使用Docker容器进行服务器资源分配：实践过程

前言

通过上篇文章我们已经了解了Docker的优势，包括资源隔离、环境简化和防止资源竞争等。在本文中我详细记录如何在A01服务器上利用Docker将计算资源分配给四个小组的实践过程。

需求分析

首先需要明确各小组的具体需求和可用资源，进行合理规划：
1. 四个小组中有两个需要RAG功能
2. 所有小组都需要使用deepseek r1:70b模型
3. A01服务器拥有多张GPU（8张3090）和充足的CPU核心（96逻辑核心）
4. 需要平衡资源分配，避免竞争和浪费

基于上述需求，我们将采用"2+2"的容器组合方案：两个Ollama容器运行模型，两个AnythingLLM容器提供RAG服务。

我们的A01服务器需要支持四个小组同时使用，其中两个小组有RAG(检索增强生成)需求，所有小组都需要使用deepseek r1:70b模型。

资源需求评估

deepseek r1:70b模型在Ollama平台上的量化版本(q4_k_m)约43GB大小，配置要求：

- 2-4张RTX 3090 GPU
- 64-128GB内存
- 8-16个CPU线程

基于A01服务器的硬件规格，我们可以同时运行多个模型实例，采用以下资源分配策略：

1. 部署两个Ollama容器运行大模型，每个容器服务两个小组
2. 部署两个AnythingLLM容器提供RAG服务，分配给有需求的小组

为什么不使用单一容器?

采用多容器方案的主要优势：
1. 资源利用率：单容器方案下，一个70b模型仅使用2-3张GPU，导致其他GPU资源闲置
2. 并发性能：多个小组同时使用单一模型实例会造成请求排队和响应延迟
3. 隔离性：多容器可以实现更好的资源隔离，避免一个小组的高负载影响其他小组
4. 定制化：不同容器可以针对不同小组需求进行独立配置

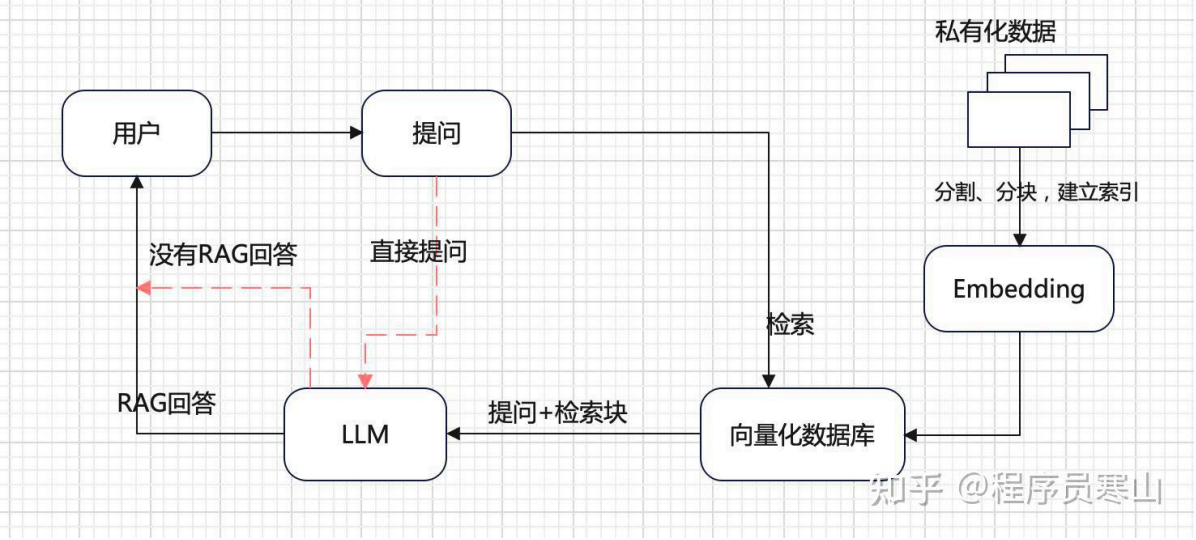
采用多容器方案的主要原因是为了提高资源利用率和服务质量：

1. **资源充分利用**：一个70b模型实例仅使用2-3张GPU，采用多容器可以充分利用所有GPU资源
2. **避免并发卡顿**：多个小组同时使用单一模型实例会导致请求队列积压，造成响应延迟
3. **灵活配置**：不同容器可以根据小组需求独立配置参数

当然，对于需要集中大量资源的超大模型（如需要8张GPU协同工作的模型），使用单一容器是更合适的选择。

RAG简介

RAG(检索增强生成)是大模型应用的关键技术：
1. 核心原理：在生成回答前，先从知识库检索相关信息，然后结合检索内容与语言模型能力生成更准确的回答
2. 主要组件：向量数据库、文档处理器、embedding模型、大语言模型
3. 优势：克服大模型知识时效性限制，提供可溯源的专业领域回答，减少幻觉
4. 实现架构：文档拆分→向量化→存储→相似度检索→上下文融合→大模型生成



RAG(Retrieval-Augmented Generation，检索增强生成)是一种结合文档检索和生成式AI的技术方案，能够显著提升大语言模型在专业领域的表现。

RAG的工作流程：

1. 将知识库文档分割成小块并向量化存储
2. 用户提问时，系统检索与问题最相关的文档片段
3. 将检索到的相关内容作为上下文提供给大语言模型
4. 大语言模型基于检索内容和自身能力生成回答

AnythingLLM是一个集成了RAG功能的开源平台，提供了文档管理、向量存储、多模型接入等功能，大大简化了RAG系统的部署难度。

容器配置方案

基于需求分析，我们采用如下资源分配方案：

Ollama容器（大模型服务）

- **ollama-1**:
  - 内存: 64GB
  - GPU: 0,1,2号卡
  - CPU: 8-23核
  - 服务于小组1和小组2
- **ollama-2**:

- 内存: 64GB
- GPU: 3,4,5号卡
- CPU: 24-39核
- 服务于小组3和小组4

AnythingLLM容器（RAG服务）

- **anythingllm-1**:
  - 内存: 16GB
  - CPU: 56-63核
  - 连接到ollama-1
  - 服务于小组1
- **anythingllm-2**:
  - 内存: 16GB
  - CPU: 72-79核
  - 连接到ollama-2
  - 服务于小组2

从零开始的部署过程

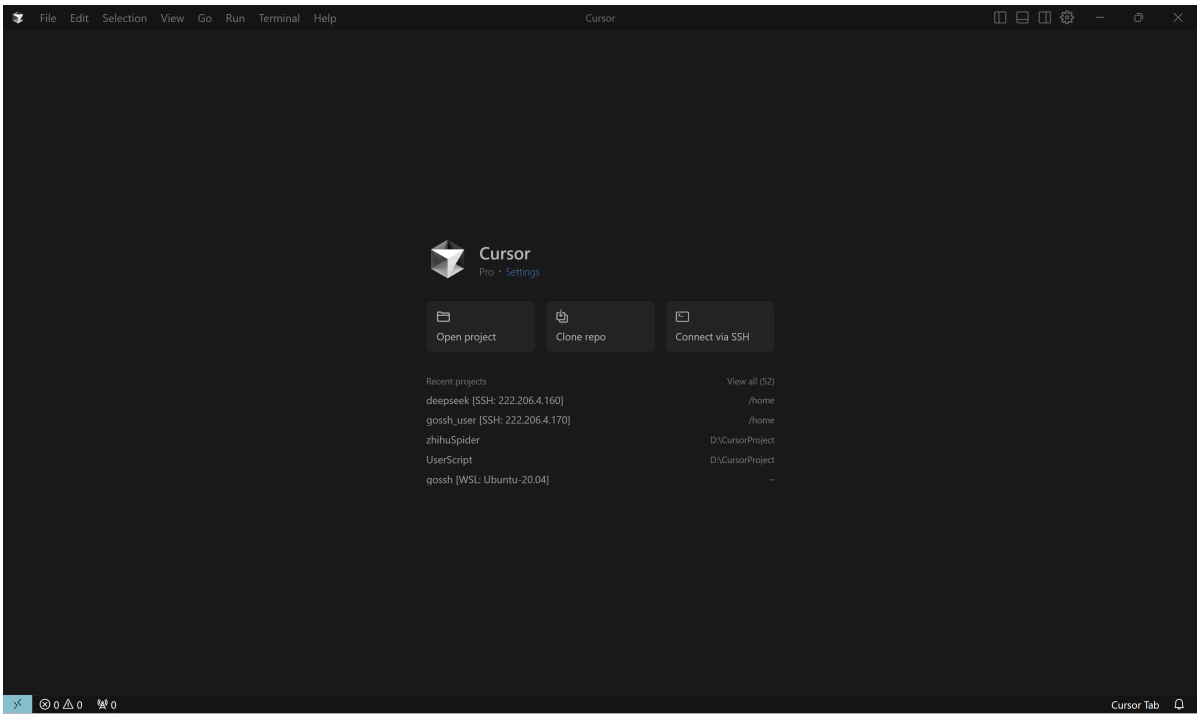
完整部署过程需要包含以下关键步骤：
1. ssh连接上服务器。
2. 环境准备：确认Docker和Docker Compose安装和配置（这些我都已经配置好）
3. 配置文件编写：创建合适的docker-compose.yml文件
4. 容器启动与验证：启动容器并确认服务正常
5. 用户访问配置：设置访问方式和权限

1.SSH远程连接服务器

我采用VSCode（Cursor版）的远程SSH功能连接服务器，与传统终端相比，这种方式提供了完整的IDE开发体验，使远程开发如同本地操作一般便捷。

1.1 配置远程SSH连接

1. 打开VSCode/Cursor主界面，在左侧活动栏找到远程资源管理器图标或点击左下角的状态栏
2. 选择`Connect to Host...` > `Connect via SSH`
3. 如系统提示"Remote SSH插件未安装"，请先安装该插件



1.2 建立连接

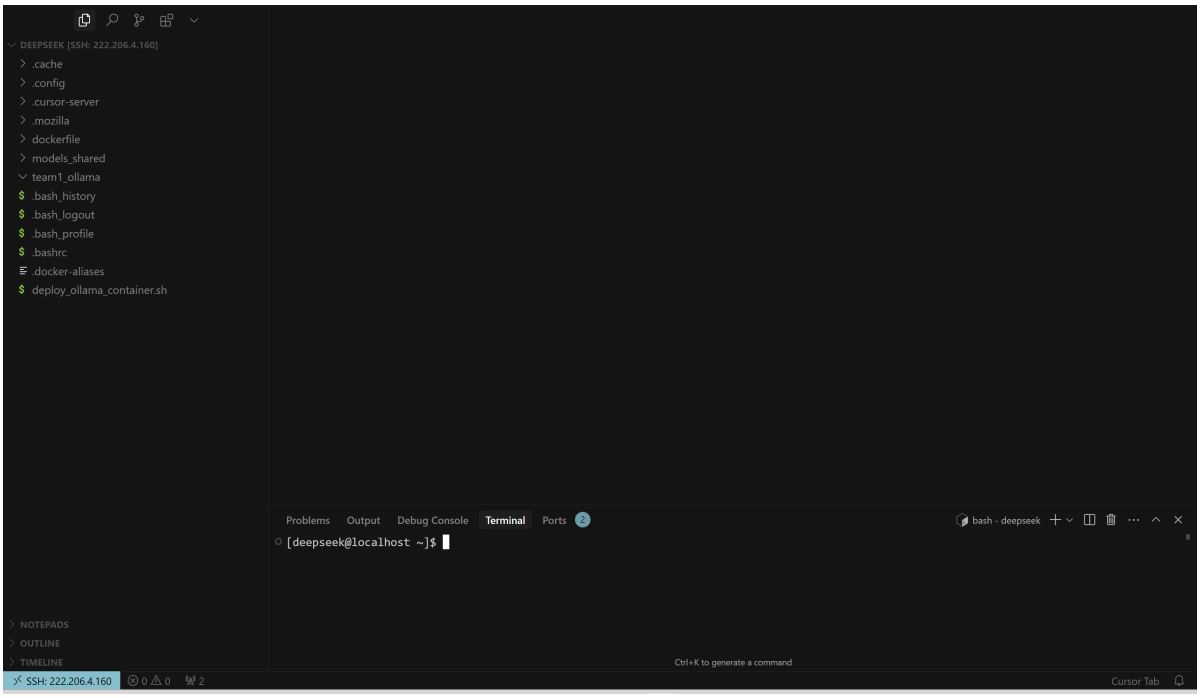
1. 在弹出的输入框中输入连接信息，格式为：`username@server_ip`（例如：deepseek@222.206.4.160）
2. 首次连接时选择SSH配置文件保存位置
3. 输入服务器登录密码
4. 等待VSCode/Cursor下载并安装远程服务器所需依赖

> **注意**：如遇到"服务器SSH版本过低"或"版本不匹配"的警告，通常可以忽略，不影响基本功能使用。

1.3 确认连接状态

连接成功后，可通过以下方式确认：

1. 查看左下角状态栏，显示`SSH: 222.206.4.160`（或您连接的服务器IP）表示已成功连接
2. 左侧资源管理器显示的是服务器上的文件结构，而非本地文件
3. 集成终端自动切换至服务器环境，所有命令将在远程服务器上执行



至此，您已成功建立远程开发环境，可以像使用本地VSCode一样在远程服务器上进行开发工作。

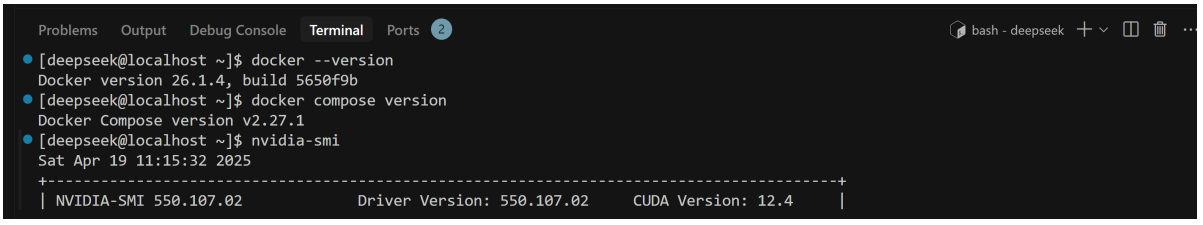
2. 环境准备

首先确认服务器已安装Docker和Docker Compose（这里我已经全部配置完毕了）：

```
# 检查Docker版本
docker --version

# 检查Docker Compose版本
docker compose version

# 确认NVIDIA Docker支持
nvidia-smi
```



3. 创建项目目录

每台配备3090显卡的服务器均挂载了一个10TB容量的`/models_shared`目录。该目录设置了完全访问权限（权限值777 = 所有用户可读、可写、可执行），这一步骤至关重要，否则后续部署过程中将遇到权限不足问题。

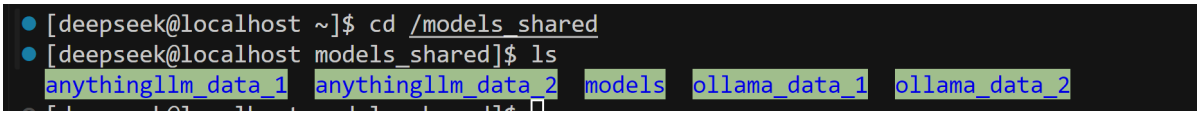
> **特别注意**：此目录位于根目录`/`下，不是用户家目录`~/models_shared`。系统中可能同时存在两个同名目录，请确保使用正确的路径。

```
# 创建项目层级目录结构，用于Docker卷挂载
# 采用外部挂载而非Docker自动管理，确保大容量数据存储和共享
mkdir -p
/models_shared/{models,ollama_data_1,ollama_data_2,anythingllm_data_1,anythingllm
_data_2}
```

目录结构说明：

- `models`：集中存储多个ollama容器共享的模型文件
- `ollama_data_1/2`：存储不同ollama实例的配置和运行数据
- `anythingllm_data_1/2`：存储anythingllm应用的数据文件和配置

权限验证：使用`ls -la /models_shared`命令检查，正确的权限显示应为`drwxrwxrwx`，表示所有用户对该目录具有完全控制权。



4. 编写Docker Compose配置文件

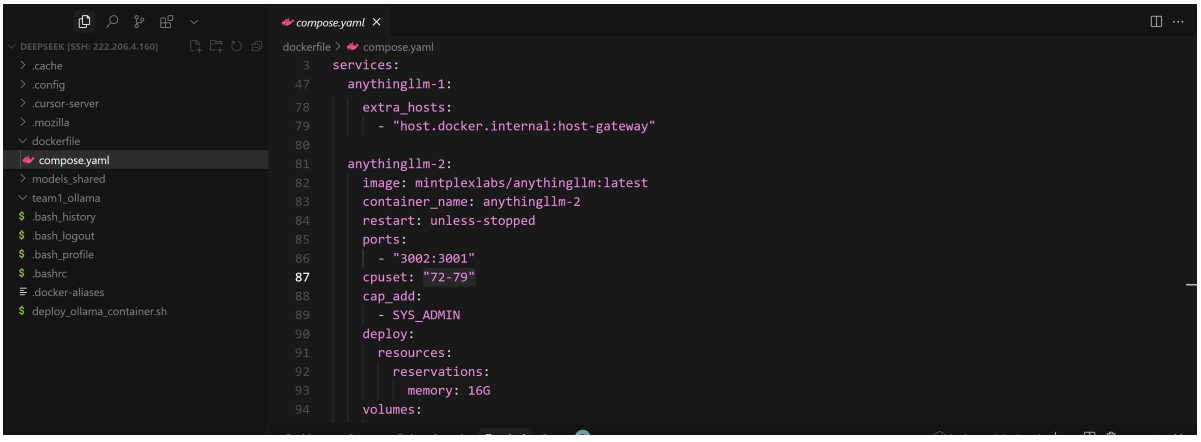
Docker Compose是一种用于定义和运行多容器Docker应用程序的工具。通过一个YAML格式的配置文件，您可以同时配置、启动和管理多个相互关联的容器服务。

在任意合适的位置创建`compose.yaml`文件：

```
mkdir -p ~/dockerfile     # 创建目录（如果不存在）
nano ~/dockerfile/compose.yaml
```

> **提示**：文件位置可以根据您的偏好和组织习惯自由选择，本例使用`~/dockerfile/`目录仅作为参考示例。

配置文件保存位置示意：



请在编辑器中填入以下配置内容。**建议仔细阅读每个配置段落的注释**，这将帮助您理解各项参数的作用，为后续根据实际需求进行自定义修改奠定基础：

```
version: '3.8'

services:
  ollama-1:
```

```yaml
    image: ollama/ollama:latest
    container_name: ollama-1
    restart: unless-stopped
    ports:
      - "11434:11434"
    deploy:
      resources:
        reservations:
          memory: 64G
          devices:
            - driver: nvidia
              device_ids: ['0', '1', '2']
              capabilities: [gpu]
    cpuset: "8-23"
    volumes:
      - /models_shared/ollama_data_1:/root/.ollama/config
      - /models_shared/models:/root/.ollama/models
    environment:
      - OLLAMA_HOST=0.0.0.0

  ollama-2:
    image: ollama/ollama:latest
    container_name: ollama-2
    restart: unless-stopped
    ports:
      - "11435:11434"
    deploy:
      resources:
        reservations:
          memory: 64G
          devices:
            - driver: nvidia
              device_ids: ['3', '4', '5']
              capabilities: [gpu]
    cpuset: "24-39"
    volumes:
      - /models_shared/ollama_data_2:/root/.ollama/config
      - /models_shared/models:/root/.ollama/models
    environment:
      - OLLAMA_HOST=0.0.0.0

  anythingllm-1:
    image: mintplexlabs/anythingllm:latest
    container_name: anythingllm-1
    restart: unless-stopped
    ports:
      - "3001:3001"
    cpuset: "56-63"
    cap_add:
      - SYS_ADMIN
    deploy:
      resources:
        reservations:
          memory: 16G
    volumes:
      - /models_shared/anythingllm_data_1:/app/server/storage
```

```
    environment:
      - OLLAMA_API_HOST=http://172.17.0.1:11434
      - LLM_PROVIDER=ollama
      - OLLAMA_BASE_PATH=http://172.17.0.1:11434
      - OLLAMA_MODEL_PREF=llama2
      - OLLAMA_MODEL_TOKEN_LIMIT=4096
      - EMBEDDING_ENGINE=ollama
      - EMBEDDING_BASE_PATH=http://172.17.0.1:11434
      - EMBEDDING_MODEL_PREF=nomic-embed-text:latest
      - EMBEDDING_MODEL_MAX_CHUNK_LENGTH=8192
      - VECTOR_DB=lancedb
      - STORAGE_DIR=/app/server/storage
      - JWT_SECRET=your_secret_key_change_this_to_something_secure_and_random
      - PASSWORDMINCHAR=8
      - WHISPER_PROVIDER=local
      - TTS_PROVIDER=native
    extra_hosts:
      - "host.docker.internal:host-gateway"

  anythingllm-2:
    image: mintplexlabs/anythingllm:latest
    container_name: anythingllm-2
    restart: unless-stopped
    ports:
      - "3002:3001"
    cpuset: "72-79"
    cap_add:
      - SYS_ADMIN
    deploy:
      resources:
        reservations:
          memory: 16G
    volumes:
      - /models_shared/anythingllm_data_2:/app/server/storage
    environment:
      - OLLAMA_API_HOST=http://172.17.0.1:11435
      - LLM_PROVIDER=ollama
      - OLLAMA_BASE_PATH=http://172.17.0.1:11435
      - OLLAMA_MODEL_PREF=llama2
      - OLLAMA_MODEL_TOKEN_LIMIT=4096
      - EMBEDDING_ENGINE=ollama
      - EMBEDDING_BASE_PATH=http://172.17.0.1:11435
      - EMBEDDING_MODEL_PREF=nomic-embed-text:latest
      - EMBEDDING_MODEL_MAX_CHUNK_LENGTH=8192
      - VECTOR_DB=lancedb
      - STORAGE_DIR=/app/server/storage
      - JWT_SECRET=your_secret_key_change_this_to_something_secure_and_random
      - PASSWORDMINCHAR=8
      - WHISPER_PROVIDER=local
      - TTS_PROVIDER=native
    extra_hosts:
      - "host.docker.internal:host-gateway"

networks:
  app_network:
    driver: bridge
```

4. 配置说明

这个配置文件中有几个关键设置需要特别说明：
1. 资源分配：CPU核心绑定(cpuset)、GPU设备ID分配和内存限制
2. 网络设置：端口映射和内部通信配置
3. 存储卷挂载：模型共享和数据持久化
4. 环境变量：服务参数和连接配置
5. 安全配置：JWT密钥和权限设置

特别注意的是AnythingLLM容器如何连接到对应的Ollama服务。

Ollama容器配置关键点：

- **端口映射**：ollama-1使用11434端口，ollama-2使用11435端口，避免冲突
- **GPU分配**：通过`device_ids`指定每个容器使用的GPU编号
- **CPU绑定**：通过`cpuset`参数绑定特定CPU核心，确保计算资源不冲突
- **内存预留**：每个容器预留64GB内存
- **模型共享**：两个容器共享相同的模型目录，避免模型重复下载

AnythingLLM容器配置关键点：

- **连接到Ollama**：通过`OLLAMA_API_HOST`和`OLLAMA_BASE_PATH`环境变量指向对应的Ollama服务
- **向量数据库**：使用内置的LanceDB作为向量存储
- **数据持久化**：通过卷挂载实现数据持久存储
- **安全设置**：设置JWT密钥保障API安全（实际部署时应使用强密码）

5. 启动服务

```
# 进入配置文件目录（只能在有配置文件的目录下运行compose命令）
cd dockerfile

# 启动所有容器
docker compose up -d

#也可以选择启动方便测试
docker compose up -d ollama-1 ollama-2

# 查看容器运行状态
docker compose ps
```

```
[deepseek@localhost dockerfile]$ docker compose ps
WARN[0000] /home/deepseek/dockerfile/compose.yaml: `version` is obsolete
NAME            IMAGE                              COMMAND                  SERVICE         CREATED        STATUS                   PORTS
anythingllm-1   mintplexlabs/anythingllm:latest   "/bin/bash /usr/loca…"   anythingllm-1   12 hours ago   Up 12 hours (healthy)   0.0.0.0
:3001->3001/tcp, :::3001->3001/tcp
anythingllm-2   mintplexlabs/anythingllm:latest   "/bin/bash /usr/loca…"   anythingllm-2   12 hours ago   Up 12 hours (healthy)   0.0.0.0
:3002->3001/tcp, :::3002->3001/tcp
ollama-1        ollama/ollama:latest             "/bin/ollama serve"      ollama-1        12 hours ago   Up 12 hours             0.0.0.0
:11434->11434/tcp, :::11434->11434/tcp
ollama-2        ollama/ollama:latest             "/bin/ollama serve"      ollama-2        12 hours ago   Up 12 hours             0.0.0.0
:11435->11434/tcp, :::11435->11434/tcp
```

6. 模型加载与配置

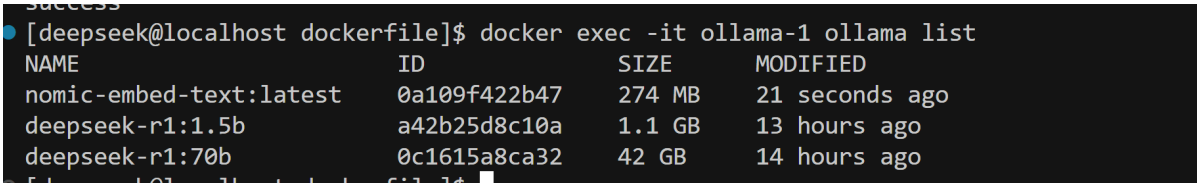
由于我们配置了模型目录共享，只需在一个容器中拉取模型，其他容器即可共享使用：

```
# 在ollama-1中加载deepseek-r1:70b大语言模型
docker exec -it ollama-1 ollama pull deepseek-r1:70b

# 加载文本嵌入模型(用于向量化文档)
docker exec -it ollama-1 ollama pull nomic-embed-text:latest

# 验证模型是否成功加载
docker exec -it ollama-1 ollama list
```

模型加载成功后，终端将显示类似下图的输出，列出已安装的模型及其大小：



注意事项:

- **大模型下载时间**：deepseek-r1:70b是一个大型模型，即使是量化版本也有约43GB，下载过程可能需要较长时间，请耐心等待
- **共享机制**：由于我们配置了模型目录共享(`/models_shared/models`)，一个容器中下载的模型自动对所有容器可用
- **网络问题排查**：如遇到下载失败、DNS解析错误等问题，很可能是防火墙配置导致，服务器已针对此问题进行了预配置
- **磁盘空间**：确保服务器有足够的磁盘空间用于存储模型文件

成功加载模型后，可以启动AnythingLLM容器，开始配置RAG系统。

7. AnythingLLM平台配置

AnythingLLM是一个功能丰富的RAG平台，建议小组成员通过自主探索掌握其使用方法。以下提供基础配置参考：

7.1 平台访问地址

各小组可通过以下地址访问各自的AnythingLLM实例：

- 小组1: http://服务器IP:3001
- 小组2: http://服务器IP:3002

7.2 基础配置流程

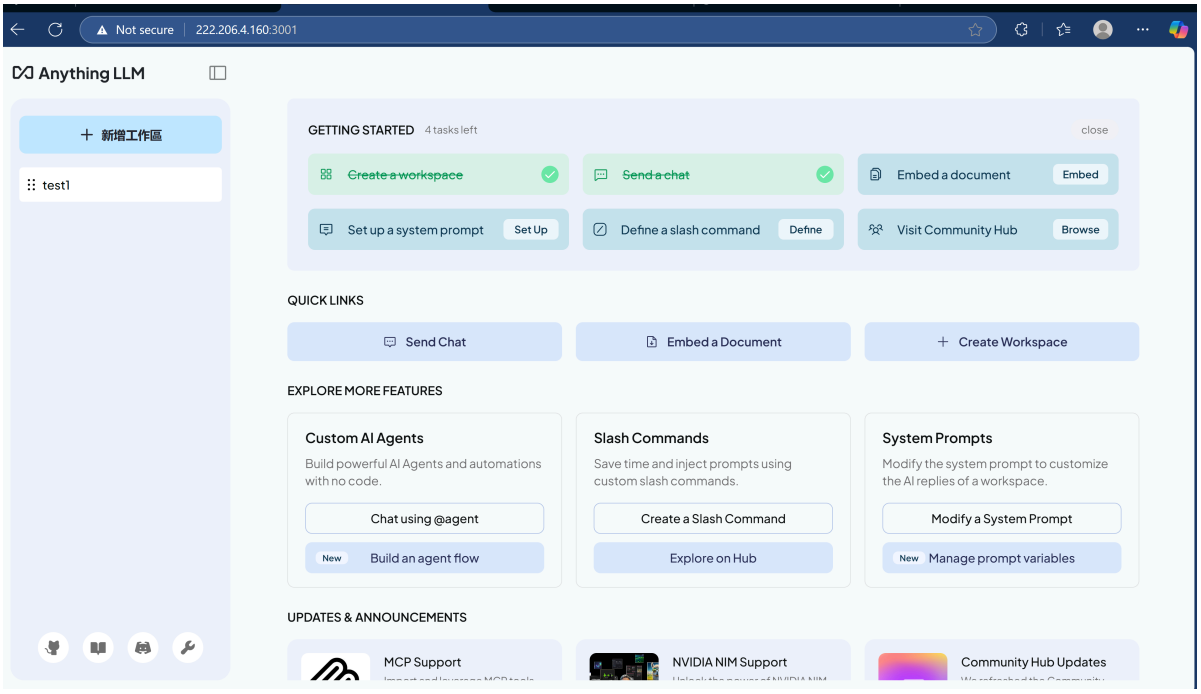
平台首次使用需完成以下基本设置：

1. **系统初始化**：创建管理员账户，设置安全密码
2. **语言模型连接**：选择"Ollama"作为LLM提供商，配置API地址（分别为http://172.17.0.1:11434和http://172.17.0.1:11435）
3. **向量引擎设置**：选择"Ollama"作为Embedding引擎，指定nomic-embed-text模型

4. **工作空间创建**：建立知识库工作空间，导入相关文档资料
5. **用户权限管理**：创建团队成员账户，分配适当的工作空间访问权限

AnythingLLM平台拥有丰富的功能和配置选项，鼓励各小组根据实际需求深入探索平台功能，定制专属知识库解决方案。

> **提示**：平台提供详细的内置文档和帮助指南，建议在配置过程中参考这些资源。



8. 用户访问配置

为小组成员提供服务访问信息：

1. **Ollama API访问**：
   - 小组1和2访问: http://服务器IP:11434
   - 小组3和4访问: http://服务器IP:11435
   - ollama API请参阅：ollama/docs/api.md at main · ollama/ollama · GitHub
2. **AnythingLLM访问**：
   - 小组1访问: http://服务器IP:3001
   - 小组2访问: http://服务器IP:3002

维护与监控

```
# 查看容器日志
docker logs ollama-1
docker logs anythingllm-1

# 监控GPU使用情况
watch -n 1 nvidia-smi

# 查看容器资源使用情况
docker stats
```

定期检查服务状态和资源使用情况，确保系统稳定运行。

总结

通过Docker容器技术，成功地将A01服务器的计算资源分配给四个研究小组，实现了资源隔离和高效利用。每个小组都能获得稳定的大模型服务，有RAG需求的小组还可以使用专用的AnythingLLM实例进行知识库构建和查询。

这种方案不仅提高了资源利用率，也避免了并发访问造成的性能下降，为各个研究小组提供了更好的使用体验。