Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

使用远程服务器搭建深度学习开发环境

尹培宇　硕士研究生

2020年2月16日

内容提要

- 虚拟机和容器
- 深度学习环境的搭建
- 其他可用深度学习远程开发策略

预期收获

- 虚拟机和容器的原理和使用
- 基于TensorFlow的深度学习环境搭建
- 常见的深度学习远程开发策略

虚拟机和容器

虚拟化（Virtualization）

- 什么是虚拟化技术
  - 将计算元件和硬件隔离开来，隐藏底层的硬件物理特性，为用户提供抽象、统一的模拟计算环境。
- 为什么要使用虚拟化技术
  - 为了实现隔离性、可扩展性、安全性、资源更充分利用。
- 如何实现虚拟化技术
  - 把硬件资源虚拟化，为应用程序模拟底层环境

虚拟化技术的分类

- 按应用场景分类
  - 操作系统虚拟化
  - 应用虚拟化
  - 桌面虚拟化
  - 存储虚拟化
  - 网络虚拟化
- 按照应用模式分类
  - 一对多：其中将一个物理服务器划分为多个虚拟服务器，这是典型的服务器整合模式。
  - 多对一：其中整合了多个虚拟服务器，并将它们作为一个资源池，这是典型的网格计算模式。
  - 多对多：将前两种模式结合在一起。

虚拟化技术的分类

- 按硬件资源调用模式分类
  - 全虚拟化
    - 虚拟化操作系统与底层硬件完全隔离。由中间的 Hypervisor 层转化虚拟化客户操作系统对底层硬件的调用代码，全虚拟化无需更改客户端操作系统，兼容性好。典型代表有：Vmware Workstation、KVM。
  - 半虚拟化
    - 在虚拟客户操作系统中加入特定的虚拟化指令，通过这些指令可以直接通过 Hypervisor 层调用硬件资源，免除有 Hypervisor 层转换指令的性能开销。半虚拟化的典型代表 Microsoft Hyper-V、Vmware 的 vSphere。
  - 注：针对 IO 层面半虚拟化要比全虚拟化要好，因为磁盘 IO 多一层必定会慢。

硬件层面的虚拟化

- 根据实现方式不同，可以分为硬件层面的虚拟化和软件层面的虚拟化
- 硬件层面的虚拟化
  - 通过模拟硬件的方式来获得真实计算机的环境，可以运行一个完整的操作系统。又有Full Virtualization、Partial Virtualization和Paravirtualization等不同的实现方式。
  - 在硬件虚拟化的层面，现代虚拟化技术通常是全虚拟和半虚拟的混合体。常见的虚拟化技术例如VMWare、Xen和KVM都同时支持全虚拟化和半虚拟化。
  - 一般会提供虚拟机，都独立的运行着一个完整的操作系统，这样在同一台物理宿主机上存在大量相同或者相似的进程和内存页，从而导致较大的性能损耗
  - 因此，硬件虚拟化也被称为重量级虚拟化，在同一宿主机上能够同时运行的虚拟机数量相当有限。

硬件层面的虚拟化

- 硬件层面的虚拟化
  - 这种技术历史悠久，比较成熟(发展了40 多年)，但是也具有一些问题
    - 在虚拟机上运行了一个完整的操作系统(GuestOS)，在其下执行的还有虚拟化层和宿主机操作系统，一定比直接在物理机上运行相同的服务性能差
    - 有 GuestOS 的存在，虚拟机镜像往往有几个 G 到几十个 G，占用的存储空间大，便携性差
    - 想要使用更多硬件资源，需要启动一台新的虚拟机。要等待 GuesOS 启动，可能需要几十秒到几分钟不等。

硬件层面的虚拟化

- 硬件层面的虚拟化
  - 通过虚拟机监视器将硬件设备虚拟化，向客户机提供模拟的硬件设备。
  - Hypervisor可以直接运行在裸机上(Xen、VMware EXSi)，也可以运行在操作系统上(KVM、VMware Workstation)。

类型I
bare-metal hypervisors

类型II
Hosted hypervisors

软件层面的虚拟化

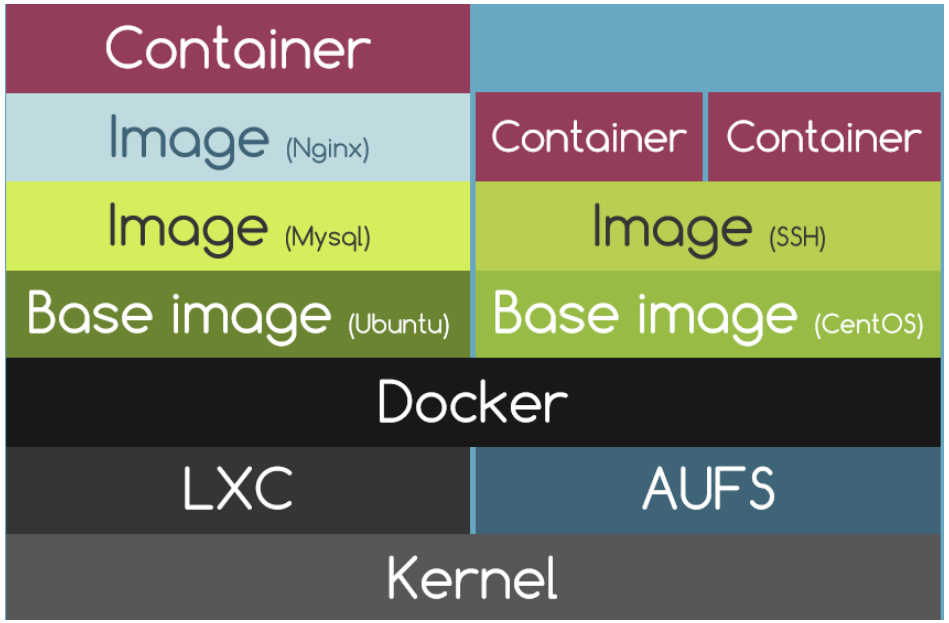
- 软件层面的虚拟化
  - 指在同物理服务器上提供多个隔离的虚拟运行环境，也被称为容器技术。
  - 同一宿主机上的所有虚拟机（又称Container）共享宿主机的操作系统实例，不存在由于运行多个操作系统实例所造成的性能损耗。
  - 因此，软件虚拟化也被称为轻量级虚拟化，在同一宿主机上能够同时运行的虚拟运行环境数量比较宽松。

软件层面的虚拟化

- 软件层面的虚拟化
  - 容器是没有 GuestOS 的轻量级虚拟机，多个容器共享一个 OS 内核。
  - 由于共享操作系统内核，所以容器依赖于底层的操作系统。各个操作系统大都有自己的容器技术和容器工具。
- 容器的实现方式
  - 基于Linux 内核提供的两个机制 Cgroups(实现资源按需分配)和 Namespace(实现任务隔离)，实现进程和资源的隔离和控制。
  - Linux 容器工具有很多，OpenVZ、LXC/ LXD 、Docker、Rocket、Lmctfy
- Docker
  - Docker是在 LXC （Linux Container）的基础上进一步的封装，将应用和其依赖环境全部打包到一个单一对象中。
  - 轻量、可移植性好，可以实现一次打包， 多处部署。

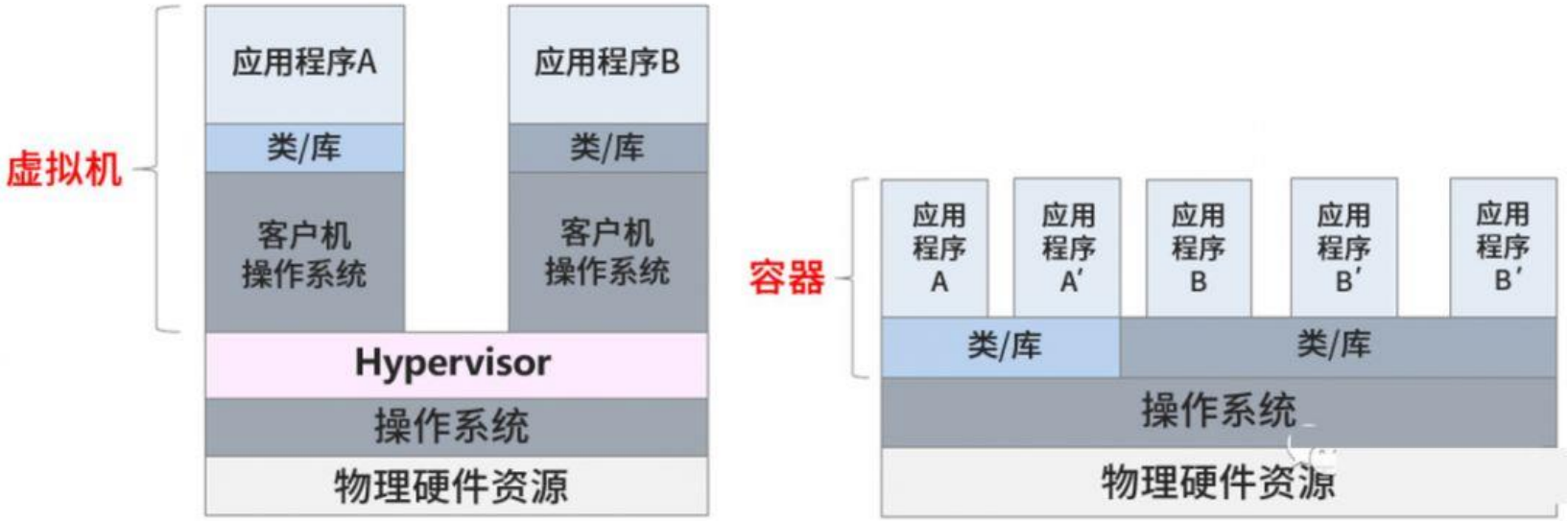
LXC和Docker的比较

- LXC和Docker的比较
  - LXC是操作系统容器，Docker是一个应用程序容器。
  - LXC移植性差，而Docker可以方便地跨机器甚至跨平台移植。



虚拟机和容器的比较

- 两种虚拟化技术的对比
  - 目的都是为了创造“隔离环境”。虚拟机是操作系统级别的资源隔离，而容器本质上是进程级的资源隔离。
  - 虚拟机由于有 VMM 的存在，虚拟机之间、虚拟机和宿主机之间隔离性很好。而容器之间公用宿主机的内核，共享系统调用和一些底层的库，隔离性相对较差;
  - 虚拟机由于有 GuestOS 存在，可以和宿主机运行不同 OS，而容器只能支持和宿主机内核相同的操作系统



虚拟机和容器的比较

- 两种虚拟化技术的对比
  - 容器比虚拟机明显更轻量级，对宿主机操作系统而言，容器就跟一个进程差不多。因此容器有着更快的启动速度(秒级甚至更快)，更高密度的存储和使用(镜像小)、更方便的集群管理等优点。同时由于没有 GuestOS 存在，在容器中运行应用和直接在宿主机上几乎没有性能损失，比虚拟机明显性能上有优势。

特性	虚拟机	容器
隔离级别	操作系统级	进程级
隔离策略	Hypervisor	CGroups
系统资源	5~15%	0~5%
启动时间	分钟级	秒级
镜像存储	GB-TB	KB-MB
集群规模	上百	上万
高可用策略	备份、容灾、迁移	**弹性、附在、动态**

虚拟机和容器的选择

- 云计算的层次
  - SaaS: Software-as-a-Service（软件即服务）
  - PaaS: Platform-as-a-Service（平台即服务）
  - IaaS: Infrastructure-as-a-Service（基础设施即服务）

  - 虚拟化是云计算构建资源池的一个主要方式。

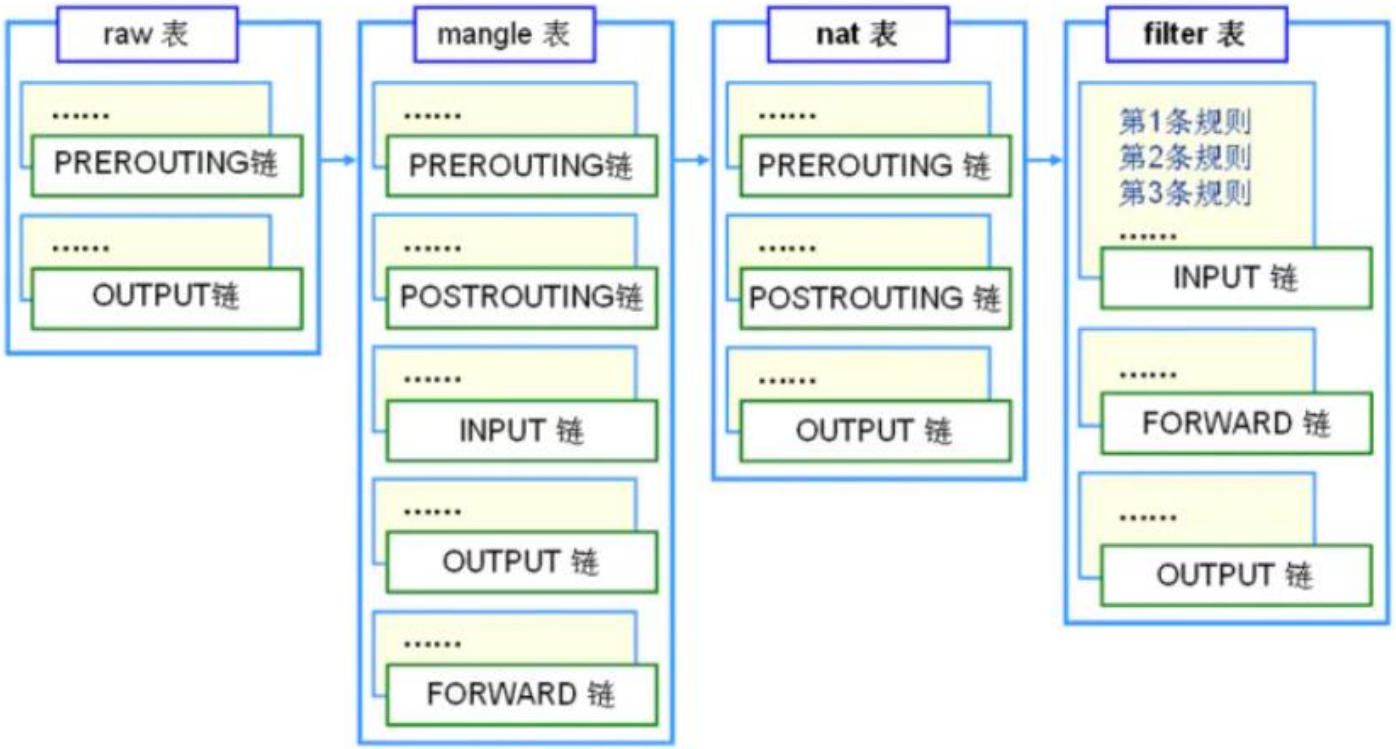
实验室的LXC使用

- 选择
  - 尽可能地建立独立的操作系统环境，尽可能地高效利用资源
  - 选择使用LXC/LXD管理容器。
- 使用
  - 在服务器实体机上，建立归属于每个用户的容器。
  - 实际使用过程中，通过端口映射，通过主机的端口，访问内部的容器。
- 实现
  - 不同进程之间通过不同端口进行区分，实现网络通信
  - 当我们想要访问的容器位于宿主服务器内部时，无法通过ip地址直接访问
  - 可以通过端口映射的方式，将主机某端口（如3011）的流量转发到某个容器的特定端口（如22），从而实现访问内部容器

iptables的使用

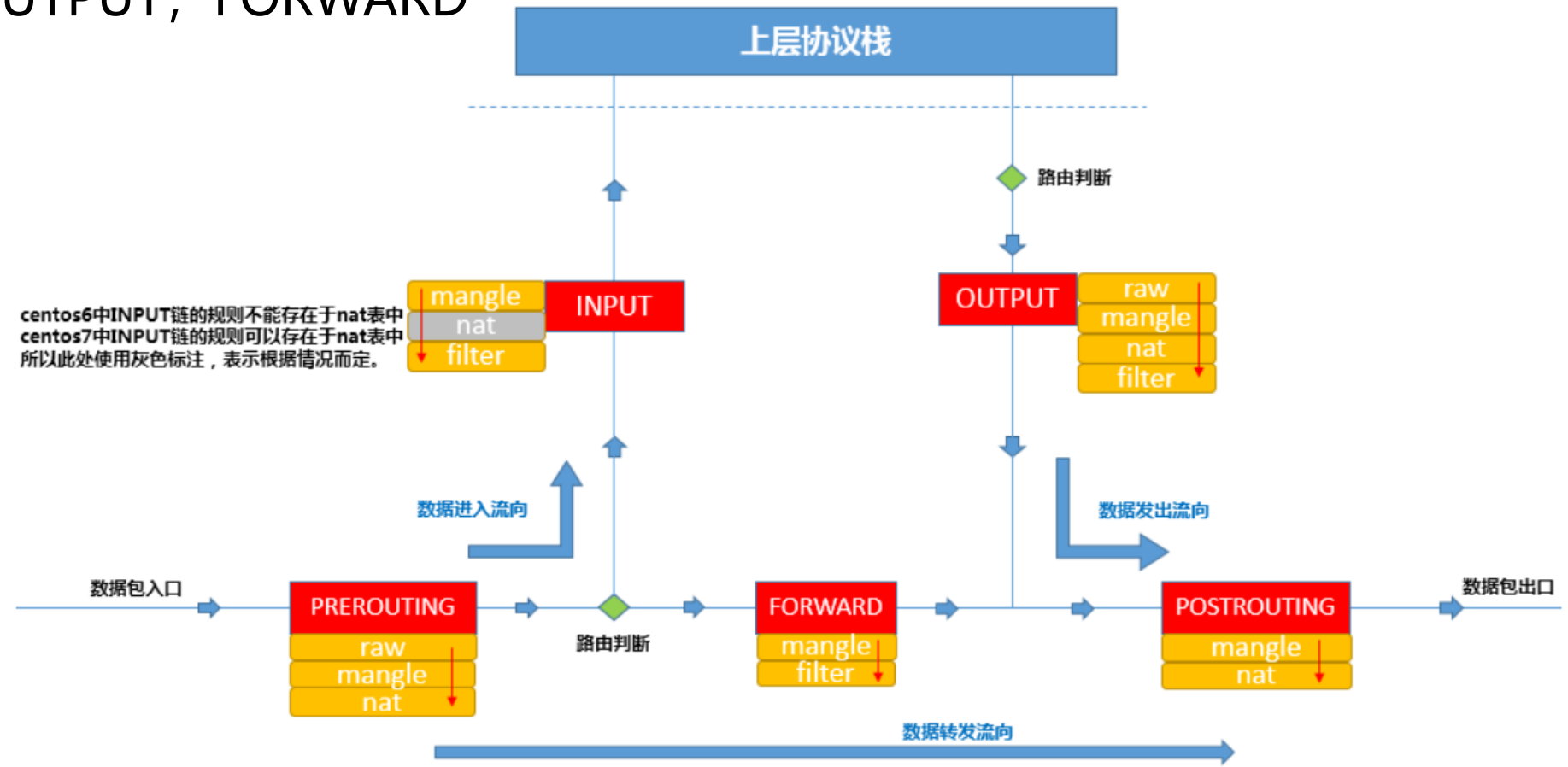
- iptables防火墙
  - 可以用于创建过滤与NAT（Network Address Translation）规则。
  - 依次由 规则->链->表 三级结构构成

  - 根据功能分为4个表
  - Filter：过滤数据包
  - Nat：网络地址转换
  - Mangle：修改报文
  - Raw：是否启用连接追踪机制



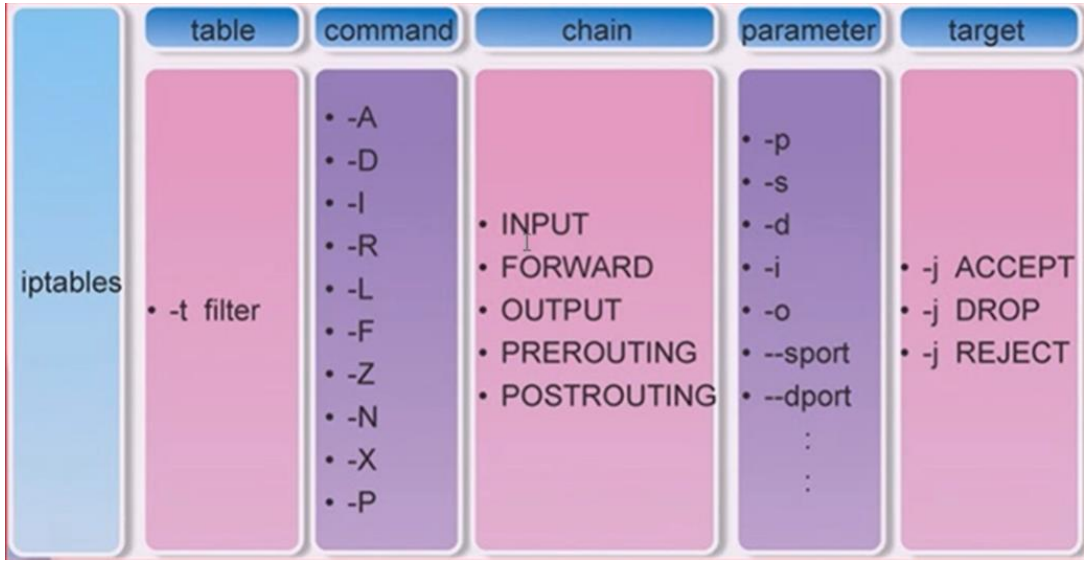
iptables的使用

- iptables防火墙的数据流向
  - 根据数据的流向分为5个链：PREROUTING，POSTROUTING，INPUT，OUTPUT，FORWARD



iptables的使用

- Iptables命令



- 使用iptables –h 查看相关说明
- 一个例子
- iptables -t nat -A PREROUTING -i eno1 -p tcp --dport 3020 -j DNAT --to 10.0.3.120:22

深度学习环境的搭建

深度学习环境的搭建

- 深度学习环境的搭建流程
  - 远程服务器获取
  - 远程服务器连接
  - 确认显卡驱动，CUDA，cuDNN版本依赖关系
  - 安装CUDA
  - 安装cuDNN
  - 安装TensorFlow
  - Jupyter Notebook使用

深度学习环境

- CUDA（Compute Unified Device Architecture）
  - 显卡厂商NVIDIA推出的通用并行计算架构，可以理解为是一个并行计算平台和编程模型。
  - NVIDIA CUDA Toolkit 基本上就是能最大程度利用英伟达 GPU 的应用和程序的开发环境。能够使得使用GPU进行大量并行计算变得简单和优雅。
  - GPU 加速的 CUDA 库支持跨多个域的嵌入式加速，包括线性代数、图像和视频处理、深度学习以及图形分析。
  - 开发人员仍然使用熟悉的C，C ++，Fortran等语言进行编程，并以一些基本关键字的形式对这些语言进行扩展。
  - 这些关键字使开发人员方便地利用GPU实现并行计算。

深度学习环境

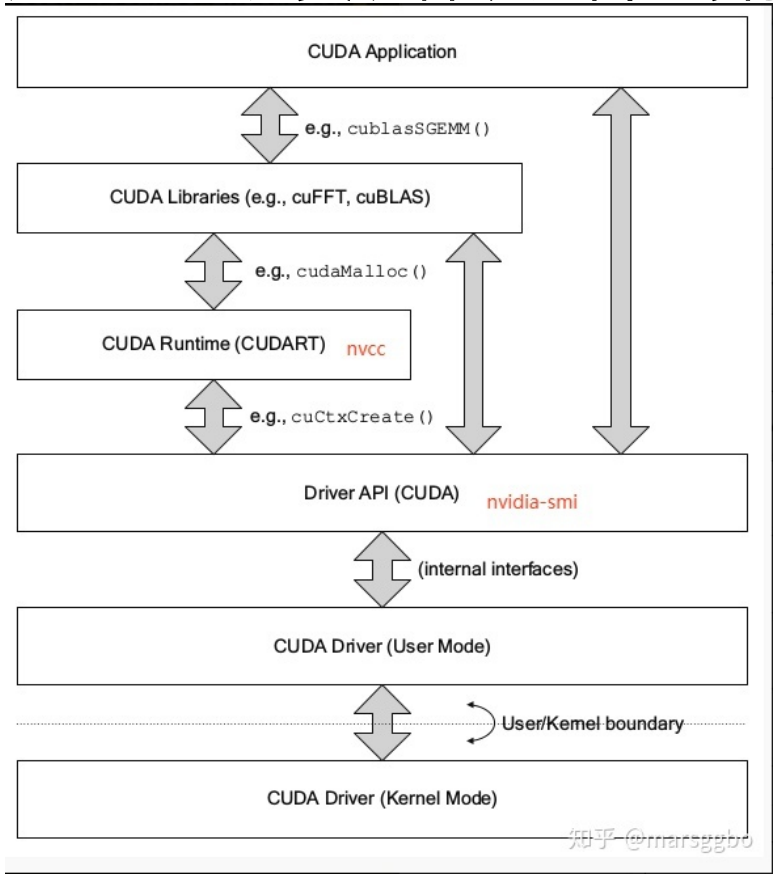
- cuDNN（CUDA Deep Neural Network library)
  - cuDNN （ CUDA深度神经网络库）是用于深度神经网络的GPU加速库。
  - 它强调性能、易用性和低内存开销。
  - 为神经网络中的标准例程提供了高度优化的实现，包括正向和反向卷积、池化、归一化和激活层。深度学习从业者可以依赖 cuDNN 加速在 GPU 上实现高性能现代并行计算。

  - 从官方安装指南可以看出，只要把cuDNN文件复制到CUDA的对应文件夹里就可以，即是所谓插入式设计，把cuDNN数据库添加CUDA里，cuDNN是CUDA的扩展计算库，不会对CUDA造成其他影响

深度学习环境

- cuDNN（CUDA Deep Neural Network library)
  - 从图中可以看到，还有很多其他的软件库和中间件，包括实现c++ STL的thrust、实现gpu版本blas的cublas、实现快速傅里叶变换的cuFFT、实现稀疏矩阵运算操作的cuSparse等等

GPU Computing Applications						
Libraries and Middleware						
cuDNN TensorRT	cuFFT, cuBLAS, cuRAND, cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL, SVM, OpenCurrent	PhysX, OptiX, iRay	MATLAB Mathematica
Programming Languages						
C	C++	Fortran	Java, Python, Wrappers	DirectCompute	Directives (e.g., OpenACC)	
CUDA-enabled NVIDIA GPUs						
Turing Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier	GeForce 2000 Series	Quadro RTX Series	Tesla T Series		
Volta Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier			Tesla V Series		
Pascal Architecture (Compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series		
Maxwell Architecture (Compute capabilities 5.x)	Tegra X1	GeForce 900 Series	Quadro M Series	Tesla M Series		
Kepler Architecture (Compute capabilities 3.x)	Tegra K1	GeForce 700 Series GeForce 600 Series	Quadro K Series	Tesla K Series		
	EMBEDDED	CONSUMER DESKTOP, LAPTOP	PROFESSIONAL WORKSTATION	DATA CENTER		



深度学习的环境依赖

- CUDA对显卡驱动的版本依赖

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 10.1 (10.1.105 general release, and updates)	>= 418.39	>= 418.96
CUDA 10.0.130	>= 410.48	>= 411.31
CUDA 9.2 (9.2.148 Update 1)	>= 396.37	>= 398.26
CUDA 9.2 (9.2.88)	>= 396.26	>= 397.44
CUDA 9.1 (9.1.85)	>= 390.46	>= 391.29
CUDA 9.0 (9.0.76)	>= 384.81	>= 385.54
CUDA 8.0 (8.0.61 GA2)	>= 375.26	>= 376.51
CUDA 8.0 (8.0.44)	>= 367.48	>= 369.30
CUDA 7.5 (7.5.16)	>= 352.31	>= 353.66
CUDA 7.0 (7.0.28)	>= 346.46	>= 347.62

深度学习的环境依赖

- TensorFlow的版本依赖

版本	Python 版本	编译器	编译工具	cuDNN	CUDA
tensorflow-2.0.0	2.7、3.3-3.6	GCC 7.3.1	Bazel 0.26.1	7.4	10
tensorflow_gpu-1.13.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.19.2	7.4	10
tensorflow_gpu-1.12.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.11.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.10.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.9.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.11.0	7	9
tensorflow_gpu-1.8.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.10.0	7	9
tensorflow_gpu-1.7.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9
tensorflow_gpu-1.6.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9
tensorflow_gpu-1.5.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.8.0	7	9
tensorflow_gpu-1.4.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.5.4	6	8
tensorflow_gpu-1.3.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.4.5	6	8
tensorflow_gpu-1.2.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.4.5	5.1	8
tensorflow_gpu-1.1.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8
tensorflow_gpu-1.0.0	2.7、3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8

深度学习环境搭建

- 向管理员申请个人容器，考虑是否需要预装应用程序
  - 目前存在的预装环境为tensorflow-1.12，python-34，cuda-9.0，cudnn-7.3.1
- 获得容器信息
  - 服务器IP地址（如：10.11.12.13），端口（如：1111），用户名name，密码password
- 利用ssh客户端访问容器
  - ssh name@10.11.12.13 –p 1111
  - 成功访问后，命令行开头应为 `name@XXX`，请确认 XXX 为个人姓名首字母缩写

深度学习环境搭建

- 注意事项
  - 1. 安装 CUDA 前，请严格按照官网教程验证环境，需自行安装所需 CUDA 版本对应支持的 g++，若 CUDA 选用版本较新且支持最新版的 g++，则可使用 ‘sudo apt install build-essential gcc-multilib dkms’ 安装各支持库。
    注意，kernel headers and development packages 已预先安装完毕
  - 2. 如前所述，LXC容器将会共用主机显卡驱动，目前版本为390.48，故在安装 CUDA 时，务必阻止其自动安装其他版本的驱动程序，以 .run 文件安装 CUDA 为例，第一个选项
    - Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 387.26? (y)es/(n)o/(q)uit: n

    应选择no

深度学习环境搭建

- 注意事项
  - 3. 受容器配置限制，目前容器内无法访问 IPv6，意味着 ‘apt’ 等命令将无法解析 IPv6 源，故在下载、更新资源时，可能需登陆校园网账号（北理源一般无需登录）。登陆脚本已预先存储在 bfs 账号根目录下，使用命令为
    - ./connect 上网账号 密码
  - 4. 容器只桥接于服务器的外网网卡（10.15.网段），故在机房断电后，访问地址可能变更，到时管理员将统一通知
  - 5. 容器内的 apt 源已更新为北理源，但个人在使用 conda 等服务时请及时更换源

深度学习环境搭建

- 注意事项
  - 6. CUDA、CUDNN、Anaconda可直接取用
    - 如需使用列表中的安装包，可直接联系管理员本机复制
    - GPU 服务器(10.15.8.75)中，预先存放有
      - 1. cuda_9.0.176_384.81_linux.run
      - 2. cudnn-9.0-linux-x64-v7.3.1.20.tgz
      - 3. Anaconda3-5.1.0-Linux-x86_64.sh
  - 7.常用的远程文件传输命令scp，能实现基于ssh的文件安全传输

```
bfs@ubuntu:~$ scp Installer/* bfs@10.0.3.118:
bfs@10.0.3.118's password:
Anaconda3-5.1.0-Linux-x86_64.sh                 100%  551MB 183.7MB/s   00:03
cuda_9.0.176_384.81_linux.run                   100% 1567MB 120.6MB/s   00:13
cudnn-9.0-linux-x64-v7.1.tgz                    100%  388MB 193.9MB/s   00:02
cudnn-9.0-linux-x64-v7.3.1.20.tgz               100%  345MB 115.0MB/s   00:03
NVIDIA-Linux-x86_64-390.48.run                  100%   78MB  77.6MB/s   00:00
bfs@ubuntu:~$
```

深度学习环境搭建

- GPU 支持的 Tensorflow 安装教程
- 1. CUDA 安装
  - 安装包(https://developer.nvidia.com/cuda-downloads)
  - 教程(https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#pre-installation-actions)
  - Cuda 安装方法（快，稳，建议）：
    - sudo apt update
    -
```
bfs@ypy:~$ sudo apt install build-essential gcc-multilib dkms
```
    - sudo sh cuda_9.0.176_384.81_linux.run

```
Need to get 2,142 kB/64.5 MB of archives.
After this operation, 245 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

```
bfs@ypy:~$ sudo sh cuda_9.0.176_384.81_linux.run
```

深度学习环境搭建

– 阅读license时，按 `q` 跳至结尾

```
Windows platform:

%ProgramFiles%\NVIDIA GPU
Do you accept the previou
accept/decline/quit: acce

Install NVIDIA Accelerate
(y)es/(n)o/(q)uit: n

Install the CUDA 9.0 Tool
(y)es/(n)o/(q)uit: y

Enter Toolkit Location
 [ default is /usr/local/

Do you want to install a
(y)es/(n)o/(q)uit: y

Install the CUDA 9.0 Samp
(y)es/(n)o/(q)uit: y

Enter CUDA Samples Locati
 [ default is /home/bfs ]

Installing the CUDA Toolk
```

```
Installing the CUDA Toolkit in /usr/local/cuda-9.0 ...
Missing recommended library: libGLU.so
Missing recommended library: libX11.so
Missing recommended library: libXi.so
Missing recommended library: libXmu.so

Installing the CUDA Samples in /home/bfs ...
Copying samples to /home/bfs/NVIDIA_CUDA-9.0_Samples now...
Finished copying samples.

===========
= Summary =
===========

Driver:   Not Selected
Toolkit:  Installed in /usr/local/cuda-9.0
Samples:  Installed in /home/bfs, but missing recommended libraries

Please make sure that
 -   PATH includes /usr/local/cuda-9.0/bin
 -   LD_LIBRARY_PATH includes /usr/local/cuda-9.0/lib64, or, add /usr/local/cuda-9.0/lib64 to /etc/ld.so.co
nf and run ldconfig as root

To uninstall the CUDA Toolkit, run the uninstall script in /usr/local/cuda-9.0/bin

Please see CUDA_Installation_Guide_Linux.pdf in /usr/local/cuda-9.0/doc/pdf for detailed information on set
ting up CUDA.

***WARNING: Incomplete installation! This installation did not install the CUDA Driver. A driver of version
 at least 384.00 is required for CUDA 9.0 functionality to work.
To install the driver using this installer, run the following command, replacing <CudaInstaller> with the n
ame of this run file:
    sudo <CudaInstaller>.run -silent -driver

Logfile is /tmp/cuda_install_1643.log
bfs@vpy:~$
```

- Install the CUDA 9.0 S
- (y)es/(n)o/(q)uit: y
- Enter CUDA Samples L
- [default is /home/bfs]:

深度学习环境搭建

- 验证
  - cd NV
  - make
  - cd ./bi
  - ./devic
- 结果中显

```
Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1080 Ti"
  CUDA Driver Version / Runtime Version          9.1 / 9.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 11178 MBytes (11721506816 bytes)
  (28) Multiprocessors, (128) CUDA Cores/MP:     3584 CUDA Cores
  GPU Max Clock rate:                            1633 MHz (1.63 GHz)
  Memory Clock rate:                             5505 Mhz
  Memory Bus Width:                              352-bit
  L2 Cache Size:                                 2883584 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 4 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.1, CUDA Runtime Version = 9.0, NumDevs = 1
Result = PASS
bfs@ypy:~/NVIDIA_CUDA-9.0_Samples/bin/x86_64/linux/release$
```

深度学习环境搭建

- 2. CUDNN 安装
  - 官网自行下载需注册账号并认证，登陆，申请，官网教程
    - cd
    - tar -xzvf cud
    - sudo cp cud
    - sudo cp cud
    - sudo chmod
  - 添加环境变量
    - nano .bashrc
    -
    -
    -
  - save and quit，断开ssh，重新连接

```
bfs@ypy:~$ cd
bfs@ypy:~$ tar -xzvf cudnn-9.0-linux-x64-v7.3.1.20.tgz
cuda/include/cudnn.h
cuda/NVIDIA_SLA_cuDNN_Support.txt
cuda/lib64/libcudnn.so
cuda/lib64/libcudnn.so.7
cuda/lib64/libcudnn.so.7.3.1
cuda/lib64/libcudnn_static.a
```

```
bfs@ypy:~$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
bfs@ypy:~$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
bfs@ypy:~$ sudo chmod a+r /usr/local/cuda/include/cudnn.h
bfs@ypy:~$ nano .bashrc
```

```
# added by Anaconda3 installer
export PATH="/home/bfs/anaconda3/bin:$PATH"

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/extras/CUPTI/lib64
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
```

深度学习环境搭建

- 3. Tensorflow 安装
  - 教程官网

    不同 python 版本的 tf 路径
  - **注意**：官网中，中文版要老于英文版！尽量看英文版教程！*请尽量阅读一手资料*
  - #更改pip源
  - mkdir ~/.pip
  - nano ~/.pip/pip.conf

  - # 添加以下内容
  - [global]
  - trusted-host = mirrors.aliyun.com
  - index-url = https://mirrors.aliyun.com/pypi/simple
  - # 保存并退出

```
[global]
trusted-host = mirrors.aliyun.com
index-url = https://mirrors.aliyun.com/pypi/simple
```

深度学习环境搭建

- 3.Tensorflow 安装
  - # 创建一个名为 tensorflow 的 python3.4 虚拟环境
  - conda create -n tensorflow pip python=3.4

```
bfs@ypy:~$ conda create -n tensorflow pip python=3.4
Solving environment: |
```

  - # 激活虚拟环境，即之后的安装操作，均会将库安装到此虚拟环境内
  - source activate tensorflow
  - (tensorflow)

```
bfs@ypy:~$ source activate tensorflow
(tensorflow) bfs@ypy:~$ pip install --upgrade pip
Looking in indexes: https://mirrors.aliyun.com/pypi/simple
```

  - # 更新 pip
  - pip install --upgrade pip

  - # 对应 python 版本的最新版 GPU 支持 tf 库
  - pip install tensorflow-gpu==1.12

深度学习环境搭建

- 测试
  - 从 shell 中调用 Python，如下所示：
    - (tensorflow)$ python
  - 在 Python 交互式 shell 中输入以下几行简短的程序代码：
  - 如

```
(tensorflow) bfs@ypy:~$ python
Python 3.4.5 |Continuum Analytics, Inc.| (default, Jul  2 2016, 17:47:47)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
2020-02-23 04:16:04.280564: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instruct
ions that this TensorFlow binary was not compiled to use: AVX2 FMA
2020-02-23 04:16:05.866605: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with pr
operties:
name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.6325
pciBusID: 0000:04:00.0
totalMemory: 10.92GiB freeMemory: 10.76GiB
2020-02-23 04:16:05.866680: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu dev
ices: 0
2020-02-23 04:16:06.183112: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect Str
eamExecutor with strength 1 edge matrix:
2020-02-23 04:16:06.183163: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0
2020-02-23 04:16:06.183171: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
2020-02-23 04:16:06.183502: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow dev
ice (/job:localhost/replica:0/task:0/device:GPU:0 with 10405 MB memory) -> physical GPU (device: 0, name: G
eForce GTX 1080 Ti, pci bus id: 0000:04:00.0, compute capability: 6.1)
>>> print(sess.run(hello))
b'Hello, TensorFlow!'
>>>
```

深度学习环境搭建

- Jupyter Notebook 安装
  - 配置jupyter notebook服务端，以实现在本机浏览器中访问容器 python 调试界面，方便代码编写，同时支持虚拟环境
  - 1. 基本环境需求（done）
    - tmux（若无，sudo apt install tmux）
    - anaconda中自带notebook，故只需进一步配置即可
  - 2. 新建notebook配置文件（done）
    - jupyter notebook --generate-config
  - 3. 修改配置文件（vim 或 nano 修改上一步创建的文件，路径会在创建后打印在

```
(tensorflow) bfs@ypy:~$ tmux
```

```
(tensorflow) bfs@ypy:~$ jupyter notebook --generate-config
Overwrite /home/bfs/.jupyter/jupyter_notebook_config.py with default config? [y/N]n
(tensorflow) bfs@ypy:~$ nano .jupyter/jupyter_notebook_config.py
```

    - c.NotebookApp.ip = '0.0.0.0'（done）
    - c.NotebookApp.open_browser = False（done）
    - c.NotebookApp.port = 8888

深度学习环境搭建

- Jupyter Notebook 安装
  - 4. 设置密码（为避免遗忘，建议为 isc123 ）
    - jupyter notebook password
  - 5. 安装notebook对虚拟环境的支持
    - conda install nb_conda
  - 6. 进入虚拟环境（若还未安装虚拟环境，可跳过5、6步，安装后，请补充）
    - source activate 虚拟环境名称
  - 7. 安装该虚拟环境的notebook支持
    - conda install ipykernel
  - 8. 新建虚拟窗口
    - tmux

深度学习环境搭建

- Jupyter Notebook 安装
  - 以下为虚拟窗口常用操作命令
  - # 新建 session
  - tmux new -s 虚拟窗口名称
  - # 切换到指定 session
  - tmux attach -t 虚拟窗口名称
  - # 列出所有 session
  - tmux list-sessions
  - # 退出当前 session，返回前一个 session
  - tmux detach
  - # 杀死指定 session
  - tmux kill-session -t 虚拟窗口名称

深度学习环境搭建

- Jupyter Notebook 安装
  - 9. 在虚拟窗口里执行notebook server
    - jupyter notebook

    可以看到 notebook 已运行!

    此时，在不中断 notebook，退出虚拟窗口的方法为：
    - ctrl + b，然后 d
  - 10. 在任意本地浏览器中访问jupyter notebook
    - 浏览器访问：10.11.12.13:11111
  - 11. 选择 conda env:tensorflow，即新建支持 tf 的 ipy 文件，编写完成后，导出 .py，上传到服务器，在相应虚拟环境下运行，即可
  - **12. 每次用完，进入虚拟窗口，关闭 notebook！减少资源占用或是因调试过程中逻辑不严密导致的资源未释放**

深度学习环境搭建

- Jupyter Notebook 安装
  - 远程访问
    - 当我们在容器中运行jupyter notebook时，需要通过容器的 8888 端口访问进行交互，这时就可以将其映射到主机上的某个端口（如11111），从而通过该端口实现远程访问。
    - 于是可以实现浏览器远程访问：10.11.12.13:11111
  - 常用管理命令
    - 查看使用某端口的进程
    - lsof -i:8888
    - netstat -ap|grep 8090
    - 查看某进程占用的端口
    - netstat -nap|grep 7779

基于云服务的深度学习环境

基于云服务的深度学习环境

- 最小化配置基于云的深度学习环境
  - 使用预配置的基于云的深度学习环境可以快速地开始深度学习工作。以下列出一些常用的基于云端的深度学习服务器供应商，通常可以使用预先安装了流行 ML 框架（如 TensorFlow、PyTorch 或 scikit-learn 等）的计算引擎。

  - Google Colaboratory：https://colab.research.google.com/
  - Paperspace Gradient°：https://www.paperspace.com/gradient
  - FloydHub Workspace：https://www.floydhub.com/product/build
  - Lambda GPU Cloud：https://lambdalabs.com/service/gpu-cloud
  - AWS Deep Learning AMIs：https://aws.amazon.com/machine-learning/amis/
  - GCP Deep Learning VM Images：https://cloud.google.com/deep-learning-vm

基于云服务的深度学习环境

- 建立基于云端的深度学习环境
  - 如果要构建自定义的基于云端的深度学习环境，通常需要选择云供应商，然后创建虚拟服务器，后续的深度学习环境搭建过程基本一致。
  - 一些可供选择的云服务商
    - 利用阿里云容器服务进行深度学习模型开发和训练
    - 腾讯GPU 云服务器
    - 百度GPU 云服务器
    - 以及亚马逊的 AWS、微软的 Azure 和谷歌的 GCP等

道德经

大成若缺，其用不弊。
大盈若冲，其用不穷。
大直若屈。大巧若拙。
大辩若讷。静胜躁，寒
胜热。清静为天下正。

谢谢！