

使用Docker搭建实验室共享GPU服务器

原创

置顶  于 2019-03-18 20:06:50 发布  9214  收藏 72

分类专栏: [经验](#) [机器学习](#) [Linux](#) 文章标签: [Docker](#) [服务器](#) [Ubuntu16.04](#) [GPU](#) [深度学习](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/hangvane123/article/details/88639279>

版权



[经验](#) 同时被 3 个专栏收录

8 篇文章 0 订阅

订阅专栏



[机器学习](#)

3 篇文章 0 订阅

订阅专栏



[Linux](#)

3 篇文章 0 订阅

订阅专栏

目录

引入

背景

服务器配置

方案决策

为什么选择这个方案

安装步骤

概述

接下来要注意的问题

安装Ubuntu16.04 LTS

宿主机换apt源

禁止Linux内核更新

安装显卡驱动

安装Docker

安装nvidia-docker2

加入docker组

Docker换源，换存储路径，限制容器日志大小

安装Shipyards中文版

创建容器

配置容器环境

解决中文乱码

打包为镜像

使用Dockerfile构建镜像

新建容器

附：实验室使用的最新版镜像

尾言

存在的问题

参考

引入

背景

实验室已有一台GPU服务器由学长管理，BOSS新购了一台服务器并希望能够像已有的那个服务器一样，让多人共同使用GPU资源而不相互干扰，同时系统资源分配比较灵活。经过一番考虑和踩坑，终于完美搭建GPU平台，记录下以供后人参考。

服务器配置

1. CPU: I7-9700K
2. 主板: 微星Z390 gaming pro carbon
3. 内存: 海盗船16G DDR4 3200MHZ x4
4. 固态: Intel 760P 1T NVME固态
5. 机械: 希捷2T
6. 显卡: 技嘉2080Ti WF3 x2

方案决策

1. 宿主机选择Ubuntu16.04 LTS，引导方式为UEFI，1T固态挂载点为 `/`，2T机械挂载点为 `/home`。
2. 虚拟机容器选择Docker，为了支持在虚拟机中使用GPU资源，使用nvidia-docker插件。
3. 容器使用bridge方式联网，通过端口映射开放部分端口，与宿主机映射部分共用文件夹以方便管理和文件传输。
4. Docker日常管理使用Shipyard中文版这个Docker Web GUI。

为什么选择这个方案

1. 考虑过LXD，其实第一台服务器就是学长使用LXD搭建的。但是经过一段时间的踩坑，发现LXD生态完全不能和Docker相比，很多问题只能Google甚至无从查找，存储等设置对于笔者这种不太懂Linux的同学也过于复杂（创建存储池？指定已存在的块设备还是新建？zfs loop设备无法自定义存储位置？如何从虚拟机打包镜像？），很多问题不知道该怎么解决也很难搜索到解答，和学长再次交流后决定弃LXD转Docker，同样是踩坑，起码Docker还可以baidu到解答。
2. 相较于通过Linux多用户权限分配/口头约定的方式进行共享，基于容器的方案能够彻底隔离不同用户对环境的干扰，新建和删除已有容器非常简单，用户在容器内可以随便操作，不会影响宿主机的运行和其他用户（但是计算/存储资源仍是共享的，且小概率下单个容器的卡死会影响整个宿主机卡死）
3. 经过调研，Docker只要使用nvidia-docker插件就能够使容器访问GPU，并支持GPU资源的灵活分配。
4. 为了能够充分利用固态和机械，仿照之前那台服务器的做法将机械挂载到 `/home`，之后将docker存储设置到 `/home` 里就可以充分利用机械盘。
5. 为了以后能够更方便地管理服务器，调研了Weave Scope、Shipyard、DockerUI三款产品。由于Weave Scope并非专门的Docker Web UI，DockerUI没有登录体系，功能也少，另外我发现了Shipyard的中文版，脚本安装也非常简单，直接选择Shipyard。

安装步骤

概述

本文详细记述了从安装系统到最后实际运行的全过程，主要包括：

1. 安装Ubuntu16.04 LTS
2. 安装显卡驱动
3. 安装nvidia-docker并配置
4. 编辑Docker容器，生成模板镜像供以后使用

接下来要注意的问题

1. 系统的EFI分区建议分配不少于550M
2. 安装完nvidia-docker前不要配置daemon.json文件，安装时会覆盖
3. 设置apt源时一定要设置对应系统版本的源，否则不会立即报错但会出现各种问题

安装Ubuntu16.04 LTS

记得勾选安装第三方软件

在安装选项那里选择自定义安装，全部设置为主分区，硬盘具体分区为：

硬盘	大小	格式	挂载点
固态	550M	EFI	-
固态	4G	SWAP	-
固态	900G+	EXT4	/
机械	2T	EXT4	/home

安装过程比较简单，但估计是因为从MBR格式转为GPT格式的问题，安装过程一直失败，最后经过一次全默认安装后再自定义安装就可以了。

宿主机换apt源

接下来使用 `vim` 命令，直接在本机显示器上操作的可以将下面带 `vim` 的指令替换为 `gedit`，在本机打开文本编辑器更好操作。

备份删除旧源

```
sudo mv /etc/apt/sources.list /etc/apt/sources.list.bak
```

写入新源

```
sudo vim /etc/apt/sources.list
```

粘贴进去保存退出

清华源的链接参考<https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu/>

```
vim的使用方法：（按 I 进入insert模式，粘贴后按 ESC 退出insert模式，再输入 :wq! 回车强制保存退出）
```

更新

```
sudo apt-get update
```

禁止Linux内核更新

查看已安装的内核

```
sudo dpkg --get-selections | grep linux
```

禁止更新内核

```
sudo apt-mark hold linux-image-x.xx.x-xx-generic
```

以后如需要恢复更新

```
sudo apt-mark unhold linux-image-x.xx.x-xx-generic
```

安装显卡驱动

参考

去官网 <https://www.nvidia.cn/Download/index.aspx?lang=cn> 下载合适驱动.run文件，放到 `~` 或其他目录下

卸载原有驱动（可能由于2080Ti没有默认驱动，我执行后并没有卸载掉什么包）

```
sudo apt-get purge nvidia*
```

禁用nouveau

```
sudo vim /etc/modprobe.d/blacklist-nouveau.conf
```

内容写上

```
blacklist nouveau
options nouveau modeset=0
```

更新

```
sudo update-initramfs -u
```

按 `Ctrl` + `Alt` + `F4` 进入tty4控制台

结束X-window服务

```
sudo service lightdm stop
```

安装驱动

```
cd ~
sudo sh NVIDIA*
```

这里开始会有个警告不用管他，安装完后会提示是否更新X-windows配置要选yes

重启X-window

```
sudo service lightdm start
```

如果没有图像，按 `Ctrl` + `Alt` + `F7` 返回图形界面

检查能否读取到显卡信息

```
nvidia-smi
```

如果安装不成功需要卸载重来，或重启后偶尔出现找不到显卡，运行 `nvidia-smi` 提示 `NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver.` 的情况下（数月后更新了显卡驱动解决），需要重装驱动

```
sudo sh NVIDIA* --uninstall
```

如果装完驱动后，登录系统反复闪退回系统登录界面，则需要卸载重装驱动，在安装驱动时添加 `--no-opengl-files` 参数

```
sudo sh NVIDIA* --no-opengl-files
```

安装Docker

参考

和官网教程一致，主要是改为了国内源改善安装速度

通过https，允许apt使用repository安装软件包

```
sudo apt-get install -y \  
  apt-transport-https \  
  ca-certificates \  
  curl \  
  software-properties-common
```

添加Docker官方GPG key

```
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
```

验证key的指纹

```
sudo apt-key fingerprint 0EBFCD88
```

添加稳定版repository

```
sudo add-apt-repository \  
  "deb [arch=amd64] https://mirrors.aliyun.com/docker-ce/linux/ubuntu \  
  $(lsb_release -cs) \  
  stable"
```

更新apt包索引

```
sudo apt-get update
```

安装最新版本的Docker CE

```
sudo apt-get install -y docker-ce
```

验证Docker CE正确安装（可选，之后更换存储位置后建议将默认位置/var/lib/docker删除）

```
sudo docker run hello-world
```

之后不要急着配置**Docker**，因为安装nvidia-docker会覆盖daemon.json。

安装nvidia-docker2

参考

添加repositories

```
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \  
  sudo apt-key add -  
curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu16.04/amd64/nvidia-docker.list | \  
  sudo tee /etc/apt/sources.list.d/nvidia-docker.list  
sudo apt-get update
```

安装nvidia-docker2并重新载入daemon.json

```
sudo apt-get install -y nvidia-docker2
```

加入docker组

加入docker组，以允许非root用户免 `sudo` 执行 `docker` 命令

```
sudo gpasswd -a 用户名 docker
```

如果不重启并重连ssh客户端的话，需要手动重启服务并刷新docker组成员

```
sudo service docker restart  
newgrp - docker
```

Docker换源，换存储路径，限制容器日志大小

备份daemon.json

```
sudo cp /etc/docker/daemon.json /etc/docker/daemon.json.bak
```

修改daemon.json

```
sudo vim /etc/docker/daemon.json
```

安装完nvidia-docker后默认应该是这样的配置

```
{  
  "runtimes": {  
    "nvidia": {  
      "path": "nvidia-container-runtime",  
      "runtimeArgs": []  
    }  
  }  
}
```

修改默认运行时为nvidia-docker，添加国内源，修改存储位置为 `/home/docker`，限制日志大小后daemon.json为

```
{  
  "default-runtime": "nvidia",  
  "runtimes": {  
    "nvidia": {  
      "path": "nvidia-container-runtime",  
      "runtimeArgs": []  
    }  
  },  
  "registry-mirrors": [  
    "https://kfwkfulq.mirror.aliyuncs.com",  
    "https://2lqq34jg.mirror.aliyuncs.com",  
    "https://pee6w651.mirror.aliyuncs.com",  
    "https://registry.docker-cn.com",  
    "http://hub-mirror.c.163.com"  
  ],  
  "data-root": "/home/docker",  
  "log-opts": { "max-size": "50m", "max-file": "1"}  
}
```

保存退出，并在/home下建立docker文件夹

```
cd /home  
sudo mkdir docker
```

在重启前，建议一并更改下网络设置，否则使用bridge模式连接的Docker容器可能无法访问外网
禁用dnsmasq

```
sudo vim /etc/NetworkManager/NetworkManager.conf
```

加#号注释掉 `dns=dnsmasq`

如果不重启电脑的话需要重启服务以完成更改，不过还是连着Docker一并重启电脑吧

```
sudo restart network-manager
```

```
sudo restart docker
```

重启后使用 `docker info` 查看是否修改成功

```
...
Docker Root Dir: /home/docker
...
Registry Mirrors:
  https://kfwkfulq.mirror.aliyuncs.com/
  https://2lqq34jg.mirror.aliyuncs.com/
  https://pee6w651.mirror.aliyuncs.com/
  https://registry.docker-cn.com/
  http://hub-mirror.c.163.com/
...
```

可以删除之前的docker存储目录了。

安装Shipyards中文版

官网脚本链接已失效，下载github缓存的脚本文件

```
wget https://raw.githubusercontent.com/shipyards/shipyards-project.com/master/site/themes/shipyards/static/deploy
```

修改脚本

```
vim deploy
```

将deploy脚本中 `IMAGE=${IMAGE:-shipyards/shipyards:latest}` 修改为 `IMAGE=${IMAGE:-dockerclub/shipyards:latest}` 即可切换为Docker Hub中的中文版镜像源

修改 `SHIPYARDS_PORT=${PORT:-7777}` 中的端口号可以自定义Shipyards的Web访问端口

修改完成后保存执行

```
sudo bash deploy
```

有选择的话默认即可

安装完成后等待docker容器启动完毕即可通过 `localhost:port` 访问到Web控制台，默认用户名 `admin`，密码 `shipyards`。

创建容器

由于运行容器需要 `nvidia-docker run` 而不是 `docker run` 才能正确初始化使用GPU的容器，接下来还是采用命令行执行。当容器使用 `nvidia-docker run` 正确启动后，就可再使用Docker或Shipyards进行管理了。

在 `daemon.json` 中修改了 `default-runtime` 后，运行容器可以直接使用Shipyards或者Docker进行初始化 `run` 了。但是注意经测试用于控制GPU可见性的 `NV_GPU` 环境变量依然需要使用命令行 `nvidia-docker run` 才能正确设置给容器，如果使用 `docker` 指令，则应该使用 `docker run` 的 `-e NVIDIA_VISIBLE_DEVICES=1` 来控制GPU的可见性。

接下来是手动创建Docker模板容器的过程，为了简化过程笔者已经将容器打包上传Dockerhub，不想自己动手创建容器的读者可以直接跳转文末下载笔者制作好的镜像【附：实验室使用的最新版镜像】

下载有CUDA和CUDNN环境的镜像

```
docker pull nvidia/cuda:10.1-cudnn7-runtime-ubuntu16.04
```

这里如果直接 `pull nvidia/cuda` 的话拉取的是latest版本的镜像，系统为ubuntu18.04，因此我们手动指定版本为 `CUDA:10.1, CUDNN:7, Ubuntu:16.04` 的镜像。更多版本的镜像可以通过 <https://hub.docker.com/r/nvidia/cuda/tags> 查看。下面 是以Ubuntu16.04为实例进行配置。

创建容器

```
nvidia-docker run -dit -p2345:22 --name=cuda1 -h=LAB_VM nvidia/cuda:10.1-cudnn7-runtime-ubuntu16.04
```

参数	含义
-dit	容器保持后台运行
-p2345:22	容器22端口映射到宿主主机2345端口
--net=host	使用主机模式，区别于默认的--net=bridge通过网桥连接宿主主机网络，host模式直接使用宿主主机网络空间，此模式下无法使用-p参数映射端口
--name=cuda1	命名容器
-h=LAB_VM	命名机器
--privileged	以特权模式启动，容器内能访问操作宿主主机设备，如mininet等需要控制网卡等类似的应用需要
--restart=always	docker启动后（如宿主主机断电重启）自动启动容器
-v /home/docker-common-dir:/home/common-dir	将宿主主机/home/docker-common-dir映射到容器/home/common-dir，一般用于共享目录

进入容器

```
docker exec -it cuda1 /bin/bash
```

查看GPU是否能正常访问

```
nvidia-smi
```

配置容器环境

注：这里使用的主要是交互式教程，可以参考结尾给出的Dockerfile使用非交互式指令配置容器

容器内的Ubuntu是一个非常精简的系统，缺乏包括 `ping` `vi` 等一系列常用指令，需要按照以下方式配置

更新apt源

```
apt-get update
```

如果一直卡在connect这里进行不下去，说明容器内无法连接外网，此时需要配置

退出容器命令行

```
exit
```

设置Linux内核允许IP转发

```
sudo sysctl net.ipv4.conf.all.forwarding=1
```

将iptables FORWARD 的值更从DROP更改为ACCEPT:

```
sudo iptables -P FORWARD ACCEPT
```

资料显示这里的两个配置不会持久化，每次重启都需要重新配置，需要将它们添加到start-up script，但是笔者实测不需要重新配置

之后重新进入容器控制台执行 `apt-get update`，应该可以执行成功了，如果有卡在一半的地方是在更新NVIDIA的源，需要检查网络环境是否能够连通NVIDIA的网站

安装vim

```
apt-get install vim
```

更换apt源

```
cp /etc/apt/sources.list /etc/apt/sources.list.bak
rm /etc/apt/sources.list
vim /etc/apt/sources.list
```

粘贴入16.04的清华源内容<https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu/>

这里如果意外设置成18.04的源不会马上报错，但是接下来会出现各种问题，一定要看好

更新apt源

```
apt-get update
```

安装ssh

```
apt-get install openssh-server
```

允许root远程连接

```
vim /etc/ssh/sshd_config
```

把 `PermitRootLogin prohibit-password`

修改为 `PermitRootLogin yes`

为root设置密码

```
passwd root
```

如果是宿主机Ubuntu16.04，安装ssh服务应该会自行启动并且已经加入自启项，但是在容器内安装则不会自动启动，而且一般设置自启项的方法对**Docker**容器不起作用，需要按照如下方法设置启动脚本

启动ssh服务（此时可以通过 `ssh://root:密码@宿主机IP -p2345` 连接容器了）

```
service ssh start
```

创建启动脚本

```
cd /home
vim startup.sh
```

输入启动脚本内容

```
#!/bin/bash
service ssh start
/bin/bash
```

`/bin/bash` 的作用是保持Docker容器的后台运行，使用 `-dit` 参数的时候会附加执行这个命令，但是当设置了启动脚本后就不会附加执行了，需要手动执行。

附加执行权限

```
chmod 777 shartup.sh
```

这样基本的环境就配置好了，接下来还可以再安装些常用的库如 `ping` 等等，如果需要安装Anaconda的话会提示缺少一个库，通过 `apt-get install bzip2` 即可安装。还可以修改下ubuntu的ssh连接欢迎文字，下一步就开始打包了。

下面列举几个容器中缺少但常用的库

库	作用
vim	强大的Linux文本编辑库
openssh-server	ssh远程连接库
net-tools	包含ifconfig, netstat等指令
iputils-ping	包含ping指令
wget	下载文件指令
curl	网络请求指令
git	版本控制
bzip2	conda的依赖
iptables	包过滤防火墙，可以控制转发策略
command-not-found	在你输入一个未安装的指令时提示安装

解决中文乱码

docker容器内出现无法输入中文，查看中文字符出现乱码情况，解决方法：在启动脚本中加入更换编码指令

```
vim /etc/profile
```

追加 `export LANG=C.UTF-8` 保存退出

打包为镜像

为了便于以后新建虚拟机，将容器作为镜像打包

```
nvidia-docker commit -m "一些说明" -a "作者" cuda1 cuda-base:1.0
```

其中 `cuda1` 为容器的名字， `cuda-base:1.0` 为镜像的名字和tag。

使用Dockerfile构建镜像

上述方法如果出现了误操作则需要重新构建，为此我将相关指令编写为Dockerfile

```

FROM nvidia/cuda:10.1-cudnn7-runtime-ubuntu16.04
MAINTAINER author<author@xxx.com>
RUN echo "export LANG=C.UTF-8" >>/etc/profile \
&& cp /etc/apt/sources.list /etc/apt/sources.list.bak \
&& echo "deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial main restricted universe multiverse\n\
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-updates main restricted universe multiverse\n\
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-backports main restricted universe multiverse\n\
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ xenial-security main restricted universe multiverse" >/etc/apt/
sources.list \
&& apt-get update \
&& apt-get install vim openssh-server iputils-ping wget curl -y \
&& sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/g' /etc/ssh/sshd_config \
&& echo "root:rootpassword" | chpasswd \
&& echo "#!/bin/bash\n\
service ssh start\n\
/bin/bash" >/home/startup.sh \
&& chmod 777 /home/startup.sh \
&& echo "\
printf '\\n'\n\
printf '\\\033[0;34m'\n\
printf ' 欢迎使用GPU服务器\\n'\n\
printf '\\\033[0m'\n\
printf '\\n'\n\
" >>/etc/update-motd.d/10-help-text

ENTRYPOINT /home/startup.sh

```

镜像打包指令，注意要在有 `Dockerfile` 文件的目录下执行

```
nvidia-docker image build -t cuda-base:1.0 .
```

新建容器

新建公共目录

```
cd /home
sudo mkdir docker-common-dir
```

新建两个容器

```

NV_GPU=0 nvidia-docker run -dit --restart=always -v /home/docker-common-dir:/home/common-dir -p6001:22 --name=vm
1 -h=LAB_VM cuda-base:1.0 /home/startup.sh
NV_GPU=1 nvidia-docker run -dit --restart=always -v /home/docker-common-dir:/home/common-dir -p6002:22 --name=vm
2 -h=LAB_VM cuda-base:1.0 /home/startup.sh

```

其中 `NV_GPU` 环境变量是为了控制容器可访问的GPU，如果不加这个参数则容器可以访问全部的GPU。 `--restart=always` 是为了设置容器随Docker自动启动。将宿主机的 `/home/docker-common-dir` 映射到了容器的 `/home/common-dir` 目录，便于和容器的文件传输，容器的文件存储以及容器之间的文件传输。最后跟的是启动脚本，如果不输入则每次容器启动都不会自动启动ssh服务。

如果一切正常的话，现在已经可以通过ssh客户端连接两个容器了。

附：实验室使用的最新版镜像

此版本的镜像迎合实验室的需求，配置了Miniconda和一个py37的默认环境并换清华源。设置了 `ENTRYPOINT`，创建容器时无需指定启动脚本。启动容器时可通过 `SSH_PORT` 控制容器的SSH端口（指容器内部端口，默认为22）。

Docker Hub主页: <https://hub.docker.com/r/hangvane/cuda-conda-desktop>

Github主页: <https://github.com/hangvane/cuda-conda-desktop>

拉取镜像:

```
docker pull hangvane/cuda-conda-desktop:ubuntu16.04
```

创建容器:

```
NV_GPU=0 nvidia-docker run -dit -p6009:22 --name=vm -h=LAB_VM -e SSH_PORT=22 hangvane/cuda-conda-desktop:ubuntu16.04
```

尾言

容器启动完成后, 就可以使用Shipyards进行容器和镜像的可视化管理了, 和不使用nvidia-docker一样。也可以给老师和同学们分配Shipyards的账户让他们自行管理, 再次注意如果使用Shipyards Web端建立容器相当于在宿主机执行 `docker run` 命令, 注意前述的 `NV_GPU` 环境变量的配置问题。

建议不要删除模板容器 `cuda1`, 以后更新 `cuda-base` 镜像还需要这个容器, 如果再通过 `cuda-base` 镜像创建容器更新的话就会有一连串镜像依赖。

存在的问题

目前每次系统重启后, 无法找到显卡, 表现为 `nvidia-smi` 命令报错, 之前使用 `nvidia-docker` 命令启动的容器无法启动。暂时的解决方法是卸载重装驱动。(数月后更新了显卡驱动解决)

参考

- [ubuntu安装Docker CE - yjk13703623757的博客 - CSDN博客](#)
- [【第08课: 用 Docker 建立一个公用 GPU 服务器】 - CSDN](#)
- [深度学习环境配置: 2080ti显卡驱动+ubuntu16.04+cuda10+cudnn7.3+tensorflow_gpu1.12 - yt1242228309的博客 - CSDN博客](#)
- [【Docker】如何修改Docker的默认镜像存储位置\(二\) - 一起长大的约定 - CSDN博客](#)
- [docker虚拟化之将容器做成镜像 - 莫林1 - 博客园](#)
- [Docker 修改默认存储路径的一个方法 - 周大侠的博客 - CSDN博客](#)
- [Ubuntu修改ssh登录欢迎信息 - BLSpan的博客 - CSDN博客](#)
- [关于Docker目录挂载的总结 - iVictor - 博客园](#)
- [别为Docker本地实现不支持GPU发愁, 解决方案在此! - 开源云中文社区 - CSDN博客](#)
- [Ubuntu Docker CE的安装和卸载 - saspyair的博客 - CSDN博客](#)
- [安装 NVIDIA Docker 2 來讓容器使用 GPU - KaiRen's Blog](#)
- [About Docker CE | Docker Documentation](#)