

LDS 实验室服务器使用手册 v1.5

朱宏民
FOR
LDS 实验室

2020 年 7 月 18 日



中国科学技术大学

All Rights © [USTC LDS](#)

目录

1 服务器部署介绍	2
2 如何登陆服务器	2
3 使用服务器第一步: 了解 Docker	2
3.1 了解 Docker	3
3.2 镜像仓库	3
3.3 制作镜像	3
4 使用服务器第二步: 用 Docker 调试代码	5
5 使用服务器第三步: 提交 shell 脚本	6
6 使用服务器第四步: 管理任务	7
7 关于 Anaconda	8
8 公共数据集	8
9 共享存储	8
10 如何拷贝服务器数据	9
11 注意事项	9

1 服务器部署介绍

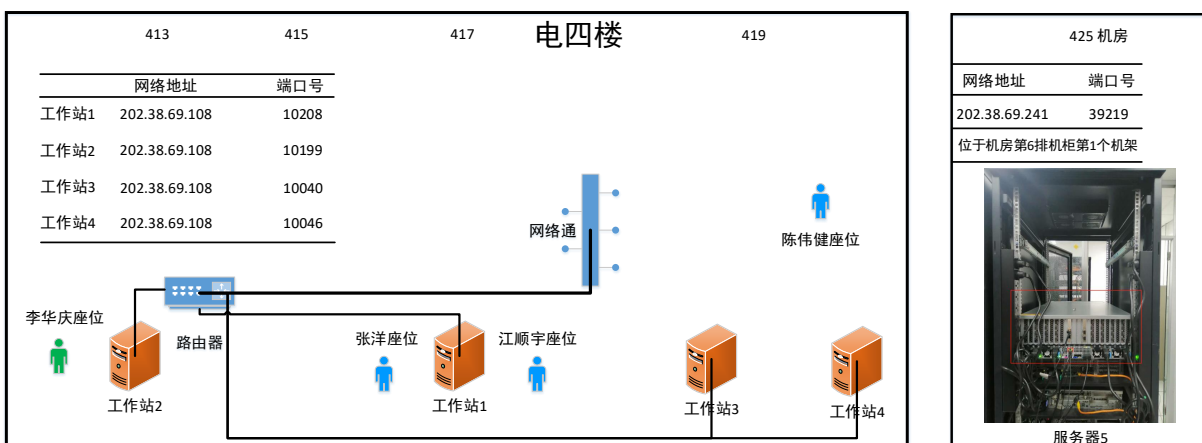


图 1: 实验室 GPU 服务器部署示意图. 一共 24 张显卡.

实验室四台工作站 (i.e. 服务器) 部署情况如图 1 (左) 所示. 四台工作站硬件设备一样: 均配有 7T 左右的机械硬盘, 900G 固态硬盘, 4 张 2080Ti 显卡, 128G 内存, 每张显卡有 11G 显存. 四台工作站都配备 Docker 和 Anaconda 环境.

实验室 8 卡服务器部署在 425 机房, 如图 1 (右) 所示. 硬件配置: 1.0TB 内存, 8 张 2080Ti 显卡, 每张显卡有 11G 显存, 1.8TB 固态硬盘, 500G 共享内存. 同时配备 Docker 和 Anaconda 环境.

使用实验室服务器之前建议: 先学会使用学院的集群, 可以参考[学院集群手册](#).

2 如何登陆服务器

如果你要使用服务器, 请发信息告诉我, 我来给你创建账号和密码. 有了账号和密码之后, Windows 用户可以下载两个 ssh 客户端软件: [下载XShell](#) 和 [下载WinSCP](#). 通过使用这两个软件, 再结合图 1 中提供的网络地址和端口号, 就可以登陆服务器了. 目前, 外网可以访问实验室计算资源.

3 使用服务器第一步: 了解 Docker

服务器使用 Docker 的方式来运行大家的程序, 并且建立了实验室自己的私有镜像仓库.

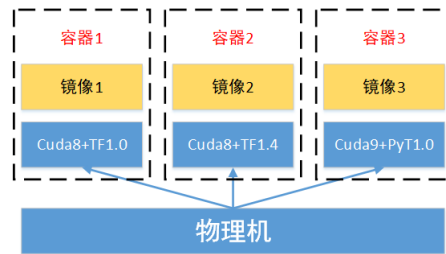


图 2: Docker 运行原理.

3.1 了解 Docker

Docker 的运行方式如图 2 所示.

采用 **Docker** 的目的是: 使得大家的程序互不干扰; 可以使用私有镜像仓库; 方便管理代码; 满足不同用户对软件版本的需求. 比如图 2 中的三个容器, 可以表示三个用户的代码运行在同一台服务器上面.

镜像的作用是: 提供一个特定的深度学习环境, 来运行代码. 比如图 2 中三个镜像提供的深度学习环境不一样. 容器可以理解成虚拟机.

3.2 镜像仓库

```
zhuhm@ubuntu:~$ curl http://192.168.50.199:5000/v2/_catalog
{"repositories":["chwj-py36-torch110-pyg-metis-v3","hello-world","wangyf-py36-tf1.14.0","wangyf_py36_tf.1.14.0","wujc-deepo9-pandas-tf1.12","zhuhm-deep9-xgb-sklearn-pandas","zhuhm-deepo-9-mcc-cluster","zhuhm-deepo-mit-latest","zhuhm-deepo9","zhuhm-torch1.0.1-networkx"]}
```

图 3: 镜像仓库.

在工作站 2 里面, 新建了 **实验室自己的私有镜像仓库: 192.168.50.199:5000**, 可以使用 `curl http://192.168.50.199:5000/v2/_catalog` 命令来查看仓库里面有哪些镜像, 如图 3 所示. 四台工作站都可以访问私有镜像仓库.

私有镜像仓库的作用: 用来存储实验室同学常用的镜像. 类似于学院集群的 bit:5000.

使用镜像常见的问题有:

1. 镜像里面缺少了某一个软件包导致代码不能运行;
2. 镜像里面有代码要求的软件包, 但是这个软件包的版本不对导致代码不能运行.

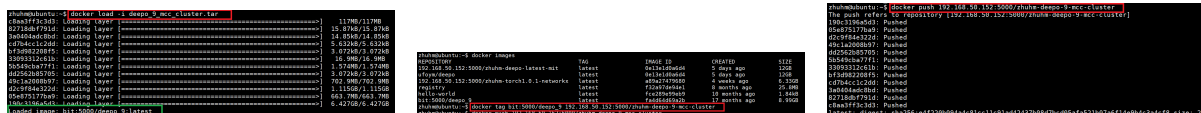
解决办法参考 3.3.

3.3 制作镜像

制作符合自己代码要求的镜像有两种办法:

- (1) 从学院 GPU 集群上面下载符合要求的镜像, 然后再上传到实验室自己的私有镜像仓库里面.
- (2) 写 Dockerfile 文件来制作想要的镜像.

3.3.1 下载学院 GPU 集群的镜像



(a) 加载 tar 格式镜像文件.

(b) 添加标签.

(c) 上传到实验室私有镜像仓库.

图 4: 制作镜像方法 1 流程.

下载学院 GPU 集群的镜像步骤:

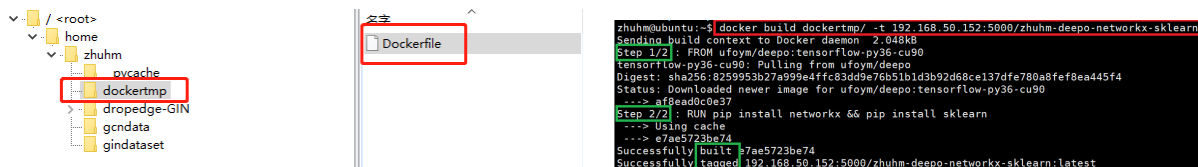
- (1) 在学院 GPU 集群的镜像仓库里面找到想要的镜像.
- (2) 通知学院 GPU 集群管理员帮你把镜像压缩到你自己的目录.
- (3) 使用 WinSCP 把压缩的镜像上传到任意一台工作站.
- (4) 使用 XShell 软件按照图 4 的顺序进行操作: 先解压镜像 tar 文件 4(a), 然后给解压好的镜像添加标签 4(b), 最后把镜像上传到实验室的私有镜像仓库 4(c).

示例: 假设第三步你把 `deepo_9_mcc_cluster.tar` 镜像文件上传到了你的主目录下面, 接下来先按照图 4(a) 加载镜像文件, 加载成功后会提示 `Loaded image: bit:5000/deepo_9:latest`, 表示你已经在工作站上面成功加载了名字叫做 `bit:5000/deepo_9:latest` 的镜像. 接下来, 你需要按照图 4(b) 给刚刚加载的镜像添加标签, 也就是给这个镜像换一个名字, **名字前缀一定要是实验室工作站私有镜像仓库的网络地址**. 添加好标签后, 按照图 4(c) 把换了名字的镜像放到实验室私有镜像仓库里面, 以便在任意一台工作站都可以使用这个镜像. 最后, 你可以使用 3.2 的命令来检查你的镜像是否已经存在于实验室私有镜像仓库里面.

3.3.2 写 Dockerfile

有同学可能会遇到这样的情况: 学院集群的镜像里面少了某些软件包; 学院集群的镜像里面某些软件包版本不对. 这种情况下就只能自己制作镜像了.

Dockerfile 文件的作用: 自定义镜像文件, 用来制作自己想要的镜像.



(a) WinSCP 软件下的 Dockerfile 文件.

(b) 制作镜像.

图 5: 制作镜像方法 2 流程.

示例: 假设原以为可以用的镜像是 `ufoym/deepo:tensorflow-py36-cu90`, 通过正确操作把这个镜像上传到实验室私有镜像仓库, 最后发现这个镜像里面少了两个软件包: `networkx` 和 `sklearn`. 我们可以通过写 Dockerfile 文件来解决上述问题:

FROM `ufoym/deepo:tensorflow-py36-cu90`

RUN `pip install --no-cache-dir networkx sklearn==0.23.1`

解决方法: 首先我们来建立一个文件夹 `dockertmp`, 在这个文件夹里面新建一个文件 `Dockerfile`, 如图 5(a), 然后把上面两行代码写入 `Dockerfile` 文件. 当然你需要根据自己的实际情况来写 `Dockerfile` 代码 (修改 **红色字体**). 接下来如图 5(b) 制作镜像, 其中 `-t` 后面的参数是给这个镜像取的名字. 制作完成后, 把你制作的镜像放到实验室私有镜像仓库里面.

注意事项: 不管是给镜像添加标签 (tag), 还是把镜像放到实验室私有镜像仓库 (push), 还是使用镜像, 都要添加 `192.168.50.199:5000/` 标识. 类似于学院集群的 `bit:5000/`.

到目前为止, 我们已经学会用两种方法制作镜像了, 那么我们要怎么来使用实验室私有镜像仓库里面的镜像来跑代码呢? 请参考 4 和 5.

4 使用服务器第二步: 用 Docker 调试代码

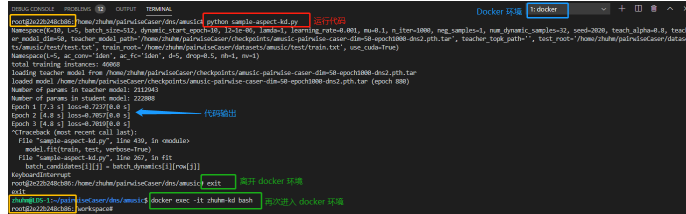
很多同学希望可以通过软件 (PyCharm, VSCode) 远程连接服务器, 然后在 Windows 电脑上面通过 UI 界面形式来调试代码, 并且希望对代码的修改可以直接保存在服务器里面, 从而同步更新代码文件.

VSCode 软件使用 Docker 调试代码的步骤如下 (示例):

- (1) 开一个 Docker 容器, 容器里面的深度学习环境可以用来跑代码. 如图 6(a) [红色], 使用镜像 `192.168.50.199:5000/zhuhm-torch1.0.1-networkx` 来生成容器, 容器名叫 `zhuhm-kd`, 挂载 `/home/zhuhm/` 目录, 使用 2 号 GPU 显卡 (进入容器后, GPU 编号从 0 开始).
- (2) 如图 6(a) [红色], 使用 `docker ps -a` 来查看已开容器的基本信息.
- (3) 为了跑代码, 需要进入到容器 `zhuhm-kd` 里面去, 如图 6(a) [绿色]. 进入容器后, 切换到代码目录.



(a) 进入 Docker 容器。



(b) 运行代码。

图 6: VSCode 软件使用 Docker 远程调试代码。

- (4) 如图 6(b) [红色], 因为此时已经进入到深度学习环境中, 所以使用 python 运行代码即可。
- (5) 如图 6(b) [绿色], 离开调试环境和再次调试。
- (6) 调试过程如果关闭客户端, 代码会断掉。解决方法: 可以使用 nohup & 让代码挂在后台运行。

5 使用服务器第三步: 提交 shell 脚本

示例: 假设要运行的代码 main.py 在目录 /home/zhuhm/GIN-reproduce/bio/mutag/ 下面, 使用实验室私有镜像仓库里面的镜像 zhuhm-torch1.0.1-networkx, 另外想程序的输出 (包括正确输出信息和错误输出信息) 保存在目录 /home/zhuhm/GIN-reproduce/bio/mutag/ 下面的 output.log 文件里面, 此外想让程序运行在工作站第一张 GPU 卡上面, 最后提交程序后想让 Docker 容器 (使用镜像跑代码会生成一个 Docker 容器) 的名字叫做 mutag. 我们可以写如下所示的 shell 脚本 (shell 脚本的名字是 docker.sh):

```
#!/bin/bash
jobname='mutag'
mountpath='/home/zhuhm/'
imagename='192.168.50.199:5000/zhuhm-torch1.0.1-networkx'
codepath='/home/zhuhm/GIN-reproduce/bio/mutag/main.py'
stdoutpath='/home/zhuhm/GIN-reproduce/bio/mutag/output.log'
docker run -itd --rm --gpus '"device=0"' --name $jobname -v $mountpath:$mountpath $imagename
/bin/bash -c 'python '$codepath' > '$stdoutpath' 2>&1'
```

你可以根据实际情况修改红色字体就可以了。如果想使用两张卡 (比如卡 1 和卡 3), 只需要改动 "device=1,3"。

关于卡号: 使用 1,3 这两张卡, 代码里面调用卡的时候: 1,3 这两张卡的编号在 Docker 容器里面是 0,1 (Docker 容器里面对卡的编号是从 0 开始的). 如果使用 device 申请了 3 张 GPU 显卡, 那么在 Docker 容器里面的编号就是: 0,1,2.

脚本需要注意的地方: 不要复制粘贴脚本到服务器, 会出现格式转换问题; 脚本从 docker run 到 &' 是一行代码, 不要写成两行; /bin/bash 和 imagename 之间有空格.

docker.sh 脚本的作用: 生成一个 Docker 容器, 容器里面使用特定的镜像来跑代码. 类似于学院集群的 pbs 文件.

使用 Docker 提交代码后, 如何管理代码呢? 请继续往下读.

6 使用服务器第四步: 管理任务

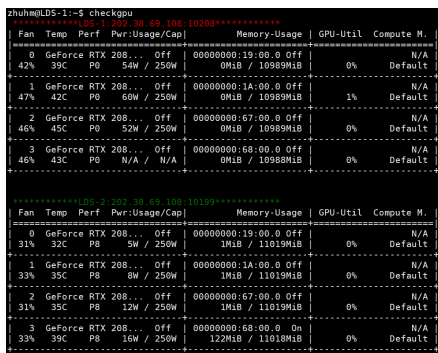
命令	作用	用法
docker ps -a	查看容器	docker ps -a
docker stop	停止容器 (终止任务)	docker stop 容器名
docker images	查看镜像	docker images

表 1: Docker 管理代码.

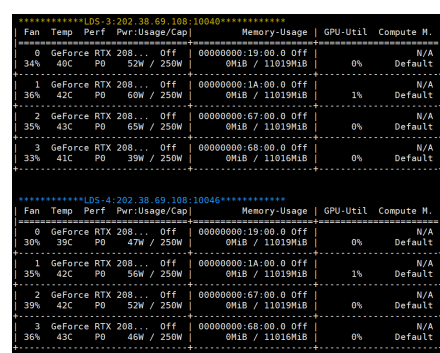
表 1 提供了最常用的 Docker 命令和用法. 另外, 关于 XShell 软件有两个有用的命令: sz 和 rz, 分别用于在服务器和 Windows 电脑之间进行下载和上传文件. 此外, ps -aux 可以查看代码的进程信息.

实验室工作站实现了: 在任意一台工作站上面, 可以观察到所有工作站的 GPU 显卡使用情况, 如图 7, 便于大家寻找合适的 GPU 显卡跑代码.

用法: checkgpu



(a)



(b)

图 7: 观察四台工作站 GPU 显卡情况.

7 关于 Anaconda

有不少同学希望使用 Anaconda 环境. 所有服务器上面都安装了单独的 Anaconda 环境. 并且每个用户都已经加入了 Anaconda 环境变量. Tips: 可以在 Docker 镜像里面安装 Anaconda.

Anaconda 的缺点: 四台工作站不能共享用户创建的 conda 虚拟环境.

8 公共数据集

在工作站 1 上面建立了公共数据集, 目录在 /home/PublicDataset/, 用于存放一些比较适合科研的数据集. 每个数据集文件夹里面都要有数据集说明文件 README. 可以使用 cat README 查看数据集介绍. 也可以下载到本地研究.

9 共享存储

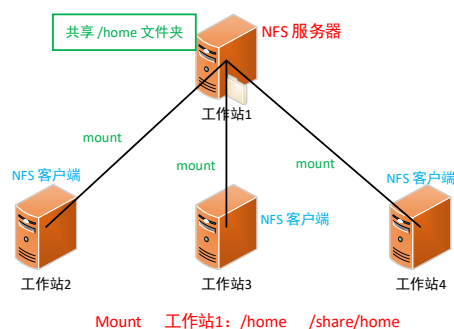


图 8: 共享存储示意图.

目前, 四台工作站通过 NFS 技术实现了共享存储. 工作站 1 作为 NFS 服务器, 向其他三台工作站提供共享文件夹: /home. 工作站 2, 工作站 3 和工作站 4 可以访问到工作站 1 上面的文件, 方法是在工作站 2, 工作站 3 和工作站 4 上面使用命令:

```
cd /share/home/<username>
```

就可以切换到用户自己的共享文件夹里面.

共享存储的作用: 方便大家管理自己的代码 (代码可以都放在工作站 1 里面).

NFS 原理: NFS 服务器提供共享文件夹, NFS 客户端通过挂载 (mount) 和 RPC 协议获得共享文件夹.

10 如何拷贝服务器数据

如果通过软件传输大数据速度太慢, 可以用硬盘直接从服务器拷贝出来. 具体步骤如下 (示例):

- (1) 首先把硬盘插到服务器 USB 接口.
- (2) 然后通过命令行挂载硬盘: `mount /dev/sdc/ /mnt/`
- (3) 挂载好后, 进入到挂载目录查看: `cd /mnt/`
- (4) 如果没问题, 就可以拷贝文件了, 使用 `cp` 或者 `mv` 命令.
- (5) 最后, 拷贝好后, 解除挂载: `umount -l /mnt/`, 拔出硬盘.

11 注意事项

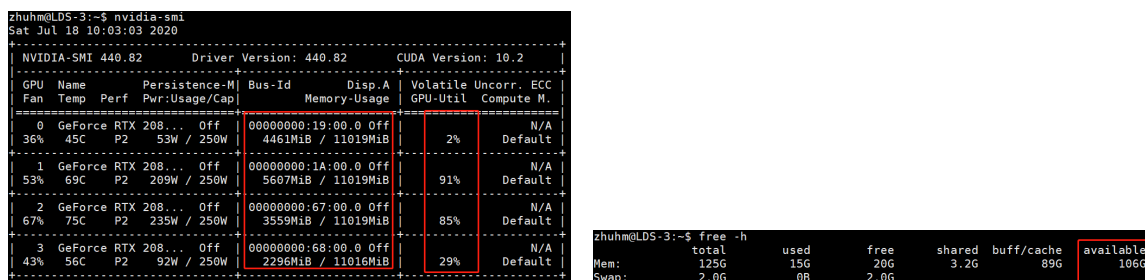


Figure 9 consists of two terminal screenshots. Screenshot (a) shows the output of the `nvidia-smi` command, displaying GPU usage for four RTX 2080 cards. Screenshot (b) shows the output of the `free -h` command, displaying system memory usage.

GPU	Name	Temp	Perf	Pwr:Usage/Cap	Bus-Id	Disp-A	Memory-Usage	GPU-Util	Compute M.
0	GeForce RTX 208...	36%	45C	P2 53W / 250W	Off	00000000:19:00.0	Off	2%	N/A
1	GeForce RTX 208...	53%	69C	P2 208W / 250W	Off	00000000:1A:00.0	Off	91%	N/A
2	GeForce RTX 208...	67%	75C	P2 235W / 250W	Off	00000000:67:00.0	Off	85%	N/A
3	GeForce RTX 208...	43%	56C	P2 92W / 250W	Off	00000000:68:00.0	Off	29%	N/A

	total	used	free	shared	buff/cache	available
Mem:	125G	15G	20G	3.2G	89G	106G
Swap:	2.0G	0B	2.0G			

(a) 查看显卡.

(b) 查看内存.

图 9: 显存和内存.

- (1) 请大家在跑代码之前使用 `nvidia-smi` 来查看 GPU 显卡的使用情况 (如图 9(a), 主要查看显存和 GPU 利用率), 然后再决定使用哪些 GPU 显卡.
- (2) 请大家尽量不要在自己的目录下载大型软件, 因为这会消耗硬盘空间.
- (3) 请大家运行消耗内存的程序之前, 使用 `free -h` 来查看可用内存, 如图 9(b).
- (4) 不使用的 Docker 容器要用 `docker rm` 命令及时删除, 否则会占用显存.
- (5) 自己制作镜像的时候, 命名方式应该能够简洁清楚地表达这个镜像的内容.
- (6) 在 GPU 资源非常紧缺的情况下, 请体谅赶 ddl 的同学.