末尾添加以下内容:

```
# =====
# AI Agent 101 环境变量配置
# =====

# --- OpenAI 配置 ---
export OPENAI_BASE_URL="https://api.openai.com/v1"
export OPENAI_API_KEY="sk-your_openai_api_key_here"
export MODEL_NAME="gpt-4o"

# 或使用 DeepSeek 作为 OpenAI 兼容接口
# export OPENAI_BASE_URL="https://api.deepseek.com"
# export OPENAI_API_KEY="sk-your_deepseek_api_key_here"
# export MODEL_NAME="deepseek-chat"

# --- DeepSeek 配置 ---
export DEEPSEEK_BASE_URL="https://api.deepseek.com"
export DEEPSEEK_API_KEY="sk-your_deepseek_api_key_here"

# --- Langfuse 配置 (监控评估) ---
export LANGFUSE_HOST="https://cloud.langfuse.com"
export LANGFUSE_PUBLIC_KEY="pk-1f-your_public_key_here"
export LANGFUSE_SECRET_KEY="sk-1f-your_secret_key_here"

# --- 搜索工具配置 ---
export SERPAPI_API_KEY="your_serpapi_key_here"
export TAVILY_API_KEY="tvly-your_tavily_key_here"

# --- LangSmith 配置 (可选) ---
export LANGSMITH_API_KEY="1sv2_your_langsmith_key_here"
export LANGSMITH_PROJECT="aiagent101"
export LANGSMITH_TRACING_V2="true"

# --- 和风天气 API 配置 ---
# 参考: https://dev.qweather.com/
export QWEATHER_API_BASE="your_qweather_base_url"
export QWEATHER_API_KEY="your_qweather_key_here"

# =====
# 代理配置 (如果使用 v2raya)
# =====
# export http_proxy="http://127.0.0.1:20171"
# export https_proxy="http://127.0.0.1:20171"
# export no_proxy="localhost,127.0.0.1,192.168.0.0/16,10.0.0.0/8"
```

应用配置：

```
source ~/.bashrc
```

7.3 API 密钥获取指南

| API 服务 | 获取地址 | 用途 | 费用 | 备注 |
|------------------|---|------------------|---------------|---|
| OpenAI | platform.openai.com | GPT-4/GPT-4o 模型 | 按量付费 | 需要信用卡 |
| OpenAI等国外大模型国内代理 | https://api.apiyi.com/regis?aff_code=we80 | GPT/Claude 等 模型 | 按量付费 | 新用户注册送0.1美金，注册成功后在以下表格中填写你的账号，平台会再赠送2美金（5个工作日到账） ---- 【腾讯文档】 API易账号收集 https://docs.qq.com/form/page/DQm1qb1VBQU9wR2xq |
| DeepSeek | platform.deepseek.com | DeepSeek-Chat/V3 | 价格实惠 | 支持国内支付 |
| 和风天气 | dev.qweather.com | 天气查询工具 | 免费额度 1000 次/天 | 需要实名认证 |
| Tavily | tavily.com | AI 搜索 API | 免费 1000次/月 | 邮箱注册即可 |
| SerpAPI | serpapi.com | Google 搜索 API | 免费 100次/月 | 需要信用卡验证 |
| Langfuse | cloud.langfuse.com | LLM 监控评估 | 免费版 5万 events | GitHub 登录 |
| LangSmith | smith.langchain.com | 调试追踪 | 免费 5 千 traces | GitHub 登录 |

7.4 项目级 .env 配置 (可选)

如果需要项目独立配置，可以在项目目录创建 `.env` 文件：

```
# 进入项目目录
cd /Agent101/code/Agent_In_Action/03-agent-build-docker-deploy/backend

# 创建 .env 文件
vim .env
```

`.env` 文件内容：

```
# OpenAI 配置
OPENAI_API_KEY=sk-your_key_here
OPENAI_BASE_URL=https://api.openai.com/v1

# 其他配置...
```

在 Python 代码中加载：

```
from dotenv import load_dotenv
import os

# 加载 .env 文件
load_dotenv()

# 读取环境变量
api_key = os.getenv("OPENAI_API_KEY")
```

7.5 验证环境变量

```
# 查看环境变量
echo $OPENAI_API_KEY
echo $DEEPSEEK_API_KEY
echo $LANGFUSE_PUBLIC_KEY

# 在 Python 中验证
python << EOF
import os
print("OpenAI API Key:", os.getenv("OPENAI_API_KEY")[:20] + "...")
print("DeepSeek API Key:", os.getenv("DEEPSEEK_API_KEY")[:20] + "...")
print("Langfuse Public Key:", os.getenv("LANGFUSE_PUBLIC_KEY")[:20] + "...")
EOF
```

步骤8：JupyterLab 服务管理

8.1 创建 JupyterLab 启动脚本

创建专用启动脚本，解决 Conda 环境问题：

```
# 创建脚本目录  
sudo mkdir -p /Agent101/app/jupyter  
  
# 创建启动脚本  
sudo vim /Agent101/app/jupyter/start_jupyter.sh
```

脚本内容：

```
#!/bin/bash  
  
# =====  
# JupyterLab 启动脚本  
# =====  
  
# --- API 环境变量配置 ---  
# 注意：将下面的占位符替换为实际的 API Key  
  
export OPENAI_BASE_URL="https://api.openai.com/v1"  
export OPENAI_API_KEY="sk-your_openai_api_key_here"  
export MODEL_NAME="gpt-4o"  
  
export DEEPSEEK_BASE_URL="https://api.deepseek.com"  
export DEEPSEEK_API_KEY="sk-your_deepseek_api_key_here"  
  
export LANGFUSE_HOST="https://cloud.langfuse.com"  
export LANGFUSE_PUBLIC_KEY="pk-1f-your_public_key_here"  
export LANGFUSE_SECRET_KEY="sk-1f-your_secret_key_here"  
  
export SERPAPI_API_KEY="your_serpapi_key_here"  
export TAVILY_API_KEY="tvly-your_tavily_key_here"  
  
export LANGSMITH_API_KEY="1sv2_your_langsmith_key_here"  
export LANGSMITH_PROJECT="aiagent101"  
  
export QWEATHER_API_BASE="your_qweather_base_url"  
export QWEATHER_API_KEY="your_qweather_key_here"  
  
# =====  
# Conda 初始化  
# =====  
__conda_setup=$( '/root/miniconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)  
if [ $? -eq 0 ]; then  
    eval "$__conda_setup"  
else  
    if [ -f "/root/miniconda3/etc/profile.d/conda.sh" ]; then  
        . "/root/miniconda3/etc/profile.d/conda.sh"
```

```
else
    export PATH="/root/miniconda3/bin:$PATH"
fi
fi
unset __conda_setup

# 激活 conda 环境
conda activate agent101

# 运行 JupyterLab
# 使用 exec 确保 systemd 能正确跟踪主进程
exec /usr/local/bin/jupyter-lab \
--config=/root/.jupyter/jupyter_lab_config.py \
--no-browser
```

重要说明:

- 将所有 `your_*_key_here` 替换为实际的 API Key
- 或者通过 `source ~/.bashrc` 从全局配置加载

设置脚本权限:

```
sudo chmod +x /Agent101/app/jupyter/start_jupyter.sh
```

测试脚本:

```
# 手动运行测试
/Agent101/app/jupyter/start_jupyter.sh
# 按 Ctrl+C 停止
```

8.2 创建 systemd 服务

创建 systemd 服务文件:

```
sudo vim /etc/systemd/system/jupyter.service
```

服务配置内容:

```
[Unit]
Description=Jupyterlab service
Documentation=https://jupyter.org/
After=network.target

[Service]
Type=simple
User=root
Group=root
WorkingDirectory=/Agent101/code
Execstart=/Agent101/app/jupyter/start_jupyter.sh
Restart=always
RestartSec=10
```

```
standardOutput=journal  
standardError=journal  
  
[Install]  
WantedBy=multi-user.target
```

配置说明：

| 配置项 | 说明 | 值 |
|------------------|--------|-------------------------|
| Description | 服务描述 | JupyterLab Service |
| After | 启动顺序 | network.target (网络服务之后) |
| Type | 服务类型 | simple |
| User | 运行用户 | root (或你的用户名) |
| WorkingDirectory | 工作目录 | /Agent101/code |
| ExecStart | 启动命令 | 启动脚本路径 |
| Restart | 重启策略 | always (失败自动重启) |
| RestartSec | 重启等待时间 | 10秒 |

8.3 启动和管理服务

```
# 1. 重载 systemd 配置  
sudo systemctl daemon-reload  
  
# 2. 启用开机自启  
sudo systemctl enable jupyter.service  
  
# 3. 启动服务  
sudo systemctl start jupyter.service  
  
# 4. 查看服务状态  
sudo systemctl status jupyter.service
```

服务状态输出示例：

```
● jupyter.service - Jupyterlab Service
   Loaded: loaded (/etc/systemd/system/jupyter.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2025-01-01 10:00:00 CST; 5min ago
       Main PID: 12345 (jupyter-lab)
          Tasks: 12
         Memory: 150.0M
            CPU: 2.5s
           CGroup: /system.slice/jupyter.service
                     └─12345 /root/miniconda3/envs/agent101/bin/python
/usr/local/bin/jupyter-lab...
```

8.4 服务管理命令

```
# 启动服务
sudo systemctl start jupyter.service

# 停止服务
sudo systemctl stop jupyter.service

# 重启服务
sudo systemctl restart jupyter.service

# 重新加载配置（不中断服务）
sudo systemctl reload jupyter.service

# 查看服务状态
sudo systemctl status jupyter.service

# 查看服务日志
sudo journalctl -u jupyter.service

# 实时查看日志
sudo journalctl -u jupyter.service -f

# 查看最近 50 行日志
sudo journalctl -u jupyter.service -n 50

# 禁用开机自启
sudo systemctl disable jupyter.service
```

8.5 访问 JupyterLab

服务启动后，可以通过以下方式访问：

```
# 本地访问  
http://localhost:8000  
  
# 远程访问（使用虚拟机 IP）  
http://192.168.x.x:8000  
  
# 使用 token 登录  
Token: fly123
```

首次登录后设置密码（可选）：

- 登录后 → 右上角 → Settings → Set Password
- 设置永久密码，无需每次输入 token

步骤9：验证环境

9.1 验证系统环境

```
# 检查操作系统版本  
lsb_release -a  
# 输出应包含: Description: Ubuntu 22.04.4 LTS  
  
# 检查系统资源  
free -h      # 内存  
df -h       # 磁盘  
nproc        # CPU 核心数
```

9.2 验证 Python 和 Conda

```
# 激活环境  
conda activate agent101  
  
# 检查 Python 版本  
python --version  
# 输出: Python 3.10.18  
  
# 检查 conda 版本  
conda --version  
  
# 检查 pip 版本  
pip --version  
  
# 查看已安装的环境  
conda env list  
  
# 查看当前环境的包  
pip list | head -20
```

9.3 验证 JupyterLab

```
# 检查 JupyterLab 版本
jupyter lab --version

# 检查已注册的内核
jupyter kernelspec list
# 应该看到 agent101 内核

# 检查 JupyterLab 服务状态
sudo systemctl status jupyter.service
# 应该显示 active (running)
```

9.4 验证环境变量

```
# 检查环境变量
echo "OpenAI Base URL: $OPENAI_BASE_URL"
echo "OpenAI API Key: ${OPENAI_API_KEY:0:20}..."
echo "DeepSeek API Key: ${DEEPESEEK_API_KEY:0:20}..."
echo "Langfuse Host: $LANGFUSE_HOST"

# Python 中验证
python << 'EOF'
import os
import sys

print("=*60)
print("环境验证报告")
print("=*60)

# Python 版本
print(f"\n✓ Python 版本: {sys.version}")

# 环境变量
env_vars = [
    "OPENAI_API_KEY",
    "DEEPESEEK_API_KEY",
    "LANGFUSE_PUBLIC_KEY",
    "TAVILY_API_KEY"
]

print("\n环境变量配置:")
for var in env_vars:
    value = os.getenv(var)
    if value:
        print(f" ✓ {var}: {value[:20]}...")
    else:
        print(f" ✗ {var}: 未配置")

# 导入测试
print("\n核心包导入测试:")
```

```

packages = {
    "langchain": "langchain",
    "langgraph": "langgraph",
    "openai": "openai",
    "fastapi": "fastapi",
    "langfuse": "langfuse"
}

for name, module in packages.items():
    try:
        pkg = __import__(module)
        version = getattr(pkg, "__version__", "unknown")
        print(f" ✓ {name}: {version}")
    except ImportError:
        print(f" ✗ {name}: 未安装")

print("\n"*60)
EOF

```

9.5 在 JupyterLab 中测试

访问 JupyterLab: <http://localhost:8000>

创建新的 Notebook, 选择 `Python (agent101)` 内核, 运行以下代码:

```

# Cell 1: 系统信息
import sys
import platform

print(f"Python 版本: {sys.version}")
print(f"平台: {platform.platform()}")
print(f"处理器: {platform.processor()}")

# Cell 2: 环境变量
import os

print("环境变量检查:")
print(f"OPENAI_API_KEY: {os.getenv('OPENAI_API_KEY', 'Not Set')[:20]}...")
print(f"DEEPEEK_API_KEY: {os.getenv('DEEPEEK_API_KEY', 'Not Set')[:20]}...")
print(f"LANGFUSE_PUBLIC_KEY: {os.getenv('LANGFUSE_PUBLIC_KEY', 'Not Set')}[:20]...")

# Cell 3: 核心包测试
import langchain
import langgraph
import openai
import fastapi
import streamlit

print("核心包版本:")
print(f"LangChain: {langchain.__version__}")
print(f"OpenAI: {openai.__version__}")
print(f"FastAPI: {fastapi.__version__}")

```

```

print(f"Streamlit: {streamlit.__version__}")

# Cell 4: 简单 API 测试（需要有效的 API Key）
from openai import OpenAI

try:
    client = OpenAI(
        api_key=os.getenv("OPENAI_API_KEY"),
        base_url=os.getenv("OPENAI_BASE_URL")
    )

    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": "Hello!"}],
        max_tokens=10
    )

    print("✓ OpenAI API 连接成功!")
    print(f"响应: {response.choices[0].message.content}")
except Exception as e:
    print(f"✗ API 测试失败: {str(e)}")

```

9.6 验证项目代码

```

# 进入项目目录
cd /Agent101/code/Agent_In_Action

# 查看项目结构
tree -L 2 -d

# 检查各项目的依赖文件
find . -name "requirements.txt" -type f

# 验证项目1的代码
cd 01-agent-tool-mcp/mcp-demo
ls -la

# 返回根目录
cd /Agent101/code/Agent_In_Action

```

9.7 完整验证脚本

创建一个验证脚本：

```
vim /tmp/verify_env.sh
```

脚本内容：

```

#!/bin/bash

echo "====="

```

```

echo " AI Agent 101 环境验证脚本"
echo "====="

# 颜色定义
RED='\033[0;31m'
GREEN='\033[0;32m'
NC='\033[0m' # No Color

# 检查函数
check_command() {
    if command -v $1 &> /dev/null; then
        echo -e "${GREEN}✓${NC} $1: $(command -v $1)"
        return 0
    else
        echo -e "${RED}✗${NC} $1: 未安装"
        return 1
    fi
}

# 系统信息
echo -e "\n[1] 系统信息"
echo "操作系统: $(lsb_release -d | cut -f2)"
echo "内核版本: $(uname -r)"
echo "CPU 核心: $(nproc)"
echo "内存: $(free -h | awk '/^Mem:/ {print $2}')"

# 基础工具
echo -e "\n[2] 基础工具"
check_command git
check_command wget
check_command curl
check_command vim

# Python 和 Conda
echo -e "\n[3] Python 和 Conda"
check_command python
check_command conda
check_command pip
echo "Python 版本: $(python --version 2>&1)"
echo "Conda 版本: $(conda --version 2>&1)"

# Jupyter
echo -e "\n[4] JupyterLab"
check_command jupyter
if [ $? -eq 0 ]; then
    echo "JupyterLab 版本: $(jupyter lab --version 2>&1)"
    echo "已注册的内核:"
    jupyter kernelspec list
fi

# JupyterLab 服务
echo -e "\n[5] JupyterLab 服务"
if systemctl is-active --quiet jupyter.service; then

```

```

echo -e "${GREEN}✓${NC} JupyterLab 服务: 运行中"
else
    echo -e "${RED}✗${NC} JupyterLab 服务: 未运行"
fi

# 环境变量
echo -e "\n[6] 环境变量"
vars=("OPENAI_API_KEY" "DEEPEEK_API_KEY" "LANGFUSE_PUBLIC_KEY")
for var in "${vars[@]}"; do
    if [ -n "${!var}" ]; then
        echo -e "${GREEN}✓${NC} $var: 已配置"
    else
        echo -e "${RED}✗${NC} $var: 未配置"
    fi
done

# 项目代码
echo -e "\n[7] 项目代码"
if [ -d "/Agent101/code/Agent_In_Action" ]; then
    echo -e "${GREEN}✓${NC} 项目目录: /Agent101/code/Agent_In_Action"
    echo "项目结构:"
    tree -L 1 -d /Agent101/code/Agent_In_Action 2>/dev/null || ls -d
/Agent101/code/Agent_In_Action/*
else
    echo -e "${RED}✗${NC} 项目目录不存在"
fi

# Docker (可选)
echo -e "\n[8] Docker (可选)"
if check_command docker &>/dev/null; then
    echo "Docker 版本: $(docker --version 2>&1)"
else
    echo "Docker 未安装 (某些项目需要)"
fi

echo -e "\n====="
echo "  验证完成"
echo "====="

```

运行验证:

```

chmod +x /tmp/verify_env.sh
/tmp/verify_env.sh

```



故障排除

问题 1：Conda 命令不可用

症状：

```
conda: command not found
```

解决方案：

```
# 方法1：重新初始化 conda
$HOME/miniconda3/bin/conda init bash
source ~/.bashrc

# 方法2：手动添加到 PATH
export PATH="$HOME/miniconda3/bin:$PATH"
source ~/.bashrc

# 方法3：使用绝对路径
$HOME/miniconda3/bin/conda --version
```

问题 2：JupyterLab 中 Conda 环境不可用

症状：

- 终端中 `conda activate` 不工作
- 没有 agent101 内核选项

解决方案：

```
# 确保 ipykernel 已安装
conda activate agent101
pip install ipykernel

# 重新注册内核
python -m ipykernel install --user --name=agent101 --display-name="Python
(agent101)"

# 验证
jupyter kernelspec list

# 如果还不行，检查 JupyterLab 启动脚本中的 Conda 初始化
cat /Agent101/app/jupyter/start_jupyter.sh
```

问题 3：systemd 服务启动失败

症状：

```
sudo systemctl status jupyter.service
# 显示：Failed to start Jupyterlab Service
```

解决方案：

```
# 1. 查看详细日志
sudo journalctl -u jupyter.service -n 50 --no-pager

# 2. 检查脚本权限
ls -l /Agent101/app/jupyter/start_jupyter.sh
# 应该显示: -rwxr-xr-x

# 如果权限不对
sudo chmod +x /Agent101/app/jupyter/start_jupyter.sh

# 3. 检查工作目录是否存在
ls -ld /Agent101/code

# 如果不存在
sudo mkdir -p /Agent101/code
sudo chown -R $USER:$USER /Agent101/code

# 4. 手动测试脚本
/Agent101/app/jupyter/start_jupyter.sh
# 查看错误信息

# 5. 检查 Jupyter 路径
which jupyter-lab
# 确保脚本中的路径正确

# 6. 重新加载服务
sudo systemctl daemon-reload
sudo systemctl restart jupyter.service
```

问题 4：端口被占用

症状：

```
OSErrror: [Errno 98] Address already in use
```

解决方案：

```
# 查看端口占用情况
sudo netstat -tulpn | grep 8000
# 或
sudo lsof -i :8000

# 结束占用端口的进程
sudo kill -9 <PID>

# 或修改配置文件使用其他端口
vim ~/.jupyter/jupyter_lab_config.py
# 修改: c.ServerApp.port = 8001
```

```
# 重启服务  
sudo systemctl restart jupyter.service
```

问题 5：虚拟机网络连接问题

症状：

- 无法访问外网
- ping 不通 8.8.8.8

解决方案：

```
# 1. 检查网络接口状态  
ip link show  
ip addr show  
  
# 2. 检查 DNS 配置  
cat /etc/resolv.conf  
  
# 3. 测试网络连通性  
ping 8.8.8.8      # 测试 IP 连通性  
ping google.com   # 测试 DNS 解析  
  
# 4. 重启网络服务  
sudo systemctl restart NetworkManager  
  
# 5. 检查虚拟机网络设置  
# VMware: 编辑 → 虚拟网络编辑器  
# VirtualBox: 设置 → 网络 → 网卡 1 → NAT  
  
# 6. 重新配置 DNS (临时)  
sudo vim /etc/resolv.conf  
# 添加：  
# nameserver 8.8.8.8  
# nameserver 114.114.114.114
```

问题 6：Python 包安装失败

症状：

```
ERROR: Could not find a version that satisfies the requirement...
```

解决方案：

```
# 1. 检查 pip 版本  
pip --version  
  
# 2. 升级 pip  
pip install --upgrade pip  
  
# 3. 检查 pip 镜像源
```

```
pip config list

# 4. 临时使用国内镜像安装
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple package_name

# 5. 清除 pip 缓存
pip cache purge

# 6. 检查 Python 版本兼容性
python --version
# 某些包可能不支持 Python 3.10
```

问题 7: JupyterLab 无法访问

症状:

- 浏览器无法打开 <http://localhost:8000>
- 连接被拒绝

解决方案:

```
# 1. 检查服务状态
sudo systemctl status jupyter.service

# 2. 检查端口监听
sudo netstat -tulpn | grep 8000

# 3. 检查防火墙
sudo ufw status
# 如果启用了防火墙, 添加规则
sudo ufw allow 8000/tcp

# 4. 检查配置文件中的 IP 设置
grep "c.ServerApp.ip" ~/.jupyter/jupyter_lab_config.py
# 应该是: c.ServerApp.ip = '0.0.0.0'

# 5. 使用虚拟机 IP 访问
ip addr show | grep inet
# 尝试 http://虚拟机IP:8000

# 6. 检查 token
grep "c.ServerApp.token" ~/.jupyter/jupyter_lab_config.py
```

问题 8: API 调用失败

症状:

```
openai.error.AuthenticationError: Incorrect API key provided
```

解决方案:

```

# 1. 检查环境变量
echo $OPENAI_API_KEY
echo $OPENAI_BASE_URL

# 2. 在 Python 中检查
python << 'EOF'
import os
print("API Key:", os.getenv("OPENAI_API_KEY"))
print("Base URL:", os.getenv("OPENAI_BASE_URL"))
EOF

# 3. 重新加载环境变量
source ~/.bashrc

# 4. 重启 JupyterLab 服务
sudo systemctl restart jupyter.service

# 5. 在 Jupyter Notebook 中手动设置
# 在代码中添加：
# import os
# os.environ["OPENAI_API_KEY"] = "your-key-here"

# 6. 验证 API Key 有效性
curl https://api.openai.com/v1/models \
-H "Authorization: Bearer $OPENAI_API_KEY"

```

问题 9：磁盘空间不足

症状：

```
OSSError: [Errno 28] No space left on device
```

解决方案：

```

# 1. 检查磁盘使用情况
df -h

# 2. 查找大文件
sudo find / -type f -size +100M 2>/dev/null | head -20

# 3. 清理 apt 缓存
sudo apt clean
sudo apt autoremove

# 4. 清理 conda 缓存
conda clean -a

# 5. 清理 pip 缓存
pip cache purge

# 6. 清理 Docker (如果安装了)
docker system prune -a

```

```
# 7. 清理日志  
sudo journalctl --vacuum-size=100M  
  
# 8. 扩展虚拟机硬盘（见下一节）
```

？常见问题

Q1：为什么推荐使用 Conda 而不是系统 Python？

A:

- ✓ 环境隔离：避免不同项目依赖冲突
- ✓ 版本管理：轻松切换 Python 版本
- ✓ 包管理：conda 包管理比 pip 更强大
- ✓ 跨平台：Windows、macOS、Linux 统一体验

Q2：虚拟机 vs Docker vs 云服务器，如何选择？

| 方案 | 优势 | 劣势 | 适用场景 |
|--------|-----------|------------|-----------|
| 虚拟机 | 完整系统、灵活配置 | 占用资源多、性能损耗 | 学习开发 |
| Docker | 轻量级、快速部署 | 不适合 GUI 应用 | 生产部署 |
| 云服务器 | 性能好、随时访问 | 需要付费 | 团队协作、生产环境 |

推荐方案：

- 初学者：使用预配置虚拟机（快速开始）
- 进阶用户：本地 Ubuntu + Docker（灵活开发）
- 团队项目：云服务器（稳定可靠）

Q3：为什么要使用 systemd 管理 JupyterLab？

A:

- ✓ 开机自启：系统启动自动运行
- ✓ 后台运行：不占用终端
- ✓ 自动重启：进程崩溃自动恢复
- ✓ 日志管理：systemd 统一管理日志
- ✓ 资源控制：可以限制CPU、内存使用

Q4: JupyterLab vs Jupyter Notebook, 有什么区别?

A:

| 特性 | JupyterLab | Jupyter Notebook |
|-----|--|------------------|
| 界面 | 现代化、多标签页 | 传统、单文件 |
| 功能 | 更强大 (编辑器、终端、文件浏览) | 基础功能 |
| 扩展性 | 支持丰富的扩展 | 扩展较少 |
| 性能 | 更好 | 一般 |
| 推荐度 | <input checked="" type="checkbox"/> 推荐 | 已过时 |

Q5: 如何提升虚拟机性能?

A:

1. 分配更多资源

- 增加 CPU 核心数
- 增加内存大小
- 使用 SSD 存储

2. 优化虚拟机设置

- VMware: 启用硬件虚拟化、VT-x/AMD-V
- VirtualBox: 启用 VT-x/AMD-V、硬件加速

3. 系统优化

```
# 禁用不必要的服务
sudo systemctl disable bluetooth
sudo systemctl disable cups

# 减少 Swap 使用
sudo sysctl vm.swappiness=10
```

4. 使用轻量级桌面环境

- 考虑使用 Xfce 替代 GNOME
- 或使用纯命令行 (无 GUI)

Q8: API Key 如何安全管理?

A:

⚠ 不要做的事:

- ✗ 将 API Key 提交到 Git 仓库
- ✗ 在代码中硬编码 API Key

- ✗ 在公开场所分享截图（包含 API Key）

推荐做法：

1. 使用环境变量（当前方案）
2. 使用 .env 文件 + .gitignore

```
# .gitignore
.env
*.key
```

3. 使用密钥管理工具

- AWS Secrets Manager
- HashiCorp Vault
- 1Password

4. 设置 API Key 权限和限制

- 限制 API Key 的使用范围
- 设置每月使用额度
- 定期轮换 API Key

学习资源

官方文档

- **Ubuntu:** <https://ubuntu.com/server/docs>
- **Conda:** <https://docs.conda.io/>
- **JupyterLab:** <https://jupyterlab.readthedocs.io/>
- **LangChain:** <https://python.langchain.com/docs/>
- **LangGraph:** <https://langchain-ai.github.io/langgraph/>

社区资源

- **Ubuntu 中文论坛:** <https://forum.ubuntu.org.cn/>
- **Python 官方教程:** <https://docs.python.org/3/tutorial/>
- **GitHub:** https://github.com/FlyAIBox/Agent_In_Action

获取帮助

如果在环境搭建过程中遇到问题，可以通过以下方式获取帮助：

- **GitHub Issues:** https://github.com/FlyAIBox/Agent_In_Action/issues
- **邮件:** fly910905@sina.com
- **微信公众号:** 茵火AI百宝箱

★ 如果本指南对你有帮助, 请给项目点个 Star 支持! ★

[GitHub 项目地址](#)

文档版本: v2.0

更新日期: 2025-11-15

维护者: 茧火AI百宝箱

祝你学习顺利! 🎉