在Centos7服务器上使用Docker和GPU部署Ollama大模型

1. 任务目标

我们学校有多台配置比较高的服务器（多核CPU、好几块GPU、大内存），但每个小组其实只需要用很小一部分资源。比如A02这台机器配置如下，跑个7B的模型，可能就只需要1张3090GPU和一小部分CPU核心。下面看我们的任务目标

A02	Intel Xeon Gold 5318Y 架构:x86_64 核心数: 48核 / 96线程 基础频率: 2.10GHz，睿频: 3.4GHz 三级缓存: 36864K	8× NVIDIA GeForce RTX 3090 NVIDIA驱动版本：550.107.02 CUDA版本：12.4	总容量: 503GB	1. 物理存储总容量: sda 893.8G disk MR9361-8i sdb 49.1T disk MR9361-8i 2. 根分区使用情况: 根分区(/): 总容量: 50G, 已用: 6.9G (14%)	CentOS Linux 7 (Core) 版本: 7 内核: Linux

目标：想办法把机器资源切成几份，让大家都能用上自己需要的那部分，在需要的部分部署大模型，不浪费也不互相打扰。

我经过实践，发现使用Docker容器+nvidia container toolkit这个环境可以将环境之间隔离开来，分别运行。为了明白如何在容器上配置大模型需要首先了解如下内容

2. Docker是什么？容器，镜像，卷

刚开始接触Docker时我一头雾水，但理解了这三个概念后就豁然开朗了：

容器（Container）

我理解的容器就像是一个"迷你电脑"：

- 它就像你租了一台虚拟的Linux机器，但启动超快
- 进去之后跟正常Linux用起来没啥区别
- 但它只占用你分配给它的那部分资源
- 最棒的是，它已经装好了所有复杂的依赖，省去了配环境的痛苦

镜像（Image）

镜像就像是容器的"出厂设置"：

- 可以把它想象成一个装好了特定软件的Linux系统"快照"
- 就像游戏存档，一键加载就能用
- 例如：ollama/ollama这个镜像已经帮我装好了运行Ollama的一切东西

数据卷（Volume）

这个我一开始不理解，但是它很重要：

- 容器删了重建，里面的数据会丢失
- 但放在卷里的数据永远不会丢
- 相当于给容器插了个"移动硬盘"
- 我们的模型文件动辄十几GB，肯定得放卷里

3. 什么是Ollama平台?

Ollama是什么?

简单说，它就是个让你非常容易用起大模型的平台工具：

- 不需要写复杂的Python代码
- 一条命令就能下载并运行相应的模型
- 提供API接口，任何应用都能轻松调用
- 最重要的是，它对GPU的支持特别好

我试了几个平台，Ollama是最容易上手的！

关于Ollama的模型

- Ollama本身不是模型，它只是个"播放器"
- 模型是单独下载的"音乐文件"
- 模型有大有小，从小小的1.5B到超大的70B不等
- 模型越大，需要的显存和算力就越多

不只有Ollama

我不是Ollama的"粉丝"，它只是众多选择之一。我们也可以换成：

- KTransformers
- vLLM
- Text Generation Inference (TGI)
- LM Studio

原理都差不多，只要换个镜像就行。我选Ollama纯粹因为它最简单。

4. "共享模型"方案

经过思考测试，我决定使用如下方案——"共享模型目录"

简单来说，这个方案两件事：

1. **模型文件集中存放**：

- 我创建了每个服务器一个中央"模型库"文件夹：`/home/deepseek/models_shared`
- 然后让所有Ollama容器共享访问这个目录
- 技术上讲，我把每个容器内的`/root/.ollama/models`路径都映射到这个共享目录

2. **配置文件分开存放**：

- 每个小组有自己独立的配置目录：如`/home/deepseek/team1_ollama`
- 我把容器内的`/root/.ollama/config`路径映射到各自的小组目录
- 这样每个团队的设置和历史记录都是独立的

为什么这么做?

- 一个70B的模型可能占用40GB存储空间
- 如果10个团队各自下载一份，那就是400GB!
- 而且下载时间也很长
- 我的方案：模型只下载一次，大家一起用，但配置各自独立

文件夹安排

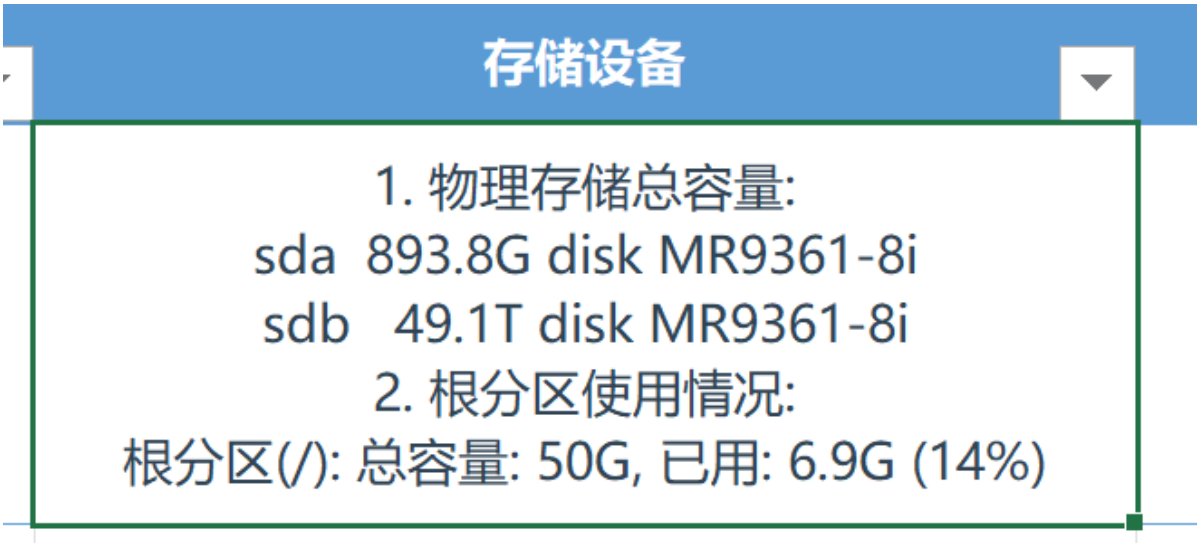
我是这样组织目录的:

```
/home/deepseek/
├── models_shared/      👈 这个文件夹装所有模型，大家一起用
├── team1_ollama/       👈 第1个小组的配置目录
├── team2_ollama/       👈 第2个小组的配置目录
└── ...
```

用这种方式，我们既能共享大模型文件，又能保持各个团队的独立性.

> 不过，可能会面临存储不足的问题。因为大多数服务器的`home`目录空间有限，一般只有800多GB，甚至有些只有100多GB。因此，我们需要根据服务器的存储容量决定是否将模型存储到其他硬盘。例如，A系列服务器的`/home`目录所在的`sda`硬盘只有800多GB，而`sdb`硬盘的容量有49.1TB，但尚未挂载。为了演示的简便起见，我**暂时选择**将模型存储在`home`**目录**下，以后会挂载磁盘，请使用别的共享模型文件夹。

> **更新说明：我已经将磁盘挂载到每个服务器上了，共享模型目录请选择`/models_shared`而不是`/home/deepseek/models_shared`**



5. 实践部署 - 从零到一跑起来

下面我来演示如何我是从头开始部署一个Ollama容器并运行模型。

每台服务器我**已配置docker环境和nvidia container toolkit环境**（除了A02和C01）。如果发现有服务器上没有环境可以来找我。

第一步：使用ssh连接上服务器。

我为每个服务器上创建了一个`deepseek`用户，具有Docker和GPU权限,大家可以使用这个用户来访问服务器。下面我用A01号机做演示。A1的ip地址为222.206.4.160

```
djc@djc:~/gossh$ ssh deepseek@222.206.4.160
deepseek@222.206.4.160's password:
[deepseek@localhost ~]$ ls
deploy_ollama_container.sh  models_shared
[deepseek@localhost ~]$
```

连接后可以看见在当前主目录有一个文件夹`models_shared`和一个脚本`deploy_ollama_container.sh`

脚本是我让**AI辅助**写的,很简单，主要为了方便配置运行ollama容器。

第二步：准备工作

```
# 先创建一个小组的配置目录，后缀表示这个小组使用ollama平台来部署大模型。
mkdir -p /home/deepseek/team1_ollama       # 这是第一个小组的配置目录
#使用命令为脚本授予执行权
chmod +x /home/deepseek/deploy_ollama_container.sh
```

```
[deepseek@localhost ~]$ mkdir -p /home/deepseek/team1_ollama
[deepseek@localhost ~]$ ls
deploy_ollama_container.sh  models_shared  team1_ollama
```

```
[deepseek@localhost ~]$ chmod +x /home/deepseek/deploy_ollama_container.sh
[deepseek@localhost ~]$ ls
deploy_ollama_container.sh  models_shared  team1_ollama
```

第三步：使用脚本部署容器

> 脚本实现其实就是使用docker的命令来创建容器，不过就是将其表述的更加易懂了，有了些中文化的输出.你也可以使用**不使用上面的脚本**而如下**docker命令代替**，也就是说你可自己更换平台，不使用ollama。
>
> ```
> docker run -d --name ollama-team1 --cpus=8 --memory=16g --gpus '"device=0"' -p 11434:11434 -v
> /home/deepseek/models_shared:/root/.ollama/models -v /home/deepseek/team1_ollama:/root/.ollama/config
> ollama/ollama:latest
> ```

这里我使用脚本执行以下命令用来做示例：

```
./deploy_ollama_container.sh \
  -n ollama-team1 \            # 给容器取个名字
  -p 11434 \                   # 用这个端口访问(不同小组用不同端口)
  -g 0 \                       # 只用0号GPU
  -c 8 \                       # 只用8个CPU核心
  -m 16g \                     # 分配16GB内存
  --shared-models /home/deepseek/models_shared \  # 共享模型目录
  -s /home/deepseek/team1_ollama                  # 团队配置目录
```

> 这条命令：
>
> - 它告诉Docker：分配多少资源给这个容器
> - 它告诉Ollama：模型和配置文件放在哪里

脚本的参数列表如下：

```
用法: ./deploy_ollama_container.sh [选项]
必需选项:
  -n, --name NAME       容器名称 (必需参数)例: ollama-team1
  -p, --port PORT       映射端口 (必需参数)例: 11434
  -g, --gpus GPU_IDS    GPU设备ID (如 '0' 或 '0,1' 或 'all', 必需参数)如0
可选选项:
  -c, --cpu CPU_CORES   CPU核心数量 (默认: 8)
  -m, --memory MEMORY   内存限制 (如 '8g', 默认: 16g)
  -v, --volume NAME     数据卷名称 (默认: {容器名}-data)
  -i, --image IMAGE     镜像名称 (默认: ollama/ollama:latest)
  --ip IP_ADDRESS       监听的IP地址 (默认: 0.0.0.0, 表示所有网络接口)
  -s, --storage PATH    数据存储绑定路径 (默认: 不使用数据目录而使用Docker卷)
  --shared-models PATH 共享模型目录路径 (需与-s/--storage一起使用)
  -h, --help            显示帮助信息
```

脚本会自动完成工作，包括检查环境、拉取镜像、创建容器等。

第四步：使用容器运行大模型

成功了可以看到如下输出，你可以打开下面的api看看。

```
容器 ollama-team1 正在运行
容器详情:
表ID:   9b2bc683bcdf
名称:   ollama-team1
状态:   Up 2 seconds
端口:   0.0.0.0:11434->11434/tcp, :::11434->11434/tcp

==== Ollama容器部署完成 ====
使用说明:
1. 访问Ollama API: http://222.206.4.160:11434
2. 查看容器日志: docker logs ollama-team1
3. 进入容器: docker exec -it ollama-team1 /bin/bash
4. 停止容器: docker stop ollama-team1
5. 启动容器: docker start ollama-team1
6. 删除容器: docker rm ollama-team1

您使用了共享模型配置:
- 模型存储在: /home/deepseek/models_shared
- 配置存储在: /home/deepseek/team1_ollama
- 模型只需下载一次，多个容器可共享使用
- 下载模型: docker exec ollama-team1 ollama pull MODEL_NAME
```

```
# 如果有问题，可以查看日志
docker logs ollama-team1
```

> 这里需要注意的是，将容器关闭后重新启动时要使用**docker start**命令而不是docker run，run是根据镜像重新创建一个容器再跑起来，就不是原来的容器了。

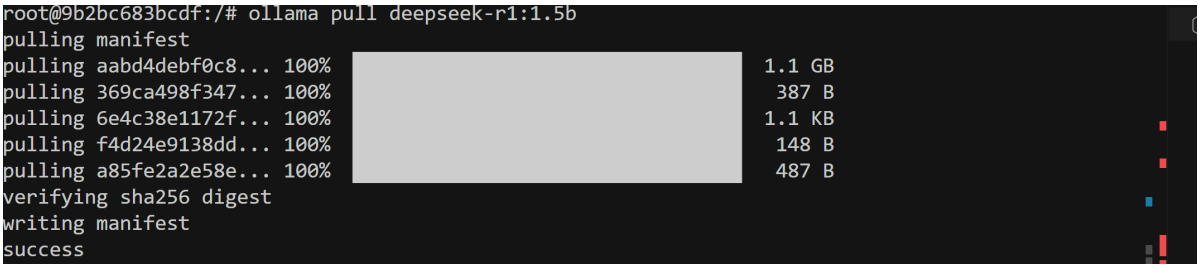
一切正常的话，你会看到容器已经在运行了！

下载并运行一个小模型

> ollama支持的模型可以在官网上找到，你可以自由选择，但是请注意资源分配的是否足够。

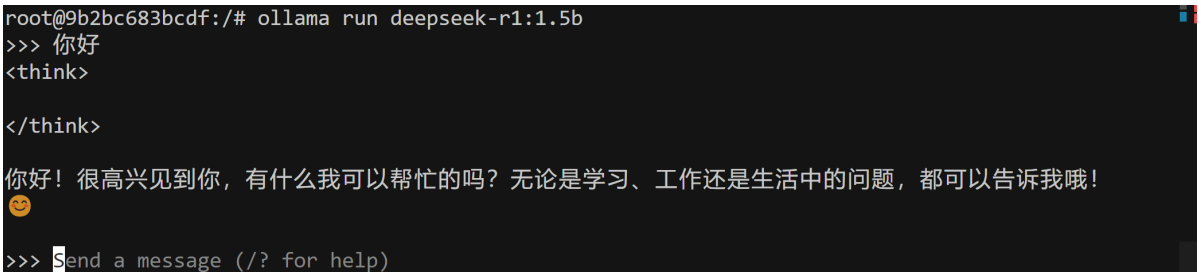
在这里我先尝试1.5b的小模型，看看能不能正常工作：

```
#进入容器执行下载一个1.5B的代码模型
docker exec -it ollama-team1 /bin/bash
ollama pull deepseek-r1:1.5b

# 或者在容器外直接使用这条命令
docker exec -it ollama-team1 ollama pull deepseek-r1:1.5b
```



下载完成后，我们就可以用它了：

```
#在容器内，进入交互模式，就像ChatGPT那样对话
ollama run deepseek-r1:1.5b

# 或在容器外
docker exec -it ollama-team1 ollama run deepseek-r1:1.5b
```



或者，用API方式调用它：

```
# 用API让模型生成代码
POST http://222.206.4.160:11434/api/generate
json:
'{
  "model": "deepseek-r1:1.5b",
  "prompt": "写一个Python函数计算斐波那契数列"
}'
```

这里我使用apifox进行了测试：

> 但是以上存在一个安全问题：没有校验机制，因为将容器ip映射到了0.0.0.0端口，任何人都可以通过该服务器公网ip来访问这个模型，所有后续可能需要加上一些校验代理机制。

退出SSH连接

当你完成配置后，使用`exit`退出SSH连接。容器会继续运行，不受影响。

```
root@9b2bc683bcdf:/# exit
exit
[deepseek@localhost ~]$ eixt
bash: eixt: command not found...
[deepseek@localhost ~]$ exit
logout
Connection to 222.206.4.160 closed.
djc@djc:~/gossh$
```

6.给其他小组部署

如果要给第二个小组部署，只需要改一下参数。

> 或者你选择使用不同的平台，不使用ollama这个配置脚本，因为不同的平台跑同样的模型需要的配置不一样，所以这个由需求来决定：

```
./deploy_ollama_container.sh \
  -n ollama-team2 \            # 换个名字
  -p 11435 \                   # 换个端口号
  -g 1 \                       # 用1号GPU
  -c 8 \                       # 也用8个CPU核心
  -m 16g \                     # 也分配16GB内存
  --shared-models /home/deepseek/models_shared \  # 共享同一个模型目录
  -s /home/deepseek/team2_ollama                  # 但用不同的配置目录
```

总结

通过这种方法，成功地：

- 把一台服务器"切"成了多个独立环境
- 让每个小组都能用上分配给他们的资源
- 节省了大量存储空间（模型只存一份）
- 避免了环境配置的痛苦
- 让团队成员可以专注于使用模型，而不是配置环境

> 简单来说就是使用docker跑了一个ollama容器，在这个ollama容器上运行大模型，然后把端口暴露出去