ModelArts

用户指南（Lite Server）

文档版本 01

发布日期 2024-11-20

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

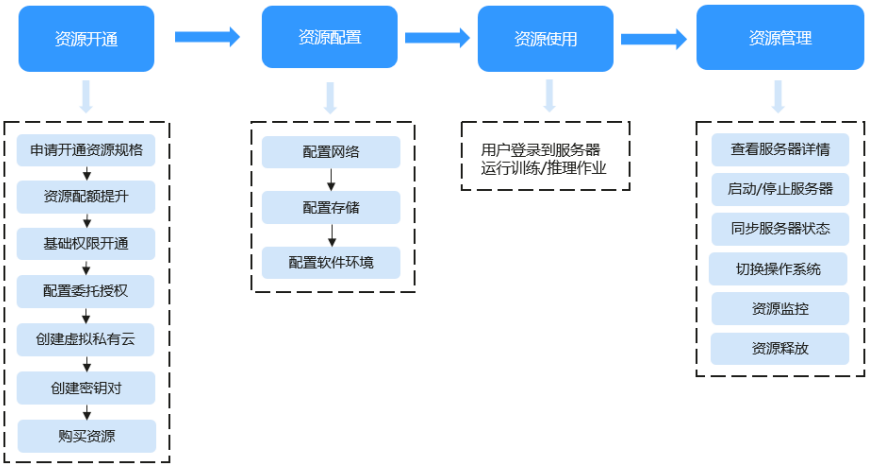
1 Lite Server 使用前必读

1.1 Lite Server 使用流程

ModelArts Lite Server提供多样化的xPU裸金属服务器，赋予用户以root账号自主安装和部署AI框架、应用程序等第三方软件的能力，为用户打造专属的云上物理服务器环境。用户只需轻松选择服务器的规格、镜像、网络配置及密钥等基本信息，即可迅速创建弹性裸金属服务器，获取所需的云上物理资源，充分满足算法工程师在日常训练和推理工作中的需求。

本文旨在帮助您了解Lite Server的基本使用流程，帮助您快速上手，使用流程包含以下步骤。

图 1-1 使用流程



1. 资源开通

   由于Server为一台裸金属服务器，因此需要先购买资源后才能使用。

   a. 首先请联系客户经理确认Server资源方案，部分规格为受限规格，因此需要申请开通您所需的资源规格。

   b. Server所需资源可能会超出华为云默认提供的资源配额（如ECS、EIP、SFS），因此需要提交工单提升资源配额。

c. 为子用户账号开通Server功能所需的基础权限。
d. 由于ModelArts服务在使用过程中会访问其他依赖服务，因此需要给ModelArts进行委托授权。
e. 购买Server资源时，需要选择虚拟私有云用于网络通信，您可以使用已有的虚拟私有云或新创建的虚拟私有云。
f. 若使用密钥对作为登录裸金属服务器的鉴权方式，您可以使用已有的密钥对或新创建的密钥对。
g. 在ModelArts控制台购买Server资源。

2. 资源配置

   完成资源购买后，需要对网络、存储、软件环境进行相关配置。

3. 资源使用

   完成资源配置后，您可以登录到服务器进行训练和推理，具体案例可参考**Lite Server资源使用**。

4. 资源管理

   Lite Server提供启动、停止、切换操作系统等管理手段，您可在ModelArts控制台上对资源进行管理。

表 1-1 相关名词解释

名词	含义
裸金属服务器	裸金属服务器是一款兼具虚拟机弹性和物理机性能的计算类服务，为您和您的企业提供专属的云上物理服务器，为核心数据库、关键应用系统、高性能计算、大数据等业务提供卓越的计算性能以及数据安全。 由于Server是一台裸金属服务器，在ModelArts管理控制台购买Server后，会在BMS管理控制台上创建一台与Server对应的裸金属服务器，后续挂载磁盘、绑定弹性网络IP等操作可在BMS服务控制台上完成。
xPU	xPU泛指GPU和NPU。 ● GPU，即图形处理器，主要用于加速深度学习模型的训练和推理。 ● NPU，即神经网络处理器，是专门为加速神经网络计算而设计的硬件。与GPU相比，NPU在神经网络计算方面具有更高的效率和更低的功耗。
密钥对	弹性裸金属支持SSH密钥对的方式进行登录，用户无需输入密码就可以登录到弹性裸金属服务器，因此可以防止由于密码被拦截、破解造成的账户密码泄露，从而提高弹性裸金属服务器的安全性。 **说明** 为保证云服务器安全，未进行私钥托管的私钥只能下载一次，请妥善保管。
虚拟私有云	虚拟私有云（Virtual Private Cloud，VPC）为裸金属服务器构建隔离的、用户自主配置和管理的虚拟网络环境，提升用户云中资源的安全性，简化用户的网络部署。您可以在VPC中定义安全组、VPN、IP地址段、带宽等网络特性。用户可以通过VPC方便地管理、配置内部网络，进行安全、快捷的网络变更。同时，用户可以自定义安全组内与组间的访问规则，加强裸金属服务器的安全保护。

1.2 Lite Server 高危操作一览表

ModelArts Lite Server在日常操作与维护过程中涉及的高危操作，需要严格按照操作指导进行，否则可能会影响业务的正常运行。

高危操作风险等级说明：

- 高：对于可能直接导致业务失败、数据丢失、系统不能维护、系统资源耗尽的高危操作。
- 中：对于可能导致安全风险及可靠性降低的高危操作。
- 低：高、中风险等级外的其他高危操作。

表 1-2 高危操作一览表

操作对象	操作名称	风险描述	风险等级	应对措施
操作系统	升级/修改操作系统内核或者驱动。	如果升级/修改操作系统内核或者驱动，很可能导致驱动和内核版本不兼容，从而导致OS无法启动，或者基本功能不可用。相关高危命令如：apt-get upgrade。	高	如果需要升级/修改，请**联系华为云技术支持**。
	切换或者重置操作系统。	服务器在进行过“切换或者重置操作系统”操作后，EVS系统盘ID发生变化，和下单时订单中的EVS ID已经不一致， 因此EVS系统盘将不支持扩容，并显示信息：“当前订单已到期，无法进行扩容操作，请续订”。	中	切换或者重置操作系统后，建议通过挂载数据盘EVS或挂载SFS盘等方式进行存储扩容。

1.3 Lite Server 算力资源和镜像版本配套关系

Lite Server提供多种NPU、GPU镜像，您可在购买前了解当前支持的镜像及对应详情。

NPU Snt9 裸金属服务器支持的镜像详情

镜像名称：ModelArts-Euler2.8_Aarch64_Snt9_C78

表 1-3 镜像详情

软件类型	版本详情
操作系统	EulerOS 2.0 (SP8)
内核版本	4.19.36-vhulk1907.1.0.h619.eulerosv2r8.aarch64
架构类型	aarch64
mlnx-ofed-linux	21.0.2

NPU Snt9B 裸金属服务器支持的镜像详情

- 镜像名称：EulerOS2.10-Arm-64bit-for-Snt9B-BareMetal-with-23.0.6-7.1.0.9.220-CANN7.0.1.5

  表 1-4 镜像详情

软件类型	版本详情
操作系统	EulerOS 2.10
内核版本	Linux 4.19.90-vhulk2211.3.0.h1543.eulerosv2r10.aarch64
架构类型	aarch64
固件版本	7.1.0.9.220
npu-driver	23.0.6
Ascend-cann-toolkit	7.0.1.5
cann-kernels	7.0.1.5
Ascend-mindx-toolbox	5.0.1.1
Docker	24.0.7
Ascend-docker-runtime	5.0.1.1
MindSpore Lite	2.1.0-cp37-cp37m
Mpich	3.2.1

- 镜像名称：HCE2.0-Arm-64bit-for-Snt9B-BareMetal-with-23.0.6-7.1.0.9.220-CANN7.1.0.5

表 1-5 镜像详情

软件类型	版本详情
操作系统	HCE2.0
内核版本	Linux 5.10.0-60.18.0.50.r865_35.hce2.aarch64
架构类型	aarch64
固件版本	7.1.0.9.220
npu-driver	23.0.6
Ascend-cann-toolkit	7.0.1.5
cann-kernels	7.0.1.5
Ascend-mindx-toolbox	5.0.1.1
Docker	18.09
Ascend-docker-runtime	5.0.1.1
MindSpore Lite	2.1.0-cp37-cp37m
Mpich	4.1.3

GP Ant8 裸金属服务器支持的镜像详情

- 镜像名称：Ubuntu-20.04-for-Ant8-with-RoCE-and-NVIDIA-525-CUDA-12.0-Uniagent

表 1-6 镜像详情

软件类型	版本详情
操作系统	Ubuntu 20.04 server 64bit
内核版本	5.4.0-144-generic
架构类型	x86
驱动版本	525.105.17
cuda	12.0
container-toolkit	1.13.3-1
fabricmanager	525.105.17
mlnx-ofed-linux	5.8-2.0.3.1-ubuntu20.04-x86_64
peer-memory-dkms	1.2-0
libnccl2	2.18.1

软件类型	版本详情
nccl-test	v.2.13.6
docker	20.10.23
RoCE路由配置	支持

- 镜像名称：Ubuntu-20.04-for-Ant8-with-RoCE-and-NVIDIA-515-CUDA-11.7-Uniagent（乌兰察布一、北京四、乌兰察布-汽车一）

表 1-7 镜像详情

软件类型	版本详情
操作系统	Ubuntu 20.04 server 64bit
内核版本	5.4.0-144-generic
架构类型	x86
驱动版本	515.105.01
cuda	11.7
container-toolkit	1.13.3-1
fabricmanager	515.105.01-1
mlnx-ofed-linux	5.8-2.0.3.1-ubuntu20.04-x86_64
peer-memory-dkms	1.2-0
libnccl2	2.14.3
nccl-test	v.2.13.6
docker	20.10.23
RoCE路由配置	支持

GP Vnt1 裸金属服务器支持的镜像详情

说明

Vnt1规格在北京四、北京一和上海一虽然规格相同，但是产品的配置、发布时间都存在很大差异，因此镜像不能共用。

- 镜像名称：Ubuntu-18.04-for-BareMetal-Vnt1-p3-with-NVIDIA-470-CUDA-11.4-Uniagent（仅限于北京一、北京四、广州）

表 1-8 镜像详情

软件类型	版本详情
操作系统	Ubuntu 18.04 server 64bit

软件类型	版本详情
内核版本	4.15.0-45-generic
架构类型	x86
驱动版本	470.182.03
cuda	11.4
container-toolkit	1.15.0.-1
mlnx-ofed-linux	5.7-1.0.2.1-ubuntu18.04-x86_64
libnccl2	2.10.3-1
nccl-test	v2.13.9
docker	24.0.2

- 镜像名称：Ubuntu-18.04-for-BareMetal-Vnt1-p6-with-NVIDIA-470-CUDA-11.4-Uniagent（仅限于上海一）

  表 1-9 镜像详情

软件类型	版本详情
操作系统	Ubuntu 18.04 server 64bit
内核版本	4.15.0-45-generic
架构类型	x86
驱动版本	470.182.03
cuda	11.4
container-toolkit	1.15.0.-1
mlnx-ofed-linux	5.7-1.0.2.1-ubuntu18.04-x86_64
libnccl2	2.10.3-1
nccl-test	v2.13.9
docker	24.0.2

- 镜像名称：Euler2.9-X86-for-Vnt1-BareMetal（仅限于北京四和上海一）

  表 1-10 镜像详情

软件类型	版本详情
操作系统	EulerOS 2.9 64bit
架构类型	x86

- 镜像名称：CentOS-7.9-64bit-for-BareMetal-Vnt1-with-NVIDIA-515-CUDA-11.7-Uniagent（仅限于北京一、北京四、广州）

  表 1-11 镜像详情

软件类型	版本详情
操作系统	CentOS 7.9 64bit
架构类型	x86

GPU Ant1 裸金属服务器支持的镜像详情

- 镜像名称：Ubuntu-20.04-x86-for-Ant1-BareMetal-with-RoCE-and-GP-515-CUDA-11.7-AIGC(仅限北京四和乌兰察布一)

  表 1-12 镜像详情

软件类型	版本详情
操作系统	Ubuntu 20.04 server 64bit
架构类型	x86
RoCE路由配置	不支持自动配置，需创建后手动配置。

- 镜像名称：Ubuntu-20.04-x86-for-Ant1-BareMetal-with-RoCE-and-NVIDIA-525-CUDA-12.0-AIGC(仅限乌兰察布一)

  表 1-13 镜像详情

软件类型	版本详情
操作系统	Ubuntu 20.04 server 64bit
架构类型	x86
RoCE路由配置	不支持自动配置，需创建后手动配置。

2 Lite Server 资源开通

图 2-1 Server 资源开通流程图

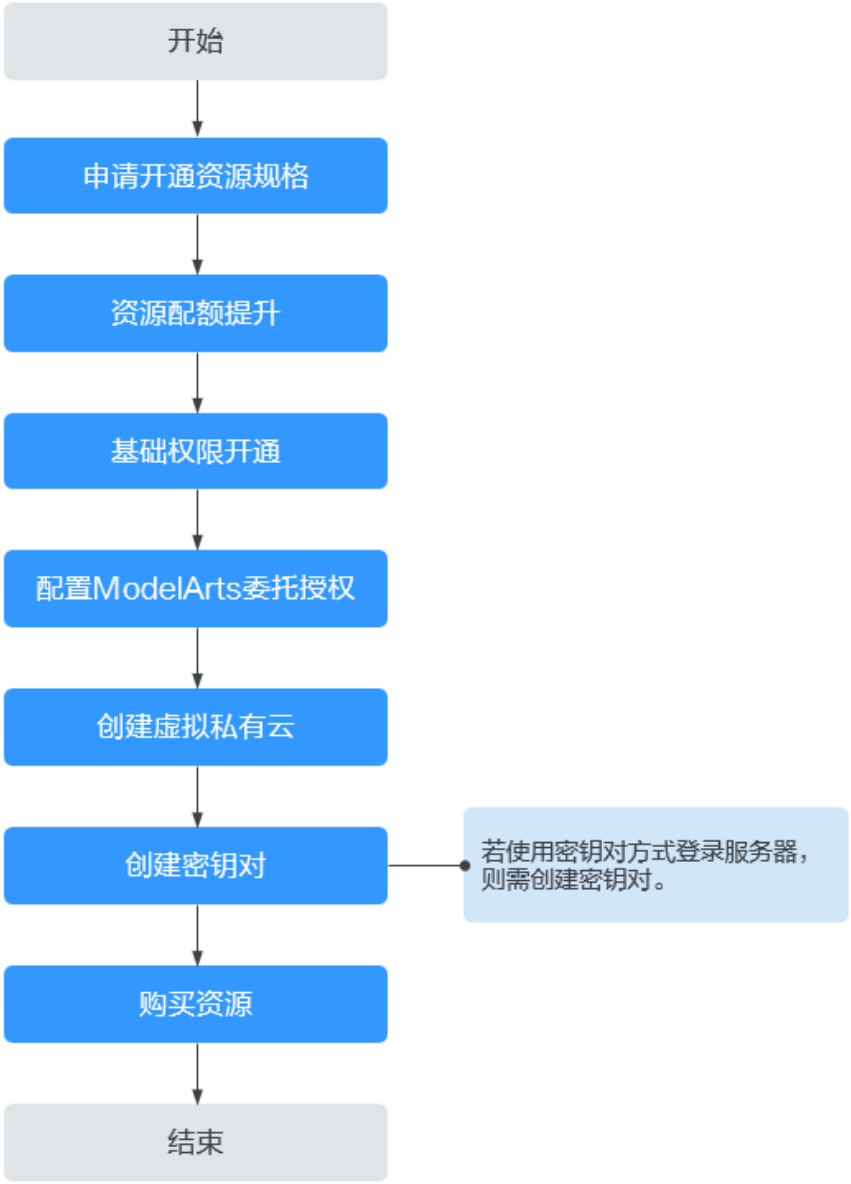


表 2-1 Server 资源开通流程

阶段	任务
准备工作	1、申请开通资源规格。
	2、资源配额提升。
	3、基础权限开通。
	4、配置ModelArts委托授权。
	5、创建虚拟私有云。

阶段	任务
	6、创建密钥对。（可选，若为密码登录方式则不需要）
购买Server资源	7、在ModelArts控制台上购买资源池。

步骤 1：申请开通资源规格

请联系华为云客户经理确认Server资源方案、申请要开通资源的规格（若无客户经理可提交工单）。

步骤 2：资源配额提升

由于Server所需资源可能会超出华为云默认提供的资源（如ECS、EIP、SFS、内存大小、CPU核数），因此需要提升资源配额。

步骤1 登录华为云管理控制台。

步骤2 在顶部导航栏单击“资源 > 我的配额”，进入服务配额页面。

步骤3 单击右上角“申请扩大配额”，填写申请材料后提交工单。

说明

配额需大于需要开通的资源，且在购买开通前完成提升，否则会导致资源开通失败。

----结束

步骤 3：基础权限开通

基础权限开通需要登录管理员账号，为子用户账号开通Server功能所需的基础权限（ModelArts FullAccess/BMS FullAccess/ECS FullAccess/VPC FullAccess/VPC Administrator/VPCEndpoint Administrator）。

步骤1 登录统一身份认证服务管理控制台。

步骤2 单击目录左侧“用户组”，然后在页面右上角单击“创建用户组”。

步骤3 填写“用户组名称”并单击“确定”。

步骤4 在操作列单击“用户组管理”，将需要配置权限的用户加入用户组中。

步骤5 单击用户组名称，进入用户组详情页。

步骤6 在权限管理页签下，单击“授权”。

图 2-2 “配置权限”

步骤7 在搜索栏输入“ModelArts FullAccess”，并勾选“ModelArts FullAccess”。

图 2-3 ModelArts FullAccess

以相同的方式，依次添加：BMS FullAccess、ECS FullAccess、VPC FullAccess、VPC Administrator、VPCEndpoint Administrator。（Server Administrator、DNS Administrator为依赖策略，会自动被勾选）。

步骤8 单击“下一步”，授权范围方案选择“所有资源”。

步骤9 单击“确认”，完成基础权限开通。

----结束

步骤 4：配置 ModelArts 委托授权

由于ModelArts服务在使用过程中会访问其他依赖服务，因此需要给ModelArts进行委托授权，详细操作参考**配置ModelArts委托**。

步骤 5：创建虚拟私有云

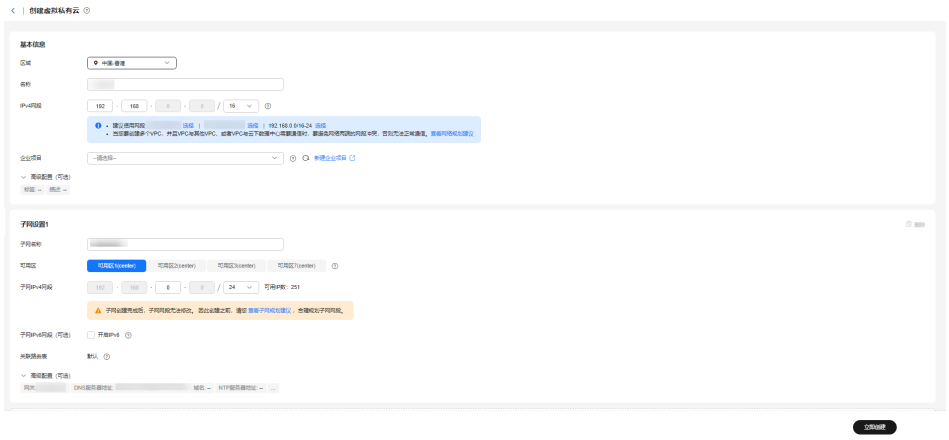
创建虚拟私有云需要登录管理员账号，IP地址段请根据现网情况合理规划。

步骤1 登录管理控制台。

步骤2 在左侧服务列表中，单击“网络 > 虚拟私有云 VPC”，进入虚拟私有云页面。

步骤3 单击右上角“创建虚拟私有云”后，根据界面提示配置虚拟私有云参数（参数介绍可参考**此处**），然后单击“立即创建”。

图 2-4 新建虚拟私有云



----结束

步骤 6：创建密钥对

说明

若使用密码方式登录裸金属服务器，则不需要创建密钥对。

步骤1 登录ModelArts管理控制台。

步骤2 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“弹性节点 Server”列表。

步骤3 单击“购买AI专属节点”，进入“购买AI专属节点”页面。

步骤4 单击“新建密钥对”。

步骤5 在新页面勾选“我已阅读并同意...”并单击“确定”，并将密钥对保存至本地。

----结束

步骤 7：购买资源

当前支持的裸金属镜像请见**Lite Server算力资源和镜像版本配套关系**，在创建Server实例时，根据所需镜像选择对应的规格。

步骤1 登录ModelArts管理控制台。

步骤2 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“弹性节点 Server”列表。

步骤3 单击“购买AI专属节点”，进入“购买AI专属节点”页面，在该页面填写相关参数信息。

表 2-2 基础配置参数说明

参数名称	说明
资源类型	• 裸金属服务器是一款兼具弹性云服务器和物理机性能的计算类服务器，为您和您的企业提供专属的云上物理服务器。 • 弹性云服务器是一种可随时自助获取、可弹性伸缩的云服务器，可帮助您打造可靠、安全、灵活、高效的应用环境，确保服务持久稳定运行，提升运维效率。
计费模式	选择“按需计费”或“包年/包月”模式。 • 包年/包月 包年/包月是预付费模式，按订单的购买周期计费，适用于可预估资源使用周期的场景，价格比按需计费模式更优惠。 • 按需计费 按需计费是后付费模式，按云服务器的实际使用时长计费，可以随时开通/删除云服务器。
区域	不同区域的云服务产品之间内网互不相通；请就近选择靠近您业务的区域，可减少网络时延，提高访问速度。资源购买完成后，您可在控制台左上角切换区域，查看对应的资源。

参数名称	说明
可用区	可用区是同一服务区内，电力和网络互相独立的地理区域，一般是一个独立的物理机房，这样可以保证可用区的独立性。是否将资源放在同一可用区内，主要取决于您对容灾能力和网络时延的要求。 ● 如果您的应用需要较高的容灾能力，建议您将资源部署在同一区域的不同可用区内。 ● 如果您的应用要求实例之间的网络延时较低，则建议您将资源创建在同一可用区内。 若您使用了CloudPond云服务，您可以在此处看到对应的边缘可用区。边缘可用区将云基础设施和云服务部署到企业现场，适合对应用访问时延、数据本地化留存及本地系统交互等有高要求的场景，可便捷地将云端丰富应用部署到本地，CloudPond介绍可参考**产品介绍**。

表 2-3 规格配置参数说明

参数名称	说明
服务器名称	Server的名称。只能包含数字、大小写字母、下划线和中划线，长度不能超过64位且不能为空。
CPU架构	资源类型的CPU架构，支持X86和ARM。请根据所需规格选择CPU架构，若使用GPU选择X86，若使用NPU则选择ARM。具体规格可有区域差异，以最终显示为准。 **说明** 如果界面无可选规格，请**联系华为云技术支持**申请开通。
系统盘	系统盘和规格有关，选择支持挂载的规格才会显示此参数。可以在创建完成后在云服务器侧实现数据盘挂载或系统盘的扩容，建议取值至少100GB。

表 2-4 镜像说明

参数名称	说明
镜像	● 公共镜像 常见的标准操作系统镜像，所有用户可见，包括操作系统以及预装的公共应用（SDI卡驱动、bms-network-config网络配置程序、Cloud-init初始化工具等）。请根据您的实际需要自助配置应用环境或相关软件。ModelArts服务提供镜像支持多种操作系统，内置AI场景相关驱动和软件，预置ModelArts自定义OS优化组件，当前支持的镜像请参考**Lite Server算力资源和镜像版本配套关系**。 ● 私有镜像 用户基于外部镜像文件或裸金属服务器创建的个人镜像，仅用户自己可见。包含操作系统、预装的公共应用以及用户的私有应用。选择私有镜像创建，可以节省您重复配置服务器的时间。

表 2-5 选择网络参数说明

参数名称	说明
虚拟私有云	配置Server的虚拟私有云（Virtual Private Cloud，简称VPC），建议选择VPC时与其它云服务(如MRS、CCE等云服务) 保持一致，便于网络互通。
子网	选择该VPC下的一个子网。
IPv6网络	若当前网络配置的子网、规格、镜像都支持IPv6，则会显示该参数，打开后可启用IPv6功能。 请确保您的子网已开启IPv6功能，若未开启请参考**为虚拟私有云创建新的子网**。 不同规格、镜像对IPv6支持的情况不同，若不支持则不会显示IPv6网络参数，请以控制台实际显示为准。
RoCE网络	当前使用A系列GPU时，进行分布式训练为了将硬件上的RoCE网卡使用起来，需要配置RoCE网络。 该参数与所选规格有关，若未选中规格或规格不支持RoCE网络，则不显示。 若规格支持RoCE网络但未创建过，单击“新建RoCE网络”即可完成创建。 若规格支持RoCE网络且已创建过RoCE网络，直接选择已有RoCE网络即可（不支持重复创建）。
安全组	安全组是一个逻辑上的分组，为同一个VPC内具有相同安全保护需求并相互信任的Server提供访问策略。

表 2-6 管理参数说明

参数名称	说明
登录凭证	“密钥对”方式创建的裸金属服务器安全性更高，建议选择“密钥对”方式。如果您习惯使用“密码”方式，请增强密码的复杂度，保证密码符合要求，防止被恶意攻击。 ● 密钥对 指使用密钥对作为登录裸金属服务器的鉴权方式。您可以选择使用已有的密钥对，或者单击“新建密钥对”创建新的密钥。 **说明** 如果选择使用已有的密钥，请确保您已在本地获取该文件，否则，将影响您正常登录裸金属服务器。 ● 密码 指使用设置初始密码方式作为裸金属服务器的鉴权方式，此时，您可以通过用户名密码方式登录裸金属服务器。 Linux操作系统时为root用户的初始密码，Windows操作系统时为Administrator用户的初始密码。密码复杂度需满足以下要求： – 长度为8至26个。 – 至少包含大写字母、小写字母、数字及特殊符号(!@$%^-_=+[{}]:,./?)中的3种 – 不能与用户名或倒序的用户名相同。 – 不能包含root或administrator及其逆序。

表 2-7 高级配置参数说明

参数名称	说明
企业项目	该参数针对企业用户使用，只有开通了企业项目的客户，或者权限为企业主账号的客户才可见。如需使用该功能，请联系您的客户经理申请开通。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理，默认项目为default。 请从下拉列表中选择所在的企业项目。更多关于企业项目的信息，请参见《企业管理用户指南》。

表 2-8 购买配置参数说明

参数名称	说明
购买数量	支持同时购买多台机器，输入值必须在1到10之间。 若有多台机器资源，会生成对应多笔订单，需逐一支付每笔订单，不可合并支付。

步骤4 单击“立即创建”，完成实例的创建，随后进入付款界面。

步骤5 支付对应资源的订单。

说明

若有多台机器资源，会生成对应多笔订单，需逐一支付每笔订单，不可合并支付。

步骤6 支付完成后，由于Server资源创建约20~60分钟，请耐心等待资源创建成功。

说明

- 若ModelArts弹性节点Server创建失败，可能由多种原因导致，以下给出了几种类型的可能原因进行快速排查和定位解决。
  - 资源不足：跳转到BMS页面，查看要购买的规格是否售罄，如果该规格售罄，说明无该规格资源，需要联系客户经理获取到资源后再进行购买。
  - 配额不足：查看账户的资源配额是否满足，若该账号下资源配额，包括核心数、RAM等，如果未满足也会导致创建失败，需要申请配额后再进行购买。
  - BMS机器内部错误：查看BMS界面，创建失败出现内部错误，该问题需要提工单给BMS进行进一步定位失败原因并解决。
- 当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

----结束

3 Lite Server 资源配置

3.1 Lite Server 资源配置流程

在开通Lite Server资源后，需要完成相关配置才能使用，配置流程如下图所示。

图 3-1 Lite Server 资源配置流程图

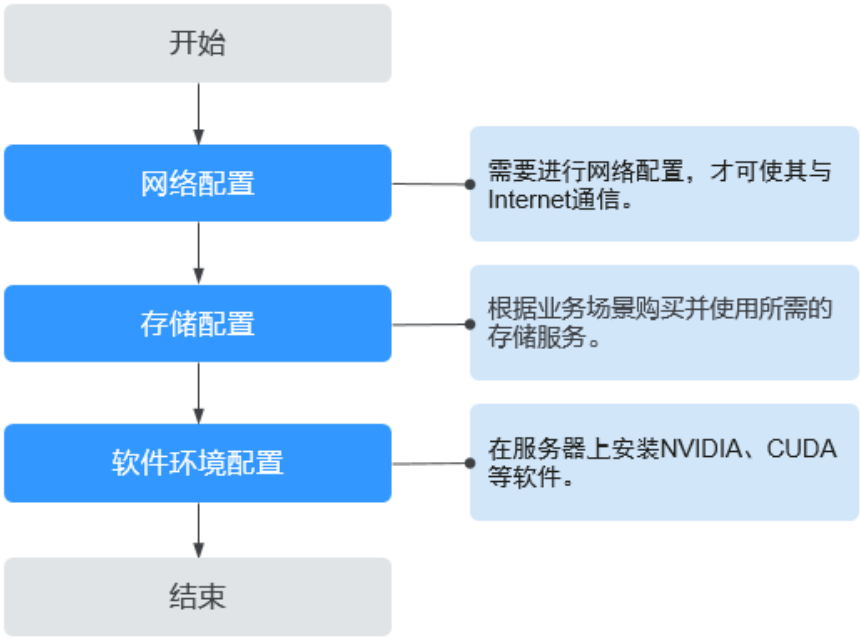


表 3-1 Server 资源配置流程

配置顺序	配置任务	场景说明
1	**配置Lite Server网络**	Server资源开通后，需要进行网络配置，才可使其与Internet通信。在后续配置存储和软件环境时需要Server服务器能够访问网络，因此需要先完成网络配置。
2	**配置Lite Server存储**	Server资源需要挂载数据盘用于存储数据文件，当前支持SFS、OBS、EVS三种云存储服务，提供了多种场景下的存储解决方案。
3	**配置Lite Server软件环境**	不同镜像中预安装的软件不同，您通过**Lite Server算力资源和镜像版本配套关系**章节查看已安装的软件。当Server服务器中预装的软件无法满足业务需求时，您可在Server服务器中配置所需要的软件环境。

3.2 配置 Lite Server 网络

Server创建后，需要进行网络配置，才可使其与Internet通信，本章节介绍网络配置步骤。网络配置主要分为以下两个场景：

- **单个弹性公网IP用于单个Server服务器**：为单台Server服务器绑定一个弹性公网IP，该Server服务器独享网络资源。
- **单个弹性公网IP用于多个Server服务器**：一个VPC配置一个EIP（弹性公网IP），通过NAT网关配置进行EIP资源共享，实现该VPC下的所有Server服务器均可以通过该EIP进行公网访问，Server服务器共享网络资源。

单个弹性公网 IP 用于单个 Server 服务器

步骤1 登录ModelArts管理控制台。

步骤2 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“弹性节点 Server”列表页面。

步骤3 单击Server服务器名称，进入Server服务器详情页面，单击裸金属服务器名称，跳转至裸金属服务器详情页。

图 3-2 裸金属服务器

步骤4 单击“弹性公网IP”页签，然后单击“绑定弹性公网IP”。

弹出“绑定弹性公网IP”对话框。

选择要绑定的弹性公网IP，单击“确定”，完成绑定。

图 3-3 绑定弹性公网 IP

说明

一个网卡只能绑定一个弹性公网IP。

----结束

单个弹性公网 IP 用于多个 Server 服务器

说明

所有Server资源必须位于同一个VPC，并且该VPC没有NAT网关以及默认路由。

步骤1 购买弹性公网IP。

1. 登录华为云管理控制台。
2. 在左侧服务列表中，单击“网络 > 弹性公网IP EIP”，进入弹性公网IP页面。
3. 单击“购买弹性公网IP”。
4. 参数配置可使用默认值，单击“立即购买”。
5. 在产品配置信息确认页面，再次核对弹性公网IP信息，阅读并勾选“弹性公网IP服务声明”。
   - 选择按需计费的弹性公网IP时，单击“提交”。
   - 选择包年/包月计费的弹性公网IP时，单击“去支付”。
     进入订单支付页面，确认订单信息，单击“确认付款”。

步骤2 购买公网NAT网关。

1. 登录华为云管理控制台。
2. 在左侧服务列表中，单击“网络 > NAT网关 NAT”，进入公网NAT网关页面。
3. 单击“购买公网NAT网关”。
4. 选择Server所使用“虚拟私有云”和“子网”，计费模式根据实际需求选择。其余参数配置可使用默认值，单击“立即购买”。
5. 在产品配置信息确认页面，再次核对弹性公网IP信息。
   - 选择按需计费的NAT网关时，单击“提交”。
   - 选择包年/包月计费的NAT网关时，单击“去支付”。
     进入订单支付页面，确认订单信息，单击“确认付款”。

   说明

   虚拟私有云和子网和Server资源的网络保持一致。

步骤3 配置SNAT规则。

SNAT功能通过绑定弹性公网IP，实现私有IP向公有IP的转换，可实现VPC内跨可用区的多个云主机共享弹性公网IP、安全高效地访问互联网。

1. 公网NAT网关页面，单击创建的NAT网关名称，进入NAT网关详情页。
2. 在SNAT规则页签下，单击“添加SNAT规则”。
3. 在弹出的“添加SNAT规则页面”，配置SNAT规则：
   - 使用场景：选择“虚拟私有云”。
   - 子网：选择“使用已有”，选择子网。
   - 弹性公网IP：勾选创建的弹性公网IP。
4. 单击“确定”。

步骤4 配置DNAT规则。

通过添加DNAT规则，则可以通过映射方式为VPC内的Server提供SSH访问服务，一个Server的一个端口对应一条DNAT规则，一个端口只能映射到一个EIP，不能映射到多个EIP。

1. 在DNAT规则页签下，单击“添加DNAT规则”。
2. 在弹出的“添加DNAT规则页面”，配置DNAT规则：
   - 使用场景：选择“虚拟私有云”。

- 端口类型：选择“具体端口”。
- 支持协议：选择“TCP”。
- 公网IP类型：选择已创建的弹性公网IP。
- 公网端口：建议选择区间为20000-30000，保证该端口号不冲突。
- 实例类型：单击“服务器”，选择Server服务器。
- 网卡：选择服务器网卡。。
- 私网端口：端口号22。

3. 单击“确定”。

----结束

3.3 配置 Lite Server 存储

Server服务器支持SFS、OBS、EVS三种云存储服务，提供了多种场景下的存储解决方案，主要区别如下表所示。若需要对本地盘进行配置，请参考**物理机环境配置**。

表 3-2 表 1 SFS、OBS、EVS 服务对比

对比维度	弹性文件服务SFS	对象存储服务OBS	云硬盘EVS
概念	提供按需扩展的高性能文件存储，可为云上多个云服务器提供共享访问。弹性文件服务就类似Windows或Linux中的远程目录。	提供海量、安全、高可靠、低成本的数据存储能力，可供用户存储任意类型和大小的数据。	可以为云服务器提供高可靠、高性能、规格丰富并且可弹性扩展的块存储服务，可满足不同场景的业务需求。云硬盘就类似PC中的硬盘。
存储数据的逻辑	存放的是文件，会以文件和文件夹的层次结构来整理和呈现数据。	存放的是对象，可以直接存放文件，文件会自动产生对应的系统元数据，用户也可以自定义文件的元数据。	存放的是二进制数据，无法直接存放文件，如果需要存放文件，需要先格式化文件系统后使用。
访问方式	在BMS中通过网络协议挂载使用，支持NFS和CIFS的网络协议。需要指定网络地址进行访问，也可以将网络地址映射为本地目录后进行访问。	可以通过互联网或专线访问。需要指定桶地址进行访问，使用的是HTTP和HTTPS等传输协议。	只能在BMS中挂载使用，不能被操作系统应用直接访问，需要格式化成文件系统进行访问。

对比维度	弹性文件服务SFS	对象存储服务OBS	云硬盘EVS
使用场景	如高性能计算、媒体处理、文件共享和内容管理和Web服务等。 **说明** 高性能计算：主要是高带宽的需求，用于共享文件存储，比如基因测序、图片渲染这些。	如大数据分析、静态网站托管、在线视频点播、基因测序和智能视频监控等。	如高性能计算、企业核心集群应用、企业应用系统和开发测试等。 **说明** 高性能计算：主要是高速率、高IOPS的需求，用于作为高性能存储，比如工业设计、能源勘探这些。
容量	PB级别	EB级别	TB级别
时延	3~10ms	10ms	亚毫秒级
IOPS/TPS	单文件系统 10K	千万级	单盘 128K
带宽	GB/s级别	TB/s级别	MB/s级别
是否支持数据共享	是	是	是
是否支持远程访问	是	是	否
是否支持在线编辑	是	否	是
是否能单独使用	是	是	否（EVS要搭配BMS才能存储文件）

使用弹性文件服务 SFS 作为存储

若使用SFS服务作为存储方案，推荐使用SFS Turbo文件系统。SFS Turbo提供按需扩展的高性能文件存储，还具备高可靠和高可用的特点，支持根据业务需要弹性扩容，且性能随容量增加而提升，可广泛应用于多种业务场景。

步骤1 在SFS服务控制台上创建文件系统，具体步骤请参考**创建SFS Turbo文件系统**。同一区域不同可用区之间文件系统与云服务器互通，因此保证SFS Turbo与Server服务器在同一区域即可。

步骤2 当创建文件系统后，您需要使用弹性裸金属服务器来挂载该文件系统，具体步骤请参考**挂载NFS协议类型文件系统到云服务器（Linux）**。

步骤3 为避免已挂载文件系统的云服务器重启后，挂载信息丢失，您可以在云服务器设置重启时进行自动挂载，具体步骤请参考**服务器重启后自动挂载指南**。

----结束

使用对象存储服务 OBS 作为存储

若使用OBS服务作为存储方案，推荐使用“并行文件系统+obsutil”的方式，并行文件系统是OBS服务提供的一种经过优化的高性能文件语义系统，提供毫秒级别访问时延，TB/s级别带宽和百万级别的IOPS。obsutil是一款用于访问管理华为云对象存储服

务（Object Storage Service，OBS）的命令行工具，您可以使用该工具对OBS进行常用的配置管理操作，如创建桶、上传文件/文件夹、下载文件/文件夹、删除文件/文件夹等。对于熟悉命令行程序的用户，obsutil能在执行批量处理、自动化任务场景能为您带来更优体验。

步骤1 在OBS服务控制台上创建并行文件系统，具体步骤请参考**创建并行文件系统**。

步骤2 针对您的操作系统，下载对应版本的obsutil至弹性裸金属服务器，并完成安装，具体步骤请参考**下载和安装obsutil**。

步骤3 使用obsutil之前，您需要配置obsutil与OBS的对接信息，包括OBS终端节点地址（Endpoint）和访问密钥（AK和SK）。获得OBS的认证后，才能使用obsutil执行OBS桶和对象的相关操作，具体步骤请参考**初始化配置**。

步骤4 配置完成后，您可以通过命令行的方式在弹性裸金属服务器中对OBS的文件进行上传下载等操作，关于命令行介绍请参考**命令行结构**。

----结束

使用云硬盘 EVS 作为存储

步骤1 在EVS服务控制台上购买磁盘，选择裸金属服务器所在的可用区，挂载方式选择“暂不挂载”，计费模式选择“包年/包月”或者“按需计费”均可以，磁盘大小根据自身需求进行选择购买，更多EVS购买参数介绍可参考**购买云硬盘**。

图 3-4 购买磁盘

说明

由于产品特性设计，暂不支持在购买EVS云硬盘时立即挂载到云服务器，此时网页界面会提示“该包年/包月云服务器还未同步到运营系统，请休息片刻再重试。您可以到费用中心> 续费管理页面确认该云服务器是否已同步到运营系统”，挂载方式选择暂不挂载即可。

步骤2 在完成EVS数据盘购买后，进入Server对应的裸金属服务器详情界面，单击“挂载磁盘”，选择刚才购买的EVS数据盘进行挂载即可。

图 3-5 挂载磁盘

说明

在退订裸金属服务时，挂载的EVS数据盘不会自动删除。用户可根据自身需求，将其挂载在其他裸金属服务器上或者进行手动删除。

----结束

3.4 配置 Lite Server 软件环境

3.4.1 NPU 服务器上配置 Lite Server 资源软件环境

场景描述

本文旨在指导如何在Snt9b裸金属服务器上，进行磁盘合并挂载、安装docker等环境配置。在配置前请注意如下事项：

- 首次装机时需要配置存储、固件、驱动、网络访问等基础内容，这部分配置尽量稳定减少变化。
- 裸机上的开发形式建议开发者启动独立的Docker容器作为个人开发环境。Snt9b的裸机包含8卡算力资源，一般来说多人可以共用这个裸机完成开发与调测工作。多人使用为了避免冲突，建议各自在自己的docker容器中进行独立开发，并提前规划好每个人使用的具体卡号，避免相互影响。
- ModelArts提供了标准化基础容器镜像，在容器镜像中已经预置了基础MindSpore或PyTorch框架和开发调测工具链，推荐用户直接使用该镜像，用户也可以使用自己的业务镜像或昇腾AscendHub提供的镜像。如果镜像中预置的软件版本不是您期望的版本，可以自行安装替换。
- 开发形式推荐通过容器中暴露的SSH端口以远程开发的模式(VSCode SSH Remote、 Xshell)连接到容器中进行开发，可以在容器中挂载宿主机的个人存储目录，用于存放代码和数据。

说明

当前指导中很多操作步骤在最新发放的Snt9b裸机环境中已经预置，无需用户再手动配置，用户在操作中如发现某个步骤已有预置配置可直接跳过该步骤。

物理机环境配置

步骤1 配置超时参数。

SSH登录到Server服务器后，查看机器配置的超时参数。

```
echo $TMOUT
```

如果该值为300，则代表默认空闲等待5分钟后会断开连接，可以增大该参数延长空闲等待时间（若值已经为0可跳过该步骤）。修改方法如下：

```
vim /etc/profile
# 在文件最后修改TMOUT值，由300改为0，0表示不会空闲断开
export TMOUT=0
```

执行命令使其在当前terminal生效。

```
TMOUT=0
```

步骤2 磁盘合并挂载。

成功购买裸金属服务器后，服务器上可能会有多个未挂载的nvme磁盘。因此在首次配置环境前，需要完成磁盘合并挂载。此操作需要放在最开始完成，避免使用一段时间后再挂载会冲掉用户已存储的内容。

1. 首先通过“lsblk”查看是否有3个7T的磁盘未挂载，如下图所示nvme0n1、nvme1n1、nvme2n1为未挂载。

图 3-6 磁盘未挂载

```
[root@devserver-7354 ~]# lsblk
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda       8:0    0  150G  0 disk
├─sda1    8:1    0    1G  0 part /boot/efi
└─sda2    8:2    0  149G  0 part /
nvme0n1 259:0    0    7T  0 disk
nvme1n1 259:1    0    7T  0 disk
nvme2n1 259:2    0    7T  0 disk
[root@devserver-7354 ~]#
```

2. 如下图所示，每个盘后已有MOUNTPOINT，则代表已经执行过挂载操作，可跳过此章节，只用直接在/home目录下创建自己的个人开发目录即可。

图 3-7 磁盘已挂载

```
[root@devserver-7354 ~]# lsblk
NAME                      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                         8:0    0  150G  0 disk
├─sda1                      8:1    0    1G  0 part /boot/efi
└─sda2                      8:2    0  149G  0 part /
nvme0n1                   259:0    0    7T  0 disk /home
nvme1n1                   259:1    0    7T  0 disk
└─nvme_group-docker_data  253:0    0   14T  0 lvm  /docker
nvme2n1                   259:2    0    7T  0 disk
└─nvme_group-docker_data  253:0    0   14T  0 lvm  /docker
```

执行自动化挂载脚本，将“/dev/nvme0n1”挂载在“/home”下供每个开发者创建自己的家目录，另两个合并挂载到“/docker”下供容器使用（如果不单独给“/docker”分配较大空间，当多人共用创建多个容器实例时容易将根目录占满）。

```
cd /root/tools/
sh create_disk_partitions.sh
```

配置完成后，执行“df -h”可以看到新挂载的磁盘信息。

图 3-8 查看新挂载的磁盘

```
[root@devserver-modelarts home]# df -h
Filesystem                          Size  Used Avail Use% Mounted on
devtmpfs                            756G     0  756G   0% /dev
tmpfs                               756G     0  756G   0% /dev/shm
tmpfs                               756G   28M  756G   1% /run
tmpfs                               756G     0  756G   0% /sys/fs/cgroup
/dev/sda2                           196G  2.4G  185G   2% /
tmpfs                               756G   40K  756G   1% /tmp
/dev/sda1                          1022M  8.3M 1014M   1% /boot/efi
/dev/mapper/nvme_group-docker_data   14T  121G   14T   1% /docker
/dev/nvme0n1                        7.0T   50G  7.0T   1% /home
```

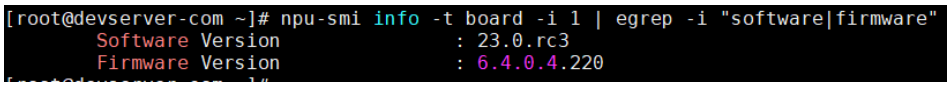
磁盘合并挂载后，即可在“/home”下创建自己的工作目录，以自己的名字命名。

步骤3 （可选）安装固件和驱动。

1. 查看环境信息。执行如下命令查看当前拿到的机器的固件和驱动版本。

```
npu-smi info -t board -i 1 | egrep -i "software|firmware"
```

图 3-9 查看固件和驱动版本



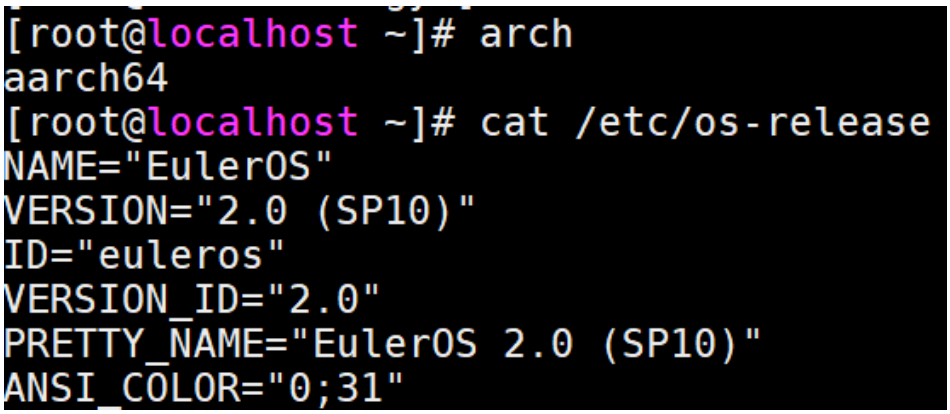
其中firmware代表固件版本，software代表驱动版本，当前昇腾商用发布的最新版本为上图所示的版本，可以不用执行本章节后续的固件驱动安装步骤。

如果机器上的版本不是所需的版本（例如需要换成社区最新调测版本），可以参考后续步骤进行操作。

2. 查看机器操作系统版本，以及架构是aarch64还是x86_64，并从昇腾官网获取相关的固件驱动包。固件包名称为“Ascend-hdk-型号-npu-firmware_版本号.run”，驱动包名称为“Ascend-hdk-型号-npu-driver_版本号_linux-aarch64.run”，商用版是权限受控，仅华为工程师和渠道用户有权限下载，下载地址请见**固件驱动包下载链接**。

```
arch
cat /etc/os-release
```

图 3-10 查看机器操作系统版本及架构



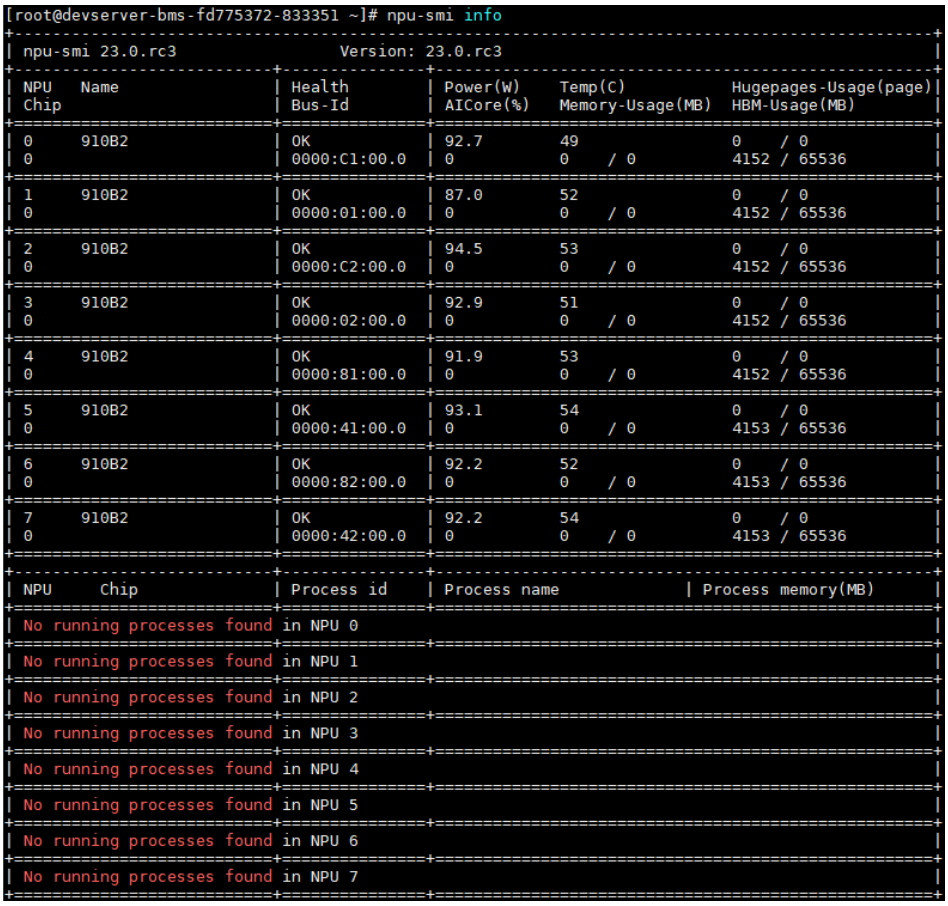
下文均以适配EulerOS 2.0（SP10）和aarch64架构的包为例来进行讲解。

3. 安装固件和驱动包。

   a. 首先检查npu-smi工具是否可以正常使用，该工具必须能正常使用才能继续后面的固件驱动安装，输入命令“npu-smi info”，完整输出下图内容则为正常。

      如果命令未按照下图完整输出（比如命令报错或只输出了上半部分没有展示下面的进程信息），则需要先尝试恢复npu-smi工具（提交工单**联系华为云技术支持**），将npu-smi恢复后，再进行新版本的固件驱动安装。

图 3-11 检查 npu-smi 工具



b. 工具检查正常后，进行固件和驱动安装。

说明

1. 固件和驱动安装时，注意安装顺序：
   1. 首次安装场景：硬件设备刚出厂时未安装驱动，或者硬件设备前期安装过驱动固件但是当前已卸载，上述场景属于首次安装场景，需按照“驱动->固件”的顺序安装驱动固件。
   2. 覆盖安装场景：硬件设备前期安装过驱动固件且未卸载，当前要再次安装驱动固件，此场景属于覆盖安装场景，需按照“固件->驱动”的顺序安装固件驱动。

通常Snt9b出厂机器有预装固件驱动，因此本案例中是“覆盖安装场景”，注意：

i. 如果新装的固件驱动比环境上已有的版本低，只要npu-smi工具可用，也是直接装新软件包即可，不用先卸载环境上已有的版本。

ii. 如果固件驱动安装失败，可先根据报错信息在开发者社区搜索解决方案。

安装命令如下：

i. 安装固件，安装完后需要reboot重启机器。

```
chmod 700 *.run
#  注意替换成实际的包名
./Ascend-hdk-型号-npu-firmware_版本号.run  --full
reboot
```

ii. 安装驱动，提示处输入“y”，安装完后直接生效不用重启机器。

```
# 注意替换成实际的包名
./Ascend-hdk-型号-npu-driver_版本号_linux-aarch64.run --full --install-for-all
```

iii. 安装完成后，执行下述命令检查固件和驱动版本，正常输出代表安装成功。

```
npu-smi info -t board -i 1 | egrep -i "software|firmware"
```

图 3-12 检查固件和驱动版本

```
[root@devserver-com ~]# npu-smi info -t board -i 1 | egrep -i "software|firmware"
        Software Version                : 23.0.rc3
        Firmware Version                : 6.4.0.4.220
```

步骤4 安装docker环境。

1. 先执行“docker -v”检查机器是否已安装docker，若已安装，则可跳过此步骤。

   安装docker命令如下。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

   使用docker -v检查是否安装成功：

   图 3-13 查看 docker 版本

   ```
   [root@localhost ~]# docker -v
   Docker version 18.09.0, build ba6df24
   ```

2. 配置IP转发，用于容器内的网络访问。

   执行下述命令查看net.ipv4.ip_forward配置项值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果不为1，执行下述命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

3. 查看环境是否已安装并配置Ascend-docker-runtime。

   ```
   docker info |grep Runtime
   ```

   如果输出的runtime为“ascend”，则代表已安装配置好，可跳过此步骤。

   图 3-14 Ascend-docker-runtime 查询

   ```
   [root@devserver-modelarts-demanager-0eaabe8f ~]# docker info |grep Runtime
   Runtimes: ascend runc
   Default Runtime: ascend
   ```

   若未安装，则单击链接下载社区版**Ascend Docker Runtime**，该软件包是昇腾提供的docker插件，在docker run时可以自动挂载昇腾driver等路径到容器，无需在启动容器时手工指定--device参数。下载好后将包上传到服务器并进行安装。

   ```
   chmod 700 *.run
   ./Ascend-hdk-型号-npu-driver_版本号_linux-aarch64.run --install
   ```

   关于Ascend Docker Runtime的更多使用指导，请参考**Ascend Docker Runtime用户指南**。

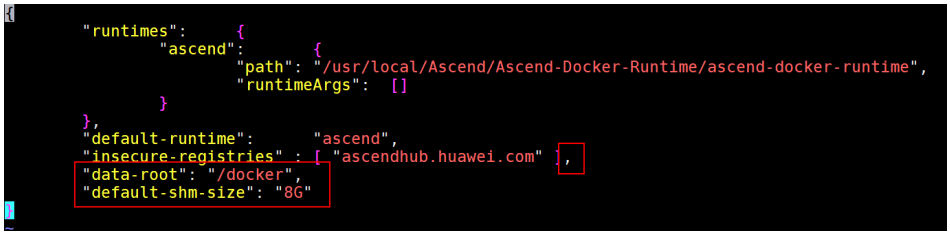
4. 将新挂载的盘设置为docker容器使用路径。

   编辑“/etc/docker/daemon.json”文件内容，如果文件不存在则新建即可。

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

   增加如下两项配置，注意insecure-registries行末尾增加一个逗号，保持json格式正确。其中“data_root”代表docker数据存储路径，“default-shm-size”代表容器启动默认分配的共享内容大小，不配置时默认为64M，可以根据需要改大，避免分布式训练时共享内存不足导致训练失败。

图 3-15 docker 配置



保存后，执行如下命令重启docker使配置生效。

```
systemctl daemon-reload && systemctl restart docker
```

步骤5 （可选）安装pip。

1. 执行如下命令检查是否已安装pip且pip源正常访问，如果能正常执行，可跳过此章节。

   ```
   pip install numpy
   ```

2. 若物理机上没有安装pip，可执行如下命令安装。

   ```
   python -m ensurepip --upgrade
   ln -s /usr/bin/pip3 /usr/bin/pip
   ```

3. 配置pip源。

   ```
   mkdir -p ~/.pip
   vim ~/.pip/pip.conf
   ```

   在“~/.pip/pip.conf”中写入如下内容。

   ```
   [global]
   index-url = http://mirrors.myhuaweicloud.com/pypi/web/simple
   format = columns
   [install]
   trusted-host=mirrors.myhuaweicloud.com
   ```

步骤6 RoCE网络测试。

1. 安装cann-toolkit。

   查看服务器是否已安装CANN Toolkit，如果显示有版本号则已安装。

   ```
   cat /usr/local/Ascend/ascend-toolkit/latest/aarch64-linux/ascend_toolkit_install.info
   ```

   如果未安装，则需要从官网下载相关软件包，其中社区版可以直接下载（**下载地址**），商用版是权限受控，仅华为工程师和渠道用户有权限下载（**下载链接**）。

   安装CANN Toolkit，注意替换包名。

   ```
   chmod 700 *.run
   ./Ascend-cann-toolkit_6.3.RC2_linux-aarch64.run --full --install-for-all
   ```

2. 安装mpich-3.2.1.tar.gz。

   单击**此处**下载，并执行以下命令安装。

   ```
   mkdir -p /home/mpich
   mv /root/mpich-3.2.1.tar.gz /home/
   cd /home/;tar -zxvf mpich-3.2.1.tar.gz
   cd /home/mpich-3.2.1
   ./configure --prefix=/home/mpich --disable-fortran
   make && make install
   ```

3. 设置环境变量和编译hccl算子。

   ```
   export PATH=/home/mpich/bin:$PATH
   cd /usr/local/Ascend/ascend-toolkit/latest/tools/hccl_test
   export LD_LIBRARY_PATH=/home/mpich/lib/:/usr/local/Ascend/ascend-toolkit/latest/
   lib64:$LD_LIBRARY_PATH
   make MPI_HOME=/home/mpich ASCEND_DIR=/usr/local/Ascend/ascend-toolkit/latest
   ```

   算子编译完成后显示内容如下：

图 3-16 算子编译完成

```
[root@devserver-com hccl_test]# make MPI_HOME=/home/mpich ASCEND_DIR=/usr/local
/Ascend/ascend-toolkit/latest
g++ -std=c++11 -Werror -fstack-protector-strong -fPIE -pie -O2 -s -Wl,-z,relro
-Wl,-z,now -Wl,-z,noexecstack -Wl,--copy-dt-needed-entries ./common/src/hccl_ch
eck_buf_init.cc ./common/src/hccl_check_common.cc ./common/src/hccl_opbase_root
info_base.cc ./common/src/hccl_test_common.cc ./common/src/hccl_test_main.cc ./
opbase_test/hccl_allgather_rootinfo_test.cc -I./common/src -I/usr/local/Ascend/
ascend-toolkit/latest/include -I/usr/local/Ascend/ascend-toolkit/latest/include
 -I/home/mpich/include -I./opbase_test -o all_gather_test -L/usr/local/Ascend/a
scend-toolkit/latest/lib64 -lhccl -L/usr/local/Ascend/ascend-toolkit/latest/lib
64 -lascendcl -L/home/mpich/lib -lmpi
all_gather_test compile completed
g++ -std=c++11 -Werror -fstack-protector-strong -fPIE -pie -O2 -s -Wl,-z,relro
-Wl,-z,now -Wl,-z,noexecstack -Wl,--copy-dt-needed-entries ./common/src/hccl_ch
eck_buf_init.cc ./common/src/hccl_check_common.cc ./common/src/hccl_opbase_root
info_base.cc ./common/src/hccl_test_common.cc ./common/src/hccl_test_main.cc ./
opbase_test/hccl_allreduce_rootinfo_test.cc -I./common/src -I/usr/local/Ascend/
ascend-toolkit/latest/include -I/usr/local/Ascend/ascend-toolkit/latest/include
 -I/home/mpich/include -I./opbase_test -o all_reduce_test -L/usr/local/Ascend/a
scend-toolkit/latest/lib64 -lhccl -L/usr/local/Ascend/ascend-toolkit/latest/lib
64 -lascendcl -L/home/mpich/lib -lmpi
all_reduce_test compile completed
```

4. 单机场景下进行all_reduce_test。

   进入hccl_test目录。

   ```
   cd /usr/local/Ascend/ascend-toolkit/latest/tools/hccl_test
   ```

   若是单机单卡，则执行下述命令。

   ```
   mpirun -n 1 ./bin/all_reduce_test -b 8 -e 1024M -f 2 -p 8
   ```

   若是单机多卡，则执行下述命令。

   ```
   mpirun -n 8 ./bin/all_reduce_test -b 8 -e 1024M -f 2 -p 8
   ```

图 3-17 all_reduce_test

```
[root@devserver-com hccl_test]# mpirun -n 8 ./bin/all_reduce_test -b 8 -e 1024M
the minbytes is 8, maxbytes is 1073741824, iters is 20, warmup_iters is 5
data_size(Bytes): | aveg_time(us): | alg_bandwidth(GB/s): | check_result:
8                 | 1323.66       | 0.00001              | success
16                | 1537.41       | 0.00001              | success
32                | 1567.12       | 0.00002              | success
64                | 1530.88       | 0.00004              | success
128               | 1567.90       | 0.00008              | success
256               | 1544.79       | 0.00017              | success
512               | 1534.98       | 0.00033              | success
1024              | 1771.28       | 0.00058              | success
2048              | 1457.74       | 0.00140              | success
4096              | 1619.05       | 0.00253              | success
8192              | 1570.33       | 0.00522              | success
16384             | 1575.37       | 0.01040              | success
32768             | 1542.54       | 0.02124              | success
65536             | 1568.91       | 0.04177              | success
131072            | 1554.22       | 0.08433              | success
262144            | 1552.85       | 0.16881              | success
524288            | 1573.59       | 0.33318              | success
1048576           | 1540.16       | 0.68082              | success
2097152           | 1544.21       | 1.35807              | success
4194304           | 1555.34       | 2.69671              | success
8388608           | 1558.78       | 5.38153              | success
16777216          | 1556.50       | 10.77880             | success
33554432          | 1425.38       | 23.54074             | success
67108864          | 1349.46       | 49.72998             | success
134217728         | 2460.50       | 54.54894             | success
268435456         | 4623.78       | 58.05536             | success
536870912         | 9194.49       | 58.39050             | success
1073741824        | 18450.20      | 58.19677             | success
```

5. 多机ROCE网卡带宽测试。

   a. 执行以下命令查看昇腾的RoCE IP。

```
cat /etc/hccn.conf
```

图 3-18 查看昇腾的 RoCE IP

```
[root@devserver-com hccl_test]# cat /etc/hccn.conf
address_0=29.89.132.13
netmask_0=255.255.0.0
netdetect_0=29.89.0.1
gateway_0=29.89.0.1
send_arp_status_0=1
address_1=29.89.20.64
netmask_1=255.255.0.0
netdetect_1=29.89.0.1
gateway_1=29.89.0.1
send_arp_status_1=1
address_2=29.89.155.174
netmask_2=255.255.0.0
netdetect_2=29.89.0.1
gateway_2=29.89.0.1
send_arp_status_2=1
address_3=29.89.148.38
netmask_3=255.255.0.0
netdetect_3=29.89.0.1
gateway_3=29.89.0.1
send_arp_status_3=1
address_4=29.89.134.236
netmask_4=255.255.0.0
netdetect_4=29.89.0.1
gateway_4=29.89.0.1
send_arp_status_4=1
address_5=29.89.133.119
netmask_5=255.255.0.0
netdetect_5=29.89.0.1
gateway_5=29.89.0.1
send_arp_status_5=1
address_6=29.89.51.253
netmask_6=255.255.0.0
netdetect_6=29.89.0.1
gateway_6=29.89.0.1
send_arp_status_6=1
address_7=29.89.96.167
netmask_7=255.255.0.0
netdetect_7=29.89.0.1
gateway_7=29.89.0.1
```

b. RoCE测试。

在Session1：在接收端执行-i卡id。

```
hccn_tool -i 7 -roce_test reset
hccn_tool -i 7 -roce_test ib_send_bw -s 4096000 -n 1000 -tcp
```

在Session2：在发送端执行-i卡id，后面的ip为上一步接收端卡的ip。

```
cd /usr/local/Ascend/ascend-toolkit/latest/tools/hccl_test
hccn_tool -i 0 -roce_test reset
hccn_tool -i 0 -roce_test ib_send_bw -s 4096000 -n 1000 address 192.168.100.18 -tcp
```

RoCE测试结果如图：

图 3-19 RoCE 测试结果（接收端）

```
[root@devserver-com hccl_test]# hccn_tool -i 7 -roce_test ib_send_bw -s 4096000 -n 1000 -tcp
Dsmi get perftest status end. (status=1)
Dsmi start roce perftest end. (out=1)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=2)
Dsmi get perftest status end. (status=1)
roce_report:

************************************
* Waiting for client to connect... *
************************************
---------------------------------------------------------------------------------------
                    Send BW Test
 Dual-port       : OFF          Device         : hns_0
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 RX depth        : 512
 CQ Moderation   : 100
 Mtu             : 4096[B]
 Link type       : Ethernet
 GID index       : 3
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x000a PSN 0xf97ccb
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:89:96:167
 remote address: LID 0000 QPN 0x001a PSN 0x3a835e
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:89:132:13
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 4096000    1000             0.00               23395.00                 0.005989
---------------------------------------------------------------------------------------
```

图 3-20 RoCE 测试结果（服务端）

```
[root@devserver-com hccl_test]# hccn_tool -i 0 -roce_test ib_send_bw -s 4096000 -n 1000 address 29.89.96.167 -tcp
Dsmi get perftest status end. (status=1)
Dsmi start roce perftest end. (out=1)
Dsmi get perftest status end. (status=1)
roce_report:
---------------------------------------------------------------------------------------
                    Send BW Test
 Dual-port       : OFF          Device         : hns_0
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 TX depth        : 128
 CQ Moderation   : 100
 Mtu             : 4096[B]
 Link type       : Ethernet
 GID index       : 3
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x001a PSN 0x3a835e
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:89:132:13
 remote address: LID 0000 QPN 0x000a PSN 0xf97ccb
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:89:96:167
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 4096000    1000             23372.40           23369.61                 0.005983
---------------------------------------------------------------------------------------
```

说明

- 当某网卡已经开始RoCE带宽测试时，再次启动任务会有如下报错：

  图 3-21 报错信息

  ```
  [root@devserver-com hccl_test]# hccn_tool -i 7 -roce_test ib_send_bw -s 4096 -n
   1000 -tcp
  Dsmi get perftest status end. (status=2)
  Roce perftest is doing, please try later.
  Cmd execute failed!
  ```

  需要执行下述命令后关闭roce_test任务后再启动任务。

  ```
  hccn_tool -i 7 -roce_test reset
  ```

- 可执行如下命令查看网卡状态。

  ```
  for i in {0..7};do hccn_tool -i ${i} -link -g;done
  ```

- 可执行如下命令查看单节点内网卡IP连通性。

  ```
  for i in $(seq 0 7);do hccn_tool -i $i -net_health -g;done
  ```

----结束

容器化个人调测环境搭建

步骤1 准备业务基础镜像。

当前推荐的开发模式是在物理机上启动自己的docker容器进行开发。容器镜像可以使用自己的实际业务镜像，也可以使用ModelArts提供的基础镜像，ModelArts提供两种基础镜像：Ascend+PyTorch镜像、Ascend+MindSpore镜像。

1. 根据所需要的环境拉取Ascend+PyTorch或Ascend+MindSpore镜像：

   ```
   # 配套Snt9b的容器镜像，示例如下：
   docker pull swr.<region-code>.myhuaweicloud.com/atelier/<image-name>:<image-tag>
   ```

2. 启动容器镜像，注意多人多容器共用机器时，需要将卡号做好预先分配，不能使用其他容器已使用的卡号。

   ```
   # 启动容器，请注意指定容器名称、镜像信息。ASCEND_VISIBLE_DEVICES指定容器要用的卡，0-1,3代表0 1 3这3块卡，-用于指定范围
   # -v /home:/home_host是指将宿主机home目录挂载到容器home_host目录，建议在容器中使用该挂载目录进行代码和数据的存储，以便持久化保存数据
   docker run -itd --cap-add=SYS_PTRACE -e ASCEND_VISIBLE_DEVICES=0  -v /home:/home_host -p 51234:22 -u=0 --name 自定义容器名称  上一步拉取的镜像SWR地址  /bin/bash
   ```

3. 执行下述命令进入容器。

   ```
   docker exec -ti 上一命令中的自定义容器名称 bash
   ```

4. 执行下述命令进入conda环境。

   ```
   source /home/ma-user/.bashrc
   cd ~
   ```

5. 查看容器中可以使用的卡信息。

   ```
   npu-smi info
   ```

   如果命令报如下错误，则代表容器启动时指定的“ASCEND_VISIBLE_DEVICES”卡号已被其他容器占用，此时需要重新选择卡号并重新启动新的容器。

   图 3-22 报错信息

   ```
   (PyTorch-1.11.0) [root@8e2a7f7f9f7a ma-user]# npu-smi info
   DrvMngGetConsoleLogLevel failed. (g_conLogLevel=3)
   dcmi model initialized failed, because the device is used. ret is -8020
   ```

6. npu-smi info检测正常后，可以执行一段命令进行简单的容器环境测试，能正常输出运算结果代表容器环境正常可用。

- pytorch镜像测试：

```
python3 -c "import torch;import torch_npu; a = torch.randn(3, 4).npu(); print(a + a);"
```

- mindspore镜像测试：

```
# 由于mindspore的run_check程序当前未适配Snt9b，需要先设置2个环境变量才能测试
unset MS_GE_TRAIN
unset MS_ENABLE_GE
python -c "import
mindspore;mindspore.set_context(device_target='Ascend');mindspore.run_check()"
# 测试完需要恢复环境变量，实际跑训练业务的时候需要用到
export MS_GE_TRAIN=1
export MS_ENABLE_GE=1
```

图 3-23 进入 conda 环境并进行测试

```
[root@devserver-modelarts-demanager-0eaabe8f ~]# docker run -itd --cap-add=SYS_PTRACE -e ASCEND_VISIBLE_DEVICES=3 -v
/home:/host_home -u=0 --name pytorch_test swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.
11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-d910b-20230815141604-3685231 /bin/bash
0292be41ac1ef03a37b7c78adcff4fc999a967e1163e5f6e565edbe6a638c69b
[root@devserver-modelarts-demanager-0eaabe8f ~]# docker exec -ti 0292be41a bash
The environment has been set
[root@0292be41ac1e ma-user]# source .bashrc
The environment has been set
The environment has been set
(PyTorch-1.11.0) [root@0292be41ac1e ma-user]# python3 -c "import torch;import torch_npu; a = torch.randn(3, 4).npu();
 print(a + a);"
tensor([[-1.0911, -0.4146,  1.6027,  1.8585],
        [ 3.2549,  0.7026,  2.9356,  0.9544],
        [ 5.1409, -0.8820, -0.3400,  0.0257]], device='npu:0')
(PyTorch-1.11.0) [root@0292be41ac1e ma-user]#
```

步骤2 （可选）配置容器SSH可访问。

若在开发时，需要使用VS Code或SSH工具直接连接到容器中进行开发，需要进行以下配置。

1. 进入容器后，执行SSH启动命令来启动SSH服务：

```
ssh-keygen  -A
/usr/sbin/sshd
#  查看ssh进程已启动
ps -ef |grep ssh
```

2. 设置容器root密码，根据提示输入密码：

```
passwd
```

图 3-24 设置 root 密码

```
[root@9f4f3b6794f7 ~]$ passwd
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@9f4f3b6794f7 ~]$
```

3. 执行exit命令退出容器，在宿主机上执行ssh测试：

```
ssh root@宿主机IP -p 51234（映射的端口号）
```

图 3-25 执行 ssh 测试

```
[root@localhost home]# ssh root@90.90.3.71 -p 51234
root@90.90.3.71's password:

Authorized users only. All activities may be monitored and reported.
[root@9f4f3b6794f7 ~]$
```

如果在宿主机执行ssh容器测试时报错Host key verification failed，可删除宿主机上的文件~/.ssh/known_host后再重试。

4. 使用VS Code SSH连接容器环境。

   如果之前未使用过VS Code SSH功能，可参考**Step1 添加Remote-SSH插件**进行VSCode环境安装和Remote-SSH插件安装。

打开VSCode Terminal，执行如下命令在本地计算机生成密钥对，如果您已经有一个密钥对，则可以跳过此步骤：

```
ssh-keygen -t rsa
```

将公钥添加到远程服务器的授权文件中，注意替换服务器IP以及容器的端口号：

```
cat ~/.ssh/id_rsa.pub | ssh root@服务器IP -p 容器端口号 "mkdir -p ~/.ssh && cat >>  ~/.ssh/
authorized_keys"
```

打开VSCode的Remote-SSH配置文件，添加SSH配置项，注意替换服务器IP以及容器的端口号：

```
Host Snt9b-dev
    HostName 服务器IP
    User root
    port 容器SSH端口号
    identityFile ~\.ssh\id_rsa
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
    ForwardAgent yes
```

注意：**这里是使用密钥登录，如果需要使用密码登录，请去掉identityFile配置，并在连接过程中根据提示多次输入密码。**

连接成功后安装python插件，请参考**安装Python插件**。

步骤3 （可选）安装CANN Toolkit。

当前ModelArts提供的预置镜像中已安装CANN Toolkit，如果需要替换版本或者使用自己的未预置CANN Toolkit的镜像，可参考如下章节进行安装。

1. 查看容器内是否已安装CANN Toolkit，如果显示有版本号则已安装：
   ```
   cat /usr/local/Ascend/ascend-toolkit/latest/aarch64-linux/ascend_toolkit_install.info
   ```
2. 如果未安装或需要升级版本，则需要从官网下载相关软件包，其中社区版可以直接下载（**下载地址**），商用版是权限受控，仅华为工程师和渠道用户有权限下载（**下载链接**）。

   安装CANN Toolkit，注意替换包名。
   ```
   chmod 700 *.run
   ./Ascend-cann-toolkit_6.3.RC2_linux-aarch64.run --full --install-for-all
   ```
3. 如果已安装，但需要升级版本，注意替换包名：
   ```
   chmod 700 *.run
   ./Ascend-cann-toolkit_6.3.RC2_linux-aarch64.run --upgrade --install-for-all
   ```

步骤4 （可选）安装MindSpore Lite。

当前预置镜像中已安装MindSpore Lite，如果需要替换版本或者使用自己的未预置MindSpore Lite的镜像，可参考如下章节进行安装。

1. 查看容器中是否已安装MS Lite，如果已经显示出mindspore-lite软件信息和版本号，则是已经安装好的：
   ```
   pip show mindspore-lite
   ```
2. 如果未安装，则从官网下载包（**下载链接**），下载whl包和tar.gz包并执行安装，注意替换包名：
   ```
   pip install mindspore_lite-2.1.0-cp37-cp37m-linux_aarch64.whl
   mkdir -p /usr/local/mindspore-lite
   tar -zxvf mindspore-lite-2.1.0-linux-aarch64.tar.gz -C /usr/local/mindspore-lite --strip-components 1
   ```

步骤5 配置pip源和yum源。

- 配置pip源

  使用ModelArts提供的预置镜像中pip源已经直接配置好可用，如果用户使用自己的业务镜像，可参考**步骤5**进行配置。
- 配置yum源

  执行如下命令配置yum源：


```
# 自动配置yum源
wget http://mirrors.myhuaweicloud.com/repo/mirrors_source.sh && bash mirrors_source.sh

# 测试
yum update --allowerasing --skip-broken --nobest
```

步骤6 安装git-lfs并通过git clone下载代码。

git clone和git-lfs下载大模型可以参考如下操作。

1. 由于欧拉源上没有git-lfs包，所以需要从压缩包中解压使用，在浏览器中输入如下地址下载git-lfs压缩包并上传到服务器的/home目录下，该目录在容器启动时挂载到容器/home_host目录下，这样在容器中可以直接使用。
   ```
   https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```
2. 进入容器，执行安装git-lfs命令。
   ```
   cd /home_host
   tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
   cd git-lfs-3.2.0
   sh install.sh
   ```
3. 设置git配置去掉ssl校验。
   ```
   git config --global http.sslVerify false
   ```
4. git clone代码仓，以diffusers为例（注意替换用户个人开发目录）。
   ```
   #  git clone diffusers源码，-b参数可指定分支，注意替换用户个人开发目录
   cd /home_host/用户个人目录
   mkdir sd
   cd sd
   git clone https://github.com/huggingface/diffusers.git -b v0.11.1-patch
   ```
   git clone Hugging Face上的模型，以SD模型为例。如果下载时若出现“SSL_ERROR_SYSCALL”报错，多重试几次即可。另外由于网络限制以及文件较大，下载可能很慢需要数个小时，如果重试多次还是失败，建议直接从网站下载大文件后上传到服务器/home目录的个人开发目录中。如果下载时需要跳过大文件，可以设置GIT_LFS_SKIP_SMUDGE=1。
   ```
   git lfs install
   git clone https://huggingface.co/runwayml/stable-diffusion-v1-5 -b onnx
   ```

图 3-26 代码下载成功

```
[root@38a757e4636a sd]# git clone https://github.com/huggingface/diffusers.git -b v0.11.1-patch
Cloning into 'diffusers'...
remote: Enumerating objects: 34118, done.
remote: Counting objects: 100% (10965/10965), done.
remote: Compressing objects: 100% (765/765), done.
remote: Total 34118 (delta 10639), reused 10273 (delta 10190), pack-reused 23153
Receiving objects: 100% (34118/34118), 21.44 MiB | 9.58 MiB/s, done.
Resolving deltas: 100% (25313/25313), done.
[root@38a757e4636a sd]# cd diffusers/
[root@38a757e4636a diffusers]# git branch
* v0.11.1-patch
[root@38a757e4636a diffusers]#
```

步骤7 容器环境保存镜像。

配置好环境后可以进行业务代码的开发调试。通常为了避免机器重启后环境丢失，建议将已经配好的环境保存成新的镜像，命令如下：

```
# 查看需要保存为镜像的容器ID
docker ps
# 保存镜像
docker commit 容器ID  自定义镜像名:自定义镜像tag
# 查看已保存的镜像
docker images
# 如果需要将镜像分享给其他人在其他环境使用，可将镜像保存为本地文件，该命令耗时较久，保存完后ls可查看到文件
docker save -o 自定义名称.tar 镜像名:镜像tag
# 其他机器上使用时加载文件，加载好后docker images即可查看到该镜像
docker load --input 自定义名称.tar
```

到此环境配置就结束了，后续可以根据相关的迁移指导书做业务迁移到昇腾的开发调测工作。

----结束

3.4.2 GPU 服务器上配置 Lite Server 资源软件环境

场景描述

本文旨在指导如何在GPU裸金属服务器上，安装NVIDIA、CUDA驱动等环境配置。由于不同GPU预置镜像中预安装的软件不同，您通过**Lite Server算力资源和镜像版本配套关系**章节查看已安装的软件。下面为常见的软件安装步骤，您可针对需要安装的软件查看对应的内容：

- **安装NVIDIA驱动**
- **安装CUDA驱动**
- **安装Docker**
- **安装nvidia-fabricmanager**

以下提供常见的配置场景，您可查看相关文档方便您快速配置：

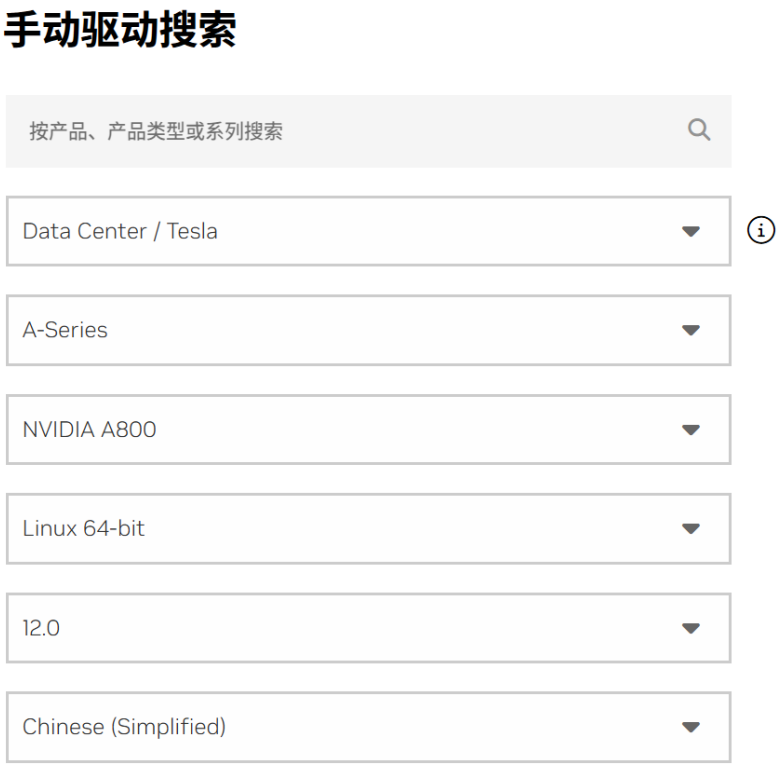
- **GP Vnt1裸金属服务器EulerOS 2.9安装NVIDIA 515+CUDA 11.7**
- **GP Vnt1裸金属服务器Ubuntu 18.04安装NVIDIA 470+CUDA 11.4**
- **GP Vnt1裸金属服务器Ubuntu18.04安装NVIDIA 515+CUDA 11.7**
- **GP Ant8裸金属服务器Ubuntu 20.04安装NVIDIA 515+CUDA 11.7**

安装 NVIDIA 驱动

步骤1 打开**NVIDIA官方网站**。

步骤2 以Ant8规格为例，根据Ant8的详细信息和您所需的cuda版本选择驱动。

图 3-27 驱动选择



选择后会自动出现Driver版本并下载，或者直接。

```
wget https://cn.download.nvidia.com/tesla/470.182.03/NVIDIA-Linux-x86_64-470.182.03.run
```

步骤3 添加权限。

```
chmod +x NVIDIA-Linux-x86_64-470.182.03.run
```

步骤4 运行安装文件。

```
./NVIDIA-Linux-x86_64-470.182.03.run
```

至此NVIDIA-DRIVER驱动安装完成。

----结束

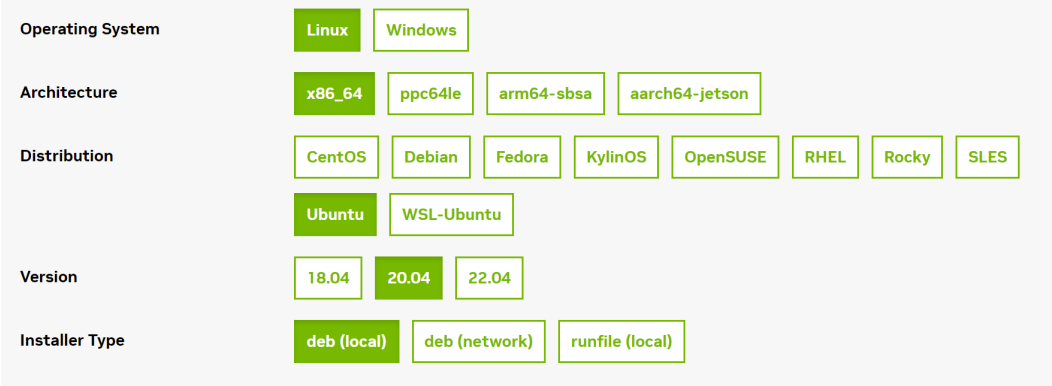
安装 CUDA 驱动

上文安装NVIDIA驱动是根据CUDA12.0选择的安装包， 因此下文默认安装CUDA 12.0。

步骤1 进入**CUDA Toolkit**页面。

步骤2 选择Operating System、Architecture、Distribution、Version、Installer Type后，会生成对应的安装命令，复制安装命令并运行即可。

图 3-28 选择版本



对应所得安装命令为：

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin
sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
wget https://developer.download.nvidia.com/compute/cuda/12.1.1/local_installers/cuda-repo-ubuntu2004-12-1-local_12.1.1-530.30.02-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2004-12-1-local_12.1.1-530.30.02-1_amd64.deb
sudo cp /var/cuda-repo-ubuntu2004-12-1-local/cuda-*-keyring.gpg /usr/share/keyrings/
sudo apt-get update
sudo apt-get -y install cuda
```

说明

若需要找到历史版本的CUDA，您可请单击CUDA历史版本的**下载链接**查找所需的CUDA版本。

----结束

安装 Docker

部分Vnt1裸金属服务器的预置镜像中未安装Docker，您可参考以下步骤进行安装。

步骤1 安装Docker。

```
curl https://get.docker.com | sh && sudo systemctl --now enable docker
```

步骤2 安装NIVDIA容器插件。

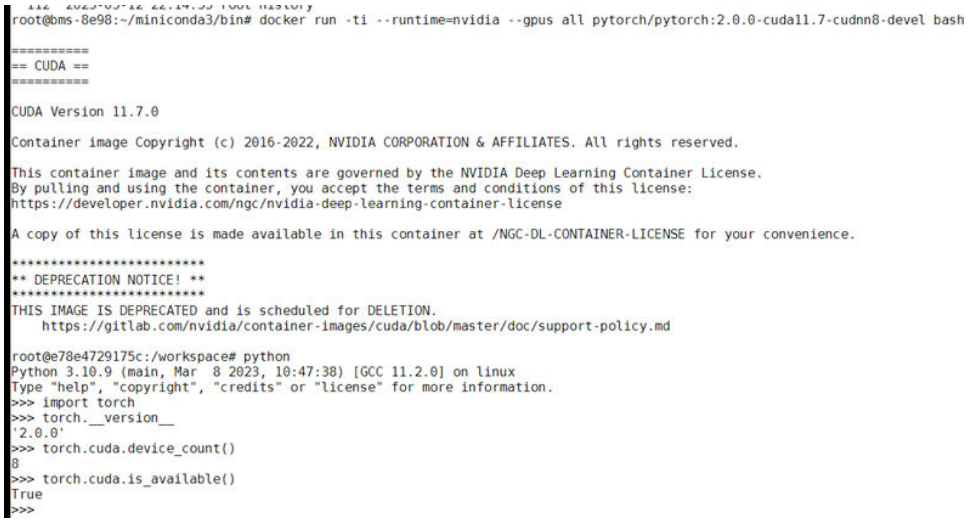
```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
&& curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
&& curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-container.list |
 sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' |
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
apt-get update
apt-get install -y nvidia-container-toolkit
nvidia-ctk runtime configure --runtime=docker
systemctl restart docker
```

步骤3 验证Docker模式环境是否安装成功。

基于PyTorch2.0镜像验证（本案例中镜像较大，拉取时间可能较长）。

```
docker run -ti --runtime=nvidia --gpus all pytorch/pytorch:2.0.0-cuda11.7-cudnn8-devel bash
```

图 3-29 成功拉取镜像



----结束

安装 nvidia-fabricmanager

Ant系列GPU支持NvLink & NvSwitch，若您使用多GPU卡的机型，需额外安装与驱动版本对应的nvidia-fabricmanager服务使GPU卡间能够互联，否则可能无法正常使用GPU实例。

说明

nvidia-fabricmanager必须和nvidia driver版本保持一致。

以安装515.105.01版本为例。

```
version=515.105.01
main_version=$(echo $version | awk -F '.' '{print $1}')
apt-get update
apt-get -y install nvidia-fabricmanager-${main_version}=${version}-*
```

验证驱动安装结果：启动fabricmanager服务并查看状态是否为“RUNNING”。

```
nvidia-smi -pm 1
nvidia-smi
systemctl enable nvidia-fabricmanager
systemctl start nvidia-fabricmanager
systemctl status nvidia-fabricmanager
```

GP Vnt1 裸金属服务器 EulerOS 2.9 安装 NVIDIA 515+CUDA 11.7

本小节旨在指导如何在GP Vnt1裸金属服务器上（Euler2.9系统），安装NVIDIA驱动版本515.105.01，CUDA版本11.7.1。

步骤1 安装NVIDIA驱动。

```
wget https://us.download.nvidia.com/tesla/515.105.01/NVIDIA-Linux-x86_64-515.105.01.run
chmod 700 NVIDIA-Linux-x86_64-515.105.01.run

yum install -y elfutils-libelf-devel
./NVIDIA-Linux-x86_64-515.105.01.run --kernel-source-path=/usr/src/kernels/
4.18.0-147.5.1.6.h998.eulerosv2r9.x86_64
```

说明

默认情况下Vnt1裸金属服务器在EulerOS 2.9使用的yum源是“http://repo.huaweicloud.com”，该源可用。若执行“yum update”时报错， 显示有软件包冲突等问题， 可通过“yum remove xxx软件包”解决该问题。

NVIDIA的驱动程序是一个二进制文件，需使用系统中的libelf库(在elfutils-libelf-devel开发包)中。它提供了一组C函数，用于读取、修改和创建ELF文件，而NVIDIA驱动程序需要使用这些函数来解析当前正在运行的内核和其他相关信息。

安装过程中的提示均选OK或YES，安装好后执行reboot重启机器，再次登录后执行命令查看GPU卡信息。

```
nvidia-smi -pm 1    #该命令执行时间较长，请耐心等待，作用为启用持久模式，可以优化Linux实例上GPU设备的性能
nvidia-smi
```

步骤2 安装CUDA。

```
wget https://developer.download.nvidia.com/compute/cuda/11.7.1/local_installers/cuda_11.7.1_515.65.01_linux.run
chmod 700 cuda_11.7.1_515.65.01_linux.run
./cuda_11.7.1_515.65.01_linux.run --toolkit --samples --silent
```

安装好后执行以下命令检查安装结果：

```
/usr/local/cuda/bin/nvcc -V
```

步骤3 PyTorch2.0安装和CUDA验证指南。

PyTorch2.0所需环境为Python3.10， 安装配置miniconda环境。

1. miniconda安装并创建alpha环境。

   ```
   wget https://repo.anaconda.com/miniconda/Miniconda3-py310_23.1.0-1-Linux-x86_64.sh
   chmod 750 Miniconda3-py310_23.1.0-1-Linux-x86_64.sh
   bash Miniconda3-py310_23.1.0-1-Linux-x86_64.sh -b -p /home/miniconda
   export PATH=/home/miniconda/bin:$PATH
   conda create --quiet --yes -n alpha python=3.10
   ```

2. 安装pytorch2.0并验证cuda状态。

   在alpha环境下安装torch2.0，使用清华PIP源完成。

   ```
   source activate alpha
   pip install torch==2.0 -i https://pypi.tuna.tsinghua.edu.cn/simple
   python
   ```

   验证torch与cuda的安装状态，输出为True即为正常。

   ```
   import torch
   print(torch.cuda.is_available())
   ```

----结束

GP Vnt1 裸金属服务器 Ubuntu 18.04 安装 NVIDIA 470+CUDA 11.4

本小节旨在指导如何在GP Vnt1裸金属服务器上（Ubuntu 18.04系统），安装NVIDIA驱动版本470，CUDA版本11.4。

步骤1 安装NVIDIA驱动。

```
apt-get update
sudo apt-get install nvidia-driver-470
```

步骤2 安装CUDA。

```
wget https://developer.download.nvidia.com/compute/cuda/11.4.4/local_installers/cuda_11.4.4_470.82.01_linux.run
chmod +x cuda_11.4.4_470.82.01_linux.run
./cuda_11.4.4_470.82.01_linux.run --toolkit --samples --silent
```


步骤3 验证NVIDIA安装结果。

```
nvidia-smi -pm 1
nvidia-smi
/usr/local/cuda/bin/nvcc -V
```

步骤4 安装Pytorch2.0和验证CUDA验证。

PyTorch2.0所需环境为Python3.10， 安装配置miniconda环境。

1. miniconda安装并创建alpha环境。

   ```
   wget https://repo.anaconda.com/miniconda/Miniconda3-py310_23.1.0-1-Linux-x86_64.sh
   chmod 750 Miniconda3-py310_23.1.0-1-Linux-x86_64.sh
   bash Miniconda3-py310_23.1.0-1-Linux-x86_64.sh -b -p /home/miniconda
   export PATH=/home/miniconda/bin:$PATH
   conda create --quiet --yes -n alpha python=3.10
   ```

2. 安装pytorch2.0并验证cuda状态。

   在alpha环境下安装torch2.0，使用清华PIP源完成。

   ```
   source activate alpha
   conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia
   python
   ```

   验证torch与cuda的安装状态，输出为True即为正常。

   ```
   import torch
   print(torch.cuda.is_available())
   ```

----结束

GP Vnt1 裸金属服务器 Ubuntu18.04 安装 NVIDIA 515+CUDA 11.7

本小节旨在指导如何在GP Vnt1裸金属服务器上（Ubuntu 18.04系统），安装NVIDIA驱动版本515、CUDA版本11.7和Docker。

步骤1 NVIDIA驱动安装。

```
wget https://us.download.nvidia.com/tesla/515.105.01/NVIDIA-Linux-x86_64-515.105.01.run
chmod +x NVIDIA-Linux-x86_64-515.105.01.run
./NVIDIA-Linux-x86_64-515.105.01.run
```

步骤2 CUDA安装。

```
wget https://developer.download.nvidia.com/compute/cuda/11.7.1/local_installers/
cuda_11.7.1_515.65.01_linux.run
chmod +x cuda_11.7.1_515.65.01_linux.run
./cuda_11.7.1_515.65.01_linux.run --toolkit --samples –silent
```

步骤3 安装Docker。

```
curl https://get.docker.com | sh && sudo systemctl --now enable docker
```

步骤4 安装NIVDIA容器插件。

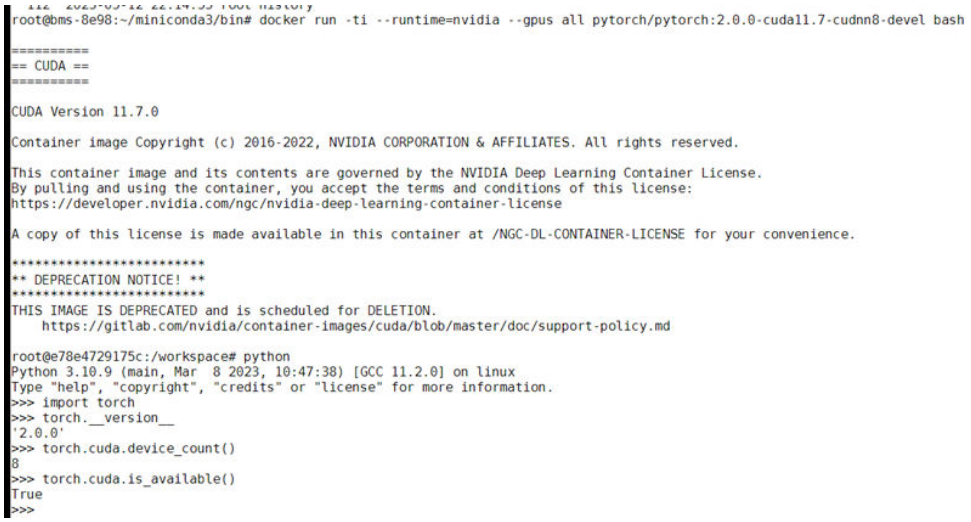
```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
&& curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/
keyrings/nvidia-container-toolkit-keyring.gpg
&& curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-container.list |
 sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' |
sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
apt-get update
apt-get install -y nvidia-container-toolkit
nvidia-ctk runtime configure --runtime=docker
systemctl restart docker
```

步骤5 验证Docker模式环境是否安装成功。

基于PyTorch2.0镜像验证（本案例中镜像较大，拉取时间可能较长）。

```
docker run -ti --runtime=nvidia --gpus all pytorch/pytorch:2.0.0-cuda11.7-cudnn8-devel bash
```


图 3-30 成功拉取镜像



----结束

GP Ant8 裸金属服务器 Ubuntu 20.04 安装 NVIDIA 515+CUDA 11.7

本小节旨在指导如何在GP Ant8裸金属服务器上（Ubuntu 20.04系统），安装NVIDIA驱动版本515、CUDA版本11.7、nvidia-fabricmanager版本515，并进行nccl-test测试。

步骤1 替换apt源。

```
sudo sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
sudo sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
sudo apt update
```

步骤2 安装nvidia驱动。

```
wget https://us.download.nvidia.com/tesla/515.105.01/NVIDIA-Linux-x86_64-515.105.01.run
chmod +x NVIDIA-Linux-x86_64-515.105.01.run
./NVIDIA-Linux-x86_64-515.105.01.run
```

步骤3 安装cuda。

```
# run包安装
wget https://developer.download.nvidia.com/compute/cuda/11.7.0/local_installers/
cuda_11.7.0_515.43.04_linux.run
chmod +x cuda_11.7.0_515.43.04_linux.run
./cuda_11.7.0_515.43.04_linux.run --toolkit --samples --silent
```

步骤4 安装nccl。

说明

- nccl安装可参考**NCCL Documentation**。
- nccl和cuda版本的配套关系和安装方法参考**NCL Downloads**。

本文使用cuda版本是11.7，因此安装nccl的命令为:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-
keyring_1.0-1_all.deb
sudo dpkg -i cuda-keyring_1.0-1_all.deb
sudo apt update
sudo apt install libnccl2=2.14.3-1+cuda11.7 libnccl-dev=2.14.3-1+cuda11.7
```

安装完成后可以查看：

图 3-31 查看 nccl

步骤5 安装nvidia-fabricmanager。

说明

nvidia-fabricmanager必须和nvidia driver版本保持一致。

```
version=515.105.01
main_version=$(echo $version | awk -F '.' '{print $1}')
apt-get update
apt-get -y install nvidia-fabricmanager-${main_version}=${version}-*
```

验证驱动安装结果：启动fabricmanager服务并查看状态是否为“RUNNING”。

```
nvidia-smi -pm 1
nvidia-smi
systemctl enable nvidia-fabricmanager
systemctl start nvidia-fabricmanager
systemctl status nvidia-fabricmanager
```

步骤6 安装nv-peer-memory。

```
git clone https://github.com/Mellanox/nv_peer_memory.git
cd ./nv_peer_memory
./build_module.sh
cd /tmp
tar xzf /tmp/nvidia-peer-memory_1.3.orig.tar.gz
cd nvidia-peer-memory-1.3
dpkg-buildpackage -us -uc
dpkg -i ../nvidia-peer-memory-dkms_1.2-0_all.deb
```

nv_peer_mem工作在linux内核态，安装完成后需要看是否加载到内核，通过执行“lsmod | grep peer”查看是否加载。

说明

- 如果git clone拉不下来代码，可能需要先设置下git的配置：
```
git config --global core.compression -1
export GIT_SSL_NO_VERIFY=1
git config --global http.sslVerify false
git config --global http.postBuffer 10524288000
git config --global http.lowSpeedLimit 1000
git config --global http.lowSpeedTime 1800
```
- 如果安装完成后lsmod看不到nv-peer-memory，可能是由于ib驱动版本过低导致，此时需要升级ib驱动，升级命令：
```
wget https://content.mellanox.com/ofed/MLNX_OFED-5.4-3.6.8.1/MLNX_OFED_LINUX-5.4-3.6.8.1-ubuntu20.04-x86_64.tgz
tar -zxvf MLNX_OFED_LINUX-5.4-3.6.8.1-ubuntu20.04-x86_64.tgz
cd MLNX_OFED_LINUX-5.4-3.6.8.1-ubuntu20.04-x86_64
apt-get install -y python3 gcc quilt build-essential bzip2 dh-python pkg-config dh-autoreconf python3-distutils debhelper make
./mlnxofedinstall --add-kernel-support
```
- 如果想安装其它更高版本的ib驱动，请参考**Linux InfiniBand Drivers**。比如要安装MLNX_OFED-5.8-2.0.3.0 (当前最新版本)，则命令为：
```
wget https://content.mellanox.com/ofed/MLNX_OFED-5.8-2.0.3.0/MLNX_OFED_LINUX-5.8-2.0.3.0-ubuntu20.04-x86_64.tgz
tar -zxvf MLNX_OFED_LINUX-5.8-2.0.3.0-ubuntu20.04-x86_64.tgz
cd MLNX_OFED_LINUX-5.8-2.0.3.0-ubuntu20.04-x86_64
apt-get install -y python3 gcc quilt build-essential bzip2 dh-python pkg-config dh-autoreconf python3-distutils debhelper make
./mlnxofedinstall --add-kernel-support
```
- 安装完nv_peer_mem， 如果想查看其状态可以输入如下指令：
```
/etc/init.d/nv_peer_mem/ status
```
如果发现没有此文件，则可能安装的时候没有默认复制过来，需要复制即可：
```
cp /tmp/nvidia-peer-memory-1.3/nv_peer_mem.conf  /etc/infiniband/
cp /tmp/nvidia-peer-memory-1.3/debian/tmp/etc/init.d/nv_peer_mem   /etc/init.d/
```

步骤7 设置环境变量。

说明

MPI路径版本需要匹配，可以通过“ls /usr/mpi/gcc/”查看openmpi的具体版本。

```
# 加入到~/.bashrc
export LD_LIBRARY_PATH=/usr/local/cuda/lib:usr/local/cuda/lib64:/usr/include/nccl.h:/usr/mpi/gcc/openmpi-4.1.2a1/lib:$LD_LIBRARY_PATH
export PATH=$PATH:/usr/local/cuda/bin:/usr/mpi/gcc/openmpi-4.1.2a1/bin
```

步骤8 安装编译nccl-test。

```
cd /root
git clone https://github.com/NVIDIA/nccl-tests.git
cd ./nccl-tests
make  MPI=1 MPI_HOME=/usr/mpi/gcc/openmpi-4.1.2a1 -j 8
```

说明

编译时需要加上MPI=1的参数，否则无法进行多机之间的测试。

MPI路径版本需要匹配，可以通过“ls /usr/mpi/gcc/”查看openmpi的具体版本。

步骤9 nccl-test测试。

- 单机测试：
```
/root/nccl-tests/build/all_reduce_perf -b 8 -e 1024M -f 2 -g 8
```
- 多机测试（btl_tcp_if_include后面替换为主网卡名称）：
```
mpirun --allow-run-as-root --hostfile hostfile -mca btl_tcp_if_include eth0 -mca btl_openib_allow_ib true -x NCCL_DEBUG=INFO -x NCCL_IB_GID_INDEX=3 -x NCCL_IB_TC=128 -x NCCL_ALGO=RING -x NCCL_IB_HCA=^mlx5_bond_0 -x LD_LIBRARY_PATH  /root/nccl-tests/build/all_reduce_perf -b 8 -e 11g -f 2 -g 8
```

hostfile格式：

```
#主机私有IP 单节点进程数
192.168.20.1 slots=1
192.168.20.2 slots=1
```

NCCL环境变量说明：

- NCCL_IB_GID_INDEX=3 ： 数据包走交换机的队列4通道，这是RoCE协议标准。
- NCCL_IB_TC=128 ： 使用RoCE v2协议，默认使用RoCE v1，但是v1在交换机上没有拥塞控制，可能会丢包，而且后续的交换机不会支持v1，会导致无法运行。
- NCCL_ALGO=RING ： nccl_test的总线bandwidth是在假定是Ring算法的情况下计算出来的。

  计算公式是有假设的： 总线带宽 = 算法带宽 * 2（N-1）/ N ，算法带宽 = 数据量 / 时间

  但是这个计算公式的前提是用Ring算法，Tree算法的总线带宽不可以这么计算。

  如果Tree算法算出来的总线带宽相当于是相对Ring算法的性能加速。算法计算总耗时减少了，所以用公式算出来的总线带宽也增加了。理论上Tree算法是比Ring算法更优的，但是Tree算法对网络的要求比Ring高，计算可能不太稳定。 Tree算法可以用更少的数据通信量完成all reduce计算，但用来测试性能不太合适。因此，会出现两节点实际带宽100，但测试出速度110，甚至130GB/s的情况。加这个参数以后，2节点和2节点以上情况的速度才会稳定一些。

  说明

  测试时需要执行mpirun的节点到hostfile中的节点间有免密登录，设置SSH免密登录方法如下：

  1. 客户端生成公私钥。

     执行如下命令，在本地客户端生成公私钥（一路回车默认即可）。

     ```
     ssh-keygen
     ```

     上面这个命令会在用户目录.ssh文件夹下创建“id_rsa.pub”（公钥）和“id_rsa”（私钥），可通过如下命令查看：

     ```
     cd ~/.ssh
     ```

  2. 上传公钥到服务器。

     例如用户名为root，服务器地址为192.168.222.213，则将公钥上传至服务器的命令如下：

     ```
     ssh-copy-id -i ~/.ssh/id_rsa.pub root@192.168.222.213
     ```

     通过如下命令可以看到客户端写入到服务器的id_rsa.pub（公钥）内容：

     ```
     cd ~/.ssh
     vim authorized_keys
     ```

  3. 测试免密登录。

     客户端通过ssh连接远程服务器，即可免密登录。

     ```
     ssh root@192.168.222.213
     ```

----结束

4 Lite Server 资源使用

4.1 GPT-2 基于 Server 适配 PyTorch GPU 的训练推理指导

场景描述

本文将介绍在GP Ant8裸金属服务器中，使用DeepSpeed框架训练GPT-2（分别进行单机单卡和单机多卡训练）。 训练完成后给出自动式生成内容，和交互式对话框模式。

背景信息

- Megatron-DeepSpeed

  Megatron-DeepSpeed是一个基于PyTorch的深度学习模型训练框架。它结合了两个强大的工具：Megatron-LM和DeepSpeed，可在具有分布式计算能力的系统上进行训练，并且充分利用了多个GPU和深度学习加速器的并行处理能力。可以高效地训练大规模的语言模型。

  Megatron-LM是一个用于大规模语言建模的模型。它基于GPT（Generative Pre-trained Transformer）架构，这是一种基于自注意力机制的神经网络模型，广泛用于自然语言处理任务，如文本生成、机器翻译和对话系统等。

  DeepSpeed是开源的加速深度学习训练的库。它针对大规模的模型和分布式训练进行了优化，可以显著提高训练速度和效率。DeepSpeed提供了各种技术和优化策略，包括分布式梯度下降、模型并行化、梯度累积和动态精度缩放等。它还支持优化大模型的内存使用和计算资源分配。

- GPT2

  GPT2（Generative Pre-trained Transformer 2），是OpenAI组织在2018年于GPT模型的基础上发布的新预训练模型，是一个基于Transformer且非常庞大的语言模型。它在大量数据集上进行了训练，直接运行一个预训练好的GPT-2模型:给定一个预定好的起始单词或者句子，可以让它自行地随机生成后续的文本。

环境准备

在华为云ModelArts Server预购相关超强算力的GPU裸金属服务器，并选择AIGC场景通用的镜像，完成使用Megatron-DeepSpeed训练GPT2模型。本最佳实践使用以下镜像和规格：

- 镜像选择：Ubuntu 20.04 x86 64bit SDI3 for Ant8 BareMetal with RoCE and NVIDIA-525 CUDA-12.0。

- 裸金属规格选择： GP Ant8，包含8张GPU卡以及8张RoCE网卡。

关于Ant8裸金属服务器的购买，可以在华为云官网提工单至ModelArts云服务， 完成资源的申请。

步骤 1 安装模型

步骤1 安装Megatron-DeepSpeed框架。

1. 使用root用户SSH的方式登录GPU裸金属服务器。
2. 拉取pytorch镜像，可以选择常用的镜像源进行下载。
   ```
   docker pull nvcr.io/nvidia/pytorch:21.10-py3
   ```
3. 启动容器。
   ```
   docker run -d -t --network=host --gpus all --privileged --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 --name megatron-deepspeed -v /etc/localtime:/etc/localtime -v /root/.ssh:/root/.ssh nvcr.io/nvidia/pytorch:21.10-py3
   ```
4. 执行以下命令，进入容器终端。
   ```
   docker exec -it megatron-deepspeed bash
   ```
5. 下载Megatron-DeepSpeed框架。
   ```
   git clone https://github.com/bigscience-workshop/Megatron-DeepSpeed
   ```

   说明

   若git clone失败，可以尝试先下载至本地，然后复制至服务器中，在docker cp至容器中。
6. 安装Megatron-DeepSpeed框架。
   ```
   cd Megatron-DeepSpeed
   pip install -r requirements.txt -i http://mirrors.myhuaweicloud.com/pypi/web/simple --trusted-host mirrors.myhuaweicloud.com
   pip install mpi4py -i http://mirrors.myhuaweicloud.com/pypi/web/simple --trusted-host mirrors.myhuaweicloud.com
   ```
7. 修改测试代码，注释掉以下文件的断言所在行。
   ```
   vim /workspace/Megatron-DeepSpeed/megatron/model/fused_softmax.py +191
   ```

   在“assert mask is None, "Mask is silently ignored due to the use of a custom kernel"”前加“#”，即：

   ```
   # assert mask is None, "Mask is silently ignored due to the use of a custom kernel"
   ```

步骤2 数据集下载和预处理。

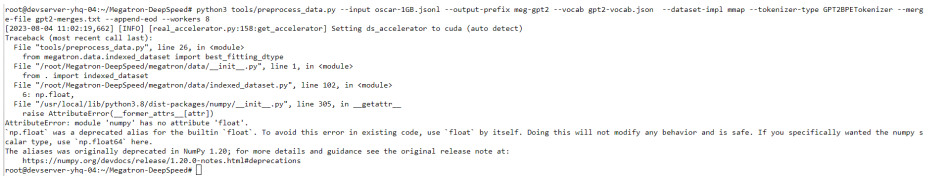
本实践中选择使用1GB 79K-record的JSON格式的OSCAR数据集。

1. 下载数据集。
   ```
   wget https://huggingface.co/bigscience/misc-test-data/resolve/main/stas/oscar-1GB.jsonl.xz
   wget https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-vocab.json
   wget https://s3.amazonaws.com/models.huggingface.co/bert/gpt2-merges.txt
   ```
2. 解压数据集。
   ```
   xz -d oscar-1GB.jsonl.xz
   ```
3. 预处理数据。
   ```
   python3 tools/preprocess_data.py \
       --input oscar-1GB.jsonl \
       --output-prefix meg-gpt2 \
       --vocab gpt2-vocab.json \
       --dataset-impl mmap \
       --tokenizer-type GPT2BPETokenizer \
       --merge-file gpt2-merges.txt \
       --append-eod \
       --workers 8
   ```


说明

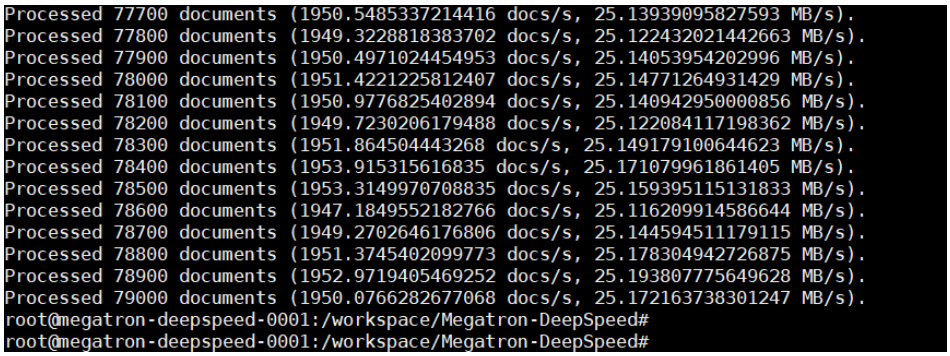
若发生如下“np.float”报错，按照报错提示修改为“float”即可。

图 4-1 预处理数据报错



4. 数据预处理完成标识。

图 4-2 数据预处理完成



5. 新建data目录并移动处理好的数据。

```
mkdir data
mv meg-gpt2* ./data
mv gpt2* ./data
```

----结束

步骤 2 单机单卡训练

本小节使用上文的服务器环境和安装好的模型， 使用GP Ant8裸金属服务器， 完成单机单卡GPT-2 MEDIUM模型的训练。

步骤1 创建预训练脚本文件。

1. 执行以下命令，创建预训练脚本文件。

```
vim pretrain_gpt2.sh
```

2. 在文件中添加以下信息。

```
#! /bin/bash

# Runs the "345M" parameter model

GPUS_PER_NODE=1
# Change for multinode config
MASTER_ADDR=localhost
MASTER_PORT=6000
NNODES=1
NODE_RANK=0
WORLD_SIZE=$(($GPUS_PER_NODE*$NNODES))

DATA_PATH=data/meg-gpt2_text_document
CHECKPOINT_PATH=checkpoints/gpt2

DISTRIBUTED_ARGS="--nproc_per_node $GPUS_PER_NODE --nnodes $NNODES --node_rank
```

```
$NODE_RANK --master_addr $MASTER_ADDR --master_port $MASTER_PORT"

python -m torch.distributed.launch $DISTRIBUTED_ARGS \
       pretrain_gpt.py \
       --tensor-model-parallel-size 1 \
       --pipeline-model-parallel-size 1 \
       --num-layers 24 \
       --hidden-size 1024 \
       --num-attention-heads 16 \
       --micro-batch-size 4 \
       --global-batch-size 8 \
       --seq-length 1024 \
       --max-position-embeddings 1024 \
       --train-iters 5000 \
       --lr-decay-iters 320000 \
       --save $CHECKPOINT_PATH \
       --load $CHECKPOINT_PATH \
       --data-path $DATA_PATH \
       --vocab-file data/gpt2-vocab.json \
       --merge-file data/gpt2-merges.txt \
       --data-impl mmap \
       --split 949,50,1 \
       --distributed-backend nccl \
       --lr 0.00015 \
       --lr-decay-style cosine \
       --min-lr 1.0e-5 \
       --weight-decay 1e-2 \
       --clip-grad 1.0 \
       --lr-warmup-fraction .01 \
       --checkpoint-activations \
       --log-interval 10 \
       --save-interval 500 \
       --eval-interval 100 \
       --eval-iters 10 \
       --fp16
```

步骤2 开始训练。

本文是单机单卡训练，使用预训练脚本参数控制：

```
GPUS_PER_NODE=1
NNODES=1
NODE_RANK=0
```

1. 执行以下命令，开始预训练。

   ```
   nohup sh ./pretrain_gpt2.sh &
   ```

   图 4-3 开始预训练

   ```
   root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
   root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed# nohup sh ./pretrain_gpt2.sh &
   [1] 855
   root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed# nohup: ignoring input and appending output to 'nohup.out'
   ```

2. 实时查看训练日志，监控程序。

   ```
   tail -f nohup.out
   ```

   如果显示如下信息， 表示模型训练完成。

图 4-4 模型训练完成

```
-----------------------------------------------------------------------------------------
valid loss at iteration 5000 | lm loss value: 4.149279E+00 | lm loss PPL: 6.338826E+01 |
-----------------------------------------------------------------------------------------
saving checkpoint at iteration    5000 to checkpoints/gpt2
  successfully saved checkpoint at iteration    5000 to checkpoints/gpt2
time (ms) | save-checkpoint: 4680.12
[after training is done] datetime: 2023-07-02 12:32:40
-----------------------------------------------------------------------------------------------------------
valid loss at the end of training for val data | lm loss value: 4.146571E+00 | lm loss PPL: 6.321684E+01 |
-----------------------------------------------------------------------------------------------------------
saving checkpoint at iteration    5000 to checkpoints/gpt2
  successfully saved checkpoint at iteration    5000 to checkpoints/gpt2
Evaluating iter 10/10
-----------------------------------------------------------------------------------------------------------
test loss at the end of training for test data | lm loss value: 4.076313E+00 | lm loss PPL: 5.892778E+01 |
-----------------------------------------------------------------------------------------------------------
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
```

在训练过程中观察单GPU卡的利用率，如下：

图 4-5 GPU 利用率

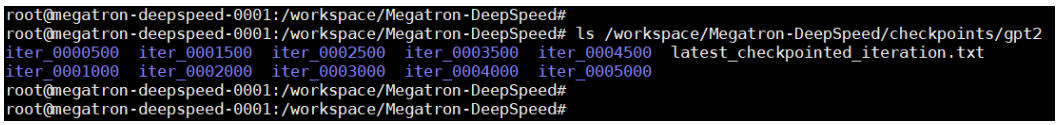
```
Every 2.0s: nvidia-smi

Sun Jul  2 19:26:05 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.105.01   Driver Version: 515.105.01   CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA [redacted]..  On  | 00000000:5B:00.0 Off |                    0 |
| N/A   53C    P0   383W / 400W |  11056MiB / 81920MiB |     97%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA [redacted]    On  | 00000000:5E:00.0 Off |                    0 |
| N/A   28C    P0    62W / 400W |      3MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA [redacted]    On  | 00000000:75:00.0 Off |                    0 |
| N/A   30C    P0    60W / 400W |      3MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA [redacted]  .  On  | 00000000:78:00.0 Off |                    0 |
| N/A   29C    P0    60W / 400W |      3MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   4  NVIDIA [redacted]  .  On  | 00000000:9D:00.0 Off |                    0 |
| N/A   30C    P0    59W / 400W |      3MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   5  NVIDIA [redacted]  .  On  | 00000000:A1:00.0 Off |                    0 |
| N/A   28C    P0    61W / 400W |      3MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   6  NVIDIA [redacted]    On  | 00000000:F5:00.0 Off |                    0 |
| N/A   29C    P0    58W / 400W |      3MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   7  NVIDIA [redacted]  .  On  | 00000000:F9:00.0 Off |                    0 |
| N/A   28C    P0    57W / 400W |      3MiB / 81920MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
```

步骤3 查看生成的模型checkpoint。

本示例生成的模型checkpoint路径设置在“/workspace/Megatron-DeepSpeed/checkpoints/gpt2”。

```
ll ./checkpoints/gpt2
```

图 4-6 模型 checkpoint



----结束

步骤 3 单机多卡训练

和单机单卡训练相比， 单机多卡训练只需在预训练脚本中设置多卡参数相关即可， 其余步骤与单机单卡相同。

步骤1 当前选择GPU裸金属服务器是8卡， 因此需要调整如下参数：

```
GPUS_PER_NODE=8
```

步骤2 调整全局批处理大小（global batch size）、微批处理大小（micro batch size）、数据并行大小（data_parallel_size）参数。三者的关系为：“global_batch_size”可被“micro_batch_size * data_parallel_size”整除。

本文设置的参数值如下：

```
global_batch_size = 64
micro_batch_size = 4
data_parallel_size = 8
```

步骤3 单机多卡完整的预训练脚本内容如下：

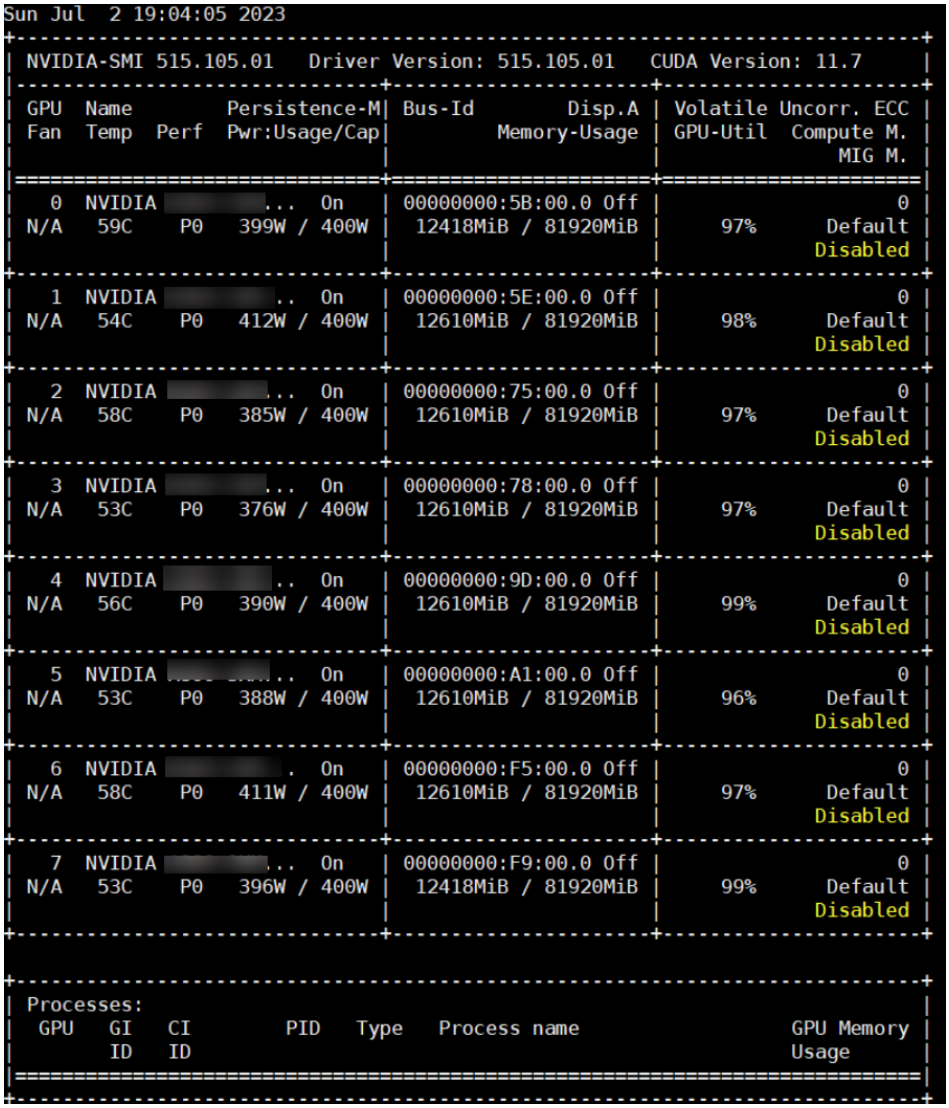
```
#! /bin/bash

# Runs the "345M" parameter model

GPUS_PER_NODE=8
# Change for multinode config
MASTER_ADDR=localhost
MASTER_PORT=6000
NNODES=1
NODE_RANK=0
WORLD_SIZE=$(($GPUS_PER_NODE*$NNODES))

DATA_PATH=data/meg-gpt2_text_document
CHECKPOINT_PATH=checkpoints/gpt2

DISTRIBUTED_ARGS="--nproc_per_node $GPUS_PER_NODE --nnodes $NNODES --node_rank $NODE_RANK
--master_addr $MASTER_ADDR --master_port $MASTER_PORT"

python -m torch.distributed.launch $DISTRIBUTED_ARGS \
       pretrain_gpt.py \
       --tensor-model-parallel-size 1 \
       --pipeline-model-parallel-size 1 \
       --num-layers 24 \
       --hidden-size 1024 \
       --num-attention-heads 16 \
       --micro-batch-size 4 \
       --global-batch-size 64 \
       --seq-length 1024 \
       --max-position-embeddings 1024 \
       --train-iters 5000 \
       --lr-decay-iters 320000 \
       --save $CHECKPOINT_PATH \
       --load $CHECKPOINT_PATH \
       --data-path $DATA_PATH \
       --vocab-file data/gpt2-vocab.json \
```



```
    --merge-file data/gpt2-merges.txt \
    --data-impl mmap \
    --split 949,50,1 \
    --distributed-backend nccl \
    --lr 0.00015 \
    --lr-decay-style cosine \
    --min-lr 1.0e-5 \
    --weight-decay 1e-2 \
    --clip-grad 1.0 \
    --lr-warmup-fraction .01 \
    --checkpoint-activations \
    --log-interval 10 \
    --save-interval 500 \
    --eval-interval 100 \
    --eval-iters 10 \
    --fp16
```

训练时监控的GPU利用率如下：

图 4-7 GPU 利用率



----结束

步骤 4 使用 GPT-2 模型生成文本

步骤1 自动式生成文本。

1. 执行以下命令，创建文本生成脚本。

```
vim generate_text.sh
```

增加内容如下：

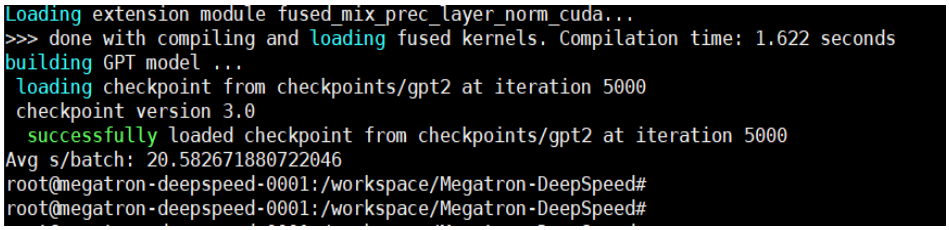
```
#!/bin/bash

CHECKPOINT_PATH=checkpoints/gpt2
VOCAB_FILE=data/gpt2-vocab.json
MERGE_FILE=data/gpt2-merges.txt

python tools/generate_samples_gpt.py \
       --tensor-model-parallel-size 1 \
       --num-layers 24 \
       --hidden-size 1024 \
       --load $CHECKPOINT_PATH \
       --num-attention-heads 16 \
       --max-position-embeddings 1024 \
       --tokenizer-type GPT2BPETokenizer \
       --fp16 \
       --micro-batch-size 2 \
       --seq-length 1024 \
       --out-seq-length 1024 \
       --temperature 1.0 \
       --vocab-file $VOCAB_FILE \
       --merge-file $MERGE_FILE \
       --genfile unconditional_samples.json \
       --num-samples 2 \
       --top_p 0.9 \
       --recompute
```

2. 执行以下脚本，生成文本。

```
sh ./generate_text.sh
```

若回显信息如下，则表示生成文本完成。

图 4-8 生成文本完成信息



3. 查看模型生成的文本文件。

```
cat unconditional_samples.json
```

回显信息如下：

图 4-9 文件信息

步骤2 开启交互式对话模式。

1. 执行以下命令，创建文本生成脚本。

   ```
   vim interactive_text.sh
   ```

   写入如下内容：

   ```
   #!/bin/bash

   CHECKPOINT_PATH=/workspace/Megatron-DeepSpeed/checkpoints/gpt2_345m
   VOCAB_FILE=/workspace/Megatron-DeepSpeed/data/gpt2-vocab.json
   MERGE_FILE=/workspace/Megatron-DeepSpeed/data/gpt2-merges.txt

   deepspeed /workspace/Megatron-DeepSpeed/tools/generate_samples_gpt.py \
       --tensor-model-parallel-size 1 \
       --num-layers 24 \
       --hidden-size 1024 \
       --load $CHECKPOINT_PATH \
       --num-attention-heads 16 \
       --max-position-embeddings 1024 \
       --tokenizer-type GPT2BPETokenizer \
       --fp16 \
       --micro-batch-size 2 \
       --seq-length 1024 \
       --out-seq-length 1024 \
       --temperature 1.0 \
       --vocab-file $VOCAB_FILE \
       --merge-file $MERGE_FILE \
       --genfile unconditional_samples.json \
       --num-samples 0 \
       --top_p 0.9 \
       --recompute
   ```

2. 执行以下脚本，开启交互式对话。

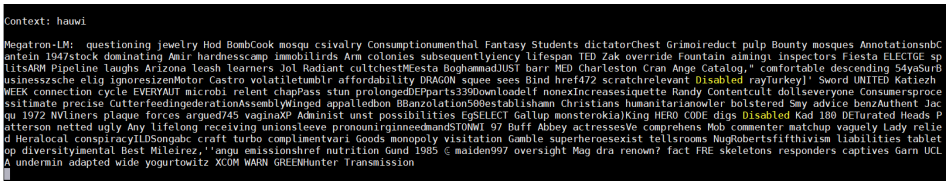
   ```
   bash interactive_text.sh
   ```

   回显信息如下，输入huawei并回车后生成内容：

   ```
   Context prompt (stop to exit) >>> huawei
   ```

   回车后自动输出相关文本， 输出内容与模型训练、数据集强相关，这里仅为示例。

图 4-10 模型输出文本信息



----结束

5 Lite Server 资源管理

5.1 查看 Lite Server 服务器详情

在您创建了Lite Server服务器后，可以通过管理控制台查看和管理您的Lite Server服务器。本节介绍如何查看Lite Server服务器的详细信息，包括名称/ID、磁盘、网卡、规格、镜像等信息。

表 5-1 详情页参数说明

参数名称	说明
名称	Lite Server服务器的名称。
规格	Lite Server服务器的规格。
ID	Lite Server服务器的ID，可用于在费用中心查询。
计费模式	Lite Server服务器当前的计费模式。
状态	Lite Server服务器的运行状态。
虚拟私有云	Lite Server服务器创建时绑定的虚拟私有云，单击链接可跳转到虚拟私有云详情页。
裸金属服务器	Lite Server服务器为一台裸金属服务器，单击链接可跳转至对应弹性裸金属服务器的详情页。
镜像	Lite Server服务器的镜像。
创建时间	Lite Server服务器的创建时间。
更新时间	Lite Server服务器的更新时间。
所属订单	Lite Server服务器对应的订单，单击链接可跳转至费用中心。

图 5-1 Lite Server 服务器详情

5.2 启动或停止 Lite Server 服务器

当您暂时不需要使用弹性节点Server的时候，可以通过对运行中的裸金属实例进行停止操作，停止对资源的消耗。当需要使用的时候，对于停止状态的弹性节点Server，可以通过启动操作重新使用弹性节点Server。

步骤1 登录ModelArts管理控制台。

步骤2 在左侧菜单栏中选择“AI专属资源池 > 弹性节点 Server”。

步骤3 执行如下操作，启动或停止弹性节点Server。

- 启动弹性节点Server：单击“启动”。只有处于“已停止/停止失败/启动失败”状态的弹性节点Server可以执行启动操作。
- 停止弹性节点Server：单击“停止”，在弹出的确认对话框中，确认信息无误，然后单击“确定”。只有处于“运行中/停止失败”状态的弹性节点Server可以执行停止操作。

说明

停止服务器为“强制关机”方式，会中断您的业务，请确保服务器上的文件已保存。

----结束

5.3 同步 Lite Server 服务器状态

Lite Server为一台弹性裸金属服务器，当用户在云服务器页面修改了裸金属服务器状态后，您可通过“同步”功能，同步其状态至ModelArts。

步骤1 登录ModelArts管理控制台。

步骤2 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“弹性节点 Server”列表页面。

步骤3 在弹性节点Server列表中，单击 ••• 的“同步”，在弹出的确认对话框中，确认信息无误，然后单击“确定”，完成同步操作。

----结束

5.4 切换 Lite Server 服务器操作系统

场景描述

Lite Server为一台弹性裸金属服务器，您可以使用BMS服务提供的切换操作系统功能，对Lite Server资源操作系统进行切换。本文介绍以下三种切换操作系统的方式：

- 在BMS控制台切换操作系统
- 使用BMS Go SDK的方式切换操作系统
- 使用Python封装API的方式切换操作系统

说明

切换操作系统需满足以下条件：

- 当前裸金属服务器状态为停止状态。
- 目标操作系统必须是该Region下的IMS公共镜像或者私有共享镜像。

在 BMS 控制台切换操作系统

步骤1 获取操作系统镜像。

由华为云官方提供给客户操作系统镜像，在IMS镜像服务的共享镜像处进行接收即可，参考如下图操作。

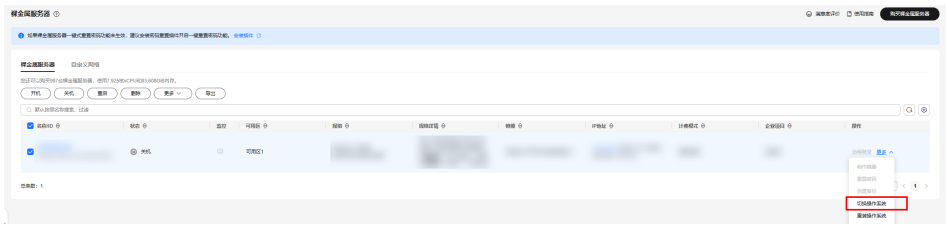
图 5-2 共享镜像

步骤2 切换操作系统。

对Lite Server资源对应的裸金属服务器，对其进行关机操作，完成关机后，才可以执行切换操作系统动作。

在裸金属服务的更多选项中，单击切换操作系统，如下图所示。

图 5-3 切换操作系统



在切换操作系统界面，选择上一步接收到的共享镜像即可。

----结束

使用 BMS Go SDK 的方式切换操作系统

以下为BMS使用Go语言通过SDK方式切换操作系统的示例代码。

```
package main

import (
    "fmt"
        "os"
    "github.com/huaweicloud/huaweicloud-sdk-go-v3/core/auth/basic"
    bms "github.com/huaweicloud/huaweicloud-sdk-go-v3/services/bms/v1"
```

```
    "github.com/huaweicloud/huaweicloud-sdk-go-v3/services/bms/v1/model"
    region "github.com/huaweicloud/huaweicloud-sdk-go-v3/services/bms/v1/region"
)

func main() {
    // 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
    // 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    ak := os.Getenv("HUAWEICLOUD_SDK_AK")
    sk := os.Getenv("HUAWEICLOUD_SDK_SK")

    auth := basic.NewCredentialsBuilder().
        WithAk(ak).
        WithSk(sk).
        Build()

    client := bms.NewBmsClient(
        bms.BmsClientBuilder().
            WithRegion(region.ValueOf("cn-north-4")).
            WithCredential(auth).
            Build())
    keyname := "KeyPair-name"
    userdata := "aGVsbG8gd29ybGQsIHdlbGNvbWUgdG8gam9pbiB0aGUgY29uZmVyZW5jZQ=="
    request := &model.ChangeBaremetalServerOsRequest{
        ServerId: "****input your bms instance id****",
        Body: &model.OsChangeReq{
            OsChange: &model.OsChange{
                Keyname: &keyname,
                Imageid: "****input your ims image id****",
                Metadata: &model.MetadataInstall{
                    UserData: &userdata,
                },
            },
        },
    }

    response, err := client.ChangeBaremetalServerOs(request)
    if err == nil {
        fmt.Printf("%+v\n", response)
    } else {
        fmt.Println(err)
    }
}
```

Python 封装 API 方式切换操作系统

以下为BMS使用Python语言通过API方式切换操作系统的示例代码。

```
# -*- coding: UTF-8 -*-

import requests
import json
import time
import requests.packages.urllib3.exceptions
from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable_warnings(InsecureRequestWarning)
class ServerOperation(object):

    ################################## IAM认证API##################################################

    def __init__(self, account, password, region_name, username=None, project_id=None):
        """
        :param username:  if IAM user,here is small user, else  big user
        :param account: account big  big user
        :param password: account
        :param region_name:
```



```
        """
        self.account = account
        self.username = username
        self.password = password
        self.region_name = region_name
        self.project_id = project_id
        self.ma_endpoint = "https://modelarts.{}.myhuaweicloud.com".format(region_name)
        self.service_endpoint = "https://bms.{}.myhuaweicloud.com".format(region_name)
        self.iam_endpoint = "https://iam.{}.myhuaweicloud.com".format(region_name)
        self.headers = {"Content-Type": "application/json",
                        "X-Auth-Token": self.get_project_token_by_account(self.iam_endpoint)}

    def get_project_token_by_account(self, iam_endpoint):
        body = {
            "auth": {
                "identity": {
                    "methods": [
                        "password"
                    ],
                    "password": {
                        "user": {
                            "name": self.username if self.username else self.account,
                            "password": self.password,
                            "domain": {
                                "name": self.account
                            }
                        }
                    }
                },
                "scope": {
                    "project": {
                        "name": self.region_name
                    }
                }
            }
        }
        headers = {
            "Content-Type": "application/json"
        }
        import json
        url = iam_endpoint + "/v3/auth/tokens"
        response = requests.post(url, headers=headers, data=json.dumps(body), verify=True)
        token = (response.headers['X-Subject-Token'])
        return token
    def change_os(self, server_id):
        url = "{}/v1/{}/baremetalservers/{}/changeos".format(self.service_endpoint, self.project_id, server_id)
        print(url)
        body = {
            "os-change": {
                "adminpass": "@Server",
                "imageid": "40d88eea-6e41-418a-ad6c-c177fe1876b8"
            }
        }
        response = requests.post(url, headers=self.headers, data=json.dumps(body), verify=False)
        print(json.dumps(response.json(), indent=1))
        return response.json()

if __name__ == '__main__':
    # 调用API前置准备，初始化认证鉴权信息
    server = ServerOperation(username="xxx",
                             account="xxx",
                             password="xxx",
                             project_id="xxx",
                             region_name="cn-north-4")

    server.change_os(server_id="0c84bb62-35bd-4e1c-ba08-a3a686bc5097")
```


5.5 监控 Lite Server 资源

5.5.1 使用 CES 监控 Lite Server 资源

场景描述

本文主要介绍如何配置华为云BMS+CES联合提供的裸金属服务器的指标监控方案，可帮助您查看CPU相关监控指标、CPU负载类相关监控指标、内存相关监控指标、磁盘相关监控指标、磁盘I/O类、文件系统类、网卡类、软RAID相关监控指标和进程相关监控指标。

裸金属服务器监控介绍

监控概述请参考**BMS官方文档**。除文档所列支持的镜像之外，目前还支持Ubuntu20.04。

监控指标采样周期1分钟。当前监控指标项已经包含CPU、内存、磁盘、网络。在主机上安装加速卡驱动后，可以自动采集的如下指标：

表 5-2 指标列表

指标英文名	指标中文名	说明	单位	维度
gpu_status	gpu健康状态。	BMS上GPU健康状态，是一个综合指标，0代表健康，1代表亚健康，2代表故障。	-	instance_id，gpu
gpu_utilization	gpu使用率。	该GPU的算力使用率。	%	instance_id，gpu
memory_utilization	显存使用率。	该GPU的显存使用率。	%	instance_id，gpu
gpu_performance	gpu性能状态。	该GPU的性能状态。	-	instance_id，gpu
encoder_utilization	编码使用率。	该GPU的编码能力使用率。	%	instance_id，gpu
decoder_utilization	解码使用率。	该GPU的解码能力使用率。	%	instance_id，gpu
volatile_correctable	短期可纠正ECC错误数量。	该GPU重置以来可纠正的ECC错误数量，每次重置后归0。	个	instance_id，gpu

指标英文名	指标中文名	说明	单位	维度
volatile_uncorrectable	短期不可纠正ECC错误数量。	该GPU重置以来不可纠正的ECC错误数量，每次重置后归0。	个	instance_id，gpu
aggregate_correctable	累计可纠正ECC错误数量。	该GPU累计的可纠正ECC错误数量。	个	instance_id，gpu
aggregate_uncorrectable	累计不可纠正ECC错误数量。	该GPU累计的不可纠正ECC错误数量。	个	instance_id，gpu
retired_page_single_bit	retired page single bit错误数量。	retired page single bit错误数量，表示当前卡隔离的单比特页数。	个	instance_id，gpu
retired_page_double_bit	retired page double bit错误数量。	retired page double bit错误数量，表示当前卡隔离的双比特页的数量。	个	instance_id，gpu

监控插件安装步骤

步骤1 当前账户需要给CES授权委托，请参考**创建用户并授权使用云监控服务**。

步骤2 当前还不支持在CES界面直接一键安装监控，需要登录到服务器上执行以下命令安装配置Agent。其它region的安装请参考**单台主机下安装Agent**。

```
cd /usr/local && curl -k -O https://obs.cn-north-4.myhuaweicloud.com/uniagent-cn-north-4/script/agent_install.sh && bash agent_install.sh
```

安装成功的标志如下：

图 5-4 安装成功提示

```
telescope/linux_arm64_bin/
telescope/linux_arm64_bin/uninstall_not_root.sh
telescope/linux_arm64_bin/telescope
telescope/linux_arm64_bin/install.sh
telescope/linux_arm64_bin/install_not_root.sh
telescope/linux_arm64_bin/telescoped
telescope/linux_arm64_bin/uninstall.sh
telescope/linux_arm64_bin/tools/
telescope/linux_arm64_bin/tools/hioadm
telescope/linux_arm64_bin/tools/nvme
telescope/linux_arm64_bin/tools/storcli64
telescope/linux_arm64_bin/tools/sas3ircu
telescope/manifest.json
telescope/telescope-2.4.8-release.json
telescope/linux_amd64_bin/
telescope/linux_amd64_bin/uninstall_not_root.sh
telescope/linux_amd64_bin/telescope
telescope/linux_amd64_bin/install.sh
telescope/linux_amd64_bin/install_not_root.sh
telescope/linux_amd64_bin/telescoped
telescope/linux_amd64_bin/uninstall.sh
telescope/linux_amd64_bin/tools/
telescope/linux_amd64_bin/tools/hioadm
telescope/linux_amd64_bin/tools/nvme
telescope/linux_amd64_bin/tools/storcli64
telescope/linux_amd64_bin/tools/sas3ircu
telescope/conf/
telescope/conf/custom_conf.json
telescope/windows_bin/
telescope/windows_bin/uninstall.bat
telescope/windows_bin/shutdown.bat
telescope/windows_bin/telescope.exe
telescope/windows_bin/install.bat
telescope/windows_bin/start.bat
telescope/windows_bin/getpid.bat
telescope/config/
telescope/config/conf_ces.json
telescope/config/logs_config.xml
telescope/config/conf.json
Current user is root.
Start to install telescope...
instance_type: physical.p7vs.8xlarge.ei
Starting telescope...
Telescope process starts successfully.
```

步骤3 在CES界面查看具体的监控项，加速卡类的监控项必须在主机安装加速卡驱动后才会有相关指标。

图 5-5 监控界面

至此，监控插件已经安装完成，相关指标的采集可以在UI界面直接查看或者根据指标值配置相关告警。

----结束

5.5.2 使用 DCGM 监控 Lite Server 资源

场景描述

本文主要介绍如何配置DCGM监控。DCGM是用于管理和监控基于Linux系统的NVIDIA GPU大规模集群的一体化工具，提供多种能力，包括主动健康监控、诊断、系统验证、策略、电源和时钟管理、配置管理和审计等。

前提条件

裸金属服务器需要安装driver、cuda、fabric-manager软件包。

1、安装 Docker

使用Docker官方脚本安装最新版Docker：

```
curl https://get.docker.com | sh
sudo systemctl --now enable docker
```

2、安装 NVIDIA 容器工具集

设置仓库地址和GPG key：

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - \
   && curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee /etc/apt/
sources.list.d/nvidia-docker.list
```

安装nvidia-docker2：

```
sudo apt-get update \
   && sudo apt-get install -y nvidia-docker2
```

编辑/etc/docker/daemon.json，改为如下内容：

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

重启Docker daemon：

```
sudo systemctl restart docker
```

3、运行 DCGM-Exporter

以Docker方式运行DCGM-Exporter：

```
DCGM_EXPORTER_VERSION=3.1.7-3.1.4 && \
docker run -d --rm \
  --gpus all \
  --net host \
  --cap-add SYS_ADMIN \
  nvcr.io/nvidia/k8s/dcgm-exporter:${DCGM_EXPORTER_VERSION}-ubuntu20.04 \
  -f /etc/dcgm-exporter/dcp-metrics-included.csv
```

说明

这里使用的是DCGM-Exporter默认的指标采集配置文件/etc/dcgm-exporter/dcp-metrics-included.csv，指标采集对象详见**dcgm-exporter**。如果采集对象不能满足要求，可通过定制镜像或挂载的方式使用自定义配置。

等待约1分钟，执行下面的命令获取GPU指标：

```
curl localhost:9400/metrics
```

指标获取结果如下：

```
# HELP DCGM_FI_DEV_SM_CLOCK SM clock frequency (in MHz).
# TYPE DCGM_FI_DEV_SM_CLOCK gauge
# HELP DCGM_FI_DEV_MEM_CLOCK Memory clock frequency (in MHz).
# TYPE DCGM_FI_DEV_MEM_CLOCK gauge
# HELP DCGM_FI_DEV_MEMORY_TEMP Memory temperature (in C).
# TYPE DCGM_FI_DEV_MEMORY_TEMP gauge
...
DCGM_FI_DEV_SM_CLOCK{gpu="0", UUID="GPU-6ad7ea4c-5517-05e7-0b54-7554cb4374d3"} 1
DCGM_FI_DEV_MEM_CLOCK{gpu="0", UUID="GPU-6ad7ea4c-5517-05e7-0b54-7554cb4374d3"} 4
DCGM_FI_DEV_MEMORY_TEMP{gpu="0", UUID="GPU-6ad7ea4c-5517-05e7-0b54-7554cb4374d3的"}
9223372036854578794
...
```

4、安装 Prometheus

在“/usr/local/prometheus”目录创建配置文件prometheus.yml内容如下：

```
global:
  scrape_interval: 15s # 采集间隔
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['xx.xx.xx.xx:9400'] # DCGM-Exporter指标获取端口，替换xx.xx.xx.xx为DCGM-Exporter所在节点的
IP地址
```

运行Prometheus：

```
docker run -d \
    -p 9090:9090 \
    -v /usr/local/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml \
    prom/prometheus
```

说明

这里使用的是Prometheus最基本的功能，如有更高级的诉求，可参考**prometheus的官方文档**。

5、安装 Grafana

运行社区最新发行的Grafana版本：

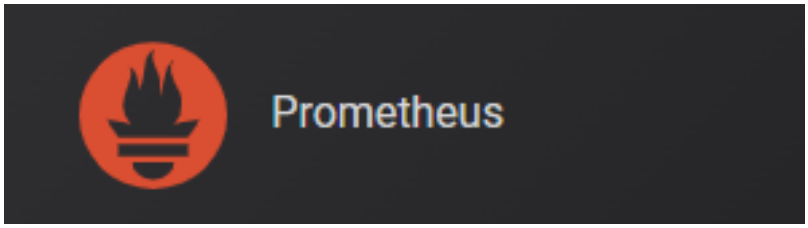
```
docker run -d -p 3000:3000 grafana/grafana-oss
```

在BMS页面打开Grafana所在节点的安全组配置，添加入方向规则，允许外部访问3000、9090端口：

在浏览器地址栏输入xx.xx.xx.xx:3000，登录Grafana，默认账号密码为：admin/admin。在配置管理页面，添加数据源，类型选择Prometheus。

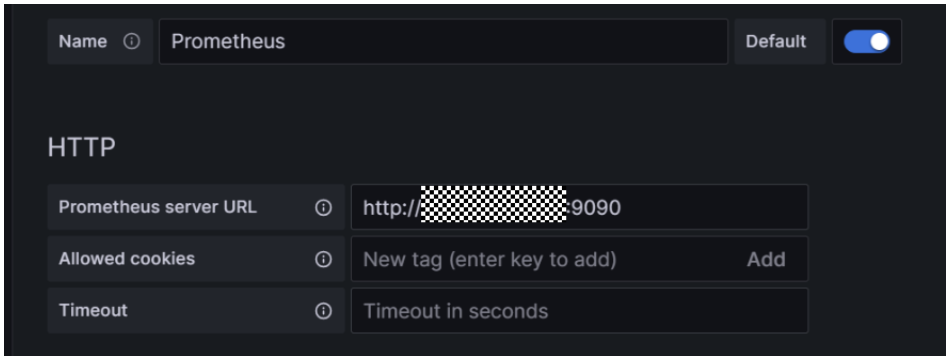
备注：xx.xx.xx.xx为Grafana的所在宿主机的IP地址

图 5-6 Prometheus



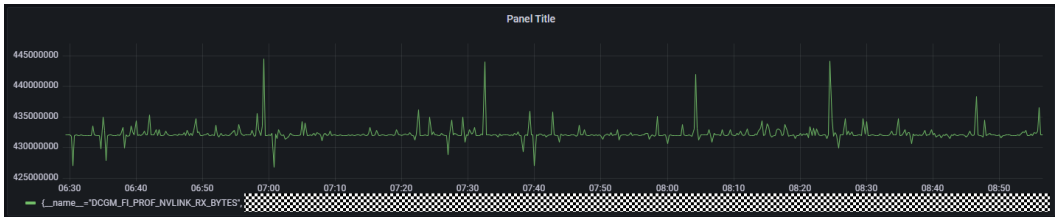
在HTTP的URL输入框中输入Prometheus的IP地址和端口号，单击Save&Test：

图 5-7 IP 地址和端口号



至此，指标监控方案安装完成。指标监控效果展示如下：

图 5-8 指标监控效果



说明

这里使用的是Grafana最基本的功能，如有更高级的诉求，可参考**Grafana的官方文档**。

5.6 NPU 日志收集上传

场景描述

当NPU出现故障，您可通过本方案收集NPU的日志信息。本方案中生成的日志会保存在节点上，并自动上传至华为云技术支持提供的OBS桶中，日志仅用于问题定位分析，因此需要您提供AK/SK给华为云技术支持，用于授权认证。

约束限制

当前仅支持在贵阳一、乌兰察布一使用该功能。

操作步骤

步骤1 获取AK/SK。该AK/SK用于后续脚本配置，做认证授权。

如果已生成过AK/SK，则可跳过此步骤，找到原来已下载的AK/SK文件，文件名一般为：credentials.csv。

如下图所示，文件包含了租户名（User Name），AK（Access Key Id），SK（Secret Access Key）。

图 5-9 credential.csv 文件内容

	A	B	C
1	User Name	Access Key Id	Secret Access Key
2	hu[redacted]dg	QTWA[redacted]UT2QVKYUC	MFyfvK41ba2[redacted]npdUKGpownRZlmVmHc

AK/SK生成步骤：

1. 登录管理控制台。
2. 单击右上角的用户名，在下拉列表中单击“我的凭证”。
3. 单击“访问密钥”。
4. 单击“新增访问密钥”。
5. 下载密钥，并妥善保管。

步骤2 准备租户名ID和IAM用户名ID，用于OBS桶配置。

将您的租户名ID和IAM用户名ID提供给华为技术支持，华为云技术支持将根据您提供的信息，为您配置OBS桶策略，以便用户收集的日志可以上传至对应的OBS桶。

华为云技术支持配置完成后，会给您提供对应的OBS桶目录“obs_dir”，该目录用于后续配置的脚本中。

图 5-10 租户名 ID 和 IAM 用户名 ID

步骤3 准备日志收集上传脚本。

修改以下脚本中NpuLogCollection的参数，将ak、sk、obs_dir替换为前面步骤中获取到的值，然后把该脚本上传到要收集NPU日志的节点上。

```
import json
import os
import sys
import hashlib
import hmac
import binascii
from datetime import datetime


class NpuLogCollection(object):
    NPU_LOG_PATH = "/var/log/npu_log_collect"
    SUPPORT_REGIONS = ['cn-southwest-2', 'cn-north-9']
    OPENSTACK_METADATA = "http://169.254.169.254/openstack/latest/meta_data.json"
    OBS_BUCKET_PREFIX = "npu-log-"

    def __init__(self, ak, sk, obs_dir):
        self.ak = ak
        self.sk = sk
        self.obs_dir = obs_dir
        self.region_id = self.get_region_id()

    def get_region_id(self):
        meta_data = os.popen("curl {}".format(self.OPENSTACK_METADATA))
        json_meta_data = json.loads(meta_data.read())
        meta_data.close()
        region_id = json_meta_data["region_id"]
        if region_id not in self.SUPPORT_REGIONS:
            print("current region {} is not support.".format(region_id))
            raise Exception('region exception')
        return region_id

    def gen_collect_npu_log_shell(self):
        collect_npu_log_shell = "# !/bin/sh\n" \
                                "rm -rf {npu_log_path}\n" \
                                "mkdir -p {npu_log_path}\n" \
                                "echo {echo_npu_driver_info}\n" \
                                "npu-smi info > {npu_log_path}/npu-smi_info.log\n" \
                                "cat /usr/local/Ascend/driver/version.info > {npu_log_path}/npu-smi_driver-version.log
\n" \
                                "/usr/local/Ascend/driver/tools/upgrade-tool --device_index -1 --component -1 --
version > {npu_log_path}/npu-smi_firmware-version.log\n" \
                                "for i in $(seq 0 7) ; do npu-smi info -t health -i $i -c 0 >> {npu_log_path}/npu-
smi_health-code.log;done;\n" \
```

```
                            "for i in $(seq 0 7);do hccn_tool -i $i -net_health -g >> {npu_log_path}/npu-smi_net-
health.log ;done\n" \
                            "for i in $(seq 0 7);do hccn_tool -i $i -link -g >> {npu_log_path}/npu-smi_link.log ;done
\n" \
                            "for i in $(seq 0 7);do hccn_tool -i $i -tls -g |grep switch >> {npu_log_path}/npu-
smi_switch.log;done\n" \
                            "for i in $(seq 0 7);do npu-smi info -t board -i $i >> {npu_log_path}/npu-
smi_board.log; done;\n" \
                            "echo {echo_npu_ecc_info}\n" \
                            "for i in $(seq 0 7);do npu-smi info -t ecc -i $i >> {npu_log_path}/npu-smi_ecc.log;
done;\n" \
                            "for i in $(seq 0 7);do hccn_tool -i $i -optical -g | grep prese >> {npu_log_path}/npu-
smi_present.log ;done\n" \
                            "lspci | grep acce > {npu_log_path}/Device-info.log\n" \
                            "echo {echo_npu_device_log}\n" \
                            "cd {npu_log_path} && msnpureport -f > /dev/null\n" \
                            "tar -czvPf {npu_log_path}/log_messages.tar.gz /var/log/message*  > /dev/null\n" \
                            "tar -czvPf {npu_log_path}/ascend_install.tar.gz /var/log/ascend_seclog/*  > /dev/null
\n" \
                            "echo {echo_npu_tools_log}\n" \
                            "tar -czvPf {npu_log_path}/ascend_toollog.tar.gz /var/log/nputools_LOG_*  > /dev/
null" \
                .format(npu_log_path=self.NPU_LOG_PATH,
                        echo_npu_driver_info="collect npu driver info.",
                        echo_npu_ecc_info="collect npu ecc info.",
                        echo_npu_device_log="collect npu device log.",
                        echo_npu_tools_log="collect npu tools log.")
            return collect_npu_log_shell

        def collect_npu_log(self):
            print("begin to collect npu log")
            os.system(self.gen_collect_npu_log_shell())
            date_collect = datetime.now().strftime('%Y%m%d%H%M%S')
            instance_ip_obj = os.popen("curl http://169.254.169.254/latest/meta-data/local-ipv4")
            instance_ip = instance_ip_obj.read()
            instance_ip_obj.close()
            log_tar = "%s-npu-log-%s.tar.gz" % (instance_ip, date_collect)
            os.system("tar -czvPf %s %s > /dev/null" % (log_tar, self.NPU_LOG_PATH))
            print("success to collect npu log with {}".format(log_tar))
            return log_tar

        def upload_log_to_obs(self, log_tar):
            obs_bucket = "{}{}".format(self.OBS_BUCKET_PREFIX, self.region_id)
            print("begin to upload {} to obs bucket {}".format(log_tar, obs_bucket))
            obs_url = "https://%s.obs.%s.myhuaweicloud.com/%s/%s" % (obs_bucket, self.region_id, self.obs_dir,
log_tar)
            date = datetime.utcnow().strftime('%a, %d %b %Y %H:%M:%S GMT')
            canonicalized_headers = "x-obs-acl:public-read"
            obs_sign = self.gen_obs_sign(date, canonicalized_headers, obs_bucket, log_tar)

            auth = "OBS " + self.ak + ":" + obs_sign
            header_date = '\"' + "Date:" + date + '\"'
            header_auth = '\"' + "Authorization:" + auth + '\"'
            header_obs_acl = '\"' + canonicalized_headers + '\"'

            cmd = "curl -X PUT -T " + log_tar + " " + obs_url + " -H " + header_date + " -H " + header_auth + " -H
" + header_obs_acl
            os.system(cmd)
            print("success to upload {} to obs bucket {}".format(log_tar, obs_bucket))

        #  calculate obs auth sign
        def gen_obs_sign(self, date, canonicalized_headers, obs_bucket, log_tar):
            http_method = "PUT"
            canonicalized_resource = "/%s/%s/%s" % (obs_bucket, self.obs_dir, log_tar)
            IS_PYTHON2 = sys.version_info.major == 2 or sys.version < '3'
            canonical_string = http_method + "\n" + "\n" + "\n" + date + "\n" + canonicalized_headers + "\n" +
canonicalized_resource
            if IS_PYTHON2:
                hashed = hmac.new(self.sk, canonical_string, hashlib.sha1)
```

```
            obs_sign = binascii.b2a_base64(hashed.digest())[:-1]
        else:
            hashed = hmac.new(self.sk.encode('UTF-8'), canonical_string.encode('UTF-8'), hashlib.sha1)
            obs_sign = binascii.b2a_base64(hashed.digest())[:-1].decode('UTF-8')
        return obs_sign

    def execute(self):
        log_tar = self.collect_npu_log()
        self.upload_log_to_obs(log_tar)


if __name__ == '__main__':
    npu_log_collection = NpuLogCollection(ak='ak',
                                          sk='sk',
                                          obs_dir='obs_dir')
    npu_log_collection.execute()
```

步骤4 执行脚本收集日志。

在节点上执行该脚本，可以看到有如下输出，代表日志收集完成并成功上传至OBS。

图 5-11 日志收集完成

```
root@ :~# python npu-log-collection.py
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1778  100  1778    0     0  21682      0 --:--:-- --:--:-- --:--:-- 21682
begin to collect npu log
collect npu driver info.
collect npu ecc info.
collect npu device log.
collect npu tools log.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    10  100    10    0     0    526      0 --:--:-- --:--:-- --:--:--   526
success to collect npu log with 10.0.0.209-npu-log-20240809164759.tar.gz
begin to upload 10.0.0.209-npu-log-20240809164759.tar.gz to obs bucket npu-log-cn-north-9
success to upload 10.0.0.209-npu-log-20240809164759.tar.gz to obs bucket npu-log-cn-north-9
```

步骤5 查看在脚本的同级目录下，可以看到收集到的日志压缩包。

图 5-12 查看结果

```
root@ :~# ls
10.0.0.209-npu-log-20240809164759.tar.gz  npu-log-collection.py
```

----结束

5.7 释放 Lite Server 资源

针对不再使用的Lite Server资源，可以删除/退订以释放资源。停止计费相关介绍请见**停止计费**。

删除“按需计费”的 Lite Server 资源

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“弹性节点 Server”列表页面。
3. 在列表中，单击 ••• 的“删除”，在弹出的确认对话框中，确认信息无误，然后输入“ DELETE”，单击“确定”，完成删除操作。

退订“包年/包月”的 Lite Server 资源

您可通过以下方式进行退订：

- 方式一：在ModelArts界面退订（单个实例资源退订）
- 方式二：在费用中心退订（单个/批量实例资源退订）

在ModelArts界面退订

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“弹性节点 Server”列表页面。
3. 顶部下拉选择“查看所有”按钮，查看所有Server实例。

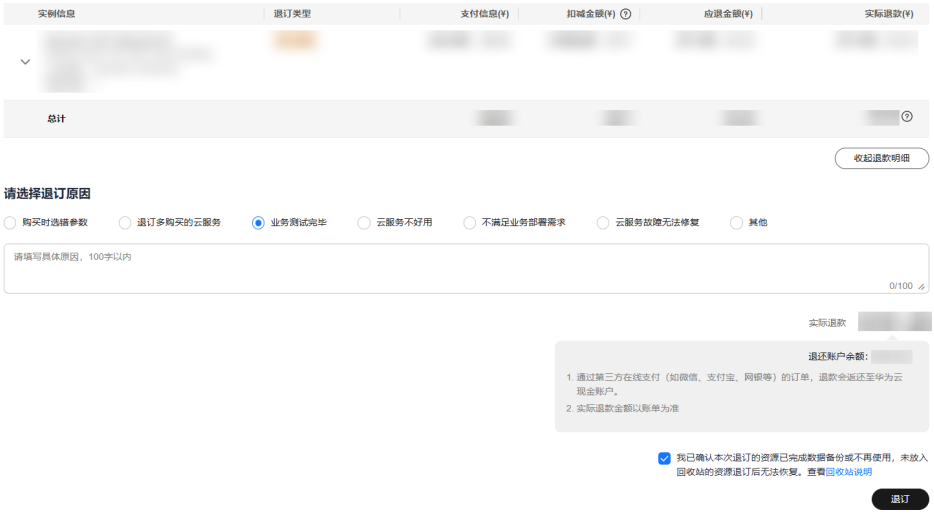
图 5-13 查看所有

说明

此时如果显示需要配置委托，请联系您的账号管理员为您配置委托权限，详细操作参考**配置ModelArts委托**。

4. 在弹性裸金属列表中，单击 ••• 的“退订”，跳转至“退订资源”页面。
5. 根据界面提示，确认需要退订的资源，并选择退订原因。

图 5-14 退订资源



6. 确认退订信息无误后，勾选“我已确认……”和“资源退订后……”提示信息。
7. 单击“退订”，再次根据界面信息确认要退订的资源。
8. 再次单击“退订”，完成包年/包月资源的退订操作。

在费用中心退订单个实例资源

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“弹性节点 Server”列表页面。
3. 顶部下拉选择“查看所有”按钮，查看所有Server实例。

图 5-15 查看所有

说明

此时如果显示需要配置委托，请联系您的账号管理员为您配置委托权限，详细操作参考**配置ModelArts委托**。

4. 复制需要退订的实例ID。

图 5-16 复制实例 ID

说明

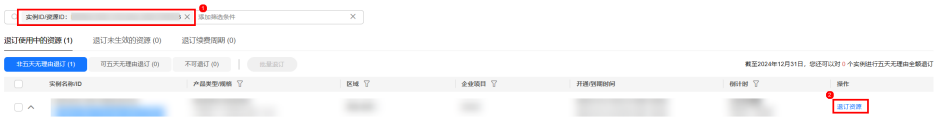
Server购买订单里绑定的资源ID为Server ID，与Server产品所封装的BMS/ECS ID不同，若要退订Server，需要在ModelArts控制台的“AI专属资源池 > 弹性节点 Server”中查询对应ID。

5. 单击顶部“费用”，进入费用中心，单击“订单管理 > 退订与退换货”。

图 5-17 退订与退换货

6. 在搜索框实例ID信息，确认信息无误后，单击右侧“退订资源”。

图 5-18 搜索实例 ID



7. 根据界面提示，确认需要退订的资源，并选择退订原因。
8. 确认退订信息无误后，勾选“我已确认……”和“资源退订后……”提示信息。
9. 单击“退订”，再次根据界面信息确认要退订的资源。
10. 再次单击“退订”，完成包年/包月资源的退订操作。

- **在费用中心批量退订实例资源**

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“专属资源池 > 弹性裸金属”，进入“弹性裸金属”列表页面。
3. 记录需要退订实例的ID。

说明

此时如果显示需要配置委托，请联系您的账号管理员为您配置委托权限，详细操作参考**配置ModelArts委托**。

4. 单击顶部“费用”，进入费用中心，单击“订单管理 > 退订与退换货”。

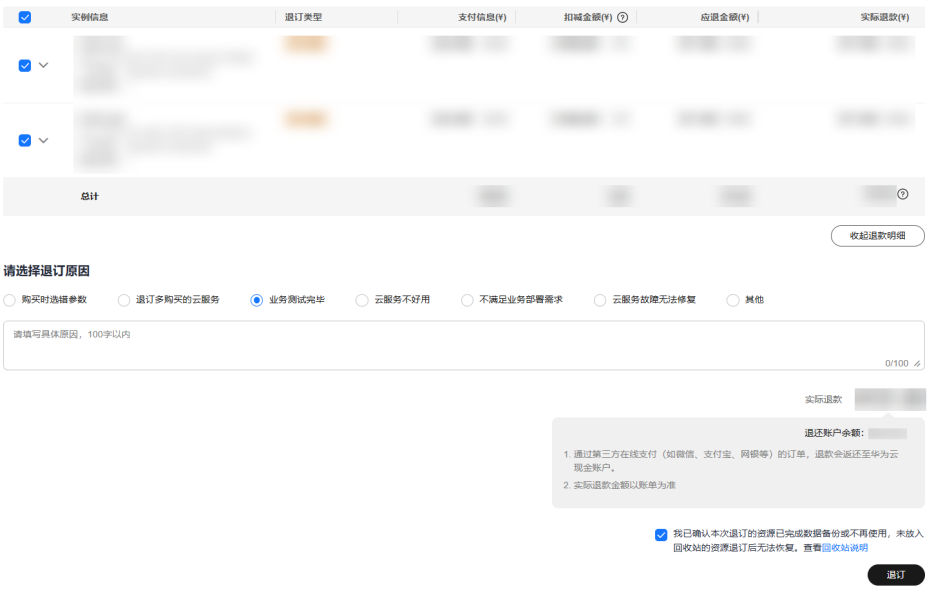
图 5-19 退订与退换货

5. 勾选需要退订的多个实例，单击“批量退订”。

图 5-20 批量退订

6. 根据界面提示，确认需要退订的资源，并选择退订原因。

图 5-21 退订资源



7. 确认退订信息无误后，勾选“我已确认……”和“资源退订后……”提示信息。
8. 单击“退订”，再次根据界面信息确认要退订的资源。
9. 再次单击“退订”，完成包年/包月资源的退订操作。