ALIBABA CLOUD

阿里云

GPU云服务器

最佳实践

文档版本：20220602

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.搭建GRID驱动的License Server

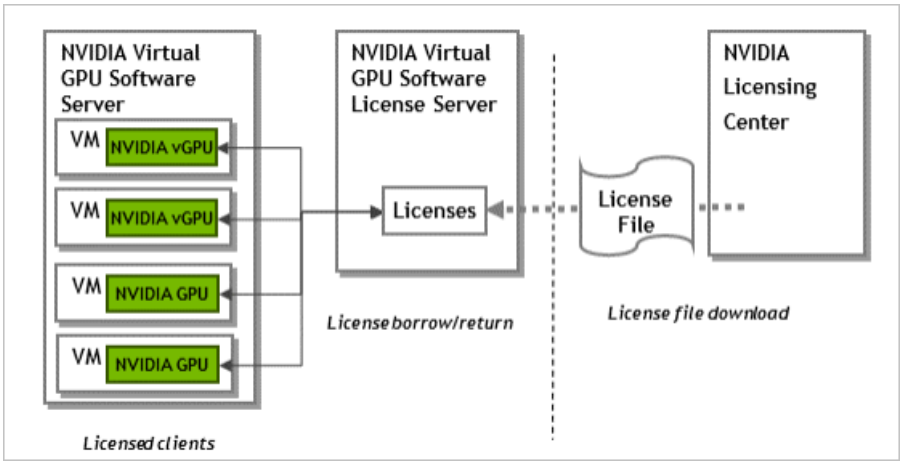
1.1. 搭建Linux环境License Server

使用GRID驱动必须申请相应的License，同时需要将License部署在License Server上，本文为您介绍搭建Linux操作系统的License Server的具体操作。

背景信息

本文以Ubuntu 18.04操作系统为例，如果您使用了其他Linux操作系统，如CentOS，需要根据实际情况替换部分命令。更多信息，请参见NVIDIA官方文档。

NVIDIA GRID License的工作原理图如下所示：



您需要从NVIDIA License中心获取License文件，并将License文件部署在已搭建的License Server上。然后使用您的GPU或vGPU实例，通过网络访问License Server激活GRID License。

步骤一：创建ECS实例

创建一台Linux操作系统的ECS实例作为License Server。具体创建操作，请参见使用向导创建实例。

您需要关注如下配置项，其他配置项根据自身业务场景设置即可。

配置项	说明
实例规格	请选择配置高于2 vCPU 4 GB的ECS实例规格。如果您的业务需要大量的License，建议您至少选择配置高于4 vCPU 16 GB的ECS实例规格，从而获得更高的性能及稳定性。
镜像	选择Linux操作系统镜像。本文以Ubuntu 18.04为例。
存储	请选择40 GiB以上系统盘。
带宽	建议您选择按使用流量计费，并将带宽峰值设置为100 Mbps。

步骤二：安装Java运行环境

1. 远程连接您用于搭建License Server的ECS实例。

   具体连接操作，请参见通过密码或密钥认证登录Linux实例。

2. 运行以下命令，验证当前实例是否已安装Java运行环境。

```
java -version
```

如果回显类似如下Java版本信息，表示已安装Java运行环境。否则，请继续执行后续操作完成安装。

```
java version "1.7.0_51"
OpenJDK Runtime Environment (rhel-2.4.5.5.el7-x86_64 u51-b31)
OpenJDK 64-Bit Server VM (build 24.51-b03, mixed mode)
```

3. 安装OpenJDK。

```
sudo apt install default-jdk
```

> **说明** 如果提示 `Unable to locate package default-jdk` 信息，请先执行 `apt update` 命令后，再执行以上命令进行安装。

回显类似如下信息，表示OpenJDK已安装成功。

```
...
Running hooks in /etc/ca-certificates/update.d...
done.
done.
```

步骤三：安装并运行Apache Tomcat服务器

1. 使用Linux发行版的软件包管理器安装所需的Apache Tomcat软件包。

```
sudo apt install tomcat8
```

2. 安装完成后，运行以下命令，将Tomcat服务设置为开机自启动。

```
sudo systemctl enable tomcat8.service
```

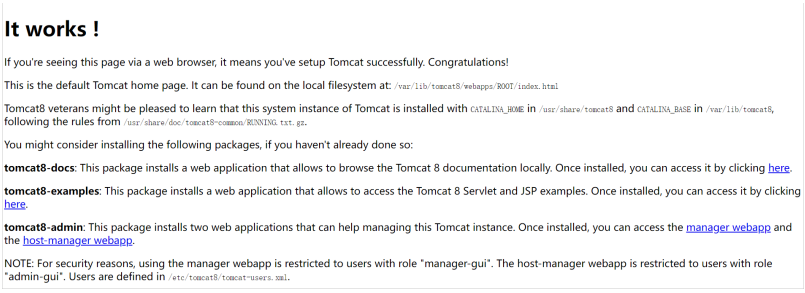
3. 启动Tomcat服务。

```
sudo systemctl start tomcat8.service
```

4. 通过本地Web浏览器访问*http://localhost:8080*，验证Tomcat服务是否可用。

> **说明** 请将访问地址中的*localhost*替换为您的ECS实例的公网地址。

如果显示如下内容，表示Tomcat服务已安装完成。



步骤四：安装License Server软件

您需要准备提前获取到的setup.bin安装文件或该文件所在的.zip压缩包。

1. 下载setup.bin安装程序或解压该程序所在的.zip压缩包。
   - 如果您通过NVIDIA官网获取了License Server的安装包，请运行以下命令，解压安装包。

     ```
     unzip 安装程序.zip
     ```

     > **说明** 请替换*安装程序.zip*为您获取到的安装包名称。

   - 如果您通过阿里云获取了License Server安装程序的下载地址，请运行以下命令，下载安装程序。

     ```
     wget https://grid-9-4-zyy.oss-cn-hangzhou.aliyuncs.com/setup.bin
     ```

2. 运行以下命令，为安装程序添加执行权限。

   ```
   chmod +x setup.bin
   ```

3. 安装License Server软件。
   i. 以root用户运行安装程序。

      ```
      sudo ./setup.bin -i console
      ```

   ii. 在**Introduction**部分，单击回车继续。

   ```
   ===============================================================================
   Introduction
   ------------

   InstallAnywhere will guide you through the installation of License Server.

   It is strongly recommended that you quit all programs before continuing with
   this installation.

   Respond to each prompt to proceed to the next step in the installation.  If
   you want to change something on a previous step, type 'back'.

   You may cancel this installation at any time by typing 'quit'.

   PRESS <ENTER> TO CONTINUE:
   ```

   iii. 在**License Agreement**部分，请通过每一次单击回车进行翻页并接受许可协议。

   当您达成许可协议时，系统会提示您接受许可协议条款，请输入*Y*，并单击回车。

   ```
   DO YOU ACCEPT THE TERMS OF THIS LICENSE AGREEMENT? (Y/N): Y
   ```

   iv. 在**Choose Install Folder**部分，请单击回车，保持默认的License Server软件安装路径。

   v. 在**Choose Local Tomcat Server Path**部分，输入Tomcat的本地路径，默认为*/var/lib/tomcat版本号*，例如：*/var/lib/tomcat8*。

   vi. 在**Choose Firewall Options**部分，确认需要在防火墙中打开的端口，单击回车，保持默认选项即可。

   ```
   Choose Firewall Options
   -----------------------

   The license server listens on port 7070. This port must be opened in the
   firewall for other machines to obtain licenses from this server.

   The license server's management interface listens on port 8080. Leave this
   port closed to prevent unauthorized access to the management interface.

     ->1- License server (port 7070)
       2- Management interface (port 8080)

   ENTER A COMMA-SEPARATED LIST OF NUMBERS REPRESENTING THE DESIRED CHOICES, OR
      PRESS <ENTER> TO ACCEPT THE DEFAULT:
   ```

vii. 在Pre-Installation Summary部分，确认信息并单击回车启动安装。

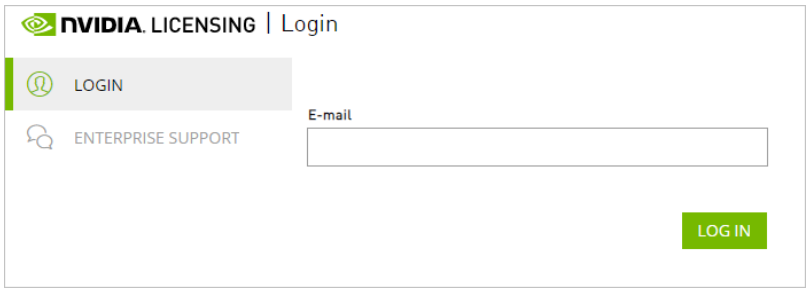
```
Pre-Installation Summary
------------------------

Please Review the Following Before Continuing:

Product Name:
    License Server

Install Folder:
    /opt/flexnetls/nvidia

Link Folder:
    /root/NVIDIA Corporation/License Server

Disk Space Information (for Installation Target):
    Required:     105,216,774 Bytes
    Available: 35,501,248,512 Bytes

PRESS <ENTER> TO CONTINUE:
```

viii. 在Install Complete部分，单击回车，结束安装。

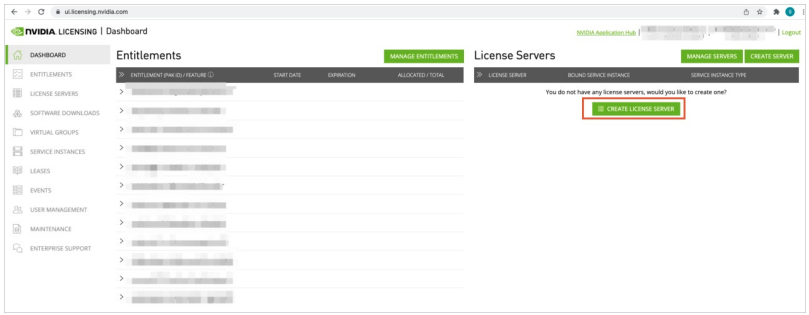
```
Install Complete
---------------

License Server has been successfully installed to:

   /opt/flexnetls/nvidia

PRESS <ENTER> TO EXIT THE INSTALLER:
```

步骤五：在NVIDIA License上创建License Server

1. 前往NVIDIA License，并使用您申请License的邮箱登录。

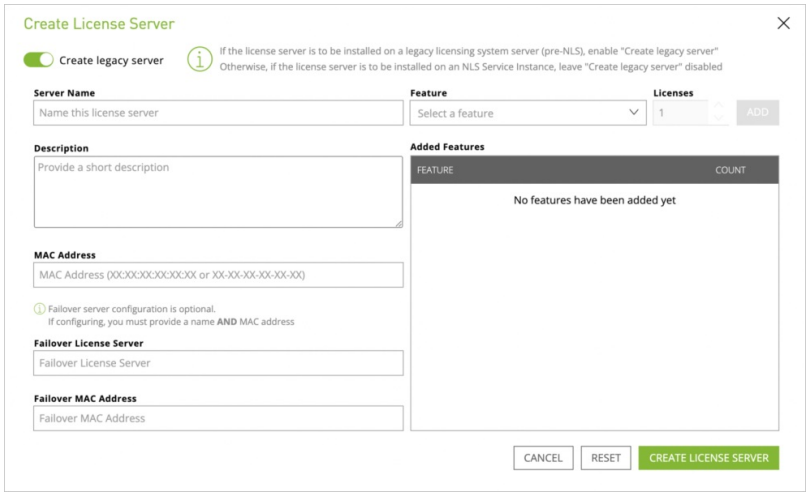


2. 在Dashboard页面，单击License Servers区域下CREATE LICENSE SERVER按钮。



Entitlements区域下，是您目前拥有的全部License。

3. 在打开的Create License Server页面，填写相关信息，然后单击CREATE LICENSE SERVER。



必填参数说明如下：

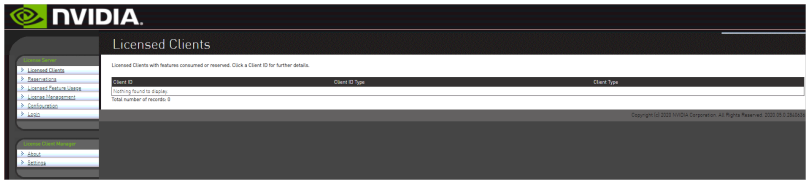
参数	说明
Server Name	自定义您需要的License Server名称。
MAC Address	填写您用于搭建License Server的ECS实例的MAC地址。 您可以登录实例，使用 `ipconfig -a` 命令进行查询。
Feature	根据需要选择，并输入待添加的License数量，单击**ADD**。

4. 创建完成后，在**License Servers**页面单击 按钮，下载License文件。

步骤六：导入License文件

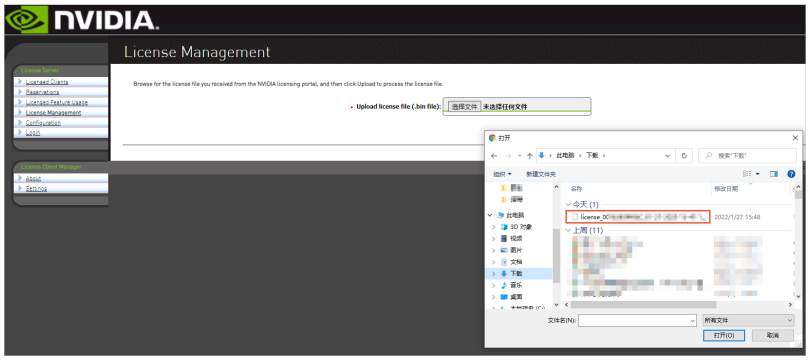
1. 通过本地Web浏览器访问*http://localhost:8080/licserver*，进入License Server管理界面。

> **说明** 请将访问地址中的*localhost*替换为您的ECS实例的公网地址。



2. 在左侧导航栏的**License Server**区域，单击**License Management**。
3. 在**License Management**页面，单击**Upload license file (.bin file)**后的**选择文件**，选择本地的

License文件，然后单击**打开(O)**。



4. 单击**Upload**。

   页面提示如下信息，表示License文件已导入成功。

   您可以在左侧导航栏的**License Server**区域，单击**License Feature Usage**，查看License数量以及使用情况。

步骤七：测试网络连接和访问

以下操作以创建一台Windows操作系统的GPU虚拟化实例vgn6i为例，您也可以直接应用您已创建的GPU实例。

1. 创建一台GPU实例。

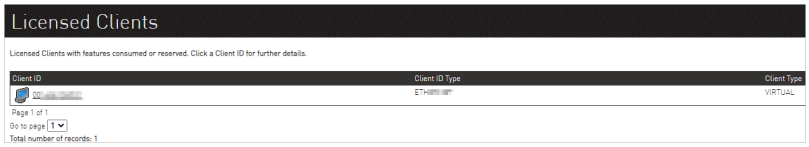
   具体操作，请参见创建未配备驱动的GPU实例。

   本示例选择的实例规格为GPU虚拟化实例vgn6i，操作系统为Windows Server 2019 数据中心版 64位中文版。

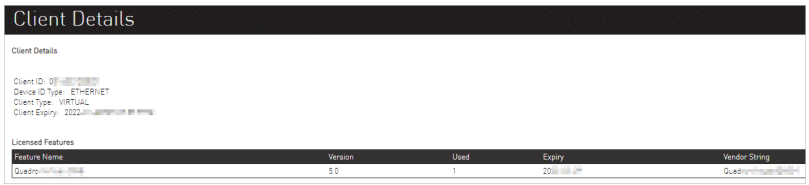
2. 安装GRID驱动，然后添加License服务器并激活License。具体操作，请参见在GPU实例上安装GRID驱动（Windows）。

   激活License时，请填写您搭建了License Server的ECS实例公网IP，端口号为7070。

3. 前往License Server管理界面（*http://localhost:8080/licserver*），在左侧导航栏的**License Server**区域，单击**Licensed Clients**，即可查看到GPU实例信息，表示该NVIDIA vGPU软件客户端虚拟机目前正在使用License的功能。



单击客户端ID，可以查看该客户端的详细信息。

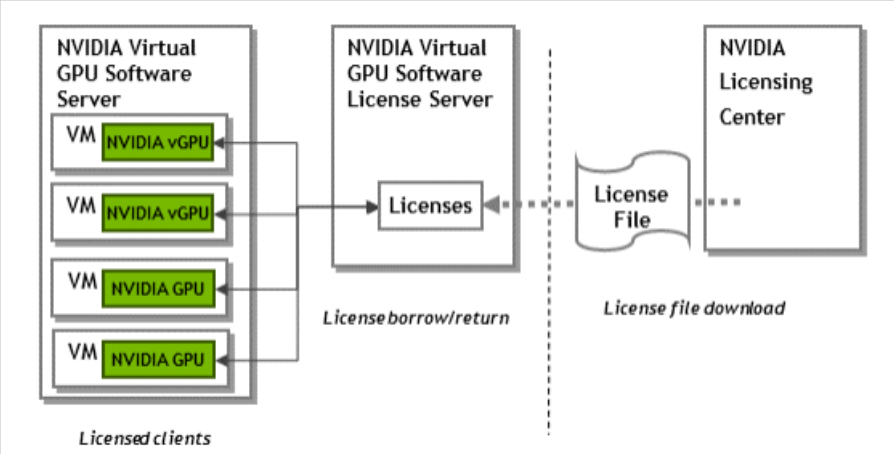


1.2. 搭建Windows环境的License Server

使用GRID驱动必须申请相应的License，同时需要将License部署在License Server上，本文为您介绍搭建Windows操作系统的License Server的具体操作。

背景信息

NVIDIA GRID License的工作原理图如下所示：



您需要从NVIDIA License中心获取License文件，并将License文件部署在已搭建的License Server上。然后使用您的GPU或vGPU实例，通过网络访问License Server激活GRID License。

步骤一：创建ECS实例

创建一台Windows操作系统的ECS实例作为License Server。具体创建操作，请参见使用向导创建实例。

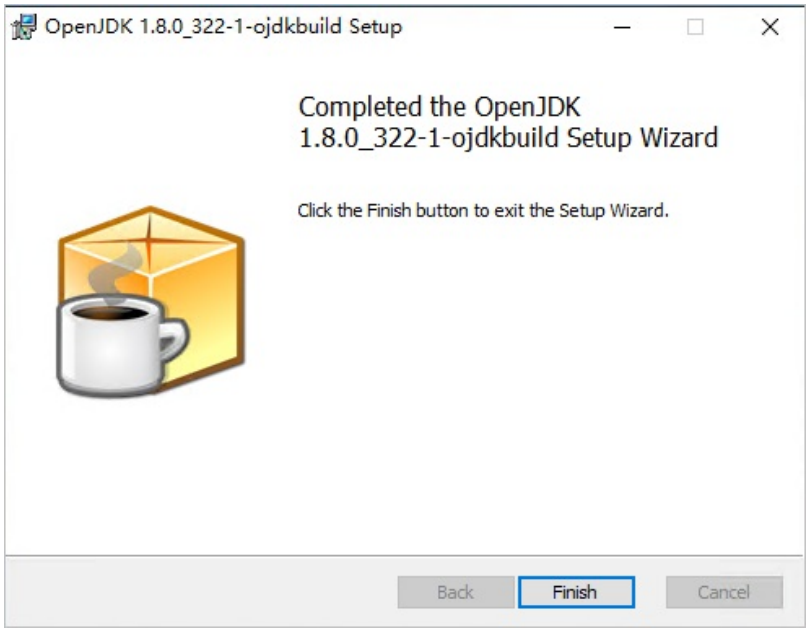
您需要关注如下配置项，其他配置项根据自身业务场景设置即可。

配置项	说明
实例规格	请选择配置高于2 vCPU 4 GB的ECS实例规格。如果您的业务需要大量的License，建议您至少选择配置高于4 vCPU 16 GB的ECS实例规格，从而获得更高的性能及稳定性。
镜像	选择Windows操作系统镜像。本文以Windows Server 2019 数据中心版 64位中文版为例。
存储	请选择40 GiB以上系统盘。
带宽	建议您选择按使用流量计费，并将带宽峰值设置为100 Mbps。

步骤二：安装Java运行环境

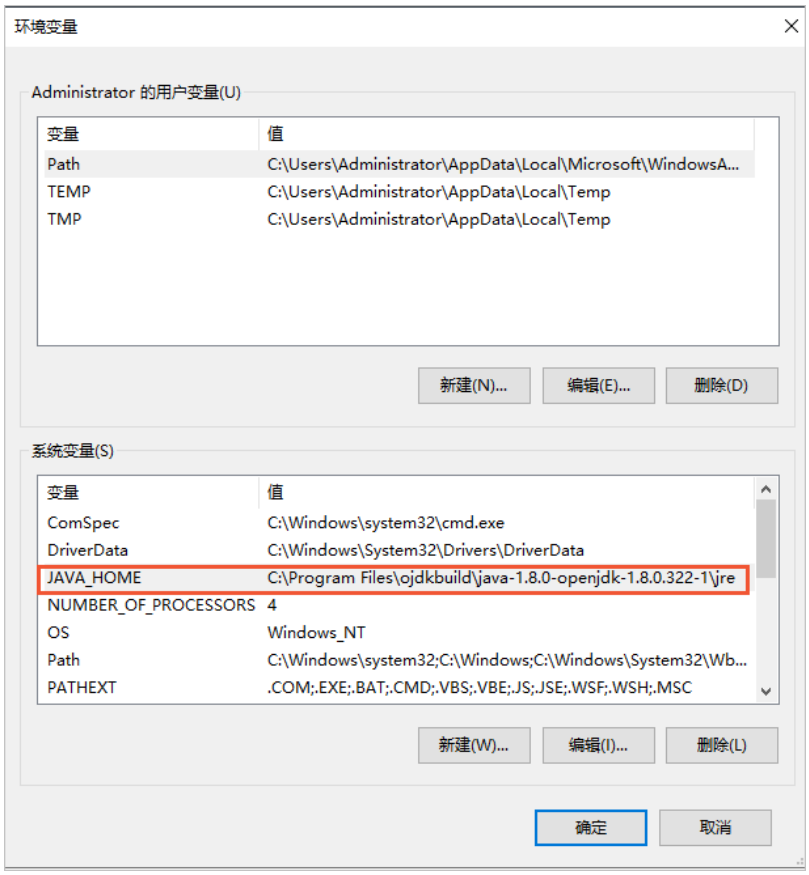
1. 远程连接您用于搭建License Server的ECS实例。

   具体连接操作，请参见通过密码或密钥认证登录Windows实例。
2. 请前往ojdkbuild下载OpenJDK JRE安装包。
3. 安装JRE。



4. 新建系统变量JAVA_HOME，并将取值设置为JRE的*jre*文件夹所在的绝对路径。

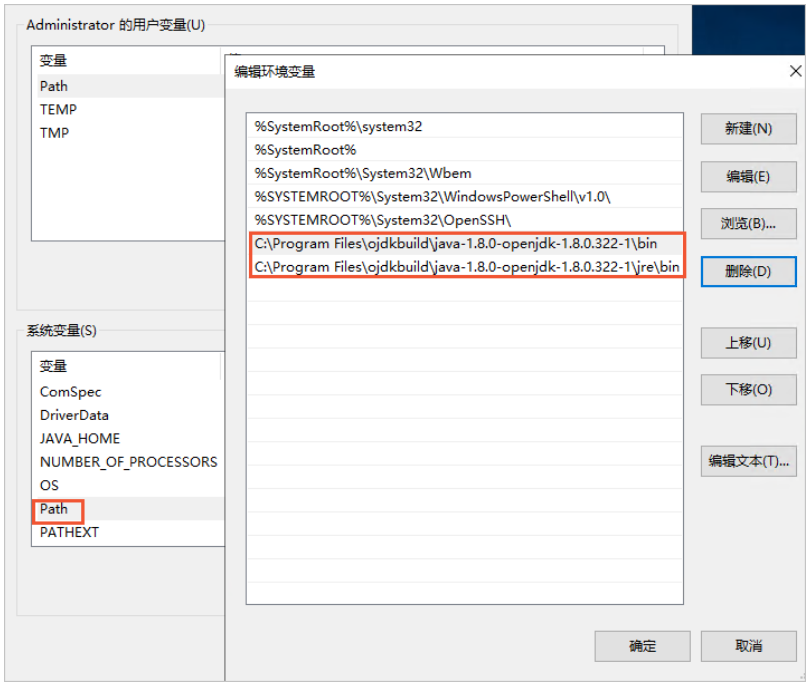
   例如，将路径设置为*C:\Program Files\ojdkbuild\java-1.8.0-openjdk-1.8.0.201-1\jre*。



> **说明** 请确保路径不包含任何尾随字符，例如反斜线（\）或空格。

5. 在系统变量Path中，确认是否已存在java.exe程序所在的绝对路径。

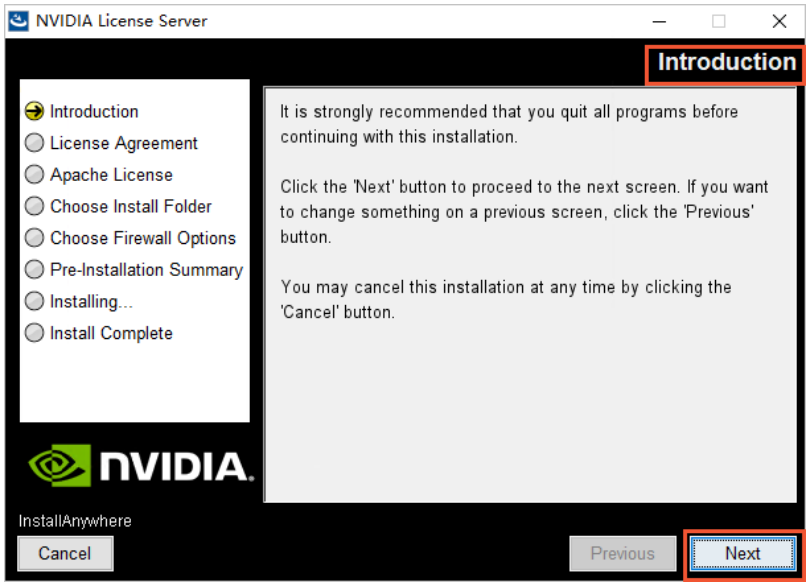
   通常情况下，安装JRE时，会自动添加此路径。



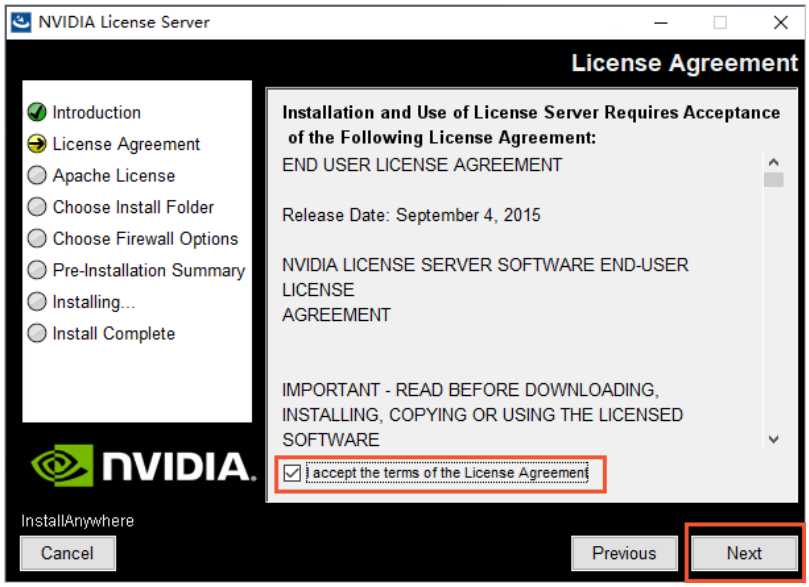
步骤三：安装License Server软件

您需要准备提前获取到的License Server安装程序。

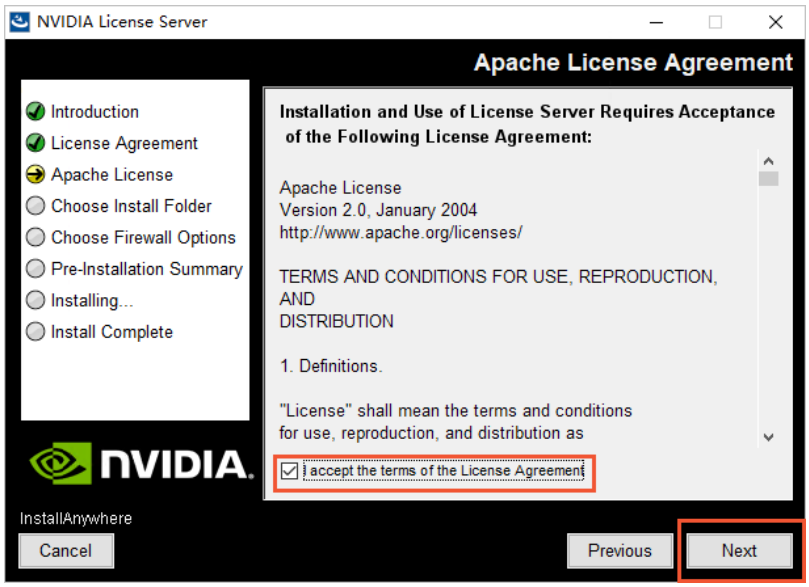
1. 解压License Server的.zip文件，并运行setup.exe安装程序。
2. 在Introduction页面，单击Next。



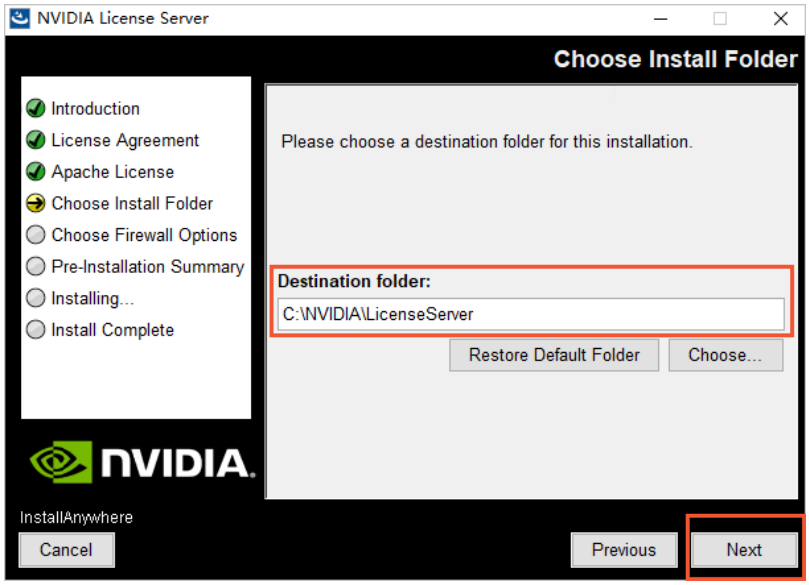
3. 在License Agreement页面，选中I accept the terms of the License Agreement，然后单击Next。



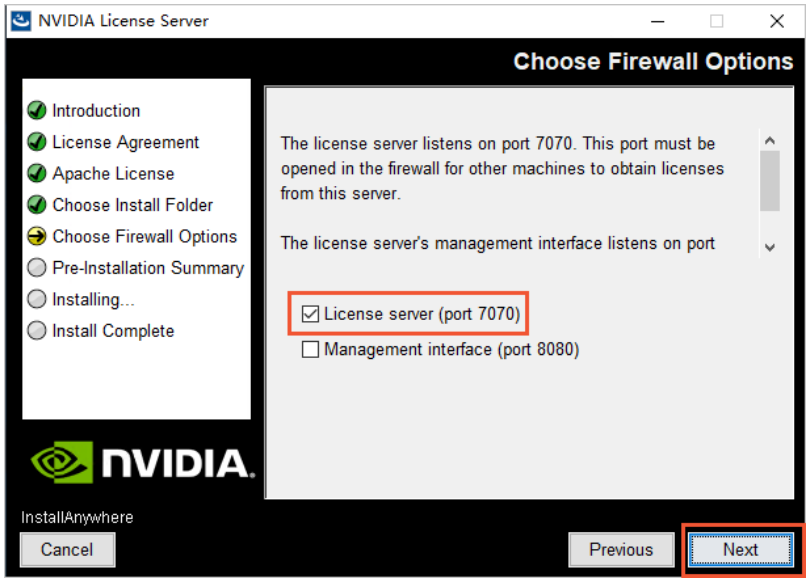
4. 在Apache License Agreement页面，选中I accept the terms of the License Agreement，然后单击Next。



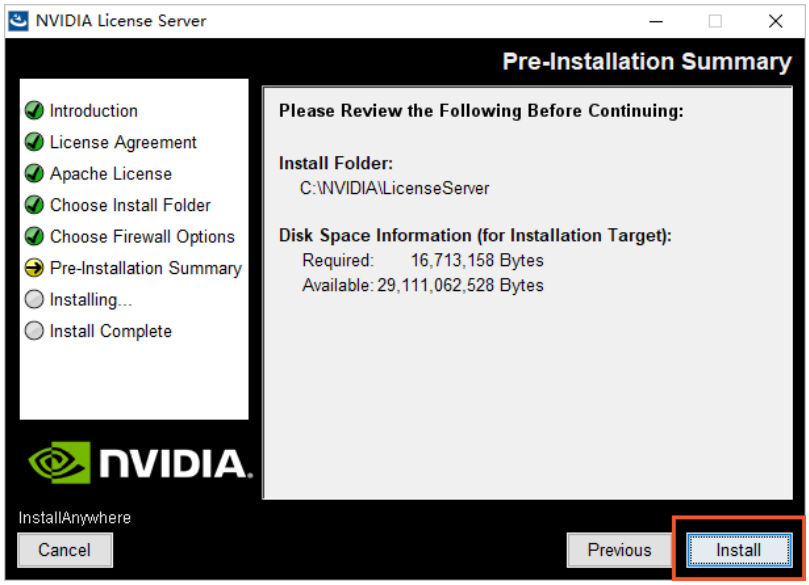
5. 在Choose Install Folder页面，设置目标地址，然后单击Next。



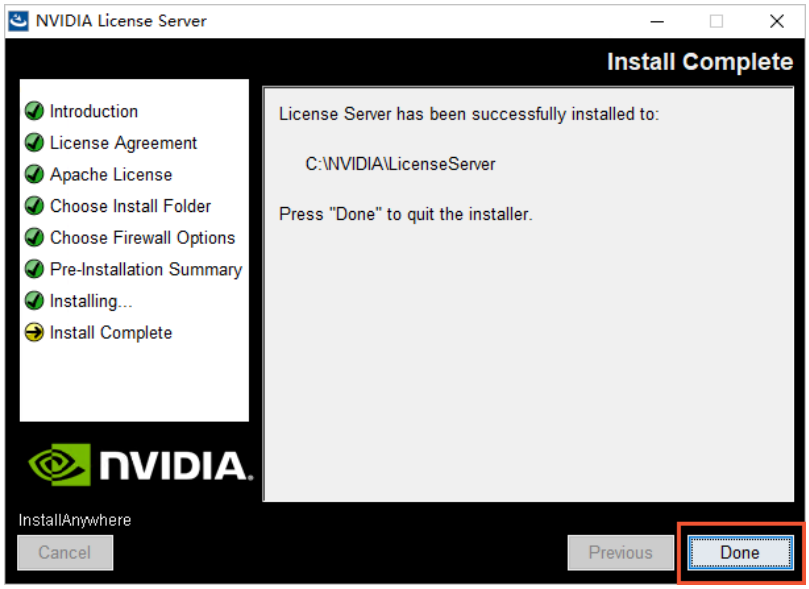
6. 在Choose Firewall Options页面，保持默认选项License server（port 7070），然后单击Next。



7. 在Pre-Installation Summary页面，确认配置信息，单击Install。

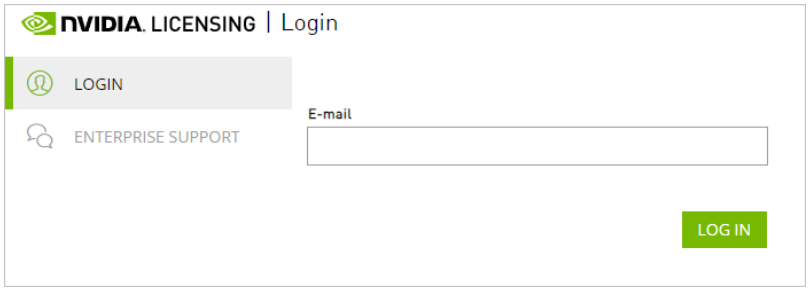


8. 安装完成后，单击Done。

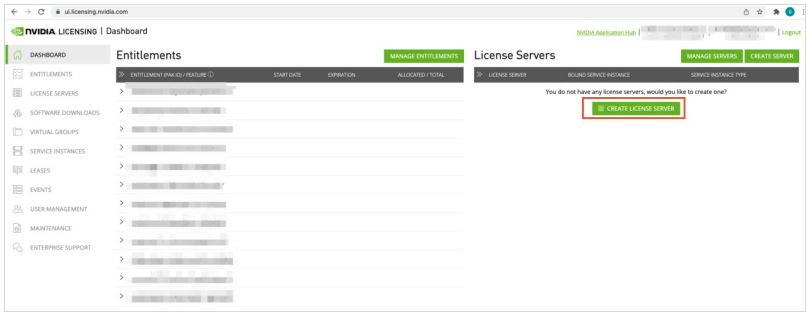


步骤四：在NVIDIA License上创建License Server

1. 前往NVIDIA License，并使用您申请License的邮箱登录。

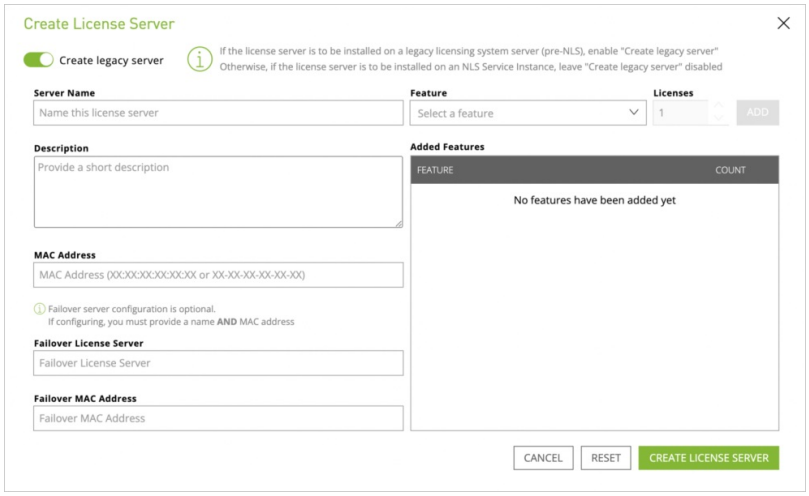


2. 在Dashboard页面，单击License Servers区域下CREATE LICENSE SERVER按钮。



Entitlements区域下，是您目前拥有的全部License。

3. 在打开的Create License Server页面，填写相关信息，然后单击CREATE LICENSE SERVER。



必填参数说明如下：

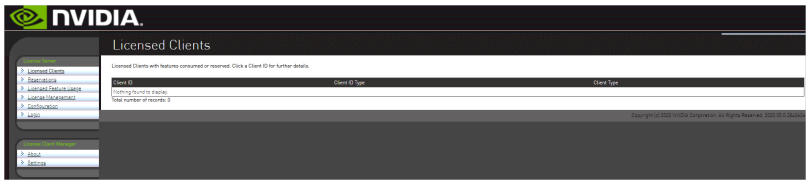
参数	说明
Server Name	自定义您需要的License Server名称。
MAC Address	填写您用于搭建License Server的ECS实例的MAC地址。 您可以登录实例，使用 `ipconfig -a` 命令进行查询。
Feature	根据需要选择，并输入待添加的License数量，单击ADD。

4. 创建完成后，在License Servers页面单击 按钮，下载License文件。

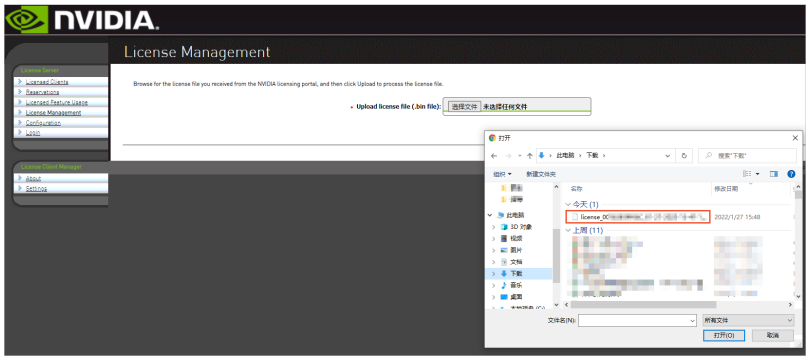
步骤五：导入License文件

1. 通过本地Web浏览器访问*http://localhost:8080/licserver*，进入License Server管理界面。

   > **说明** 请将访问地址中的*localhost*替换为您的ECS实例的公网地址。



2. 在左侧导航栏的**License Server**区域，单击**License Management**。
3. 在**License Management**页面，单击**Upload license file (.bin file)**后的**选择文件**，选择本地的License文件，然后单击**打开(O)**。



4. 单击**Upload**。

   页面提示如下信息，表示License文件已导入成功。

   您可以在左侧导航栏的**License Server**区域，单击**License Feature Usage**，查看License数量以及使用情况。

步骤六：测试网络连接和访问

以下操作以创建一台Windows操作系统的GPU虚拟化实例vgn6i为例，您也可以直接应用您已创建的GPU实例。

1. 创建一台GPU实例。

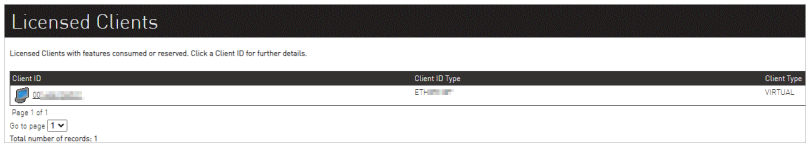
   具体操作，请参见创建未配备驱动的GPU实例。

   本示例选择的实例规格为GPU虚拟化实例vgn6i，操作系统为Windows Server 2019 数据中心版 64位中文版。

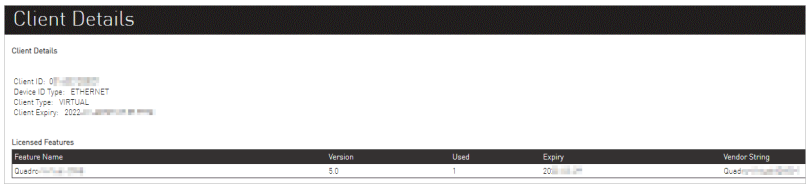
2. 安装GRID驱动，然后添加License服务器并激活License。具体操作，请参见在GPU实例上安装GRID驱动（Windows）。

   激活License时，请填写您搭建了License Server的ECS实例公网IP，端口号为7070。

3. 前往License Server管理界面（*http://localhost:8080/licserver*），在左侧导航栏的**License Server**区域，单击**Licensed Clients**，即可查看到GPU实例信息，表示该NVIDIA vGPU软件客户端虚拟机目前正在使用License的功能。



单击客户端ID，可以查看该客户端的详细信息。



2.在GPU实例上部署NGC环境

本文以搭建TensorFlow深度学习框架为例介绍如何在GPU实例上部署NGC环境。

前提条件

- 登录NGC网站，注册NGC账号。
- 登录NGC网站，获取NGC API key并保存到本地。登录NGC容器环境时需要验证您的NGC API Key。

背景信息

NGC（NVIDIA GPU CLOUD）是NVIDIA开发的一套深度学习生态系统，可以使开发者免费访问深度学习软件堆栈，建立适合深度学习的开发环境。

目前NGC在阿里云gn5实例作了全面部署，并且在镜像市场提供了针对NVIDIA Pascal GPU优化的NGC容器镜像。通过部署镜像市场的NGC容器镜像，开发者能简单快速地部署NGC容器环境，即时访问优化后的深度学习框架，极大程度缩减产品开发以及业务部署的时间，实现开发环境的预安装；同时支持调优后的算法框架，并且保持持续更新。

NGC网站提供了目前主流深度学习框架不同版本的镜像（例如Caffe、Caffe2、CNTK、MxNet、TensorFlow、Theano、Torch），您可以选择需要的镜像部署环境。

支持部署NGC环境的实例规格族包括：

- gn4、gn5、gn5i、gn6v、gn6i、gn6e
- ebmgn5i、ebmgn6i、ebmgn6v、ebmgn6e

下面以gn5实例为例，为您演示创建GPU实例和部署NGC环境的步骤。

操作步骤

1. 创建一台gn5实例。具体操作，请参见使用向导创建实例。

   在配置参数时，您需要注意以下几点：

   - **地域**：只能选择华北1（青岛）、华北2（北京）、华北3（张家口）、华北5（呼和浩特）、华东1（杭州）、华东2（上海）、华南1（深圳）。
   - **实例**：选择gn5实例规格。
   - **镜像**：单击**镜像市场**，在弹出的对话框中，找到**NVIDIA GPU Cloud VM Image**，然后单击**使用**。

   - **公网带宽**：选择**分配公网IP地址**。

     > **说明** 如果这里没有分配公网IP地址，则在实例创建成功后，需要绑定EIP地址。

   - **安全组**：选择一个安全组。安全组里必须开放TCP 22端口。如果您的实例需要支持HTTPS或DIGIT 6

服务，必须开放TCP 443（用于HTTPS）或TCP 5000（用于DIGITS 6）端口。

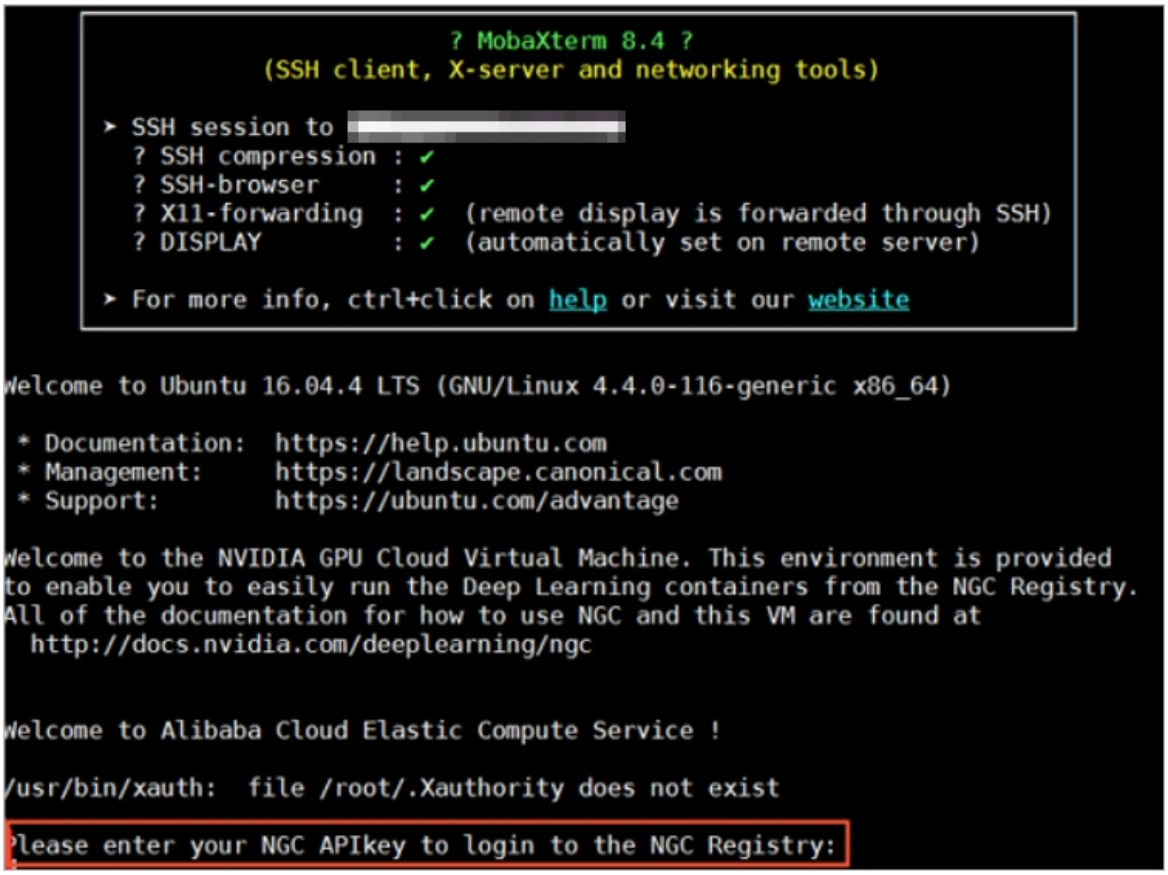
ECS实例创建成功后，请登录ECS管理控制台，记录实例的公网IP地址。

2. 连接ECS实例。

   根据创建实例时选择的登录凭证选择以下任一方式连接ECS实例：

   - 使用密码验证连接ECS实例
   - 使用SSH密钥对验证连接ECS实例

3. 按界面提示输入NGC官网获取的NGC APIKey后按回车键，即可登录NGC容器环境。



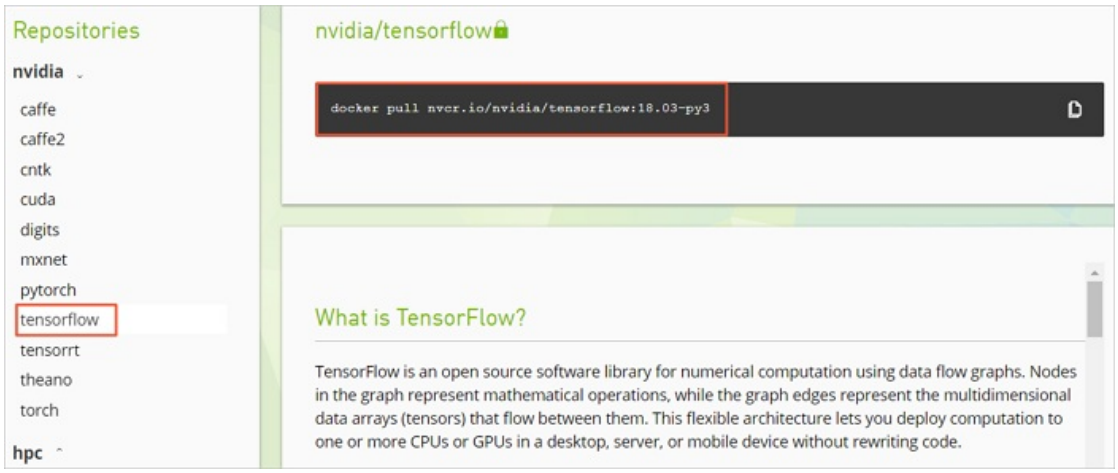
4. 运行 `nvidia-smi` 命令。

   您能查看当前GPU的信息，包括GPU型号、驱动版本等，如下图所示。

```
root@                          # nvidia-smi
Thu Mar 29 20:50:01 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.111                 Driver Version: 384.111                  |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:08.0 Off |                    0 |
| N/A   29C    P0    27W / 250W |      0MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

5. 搭建TensorFlow深度学习框架。

   i. 登录NGC网站，在TensorFlow镜像页面，获取 `docker pull` 命令。



   ii. 下载TensorFlow镜像。

```
docker pull nvcr.io/nvidia/tensorflow:18.03-py3
```

   iii. 查看下载的镜像。

```
docker image ls
```

   iv. 运行容器，完成TensorFlow开发环境的部署。

```
nvidia-docker run --rm -it nvcr.io/nvidia/tensorflow:18.03-py3
```

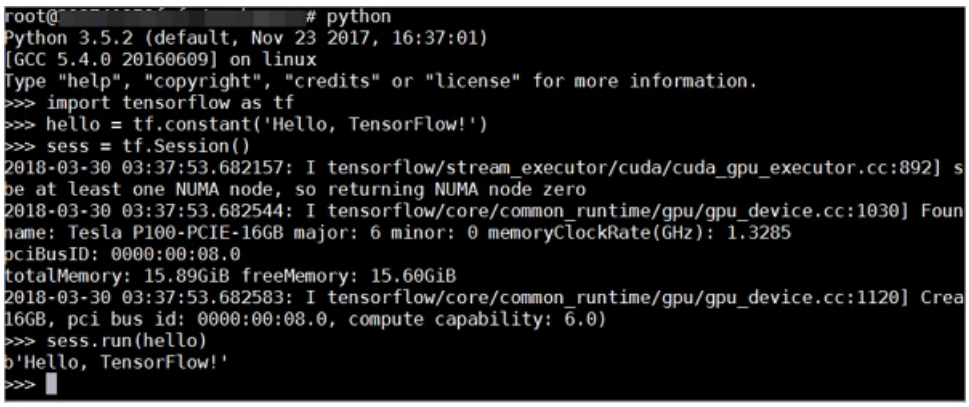
```
root@                          :~# nvidia-docker run --rm -it nvcr.io/nvidia/tensorflow:18.03-py3

================
== TensorFlow ==
================

NVIDIA Release 18.03 (build 349854)

Container image Copyright (c) 2018, NVIDIA CORPORATION.  All rights reserved.
Copyright 2017 The TensorFlow Authors.  All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying project or file.
```

6. 选择以下任一种方式测试TensorFlow。
   - 简单测试TensorFlow。

     ```
     python
     ```

     ```
     import tensorflow as tf
     hello = tf.constant('Hello, TensorFlow!')
     sess = tf.Session()
     sess.run(hello)
     ```

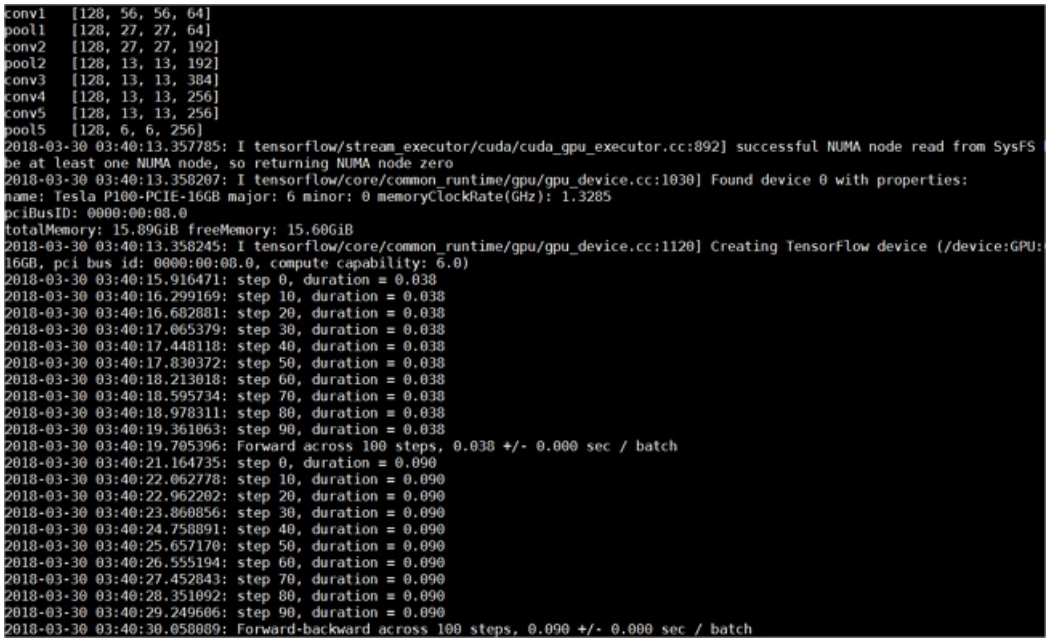
     如果TensorFlow正确加载了GPU设备，返回结果如下图所示。

   - 下载TensorFlow模型并测试TensorFlow。

     ```
     git clone https://github.com/tensorflow/models.git
     cd models/tutorials/image/alexnet
     python alexnet_benchmark.py --batch_size 128 --num_batches 100
     ```

     运行状态如下图所示。

7. 保存TensorFlow镜像的修改。否则，下次登录时配置会丢失。

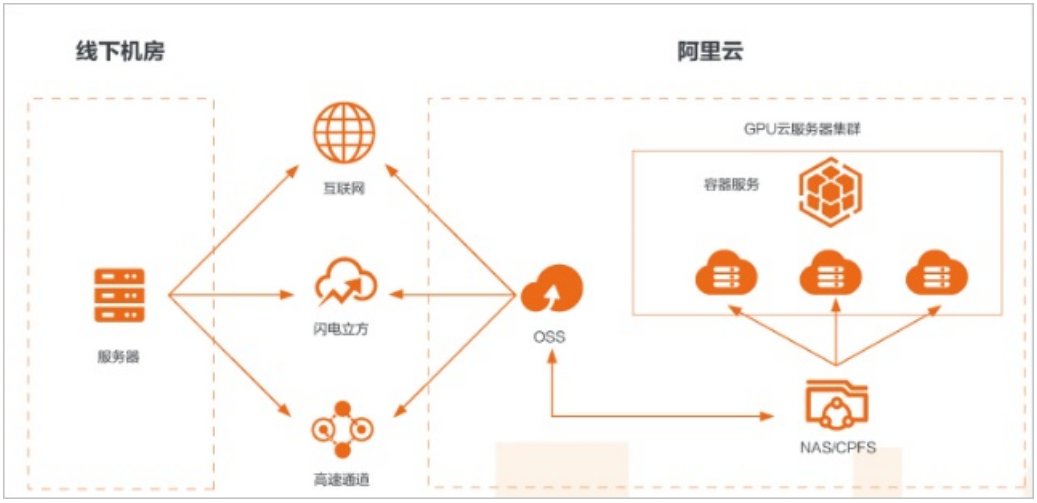
3.GPU AI模型训练最佳实践

本方案适用于AI图片训练场景，使用CPFS/NAS作为共享存储，利用容器服务Kubernetes版管理GPU云服务器集群进行图片AI训练。

实现的方式

- 搭建AI图片训练基础环境。
- 使用CPFS存储训练数据。
- 使用飞天AI加速训练服务加速训练。
- 使用Arena一键提交作业。

部署架构图



选用的产品

- GPU服务器

  GPU云服务器是基于GPU应用的计算服务，多适用于AI深度学习、视频处理、科学计算、图形可视化等应用场景。

  更多关于GPU服务器的介绍，请参见GPU服务器产品详情页。

- 文件存储NAS

  阿里云文件存储NAS是一个可共享访问、弹性扩展、高可靠、高性能的分布式文件系统。兼容POSIX文件接口，可支持数千台计算节点共享访问，可以挂载到弹性计算ECS、弹性裸金属服务器、容器服务ACK、弹性容器ECI、批量计算BCS、高性能计算EHPC、AI训练PAI等计算业务上提供高性能的共享存储，用户无需修改应用程序，即可无缝迁移业务系统上云。

  更多关于文件存储NAS的介绍，请参见文件存储NAS产品详情页。

- 文件存储CPFS

  文件存储CPFS（Cloud Parallel File Storage），是阿里云完全托管、可扩展的并行文件存储系统，针对高性能计算场景的性能要求进行了深度优化，提供对数据毫秒级的访问和高聚合IO、高IOPS的数据读写请求，可以用于AI深度训练、自动驾驶、基因计算、EDA仿真、石油勘探、气象分析、机器学习、大数据分析以及影视渲染等业务场景中。

  更多关于文件存储CPFS的介绍，请参见文件存储CPFS详情页。

- 容器服务 ACK

  容器服务Kubernetes版（ACK）提供高性能可伸缩的容器应用管理能力，支持企业级容器化应用的全生命周期管理。整合阿里云虚拟化、存储、网络和安全能力，打造云端最佳容器化应用运行环境。

  更多关于容器服务ACK的介绍，请参见容器服务 ACK产品详情页。

详细信息

点击查看最佳实践详情

更多最佳实践

点击查看更多阿里云最佳实践

4.在GPU实例上使用RAPIDS加速机器学习任务

本文介绍了如何在GPU实例上基于NGC环境使用RAPIDS加速库，加速数据科学和机器学习任务，提高计算资源的使用效率。

背景信息

RAPIDS，全称Real-time Acceleration Platform for Integrated Data Science，是NVIDIA针对数据科学和机器学习推出的GPU加速库。更多RAPIDS信息，请参见官方网站。

NGC，全称NVIDIA GPU CLOUD，是NVIDIA推出的一套深度学习生态系统，供开发者免费访问深度学习和机器学习软件堆栈，快速搭建相应的开发环境。NGC网站提供了RAPIDS的Docker镜像，预装了相关的开发环境。

JupyterLab是一套交互式的开发环境，帮助您高效地浏览、编辑和执行服务器上的代码文件。

Dask是一款轻量级大数据框架，可以提升并行计算效率。

本文提供了一套基于NVIDIA的RAPIDS Demo代码及数据集修改的示例代码，演示了在GPU实例上使用RAPIDS加速一个从ETL到ML Training端到端任务的过程。其中，ETL时使用RAPIDS的cuDF，ML Training时使用XGBoost。本文示例代码基于轻量级大数据框架Dask运行，为一套单机运行的代码。

> **说明** NVIDIA官方RAPIDS Demo代码请参见Mortgage Demo。

RAPIDS预装镜像已经发布到阿里云镜像市场，创建GPU实例时，您可以在镜像市场中搜索*NVIDIA RAPIDS*并使用RAPIDS预装镜像。

> **说明** 该RAPIDS预装镜像使用Ubuntu 16.04 64-bit操作系统。

操作步骤

如果您创建GPU实例时使用了RAPIDS预装镜像，只需运行RAPIDS Demo，从启动JupyterLab服务开始操作即可。详情请参见启动JupyterLab服务。

如果您创建GPU实例时没有使用RAPIDS预装镜像，按照以下步骤使用RAPIDS加速机器学习任务：

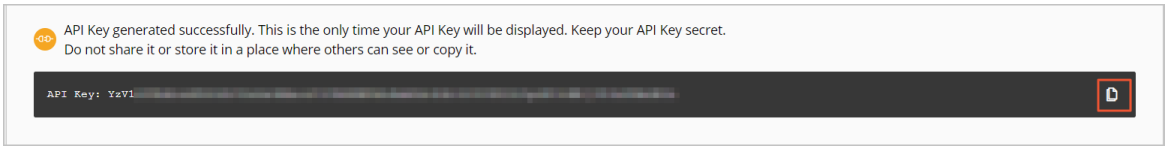
- 步骤一：获取NGC API Key
- 步骤二：获取RAPIDS镜像下载命令
- 步骤三：部署RAPIDS环境
- 步骤四：运行RAPIDS Demo

步骤一：获取NGC API Key

1. 在NGC注册页面注册NGC账号。
2. 登录NGC网站。
3. 单击页面右上角用户名，然后单击**Setup**。
4. 在**Setup**页面的**Generate API Key**区域，单击**Get API Key**。
5. 在**API Key**页面，单击**Generate API Key**。
6. 在**Generate a New API Key**对话框中，单击**Confirm**。

> **说明** 新的NGC API Key会覆盖旧的NGC API Key。如果您已持有NGC API Key，请确保不再需要旧的NGC API Key。

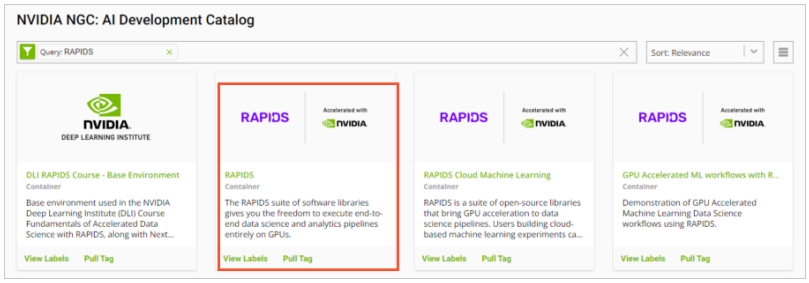
7. 复制API Key并保存到本地。



步骤二：获取RAPIDS镜像下载命令

完成以下操作，获取RAPIDS镜像下载命令：

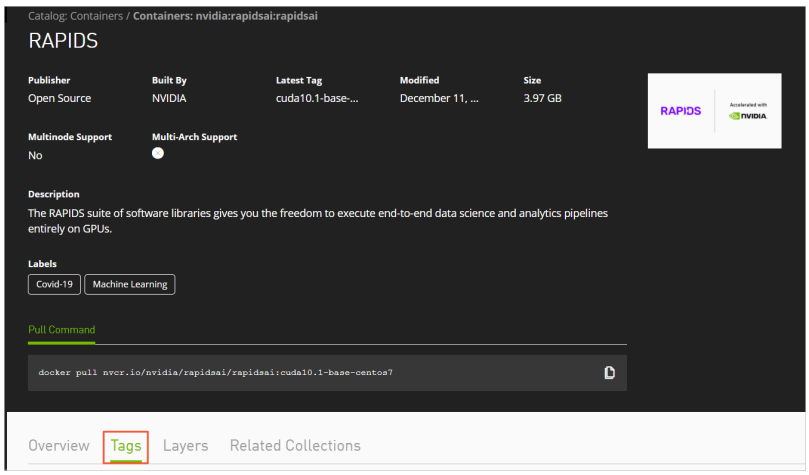
1. 登录NGC网站。
2. 在页面左侧导航栏，选择**CATALOG > Explore Catalog**。
3. 在**NVIDIA NGC: AI Development Catalog**页面的搜索栏中，输入*RAPIDS*。
4. 在搜索结果中，单击**RAPIDS**镜像。



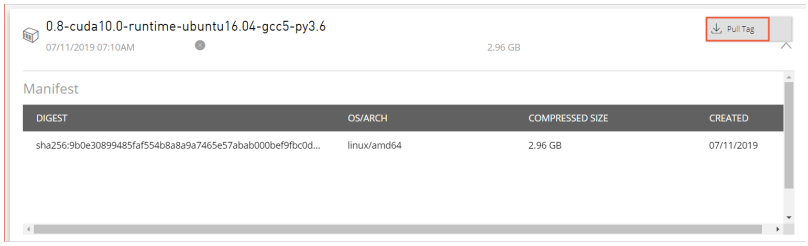
5. 获取docker pull命令。

   本文示例代码基于RAPIDS 0.8版本镜像编写，因此在运行本示例代码时，使用Tag为0.8版本的镜像。实际操作时，请选择您匹配的版本。

   i. 单击**Tags**页签。



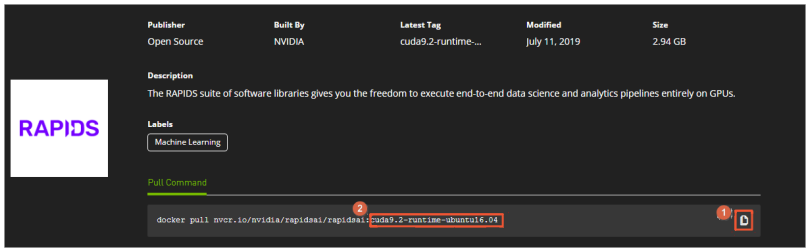
ii. 找到并复制Tag信息。本示例中，选择 `0.8-cuda10.0-runtime-ubuntu16.04-gcc5-py3.6` 。



iii. 返回页面顶部，复制**Pull Command**中的命令到文本编辑器，将镜像版本替换为对应的Tag信息，并保存。

本示例中，将 `cuda9.2-runtime-ubuntu16.04` 替换为 `0.8-cuda10.0-runtime-ubuntu16.04-gcc5-py3.6` 。

保存的docker pull命令用于下载RAPIDS镜像。关于如何下载RAPIDS镜像，请参见步骤三：部署RAPIDS环境。



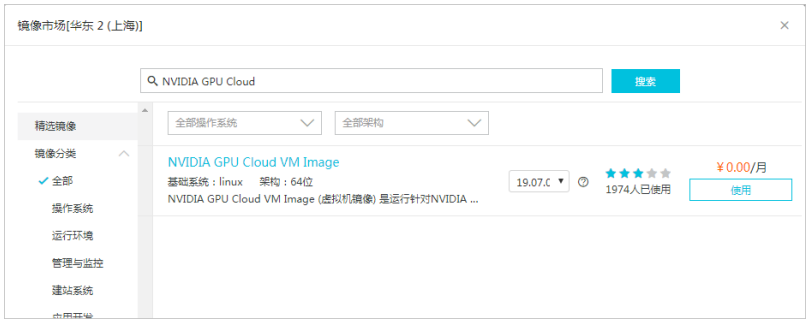
步骤三：部署RAPIDS环境

完成以下操作，部署RAPIDS环境：

1. 创建一台GPU实例。详细步骤请参见使用向导创建实例。

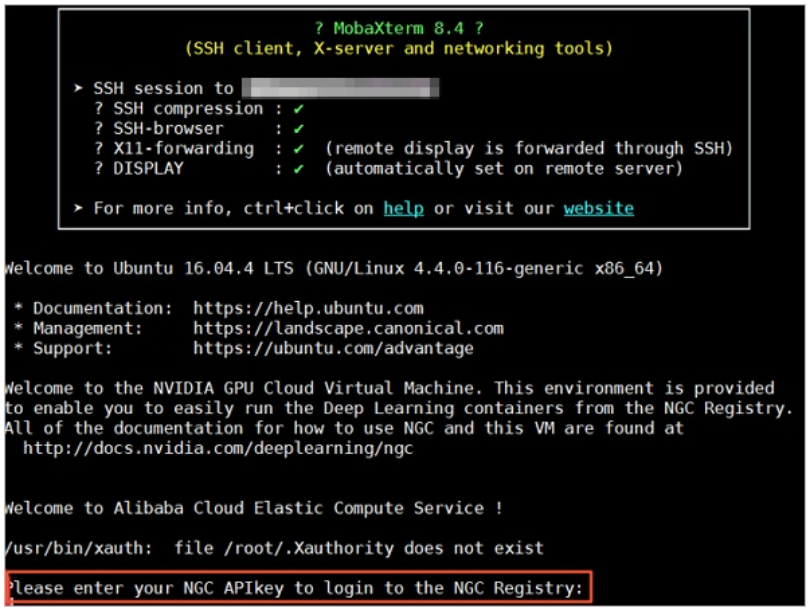
   参数配置说明如下：

   - **实例**：RAPIDS仅适用于特定的GPU型号（采用NVIDIA Pascal及以上架构），因此您需要选择GPU型号符合要求的实例规格，目前有gn6i、gn6v、gn5和gn5i，详细的GPU型号请参见实例规格族。建议您选择显存更大的gn6i、gn6v或gn5实例。本示例中，选用了显存为16 GB的GPU实例。
   - **镜像**：在镜像市场中搜索并使用 `NVIDIA GPU Cloud VM Image` 。



   - **公网带宽**：选择**分配公网IPv4地址**或者在实例创建成功后绑定EIP地址。具体操作，请参见弹性公网IP文档中的绑定ECS实例。
   - **安全组**：选择的安全组需要开放以下端口：
     - TCP 22 端口，用于SSH登录
     - TCP 8888端口，用于支持访问JupyterLab服务
     - TCP 8787端口、TCP 8786端口，用于支持访问Dask服务

2. 连接GPU实例。连接方式请参见连接方式介绍。
3. 输入NGC APIKey后按回车键，登录NGC容器环境。



4. （可选）运行**nvidia-smi**查看GPU型号、GPU驱动版本等GPU信息。

   建议您了解GPU信息，预判规避潜在问题。例如，如果NGC的驱动版本太低，新Docker镜像版本可能会不支持。

5. 运行docker pull命令下载RAPIDS镜像。

   关于如何获取docker pull命令，请参见步骤二：获取RAPIDS镜像下载命令。

```
docker pull nvcr.io/nvidia/rapidsai/rapidsai:0.8-cuda10.0-runtime-ubuntu16.04-gcc5-py3.6
```

6. （可选）查看下载的镜像。

   建议您查看Docker镜像信息，确保下载了正确的镜像。

```
docker images
```

7. 运行容器部署RAPIDS环境。

```
docker run --runtime=nvidia \
        --rm -it \
        -p 8888:8888 \
        -p 8787:8787 \
        -p 8786:8786 \
        nvcr.io/nvidia/rapidsai/rapidsai:0.8-cuda10.0-runtime-ubuntu16.04-gcc5-py3.6
```

步骤四：运行RAPIDS Demo

完成以下操作，运行RAPIDS Demo：

1. 在GPU实例上下载数据集和Demo文件。

```
# Get apt source address and download demos.
source_address=$(curl http://100.100.100.200/latest/meta-data/source-address|head -n 1)
source_address="${source_address}/opsx/ecs/linux/binary/machine_learning/"
cd /rapids
wget $source_address/rapids_notebooks_v0.8.tar.gz
tar -xzvf rapids_notebooks_v0.8.tar.gz
cd /rapids/rapids_notebooks_v0.8/xgboost
wget $source_address/data/mortgage/mortgage_2000_1gb.tgz
```

2. 在GPU实例上启动JupyterLab服务。

   推荐直接使用命令启动。

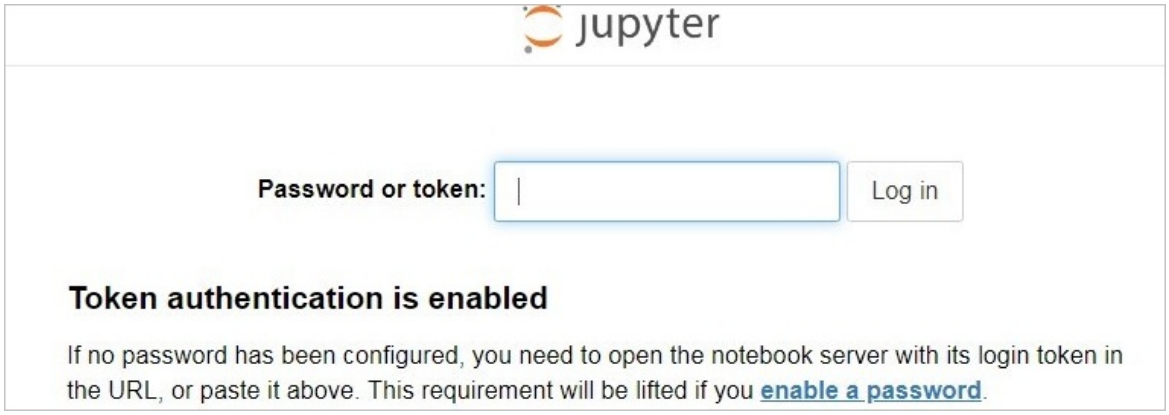
```
# Run the following command to start JupyterLab and set the password.
cd /rapids/rapids_notebooks_v0.8/xgboost
jupyter-lab --allow-root --ip=0.0.0.0 --no-browser --NotebookApp.token='YOUR PASSWORD'
# Exit JupyterLab.
sh ../utils/stop-jupyter.sh
```

   - 除使用命令外，您也可以执行脚本 `sh ../utils/start-jupyter.sh` 启动jupyter-lab，此时无法设置登录密码。
   - 您也可以连续按两次 `Ctrl+C` 退出JupyterLab服务。

3. 打开浏览器，在地址栏输入 `http://您的GPU实例IP地址:8888` 远程访问JupyterLab。

   > **说明** 推荐使用Chrome浏览器。

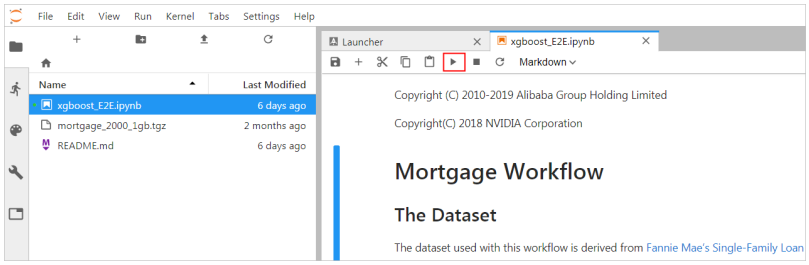
   如果您在启动JupyterLab服务时设置了登录密码，会跳转到密码输入界面。



4. 运行NoteBook代码。

   该案例是一个抵押贷款回归的任务，更多信息，请参见代码执行过程。登录成功后，可以看到NoteBook代码包括以下内容：

   - *xgboost_E2E.ipynb*文件： XGBoost Demo文件。双击文件可以查看文件详情，单击下图中的执行按钮可以逐步执行代码，每次执行一个Cell。



- *mortgage_2000_1gb.tgz*文件： 2000年的抵押贷款回归训练数据（1G分割的perf文件夹下的文件不会大于1 G，使用1 G分割的数据可以更有效的利用GPU显存）。

代码执行过程

该案例基于XGBoost演示了数据预处理到训练的端到端的过程，主要分为三个阶段：

- ETL（Extract-Transform-Load）：主要在GPU实例上进行。将业务系统的数据经过抽取、清洗转换之后加载到数据仓库。
- Data Conversion：在GPU实例上进行。将在ETL阶段处理过的数据转换为用于XGBoost训练的DMatrix格式。
- ML-Training：默认在GPU实例上进行。使用XGBoost训练梯度提升决策树 。

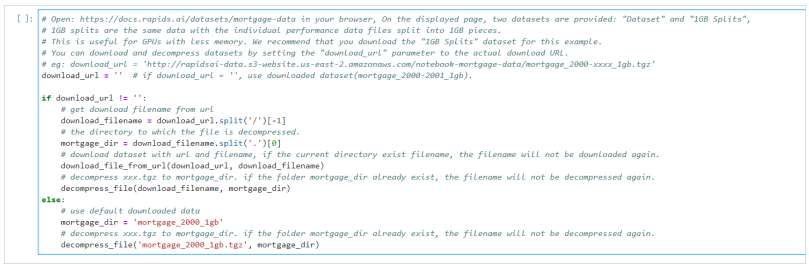
NoteBook代码的执行过程如下：

1. 准备数据集。

   本案例的Shell脚本会默认下载2000年的抵押贷款回归训练数据（*mortgage_2000_1gb.tgz*）。

   如果您想获取更多数据用于XGBoost模型训练，可以设定参数download_url指定下载路径，具体下载地址请参见Mortgage Data。

   示例效果如下：



```
[ ]: # Open: https://docs.rapids.ai/datasets/mortgage-data in your browser, On the displayed page, two datasets are provided: "Dataset" and "1GB Splits",
# 1GB splits are the same data with the individual performance data files split into 1GB pieces.
# This is useful for GPUs with less memory. We recommend that you download the "1GB Splits" dataset for this example.
# You can download and decompress datasets by setting the "download_url" parameter to the actual download URL.
# eg: download_url = 'http://rapidsai-data.s3-website.us-east-2.amazonaws.com/notebook-mortgage-data/mortgage_2000-xxxx_1gb.tgz'
download_url = ''  # if download_url = '', use downloaded dataset(mortgage_2000-2001_1gb).

if download_url != '':
    # get download filename from url
    download_filename = download_url.split('/')[-1]
    # the directory to which the file is decompressed.
    mortgage_dir = download_filename.split('.')[0]
    # download dataset with url and filename, if the current directory exist filename, the filename will not be downloaded again.
    download_file_from_url(download_url, download_filename)
    # decompress xxx.tgz to mortgage_dir. if the folder mortgage_dir already exist, the filename will not be decompressed again.
    decompress_file(download_filename, mortgage_dir)
else:
    # use default downloaded data
    mortgage_dir = 'mortgage_2000_1gb'
    # decompress xxx.tgz to mortgage_dir. if the folder mortgage_dir already exist, the filename will not be decompressed again.
    decompress_file('mortgage_2000_1gb.tgz', mortgage_dir)
```

2. 设定相关参数。

参数名称	说明
start_year	指定选择训练数据的起始时间，ETL时会处理start_year到end_year之间的数据。
end_year	指定选择训练数据的结束时间，ETL时会处理start_year到end_year之间的数据。
train_with_gpu	是否使用GPU进行XGBoost模型训练，默认为*True*。
gpu_count	指定启动worker的数量，默认为*1*。您可以按需要设定参数值，但不能超出GPU实例的GPU数量。
part_count	指定用于模型训练的performance文件的数量，默认为*2 * gpu_count*。如果参数值过大，在Data Conversion阶段会报错超出GPU内存限制，错误信息会在NoteBook后台输出。

示例效果如下：

```
[5]: acq_data_path = "{}/acq".format(mortgage_dir)
     perf_data_path = "{}/perf".format(mortgage_dir)
     col_names_path = "{}/names.csv".format(mortgage_dir)

     start_year = 2000
     end_year = 2000  # the end_year is inclusive

     # whether use GPUs for XGBoost training
     train_with_gpu = True

     # The number of GPUs to be used, value range: 1 to get_gpu_nums(). Default value: 1.
     # This parameter would use for starting dask-worker, doing ETL, doing Conversion and training model(if train_with_gpu=True).
     gpu_count = 1

     # The number of performance files in the perf folder
     part_number = len(os.listdir(perf_data_path))

     # if you download 1GB Splits train data(the filename end with '1gb.tgz'), each performance file is no large than 1GB,
     # in this example, a 16G GPU can process 2 or 3 preformance files. By default, one GPU is set to process 2 files.
     part_count = 2 * gpu_count if part_number >= 2 * gpu_count else part_number

     print('>>> Using "{}" GPU(GPUs).'.format(gpu_count))
     print('>>> ETL - process performance files from "{}" to "{}".'.format(start_year, end_year))
     print('>>> Data Conversion - select "{}" performance data processed in the ETL phase and convert to DMatrix-format for XGBoost training.'.format(part_count))
     print('>>> ML - Whether to use the GPU for XGBoost training: "{}".'.format(train_with_gpu))

     >>> Using "1" GPU(GPUs).
     >>> ETL - process performance files from "2000" to "2000".
     >>> Data Conversion - select "2" performance data processed in the ETL phase and convert to DMatrix-format for XGBoost training.
     >>> ML - Whether to use the GPU for XGBoost training: "True".
```

3. 启动Dask服务。

   代码会启动Dask Scheduler，并根据gpu_count参数启动worker用于ETL和模型训练。启动Dask服务后，您也可以通过Dask Dashboard直观地监控任务，打开方法请参见Dask Dashboard。

   示例效果如下：

```
# run dask-worker
cmd = "hostname --all-ip-addresses"
process = subprocess.Popen(cmd.split(), stdout=subprocess.PIPE)
output, error = process.communicate()
IPADDR = str(output.decode()).split()[0]

cluster = LocalCUDACluster(n_workers=gpu_count, ip=IPADDR)
client = Client(cluster)
client
```

Client	Cluster
• Scheduler: tcp://172.17.0.2:43894 • Dashboard: http://172.17.0.2:8787/status	• Workers: 1 • Cores: 1 • Memory: 507.25 GB

4. 启动ETL。

   ETL阶段会进行到表关联、分组、聚合、切片等操作，数据格式采用cuDF库的DataFrame格式（类似于pandas的DataFrame格式）。

   示例效果如下：

ETL

Perform all of ETL with a single call to

```
process_quarter_gpu(year=year, quarter=quarter, perf_file=file)
```

```
[ ]: %%time

     # NOTE: The ETL calculates additional features which are then dropped before creating the XGBoost DMatrix.
     # This can be optimized to avoid calculating the dropped features.

     gpu_dfs = []
     gpu_time = 0
     quarter = 1
     year = start_year
     count = 0
     while year <= end_year:
         for file in glob(os.path.join(perf_data_path + "/Performance_" + str(year) + "Q" + str(quarter) + "*")):
             gpu_dfs.append(process_quarter_gpu(year=year, quarter=quarter, perf_file=file))
             count += 1
         quarter += 1
         if quarter == 5:
             year += 1
             quarter = 1
     wait(gpu_dfs)

     CPU times: user 560 ms, sys: 28 ms, total: 588 ms
     Wall time: 20.9 s
```

5. 启动Data Conversion。

   将DataFrame格式的数据转换为用于XGBoost训练的DMatrix格式，每个worker处理一个DMatrix对象。

示例效果如下：

6. 启动ML Training。

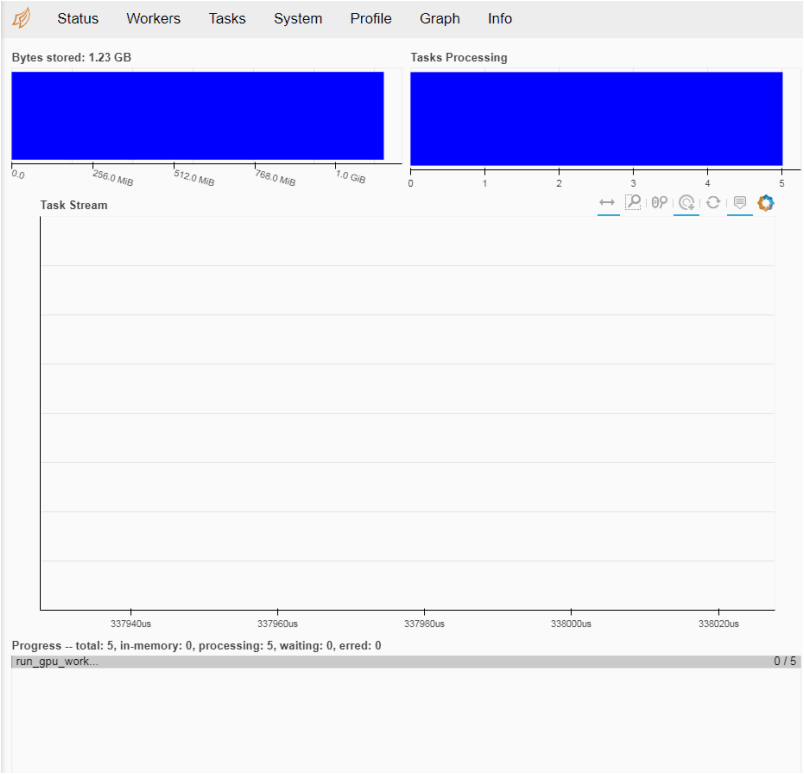
使用dask-xgboost启动模型训练，dask-xgboost负责多个dask worker间的通信协同工作，底层仍然调用xgboost执行模型训练。

示例效果如下：

Dask Dashboard

Dask Dashboard支持任务进度跟踪、任务性能问题识别和故障调试。

Dask服务启动后，在浏览器地址栏中访问 `http://您的GPU实例IP地址:8787/status` 即可进入Dashboard主界面。



相关函数

函数功能	函数名称
下载文件	def download_file_from_url(url, filename):
解压文件	def decompress_file(filename, path):
获取当前机器的GPU个数	def get_gpu_nums():
管理GPU内存	• def initialize_rmm_pool(): • def initialize_rmm_no_pool(): • def run_dask_task(func, **kwargs):
提交DASK任务	• def process_quarter_gpu(year=2000, quarter=1, perf_file=""): • def run_gpu_workflow(quarter=1, year=2000, perf_file="", **kwargs):

函数功能	函数名称
使用cuDF从CSV中加载数据	• def gpu_load_performance_csv(performance_path, **kwargs): • def gpu_load_acquisition_csv(acquisition_path, **kwargs): • def gpu_load_names(**kwargs):
处理和提取训练数据的特征	• def null_workaround(df, **kwargs): • def create_ever_features(gdf, **kwargs): • def join_ever_delinq_features(everdf_tmp, delinq_merge, **kwargs): • def create_joined_df(gdf, everdf, **kwargs): • def create_12_mon_features(joined_df, **kwargs): • def combine_joined_12_mon(joined_df, testdf, **kwargs): • def final_performance_delinquency(gdf, joined_df, **kwargs): • def join_perf_acq_gdfs(perf, acq, **kwargs): • def last_mile_cleaning(df, **kwargs):

相关文档

- 在GPU实例上使用RAPIDS加速图像搜索任务

5.在GPU实例上使用RAPIDS加速图像搜索任务

本文以使用RAPIDS加速图像搜索任务为例，介绍如何在预装镜像的GPU实例上使用RAPIDS加速库。

前提条件

使用本教程进行操作前，请确保您已经注册了阿里云账号。如还未注册，请先完成账号注册。

背景信息

RAPIDS，全称Real-time Acceleration Platform for Integrated Data Science，是NVIDIA针对数据科学和机器学习推出的GPU加速库。更多RAPIDS信息请参见官方网站。

基于图像识别和搜索，图像搜索任务可以实现以图搜图，在不同行业应用和业务场景中帮助您搜索相同或相似的图片。

图像搜索任务背后的两项主要技术是特征提取及向量化、向量索引和检索。本文案例中，使用开源框架TensorFlow和Keras配置生产环境，然后使用ResNet50卷积神经网络完成图像的特征提取及向量化，最后使用RAPIDS cuML库的KNN算法实现BF方式的向量索引和检索。

> **说明** BF（Brute Force）检索方法是一种百分百准确的方法，对距离衡量算法不敏感，适用于所有的距离算法。

本文案例在阿里云gn6v（NVIDIA Tesla V100）实例上执行。执行案例后，对比了GPU加速的RAPIDS cuml KNN与CPU实现的scikit-learn KNN的性能，可以看到GPU加速的KNN向量检索速度为CPU的近600倍。

本文案例为单机单卡的版本，即一台GPU实例搭载一块GPU卡。

操作步骤

执行以下操作完成一次图像搜索任务：

1. 创建GPU实例
2. 启动和登录JupyterLab
3. 执行图像搜索案例

步骤一：创建GPU实例

具体步骤请参见使用向导创建实例。

- **实例**：RAPIDS仅适用于特定的GPU型号（采用NVIDIA Pascal及以上架构），因此您需要选择GPU型号符合要求的实例规格，目前有gn6i、gn6v、gn5和gn5i。本文案例中，选用了ecs.gn6v-c8g1.2xlarge实例规格。
- **镜像**：在镜像市场中使用关键字*RAPIDS*，搜索并使用预装了RAPIDS加速库的镜像。

- **安全组**：选择的安全组需要开放TCP 8888端口，用于支持访问JupyterLab服务。

步骤二：启动和登录JupyterLab

1. 连接GPU实例，运行以下命令启动JupyterLab服务。

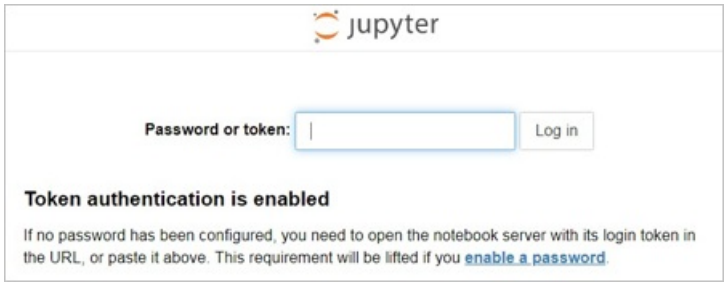
   > **说明** 连接GPU实例的步骤请参见连接方式概述ECS远程连接操作指南。

   ```
   # Go to the notebooks directory.
   cd /rapids
   # Run the following command to start JupyterLab and set the logon password:
   jupyter-lab --allow-root --ip=0.0.0.0 --no-browser --NotebookApp.token='your logon pass
   word'
   # Exit jupyterlab: press Ctrl+C twice.
   ```

2. 在您的本地机器上打开浏览器。输入 `http://(IP address of your GPU instance):8888` 远程访问JupyterLab。

   > **说明** 推荐使用Chrome浏览器。

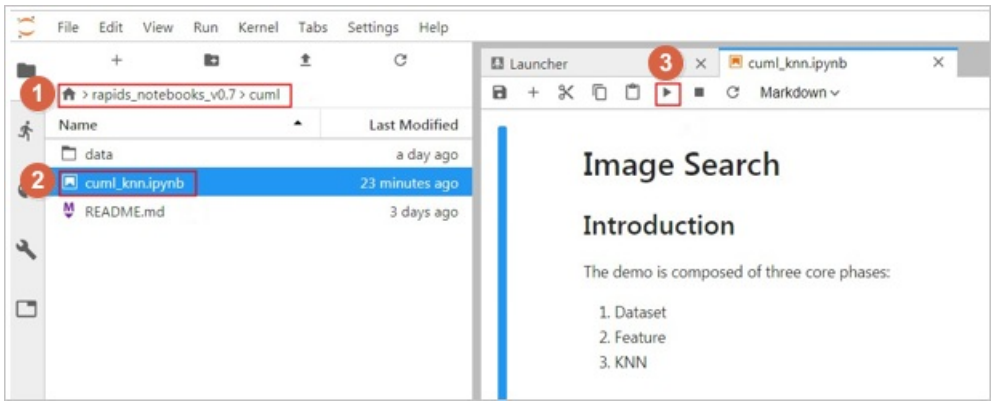
3. 输入启动命令中设置的密码，然后单击**Log in**。



步骤三：执行图像搜索案例

1. 进入案例所在目录*rapids_notebooks_v0.7/cuml*。
2. 双击*cuml_knn.ipynb*文件。
3. 单击 ▶ 。

> 说明　单击一次执行一个cell，请单击至案例执行结束，详细说明请参见案例执行过程。



案例执行过程

图像搜索案例的执行过程分为三个步骤：处理数据集、提取图片特征和搜索相似图片。本文案例结果中对比了GPU加速的RAPIDS cuml KNN与CPU实现的scikit-learn KNN的性能。

1. 处理数据集。
   i. 下载和解压数据集。

      本文案例中使用了STL-10数据集，该数据集中包含10万张未打标的图片，图片的尺寸均为：96 x 96 x 3。您可以使用其他数据集，为便于提取图片特征，请确保数据集中图片的尺寸相同。

      本文案例提供了 `download_and_extract(data_dir)` 方法供您下载和解压STL-10数据集。RAPIDS镜像中已经将数据集下载到*./data*目录，您可以执行 `download_and_extract()` 方法直接解压数据集。

      Download and Decompression

      ```
      [2]: # the directory to save data
           data_dir = './data'
           # download and decompression
           download_and_extract(data_dir)

           >>> stl10_binary.tar.gz has exist in current directory.
           >>> Decompressing from ./data/stl10_binary.tar.gz....
           Successfully decompressed
      ```

   ii. 读取图片。

      从数据集解压出的数据为二进制格式，执行 `read_all_images(path_to_data)` 方法加载数据并转换为NHWC（batch, height, width, channels）格式，以便用Tensorflow提取图片特征。

      Read Data

      ```
      [3]: # the path of unlabeled data
           path_unlabeled = os.path.join(data_dir, 'stl10_binary/unlabeled_X.bin')
           # get images from binary
           images = read_all_images(path_unlabeled)
           print('>>> images shape: ', images.shape)

           >>> images shape:  (100000, 96, 96, 3)
      ```

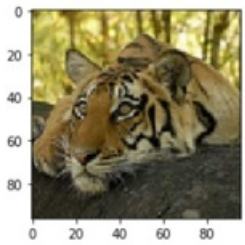
iii. 展示图片。

执行 `show_image(image)` 方法随机展示一张数据集中的图片。

Show Image

```
[4]: import random
     import matplotlib.pyplot as plt
     %matplotlib inline

     def show_image(image):
         """show image"""
         fig = plt.figure(figsize=(3, 3))
         plt.imshow(image)
         plt.show()
         fig.clear()
```

```
[10]: # random show a image
      rand_image_index = random.randint(0, images.shape[0])
      show_image(images[rand_image_index])
```



iv. 分割数据集。

按照9:1的比例把数据集分为两部分，分别用于创建图片索引库和搜索图片。

Split Dataset

```
from sklearn.model_selection import train_test_split

train_images, query_images = train_test_split(images, test_size=0.1, random_state=123)
print('train_images shape: ', train_images.shape)
print('query_images shape: ', query_images.shape)
```

```
train_images shape:  (90000, 96, 96, 3)
query_images shape:  (10000, 96, 96, 3)
```

2. 提取图片特征。

使用开源框架Tensorflow和Keras提取图片特征，其中模型为基于ImageNet数据集的ResNet50（notop）预训练模型。

i. 设定Tensorflow参数。

Tensorflow默认使用所有GPU显存，我们需要留出部分GPU显存供cuML使用。您可以选择一种方法设置GPU显存参数：

- 方法1：依据运行需求进行显存分配。

```
config.gpu_options.allow_growth = True
```

- 方法2：设定可以使用的GPU显存比例。本案例中使用方法2，并且GPU显存比例默认设置为0.3，即Tensorflow可以使用整块GPU显存的30%，您可以依据应用场景修改比例。

```
config.gpu_options.per_process_gpu_memory_fraction = 0.3
```

Image Features

```
# set tensorflow params to adjust GPU memory usage, if use default params, tensorflow would use
# nearly all of the gpu memory, we need reserve some gpu memory for cuml.
import os
# only use device 0
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

import tensorflow as tf
from keras.backend.tensorflow_backend import set_session
config = tf.ConfigProto()
# method 1: allocate gpu memory base on runtime allocations
# config.gpu_options.allow_growth = True
# method 2: determines the fraction of the onerall amount of memory
# that each visibel GPU should be allocated.
config.gpu_options.per_process_gpu_memory_fraction = 0.3
set_session(tf.Session(config=config))
```

```
Using TensorFlow backend.
```

ii. 下载ResNet50（notop）预训练模型。

连接公网下载模型（大小约91 M），下载完成后默认保存到*/root/.keras/models/*目录。

参数名称	说明
weights	取值范围： ■ *None*：随机初始化权重值。 ■ *imagenet*：权重值的初始值设置为通过ImageNet预训练过的模型的权重值。 本案例中设置为*imagenet*。
include_top	取值范围： ■ True：包含整个ResNet50网络结构的最后一个全链接层。 ■ False：不包含整个ResNet50网络结构的最后一个全链接层。 本案例中，使用神经网络模型ResNet50的主要目的是提取图片特征而非分类图片，因此设置为*False*。
input_shape	可选参数，用于设置图片的输入shape，仅在include_top设置为*False*时生效。您必须为图片设置3个inputs channels，且宽和高不应低于32。此处设为*(96, 96, 3)*。
pooling	在include_top设置为*False*时，您需要设置池化层模式，取值范围： ■ *None*：输出为4D tensor。 ■ *avg*：平均池化，输出为2D tensor。 ■ *max*：最大池化，输出为2D tensor。 本案例中设置为*max*。

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input

# download resnet50(notop) model(first running) and load model
model = ResNet50(weights='imagenet', include_top=False, input_shape=(96, 96, 3), pooling='max')
```

```
WARNING:tensorflow:From /root/anaconda3/envs/rapids/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263:
colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_k
ernels_notop.h5
94658560/94653016 [==============================] - 9s 0us/step
```

您可以执行 `model.summary()` 方法查看模型的网络结构。

```
[10]: # network summary
      model.summary()
```

```
add_16 (Add)                    (None, 3, 3, 2048)   0           bn5c_branch2c[0][0]
                                                                 activation_46[0][0]
__________________________________________________________________________________________________
activation_49 (Activation)      (None, 3, 3, 2048)   0           add_16[0][0]
__________________________________________________________________________________________________
global_max_pooling2d_1 (GlobalM (None, 2048)         0           activation_49[0][0]
==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

iii. 提取图片特征。

对分割得到的两个图片数据集执行 `model.predict()` 方法提取图片特征。

```
[10]: %%time
train_features = model.predict(train_images)
print('train features shape: ', train_features.shape)

train features shape:  (90000, 2048)
CPU times: user 33.6 s, sys: 7.94 s, total: 41.5 s
Wall time: 36.3 s

[11]: %%time
query_features = model.predict(query_images)
print('query features shape: ', query_features.shape)

query features shape:  (10000, 2048)
CPU times: user 3.64 s, sys: 704 ms, total: 4.34 s
Wall time: 3.76 s
```

3. 搜索相似图片。

i. 使用cuml KNN搜索相似图片。

通过 `k=3` 设置K值为3，即查找最相似的3张图片，您可以依据使用场景自定义K值。

其中， `knn_cuml.fit()` 方法为创建索引阶段， `knn_cuml.kneighbors()` 为搜索近邻阶段。

```
cuml KNN

[12]: from cuml.neighbors import NearestNeighbors

[13]: %%time
knn_cuml = NearestNeighbors()
knn_cuml.fit(train_features)

CPU times: user 888 ms, sys: 60 ms, total: 948 ms
Wall time: 192 ms

[14]: %%time
distances_cuml, indices_cuml = knn_cuml.kneighbors(query_features, k=3)

CPU times: user 1.59 s, sys: 492 ms, total: 2.08 s
Wall time: 791 ms
```

KNN向量检索耗时791 ms。

ii. 使用scikit-learn KNN搜索相似图片。

通过 `n_neighbors=3` 设置K值为3，通过 `n_jobs=-1` 设置使用所有CPU进行近邻搜索。

> **说明** ecs.gn6v-c8g1.2xlarge的配置为8 vCPU。

```
sklearn KNN

[15]: from sklearn.neighbors import NearestNeighbors

[16]: %%time
knn_sk = NearestNeighbors(n_neighbors=3, metric='sqeuclidean', n_jobs=-1)
knn_sk.fit(train_features)

CPU times: user 856 ms, sys: 36 ms, total: 892 ms
Wall time: 114 ms

[17]: %%time
distances_sk, indices_sk = knn_sk.kneighbors(query_features, 3)

CPU times: user 18.2 s, sys: 29.9 s, total: 48.1 s
Wall time: 7min 34s
```

KNN向量检索耗时7分34秒。

iii. 对比cuml KNN和scikit-learn KNN的搜索结果。

对比两种方式的KNN向量检索速度，使用GPU加速的cuml KNN耗时791 ms，使用CPU的scikit-learn KNN耗时7min 34s。前者为后者的近600倍。

验证两种方式的输出结果是否相同，输出结果为两个数组：

- distance：最小的K个距离值。本案例中搜索了10000张图片，K值为3，因此`distance.shape=(10000, 3)`。
- indices：对应的图片索引。`indices.shape=(10000, 3)`。

由于本案例所用数据集中存在重复图片，容易出现图片相同但索引不同的情况，因此使用distances，不使用indices对比结果。考虑到计算误差，如果两种方法得出的10000张图片中的3个最小距离值误差都小于1，则认为结果相同。

Compare

```
# compare the distance obtained while using sklearn and cuml models
(np.abs(distances_cuml - distances_sk) < 1).all()
```

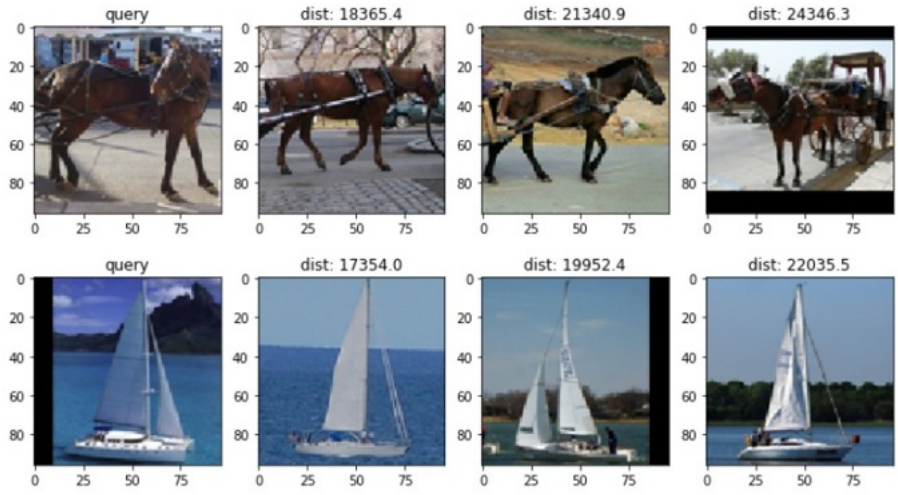
```
True
```

图片搜索结果

本案例从1万张搜索图片中随机选择5张图片并搜索相似图片，最终展示出5行4列图片。

第一列为搜索图片，第二列至第四列为图片索引库中的相似图片，且相似性依次递减。每张相似图片的标题为计算的距离，数值越大相似性越低。

```
# get random indices
random_show_index = np.random.randint(0, query_images.shape[0], size=5)
random_query = query_images[random_show_index]
random_indices = indices_cuml[random_show_index].astype(np.int)
random_distances = distances_cuml[random_show_index]

# show result images
for query_image, sim_indices, sim_dists in zip(random_query, random_indices, random_distances):
    sim_images = train_images[sim_indices]
    show_images(query_image, sim_images, sim_dists)
```



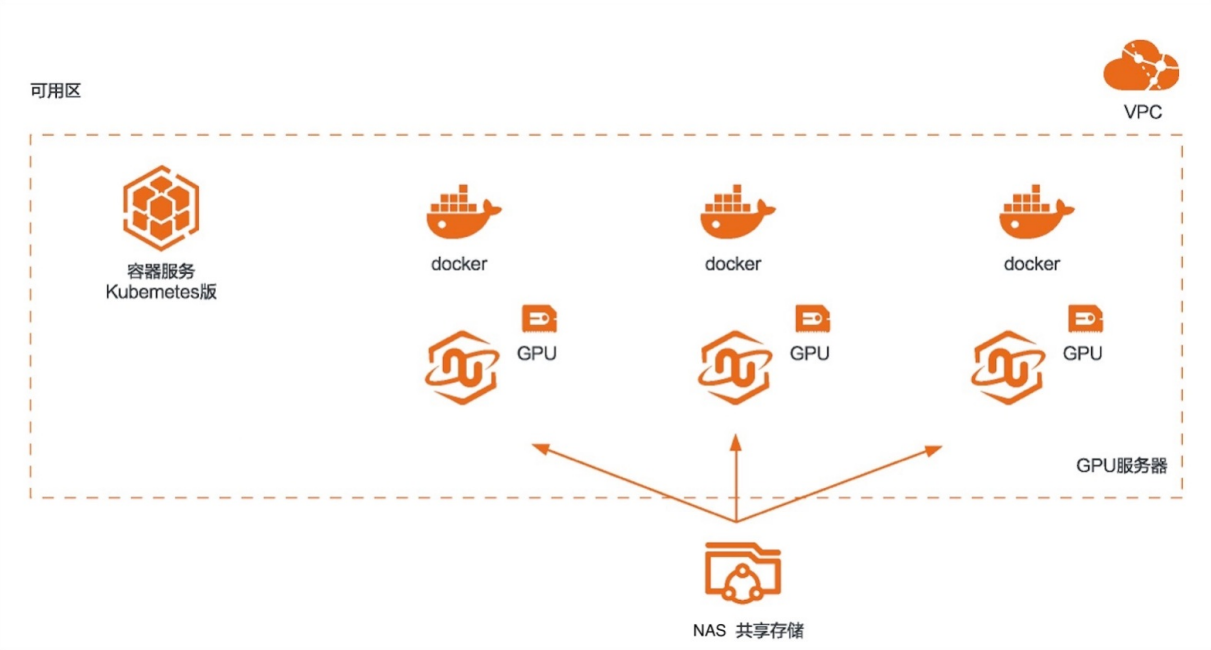
6.RAPIDS加速机器学习最佳实践

本方案适用于使用RAPIDS加速库和GPU云服务器来对机器学习任务或者数据科学任务进行加速的场景。相比CPU，利用GPU和RAPIDS在某些场景下可以取得非常明显的加速效果。

解决的问题

- 搭建RAPIDS加速机器学习环境。
- 使用容器服务Kubernetes版部署RAPIDS环境。
- 使用NAS存储计算数据。

部署架构图



选用的产品

- GPU服务器

  GPU云服务器是基于GPU应用的计算服务，多适用于AI深度学习、视频处理、科学计算、图形可视化等应用场景。

  更多关于GPU服务器的介绍，请参见GPU服务器产品详情页。

- 文件存储NAS

  阿里云文件存储NAS是一个可共享访问、弹性扩展、高可靠、高性能的分布式文件系统。兼容POSIX文件接口，可支持数千台计算节点共享访问，可以挂载到弹性计算ECS、弹性裸金属服务器、容器服务ACK、弹性容器ECI、批量计算BCS、高性能计算EHPC、AI训练PAI等计算业务上提供高性能的共享存储，用户无需修改应用程序，即可无缝迁移业务系统上云。

  更多关于文件存储NAS的介绍，请参见文件存储NAS产品详情页。

- 容器服务 ACK

  容器服务Kubernetes版（ACK）提供高性能可伸缩的容器应用管理能力，支持企业级容器化应用的全生命周期管理。整合阿里云虚拟化、存储、网络和安全能力，打造云端最佳容器化应用运行环境。

更多关于容器服务ACK的介绍，请参见容器服务 ACK产品详情页。

详细信息

点击查看最佳实践详情

更多最佳实践

点击查看更多阿里云最佳实践

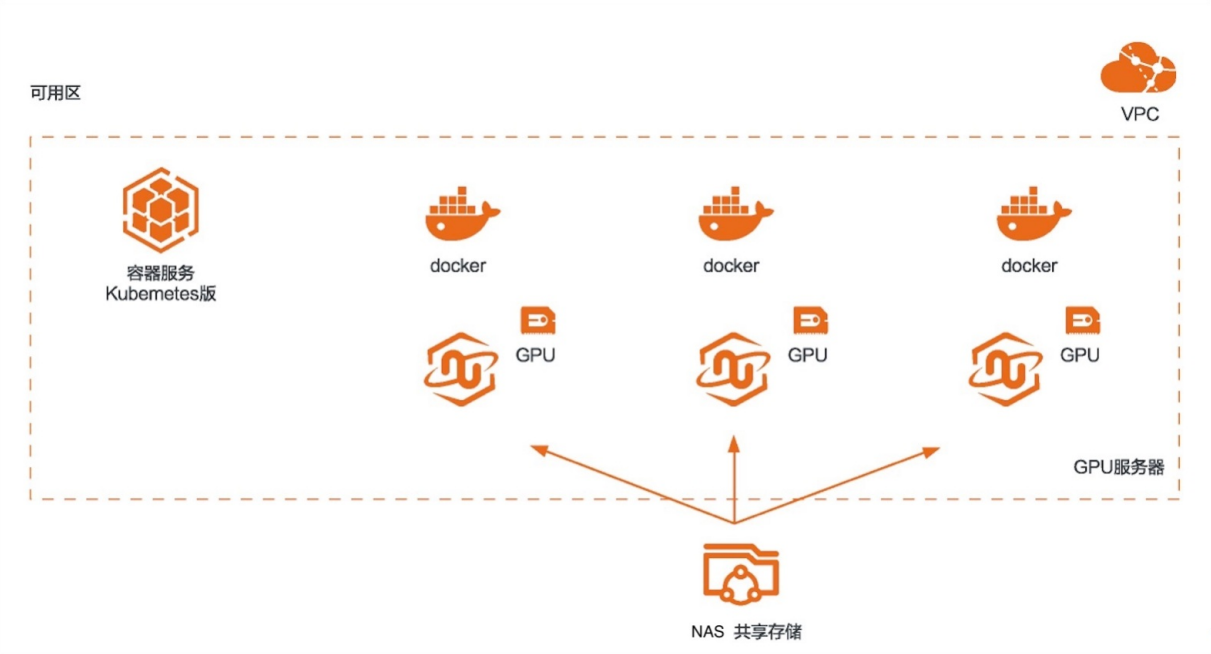
7.RAPIDS加速图像搜索最佳实践

本方案适用于使用RAPIDS加速平台和GPU云服务器来对图像搜索任务进行加速的场景。相比CPU，利用GPU+RAPIDS在图像搜索场景下可以取得非常明显的加速效果。

解决的问题

- 搭建RAPIDS加速图像搜索环境。
- 使用容器服务Kubernetes版部署图像搜索环境。
- 使用NAS存储计算数据。

部署架构图



选用的产品

- GPU服务器

  GPU云服务器是基于GPU应用的计算服务，多适用于AI深度学习、视频处理、科学计算、图形可视化等应用场景。

  更多关于GPU服务器的介绍，请参见GPU服务器产品详情页。

- 文件存储NAS

  阿里云文件存储NAS是一个可共享访问、弹性扩展、高可靠、高性能的分布式文件系统。兼容POSIX文件接口，可支持数千台计算节点共享访问，可以挂载到弹性计算ECS、弹性裸金属服务器、容器服务ACK、弹性容器ECI、批量计算BCS、高性能计算EHPC、AI训练PAI等计算业务上提供高性能的共享存储，用户无需修改应用程序，即可无缝迁移业务系统上云。

  更多关于文件存储NAS的介绍，请参见文件存储NAS产品详情页。

- 容器服务 ACK

  容器服务Kubernetes版（ACK）提供高性能可伸缩的容器应用管理能力，支持企业级容器化应用的全生命周期管理。整合阿里云虚拟化、存储、网络和安全能力，打造云端最佳容器化应用运行环境。

更多关于容器服务ACK的介绍，请参见容器服务 ACK产品详情页。

详细信息

点击查看最佳实践详情

更多最佳实践

点击查看更多阿里云最佳实践