ModelArts

Lite 用户指南

文档版本 01

发布日期 2024-08-27

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/

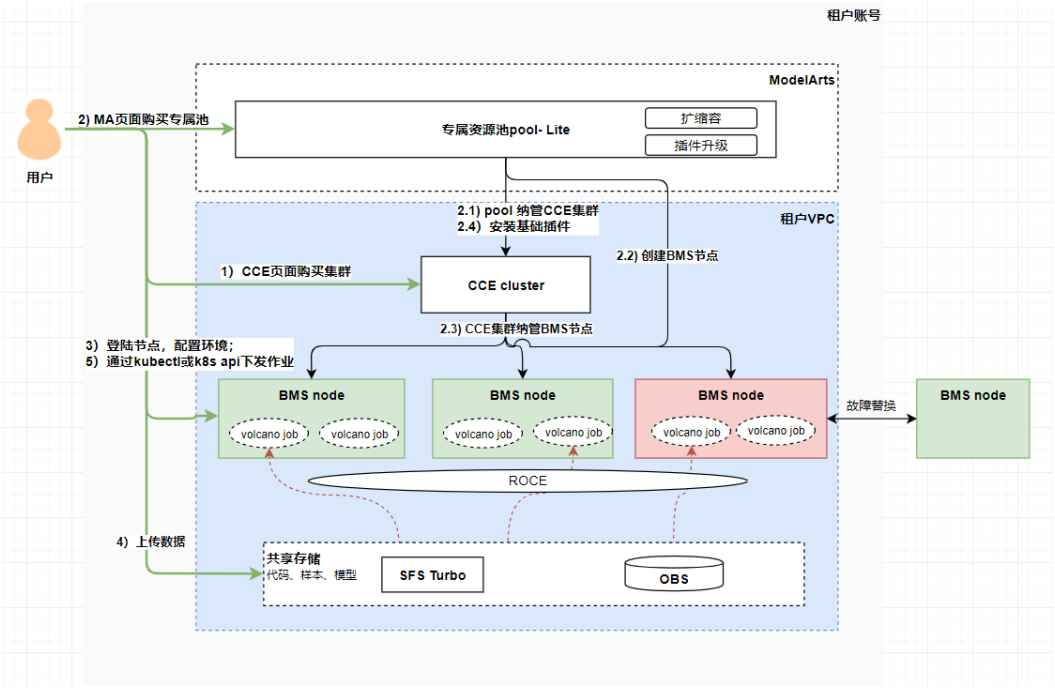
目 录

1 弹性集群 Lite Cluster

1.1 Lite Cluster 简介

ModelArts Lite Cluster面向k8s资源型用户，提供托管式k8s集群，并预装主流AI开发插件以及自研的加速插件，以云原生方式直接向用户提供AI Native的资源、任务等能力，用户可以直接操作资源池中的节点和k8s集群。

图 1-1 资源池架构图



1.2 k8s Cluster 资源开通

1.2.1 不同机型的对应的软件配套版本

由于弹性集群资源池可选择弹性裸金属或弹性云服务器作为节点资源，不同机型的节点对应的操作系统、适用的CCE集群版本等不相同，为了便于您制作镜像、升级软件等操作，本文对不同机型对应的软件配套版本做了详细介绍。

裸金属服务器的对应的软件配套版本

表 1-1 裸金属服务器

类型	卡类型	RDMA网络协议	操作系统	适用范围、约束	依赖插件
NPU	ascend-snt9b	RoCE	• 操作系统：EulerOS 2.10 64bit • 内核版本：4.19.90-vhulk2211.3.0.h1543.eulerosv2r10.aarch64 • 架构类型：aarch64	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC • 集群转发模式：iptables|ipvs	• huawei-npu • npu-driver • volcano 插件版本匹配关系请见**表1-3**。
		RoCE	• 操作系统：Huawei Cloud EulerOS 2.0 64bit • 内核版本：5.10.0-60.18.0.50.r865_35.hce2.aarch64 • 架构类型：aarch64	• 集群类型：CCE Turbo • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：ENI • 集群转发模式：iptables|ipvs	

类型	卡类型	RDMA网络协议	操作系统	适用范围、约束	依赖插件
		RoCE	• 操作系统：EulerOS 2.8 64bit • 内核版本：4.19.36-vhulk1907.1.0.h619.eulerosv2r8.aarch64 • 架构类型：aarch64	• 集群类型：CCE Standard|Turbo • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC|ENI • 集群转发模式：iptables|ipvs	
GPU	gp-ant8	RoCE	• 操作系统：EulerOS 2.10 64bit • 内核版本：4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64 • 架构类型：x86	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC 分布式训练时仅支持容器隧道网络 • 集群转发模式：iptables|ipvs	• gpu-beta • gpu-driver • rdma-sriov-dev-plugin 插件版本匹配关系请见**表1-3**。
	gp-ant1	RoCE	• 操作系统：EulerOS 2.10 64bit • 4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64 • 架构类型：x86	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC 分布式训练时仅支持容器隧道网络 • 集群转发模式：iptables|ipvs	

类型	卡类型	RDMA网络协议	操作系统	适用范围、约束	依赖插件
	gp-vnt1	RoCE|IB	• 操作系统：EulerOS 2.9 64bit（仅上海一p6|p6s规格使用） • 内核版本：147.5.1.6.h1099.eulerosv2r9.x86_64 • 架构类型：x86 • 操作系统：EulerOS 2.9 64bit（推荐） • 内核版本：4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64 • 架构类型：x86	• 集群类型：CCE Standard| • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC 分布式训练时仅支持容器隧道网络 • 集群转发模式：iptables|ipvs	

- RDMA：Remote Direct Memory Access（RDMA）是一种直接内存访问技术，将数据直接从一台计算机的内存传输到另一台计算机。
- RoCE：RDMA over Converged Ethernet（RoCE）是一种网络协议，允许应用通过以太网实现远程内存访问。
- IB：InfiniBand (IB) 是一种高性能计算机网络通信协议，专为高性能计算和数据中心互连设计。

弹性云服务器的对应的软件配套版本

表 1-2 弹性云服务器

类型	卡类型	操作系统	适用范围	依赖插件
NPU	ascend-snt3p-300i	• 操作系统：EulerOS 2.9 • 架构类型：x86	• 集群类型：CCE Standard、CCE Turbo • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC|ENI • 集群转发模式：iptables|ipvs	• huawei-npu • npu-driver • volcano 插件版本匹配关系请见**表1-3**。
	ascend-snt3	• 操作系统：EulerOS 2.5 • 架构类型：x86	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC • 集群转发模式：iptables|ipvs	
		• 操作系统：EulerOS 2.8 • 架构类型：arm	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC 集群转发模式：iptables|ipvs	

类型	卡类型	操作系统	适用范围	依赖插件
GPU	gp-vnt1	• 操作系统：EulerOS 2.9 • 架构类型：x86	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC • 集群转发模式：iptables|ipvs	• gpu-beta • gpu-driver • rdma-sriov-dev-plugin 插件版本匹配关系请见**表1-3**。
	gp-ant03	• 操作系统：EulerOS 2.9 • 架构类型：x86	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC • 集群转发模式：iptables|ipvs	
	gp-ant1-pcie40	• 操作系统：EulerOS 2.9 • 架构类型：x86	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC • 集群转发模式：iptables|ipvs	
	gp-tnt004	• 操作系统：EulerOS 2.9 • 架构类型：x86	• 集群类型：CCE Standard • 集群版本：v1.23|v1.25 • 集群规模：50|200|1000|2000 • 集群网络模式：容器隧道网络|VPC • 集群转发模式：iptables|ipvs	

插件版本与 CCE 集群版本适配关系

表 1-3 插件版本与 CCE 集群版本适配关系

类别	插件名称	插件版本	适配CCE集群版本	适用范围、约束	插件功能描述
ccePlugin	gpu-beta	2.0.48（仅适用于北京四、贵阳一、广州一、上海一、乌兰察布一、北京二、乌兰察布汽车，其余region暂不支持。）	v1.(23|25).*	GPU	支持在容器中使用GPU显卡的设备管理插件。
		1.2.15	v1.23.*		
	huawei-npu	1.2.5	v1.(23|25).*	NPU	支持容器里使用huawei NPU设备的管理插件。
	volcano	1.11.9	v1.(23|25).*	NPU	基于Kubernetes的批处理平台。
npuDriver	npu-driver	7.1.0.7.220-23.0.5（推荐）	无约束	NPU	用于升级、回滚npu驱动。
helm	rdma-sriov-dev-plugin	0.1.0	无约束	BMS、RDMA且非ascend-1980	用于支持容器里使用RDMA网卡。
	memarts	3.23.6-r002	无约束	无约束	近计算侧分布式缓存插件，用于存储加速。
	os-node-agent	6.2.0-20230925213538	无约束	无约束	OS插件，用于故障检测。
icAgent	icagent	default	CCE默认安装当前适配版本	无约束	CCE基础组件，用于日志和监控。

类别	插件名称	插件版本	适配CCE集群版本	适用范围、约束	插件功能描述
gpuDriver	gpu-driver	gpu-driver与系统内核版本有关，请见表1-4。			用于升级、回滚gpu驱动，插件依赖gpu-beta版本。

系统内核与 gpu-driver 配套关系

表 1-4 系统内核与 gpu-driver 配套关系

镜像版本	系统内核版本	适配CCE	gpu-driver版本
EulerOS 2.10	4.18.0-147.5.2.15.h1109.eulerosv2r10.x86_64	v1.(23|25|27|28).* 容器隧道网络|VPC|ENI	470.57.02
	4.18.0-147.5.2.5.h805.eulerosv2r10.x86_64	v1.(23|25|27).* 容器隧道网络|VPC|ENI	470.57.02
EulerOS 2.9	4.18.0-147.5.1.6.h841.eulerosv2r9.x86_64	v1.(23|25|27|28).* 容器隧道网络|VPC	470.57.02
EulerOS 2.3	3.10.0-514.44.5.10.h193.x86_64	v1.(23|25).* 容器隧道网络|VPC	470.57.02
	3.10.0-514.44.5.10.h254.x86_64	v1.(23|25).* 容器隧道网络|VPC	470.57.02

1.2.2 Lite Cluster 资源开通

说明

当前仅限贵阳一区域支持直接购买。

集群资源开通流程

开通集群资源过程中用户侧需要完成的任务流程如下图所示。

图 1-2 用户侧任务流程

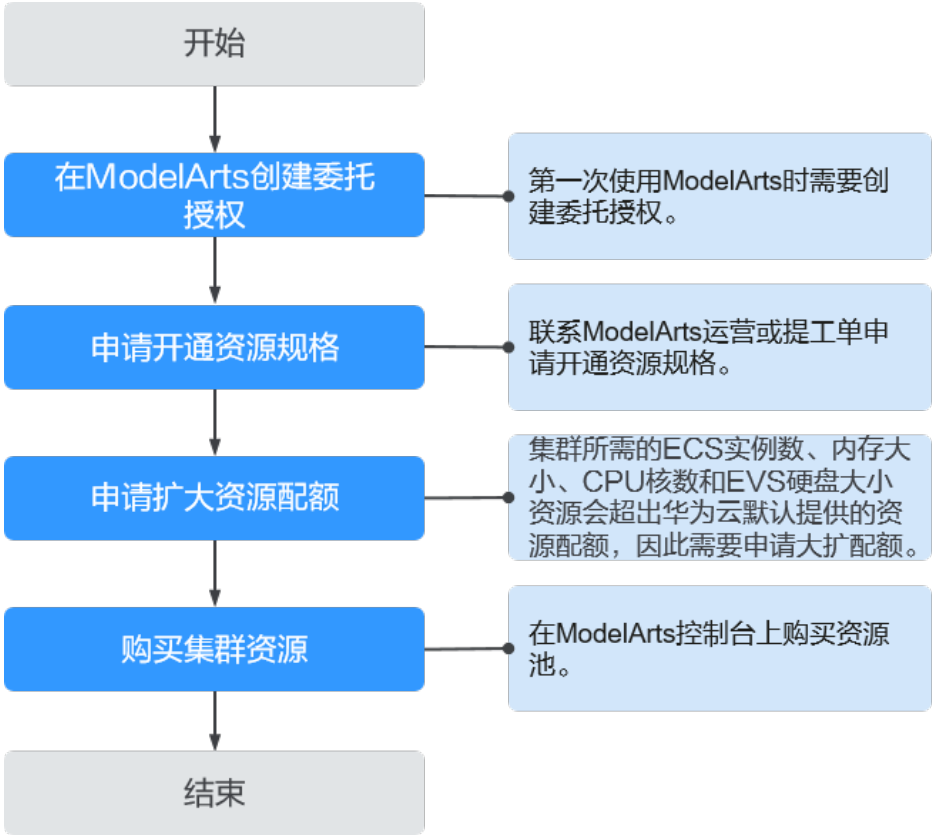


表 1-5 用户侧任务流程

阶段	任务	说明	参考文档
预购集群资源	登录华为云控制台，在ModelArts上创建委托授权。	第一次使用ModelArts时需要创建委托授权，授权允许ModelArts代表用户去访问其他云服务。 如果之前已经创建过委托授权，需要更新委托相应的权限。	**配置ModelArts访问授权**
	申请开通资源规格	当前部分规格为受限购买，需要提前联系客户经理申请开通资源规格，预计1~3个工作日内开通（若无客户经理可提交工单反馈）。	**申请开通资源规格**
	申请扩大资源配额。	集群所需的ECS实例数、内存大小、CPU核数和EVS硬盘大小资源会超出华为云默认提供的资源配额，因此需要申请扩大配额。 具体的配额方案请联系客户经理获取。 配额需大于要开通的资源，且在购买开通前完成配额提升，否则会导致资源开通失败。	**提升资源配额**
购买集群资源	在ModelArts控制台上购买资源池。	-	**购买集群资源**

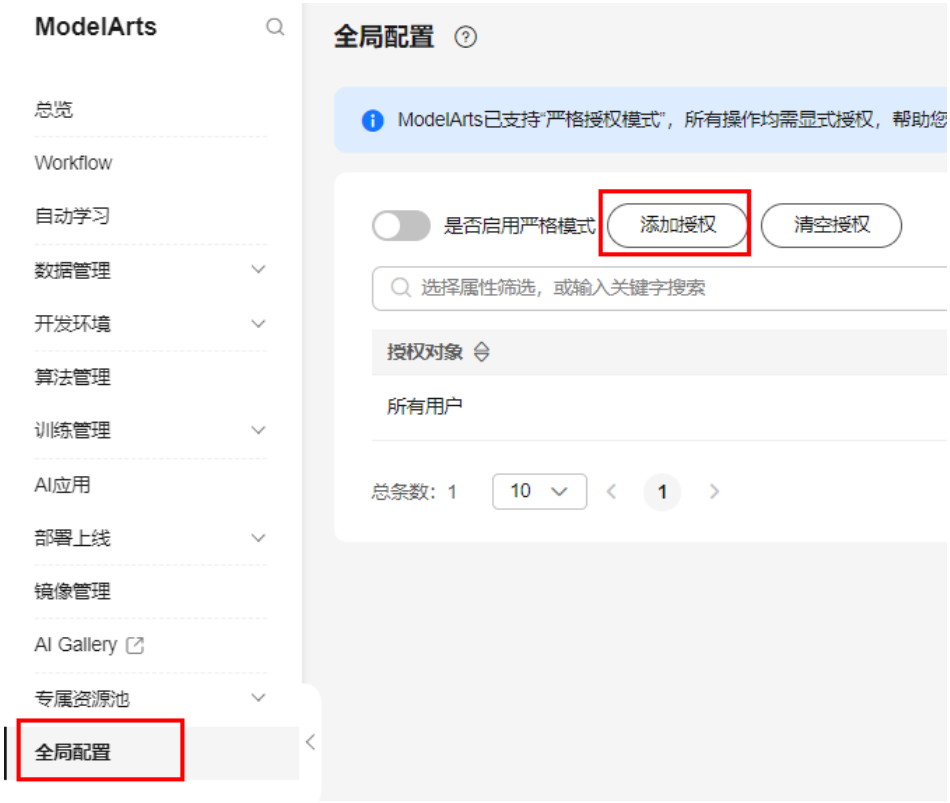
Step1 在 ModelArts 上创建委托授权

- 新建委托

  第一次使用ModelArts时需要创建委托授权，授权允许ModelArts代表用户去访问其他云服务。使用ModelArts Lite的资源池需要授权允许ModelArts代表用户访问云容器引擎服务CCE、裸金属服务BMS、镜像服务IMS和密钥管理服务DEW。

  进入到ModelArts控制台的“全局配置”页面，单击“添加授权”，根据提示进行操作。

  图 1-3 添加授权

- 更新委托

  如果之前给ModelArts创过委托授权，此处需要更新授权。

  a. 进入到ModelArts控制台的“专属资源池”页面，查看是否存在授权缺失的提示。

  b. 如果有授权缺失，根据提示，单击“此处”更新委托。根据提示选择“追加至已有授权”，单击“确定”，系统会提示权限更新成功。

Step2 申请开通资源规格

当前部分规格为受限购买（如modelarts.bm.npu.arm.8snt9b3.d），需要提前联系客户经理申请开通资源规格，预计1~3个工作日内开通（若无客户经理可提交工单反馈）。

Step3 申请扩大资源配额

集群所需的ECS实例数、内存大小、CPU核数和EVS硬盘大小资源会超出华为云默认提供的资源配额，因此需要申请扩大配额。请先联系客户经理确认资源配额提升具体方案，再参考本章节申请扩大配额。

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。

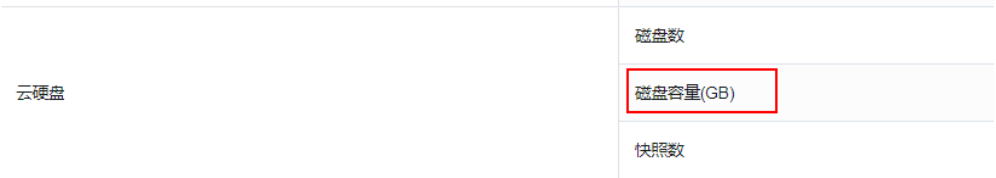
图 1-4 我的配额

步骤3 在服务配额页面，单击右上角的“申请扩大配额”，填写申请材料后提交工单。

申请扩大配额主要是申请弹性云服务器ECS实例数、核心数（CPU核数）、RAM容量（内存大小）和云硬盘EVS磁盘容量这4个资源配额。具体的配额数量请先联系客户经理获取。

图 1-5 ECS 资源类型

图 1-6 云硬盘资源类型



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

Step4 购买集群资源

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“AI专属资源池 > 弹性集群 Cluster”，进入“弹性集群 Cluster”页面。
2. 在“弹性集群 Cluster”页签，单击“购买AI专属集群”，进入购买AI专属集群界面，参见下表填写参数。

表 1-6 专属资源池的参数说明

参数名称	子参数	说明
名称	-	专属资源池的名称。 只能以小写字母开头，由小写字母、数字、中划线（-）组成，不能以中划线结尾。
描述	-	专属资源池的简要说明。
使用场景	-	选择“ModelArts Lite”。
计费模式	-	选择计费模式，“包年/包月”或“按需计费”。
CCE集群	-	在下拉列表中选择用户账户下已有的CCE集群。如果没有集群，单击右侧的“创建集群”，先去创建集群。集群配套版本请参考**不同机型的对应的软件配套版本**。 创建Cluster资源池时，请确保CCE集群为“运行中”状态。 **说明** • 当前仅支持CCE集群1.23&1.25&1.28版本。 • 若您没有可用的CCE集群，可先创建CCE集群。CCE 1.28集群版本支持通过控制台、API方式创建，CCE 1.23和CCE 1.25版本支持通过API方式创建。不同版本的CCE集群创建方式请见**Kubernetes版本策略**。 • 若您已有CCE集群，但CCE集群版本低于1.23，则可参考**升级集群的流程和方法**，建议将集群升级至1.28版本。
自定义节点名称	-	开启后，可为节点名称添加前缀。 • 添加前缀后，节点名称由前缀+随机数组成。 • 输入长度范围为1到64个字符。 • 前缀必须以小写字母开头，并由小写字母和数字组成，以“-”分隔。例如：node-com。

参数名称	子参数	说明
规格管理	-	支持添加多个规格。限制如下： ● 当选择多个相同规格时，可打开高级选项指定节点池名称，至多只有一个可不指定节点池名称。 ● 选择多个规格的CPU架构必须相同。例如都是X86，或者都是ARM。 ● 如果选择了多个GPU或NPU规格，由于不同规格的参数网络平面不互通，分布式训练时训练速度会受到影响。如果您要做分布式训练，建议您只选择一个GPU或NPU规格。 ● 一个资源池中，最多可添加10种规格。
	规格类型	请根据界面提示选择需要使用的规格。平台分配的资源规格包含了一定的系统损耗，实际可用的资源量小于规格标称的资源。实际可用的资源量可在专属资源池创建成功后，在详情页的“节点”页签中查看。
	可用区	您可以根据实际情况选择“随机分配”或“指定AZ”。可用区是在同一区域下，电力、网络隔离的物理区域。可用区之间内网互通，不同可用区之间物理隔离。 ● 随机分配：系统自动分配可用区。 ● 指定AZ：指定资源池节点在哪个可用区域。考虑系统容灾时，推荐指定节点在同一个可用区。可设置可用区的节点数。
	节点数量	选择专属资源池的节点数，选择的节点数越多，计算性能越强。 当“可用区”选择“指定AZ”时，节点数量会根据可用区的数据自动计算，此处无须再次设置。 **说明** 单次创建时，节点数建议不大于30，否则可能触发限流导致创建失败。 部分规格支持整柜购买，此时节点数量会显示为“数量*整柜”，购买的节点总数为两者的乘积。整柜购买可实现不同任务间的物理隔离，避免通信冲突，在任务规模增大的同时保证计算性能线性度不下降。整柜下的节点生命周期需保持一致，需要一起创建、一起删除。 **图 1-7** 整柜购买 

参数名称	子参数	说明
	高级选项	开启后，可设置以下参数： • 容器引擎空间大小（dockersize）。若没开启，默认容器引擎空间大小为50GiB。默认值与最小值为50GiB，不同规格的最大值不同，数值有效范围请参考界面提示。 • 容器引擎：容器引擎是Kubernetes最重要的组件之一，负责管理镜像和容器的生命周期。Kubelet通过Container Runtime Interface (CRI) 与容器引擎交互，以管理镜像和容器。 您可以在创建时资源池时选择容器引擎，也可在资源池创建完成后，在扩缩容界面修改。其中Containerd调用链更短，组件更少，更稳定，占用节点资源更少，Containerd和Docker差异对比请见**容器引擎**。 若CCE集群版本低于1.23，仅支持选择Docker作为容器引擎。若CCE集群版本大于等于1.27，仅支持选择Containerd作为容器引擎。其余CCE集群版本，支持选择Containerd或Docker作为容器引擎。 • 节点池名称：新建节点池的名称，可自定义，若未指定则默认使用“规格-default”作为节点池名称。当选择多个相同规格时，至多只有一个可不指定节点池名称。 • 虚拟私有云：默认为CCE集群所在VPC网络，不可修改。 • 节点子网：选择同一VPC网络下的子网作为节点子网，新创建的节点将会使用该子网资源。 • 关联安全组：用于指定节点池创建出来的节点使用的安全组。最多选择4个安全组。节点安全组需要放通一些端口以保障节点通信。若不关联安全组将会使用集群中默认的节点安全组规则。 • 资源标签：通过为资源添加标签，可以对资源进行自定义标记，实现资源的分类。 • K8S标签：设置附加到Kubernetes对象（比如Pod）上的键值对。最多可以添加20条标签。使用该标签可区分不同节点，可结合工作负载的亲和能力实现容器Pod调度到指定节点的功能。 • 污点：默认为空。支持给节点加污点来设置反亲和性，每个节点最多配置20条污点。 • 安装后执行脚本：请输入脚本命令，命令中不能包含中文字符，需传入Base64转码后的脚本，转码后的字符数不能超过2048。脚本将在Kubernetes软件安装后执行，不影响Kubernetes软件安装。 **说明** • 暂不支持资源池中的存量节点池修改名称。 • 请不要在安装后执行脚本中使用reboot命令立即重启，如果需要重启，可以使用“shutdown -r 1”命令延迟1分钟重启。

参数名称	子参数	说明
自定义驱动	-	默认关闭。部分GPU和Ascend规格资源池允许自定义安装驱动。集群中默认会安装驱动，无需用户操作。只有需要指定驱动版本时，需要开启。
GPU驱动/Ascend驱动	-	打开“自定义驱动”开关，显示此参数，选择GPU/Ascend驱动。若规格类型为GPU则显示“GPU驱动”，若规格类型为Ascend则显示“Ascend驱动”。 gpu-driver配套版本请参考**不同机型的对应的软件配套版本**。
购买时长	-	选择购买时长。只有选择“包年/包月”计费模式时才需填写。
登录方式	-	集群登录方式，可以设置密码登录，也可以设置密钥对登录。 ● 密码登录：默认用户名为root，用户自己设置密码。 ● 密钥对（KeyPair）登录：可以选择已有的密钥对，或者单击右侧的“创建密钥对”，先去创建一个密钥对。
高级选项	-	选中“现在配置”，可配置标签信息。 ModelArts支持对接标签管理服务TMS，在ModelArts中创建资源消耗性任务（例如：创建Notebook、训练作业、推理在线服务）时，可以为这些任务配置标签，通过标签实现资源的多维分组管理。 标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。

图 1-8 购买专属资源池（1）

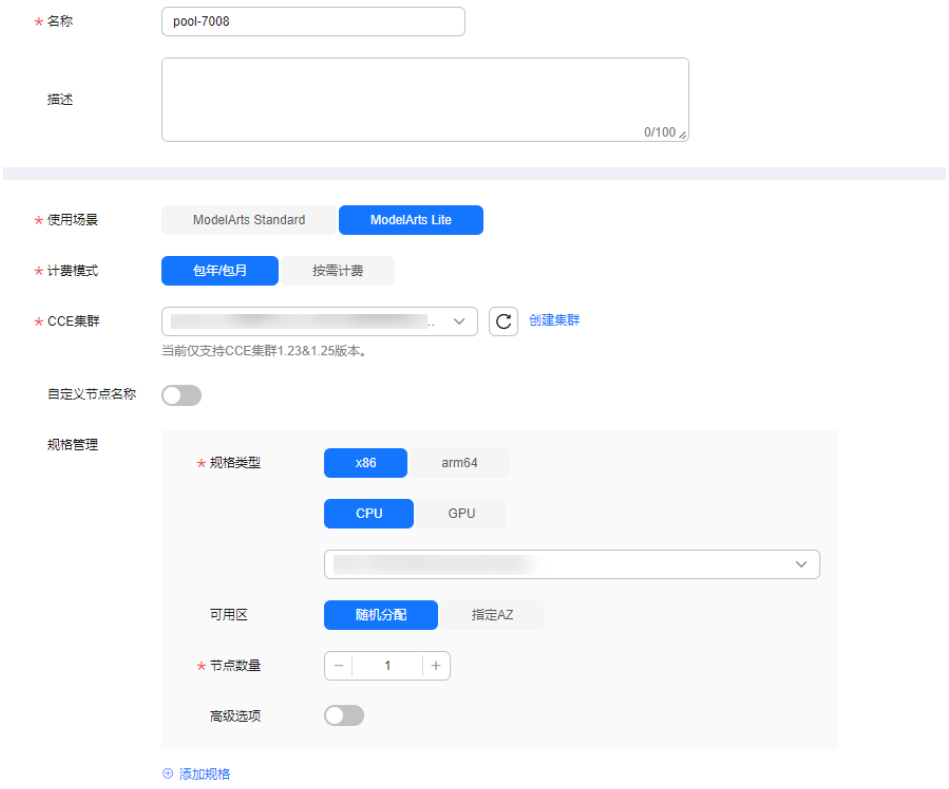
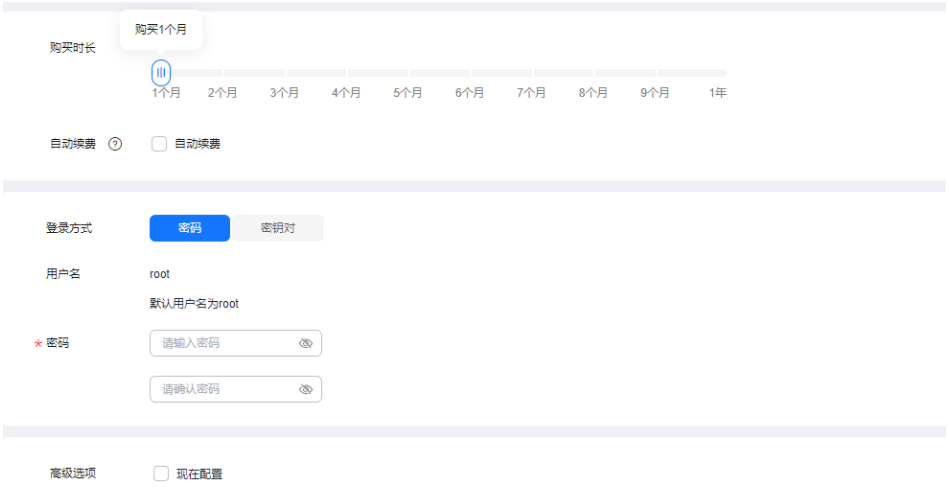


图 1-9 购买专属资源池（2）



单击“下一步”确认规格。规格确认无误后，单击“提交”，即可创建专属资源池。

- 当资源池创建成功后，资源池的状态会变成“运行中”，当“节点个数”中的“可用”和“总数”值大于0时，资源池才能下发任务。

图 1-10 查看资源池

- 可以将鼠标放在“创建中”字样上，查看当前创建过程详情。若点击查看详情，可跳转到“操作记录”中。

图 1-11 创建中状态

图 1-12 查看详情

- 可以在资源池列表左上角“操作记录”中查看资源池的任务记录。

图 1-13 操作记录

图 1-14 查看资源池状态

Step5 查看是否购买成功

当资源池创建成功后，资源池的状态会变成“运行中”。单击集群资源名称，进入资源详情页。确认购买的规格是否正确。

图 1-15 查看资源详情

1.3 Lite Cluster 资源配置

1.3.1 配置流程

本章节介绍k8s Cluster环境配置详细流程，适用于加速卡环境配置。

前提条件

- 已完成集群资源购买和开通，具体请参见**k8s Cluster资源开通**。
- 集群的配置使用需要用户具备一定的知识背景，包括但不限于**Kubernetes基础知识**、网络知识、存储和镜像知识。

配置流程

图 1-16 k8s Cluster 资源配置流程图

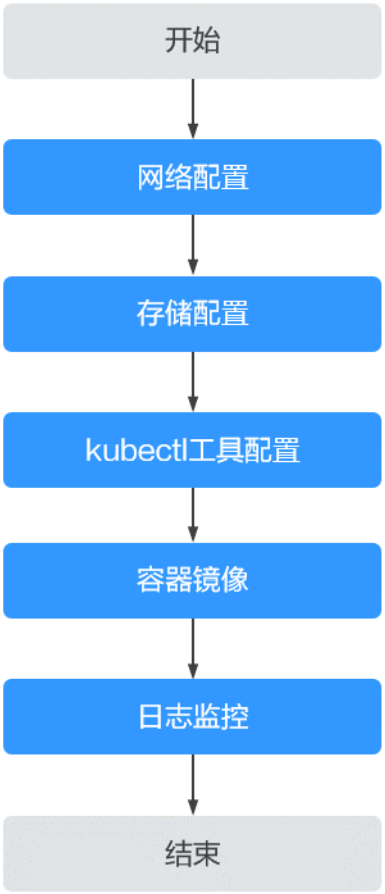


表 1-7 k8s Cluster 资源配置流程

配置顺序	配置分类	配置任务	使用场景说明	是否必选操作
1	网络	配置外网访问（EIP）	外网用户通过EIP访问集群资源中的k8s master。	必选。（二选一）
		配置外网访问（NAT）	外网用户通过公网NAT访问集群资源，入DNAT；集群内的节点通过NAT访问外网，出SNAT。	

配置顺序	配置分类	配置任务	使用场景说明	是否必选操作
		创建相同帐户下的VPC对等连接	通过对等连接可以实现同一个区域下的不同VPC之间的云上内网通信。 当资源分配在租户名下时，需要参考本章节操作，实现相同帐户下的VPC对等连接。	可选。
2	存储	容器中挂载存储	指导如何选择和使用存储服务。	可选。
3	连接集群	配置kubectl工具	指导如何配置kubectl工具。	必选。
4	容器镜像	容器镜像拉取	介绍如何拉取容器镜像。	必选。（二选一）
		镜像预热	介绍如何开启镜像预热功能。	
5	日志监控	使用ModelArts监控自定义指标	介绍如何使用ModelArts监控自定义指标。	可选。
		使用ModelArts的exporter功能在Prometheus查看监控数据	介绍如何使用ModelArts的exporter功能在Prometheus查看监控数据。	可选。

1.3.2 在 k8s 集群配置 Ascend 使用环境

前提条件

- 专属资源池未适配1.23以上的CCE版本，故在本指导适用于1.23版本CCE。
- 已购买资源，购买资源步骤请参考集群资源购买。

操作步骤

Step 1：登录节点

Step 2：配置kubectl工具

Step 3：准备业务基础镜像

Step 4：docker run方式启动任务

Step 1：登录节点

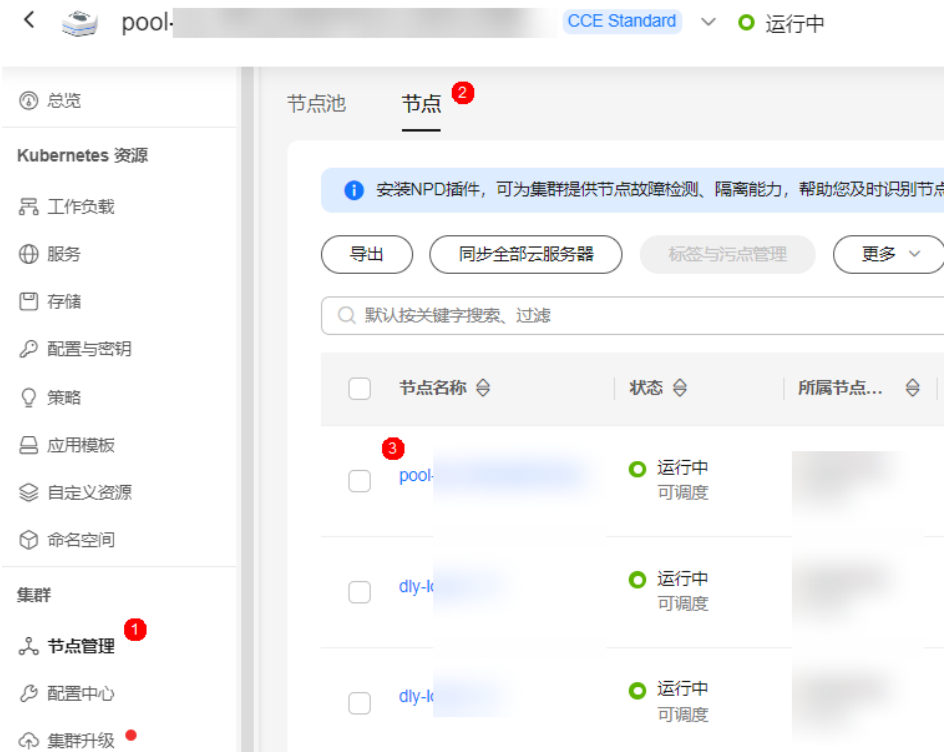
（推荐）方式1：通过绑定公网ip的方式

客户可以为需要登录的节点绑定公网ip，然后可以通过Xshell、MobaXterm等bash工具登录节点。

步骤1 使用华为云账号登录**CCE管理控制台**。

步骤2 在CCE集群详情页面，单击“节点管理”页签，在“节点”页签中单击需要登录的节点名称，跳转至弹性云服务器页面。

图 1-17 节点管理



步骤3 绑定弹性公网IP。

若已有未绑定的弹性公网IP，直接选择即可。如果没有可用的弹性公网IP，需要先购买弹性公网IP。

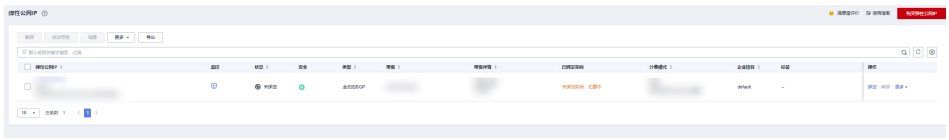
图 1-18 弹性公网 IP

单击“购买弹性公网IP”，进入购买页。

图 1-19 绑定弹性公网 IP

图 1-20 购买弹性公网 IP

图 1-21 未绑定的弹性公网 IP



完成购买后，返回弹性云服务器页面，刷新列表。

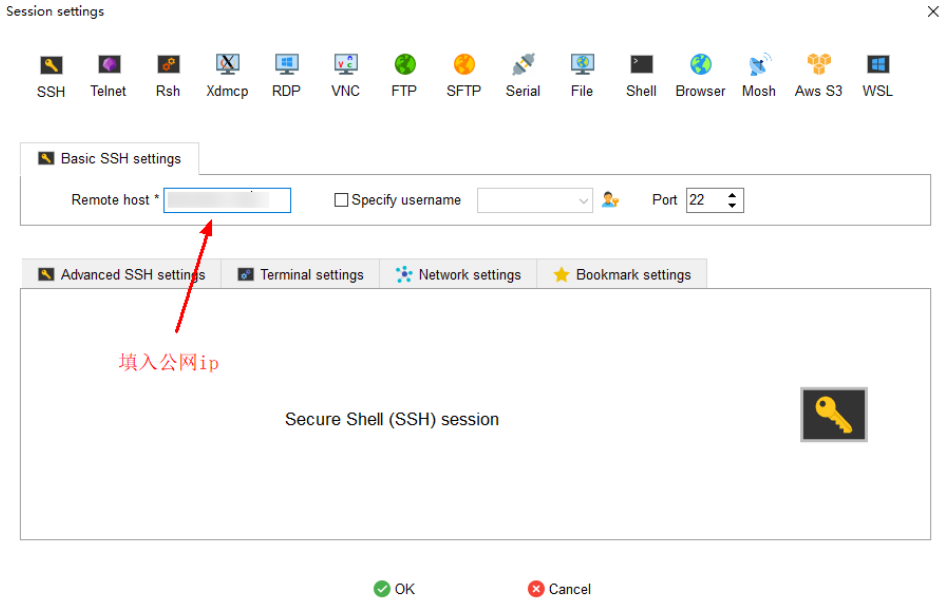
图 1-22 刷新列表

选择刚才创建的弹性公网IP，单击“确定”。

图 1-23 绑定弹性公网 IP

步骤4 绑定完成后，通过MobaXterm、Xshell登录。以MobaXterm为例，填入弹性公网IP，登录节点。

图 1-24 登录节点



----结束

方式2：通过华为云自带的远程登录功能

步骤1 使用华为云账号登录**CCE管理控制台**。

步骤2 在CCE集群详情页面，单击“节点管理”页签，在“节点”页签中单击需要登录的节点名称，跳转至弹性云服务器页面。

图 1-25 节点管理

步骤3 单击“远程登录”，在弹出的窗口中，单击“CloudShell登录”。

图 1-26 远程登录

步骤4 在CloudShell中设置密码等参数后，单击“连接”即可登录节点，CloudShell介绍可参见**远程登录Linux弹性云服务器（CloudShell方式）**。

----结束

Step 2：配置 kubectl 工具

步骤1 进入专属资源池。

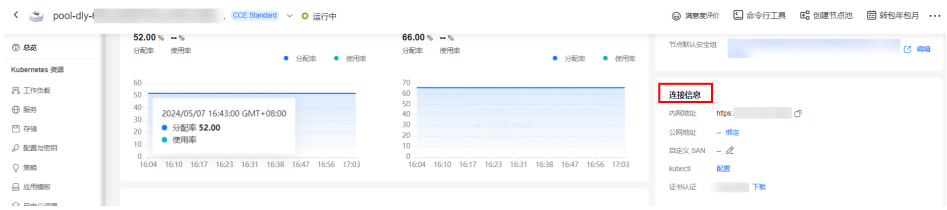
图 1-27 专属资源池页签

步骤2 点击创建的专属资源池，进入专属资源池详情页面，点击对应的CCE集群，进入CCE集群详情页面。

图 1-28 专属资源池详情

步骤3 在CCE集群详情页面中，在“集群信息”找到“连接信息”。

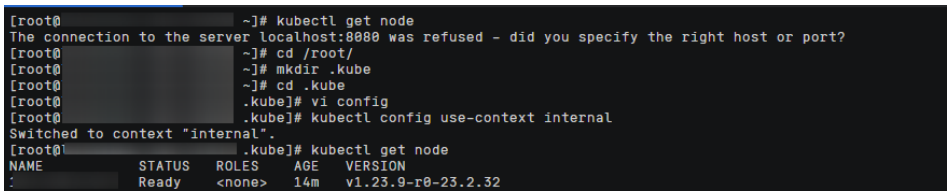
图 1-29 链接信息



步骤4 使用kubectl工具。

- 若通过内网使用kubectl工具，需要将kubectl工具安装在和集群在相同vpc下的某一台机器上。点击连接信息下kubectl后的“配置”按钮，根据界面提示使用kubectl工具。

图 1-30 通过内网使用 kubectl 工具



- 通过公网使用kubectl工具，可以将kubectl安装在任一台可以访问公网的机器。首先需要绑定公网地址，点击公网地址后的“绑定”按钮。

图 1-31 绑定公网地址

选择公网IP后单击“确定”，完成公网IP绑定。如果没有可选的公网IP，单击“创建弹性IP”跳至弹性公网IP页面进行创建。

绑定完成后，点击连接信息下kubectl后的“配置”按钮，根据界面提示使用kubectl工具。

步骤5 验证。

在安装了kubectl工具的机器上执行如下命令，显示集群节点即为成功。

```
kubectl get node
```

----结束

Step 3：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。

步骤1 根据所需要的环境拉取镜像。

- 拉取Ascend+PyTorch或Ascend+MindSpore镜像，镜像地址详见ModelArts产品发布说明中的**基础镜像**章节：

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

步骤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
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
```

步骤3 进入容器：

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

步骤4 进入conda环境：

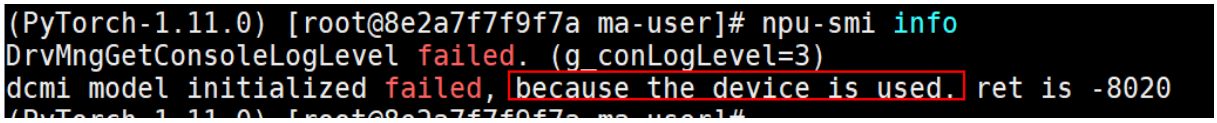
```
source /home/ma-user/.bashrc
cd ~
```

步骤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 1-32 报错信息



步骤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 1-33 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]#
```

----结束

Step 4：docker run 方式启动任务

Snt9B集群在纳管到cce集群后，都会自动安装docker，如果仅做测试验证，可以不需要通过创建deployment或者volcano job的方式，直接启动容器进行测试。训练测试用例使用NLP的bert模型，详细代码和指导可参考**Bert**。

步骤1 拉取镜像。本测试镜像为bert_pretrain_mindspore:v1，已经把测试数据和代码打进镜像中。

```
docker pull swr.cn-southwest-2.myhuaweicloud.com/os-public-repo/bert_pretrain_mindspore:v1
docker tag swr.cn-southwest-2.myhuaweicloud.com/os-public-repo/bert_pretrain_mindspore:v1
bert_pretrain_mindspore:v1
```

步骤2 启动容器。

```
docker run -tid --privileged=true \
-u 0 \
-v /dev/shm:/dev/shm \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver  \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v /etc/hccn.conf:/etc/hccn.conf \
bert_pretrain_mindspore:v1 \
bash
```

参数含义：

- --privileged=true //特权容器，允许访问连接到主机的所有设备
- -u 0 //root用户
- -v /dev/shm:/dev/shm //防止shm太小训练任务失败
- --device=/dev/davinci0 //npu卡设备

- --device=/dev/davinci1 //npu卡设备
- --device=/dev/davinci2 //npu卡设备
- --device=/dev/davinci3 //npu卡设备
- --device=/dev/davinci4 //npu卡设备
- --device=/dev/davinci5 //npu卡设备
- --device=/dev/davinci6 //npu卡设备
- --device=/dev/davinci7 //npu卡设备
- --device=/dev/davinci_manager //davinci相关的设备管理的设备
- --device=/dev/devmm_svm //管理设备
- --device=/dev/hisi_hdc //管理设备
- -v /usr/local/Ascend/driver:/usr/local/Ascend/driver //npu卡驱动挂载
- -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi //npu-smi工具挂载
- -v /etc/hccn.conf:/etc/hccn.conf //hccn.conf配置挂载

步骤3 进入容器，并查看卡信息。

```
docker exec -it xxxxxxx bash    //进入容器，xxxxxxx替换为容器id
npu-smi info    //查看卡信息
```

图 1-34 查看卡信息

```
[root@3c799939827b bert]# npu-smi info
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.rc2                 Version: 23.0.rc2.2.b030                                      |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 0     910B1               | OK            | 93.1        46                0    / 0           |
| 0                         | 0000:C1:00.0  | 0           0    / 0          4313 / 65536         |
+===========================+===============+====================================================+
| 1     910B1               | OK            | 93.5        48                0    / 0           |
| 0                         | 0000:01:00.0  | 0           0    / 0          4313 / 65536         |
+===========================+===============+====================================================+
| 2     910B1               | OK            | 93.0        46                0    / 0           |
| 0                         | 0000:C2:00.0  | 0           0    / 0          4314 / 65536         |
+===========================+===============+====================================================+
| 3     910B1               | OK            | 93.1        47                0    / 0           |
| 0                         | 0000:02:00.0  | 0           0    / 0          4339 / 65536         |
+===========================+===============+====================================================+
| 4     910B1               | OK            | 93.3        48                0    / 0           |
| 0                         | 0000:81:00.0  | 0           0    / 0          4313 / 65536         |
+===========================+===============+====================================================+
| 5     910B1               | OK            | 94.8        48                0    / 0           |
| 0                         | 0000:41:00.0  | 0           0    / 0          4181 / 65536         |
+===========================+===============+====================================================+
| 6     910B1               | OK            | 93.3        49                0    / 0           |
| 0                         | 0000:82:00.0  | 0           0    / 0          4180 / 65536         |
+===========================+===============+====================================================+
| 7     910B1               | OK            | 93.2        48                0    / 0           |
| 0                         | 0000:42:00.0  | 0           0    / 0          4180 / 65536         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| No running processes found in NPU 0                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 1                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 2                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 3                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 4                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 5                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 6                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 7                                                            |
+===========================+===============+====================================================+
```

步骤4 执行下述命令启动训练任务。

```
cd /home/ma-user/modelarts/user-job-dir/code/bert/
export MS_ENABLE_GE=1
export MS_GE_TRAIN=1
bash scripts/run_standalone_pretrain_ascend.sh 0 1 /home/ma-user/modelarts/user-job-dir/data/cn-news-128-1f-mind/
```

图 1-35 训练进程

```
[root@3c799939827b bert]# export MS_ENABLE_GE=1
[root@3c799939827b bert]# export MS_GE_TRAIN=1
[root@3c799939827b bert]# bash scripts/run_standalone_pretrain_ascend.sh 0 1 /home/ma-user/modelarts/user-job-dir/data/cn-news-128-1f-mind/
================================================================================
Please run the script as:
bash scripts/run_standalone_pretrain_ascend.sh DEVICE_ID EPOCH_SIZE DATA_DIR SCHEMA_DIR
for example: bash scripts/run_standalone_pretrain_ascend.sh 0 40 /path/zh-wiki/ [/path/Schema.json](optional)
================================================================================
[root@3c799939827b bert]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 15:55 pts/0    00:00:00 bash
root          22       0  0 15:55 pts/1    00:00:00 bash
root          61       1 99 15:56 pts/1    00:00:04 python /home/ma-user/modelarts/user-job-dir/code/bert/scripts/../run_pretrain.py --distribute=false --epoch_size=1
root         130      22  0 15:56 pts/1    00:00:00 ps -ef
```

查看卡占用情况，如图所示，此时0号卡被占用，说明进程正常启动。

```
npu-smi info    //查看卡信息
```

图 1-36 查看卡信息

```
[root@3c799939827b bert]# npu-smi info
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.rc2                 Version: 23.0.rc2.2.b030                                      |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 0     910B1               | OK            | 102.4       47                0    / 0             |
| 0                         | 0000:C1:00.0  | 0           0    / 0          19773/ 65536         |
+===========================+===============+====================================================+
| 1     910B1               | OK            | 94.8        48                0    / 0             |
| 0                         | 0000:01:00.0  | 0           0    / 0          4313 / 65536         |
+===========================+===============+====================================================+
| 2     910B1               | OK            | 93.0        47                0    / 0             |
| 0                         | 0000:C2:00.0  | 0           0    / 0          4314 / 65536         |
+===========================+===============+====================================================+
| 3     910B1               | OK            | 93.1        47                0    / 0             |
| 0                         | 0000:02:00.0  | 0           0    / 0          4338 / 65536         |
+===========================+===============+====================================================+
| 4     910B1               | OK            | 93.2        48                0    / 0             |
| 0                         | 0000:81:00.0  | 0           0    / 0          4312 / 65536         |
+===========================+===============+====================================================+
| 5     910B1               | OK            | 95.6        48                0    / 0             |
| 0                         | 0000:41:00.0  | 0           0    / 0          4180 / 65536         |
+===========================+===============+====================================================+
| 6     910B1               | OK            | 93.6        48                0    / 0             |
| 0                         | 0000:82:00.0  | 0           0    / 0          4180 / 65536         |
+===========================+===============+====================================================+
| 7     910B1               | OK            | 93.7        49                0    / 0             |
| 0                         | 0000:42:00.0  | 0           0    / 0          4180 / 65536         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| 0       0                 | 2610117       |                          | 15435                   |
+===========================+===============+====================================================+
| No running processes found in NPU 1                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 2                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 3                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 4                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 5                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 6                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 7                                                            |
+===========================+===============+====================================================+
```

训练任务大概会运行两小时左右，训练完成后自动停止。若想停止训练任务，可执行下述命令关闭进程，查询进程后显示已无运行中python进程。

```
pkill -9 python
ps -ef
```

图 1-37 关闭训练进程

```
[root@7890c1661df8 bert]# pkill -9 python
[root@7890c1661df8 bert]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 16:34 pts/0    00:00:00 bash
root          22       0  0 16:36 pts/1    00:00:00 bash
root       18252      22  0 16:43 pts/1    00:00:00 vim scripts/run_standalone_pretrain_ascend.sh
root       18255      22  0 16:54 pts/1    00:00:00 ps -ef
```

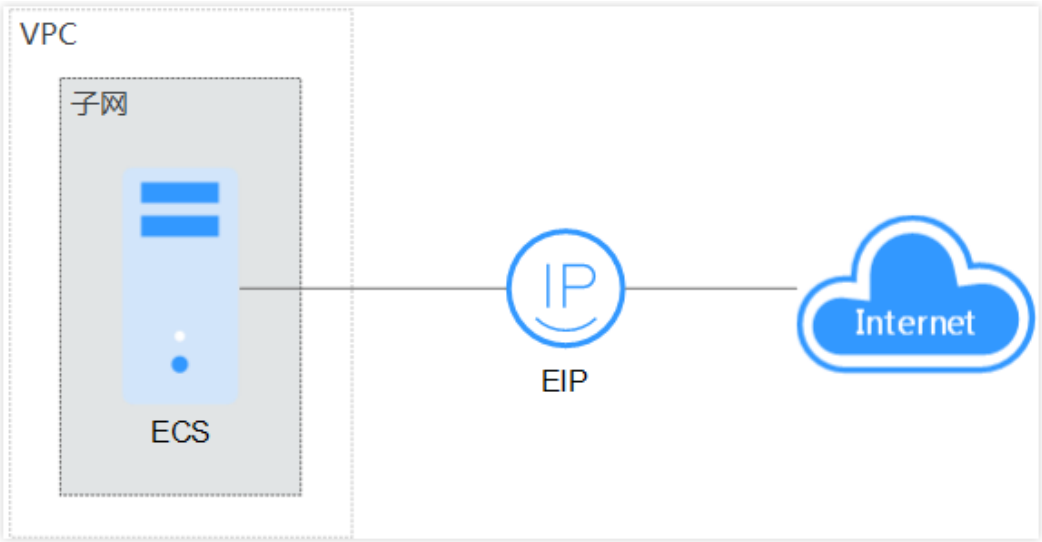
----结束

1.3.3 网络

1.3.3.1 配置外网访问（EIP）

弹性公网IP（Elastic IP，简称EIP）提供独立的公网IP资源，包括公网IP地址与公网出口带宽服务。可以与弹性云服务器、裸金属服务器、虚拟IP、弹性负载均衡、NAT网关等资源灵活地绑定及解绑。集群资源绑定EIP后，外网用户可以通过EIP访问集群资源中的k8s master。

图 1-38 外网访问（EIP）



1. 为集群资源申请并绑定弹性公网IP（EIP），具体操作请参见**为弹性云服务器申请和绑定弹性公网IP**。
2. 通过SSH方式远程访问集群资源，包括2种方式，密码方式或密钥方式，二选一即可。
   - 通过SSH密钥方式登录集群资源，具体操作请参见**SSH密钥登录方式**。
   - 通过SSH密码方式登录集群资源，具体操作请参见**SSH密码登录方式**。

1.3.3.2 创建相同帐户下的 VPC 对等连接

对等连接是建立在两个VPC之间的网络连接，不同VPC之间网络不通，通过对等连接可以实现同一个区域下的不同VPC之间的云上内网通信。

本章节指导用户创建相同帐户下的VPC对等连接，即连通的两个VPC位于同一个帐户下。

详细操作请参见：**创建相同帐户下的对等连接**。

说明

当资源分配在租户名下时，需要用户参考本章节操作，实现相同帐户下的VPC对等连接。

1.3.3.3 配置外网访问(NAT)

NAT网关（NAT Gateway）提供公网NAT网关和私网NAT网关。公网NAT网关为VPC内的云主机提供SNAT和DNAT功能，可轻松构建VPC的公网出入口。私网NAT网关为VPC内的云主机提供网络地址转换服务，使多个云主机可以共享私网IP访问用户本地数据中心或其他VPC，并支持云主机面向私网提供服务。

通过公网NAT访问集群的配置方式，具体请参见**公网NAT网关**。

1.3.4 存储

1.3.4.1 容器中挂载存储

容器中挂载存储有多种方式，不同的场景下推荐的存储方式不一样，详情如**表1-8**所示。

容器存储的基础知识了解请参见**存储基础知识**，有助您理解本章节内容。

表 1-8 容器挂载存储的方式及差异

容器挂载存储的方式	使用场景	特点	挂载操作参考
EmptyDir	适用于训练缓存场景。	Kubernetes的临时存储卷，临时卷会遵从Pod的生命周期，与Pod一起创建和删除。	**使用临时存储路径**
HostPath	适用于以下场景： 1. 容器工作负载程序生成的日志文件需要永久保存。 2. 需要访问宿主机上Docker引擎内部数据结构的容器工作负载。	节点存储。多个容器可能会共享这一个存储，会存在写冲突的问题。 Pod删除后，存储不会清理。	**使用主机路径**

容器挂载存储的方式	使用场景	特点	挂载操作参考
OBS	适用于训练数据集的存储。	对象存储。常用OBS SDK进行样本数据下载。存储量大，但是离节点比较远，直接训练速度会比较慢，通常会先将数据拉取到本地cache，然后再进行训练任务。	• 静态挂载 • 动态挂载
SFSTurbo	适用于海量小文件业务场景。	• 提供posix协议的文件系统； • 需要和资源池在同一个VPC下或VPC互通； • 价格较高。	• 静态挂载 • 动态挂载：不支持
SFS	适用于多读多写场景的持久化存储。	适用大容量扩展以及成本敏感型的业务场景，包括媒体处理、内容管理、大数据分析和分析工作负载程序等。 SFS容量型文件系统不适合海量小文件业务。	• 静态挂载 • 动态挂载
EVS	适用于Notebook场景，开发过程的数据持久化。	每个云盘只能在单个节点挂载。 存储大小根据云硬盘的大小而定。	• 静态挂载 • 动态挂载

数据盘空间分配详细说明请参见文档**数据盘空间分配说明**。

常见问题

当用户发现可访问的存储空间比较小，如果没有挂载任何外部存储，则可用存储空间根据dockerBaseSize的配置来决定。

建议：挂载外部存储空间解决存储空间受限问题。

1.3.5 kubectl 工具配置

本文介绍如何配置kubectl工具，操作步骤如下。

步骤1 进入专属资源池。

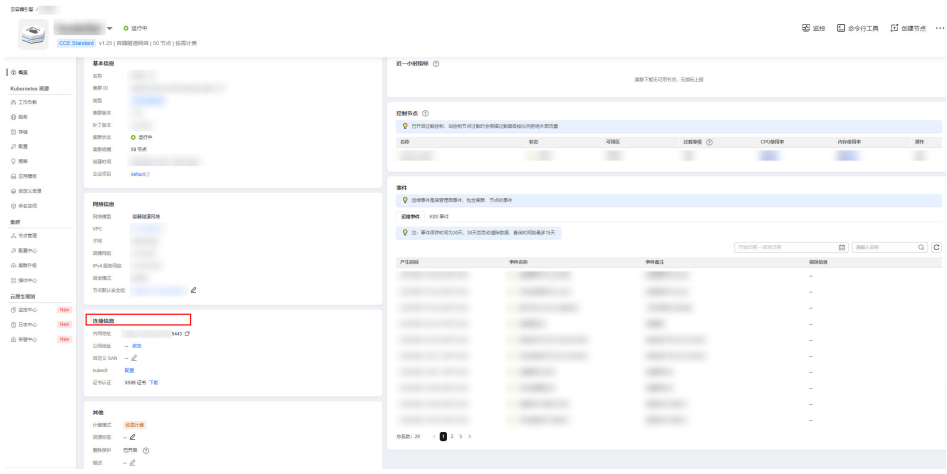
图 1-39 专属资源池页签

步骤2 单击创建的专属资源池，进入专属资源池详情页面。

图 1-40 专属资源池详情

步骤3 单击对应的CCE集群，进入CCE集群详情页面，在“集群信息”找到“连接信息”。

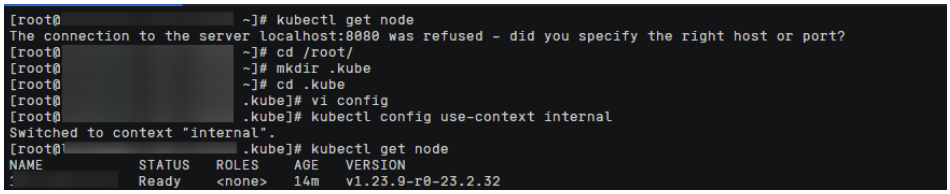
图 1-41 链接信息



步骤4 使用kubectl工具。

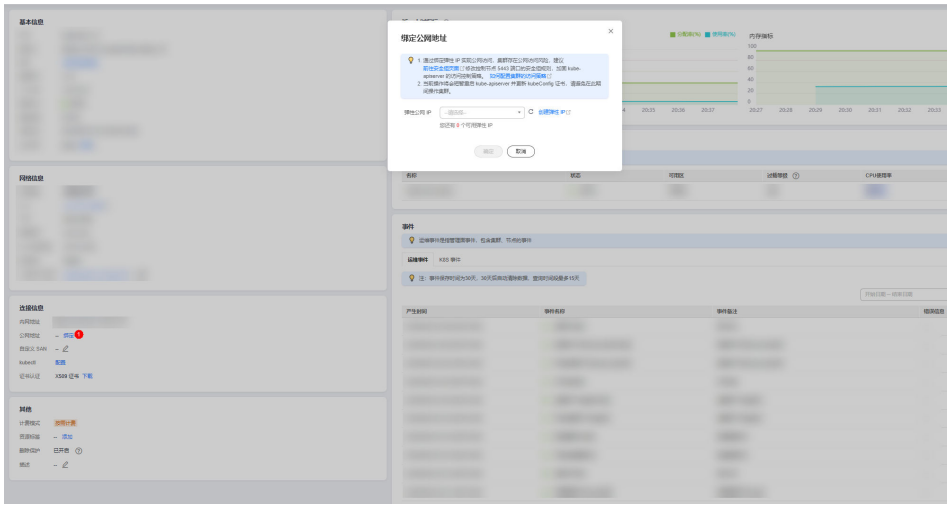
- 若通过内网使用kubectl工具，需要将kubectl工具安装在和集群在相同vpc下的某一台机器上。单击kubectl后的“配置”按钮。按照界面提示步骤操作即可。

图 1-42 通过内网使用 kubectl 工具



- 通过公网使用kubectl工具，可以将kubectl安装在任一台可以访问公网的机器。首先需要绑定公网地址，单击公网地址后的“绑定”按钮。

图 1-43 绑定公网地址

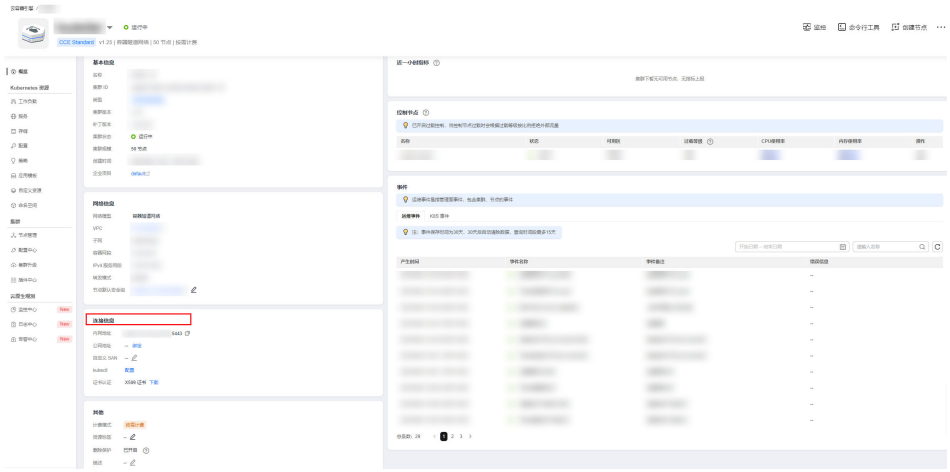


选择公网IP，或者跳至创建，创建新的弹性公网IP。

创建公网IP完成后，返回CCE集群详情页面，在“集群信息”找到“连接信息”。单击kubectl后的“配置”按钮。

按照界面提示步骤操作即可。

图 1-44 配置 kubectl



步骤5 验证。

在安装了kubectl工具的机器上执行如下命令，显示集群节点即为成功。

```
kubectl get node
```

----结束

1.3.6 容器镜像

1.3.6.1 容器镜像拉取

场景描述

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。

操作步骤

步骤1 根据所需要的环境拉取镜像。

- 拉取Ascend+PyTorch或Ascend+MindSpore镜像，镜像地址详见ModelArts产品发布说明中的**基础镜像**章节：

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

步骤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
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
```

步骤3 进入容器：

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

步骤4 进入conda环境：

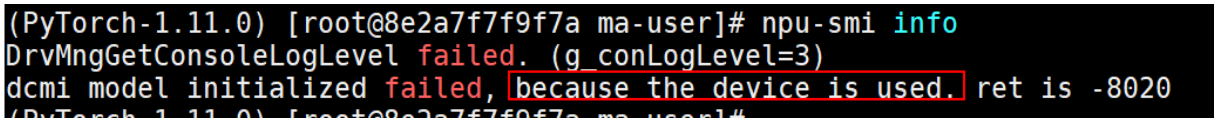
```
source /home/ma-user/.bashrc
cd ~
```

步骤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 1-45 报错信息



步骤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 1-46 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]#
```

----结束

1.3.6.2 镜像预热

场景描述

Lite Cluster资源池支持镜像预热功能，镜像预热可实现将镜像提前在资源池节点上拉取好，在推理及大规模分布式训练时有效缩短镜像拉取时间。本文将介绍如何开启镜像预热功能。

操作步骤

步骤1 单击资源池名称，进入资源池详情。

步骤2 单击左侧“配置管理”。

图 1-47 配置管理

步骤3 在镜像预热中单击编辑图标，填写镜像预热信息。

表 1-9 镜像预热参数

参数名称	说明
镜像来源	可选择“预置”或“自定义”的镜像。 ● 预置：可选择SWR服务上自有的或他人共享的镜像。 ● 自定义：可直接填写镜像地址。
添加镜像密钥	若本租户不具有预热的镜像的权限（即非公开/非本租户私有/非他人共享的镜像），此时需要添加镜像密钥。在开启镜像密钥开关后，选择命名空间及对应密钥。创建密钥方法可参考**创建密钥**，密钥类型须为kubernetes.io/dockerconfigjson类型。 若需添加多个密钥，可以单击“+”新增密钥数。
添加镜像预热配置	若需添加多个镜像，可单击此按键。

图 1-48 预置镜像预热

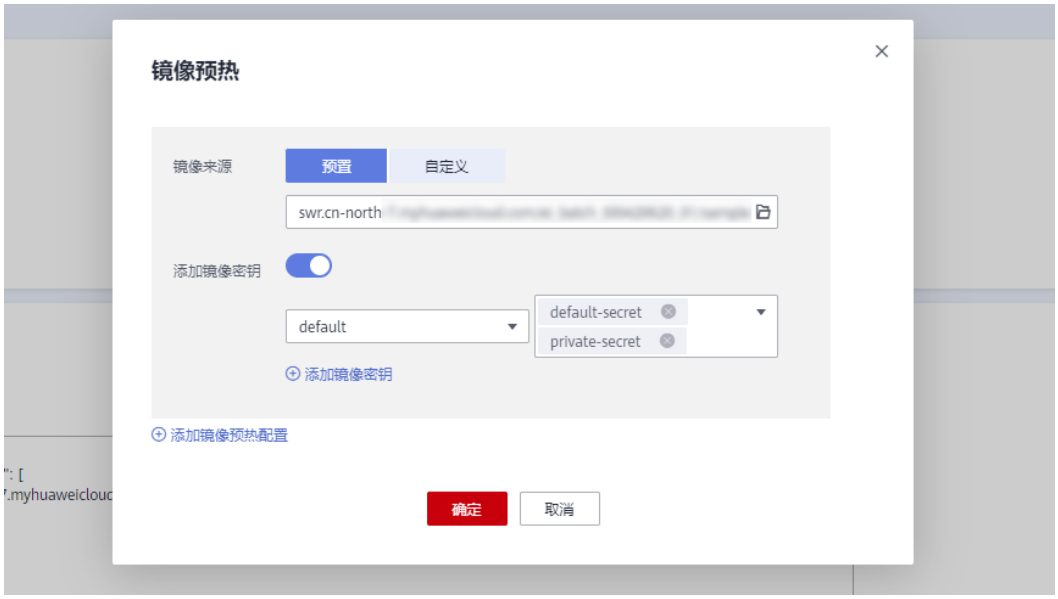


图 1-49 预置镜像选择

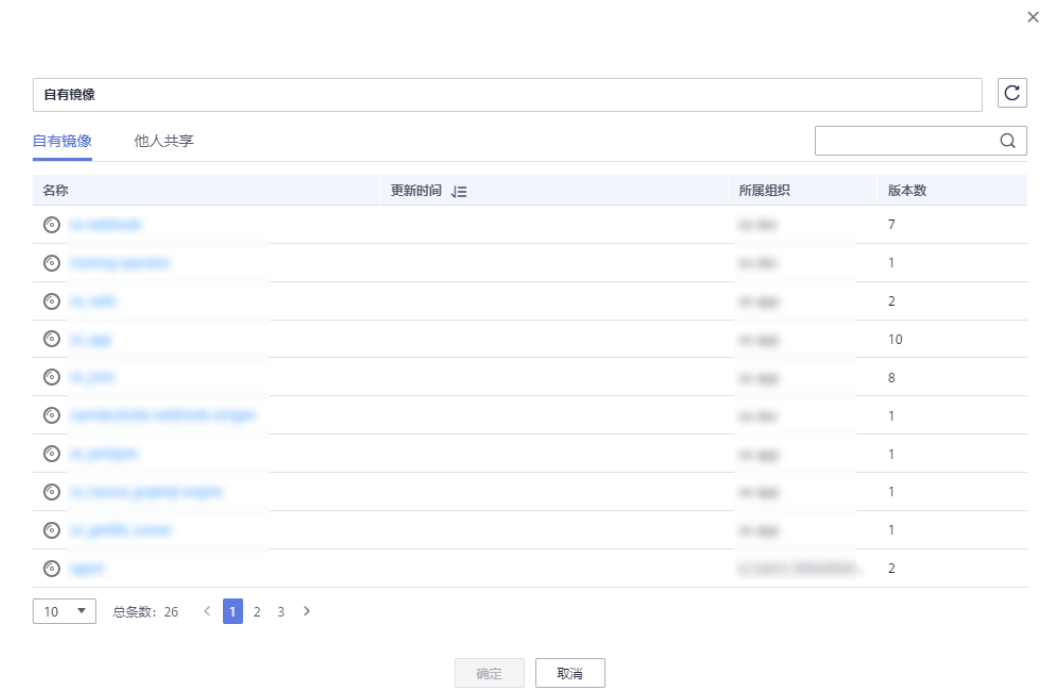


图 1-50 自定义镜像预热

创建密钥所需的仓库地址、用户名、密码、可以参考对应租户的SWR登录指令。

图 1-51 创建密钥

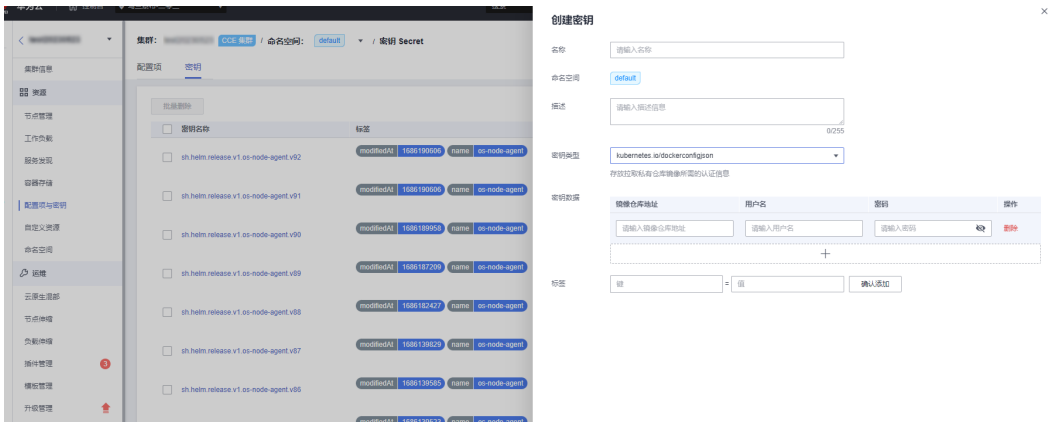
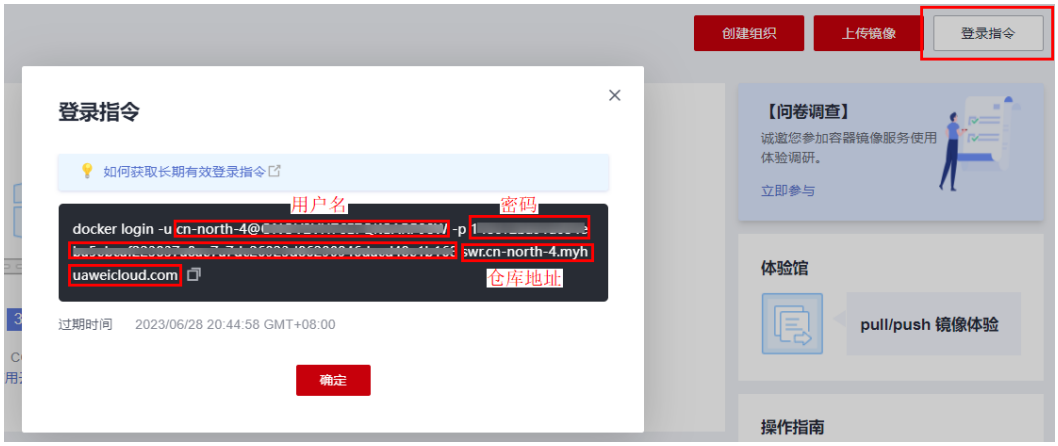


图 1-52 登录指令

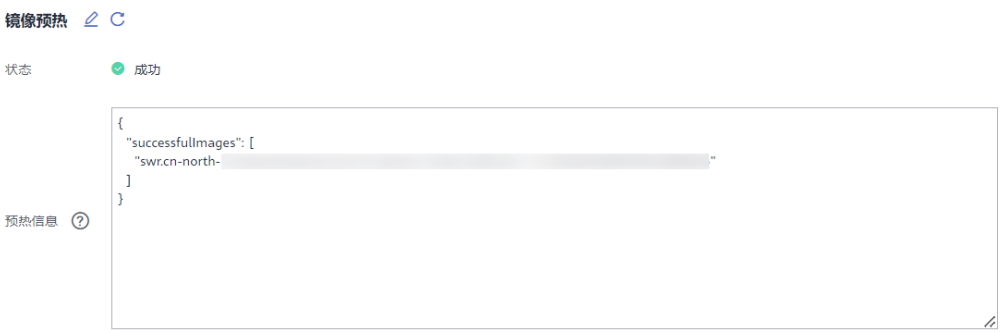


说明

上图中为临时登录指令，若需长期有效登录指令，可单击图中的“如何获取长期有效指令”链接获取指导。

步骤4 单击“确认”后，在预热信息框中可以看到已成功预热的镜像信息。

图 1-53 镜像预热成功



说明

若镜像预热失败，请检查镜像地址以及密钥是否正确。

----结束

1.3.7 日志监控

1.3.7.1 使用 ModelArts 的 exporter 功能在 Prometheus 查看监控数据

背景信息

Prometheus是一款开源监控工具，ModelArts支持Exporter功能，方便用户使用Prometheus等第三方监控系统获取ModelArts采集到的指标数据。

使用说明

- 该功能为白名单功能，如需要使用，请联系提交工单开通此功能。
- 开通此功能后，兼容Prometheus指标格式的第三方组件可通过API http://<节点IP>:<端口号>/metrics获取ModelArts采集到的指标数据。
- 开通前需要确认使用的端口号，端口号可选取10120~10139范围内的任一端口号，请确认选取的端口号在各个节点上都没有被其他应用占用。

Kubernetes 下 Prometheus 对接 ModelArts

1. 使用kubectl连接集群，详细操作请参考**通过kubectl连接集群**。
2. 配置Kubernetes的访问授权。

   使用任意文本编辑器创建prometheus-rbac-setup.yml，YAML文件内容如下：

   说明

   该YAML用于定义Prometheus要用到的角色（ClusterRole），为该角色赋予相应的访问权限。同时创建Prometheus所使用的账号（ServiceAccount），将账号与角色进行绑定（ClusterRoleBinding）。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - pods
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: default
```

3. 执行如下命令创建RBAC对应的各个资源。

```
$ kubectl create -f prometheus-rbac-setup.yml
clusterrole "prometheus" created
serviceaccount "prometheus" created
clusterrolebinding "prometheus" created
```

4. 使用任意文本编辑器创建prometheus-config.yml，内容如下。该YAML用于管理Prometheus的配置，部署Prometheus时通过文件系统挂载的方式，容器可以使用这些配置。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 10s
    scrape_configs:
    - job_name: 'modelarts'
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - role: pod
      relabel_configs:
        - source_labels: [__meta_kubernetes_pod_name]    # 指定从maos-node-agent-字符串开头的POD收集指标数据
          action: keep
          regex: ^maos-node-agent-.+
        - source_labels: [__address__]    # 指定获取指标数据的地址和端口号为__address__:9390，__address__为POD的IP地址，也是节点IP地址
          action: replace
          regex: '(.*)'
          target_label: __address__
          replacement: "${1}:10120"
```

5. 执行如下命令创建ConfigMap资源。

```
$ kubectl create -f prometheus-config.yml
configmap "prometheus-config" created
```

6. 使用任意文本编辑器创建prometheus-deployment.yml，内容如下。

说明

该YAML用于部署Prometheus。将上面创建的账号（ServiceAccount）权限赋予了Prometheus，同时将上面创建的ConfigMap资源以文件系统的方式挂载到了prometheus容器的“/etc/prometheus”目录，并且通过--config.file=/etc/prometheus/prometheus.yml参数指定了“/bin/prometheus”使用该配置文件。

```
apiVersion: v1
kind: "Service"
metadata:
  name: prometheus
  labels:
    name: prometheus
spec:
  ports:
  - name: prometheus
    protocol: TCP
    port: 9090
    targetPort: 9090
  selector:
```

```
    app: prometheus
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    name: prometheus
  name: prometheus
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      hostNetwork: true
      serviceAccountName: prometheus
      serviceAccount: prometheus
      containers:
      - name: prometheus
        image: prom/prometheus:latest
        imagePullPolicy: IfNotPresent
        command:
        - "/bin/prometheus"
        args:
        - "--config.file=/etc/prometheus/prometheus.yml"
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
        - mountPath: "/etc/prometheus"
          name: prometheus-config
      volumes:
      - name: prometheus-config
        configMap:
          name: prometheus-config
```

7. 执行如下命令创建Prometheus实例，并查看创建情况：

```
$ kubectl create -f prometheus-deployment.yml
service "prometheus" created
deployment "prometheus" created

$ kubectl get pods
NAME                                 READY     STATUS        RESTARTS   AGE
prometheus-55f655696d-wjqcl          1/1       Running       0          5s

$ kubectl get svc
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP          131d
prometheus     NodePort    10.101.255.236   <none>        9090:32584/TCP   42s
```

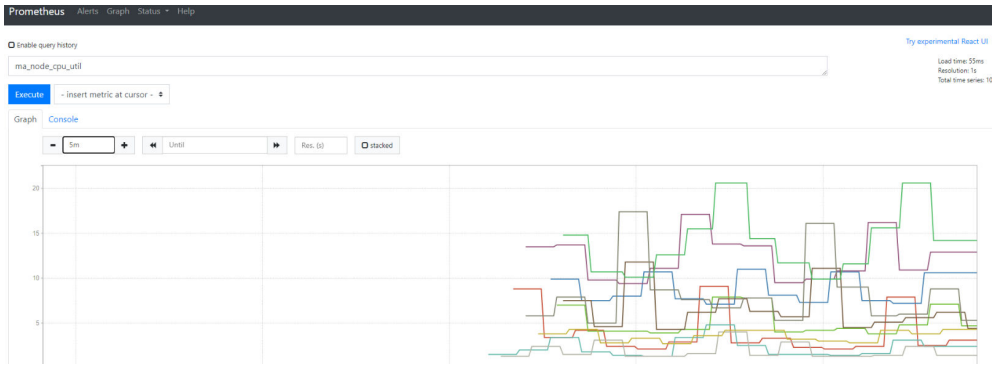
查看 Prometheus 采集的指标数据

1. 在CCE页面为Prometheus所在节点绑定弹性公网IP，并打开节点的安全组配置，添加入方向规则，允许外部访问9090端口。

   说明

   如果使用Grafana对接Prometheus制作报表，可以将Grafana部署在集群内，这里不需要对Prometheus绑定公网IP和配置安全组，只需要对Grafana绑定公网IP和配置安全组即可。

2. 在浏览器地址栏输入http://<弹性公网IP>:9090，即可打开Prometheus监控浏览页面。单击Graph菜单，在输入框输入任意一个指标名称即可看到Prometheus收集到的指标数据：



1.3.7.2 使用 ModelArts 监控自定义指标

背景信息

用户有一些自定义的指标数据需要保存到AOM，ModelArts提供了命令方式将用户的自定义指标上报保存到AOM。

约束与限制

- ModelArts以10秒/次的频率调用自定义配置中提供的命令或http接口获取指标数据。
- 自定义配置中提供的命令或http接口返回的指标数据文本不能大于8KB。

命令方式采集自定义指标数据

用于创建自定义指标采集POD的YAML文件示例如下。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-task
  annotations:
    ei.huaweicloud.com/metrics: '{"customMetrics":[{"containerName":"my-task","exec":{"command":["cat","/metrics/task.prom"]}}]}' # ModelArts从哪个容器以及使用哪个命令获取指标数据，请根据实际情况替换containerName参数和command参数
spec:
  containers:
  - name: my-task
    image: my-task-image:latest # 替换为实际使用的镜像
```

备注：业务负载和自定义指标采集可以共用一个容器，也可以由SideCar容器采集指标数据，然后将自定义指标采集容器指定到SideCar容器，这样可以不占用业务负载容器的资源。

自定义指标数据格式

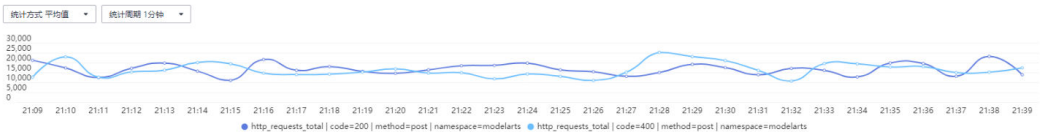
自定义指标数据的格式必须是符合open metrics规范的文本，即每个指标的格式应为：

```
<指标名称>{<标签名称>=<标签值>,...} <采样值> [毫秒时戳]
```

举例如下（#开头为注释，非必需）：

```
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total gauge
html_http_requests_total{method="post",code="200"} 1656 1686660980680
html_http_requests_total{method="post",code="400"} 2 1686660980681
```

自定义指标数据结果



1.3.7.3 如何在 AOM 上查看 ModelArts 所有监控指标?

Modelarts会定期收集资源池中各节点的关键资源（GPU、NPU、CPU、Memory等）的使用情况并上报到AOM，用户可直接在AOM上查看默认配置好的基础指标，详细步骤如下：

1. 登录控制台，搜索AOM，进入“应用运维管理 AOM”控制台。
2. 单击“监控 > 指标浏览”，进入“指标浏览”“页面”，单击“添加指标查询”。

3. 添加指标查询信息。

- 添加方式：选择“按指标维度添加”。
- 指标名称：在右侧下拉框中选择“全量指标”，然后选择想要查询的指标，参考**表1-10**、**表1-11**
- 指标维度：填写过滤该指标的标签，请参考**表4的Label名字栏**。样例如下：

4. 单击确定，即可出现指标信息。

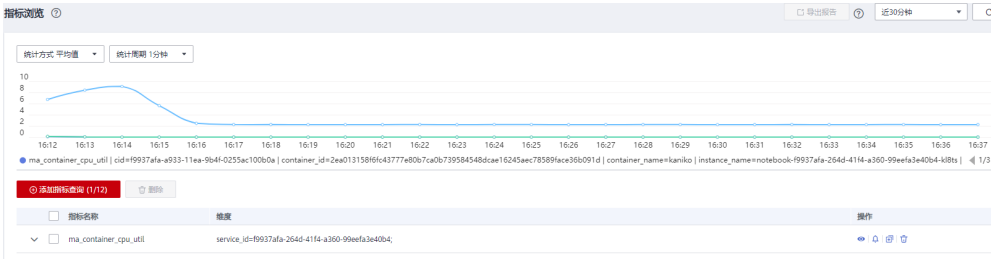


表 1-10 容器级别的指标

分类	名称	指标	指标含义	单位	取值范围
CPU	CPU使用率	ma_container_cpu_util	该指标用于统计测量对象的CPU使用率。	百分比（Percent）	0～100%
	CPU内核占用量	ma_container_cpu_used_core	该指标用于统计测量对象已经使用的CPU核个数	核（Core）	≥0
	CPU内核总量	ma_container_cpu_limit_core	该指标用于统计测量对象申请的CPU核总量。	核（Core）	≥1
内存	内存总量	ma_container_memory_capacity_megabytes	该指标用于统计测量对象申请的物理内存总量。	兆字节（Megabytes）	≥0
	物理内存使用率	ma_container_memory_util	该指标用于统计测量对象已使用内存占申请物理内存总量的百分比。	百分比（Percent）	0～100%

分类	名称	指标	指标含义	单位	取值范围
	物理内存使用量	ma_container_memory_used_megabytes	该指标用于统计测量对象实际已经使用的物理内存(对应container_memory_working_set_bytes当前内存工作集（working set）使用量。（工作区内存使用量=活跃的匿名与和缓存，以及file-baked页<=container_memory_usage_bytes）)	兆字节（Megabytes）	≥0
存储	磁盘读取速率	ma_container_disk_read_kilobytes	该指标用于统计每秒从磁盘读出的数据量。	千字节/秒（Kilobytes/Second）	≥0
	磁盘写入速率	ma_container_disk_write_kilobytes	该指标用于统计每秒写入磁盘的数据量。	千字节/秒（Kilobytes/Second）	≥0
GPU显存	显存容量	ma_container_gpu_mem_total_megabytes	该指标用于统计训练任务的显存容量。	兆字节（Megabytes）	>0
	显存使用率	ma_container_gpu_mem_util	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0～100%
	显存使用量	ma_container_gpu_mem_used_megabytes	该指标用于统计测量对象已使用的显存。	兆字节（Megabytes）	≥0

分类	名称	指标	指标含义	单位	取值范围
GPU	GPU使用率	ma_container_gpu_util	该指标用于统计测量对象的GPU使用率。	百分比（Percent）	0～100%
	GPU内存带宽利用率	ma_container_gpu_mem_copy_util	表示内存带宽利用率。以英伟达GP Vnt1为例，其最大内存带宽为900 GB/sec，如果当前的内存带宽为450 GB/sec，则内存带宽利用率为50%。	百分比（Percent）	0～100%
	GPU编码器利用率	ma_container_gpu_enc_util	表示编码器利用率	百分比（Percent）	%
	GPU解码器利用率	ma_container_gpu_dec_util	表示解码器利用率	百分比（Percent）	%
	GPU温度	DCGM_FI_DEV_GPU_TEMP	表示GPU温度。	摄氏度（℃）	自然数
	GPU功率	DCGM_FI_DEV_POWER_USAGE	表示GPU功率。	瓦特（W）	>0
	显存温度	DCGM_FI_DEV_MEMORY_TEMP	表示显存温度。	摄氏度（℃）	自然数
网络IO	下行速率	ma_container_network_receive_bytes	该指标用于统计测试对象的入方向网络流速。	字节/秒（Bytes/Second）	≥0
	接收包速率	ma_container_network_receive_packets	每秒网卡接收的数据包个数。	个/秒（Packets/Second）	≥0
	下行错包率	ma_container_network_receive_error_packets	每秒网卡接收的错误包个数。	个/秒（Packets/Second）	≥0

分类	名称	指标	指标含义	单位	取值范围
	上行速率	ma_container_network_transmit_bytes	该指标用于统计测试对象的出方向网络流速。	字节/秒（Bytes/Second）	≥0
	上行错包率	ma_container_network_transmit_error_packets	每秒网卡发送的错误包个数。	个/秒（Packets/Second）	≥0
	发送包速率	ma_container_network_transmit_packets	每秒网卡发送的数据包个数。	个/秒（Packets/Second）	≥0
NPU	NPU使用率	ma_container_npu_util	该指标用于统计测量对象的NPU使用率。（即将废止，替代指标为ma_container_npu_ai_core_util）。	百分比（Percent）	0～100%
	NPU显存使用率	ma_container_npu_memory_util	该指标用于统计测量对象已使用的NPU显存占NPU存储容量的百分比。（即将废止，snt3系列替代指标为ma_container_npu_ddr_memory_util，snt9系列替代指标为ma_container_npu_hbm_util）。	百分比（Percent）	0～100%

分类	名称	指标	指标含义	单位	取值范围
	NPU显存使用量	ma_container_npu_memory_used_megabytes	该指标用于统计测量对象已使用的NPU显存。（即将废止，snt3系列替代指标为ma_container_npu_ddr_memory_usage_bytes，snt9系列替代指标为ma_container_npu_hbm_usage_bytes）。	≥0	兆字节（Megabytes）
	NPU显存容量	ma_container_npu_memory_total_megabytes	该指标用于统计测量对象的NPU显存容量。（即将废止，snt3系列替代指标为ma_container_npu_ddr_memory_bytes，snt9系列替代指标为ma_container_npu_hbm_bytes）。	>0	兆字节（Megabytes）
	AI处理器错误码	ma_container_npu_ai_core_error_code	昇腾系列AI处理器错误码	-	-
	AI处理器健康状态	ma_container_npu_ai_core_health_status	昇腾系列AI处理器健康状态	-	• 1：健康 • 0：不健康

分类	名称	指标	指标含义	单位	取值范围
	AI处理器功耗	ma_container_npu_ai_core_power_usage_watts	昇腾系列AI处理器功耗（snt9和snt3为处理器功耗，snt3P为板卡功耗）	瓦特（W）	>0
	AI处理器温度	ma_container_npu_ai_core_temperature_celsius	昇腾系列AI处理器温度	摄氏度（℃）	自然数
	AI处理器AI CORE利用率	ma_container_npu_ai_core_util	昇腾系列AI处理器AI Core利用率	百分比（Percent）	0～100%
	AI处理器AI CORE时钟频率	ma_container_npu_ai_core_frequency_hertz	昇腾系列AI处理器AI Core时钟频率	赫兹（Hz）	>0
	AI处理器电压	ma_container_npu_ai_core_voltage_volts	昇腾系列AI处理器电压	伏特（V）	自然数
	AI处理器DDR内存总量	ma_container_npu_ddr_memory_bytes	昇腾系列AI处理器DDR内存总量	字节（Byte）	>0
	AI处理器DDR内存使用量	ma_container_npu_ddr_memory_usage_bytes	昇腾系列AI处理器DDR内存使用量	字节（Byte）	>0
	AI处理器DDR内存利用率	ma_container_npu_ddr_memory_util	昇腾系列AI处理器DDR内存利用率	百分比（Percent）	0～100%
	AI处理器HBM内存总量	ma_container_npu_hbm_bytes	昇腾系列AI处理器HBM总内存（昇腾snt9 AI处理器专属）	字节（Byte）	>0
	AI处理器HBM内存使用量	ma_container_npu_hbm_usage_bytes	昇腾系列AI处理器HBM内存使用量（昇腾snt9 AI处理器专属）	字节（Byte）	>0

分类	名称	指标	指标含义	单位	取值范围
	AI处理器HBM内存利用率	ma_container_npu_hbm_util	昇腾系列AI处理器HBM内存利用率（昇腾snt9 AI处理器专属）	百分比（Percent）	0～100%
	AI处理器HBM内存带宽利用率	ma_container_npu_hbm_bandwidth_util	昇腾系列AI处理器HBM内存带宽利用率（昇腾snt9 AI处理器专属）	百分比（Percent）	0～100%
	AI处理器HBM内存时钟频率	ma_container_npu_hbm_frequency_hertz	昇腾系列AI处理器HBM内存时钟频率（昇腾snt9 AI处理器专属）	赫兹（Hz）	>0
	AI处理器HBM内存温度	ma_container_npu_hbm_temperature_celsius	昇腾系列AI处理器HBM内存温度（昇腾snt9 AI处理器专属）	摄氏度（℃）	自然数
	AI处理器AI CPU利用率	ma_container_npu_ai_cpu_util	昇腾系列AI处理器AI CPU利用率	百分比（Percent）	0～100%
	AI处理器控制CPU利用率	ma_container_npu_ctrl_cpu_util	昇腾系列AI处理器控制CPU利用率	百分比（Percent）	0～100%

表 1-11 节点指标

分类	名称	指标	指标含义	单位	取值范围
CPU	CPU内核总量	ma_node_cpu_limit_core	该指标用于统计测量对象申请的CPU核总量。	核（Core）	≥1

分类	名称	指标	指标含义	单位	取值范围
	CPU内核占用	ma_node_cpu_used_core	该指标用于统计测量对象已经使用的CPU核数。	核（Core）	≥0
	CPU使用率	ma_node_cpu_util	该指标用于统计测量对象的CPU使用率。	百分比（Percent）	0～100%
	CPU IO等待时间	ma_node_cpu_iowait_counter	从系统启动开始累计到当前时刻，硬盘IO等待时间	jiffies	≥0
内存	物理内存使用率	ma_node_memory_util	该指标用于统计测量对象已使用内存占申请物理内存总量的百分比。	百分比（Percent）	0～100%
	物理内存容量	ma_node_memory_total_megabytes	该指标用于统计测量申请的物理内存总量。	兆字节（Megabytes）	≥0
网络IO	下行Bps	ma_node_network_receive_rate_bytes_seconds	该指标用于统计测试对象的入方向网络流速。	字节/秒（Bytes/Second）	≥0
	上行Bps	ma_node_network_transmit_rate_bytes_seconds	该指标用于统计测试对象的出方向网络流速。	字节/秒（Bytes/Second）	≥0
存储	磁盘读取速率	ma_node_disk_read_rate_kilobytes_seconds	该指标用于统计每秒从磁盘读出的数据量。只考虑被容器使用的数据盘。	千字节/秒（Kilobytes/Second）	≥0

分类	名称	指标	指标含义	单位	取值范围
	磁盘写入速率	ma_node_disk_write_rate_kilobytes_seconds	该指标用于统计每秒写入磁盘的数据量。只考虑被容器使用的数据盘。	千字节/秒（Kilobytes/Second）	≥0
	cache空间的总量	ma_node_cache_space_capacity_megabytes	该指标用于统计k8s空间的总容量。	兆字节（Megabytes）	≥0
	cache空间的使用量	ma_node_cache_space_used_capacity_megabytes	该指标用于统计k8s空间的使用量。	兆字节（Megabytes）	≥0
	容器空间的总量	ma_node_container_space_capacity_megabytes	该指标用于统计容器空间的总容量。	兆字节（Megabytes）	≥0
	容器空间的使用量	ma_node_container_space_used_capacity_megabytes	该指标用于统计容器空间的使用量。	兆字节（Megabytes）	≥0
GPU	GPU使用率	ma_node_gpu_util	该指标用于统计测量对象的GPU使用率。	百分比（Percent）	0～100%
	显存容量	ma_node_gpu_mem_total_megabytes	该指标用于统计测量对象的显存容量。	兆字节（Megabytes）	>0
	显存使用率	ma_node_gpu_mem_util	该指标用于统计测量对象已使用的显存占显存容量的百分比。	百分比（Percent）	0～100%
	显存使用量	ma_node_gpu_mem_used_megabytes	该指标用于统计测量对象已使用的显存。	兆字节（Megabytes）	≥0

分类	名称	指标	指标含义	单位	取值范围
	共享GPU任务运行数据	node_gpu_share_job_count	针对一个GPU卡，当前运行的共享资源使用的任务数量。	个	≥0
	GPU温度	DCGM_FI_DEV_GPU_TEMP	表示GPU温度。	摄氏度（℃）	自然数
	GPU功率	DCGM_FI_DEV_POWER_USAGE	表示GPU功率。	瓦特（W）	>0
	显存温度	DCGM_FI_DEV_MEMORY_TEMP	表示显存温度。	摄氏度（℃）	自然数
NPU	NPU使用率	ma_node_npu_util	该指标用于统计测量对象的NPU使用率。（即将废止，替代指标为ma_node_npu_ai_core_util）。	百分比（Percent）	0～100%
	NPU显存使用率	ma_node_npu_memory_util	该指标用于统计测量对象已使用的NPU显存占NPU存储容量的百分比。（即将废止，snt3系列替代指标为ma_node_npu_ddr_memory_util，snt9系列替代指标为ma_node_npu_hbm_util）。	百分比（Percent）	0～100%

分类	名称	指标	指标含义	单位	取值范围
	NPU显存使用量	ma_node_npu_memory_used_megabytes	该指标用于统计测量对象已使用的NPU显存。（即将废止，snt3系列替代指标为ma_node_npu_ddr_memory_usage_bytes，snt9系列替代指标为ma_node_npu_hbm_usage_bytes）	兆字节（Megabytes）	≥0
	NPU显存容量	ma_node_npu_memory_total_megabytes	该指标用于统计测量对象的NPU显存容量。（即将废止，snt3系列替代指标为ma_node_npu_ddr_memory_bytes，snt9系列替代指标为ma_node_npu_hbm_bytes）。	兆字节（Megabytes）	>0
	AI处理器错误码	ma_node_npu_ai_core_error_code	昇腾系列AI处理器错误码	-	-
	AI处理器健康状态	ma_node_npu_ai_core_health_status	昇腾系列AI处理器健康状态	-	• 1：健康 • 0：不健康
	AI处理器功耗	ma_node_npu_ai_core_power_usage_watts	昇腾系列AI处理器功耗（snt9和snt3为处理器功耗，snt3P为板卡功耗	瓦特（W）	>0

分类	名称	指标	指标含义	单位	取值范围
	AI处理器温度	ma_node_npu_ai_core_temperature_celsius	昇腾系列AI处理器温度	摄氏度（℃）	自然数
	AI处理器风扇转速	ma_node_npu_fan_speed_rpm	昇腾系列AI处理器的风扇转速	转/每分（RPM）	自然数
	AI处理器AI CORE利用率	ma_node_npu_ai_core_util	昇腾系列AI处理器AI Core利用率	百分比（Percent）	0～100%
	AI处理器AI CORE时钟频率	ma_node_npu_ai_core_frequency_hertz	昇腾系列AI处理器AI Core时钟频率	赫兹（Hz）	>0
	AI处理器电压	ma_node_npu_ai_core_voltage_volts	昇腾系列AI处理器电压	伏特（V）	自然数
	AI处理器DDR内存总量	ma_node_npu_ddr_memory_bytes	昇腾系列AI处理器DDR内存总量	字节（Byte）	>0
	AI处理器DDR内存使用量	ma_node_npu_ddr_memory_usage_bytes	昇腾系列AI处理器DDR内存使用量	字节（Byte）	>0
	AI处理器DDR内存利用率	ma_node_npu_ddr_memory_util	昇腾系列AI处理器DDR内存利用率	百分比（Percent）	0～100%
	AI处理器HBM内存总量	ma_node_npu_hbm_bytes	昇腾系列AI处理器HBM总内存（昇腾snt9 AI处理器专属）	字节（Byte）	>0
	AI处理器HBM内存使用量	ma_node_npu_hbm_usage_bytes	昇腾系列AI处理器HBM内存使用量（昇腾snt9 AI处理器专属）	字节（Byte）	>0

分类	名称	指标	指标含义	单位	取值范围
	AI处理器HBM内存利用率	ma_node_npu_hbm_util	昇腾系列AI处理器HBM内存利用率（昇腾snt9 AI处理器专属）	百分比（Percent）	0～100%
	AI处理器HBM内存带宽利用率	ma_node_npu_hbm_bandwidth_util	昇腾系列AI处理器HBM内存带宽利用率（昇腾snt9 AI处理器专属）	百分比（Percent）	0～100%
	AI处理器HBM内存时钟频率	ma_node_npu_hbm_frequency_hertz	昇腾系列AI处理器HBM内存时钟频率（昇腾snt9 AI处理器专属）	赫兹（Hz）	>0
	AI处理器HBM内存温度	ma_node_npu_hbm_temperature_celsius	昇腾系列AI处理器HBM内存温度（昇腾snt9 AI处理器专属）	摄氏度（℃）	自然数
	AI处理器AI CPU利用率	ma_node_npu_ai_cpu_util	昇腾系列AI处理器AI CPU利用率	百分比（Percent）	0～100%
	AI处理器控制CPU利用率	ma_node_npu_ctrl_cpu_util	昇腾系列AI处理器控制CPU利用率	百分比（Percent）	0～100%
infiniband或RoCE网络	网卡接收数据总量	ma_node_infiniband_port_received_data_bytes_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), received on all VLs from the port.	(counting in double words, 32 bits	≥0

分类	名称	指标	指标含义	单位	取值范围
	网卡发送数据总量	ma_node_infiniband_port_transmitted_data_bytes_total	The total number of data octets, divided by 4, (counting in double words, 32 bits), transmitted on all VLs from the port.	(counting in double words, 32 bits	≥0

分类	名称	指标	指标含义	单位	取值范围
NFS挂载状态	NFS检索文件属性操作拥塞时间	ma_node_mountstats_getattr_backlog_wait	Getattr is an NFS operation that retrieves the attributes of a file or directory, such as size, permissions, owner, etc. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS检索文件属性操作往返时间	ma_node_mountstats_getattr_rtt	Getattr is an NFS operation that retrieves the attributes of a file or directory, such as size, permissions, owner, etc. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS检查文件权限操作拥塞时间	ma_node_mountstats_access_backlog_wait	Access is an NFS operation that checks the access permissions of a file or directory for a given user. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS检查文件权限操作往返时间	ma_node_mountstats_access_rtt	Access is an NFS operation that checks the access permissions of a file or directory for a given user. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS解析文件句柄操作拥塞时间	ma_node_mountstats_lookup_backlog_wait	Lookup is an NFS operation that resolves a file name in a directory to a file handle. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS解析文件句柄操作往返时间	ma_node_mountstats_lookup_rtt	Lookup is an NFS operation that resolves a file name in a directory to a file handle. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS读文件操作拥塞时间	ma_node_mountstats_read_backlog_wait	Read is an NFS operation that reads data from a file. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS读文件操作往返时间	ma_node_mountstats_read_rtt	Read is an NFS operation that reads data from a file. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS写文件操作拥塞时间	ma_node_mountstats_write_backlog_wait	Write is an NFS operation that writes data to a file. Backlog wait is the time that the NFS requests have to wait in the backlog queue before being sent to the NFS server. It indicates the congestion on the NFS client side. A high backlog wait can cause poor NFS performance and slow system response times.	ms	≥0

分类	名称	指标	指标含义	单位	取值范围
	NFS写文件操作往返时间	ma_node_mountstats_write_rtt	Write is an NFS operation that writes data to a file. RTT stands for Round Trip Time and it is the time from when the kernel RPC client sends the RPC request to the time it receives the reply34. RTT includes network transit time and server execution time. RTT is a good measurement for NFS latency. A high RTT can indicate network or server issues.	ms	≥0

表 1-12 Label 名字栏

指标对象	Label名字	Label描述
容器级别指标	pod_name	容器所属pod的名字。
	pod_id	容器所属pod的ID。
	node_ip	容器所属的节点IP值。
	container_id	容器ID。

指标对象	Label名字	Label描述
	cluster_id	集群ID。
	cluster_name	集群名称。
	container_name	容器名称。
	namespace	是用户创建的POD所在的命名空间。
	app_kind	取自首个ownerReferences的kind字段。
	app_id	取自首个ownerReferences的uid字段。
	app_name	取自首个ownerReferences的name字段。
	npu_id	昇腾卡的ID信息，比如davinci0（即将废止）。
	device_id	昇腾系列AI处理器的Physical ID。
	device_type	昇腾系列AI处理器类型。
	pool_id	物理专属池对应的资源池id。
	pool_name	物理专属池对应的资源池name。
	gpu_uuid	容器使用的GPU的UUID。
	gpu_index	容器使用的GPU的索引。
	gpu_type	容器使用的GPU的型号。
node级别指标	cluster_id	该node所属CCE集群的ID。
	node_ip	节点的IP。
	host_name	节点的主机名。
	pool_id	物理专属池对应的资源池ID。
	project_id	物理专属池的用户的project id。
	npu_id	昇腾卡的ID信息，比如davinci0（即将废止）。
	device_id	昇腾系列AI处理器的Physical ID。
	device_type	昇腾系列AI处理器类型。
	gpu_uuid	节点上GPU的UUID。
	gpu_index	节点上GPU的索引。
	gpu_type	节点上GPU的型号。
	device_name	infiniband或RoCE网络网卡的设备名称。
	port	IB网卡的端口号。
	physical_state	IB网卡每个端口的状态。

指标对象	Label名字	Label描述
	firmware_version	IB网卡的固件版本。
	filesystem	NFS挂载的文件系统。
	mount_point	NFS的挂载点。
Diagnos	cluster_id	GPU所在节点所属的CCE集群ID。
	node_ip	GPU所在节点的IP。
	pool_id	物理专属池对应的资源池ID。
	project_id	物理专属池的用户的project id。
	gpu_uuid	GPU的UUID。
	gpu_index	节点上GPU的索引。
	gpu_type	节点上GPU的型号。
	device_name	infiniband或RoCE网络网卡的设备名称。
	port	IB网卡的端口号。
	physical_state	IB网卡每个端口的状态。
	firmware_version	IB网卡的固件版本。

1.4 Lite Cluster 资源使用

1.4.1 风险操作

当您在CCE、ECS或BMS服务控制台直接操作ModelArts Lite k8s Cluster资源时，可能会导致资源池部分功能异常，下表可帮助您定位异常出现的原因，风险操作包括但不限于以下内容：

表 1-13 操作及其对应风险

类型	操作	风险
集群	升级、修改、休眠集群、删除集群等。	可能影响ModelArts侧基本功能，包括但不限于资源池管理、节点管理、扩缩容、驱动升级等。
节点	退订、移除、关机、污点管理、修改登录密码、修改网络安全组、切换/重装操作系统等。	可能影响ModelArts侧基本功能，包括但不限于节点管理、扩缩容、驱动升级等。
网络	修改/删除集群关联网段。	影响ModelArts侧基本功能，包括但不限于节点管理、扩缩容、驱动升级等。

类型	操作	风险
插件	升级、卸载gpu-beta插件。	可能导致NPU驱动使用异常。
	升级、卸载huawei-npu插件。	可能导致NPU驱动使用异常。
	升级、卸载volcano插件。	可能导致作业调度异常。
	卸载ICAgent插件。	可能导致日志、监控功能异常。
helm	升级、回退、卸载os-node-agent。	可能影响节点故障检测。
	升级、回退、卸载rdma-sriov-dev-plugin。	可能影响容器内使用RDMA网卡。

1.4.2 在 lite 资源池上使用 Snt9B 完成分布式训练任务

场景描述

本案例介绍如何在Snt9B上进行分布式训练任务。lite资源池已经默认安装volcano调度器，训练任务默认使用volcano job形式下发lite池集群。训练测试用例使用NLP的bert模型，详细代码和指导可参考**Bert**。

操作步骤

步骤1 拉取镜像。本测试镜像为bert_pretrain_mindspore:v1，已经把测试数据和代码打进镜像中。

```
docker pull swr.cn-southwest-2.myhuaweicloud.com/os-public-repo/bert_pretrain_mindspore:v1
docker tag swr.cn-southwest-2.myhuaweicloud.com/os-public-repo/bert_pretrain_mindspore:v1
bert_pretrain_mindspore:v1
```

步骤2 在主机上新建config.yaml文件。

config.yaml文件用于配置pod，本示例中使用sleep命令启动pod，便于进入pod调试。您也可以修改command为对应的任务启动命令（如“python train.py”），任务会在启动容器后执行。

config.yaml内容如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-yourvcjobname     # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                    # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
data:                    #data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
        "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1   # The value cannot be changed. The volcano API must be used.
kind: Job                           # Only the job type is supported at present.
metadata:
  name: yourvcjobname               # job名字，需要和configmap中名字保持联系
  namespace: default                # 和configmap保持一致
```

```
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1                      # The value of minAvailable is 1 in a single-node scenario and N in an N-node distributed scenario.
  schedulerName: volcano               # 保持不动，Use the Volcano scheduler to schedule jobs.
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 3
  queue: default
  tasks:
  - name: "yourvcjobname-1"
    replicas: 1                        # The value of replicas is 1 in a single-node scenario and N in an N-node scenario. The number of NPUs in the requests field is 8 in an N-node scenario.
    template:
      metadata:
        labels:
          app: mindspore
          ring-controller.cce: ascend-1980  # 保持不动，The value must be the same as the label in ConfigMap and cannot be changed.
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                  - key: volcano.sh/job-name
                    operator: In
                    values:
                      - yourvcjobname
                topologyKey: kubernetes.io/hostname
        containers:
        - image: bert_pretrain_mindspore:v1              # 镜像地址，Training framework image, which can be modified.
          imagePullPolicy: IfNotPresent
          name: mindspore
          env:
          - name: name                          # The value must be the same as that of Jobname.
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: ip                            # IP address of the physical node, which is used to identify the node where the pod is running
            valueFrom:
              fieldRef:
                fieldPath: status.hostIP
          - name: framework
            value: "MindSpore"
          command:
          - "sleep"
          - "1000000000000000000"
          resources:
            requests:
              huawei.com/ascend-1980: "1"               # 需求卡数，key保持不变。Number of required NPUs. The maximum value is 16. You can add lines below to configure resources such as memory and CPU.
            limits:
              huawei.com/ascend-1980: "1"               # 限制卡数，key保持不变。The value must be consistent with that in requests.
          volumeMounts:
          - name: ascend-driver             #驱动挂载，保持不动
            mountPath: /usr/local/Ascend/driver
```



```
      - name: ascend-add-ons          #驱动挂载，保持不动
        mountPath: /usr/local/Ascend/add-ons
      - name: localtime
        mountPath: /etc/localtime
      - name: hccn                    #驱动hccn配置，保持不动
        mountPath: /etc/hccn.conf
      - name: npu-smi                 #npu-smi
        mountPath: /usr/local/bin/npu-smi
    nodeSelector:
      accelerator/huawei-npu: ascend-1980
    volumes:
    - name: ascend-driver
      hostPath:
        path: /usr/local/Ascend/driver
    - name: ascend-add-ons
      hostPath:
        path: /usr/local/Ascend/add-ons
    - name: localtime
      hostPath:
        path: /etc/localtime          # Configure the Docker time.
    - name: hccn
      hostPath:
        path: /etc/hccn.conf
    - name: npu-smi
      hostPath:
        path: /usr/local/bin/npu-smi
    restartPolicy: OnFailure
```

步骤3 根据config.yaml创建pod。

```
kubectl apply -f config.yaml
```

步骤4 检查pod启动情况，执行下述命令。如果显示“1/1 running”状态代表启动成功。

```
kubectl get pod -A
```

步骤5 进入容器，{pod_name}替换为您的pod名字（get pod中显示的名字），{namespace}替换为您的命名空间（默认为default）。

```
kubectl exec -it {pod_name} bash -n {namespace}
```

步骤6 查看卡信息，执行以下命令。

```
npu-smi info
```

kubernetes会根据config.yaml文件中配置的卡数分配资源给pod，如下图所示由于配置了1卡因此在容器中只会显示1卡，说明配置生效。

图 1-54 查看卡信息

```
[root@louleilei-louleilei-1-0 ma-user]# npu-smi info
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.rc2                 Version: 23.0.rc2.2.b030                                      |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 0     910B1               | OK            | 93.1        48                0    / 0             |
| 0                         | 0000:C1:00.0  | 0           0    / 0          4313 / 65536         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| No running processes found in NPU 0                                                            |
+===========================+===============+====================================================+
```

步骤7 修改pod的卡数。由于本案例中为分布式训练，因此所需卡数修改为8卡。

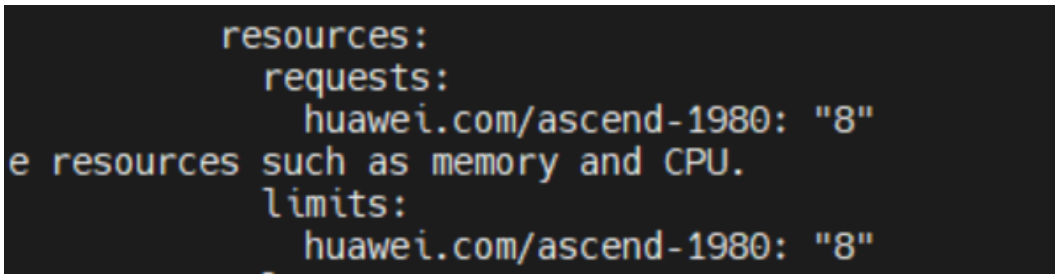
删除已创建的pod。

```
kubectl delete -f config.yaml
```

将config.yaml文件中“limit”和“request”改为8。

```
vi config.yaml
```

图 1-55 修改卡数



重新创建pod。

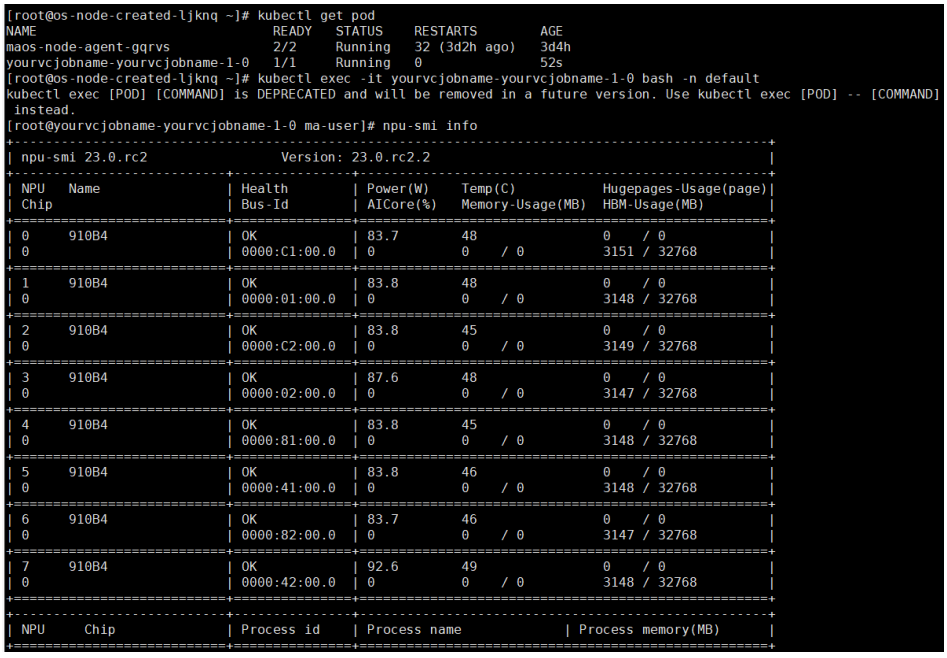
```
kubectl apply -f config.yaml
```

进入容器并查看卡信息，{pod_name}替换为您的pod名字，{namespace}替换为您的命名空间（默认为default）。

```
kubectl exec -it {pod_name} bash -n {namespace}
npu-smi info
```

如图所示为8卡，pod配置成功。

图 1-56 查看卡信息

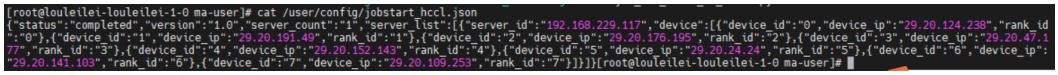


步骤8 查看卡间通信配置文件，执行以下命令。

```
cat /user/config/jobstart_hccl.json
```

多卡训练时，需要依赖“rank_table_file”做卡间通信的配置文件，该文件自动生成，pod启动之后文件地址。为“/user/config/jobstart_hccl.json”，“/user/config/jobstart_hccl.json”配置文件生成需要一段时间，业务进程需要等待“/user/config/jobstart_hccl.json”中“status”字段为“completed”状态，才能生成卡间通信信息。如下图所示。

图 1-57 卡间通信配置文件



步骤9 启动训练任务。

```
cd /home/ma-user/modelarts/user-job-dir/code/bert/
export MS_ENABLE_GE=1
export MS_GE_TRAIN=1
python scripts/ascend_distributed_launcher/get_distribute_pretrain_cmd.py --run_script_dir ./scripts/
run_distributed_pretrain_ascend.sh --hyper_parameter_config_dir ./scripts/ascend_distributed_launcher/
hyper_parameter_config.ini --data_dir /home/ma-user/modelarts/user-job-dir/data/cn-news-128-1f-mind/ --
hccl_config /user/config/jobstart_hccl.json --cmd_file ./distributed_cmd.sh
bash scripts/run_distributed_pretrain_ascend.sh /home/ma-user/modelarts/user-job-dir/data/cn-
news-128-1f-mind/ /user/config/jobstart_hccl.json
```

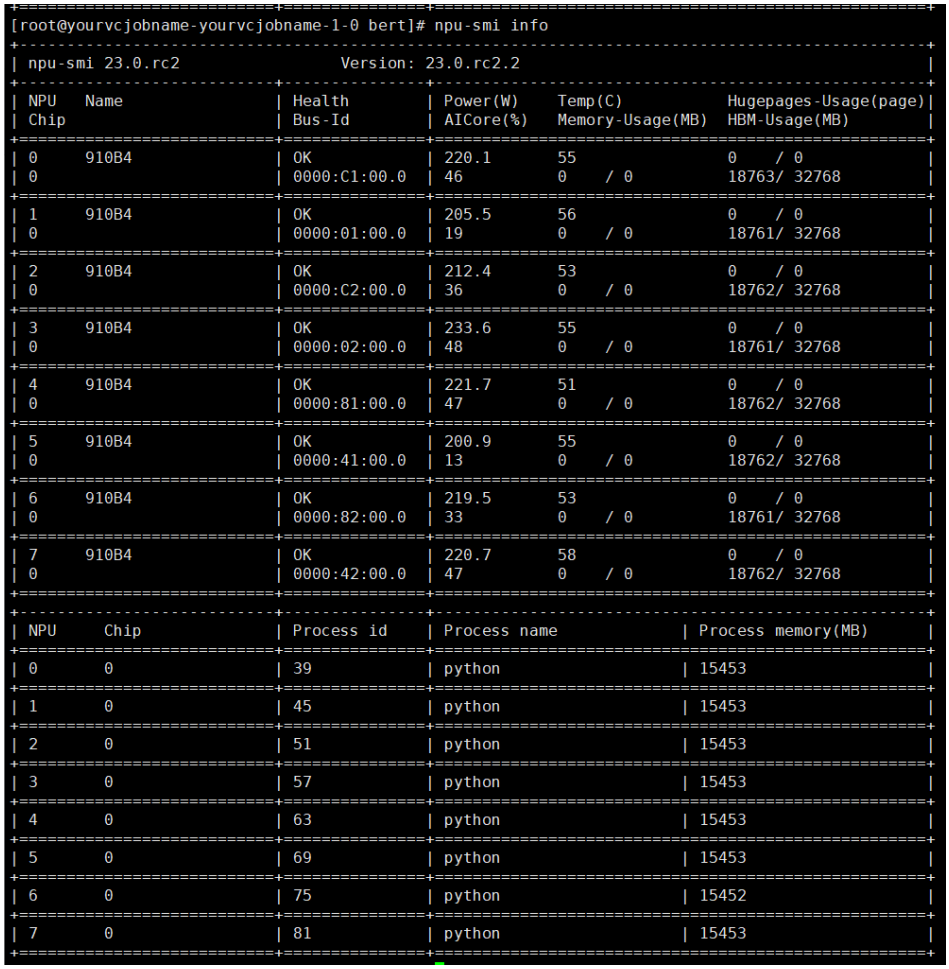
图 1-58 启动训练任务

```
[root@yourvcjobname-yourvcjobname-1-0 bert]# export MS_ENABLE_GE=1
[root@yourvcjobname-yourvcjobname-1-0 bert]# export MS_GE_TRAIN=1
[root@yourvcjobname-yourvcjobname-1-0 bert]# python scripts/ascend_distributed_launcher/get_distribute_pretrain_cmd.py
--run_script_dir ./scripts/run_distributed_pretrain_ascend.sh --hyper_parameter_config_dir ./scripts/ascend_distributed
_launcher/hyper_parameter_config.ini --data_dir /home/ma-user/modelarts/user-job-dir/data/cn-news-128-1f-mind/ --hccl_c
onfig /user/config/jobstart_hccl.json --cmd_file ./distributed_cmd.sh
start scripts/ascend_distributed_launcher/get_distribute_pretrain_cmd.py
hccl_config_dir: /user/config/jobstart_hccl.json
hccl_time_out: 120
the number of logical core: 192
total rank size: 8
this server rank size: 8
avg_core_per_rank: 24

start training for rank 0, device 0:
rank_id: 0
device_id: 0
logic_id 0
core_nums: 0-23
epoch_size: 40
data_dir: /home/ma-user/modelarts/user-job-dir/data/cn-news-128-1f-mind/
log_file_dir: /home/ma-user/modelarts/user-job-dir/code/bert/LOG0/pretraining_log.txt

start training for rank 1, device 1:
rank_id: 1
device_id: 1
logic_id 1
core_nums: 24-47
epoch_size: 40
data_dir: /home/ma-user/modelarts/user-job-dir/data/cn-news-128-1f-mind/
log_file_dir: /home/ma-user/modelarts/user-job-dir/code/bert/LOG1/pretraining_log.txt

start training for rank 2, device 2:
rank_id: 2
device_id: 2
logic_id 2
core_nums: 48-71
epoch_size: 40
data_dir: /home/ma-user/modelarts/user-job-dir/data/cn-news-128-1f-mind/
log_file_dir: /home/ma-user/modelarts/user-job-dir/code/bert/LOG2/pretraining_log.txt
```

训练任务加载需要一定时间，在等待若干分钟后，可以执行下述命令查看卡信息。如下图可见，8张卡均被占用，说明训练任务在进行中

```
npu-smi info
```

图 1-59 查看卡信息

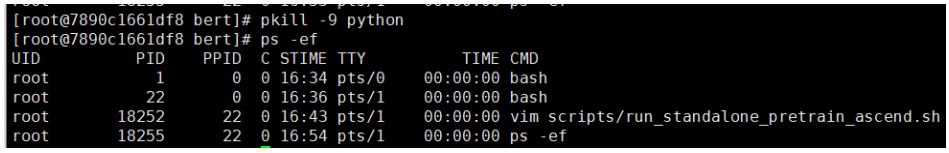


```
[root@yourvcjobname-yourvcjobname-1-0 bert]# npu-smi info
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.rc2                 Version: 23.0.rc2.2                                           |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 0     910B4               | OK            | 220.1       55                0    / 0             |
| 0                         | 0000:C1:00.0  | 46          0    / 0          18763/ 32768         |
+===========================+===============+====================================================+
| 1     910B4               | OK            | 205.5       56                0    / 0             |
| 0                         | 0000:01:00.0  | 19          0    / 0          18761/ 32768         |
+===========================+===============+====================================================+
| 2     910B4               | OK            | 212.4       53                0    / 0             |
| 0                         | 0000:C2:00.0  | 36          0    / 0          18762/ 32768         |
+===========================+===============+====================================================+
| 3     910B4               | OK            | 233.6       55                0    / 0             |
| 0                         | 0000:02:00.0  | 48          0    / 0          18761/ 32768         |
+===========================+===============+====================================================+
| 4     910B4               | OK            | 221.7       51                0    / 0             |
| 0                         | 0000:81:00.0  | 47          0    / 0          18762/ 32768         |
+===========================+===============+====================================================+
| 5     910B4               | OK            | 200.9       55                0    / 0             |
| 0                         | 0000:41:00.0  | 13          0    / 0          18762/ 32768         |
+===========================+===============+====================================================+
| 6     910B4               | OK            | 219.5       53                0    / 0             |
| 0                         | 0000:82:00.0  | 33          0    / 0          18761/ 32768         |
+===========================+===============+====================================================+
| 7     910B4               | OK            | 220.7       58                0    / 0             |
| 0                         | 0000:42:00.0  | 47          0    / 0          18762/ 32768         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+==========================+=========================+
| 0       0                 | 39            | python                   | 15453                   |
+===========================+===============+==========================+=========================+
| 1       0                 | 45            | python                   | 15453                   |
+===========================+===============+==========================+=========================+
| 2       0                 | 51            | python                   | 15453                   |
+===========================+===============+==========================+=========================+
| 3       0                 | 57            | python                   | 15453                   |
+===========================+===============+==========================+=========================+
| 4       0                 | 63            | python                   | 15453                   |
+===========================+===============+==========================+=========================+
| 5       0                 | 69            | python                   | 15453                   |
+===========================+===============+==========================+=========================+
| 6       0                 | 75            | python                   | 15452                   |
+===========================+===============+==========================+=========================+
| 7       0                 | 81            | python                   | 15453                   |
+===========================+===============+==========================+=========================+
```

若想停止训练任务，可执行下述命令关闭进程，查询进程后显示已无运行中python进程。

```
pkill -9 python
ps -ef
```

图 1-60 关闭训练进程



```
[root@7890c1661df8 bert]# pkill -9 python
[root@7890c1661df8 bert]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 16:34 pts/0    00:00:00 bash
root          22       0  0 16:36 pts/1    00:00:00 bash
root       18252      22  0 16:43 pts/1    00:00:00 vim scripts/run_standalone_pretrain_ascend.sh
root       18255      22  0 16:54 pts/1    00:00:00 ps -ef
```

说明

limit/request配置cpu和内存大小，已知单节点Snt9B机器为：8张Snt9B卡+192u1536g，请合理规划，避免cpu和内存限制过小引起任务无法正常运行。

----结束

1.4.3 在 Lite 资源池上使用 ranktable 路由规划完成 Pytorch NPU 分布式训练

场景描述

ranktable路由规划是一种用于分布式并行训练中的通信优化能力，在使用NPU的场景下，支持对节点之间的通信路径根据交换机实际topo做网络路由亲和规划，进而提升节点之间的通信速度。

本案例介绍如何在ModelArts Lite场景下使用ranktable路由规划完成Pytorch NPU分布式训练任务，训练任务默认使用Volcano job形式下发到Lite资源池集群。

约束与限制

- 该功能只支持贵阳一区域，如果要在其他区域使用请联系技术支持。
- ModelArts Lite资源池对应的CCE集群需要安装1.10.12及以上版本的华为云版Volcano插件。Volcano调度器的安装升级请参见**Volcano调度器**。仅华为云版Volcano插件支持开启路由加速特性。
- 训练使用的Python版本是3.7或3.9，否则无法实现ranktable路由加速。
- 训练作业的任务节点数要大于或等于3，否则会跳过ranktable路由加速。建议在大模型场景（512卡及以上）使用ranktable路由加速。
- 脚本执行目录不能是共享目录，否则ranktable路由加速会失败。
- 路由加速的原理是改变rank编号，所以代码中对rank的使用要统一，如果rank的使用不一致会导致训练异常。

操作步骤

步骤1 开启ModelArts Lite资源池对应的CCE集群的cabinet插件。

1. 在ModelArts Lite专属资源池列表，单击资源池名称，进入专属资源池详情页面。
2. 在基本信息页面单击CCE集群，跳转到CCE集群详情页面。
3. 在左侧导航栏选择“插件中心”，搜索“Volcano调度器”。
4. 单击“编辑”，查看高级配置的“plugins”参数下是否有“{"name":"cabinet"}”，如**图1-61**所示。

图 1-61 Volcano 调度器的高级配置

- 是，则执行**步骤2**。
- 否，则在高级配置的“plugins”参数下添加“{"name":"cabinet"}”，单击下方的“安装”使Volcano调度器更新配置，完成滚动重启。

步骤2 修改torch_npu训练启动脚本。

须知

脚本要使用torch.distributed.launch/run命令启动，不能使用mp.spawn命令启动，否则无法实现ranktable路由加速。

在使用Pytorch训练时，需要将“RANK_AFTER_ACC”环境变量赋值给“NODE_RANK”，使得ranktable路由规划生效。训练启动脚本（xxxx_train.sh）示例如下。其中“MASTER_ADDR”和“NODE_RANK”必须保持该赋值。

```
#!/bin/bash

# MASTER_ADDR
MASTER_ADDR="${MA_VJ_NAME}-${MA_TASK_NAME}-${MA_MASTER_INDEX}.${MA_VJ_NAME}"
NODE_RANK="$RANK_AFTER_ACC"
NNODES="$MA_NUM_HOSTS"
NGPUS_PER_NODE="$MA_NUM_GPUS"
# self-define, it can be changed to >=10000 port
MASTER_PORT="39888"

# replace ${MA_JOB_DIR}/code/torch_ddp.py to the actutal training script
PYTHON_SCRIPT=${MA_JOB_DIR}/code/torch_ddp.py
PYTHON_ARGS=""

# set hccl timeout time in seconds
export HCCL_CONNECT_TIMEOUT=1800
```

```
# replace ${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python to the actual python
CMD="${ANACONDA_DIR}/envs/${ENV_NAME}/bin/python -m torch.distributed.launch \
    --nnodes=$NNODES \
    --node_rank=$NODE_RANK \
    --nproc_per_node=$NGPUS_PER_NODE \
    --master_addr $MASTER_ADDR \
    --master_port=$MASTER_PORT \
    $PYTHON_SCRIPT \
    $PYTHON_ARGS
"
echo $CMD
$CMD
```

步骤3 在主机上新建“config.yaml”文件。

“config.yaml”文件用于配置pod，代码示例如下。代码中的“xxxx_train.sh”即为步骤2修改的训练启动脚本。

```
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: yourvcjobname                # job名字，根据实际场景修改
  namespace: default           # 命名空间，根据实际场景修改
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 6                    # 节点数，根据实际场景修改，对应分布式训练使用的节点数
  schedulerName: volcano               # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
      - --rank-table-version=v2          # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
      - --publish-not-ready-addresses=true   # 保持不动，pod间互相通信使用及生成一些必要环境变量
  maxRetry: 1
  queue: default
  tasks:
    - name: "worker"  # 保持不动
      replicas: 6                        # 任务数，对于pytorch而言就是节点数，与minAvailable一致即可
      template:
        metadata:
          annotations:
            cabinet: "cabinet"  # 保持不动，开启tor-topo下发的开关
          labels:
            app: pytorch-npu   # 标签，根据实际场景修改
            ring-controller.cce: ascend-1980  # 保持不动
        spec:
          affinity:
            podAntiAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
                - labelSelector:
                    matchExpressions:
                      - key: volcano.sh/job-name
                        operator: In
                        values:
                          - yourvcjobname   # job名字，根据实际场景修改
                  topologyKey: kubernetes.io/hostname
          containers:
            - image:  swr.xxxxxx.com/xxxx/custom_pytorch_npu:v1            # 镜像地址，根据实际场景修改
              imagePullPolicy: IfNotPresent
              name: pytorch-npu          # 容器名称，根据实际场景修改
              env:
                - name: OPEN_SCRIPT_ADDRESS     # 开放脚本地址，其中region-id根据实际region修改，例如cn-
southwest-2
                  value: "https://mtest-bucket.obs.{region-id}.myhuaweicloud.com/acc/rank"
                - name: NAME
```

```
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: MA_CURRENT_HOST_IP                    # 保持不动，表示运行时当前pod所在节点的ip
              valueFrom:
                fieldRef:
                  fieldPath: status.hostIP
            - name: MA_NUM_GPUS     # 每个pod使用的NPU卡数，根据实际场景修改
              value: "8"
            - name: MA_NUM_HOSTS   # 参与分布式训练的节点数，与minAvailable一致即可
              value: "6"
            - name: MA_VJ_NAME # volcano job名称
              valueFrom:
                fieldRef:
                  fieldPath: metadata.annotations['volcano.sh/job-name']
            - name: MA_TASK_NAME #任务pod名称
              valueFrom:
                fieldRef:
                  fieldPath: metadata.annotations['volcano.sh/task-spec']
          command:
            - /bin/bash
            - -c
            - "wget ${OPEN_SCRIPT_ADDRESS}/bootstrap.sh -q && bash bootstrap.sh; export
RANK_AFTER_ACC=${VC_TASK_INDEX}; rank_acc=$(cat /tmp/RANK_AFTER_ACC 2>/dev/null); [ -n \"$
{rank_acc}\" ] && export RANK_AFTER_ACC=${rank_acc};export MA_MASTER_INDEX=$(cat /tmp/
MASTER_INDEX 2>/dev/null || echo 0); bash xxxx_train.sh" #  xxxx_train.sh换成实际训练脚本路径
          resources:
            requests:
              huawei.com/ascend-1980: "8"               # 每个节点的需求卡数，key保持不变。与
MA_NUM_GPUS一致
            limits:
              huawei.com/ascend-1980: "8"               # 每个节点的限制卡数，key保持不变。与
MA_NUM_GPUS一致
          volumeMounts:
            - name: ascend-driver              #驱动挂载，保持不动
              mountPath: /usr/local/Ascend/driver
            - name: ascend-add-ons             #驱动挂载，保持不动
              mountPath: /usr/local/Ascend/add-ons
            - name: localtime
              mountPath: /etc/localtime
            - name: hccn                        # 驱动hccn配置，保持不动
              mountPath: /etc/hccn.conf
            - name: npu-smi
              mountPath: /usr/local/bin/npu-smi
        nodeSelector:
          accelerator/huawei-npu: ascend-1980
        volumes:
          - name: ascend-driver
            hostPath:
              path: /usr/local/Ascend/driver
          - name: ascend-add-ons
            hostPath:
              path: /usr/local/Ascend/add-ons
          - name: localtime
            hostPath:
              path: /etc/localtime
          - name: hccn
            hostPath:
              path: /etc/hccn.conf
          - name: npu-smi
            hostPath:
              path: /usr/local/bin/npu-smi
        restartPolicy: OnFailure
```

步骤4 执行如下命令，根据“config.yaml”创建并启动pod。容器启动后会自动执行训练作业。

```
kubectl apply -f config.yaml
```

步骤5 执行如下命令，检查pod启动情况。如果显示“1/1 running”状态代表启动成功。


```
kubectl get pod
```

图 1-62 启动成功的回显

```
yourvcjobname1-worker-0                1/1     Running                0               110m
```

步骤6 执行如下命令，查看日志。日志显示如图所示表示成功执行动态路由。

```
kubectl logs {pod-name}
```

其中{pod-name}替换为实际pod名称，可以在**步骤5**的回显信息中获取。

图 1-63 成功执行动态路由的回显

```
2024-01-30 19:45:21,397 INFO: Wait for Topo file ready
2024-01-30 19:45:21,401 INFO: Wait for Rank table file ready
2024-01-30 19:45:21,401 INFO: Rank table file [redacted]jobstart_hccl.json (K8S generated) is ready for read
2024-01-30 19:45:21,402 INFO: Rank table file [redacted]jobstart_hccl.json (K8S generated) is old format.convert it to n
ew format start...
2024-01-30 19:45:21,402 INFO: Rank table file (V1) is generated
2024-01-30 19:45:21,402 INFO: Route plan begins. Current server 192[redacted]110
2024-01-30 19:45:21,410 INFO: Load in rank_file success. rank_file [redacted]jobstart_hccl.json
2024-01-30 19:45:21,410 INFO: route plan algorithm version 2
2024-01-30 19:45:21,414 INFO: save ranktable to file [redacted]jobstart_routeplan.json
2024-01-30 19:45:21,415 INFO: Route plan ends. Route plan acceleration True
2024-01-30 19:45:21,419 INFO: Route plan acc success.custom_dev is [['16', '0', '19[redacted]0.0'], ['17', '1', '19[redacted]5.0.
1'], ['18', '2', '1[redacted]0.2'], ['19', '3', '1[redacted]0.3'], ['20', '4', '192[redacted]4'], ['21', '5', '192[redacted]5'],
['22', '6', '192[redacted]0.6'], ['23', '7', '19[redacted]0.7']] ; custom_id is 19[redacted].2 ; current_node_rank is 2
```

说明

- 只有任务节点大于等于3的训练任务才能成功执行动态路由。
- 如果执行失败可以参考**故障排除：ranktable路由优化执行失败**处理。

----结束

故障排除：ranktable 路由优化执行失败

故障现象

容器日志有error信息。

可能原因

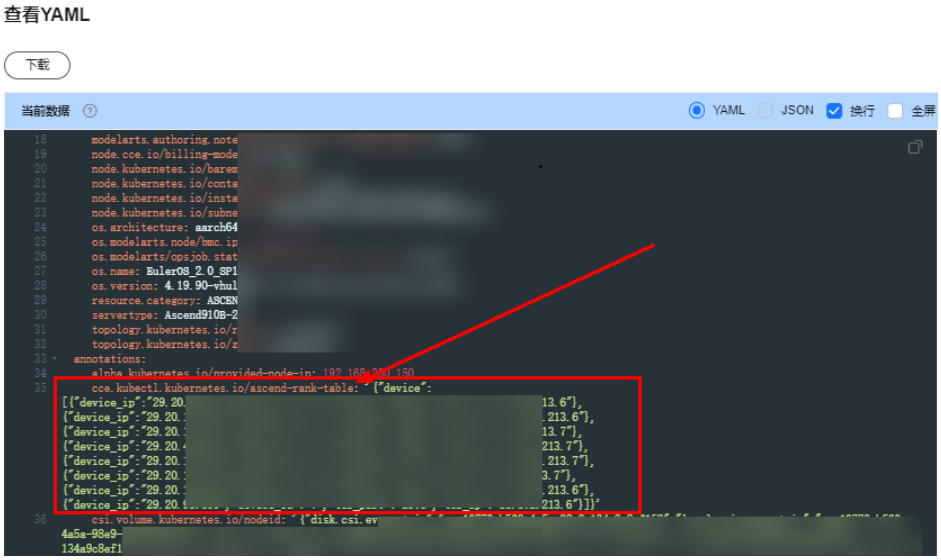
集群节点没有下发topo文件和ranktable文件。

操作步骤

1. 在ModelArts Lite专属资源池列表，单击资源池名称，进入专属资源池详情页面。
2. 在基本信息页面单击CCE集群，跳转到CCE集群详情页面。
3. 在CCE集群详情页，选择左侧导航栏的“节点管理”，选择“节点”页签。
4. 在节点列表，单击操作列的“更多 > 查看YAML”查看节点配置信息。
5. 查看节点的yaml文件里“cce.kubectl.kubernetes.io/ascend-rank-table”字段是否有值。

   如图所示，表示有值，节点已开启topo文件和ranktable文件的下发。否则，联系技术支持处理。

图 1-64 查看节点的 yaml 文件



1.4.4 在 lite 资源池上使用 Snt9B 完成推理任务

场景描述

本案例介绍如何在Snt9B上使用deployment部署在线在推理服务。

操作步骤

步骤1 拉取镜像。本测试镜像为bert_pretrain_mindspore:v1，已经把测试数据和代码打进镜像中。

```
docker pull swr.cn-southwest-2.myhuaweicloud.com/os-public-repo/bert_pretrain_mindspore:v1
docker tag swr.cn-southwest-2.myhuaweicloud.com/os-public-repo/bert_pretrain_mindspore:v1
bert_pretrain_mindspore:v1
```

步骤2 在主机上新建config.yaml文件。

config.yaml文件用于配置pod，本示例中使用sleep命令启动pod，便于进入pod调试。您也可以修改command为对应的任务启动命令（如“python inference.py”），任务会在启动容器后执行。

config.yaml内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yourapp
  labels:
    app: infers
spec:
  replicas: 1
  selector:
    matchLabels:
      app: infers
  template:
    metadata:
      labels:
        app: infers
    spec:
      schedulerName: volcano
      nodeSelector:
```

```
    accelerator/huawei-npu: ascend-1980
  containers:
  - image: bert_pretrain_mindspore:v1                # Inference image name
    imagePullPolicy: IfNotPresent
    name: mindspore
    command:
    - "sleep"
    - "10000000000000000000"
    resources:
      requests:
        huawei.com/ascend-1980: "1"            # 需求卡数，key保持不变。Number of required NPUs. The maximum value is 16. You can add lines below to configure resources such as memory and CPU.
      limits:
        huawei.com/ascend-1980: "1"            # 限制卡数，key保持不变。The value must be consistent with that in requests.
    volumeMounts:
    - name: ascend-driver              #驱动挂载，保持不动
      mountPath: /usr/local/Ascend/driver
    - name: ascend-add-ons           #驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: hccn                        #驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: npu-smi                          #npu-smi
      mountPath: /usr/local/bin/npu-smi
    - name: localtime                    #The container time must be the same as the host time.
      mountPath: /etc/localtime
  volumes:
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/bin/npu-smi
  - name: localtime
    hostPath:
      path: /etc/localtime
```

步骤3 根据config.yaml创建pod。

```
kubectl apply -f config.yaml
```

步骤4 检查pod启动情况，执行下述命令。如果显示“1/1 running”状态代表启动成功。

```
kubectl get pod -A
```

步骤5 进入容器，{pod_name}替换为您的pod名字（get pod中显示的名字），{namespace}替换为您的命名空间（默认为default）。

```
kubectl exec -it {pod_name} bash -n {namespace}
```

步骤6 激活conda模式。

```
su - ma-user   //切换用户身份
conda activate MindSpore //激活 MindSpore环境
```

步骤7 创建测试代码test.py。

```
from flask import Flask, request
import json
app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("------------ in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
```

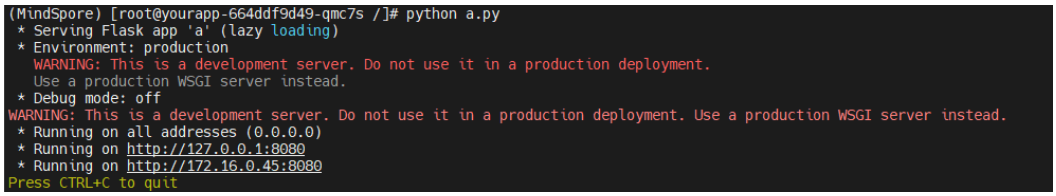


```
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

执行代码，执行后如下图所示，会部署一个在线服务，该容器即为服务端。

```
python test.py
```

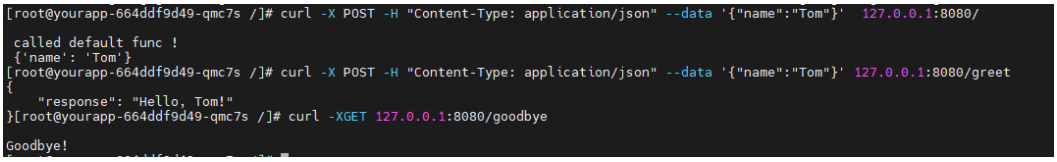
图 1-65 部署在线服务



步骤8 在XShell中新开一个终端，参考步骤5~7进入容器，该容器为客户端。执行以下命令验证自定义镜像的三个API接口功能。当显示如图所示时，即可调用服务成功。

```
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
curl -X GET 127.0.0.1:8080/goodbye
```

图 1-66 访问在线服务



说明

limit/request配置cpu和内存大小，已知单节点Snt9B机器为：8张Snt9B卡+192u1536g，请合理规划，避免cpu和内存限制过小引起任务无法正常运行。

----结束

1.4.5 AI 诊断

用户可以通过NCCl Test，测试节点GPU状态，并且测试多个节点间的通信速度。

操作步骤

步骤1 单击资源池名称，进入资源池详情。

步骤2 单击左侧“AI组件管理 > AI诊断”。

步骤3 单击“诊断”，选择“日志上传路径”和NCCL Test节点，其余参数可保持默认值或根据实际需求修改。

- 测试使用的最大数据：取值范围[1, 1024]，单位可选为“B”、“KB”、“MB”、“GB”“TB”。测试使用的最大数据须大于开始测试使用的最小数据。
- 开始测试使用的最小数据：取值范围[1, 1024]，单位可选为“B”、“KB”、“MB”、“GB”“TB”。
- 日志上传路径：AI诊断日志上传路径。
- 数据增加方式：当前支持乘法方式。
- 乘法系数：数值范围[2, 100]。
- 超过时间：数值范围[150, 3600]。
- NCCL Test节点名称列表：不可为空，且被选择的节点须为可用状态。

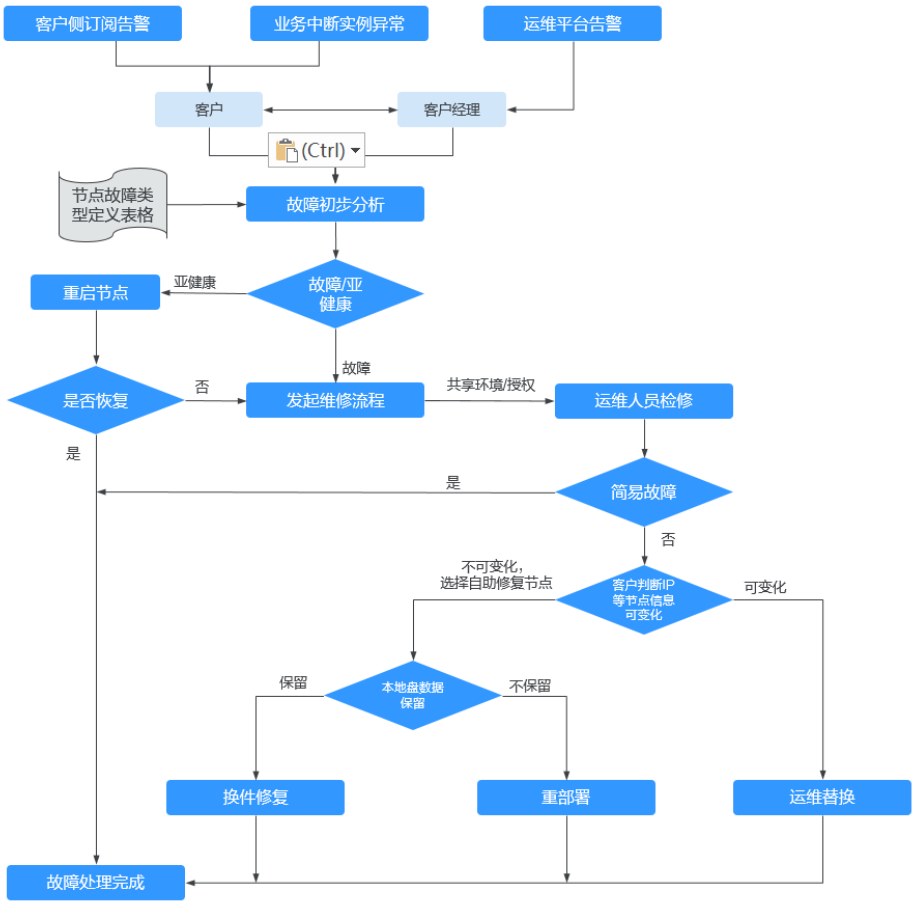
步骤4 单击“确认”，即可开始诊断。

----结束

1.4.6 节点故障定位

故障说明和处理建议

图 1-67 Lite 池故障处理流程



对于ModelArts Lite资源池，每个节点会以DaemonSet方式部署node-agent组件，该组件会检测节点状态，并将检测结果写到K8S NodeCondtition中。同时，节点故障指标默认会上报到AOM，您可在AOM配置告警通知。

当发生节点异常时，在故障初步分析阶段，您可先按表1-14识别是否为亚健康并自助进行处理，若不是，则为故障，请联系客户经理发起维修流程（若无客户经理可提交工单）。

表 1-14 节点故障类型定义

NodeCondition Type	分类	子类	异常中文描述	检测方法	处理建议
NT_NPU_DEVICE	NPU	其他	npu dcmi device异常。	NPU设备异常，昇腾dcmi接口中返回设备存在重要或紧急告警。	可能是亚健康，建议先重启节点，若重启节点后未恢复，发起维修流程。
NT_NPU_NET	NPU	链路	npu dcmi net异常。	NPU网络链接异常。	可能是亚健康，建议先重启节点，若重启节点后未恢复，发起维修流程。
NT_NPU_CARD_LOSE	NPU	掉卡	NPU卡丢失。	节点规格的NPU卡数和k8sNode中可调度卡数不一致。	可能是亚健康，建议先重启节点，若重启节点后未恢复，发起维修流程。
NT_NPU_OTHER	NPU	其他	NPU其他错误。	检测到的其他NPU错误，通常为不可自纠正的异常，请联系技术人员支持。	发起维修流程。
NT_NPU_ECC_COUNT	NPU	显存	NPU ECC次数达到维修阈值。	NPU的HBM总的多Bit Ecc隔离地址记录达到64个。	发起维修流程。
NT_NET_NTP_CHECK	Runtime	其他	ntp异常。	ntpd或者chronyd服务异常。	发起维修流程。

NodeCondition Type	分类	子类	异常中文描述	检测方法	处理建议
NT_KUBE_DISK_READONLY_CHECK	Runtime	其他	Kubelet硬盘只读	以下目录只读： /mnt/paas/kubernetes/kubelet	发起维修流程。

NodeCondition Type	分类	子类	异常中文描述	检测方法	处理建议
NT_GPU_SMI_ECC_CHECK	GPU	显存	GPU ECC错误。	通过nvidia-smi -a查询到存在Pending Page Blacklist为Yes的记录，或多比特Register File大于0。对于Ampere架构的GPU，存在以下场景： • 存在不可纠正的SRAM错误。 • 存在Remapping Failure记录。 • dmsg中存在Xid 95事件。 （参考**NVIDIA GPU Memory Error Management**） Ampere架构GPU显存错误分级： • L1: 可纠正ECC错误（单比特ECC错误），不影响业务。观测方式：nvidia-smi -a中查询到Volatile Correctable记录。 • L2: 不可纠正ECC错误（多比特ECC错误），当次业务受损，重启进程可恢复。观测方式：nvidia-smi -a中查询到Volatile Uncorrectable记录。 • L3: 错误未被抑制，可能影响后续业务，需要重置卡或重启节点。观测方式：Xid事件中包含95事件。(Remapped的Pending记录只作为提示，当业务空闲时进行卡重置触发重映射即可) • L4: 需要换卡，SRAM Uncorrectable>4或者Remapped Failed。	可能是亚健康，建议先重启节点，若重启节点后未恢复，发起维修流程。

NodeCondition Type	分类	子类	异常中文描述	检测方法	处理建议
NT_GPU_SMI_ERROR	GPU	其他	nvidia-smi返回信息中包含ERR。	通过nvidia-smi -a查询到ERR!，通常为硬件问题，如电源风扇等问题。	发起维修流程。
NT_GPU_SMI_RUNTIME	GPU	其他	nvidia-smi执行错误，超时或者不存在。	执行nvidia-smi退出码非0。	发起维修流程。
NT_GPU_SMI_ECC_COUNT	GPU	显存	ECC错误到达64次	通过nvidia-smi -a查询到Retired Pages中，Single Bit和Double Bit之和大于64。	发起维修流程。
NT_GPU_CARD_LOSE	GPU	掉卡	GPU卡丢失。	节点规格的GPU卡数和以下任意值不相等： 1. lspci可见GPU卡数。 2. nvidia-smi可见卡数。 3. k8s可调度卡数不相等。	发起维修流程。
NT_GPU_SMI_INFOROM_ERROR	GPU	其他	infoROM告警。	执行nvidia-smi的返回信息中包含“infoROM is corrupted”告警。	发起维修流程。
NT_GPU_OTHER	GPU	其他	GPU其他错误。	检测到的其他GPU错误，通常为硬件问题，请联系技术人员支持。	发起维修流程。
NT_NET_IB_CHECK	IB	链路	IB网卡异常。	ibstat查看网卡非Active状态。	可能是亚健康，建议先重启节点，若重启节点后未恢复，发起维修流程。

配置节点告警通知

节点故障指标(nt_npg)默认会上报到AOM，您可以在AOM配置短信、邮件等通知方式。

说明

以下步骤基于AOM1.0配置。

步骤1 登录AOM控制台

步骤2 在左侧导航栏选择“告警 > 告警规则”，在右上角单击“添加告警”。

步骤3 设置告警规则（以NPU掉卡为例）。

- 规则类型：选择阈值类告警。
- 监控对象：选择命令行输入。
- 命令行输入框：
  ```
  sum(nt_npg{type="NT_NPU_CARD_LOSE"} !=2) by (cluster_name, node_ip,type)
  ```

图 1-68 告警规则设置

- 告警条件：选择触发条件在1个监控周期内，如果平均值>=1达到连续1次时，产生重要告警。
- 告警通知（可选）：若需要将告警通过邮件、手机方式通知您，可在告警通知处，为此告警规则配置行动规则。若此处无行动规则，请新建告警行动规则。

----结束

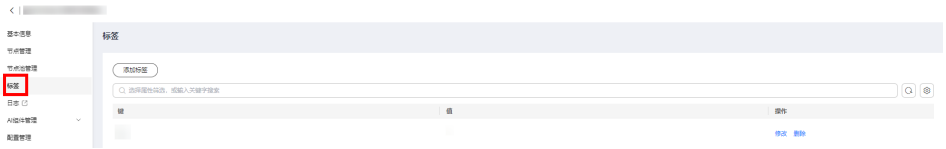
1.4.7 标签

通过给资源池添加标签，可以标识云资源，便于快速搜索资源池。

1. 在ModelArts管理控制台的左侧导航栏中选择“专属资源池 > 弹性集群”。
2. 在资源池列表中，单击资源池名称进入资源池详情页面。
3. 在资源池详情页面，单击“标签”页签查看标签信息。

   支持添加、修改、删除标签。标签详细用法请参见**ModelArts如何通过标签实现资源分组管理**。

图 1-69 标签



说明

最多支持添加20个标签。

1.4.8 扩缩容资源池

场景介绍

当专属资源池创建完成，使用一段时间后，由于用户AI开发业务的变化，对于资源池资源量的需求可能会产生变化，面对这种场景，ModelArts专属资源池提供了扩缩容功能，用户可以根据自己的需求动态调整。

说明

缩容操作可能影响到正在运行的业务，建议用户在业务空窗期进行缩容，或进入资源池详情页面，在指定空闲的节点上进行删除来实现缩容。

约束限制

- 只支持对状态为“运行中”的专属资源池进行扩缩容。
- 专属资源池不能缩容到0。

扩缩容专属资源池

资源池扩缩容有以下类型，分别为：

- 对已有规格增减节点数量
- 修改容器引擎空间大小

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“专属资源池 > 弹性集群”，默认进入“资源池”页签，查看资源池列表。
2. 增减节点数量

   单击某个资源池操作列的“扩缩容”对资源池进行扩缩容。

   在“专属资源池扩缩容”页面，设置“资源配置 > 可用区”，可用区可选择随机分配和指定AZ。设置完成后，单击“提交”，在弹出的确认框中单击“确定”完成修改。

   - 选择随机分配时，可通过增减“目标总节点数”实现扩缩容，请用户根据本身业务诉求进行调整。增加目标节点数量即表示扩容，减少目标节点数量即表示缩容。扩缩容完成后，节点的可用区分布由系统后台随机选择。
   - 选择指定AZ时，可指定扩缩容完成后节点的可用区分布。

   图 1-70 资源配置（单节点方式）

   若购买资源池时，节点数量采用整柜方式购买（部分规格支持），则在扩缩容时为整柜方式扩缩容，目标节点总数等于“数量*整柜”。“整柜”参数为创建资源池时选择，扩缩容时不可修改。用户通过增减“数量”来改变“目标总节点数”。

图 1-71 资源配置（整柜方式）

说明

用户增加节点数量时，可以通过指定节点计费模式，为资源池新创建的节点设置不同于资源池的计费模式，例如用户可以在包周期的资源池中创建按需的节点。若用户不指定该参数，创建的节点计费模式和资源池保持一致。

3. 新增节点池

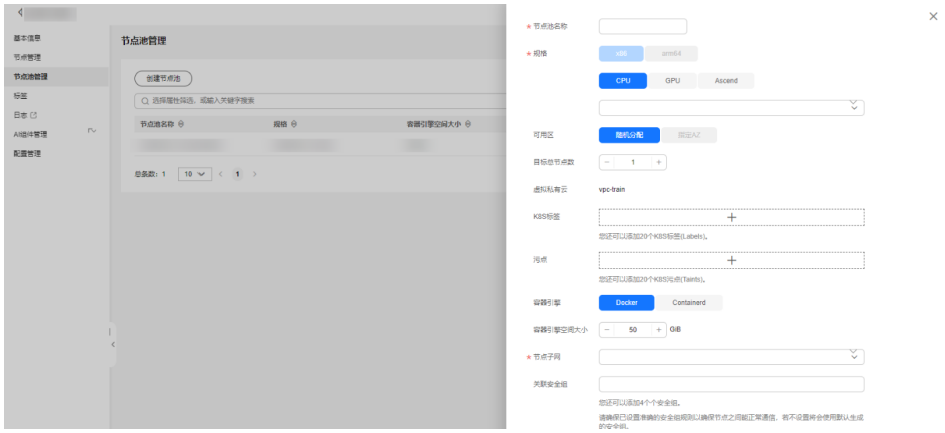
   若您需要更多的节点池，您可以通过以下操作创建新的节点池。

   方式一：在资源池详情页，单击“节点池管理”页签，单击创建节点池。

   方式二：在资源池列表页，单击某个资源池操作列下的“更多 > 新增节点池”，跳转至“节点池管理”页签，修改容器引擎空间大小。

   创建节点池相关参数请参见Lite Cluster资源开通。

图 1-72 新增节点池（单节点方式）



新增节点池时，部分规格支持整柜购买，若选中此类规格，可以通过下拉框选择整柜方式或单节点方式新增节点池。若选择整柜方式，目标总节点数等于“数量*整柜”，购买的节点总数为两者的乘积。

图 1-73 新增节点池（整柜方式）

说明

用户新增节点池时，可以通过指定节点的计费模式，为资源池新创建的节点设置不同于资源池的计费模式，例如用户可以在包周期的资源池中创建按需的节点。若用户不指定该参数，创建的节点计费模式和资源池保持一致。

4. 修改容器引擎空间大小

   若您需要更大的容器引擎空间，您可以通过以下操作调整容器引擎空间大小。

   - 对于新建的资源，支持在新建时指定容器引擎空间大小。

- 方式一：支持新建资源池时指定容器引擎空间大小，请参见**Lite Cluster资源开通**中“规格管理”参数下“高级选项”。
- 方式二：单击某个资源池名称，进入资源池详情，单击“节点池管理”页签，单击“创建节点池”，填写“容器引擎空间大小”后，单击“确认”。
- 方式三：单击某个资源池操作列下的“更多 > 新增节点池”，跳转至“节点池管理”页签，修改容器引擎空间大小。（仅包周期支持新增节点池）

– 对于存量的资源，支持修改容器引擎空间大小。

- 方式一：单击某个资源池名称，进入资源池详情，单击“节点池管理”页签，单击对应节点池操作列的“更新”，填写“容器引擎空间大小”后，单击“确认”。
- 方式二：单击某个资源池操作列的“扩缩容”，修改容器引擎空间大小。

须知

修改容器引擎空间大小仅作用在新建节点上，且会导致资源池内该规格下节点的dockerBaseSize不一致，可能会使得部分任务在不同节点的运行情况不一致。

图 1-74 修改容器引擎空间大小（节点池管理页签界面）

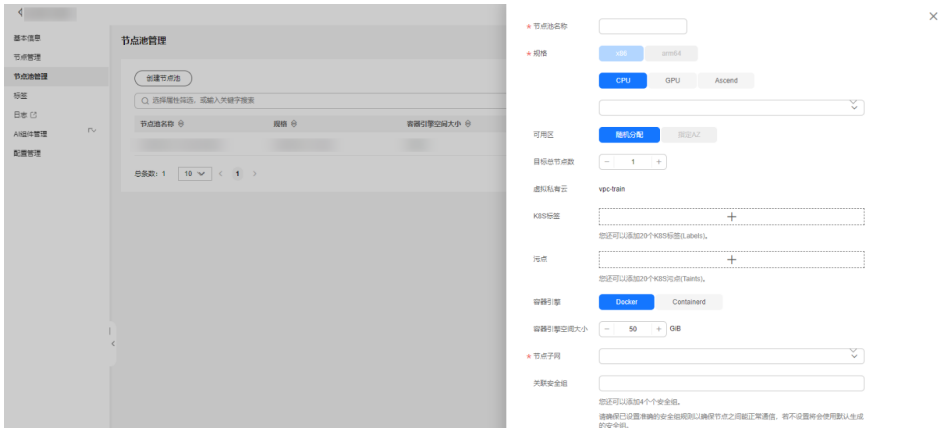


图 1-75 修改容器引擎空间大小（扩缩容界面）

您还可以在以上界面修改容器引擎类型。容器引擎是Kubernetes最重要的组件之一，负责管理镜像和容器的生命周期。Kubelet通过Container Runtime Interface

(CRI) 与容器引擎交互，以管理镜像和容器。其中Containerd调用链更短，组件更少，更稳定，占用节点资源更少，Containerd和Docker差异对比请见**容器引擎**。

若CCE集群版本低于1.23，仅支持选择Docker作为容器引擎。若CCE集群版本大于等于1.27，仅支持选择Containerd作为容器引擎。其余CCE集群版本，支持选择Containerd或Docker作为容器引擎。

1.4.9 资源池驱动升级

场景介绍

当专属资源池中的节点含有GPU/Ascend资源时，用户基于自己的业务，可能会有自定义GPU/Ascend驱动的需求，ModelArts面向此类客户提供了自助升级专属资源池GPU/Ascend驱动的能力。

驱动升级有两种升级方式：安全升级、强制升级。

说明

- 安全升级：不影响正在运行的业务，开始升级后会先将节点进行隔离（不能再下发新的作业），待节点上的存量作业运行完成后再进行升级，因需要等待作业完成，故升级周期可能比较长。
- 强制升级：忽略资源池中正在运行的作业，直接进行驱动升级，可能会导致运行中作业失败，需谨慎选择。

约束限制

- 专属资源池状态处于运行中，且专属池中的节点需要含有GPU/Ascend资源。
- 对于逻辑资源池，需要开启节点绑定后才能进行驱动升级，请提交工单联系华为工程师开启节点绑定。

驱动升级操作

1. 登录ModelArts管理控制台，在左侧导航栏中选择“专属资源池 > 弹性集群”，默认进入“资源池”页面。
2. 在资源池列表中，选择需要进行驱动升级的资源池“操作 > 驱动升级”。
3. 在“驱动升级”弹窗中，会显示当前专属资源池的驱动类型、节点数量、当前版本、目标版本和升级方式。
   - 目标版本：在目标版本下拉框中，选择一个目标驱动版本。
   - 升级方式：选择“升级方式”，可选择安全升级或强制升级。
   - 开启滚动：点击开启后，支持滚动升级的方式进行驱动升级。当前支持“按节点比例”和“按节点数量”两种滚动方式。
     - 按节点比例：每批次驱动升级的节点数量为“节点比例*资源池节点总数”。
     - 按节点数量：每批次驱动升级的节点数量为设置的节点数量。

     对于不同的升级方式，滚动升级选择节点的策略会不同：
     - 如果升级方式为安全升级，则根据滚动节点数量选择无业务的节点，隔离节点并滚动升级。
     - 如果升级方式为强制升级，则根据滚动节点数量随机选择节点，隔离节点并滚动升级。

说明

- 无业务节点定义：在资源池详情“节点管理”页签下，如果GPU/Ascend的可用数等于总数，则为无业务节点。

图 1-76 查看无业务节点

- 滚动驱动升级时，驱动异常的节点对升级无影响，会和驱动正常的节点一起升级。

图 1-77 驱动升级

4. 选择完成后，单击“确定”开始驱动升级。

1.4.10 专属资源池按需包周期互转

ModelArts支持“按需计费”和“包周期”的专属资源池互转。

约束限制

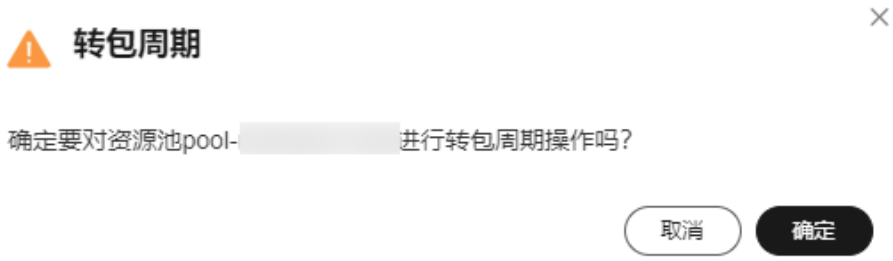
- 计费模式变更只支持以专属资源池为粒度进行整体变更，不支持以规格为粒度进行部分变更。

- 一个专属资源池的所有节点，必须为一种计费模式，全部为按需计费模式或全部为包年/包月计费模式。不支持部分节点为按需计费，部分节点为包年/包月计费。

按需转包周期

1. 登录ModelArts管理控制台，在左侧菜单栏中选择“AI专属资源池 > 弹性集群 Cluster”，进入资源池页面。
2. 在专属资源池列表中，选择“操作 > 更多”，单击“转包周期”，进入“转包周期”页面。
3. 在弹出的“转包周期”页面中单击“确定”，根据页面提示选择包年/包月的时长并勾选是否自动续费，完成支付后即可转换成功。

图 1-78 转包周期



包周期转按需

包周期转按需具体操作可参见“计费说明>变更计费模式>包年/包月转按需”。

1.4.11 设置续费策略

场景介绍

针对计费模式为“包年/包月”的资源池，ModelArts提供以下功能：

- 对于未开通自动续费的包周期资源池，支持开通自动续费。
- 对于已开通自动续费的包周期资源池，支持修改自动续费。
- 对于未开通/已开通自动续费的包周期资源池，支持手动续费。

约束限制

专属资源池状态处于“运行中”。

操作步骤

1. 登录ModelArts管理控制台，选择“AI专属资源池 > 弹性集群 Cluster”，进入资源池列表。
2. 在资源池列表中，选择目标资源池“操作 > 更多 > 设置续费策略”。
3. 在弹出的对话框中，单击“确认”，会跳转到费用中心的续费管理页面中。
4. 设置续费策略。
   - 对于未开通自动续费的包周期资源池，则在“手动续费项”页签下，选择目标资源池“操作 > 设为自动续费”，即可开通自动续费。

- 对于已开通自动续费的包周期资源池，则在“自动续费项”页签下，选择目标资源池“操作 > 修改自动续费”，即可修改自动续费（包括续费方式、续费时长、续费次数等）。
- 对于未开通/已开通自动续费的包周期资源池，则在其对应的页签下，选择目标资源池“操作 > 续费”，即可手动续费。

1.4.12 调整到期策略

场景介绍

针对计费模式为“包年/包月”的资源池，ModelArts提供支持调整到期策略，可以将其改为“到期转按需”或“到期不续费”。

约束限制

专属资源池状态处于“运行中”。

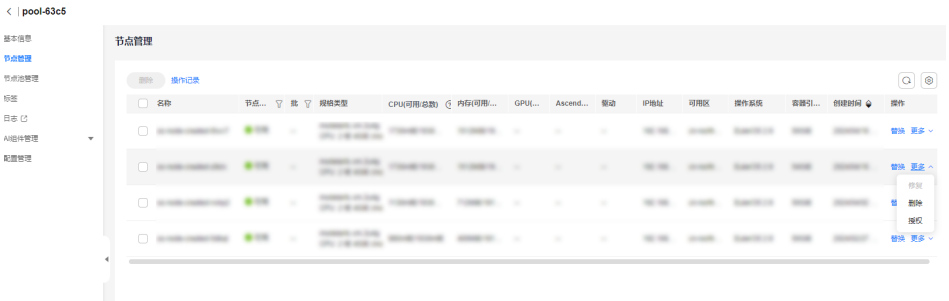
操作步骤

1. 登录ModelArts管理控制台，选择“AI专属资源池 > 弹性集群 Cluster”，进入资源池列表。
2. 在资源池列表中，选择目标资源池“操作 > 更多 > 调整到期策略”。
3. 在弹出的对话框中，单击“确认”，会跳转到费用中心的续费管理页面中。
4. 调整到期策略。
   - 若当前资源池未开通自动续费，则在“手动续费项”页签下，选择目标资源池“操作 > 更多 > 到期转按需”或“操作 > 到期不续费”。
   - 若当前资源池已开通自动续费，则在“自动续费项”页签下，选择目标资源池“操作 > 更多 > 到期不续费”或“操作 > 更多 > 到期不续费”。

1.4.13 Lite 资源池授权运维

华为云技术支持在故障定位和性能诊断时，部分运维操作需要用户授权才可进行。您可在资源池详情页的节点页签下，找到对应节点，在操作列单击“更多 > 授权”，在弹出的提示框中单击“确认”即可完成授权。

图 1-79 授权



说明

正常情况下，该授权按钮为置灰状态。当华为云技术支持发起运维申请后，按钮会变为可点状态。

在完成运维操作后，华为云技术支持会主动关闭已获得授权，无需您额外操作。

1.4.14 修复节点

说明

当前修复节点功能为白名单邀测阶段，如果您有试用需求，请联系技术支持。

若资源池节点发生硬件故障，可在资源池详情页的“节点管理”页签，查看对应故障节点。在对应节点的操作列的“更多”按钮中，修复按钮变为可单击状态，此时可单击“修复”按钮，对节点进行修复，待修复完成后，节点状态会变为“可用”。

当前支持“换件维修”和“重部署”两种修复方式：

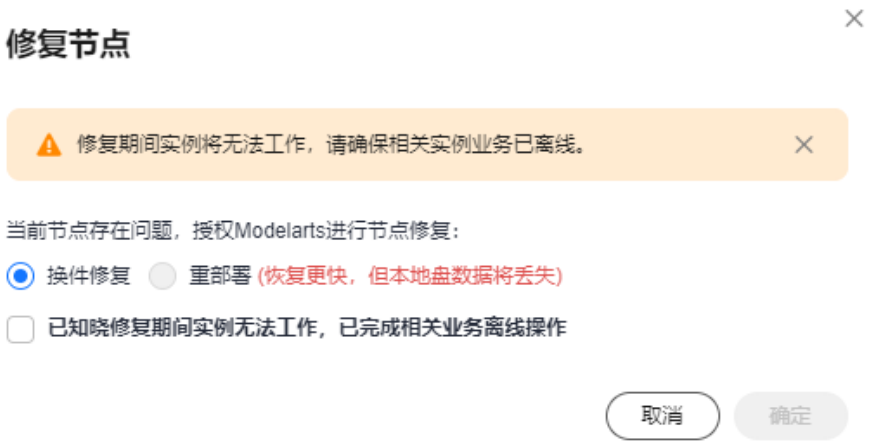
- 换件维修：通过更换硬件实现原地修复，修复耗时较长，对于非本地盘类故障，本地盘数据可以保留。

- 重部署：通过更换为新服务器实现修复，修复耗时较短，本地盘数据会丢失。

说明

- 修复期间实例将无法工作，请确保相关实例业务已离线。如果云服务器上的业务不可停止，请勿修复，并联系技术支持进行处理。
- 若选择了重部署修复方式，实例会立即关机并迁移到新服务器，本地盘数据会被清空，请提前做好业务迁移和数据备份。

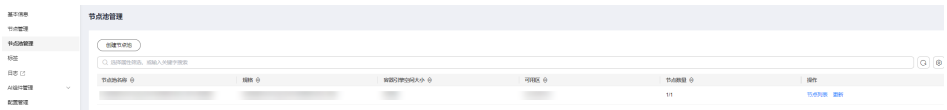
图 1-80 修复节点



1.4.15 节点池管理

在资源池详情页，单击“节点池管理”页签，您可以创建、更新和删除节点池。

图 1-81 节点池管理



- 创建节点池

  当您需要更多节点池时，可单击“创建节点池”新增节点池，相关参数请参见**Lite Cluster资源开通**。

- 查看节点列表

  当您想查看某一节点池下的节点相关信息，可单击操作列的“节点列表”，可查询节点的名称、规格及可用区。

- 更新节点池

  当您想更新节点池配置时，可单击操作列的“更新”，相关参数介绍请参见**Lite Cluster资源开通**。

  需注意，更新节点池配置时，不同参数的作用范围不同，例如K8S标签节点、污点的修改，会同步更新节点池下的存量节点。容器引擎空间大小、节点子网等参数，仅对新增的节点（扩容或重置生效），存量节点配置保持不变。

  图 1-82 更新节点池

- 删除节点池

  当有多个节点池时，支持删除节点池，此时在操作列会显示“删除”按钮，单击“删除”后输入“DELETE”并单击“确定”即可。

  每个资源池至少需要有一个节点池，当只有一个节点池时不支持删除。

1.5 Lite Cluster 资源释放

针对不再使用的k8s Cluster资源，可以释放资源，停止计费相关介绍请见**停止计费**。

说明

专属资源池资源释放后不可恢复，请谨慎操作。

删除按需计费的 k8s Cluster 资源

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 > 弹性集群 Cluster”，进入“弹性集群 Cluster”列表页面。

步骤3 在弹性集群列表中，单击操作列的“更多 > 删除”。

步骤4 在弹出的确认对话框中，输入“DELETE”，单击“确定”，删除资源池。

----结束

退订包年/包月的 k8s Cluster 资源

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 > 弹性集群 Cluster”，进入“弹性集群 Cluster”列表页面。

步骤3 在弹性集群列表中，单击操作列的“更多 > 退订”，跳转至“退订资源”页面。

步骤4 根据界面提示，确认需要退订的资源，并选择退订原因。

步骤5 确认退订信息无误后，勾选“资源退订后……”提示信息。

步骤6 单击“退订”，再次根据界面信息确认要退订的资源。

步骤7 再次单击“退订”，完成包年/包月资源的退订操作。

----结束

1.6 FAQ

1.6.1 特权池信息数据显示均为 0%如何解决?

问题现象

特权池基本信息页面数据均显示为0%（如CPU使用率、内存使用率、加速卡使用率、加速卡显存使用率）。

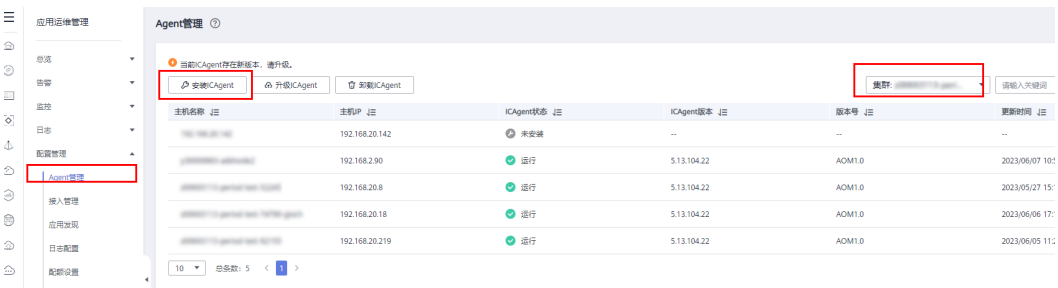
原因分析

原因是集群没有安装ICAgent。新建特权池时默认会安装ICAgent，可能由于用户自行卸载ICAgent，导致资源池数据显示异常。

处理方法

登录“应用运维管理”控制台，在“配置管理 > Agent管理”中，选择未安装ICAgent的集群，并单击“安装ICAgent”。

图 1-83 安装 ICAgent



说明

建议不要随意卸载ICAgent，否则会影响特权池详情页的参数显示。

1.6.2 资源池创建失败的原因与解决方法?

本文主要介绍在ModelArts资源池创建失败时，如何查找失败原因，并解决问题。

问题定位

您可以参考以下步骤，查看资源池创建失败的报错信息，并根据相应的解决方法解决问题：

1. 登录ModelArts控制台，单击弹性集群，单击资源池列表上方的“操作记录”查看创建失败的资源池。
2. 单击“操作记录”中失败状态的报错信息。

图 1-84 查看报错信息

解决方法

- **ModelArts全局配置的委托权限不足，导致创建失败?**
  解决方法请参见**ModelArts创建委托授权**。
- **申请的资源中包含受限购买的资源规格，导致购买失败?**
  当前modelarts.bm.npu.arm.8snt9b3.d为受限购买，需要提前联系ModelArts运营或提工单申请开通资源规格。

图 1-85 报错信息

- **ECS、EVS配额不足，导致创建失败?**

  集群所需的ECS实例数、内存大小、CPU核数和EVS硬盘大小资源会超出华为云默认提供的资源配额，因此需要申请扩大配额。解决方法请参见**申请扩大资源配额**。

图 1-86 报错信息（1）

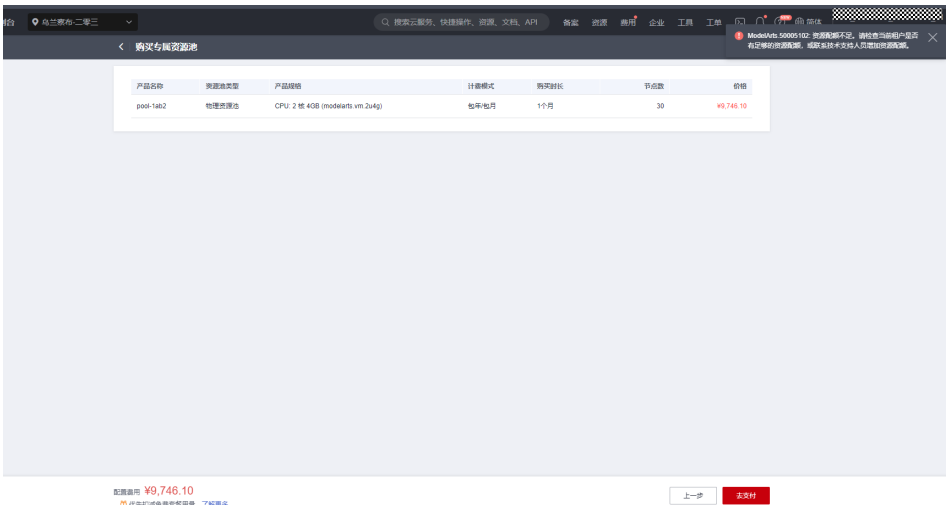
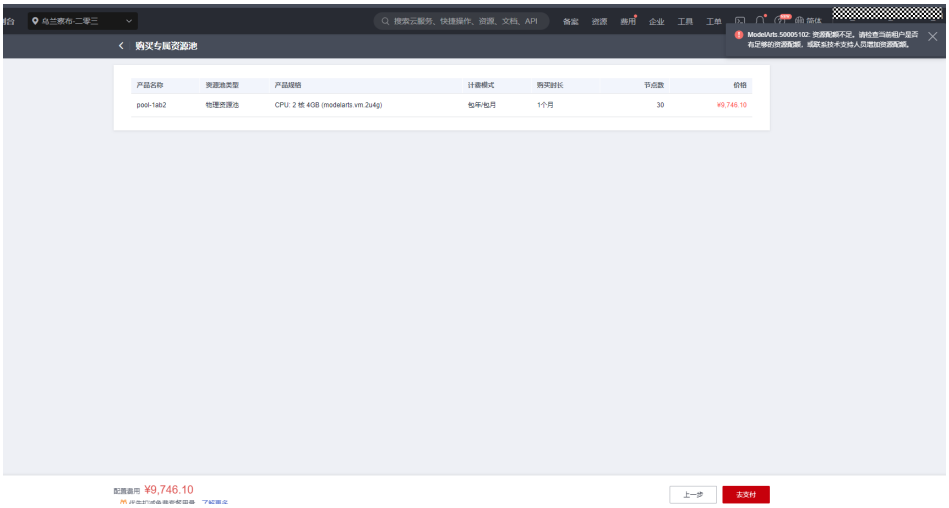


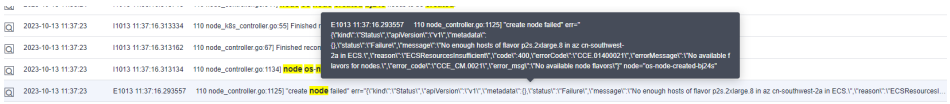
图 1-87 报错信息（2）



- **资源售罄或容量不足，导致创建失败?**

  减少资源池节点数量，或提工单给ModelArts申请更多资源。

图 1-88 报错信息



- **ECS、BMS节点创建失败?**

  查看资源池失败报错信息：

  - 包含错误码，如：Ecs.0000时，可查看**弹性云服务器 ECS_错误码**查看详细的错误信息及处理措施。
  - 包含错误码，如：BMS.0001时，可查看**裸金属服务器 BMS_错误码**查看详细的错误信息及处理措施。
  - 包含错误码，如：CCE.01400001时，可查看**云容器引擎 CCE_错误码**查看详细的错误信息及处理措施
  - 其他报错请提工单联系ModelArts运维进一步定位解决。

- **集群纳管节点失败?**

  查看资源池失败报错信息：

  - 查看资源池失败报错信息，包含错误码，如：CCE.01400001时，可查看**云容器引擎 CCE_错误码**查看详细的错误信息及处理措施。
  - 其他报错请提工单联系ModelArts运维进一步定位解决。

- **集群容器网段不足，导致创建失败?**

图 1-89 报错信息

用户可根据实际业务场景和节点规模，自定义配置容器网段，配置方式如下：

a. ModelArts Standard池，资源池创建阶段指定容器网段，根据实际需要设置更大的容器网段。

图 1-90 设置容器网段

b. ModelArts Lite池，选择/创建具有更大容器网段的CCE集群。CCE容器网段配置参见**网络规划**。

- **账号冻结导致创建失败?**

  查看资源池失败报错信息，存在"frozen deposit fail"，表示账号冻结导致资源创建失败。检查账号状态和资源欠费情况，账号解冻后重新购买资源。

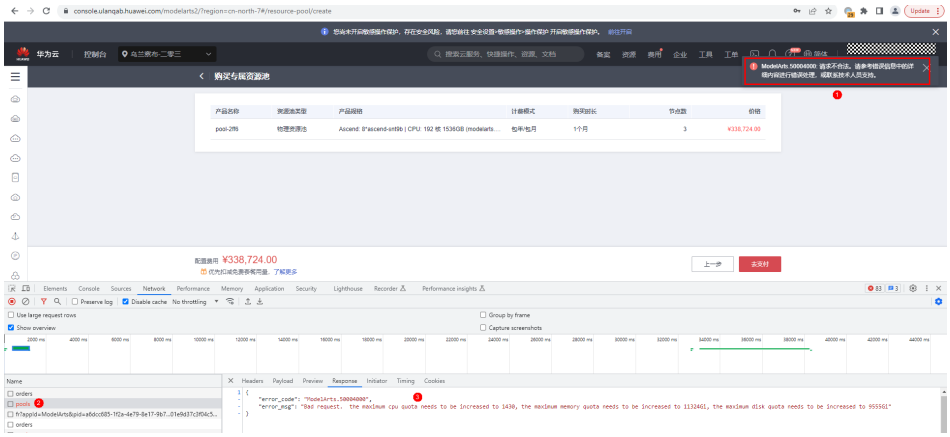
- **订单取消导致资源创建失败?**

  查看资源池失败报错信息，存在"the operation is canceled by customer"，表示资源池对应订单已取消，取消原因可能为超时未支付、用户自主取消，需重新购买。

- **其他错误**

  可通过F12查看浏览器请求信息，选择标红的pools接口，查看响应里的详细报错信息，如下图所示。通过错误提示修正输入参数后再次提交订单。

  图 1-91 报错信息

  如CCE集群不可用，请检查CCE集群版本和状态。报错信息如下：

  ```
  {
     "error_code": "ModelArts.50004000",
     "error_msg": "Bad request. spec.clusters[0].providerId: Invalid value: \"77f6f112-a631-11eb-8dae-0255ac100b0d\": the cluster 77f6f112-a631-11eb-8dae-0255ac100b0d is not available"
  }
  ```

2 弹性节点 Server

2.1 Server 简介

ModelArts Lite Server提供不同型号的xPU裸金属服务器，您可以通过弹性公网IP进行访问，在给定的操作系统镜像上可以自行安装加速卡相关的驱动和其他软件，使用SFS或OBS进行数据存储和读取相关的操作，满足算法工程师进行日常训练的需要。

2.2 Server 资源开通

2.2.1 开通流程

图 2-1 Server 资源开通流程图

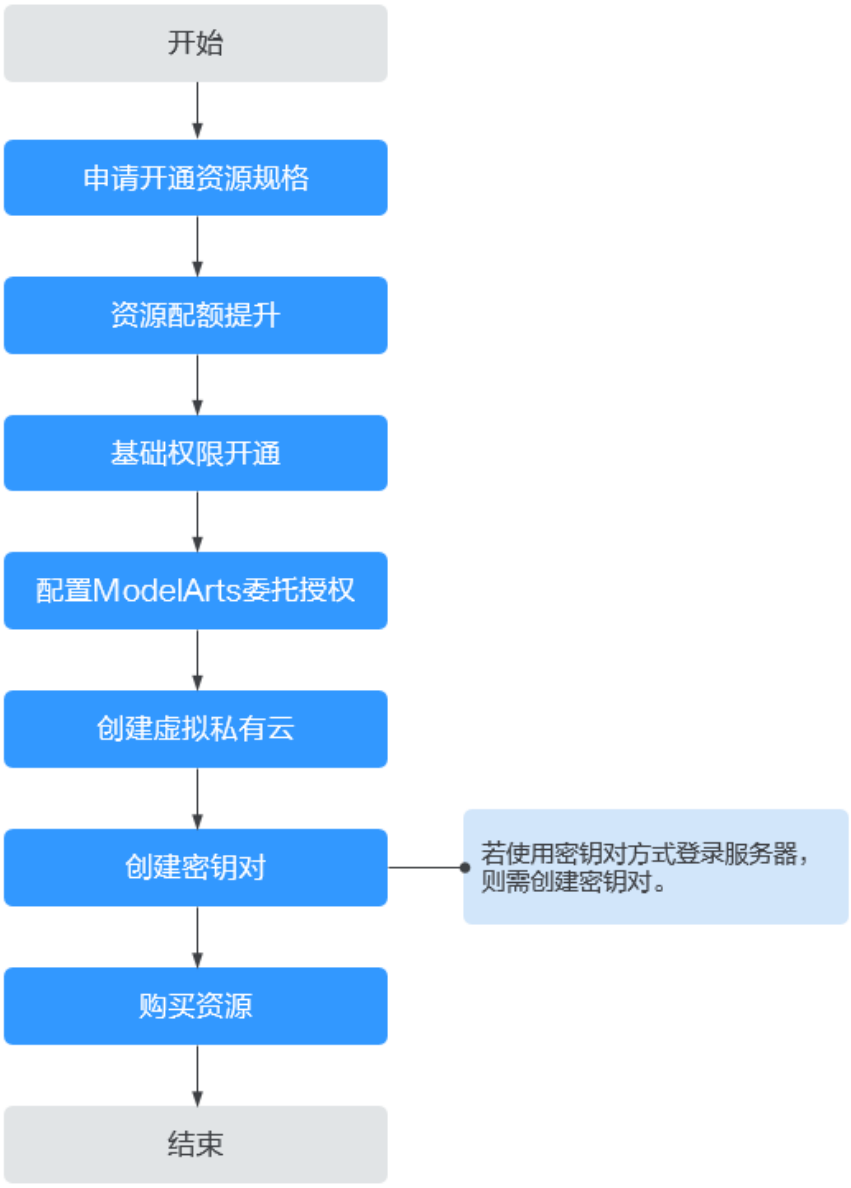


表 2-1 Server 资源开通流程

阶段	任务	参考
准备工作	1、申请开通资源规格。	**Step1：申请开通资源规格**
	2、资源配额提升。	**Step2：资源配额提升**
	3、基础权限开通。	**Step3：基础权限开通**

阶段	任务	参考
	4、配置ModelArts委托授权。	**Step4：配置ModelArts委托授权**
	5、创建虚拟私有云。	**Step5：创建虚拟私有云**
	6、创建密钥对。（可选，若为密码登录方式则不需要）	**Step6：创建密钥对**
购买Server资源	7、在ModelArts控制台上购买资源。	**购买Server资源**

2.2.2 准备工作

Step1：申请开通资源规格

请联系客户经理确认Server资源方案、申请开通资源规格（若无客户经理可提交工单）。

Step2：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），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

Step3：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”。

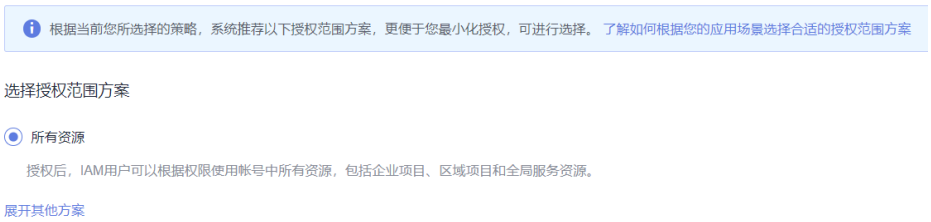
图 2-3 ModelArts FullAccess

步骤8 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）。

图 2-4 基础权限

步骤9 作用范围选择“区域级项目”，在下拉框中选择“所有资源 (包括未来在所有区域下创建的项目)”。

图 2-5 作用范围



步骤10 单击“确认”，完成基础权限开通。

----结束

Step4：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ModelaArts委托**。

Step5：创建虚拟私有云

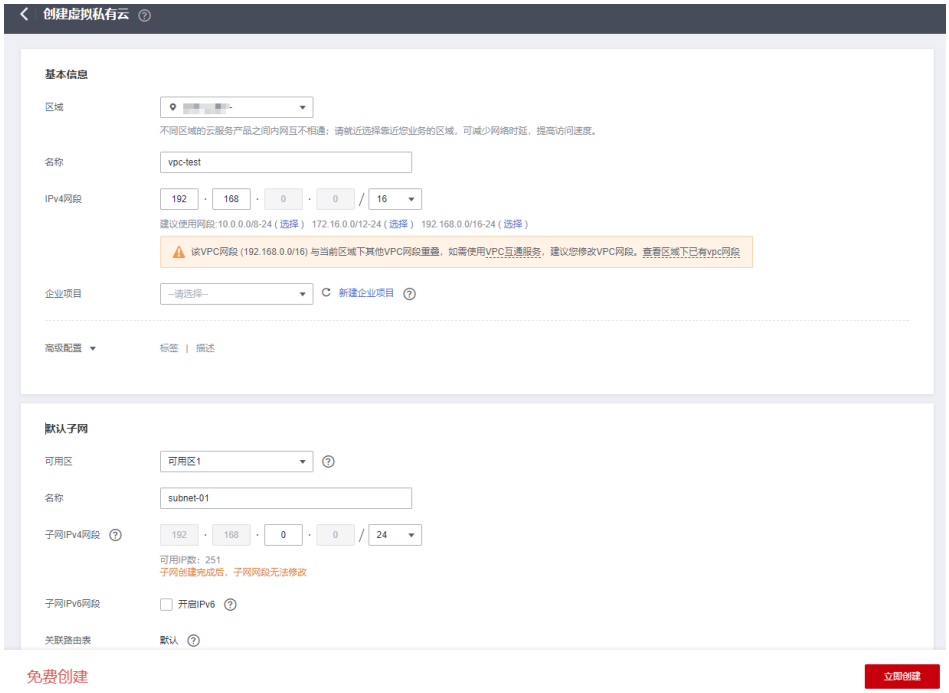
创建虚拟私有云需要登录管理员账号，IP地址段请根据现网情况合理规划。

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 2-6 新建虚拟私有云



----结束

Step6：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。

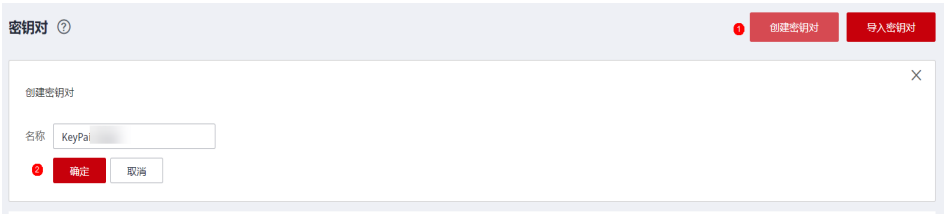
步骤2 在左侧导航栏中，选择“AI专属资源池 > 弹性节点 Server”，进入“Server”列表。

步骤3 单击“创建”，进入“创建Server”页面。

步骤4 单击“新建密钥对”。

步骤5 在新页面中单击右上角“创建密钥对”后，单击“确定”，并将密钥对保存至本地。

图 2-7 创建密钥对



----结束

2.2.3 购买 Server 资源

前提条件

已完成资源配额提升、基础权限配置、虚拟私有云创建、密钥对创建等工作，请见**准备工作**。

背景信息

当前支持的裸金属镜像请见**镜像介绍**，在创建Server实例时，根据所需镜像选择对应的规格。

操作步骤

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	Server方案请选择裸金属服务器。
计费模式	选择“按需计费”或“包年/包月”模式。
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。
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。

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**镜像介绍**。 • 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。

参数名称	说明
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，或者权限为企业主帐号的客户才可见。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》。

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。

图 2-8 支付订单

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，创建成功如下所示。

图 2-9 资源创建成功

----结束

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。
  1. 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。
  2. 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。
  3. 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**。

2.2.4 镜像介绍

2.2.4.1 GP Ant8 裸金属服务器支持的镜像详情

- 镜像1：Ubuntu20.04 With Nvidia Driver 525

  镜像名称：Ubuntu 20.04 x86 64bit SDI3 for Ant8 BareMetal with RoCE and NVIDIA-525 CUDA-12.0

  表 2-8 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	5.8-2.0.3.0-ubuntu20.04-x86_64
peer-memory-dkms	1.2-0
libnccl2	2.18.1
nccl-test	v.2.13.6
docker	20.10.23
RoCE路由配置	支持

- 镜像2：Ubuntu20.04 With Nvidia Driver 515

  镜像名称：Ubuntu 20.04 x86 64bit SDI3 for Ant8 BareMetal with RoCE and NVIDIA-515 CUDA-11.7

表 2-9 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	525.105.17
mlnx-ofed-linux	5.8-2.0.3.0-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

- 镜像3：Ubuntu20.04 纯净版
  镜像名称：Ubuntu 20.04 x86 64bit SDI3 for Ant8 BareMetal

  表 2-10 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	5.4.0-42-generic
架构类型	x86
mlnx-ofed-linux	5.3-1.0.0.1-all
RoCE路由配置	不支持自动配置，需创建后手动配置。

- 镜像4：EulerOS纯净版
  镜像名称：EulerOS2.10 x86 64bit sdi3 for Ant1 BareMetal

  表 2-11 镜像详情

软件类型	版本详情
操作系统	EulerOS 2.10 64bit

软件类型	版本详情
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。

2.2.4.2 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1：Ubuntu18.04 纯净版（仅限于北京四）
  镜像名称：Ubuntu-18.04-server-64bit-for-V100-BareMetal

  表 2-12 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
mlnx-ofed-linux	5.7-1.0.2.0

- 镜像2：Euler 2.3纯净版（仅限于上海一）
  镜像名称：ModelArts-Euler2.3_X86_GPU-ROCE-BMS-202211301010

  表 2-13 镜像详情

软件类型	版本详情
操作系统	EulerOS 2.0(SP3)
内核版本	3.10.0-514.44.5.10.h254
架构类型	x86
mlnx-ofed-linux	4.3-1.0.1.0

- 镜像3：Euler 2.9纯净版（仅限于北京四和上海一）
  镜像名称：Euler2.9-X86-for-V100-BareMetal

  表 2-14 镜像详情

软件类型	版本详情
操作系统	EulerOS 2.9 64bit

软件类型	版本详情
架构类型	x86

- 镜像4：Centos 7.3纯净版（仅限于北京一）
  镜像名称：CentOS 7.3 64bit for BareMetal WithIBDriver

  表 2-15 镜像详情

软件类型	版本详情
操作系统	CentOS 7.3 64bit
架构类型	x86

- 镜像5：Ubuntu 16.04纯净版（仅限于北京一）
  镜像名称：Ubuntu 16.04 server 64bit for BareMetal

  表 2-16 镜像详情

软件类型	版本详情
操作系统	Ubuntu 16.04 64bit
架构类型	x86

- 镜像6：CentOS 7.4 纯净版（仅限于华南广州）
  镜像名称：CentOS 7.4 64bit for BareMetal

  表 2-17 镜像详情

软件类型	版本详情
操作系统	CentOS 7.4 64bit
架构类型	x86

2.2.4.3 NPU Snt9 裸金属服务器支持的镜像详情

- 镜像：EulerOS纯净版
  镜像名称：ModelArts-Euler2.8_Aarch64_Snt9_C78

  表 2-18 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

软件类型	版本详情
架构类型	aarch64
mlnx-ofed-linux	21.0.2

2.2.4.4 GPU Ant1 裸金属服务器支持的镜像详情

- 镜像：EulerOS纯净版
  镜像名称：EulerOS2.10 x86 64bit sdi3 for Ant1 BareMetal

表 2-19 镜像详情

软件类型	版本详情
操作系统	EulerOS 2.10 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。

2.2.5 手动续费/自动续费包周期

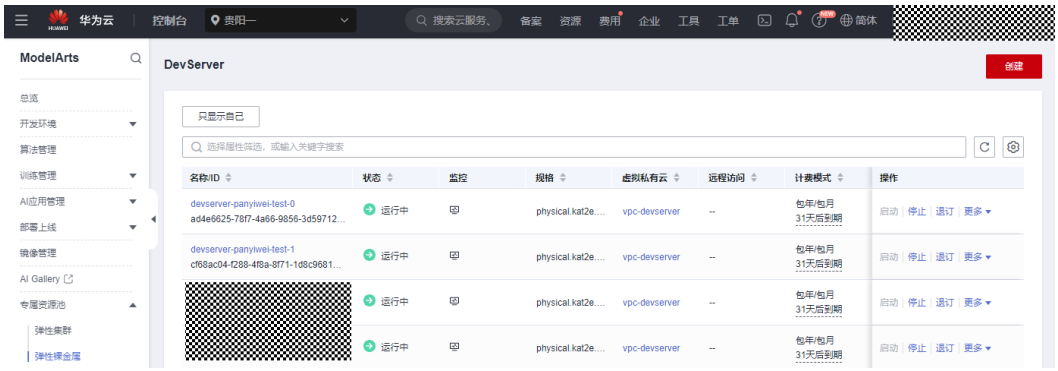
场景描述

本文旨在指导如何手动续费或者自动续费ModelArts Lite Server包周期服务器资源。

手动续费方法一：直接在 ModelArts Server 中续费指定服务器资源

步骤1 在ModelArts控制台，单击“AI专属资源池 > 弹性节点 Server”打开Server列表页面。

图 2-10 Server 列表页面

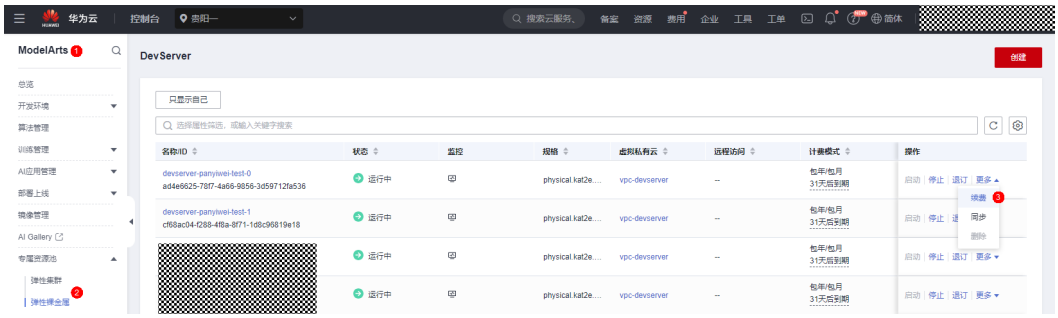


说明

此时如果显示需要配置委托，请联系您的账号管理员进行ModelArts委托权限配置。

步骤2 找到需要续费的服务器资源，在操作列单击“更多 > 续费”。

图 2-11 续费



步骤3 在续费页面中选择对应的续费时长，支付相应的订单即可。

图 2-12 选择对应的续费时长

----结束

手动续费方法二：在“订单管理”中续费指定服务器资源

步骤1 在ModelArts控制台，单击“AI专属资源池 > 弹性节点 Server”打开Server列表页面。

图 2-13 Server 列表页面

说明

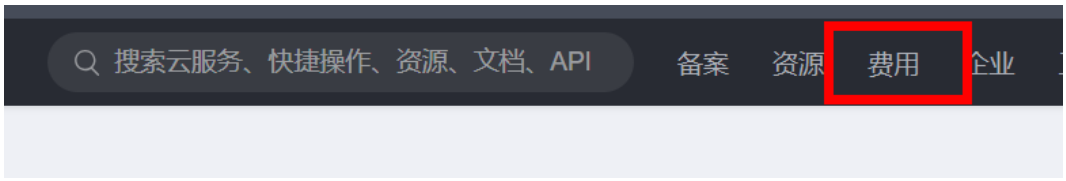
此时如果显示需要配置委托，请联系您的账号管理员进行ModelArts委托权限配置。

步骤2 复制需要续费的服务器资源的ID。

图 2-14 复制服务器资源的 ID

步骤3 在顶部菜单栏，单击“费用”，进入费用中心。

图 2-15 费用

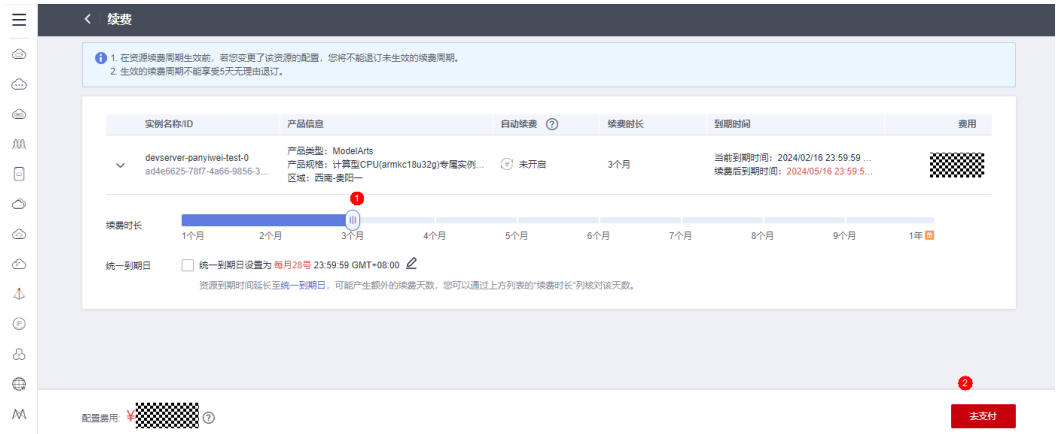


步骤4 单击“订单管理 > 续费管理”，在“手动续费项”页签中，搜索实例ID信息，在对应的服务器操作列，单击“续费”。

图 2-16 续费

步骤5 在续费页面中选择对应的续费时长，支付相应的订单即可。

图 2-17 选择对应的续费时长

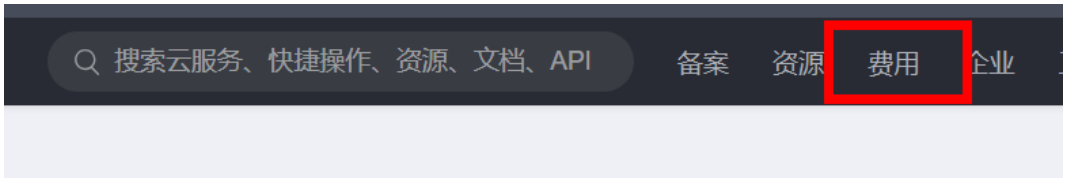


----结束

手动续费方法三：批量续费

步骤1 在顶部菜单栏，单击“费用”，进入费用中心。

图 2-18 费用



步骤2 单击“订单管理 > 续费管理”，在“手动续费项”页签中，在“产品类型/规格”列筛选ModelArts，然后勾选多个需要续费的资源后，单击“批量续费”。

图 2-19 批量续费

步骤3 在续费页面中选择对应的续费时长，支付相应的订单即可。

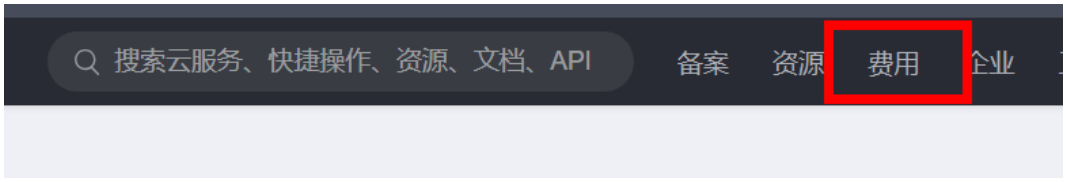
图 2-20 信息确认

----结束

自动续费

步骤1 在顶部菜单栏，单击“费用”，进入费用中心。

图 2-21 费用



步骤2 单击“订单管理 > 续费管理”，在“手动续费项”页签中，在“产品类型/规格”列筛选ModelArts，然后勾选一个或多个需要自动续费的资源后，单击“设为自动续费”。

图 2-22 设为自动续费

步骤3 设置自动续费次数。当前支持不限次数和自定义次数。

图 2-23 不限次数

图 2-24 自定义次数

----结束

2.3 Server 资源配置

2.3.1 使用前须知

ModelArts Lite Server存在以下使用限制：

- ModelArts Lite DevServer服务器操作系统升级/修改风险

  ModelArts Lite DevServer裸金属服务器，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。如果需要升级/修改，请**联系华为云技术支持**。相关高危命令如：apt-get upgrade。

- 部分ModelArts Lite DevServer服务器不支持重装系统

  如**表2-20**所示，弹性云服务器不支持重装操作系统，部分区域的裸金属服务器不支持重装操作系统，若您想重装操作系统，您可通过切换操作系统的方式解决。

  表 2-20 重装和切换操作系统能力介绍

功能	弹性云服务器	裸金属服务器	
		所在区域为华东二Region	所在区域为其他Region
重装操作系统	不支持	不支持	支持
切换操作系统	支持	支持	支持

- 重装或者切换操作系统后，ModelArts Lite Server服务器的EVS系统盘将不支持扩容

  服务器在进行过“重装或者切换操作系统”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系统盘无法扩容，并显示信息："当前订单已到期，无法进行扩容操作，请续订"。建议挂载数据盘EVS或挂载SFS盘等方式进行存储扩容。

表 2-21 扩容能力介绍

功能	弹性云服务器	裸金属服务器	
		所在区域为华东二Region	所在区域为其他Region
系统盘ID无变化下扩容	支持	支持	支持
系统盘ID有变化下扩容	不支持	不支持	不支持

2.3.2 配置流程

图 2-25 Server 资源配置流程图

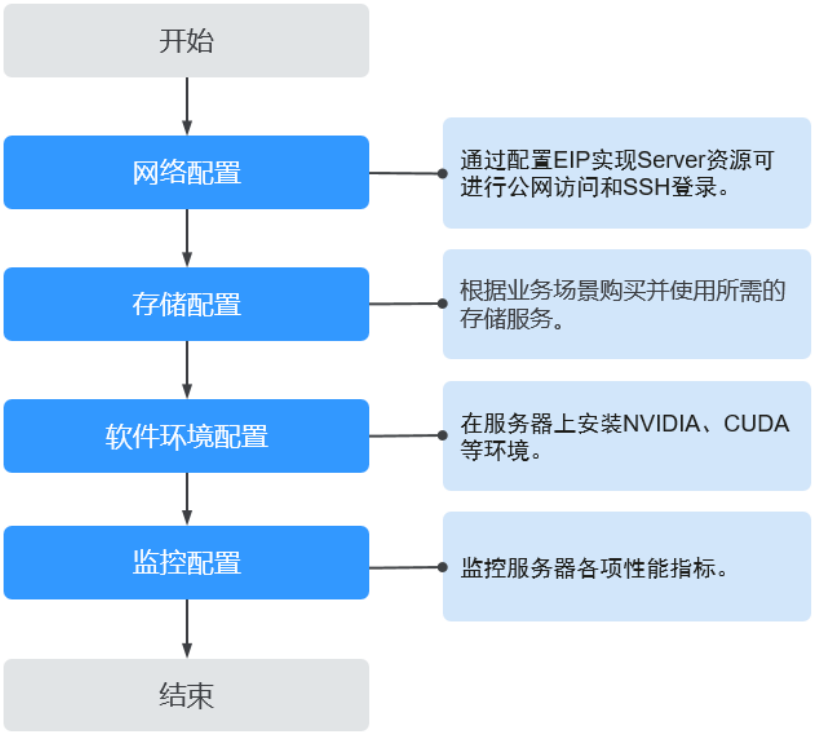


表 2-22 Server 资源配置流程

配置顺序	配置分类	配置任务	使用场景说明	是否必选操作
1	网络	**单EIP实现多节点的SSH登录和外网访问**	指导一个租户VPC配置一个EIP，通过NAT网关配置进行EIP资源共享，实现该VPC下的所有Server资源均可以通过该EIP进行公网访问和SSH登录。	必选。（二选一）
		单EIP绑定裸金属服务器	指导单EIP如何绑定在一台裸金属服务器上实现公网访问和SSH登录。	
		裸金属服务器支持IPV6解决方案	指导配置裸金属服务器实现支持IPV6。	可选。
2	存储	**裸金属服务器存储指南**	指导如何选择和使用存储服务。	必选。
3	环境配置	**环境配置**	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。	必选。
4	监控	**华为云CES服务官方监控方案**	指导如何使用华为云BMS+CES联合提供的裸金属服务器的指标监控方案。	可选。（二选一）

2.3.3 网络访问

2.3.3.1 单 EIP 实现多节点的 SSH 登录和外网访问

场景描述

Server创建后，需要进行网络配置后，才可进行SSH访问，本章节介绍网络配置步骤。一个租户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资源均可以通过该EIP进行公网访问和SSH登录。

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。

Step1：创建弹性公网 IP 与 NAT 网关

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 > 弹性IP”，进入弹性公网IP页面。

步骤3 单击“创建弹性公网IP”。

步骤4 参数配置可使用默认值，单击“立即申请”。

步骤5 单击“NAT网关”，进入公网NAT网关页面。

步骤6 单击“创建公网NAT网关”。

步骤7 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“立即创建”。

说明

虚拟私有云和子网和Server资源的网络保持一致。

图 2-26 创建公网 NAT 网关

----结束

Step2：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。

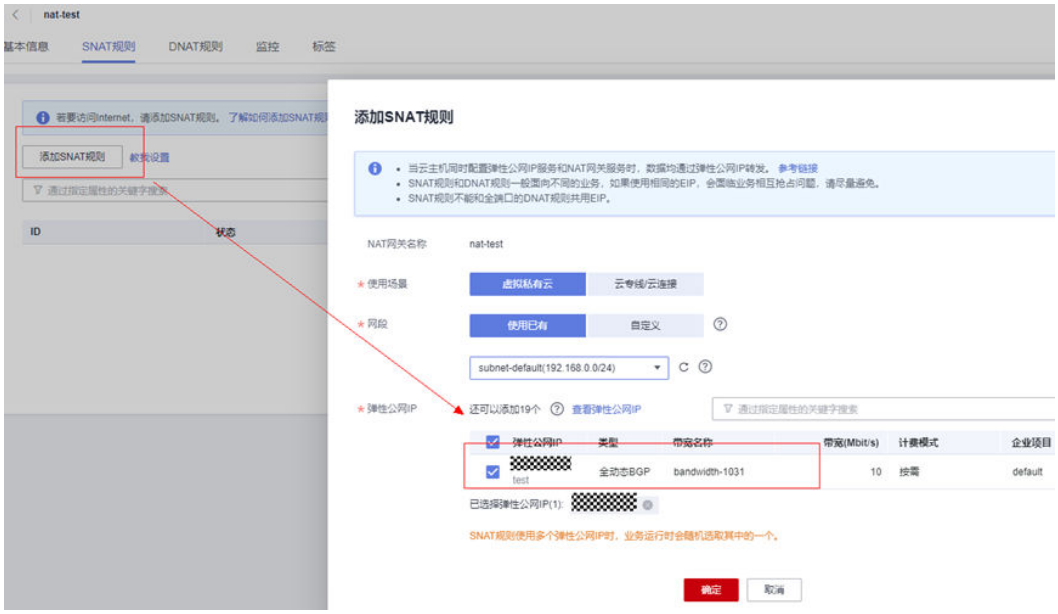
步骤1 公网NAT网关页面，单击创建的NAT网关名称，进入NAT网关详情页。

步骤2 在SNAT规则页签下，单击“添加SNAT规则”。

步骤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。

图 2-27 配置 SNAT 参数



步骤4 单击“确定”。

----结束

Step3：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。

步骤1 在DNAT规则页签下，单击“添加DNAT规则”。

步骤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”。
- 弹性公网IP：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。
- 私网IP：此处填写弹性节点Server的IP地址。可单击“查看可用云主机IP > 裸金属服务器”进行查看。
- 私网端口：端口号22。

图 2-28 配置 DNAT 参数 1

图 2-29 配置 DNAT 规则参数 2

步骤3 单击“确定”。

----结束

2.3.3.2 单 EIP 绑定裸金属服务器

弹性公网IP支持与单个裸金属服务器绑定实现公网访问和SSH登录，具体步骤可参考**绑定弹性公网IP至服务器**。

2.3.3.3 裸金属服务器支持 IPV6 解决方案

场景描述

本文旨在介绍裸金属服务器支持IPV6所需配置。裸金属服务器支持IPV6需要依赖规格、镜像、子网、安全组，对这些依赖项设置完成后，即可以在该裸金属服务上使用IPV6进行网络通信。

操作步骤

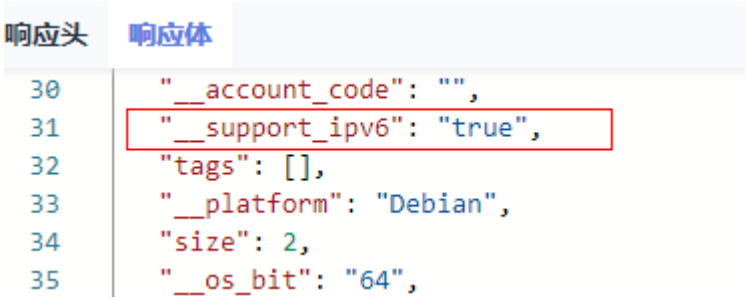
步骤1 确认裸金属服务器规格支持IPV6。需要确保要购买的华为云裸金属服务器规格支持IPV6，只有当该裸金属服务器规格支持IPV6时，才有必要进行后续IPV6依赖项的操作。

步骤2 确认镜像支持IPV6。如果未支持，需要更新该镜像属性，以支持IPV6。

查询方式请参考**查询镜像列表**。

调用API传入镜像ID，若响应体中有“__support_ipv6”："true"内容，则代表支持，否则代表不支持。

图 2-30 查询镜像是否支持 IPV6



步骤3 更新镜像属性，使得镜像支持IPV6。

更新步骤请参考**更新镜像信息**，关键代码如下：

```
{
  "op": "add",
  "path": "/__support_ipv6",
  "value": "true"
}
```

步骤4 子网支持IPV6。

单击子网，进入基本信息，在子网IPV6网段处，单击开启IPV6即可。

图 2-31 子网开启 IPV6

步骤5 安全组支持放开IPV6。

安全组默认不支持IPV6放通，因此需要添加规则，使得IPV6出方向和入方向规则都放通IPV6。

图 2-32 安全组支持放开 IPV6

步骤6 IPV6功能支持验证。

当创建成功后，进入到裸金属机器，通过ping6 IP地址，即该裸金属服务器已支持IPV6。

图 2-33 IPV6 功能支持验证

```
root@bms-88c3:~# ping6 2407:c080:1200:18ee:b4f3:84bb:[redacted]
PING 2407:c080:1200:18ee:b4f3:84bb:7d33:2734(2407:c080:1200:18ee:b4f3:84bb:7d33:[redacted]) 56 data bytes
64 bytes from 2407:c080:1200:18ee:b4f3:84bb:7d33:[redacted] icmp_seq=1 ttl=64 time=0.017 ms
64 bytes from 2407:c080:1200:18ee:b4f3:84bb:7d33:[redacted] icmp_seq=2 ttl=64 time=0.005 ms
64 bytes from 2407:c080:1200:18ee:b4f3:84bb:7d33:[redacted] icmp_seq=3 ttl=64 time=0.003 ms
64 bytes from 2407:c080:1200:18ee:b4f3:84bb:7d33:[redacted] icmp_seq=4 ttl=64 time=0.004 ms
64 bytes from 2407:c080:1200:18ee:b4f3:84bb:7d33:[redacted] icmp_seq=5 ttl=64 time=0.003 ms
64 bytes from 2407:c080:1200:18ee:b4f3:84bb:7d33:[redacted] icmp_seq=6 ttl=64 time=0.003 ms
```

----结束

2.3.4 存储

2.3.4.1 裸金属服务器存储指南

裸金属服务器结合OBS、SFS两种云存储服务，提供了多种场景下的存储解决方案。两种存储类型对比可参考**SFS和OBS的区别**。

1、弹性文件服务 SFS

提供按需扩展的高性能文件存储（NAS），可为云上多个裸金属服务器（BMS）提供共享访问。

- **购买指南**
- **挂载指南**
- **服务器重启后自动挂载指南**

2、对象存储服务 OBS

对象存储服务是一个基于对象的存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力，使用时无需考虑容量限制。

- 购买指南
- 使用指南

2.3.5 环境配置

2.3.5.1 GPU 裸金属服务器环境配置

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

场景描述

本文旨在指导如何在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。

操作步骤

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
```

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

说明

默认场景华为云Vnt1裸金属服务器在euler2.9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。

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

场景描述

本文旨在指导如何在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。

操作步骤

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

2.3.5.1.3 GP Vnt1 裸金属服务器的 Docker 模式环境搭建

场景描述

本文指导如何在Vnt1裸金属服务器上安装Docker环境（服务器系统需为Ubuntu18.04或Ubuntu20.04）。

操作步骤

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（可选，若在宿主机上不开发，可以忽略此步骤）。

```
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。

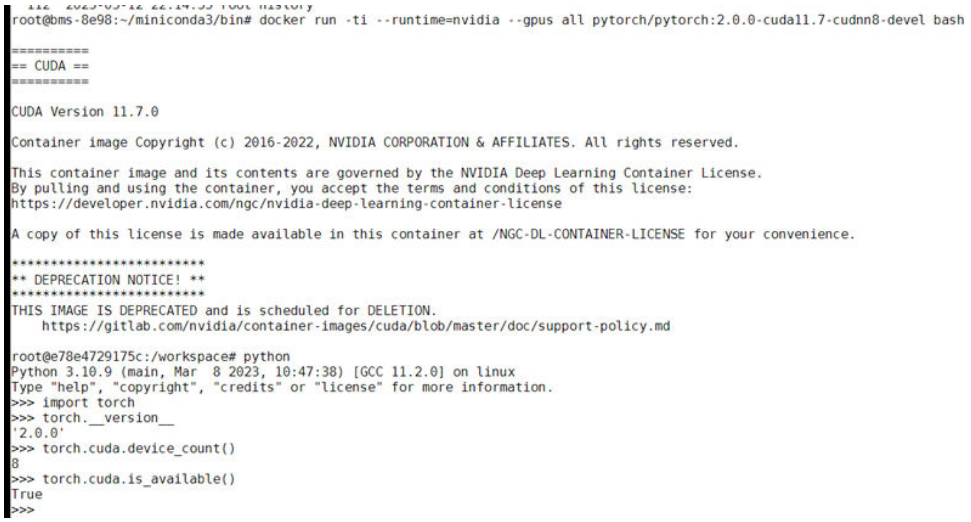
```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
&& curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/
keyrings/nvidia-container-toolkit-keyring.gpg
&& curl -s -L https://nvidia.github.io/libnvidia-container/$distribution/libnvidia-container.list |
 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 2-34 成功拉取镜像



----结束

2.3.5.1.4 GP Ant8 裸金属服务器 Ubuntu 20.04 安装 NVIDIA 525+CUDA 12.0

场景描述

本文旨在指导完成GP Ant8裸金属服务器装机和nccl-test测试。装机软件预览如下：

表 2-23 装机软件

软件类型	版本详情
预置操作系统	Ubuntu 20.04 server 64bit
驱动版本	525.105.17
nvidia-cuda	12.0
nvidia-fabricmanager	515.10.17（必须和驱动版本保持一致）
mlnx-ofed-linux	5.8-2.0.3.0/5.4-3.6.8.1(可选)
nvidia-peer-memory-dkms	1.2-0
nccl	libnccl2=2.16.2-1+cuda12.0 libnccl-dev=2.16.2-1+cuda12.0
nccl-test	v.2.13.6

前提条件

- 华为云Ant8裸金属服务器，使用IMS公共镜像Ubuntu 20.04 x86 64bit sdi3 for Ant8 BareMetal。镜像中仅预置IB驱动。NVIDIA驱动均未安装。
- 已联系客户经理完成RoCE网络配置。

操作步骤

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。

注意替换驱动版本为自己期望的。

```
sudo apt install nvidia-driver-525
```

步骤3 安装cuda-toolkit，可参考**CUDA Toolkit Downloads**。

```
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/12.0.0/local_installers/cuda-repo-ubuntu2004-12-0-local_12.0.0-525.60.13-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2004-12-0-local_12.0.0-525.60.13-1_amd64.deb
sudo cp /var/cuda-repo-ubuntu2004-12-0-local/cuda-*-keyring.gpg /usr/share/keyrings/
sudo apt-get update
sudo apt-get -y install cuda
```

步骤4 安装NCCL，可参考**NCCL Documentation**。

```
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 libnccl-dev
```

步骤5 安装nvidia-fabricmanager。

```
sudo apt install nvidia-fabricmanager-525
systemctl enable nvidia-fabricmanager
systemctl start 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
```

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
```

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 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：
```
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
```

说明

- 需要执行mpirun的节点到hostfile中的节点间有免密登录(**如何设置SSH免密登录**)。
- btl_tcp_if_include后面替换为主网卡名称。
- NCCL环境变量:

  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，可能丢包，而且后面的交换机不会支持v1，就跑不起来了。

  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算法的总线带宽不能这么算。

  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。

----结束

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

场景描述

本文旨在指导完成GP Ant8裸金属服务器装机和nccl-test测试。装机软件预览如下:

表 2-24 装机软件

软件类型	版本详情
预置操作系统	Ubuntu 20.04 server 64bit
驱动版本	515.10.01
nvidia-cuda	11.7
nvidia-fabricmanager	515.10.01（必须和驱动版本保持一致）
mlnx-ofed-linux	5.8-2.0.3.0/5.4-3.6.8.1(可选)
nvidia-peer-memory-dkms	1.2-0
nccl	libnccl2=2.14.3-1+cuda11.7 libnccl-dev=2.14.3-1+cuda11.7
nccl-test	v.2.13.6

前提条件

- 华为云Ant8裸金属服务器，使用IMS公共镜像Ubuntu 20.04 x86 64bit sdi3 for Ant8 BareMetal。镜像中仅预置IB驱动。NVIDIA驱动均未安装。
- 已联系客户经理完成RoCE网络配置。

操作步骤

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 2-35 查看 nccl

```
root@wangjianfeng:~#
root@wangjianfeng:~#
root@wangjianfeng:~# dpkg -l | grep nccl
ii  libnccl-dev                  2.14.3-1+cuda11.7                amd64        NVIDIA Collective Communication Library (NCCL) Development Files
ii  libnccl2                     2.14.3-1+cuda11.7                amd64        NVIDIA Collective Communication Library (NCCL) Runtime
root@wangjianfeng:~#
root@wangjianfeng:~#
```

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拷贝过来，需要拷贝即可：
  ```
  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 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：

```
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
```

说明

- 需要执行mpirun的节点到hostfile中的节点间有免密登录(**如何设置SSH免密登录**)。
- btl_tcp_if_include后面替换为主网卡名称。
- NCCL环境变量：

  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，可能丢包，而且后面的交换机不会支持v1，就跑不起来了。

  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算法的总线带宽不能这么算。

  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。

----结束

2.3.5.1.6 GP Ant8 裸金属服务器 Ubuntu 20.04 安装 NVIDIA 470+CUDA 11.3

场景描述

本文介绍如何配置NVIDIA驱动、CUDA和FabricManager， 并安装PyTorch2.0，最后验证是否正常运行。

- 服务器信息： GP Ant8裸金属服务器
- 操作系统：Ubuntu 20.04 server 64bit

- 选择安装环境相关版本： GPU驱动版本为470.182.03、CUDA版本为11.3.0

1、GPU 环境安装指南

步骤1 安装NVIDIA驱动。

```
wget https://us.download.nvidia.cn/XFree86/Linux-x86_64/470.182.03/NVIDIA-Linux-x86_64-470.182.03.run
chmod +x NVIDIA-Linux-x86_64-470.182.03.run
./NVIDIA-Linux-x86_64-470.182.03.run
```

步骤2 安装CUDA。

说明

不能选择Driver，否则会覆盖已安装的NVIDIA驱动。

```
wget https://developer.download.nvidia.com/compute/cuda/11.3.1/local_installers/
cuda_11.3.1_465.19.01_linux.run
chmod +x cuda_11.3.1_465.19.01_linux.run
./cuda_11.3.1_465.19.01_linux.run --toolkit --samples --silent
```

步骤3 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

fabricmanager版本一定要和nvidia驱动版本必须保持一致。

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/nvidia-
fabricmanager-470_470.182.03-1_amd64.deb
sudo dpkg -i ./nvidia-fabricmanager-470_470.182.03-1_amd64.deb
```

步骤4 验证fabricmanager安装结果。

验证驱动安装结果、启动fabricmanager服务并查看状态。

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

步骤5 汇总安装脚本。

在GP Ant8型实例中， 使用bash一键执行如下脚本，实现快速安装GPU环境。

```
wget https://us.download.nvidia.cn/XFree86/Linux-x86_64/470.182.03/NVIDIA-Linux-x86_64-470.182.03.run
chmod +x NVIDIA-Linux-x86_64-470.182.03.run
./NVIDIA-Linux-x86_64-470.182.03.run --silent --no-questions

wget https://developer.download.nvidia.com/compute/cuda/11.3.1/local_installers/
cuda_11.3.1_465.19.01_linux.run
chmod +x cuda_11.3.1_465.19.01_linux.run
./cuda_11.3.1_465.19.01_linux.run --toolkit --samples --silent

version=470.182.03
main_version=$(echo $version | awk -F '.' '{print $1}')
apt-get update
apt-get -y install nvidia-fabricmanager-${main_version}=${version}-*
```

----结束

2、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。

步骤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
```

步骤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
```

步骤3 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

2.3.5.1.7 GPU A 系列裸金属服务器更换 NVIDIA 和 CUDA

场景描述

当裸金属服务器预置的NVIDIA版本和业务需求不匹配时，需要更换NVIDIA驱动和CUDA版本。本文介绍华为云A系列GPU裸金属服务器（Ubuntu20.04系统）如何从“NVIDIA 525+CUDA 12.0”更换为“NVIDIA 515+CUDA 11.7”。

操作步骤

步骤1 卸载原有版本的NVIDIA和CUDA。

1. 查看使用apt包管理方式安装的nvidia软件包， 执行如下命令实现查看和卸载。

   ```
   dpkg -l | grep nvidia
   dpkg -l | grep cuda
   sudo apt-get autoremove --purge nvidia-*
   sudo apt-get autoremove --purge cuda-*
   ```

   以上命令可以卸载nvidia-driver、cuda、nvidia-fabricmanager、nvidia-peer-memory四个软件。

   但是如果nvidia和cuda是使用runfile(local)方式安装的，那么需要在下一步中再次卸载。

2. 若使用nvidia run包直接安装的驱动，需要找到对应的卸载命令。

   ```
   sudo /usr/bin/nvidia-uninstall
   sudo /usr/local/cuda-11.7/bin/cuda-uninstaller
   ```

3. 验证是否卸载完成。

   ```
   nvidia-smi
   nvcc -V
   dpkg -l | grep peer
   dpkg -l | grep fabricmanager
   dpkg -l | grep nvidia
   ```

步骤2 卸载nccl相关软件。

由于nccl和cuda是配套关系， 当cuda版本从12.0更换为11.7的时候，libnccl和libnccl-dev都需要更换为和cuda11.7匹配的版本。因此必须卸载掉原版本。

```
sudo apt-get autoremove --purge *nccl*
```

步骤3 删除原nccl-test的编译后文件。

由于nccl-test make编译也是基于当前cuda12.0版本的。 当cuda版本更换后，需要重新编译， 因此删除它。默认该文件在/root/nccl-tests直接删除即可。

步骤4 从内核中卸载nvidia相关的所有进程。

在安装nvidia驱动时，必须把内核中加载nvidia相关的进程卸载，否则会失败。具体操作请参考**GPU裸金属服务器更换NVIDIA驱动后执行nvidia-smi提示Failed to initialize NVML**。

说明

若遇到加载到内核的nvidia进程循环依赖，无法从内核中卸载nvidia，此时执行reboot命令重启服务器即可。

步骤5 安装NVIDIA-515和CUDA-11.7配套软件环境。具体步骤请参考**GP Ant8裸金属服务器Ubuntu 20.04安装NVIDIA 515+CUDA 11.7**。

----结束

2.3.5.1.8 GPU A 系列裸金属服务器 RoCE 性能带宽测试

场景描述

本文主要指导如何在GPU A系列裸金属服务器上测试RoCE性能带宽。

前提条件

GPU A系列裸金属服务器已经安装了IB驱动。（网卡设备名称可以使用ibstatus或者ibstat获取。华为云Ant8裸金属服务器使用Ubuntu20.04操作系统默认已经安装IB驱动。）

操作步骤

方法1：使用mlx硬件计数器，估算ROCE网卡收发流量

统计300s内流量，统计脚本如下：

```
x=$(cat /sys/class/infiniband/mlx5_2/ports/1/counters/port_rcv_data)
sleep 300
y=$(cat /sys/class/infiniband/mlx5_2/ports/1/counters/port_rcv_data)
res=$(($y-$x))
echo $res
```

上述获取的值*4/300 ，即为当前网卡的接收速率，单位Byte/s。

方法2：使用ib_write_bw测试RDMA的读写处理确定带宽

服务器A：服务端从mlx4_0网卡接收数据

```
ib_write_bw -a -d mlx5_0
```

服务器B：客户端向服务端mlx4_0网卡发送数据。

```
ib_write_bw -a -F 服务器A的IP -d mlx5_0 --report_gbits
```

图 2-36 服务器 A 执行结果

```
(base) root@devserver-gpu-[redacted]-roce-2:~# ib_write_bw -a -d mlx5_0

************************************
* Waiting for client to connect... *
************************************
---------------------------------------------------------------------------------------
                    RDMA_Write BW Test
Dual-port       : OFF          Device         : mlx5_0
Number of qps   : 1            Transport type : IB
Connection type : RC           Using SRQ      : OFF
PCIe relax order: ON
ibv_wr* API     : ON
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 0x00be PSN 0x101d60 RKey 0x189535 VAddr 0x007fc0683d9000
GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:31:55:128
remote address: LID 0000 QPN 0x00be PSN 0x342cce RKey 0x189535 VAddr 0x007fb1d0f2f000
GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:31:53:22
---------------------------------------------------------------------------------------
#bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
8388608    5000             73.96              64.82              0.000966
---------------------------------------------------------------------------------------
```

图 2-37 服务器 B 执行结果

```
(base) root@devserver-gpu-[redacted]-roce-3:~# ib_write_bw -a -F 192.168.102.236 -d mlx5_0 --report_gbits
---------------------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port       : OFF          Device         : mlx5_0
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API     : ON
 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 0x00be PSN 0x342cce RKey 0x189535 VAddr 0x007fb1d0f2f000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:31:53:22
 remote address: LID 0000 QPN 0x00be PSN 0x101d60 RKey 0x189535 VAddr 0x007fc0683d9000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:29:31:55:128
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[Gb/sec]    BW average[Gb/sec]   MsgRate[Mpps]
 2          5000           0.042403           0.020440             1.277489
 4          5000             0.14               0.14               4.327294
 8          5000             0.27               0.20               3.184282
 16         5000             0.54               0.50               3.925457
 32         5000             1.11               1.10               4.296528
 64         5000             1.97               0.76               1.483763
 128        5000             4.43               3.70               3.615876
 256        5000             8.81               5.88               2.872231
 512        5000             17.19              17.17              4.192633
 1024       5000             34.66              34.62              4.225802
 2048       5000             62.22              25.17              1.536477
 4096       5000             91.45              91.37              2.788505
 8192       5000             95.12              39.66              0.605207
 16384      5000             96.05              56.11              0.428078
 32768      5000             95.14              52.60              0.200663
 65536      5000             94.60              44.86              0.085564
 131072     5000             64.14              49.90              0.047587
 262144     5000             81.49              56.76              0.027066
 524288     5000             90.90              59.10              0.014090
 1048576    5000             88.51              65.10              0.007760
 2097152    5000             91.43              88.60              0.005281
 4194304    5000             90.91              86.47              0.002577
 8388608    5000             73.96              64.82              0.000966
---------------------------------------------------------------------------------------
```

2.3.5.1.9 NVIDIA 和 CUDA 驱动安装指南

场景描述

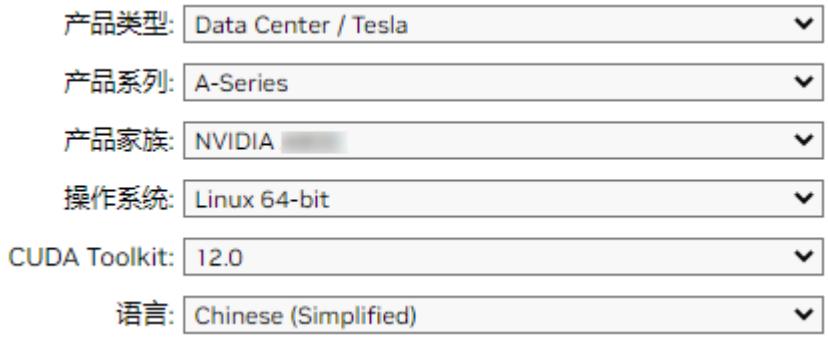
本文介绍如何华为云裸金属服务器中安装NVIDIA和CUDA驱动（以Ant8规格为例）。

1、安装 NVIDIA 驱动

步骤1 打开NVIDIA官方网站。

步骤2 根据Ant8的详细信息和您所需的cuda版本选择驱动

图 2-38 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

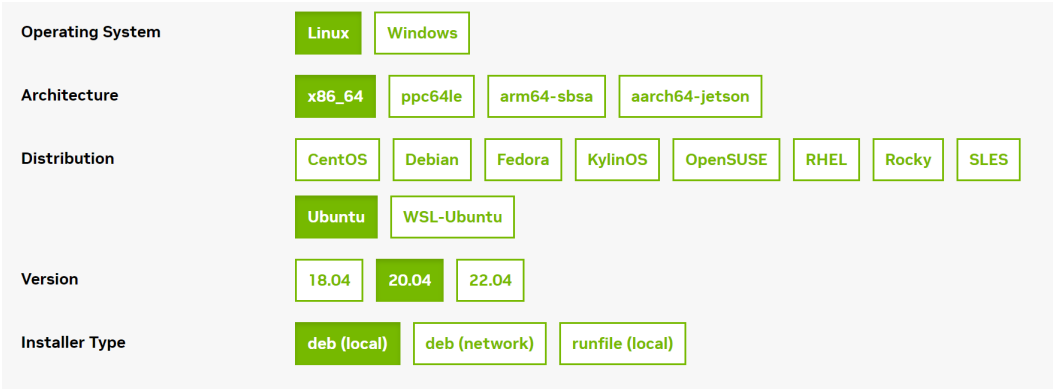
2、 安装 CUDA 驱动

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 2-39 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
```

----结束

常见问题

1、如何找到历史版本的CUDA?

请单击CUDA历史版本的**下载链接**查找所需的CUDA版本。

2、如何选择NVIDIA驱动?

在**NVIDIA驱动选择页面**，根据GPU型号选择NVIDIA驱动。

2.3.5.2 NPU Snt9B 裸金属服务器环境配置

2.3.5.2.1 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。

2.3.5.2.2 物理机环境配置

步骤 1：配置超时参数

SSH登录到机器后，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。

步骤1 首先通过“lsblk”查看是否有3个7T的磁盘未挂载，如下图所示为未挂载。

图 2-40 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 ~]#
```

若是下图所示，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 2-41 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
```

步骤2 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
```

步骤3 配置完成后，执行“df -h”可以看到新挂载的磁盘信息。

图 2-42 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
```

步骤4 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。

----结束

步骤 3：（可选）安装固件和驱动

步骤1 查看环境信息。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 2-43 查看固件和驱动版本

```
[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
```

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。

步骤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 2-44 查看机器操作系统版本及架构

```
[root@localhost ~]# arch
aarch64
[root@localhost ~]# cat /etc/os-release
NAME="EulerOS"
VERSION="2.0 (SP10)"
ID="euleros"
VERSION_ID="2.0"
PRETTY_NAME="EulerOS 2.0 (SP10)"
ANSI_COLOR="0;31"
```

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

步骤3 安装固件和驱动包。

1. 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 2-45 检查 npu-smi 工具

```
[root@devserver-bms-fd775372-833351 ~]# npu-smi info
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.rc3                 Version: 23.0.rc3                                             |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 0     910B2               | OK            | 92.7        49                0    / 0           |
| 0                         | 0000:C1:00.0  | 0           0    / 0          4152 / 65536         |
+===========================+===============+====================================================+
| 1     910B2               | OK            | 87.0        52                0    / 0           |
| 0                         | 0000:01:00.0  | 0           0    / 0          4152 / 65536         |
+===========================+===============+====================================================+
| 2     910B2               | OK            | 94.5        53                0    / 0           |
| 0                         | 0000:C2:00.0  | 0           0    / 0          4152 / 65536         |
+===========================+===============+====================================================+
| 3     910B2               | OK            | 92.9        51                0    / 0           |
| 0                         | 0000:02:00.0  | 0           0    / 0          4152 / 65536         |
+===========================+===============+====================================================+
| 4     910B2               | OK            | 91.9        53                0    / 0           |
| 0                         | 0000:81:00.0  | 0           0    / 0          4152 / 65536         |
+===========================+===============+====================================================+
| 5     910B2               | OK            | 93.1        54                0    / 0           |
| 0                         | 0000:41:00.0  | 0           0    / 0          4153 / 65536         |
+===========================+===============+====================================================+
| 6     910B2               | OK            | 92.2        52                0    / 0           |
| 0                         | 0000:82:00.0  | 0           0    / 0          4153 / 65536         |
+===========================+===============+====================================================+
| 7     910B2               | OK            | 92.2        54                0    / 0           |
| 0                         | 0000:42:00.0  | 0           0    / 0          4153 / 65536         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| No running processes found in NPU 0                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 1                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 2                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 3                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 4                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 5                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 6                                                            |
+===========================+===============+====================================================+
| No running processes found in NPU 7                                                            |
+===========================+===============+====================================================+
```

2. 工具检查正常后，进行固件和驱动安装。

   说明

   固件和驱动安装时，注意安装顺序：

   - 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。
   - 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：

   a. 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。

   b. 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：

a. 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
```

b. 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
```

c. 安装完成后，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 2-46 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
```

----结束

步骤 4：安装 docker 环境

步骤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 2-47 查看 docker 版本

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

步骤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，进行配置：
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤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 2-48 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**。

步骤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 2-49 docker 配置

```
{
        "runtimes":     {
                "ascend":       {
                        "path": "/usr/local/Ascend/Ascend-Docker-Runtime/ascend-docker-runtime",
                        "runtimeArgs":  []
                }
        },
        "default-runtime":      "ascend",
        "insecure-registries" : [ "ascendhub.huawei.com" ],
        "data-root": "/docker",
        "default-shm-size": "8G"
}
```

保存后，执行如下命令重启docker使配置生效：

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

----结束

步骤 5：（可选）安装 pip

步骤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
```

步骤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
```

步骤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
```

----结束

步骤 6：RoCE 网络测试

步骤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
```

步骤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
```

步骤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 2-50 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
```

步骤4 单机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：

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

若是单机多卡，则执行：

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

图 2-51 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
```

步骤5 多机ROCE网卡带宽测试。

1. 执行以下命令查看昇腾的RoCE IP：
   ```
   cat /etc/hccn.conf
   ```

图 2-52 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
```

2. 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 2-53 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 2-54 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 2-55 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

2.3.5.2.3 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。

步骤1 根据所需要的环境拉取Ascend+PyTorch或Ascend+MindSpore镜像，镜像地址详见ModelArts产品发布说明中的**基础镜像**章节：

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

步骤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
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
```

步骤3 进入容器：

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

步骤4 进入conda环境：

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

步骤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 2-56 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
```

步骤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 2-57 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]#
```

----结束

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。

步骤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
```

步骤2 设置容器root密码，根据提示输入密码：

```
passwd
```

图 2-58 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 ~]$
```

步骤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 2-59 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。

步骤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**。

----结束

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。

步骤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
```

步骤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
```

步骤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
```

----结束

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。

步骤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
```

步骤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
```

----结束

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：（可选）安装pip**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：git 操作

git clone和git lfs下载大模型可以参考如下操作。

步骤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
```

步骤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
```

步骤3 设置git配置去掉ssl校验。

```
git config --global http.sslVerify false
```

步骤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 huggingface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 2-60 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]#
```

----结束

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。

2.3.6 监控

2.3.6.1 CES 服务监控方案

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 2-25 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
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 2-61 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 2-62 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

2.3.6.2 DCGM 监控方案

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Docker最新发型版：

```
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：

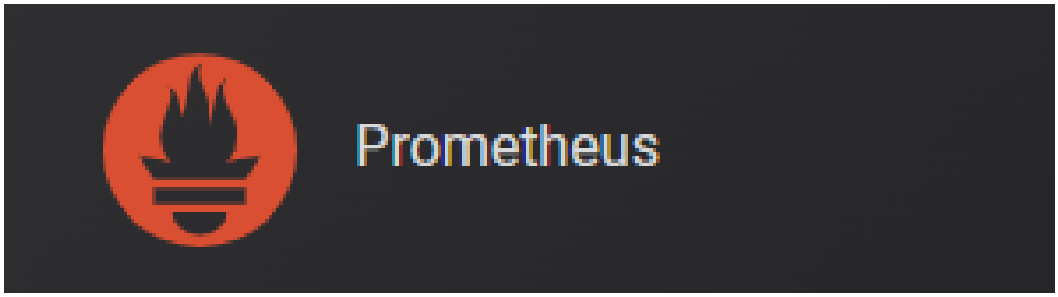
```
docker run -d -p 3000:3000 grafana/grafana-oss
```

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。

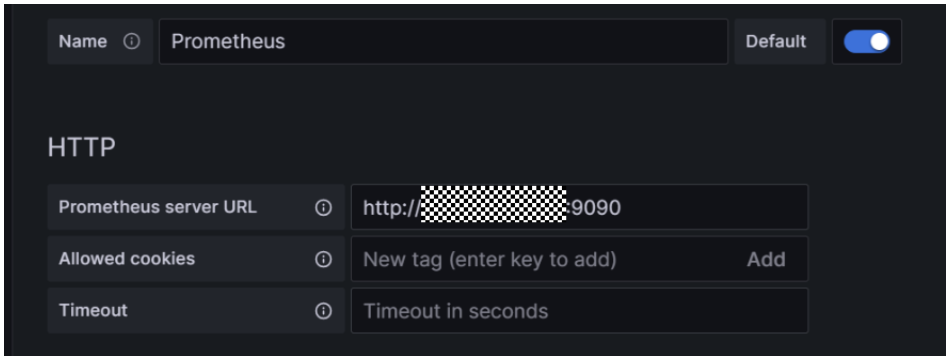
备注：xx.xx.xx.xx为Grafana的所在宿主机的IP地址

图 2-63 Prometheus



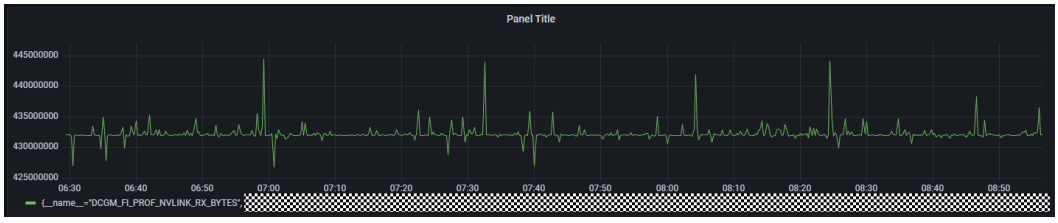
在HTTP的URL输入框中输入Prometheus的IP地址和端口号，单击Save&Test：

图 2-64 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 2-65 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**。

2.4 Server 资源使用

2.4.1 GP Ant8 裸金属服务器使用 Megatron-Deepspeed 训练 GPT2 并推理

场景描述

本文将介绍在GP Ant8裸金属服务器中， 使用Megatron-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是一个由NVIDIA开发的基于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NVIDIA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。

Step1 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，登录方式在华为云购买页面可以获取。
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，然后拷贝至服务器中，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。

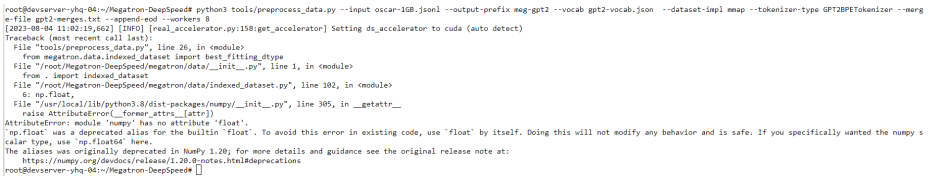
本实践中选择使用1GB 79K-record的JSON格式的OSCAR数据集。

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 2-66 预处理数据报错



4. 数据预处理完成标识。

图 2-67 数据预处理完成

```
Processed 77700 documents (1950.5485337214416 docs/s, 25.13939095827593 MB/s).
Processed 77800 documents (1949.3228818383702 docs/s, 25.122432021442663 MB/s).
Processed 77900 documents (1950.4971024454953 docs/s, 25.14053954202996 MB/s).
Processed 78000 documents (1951.4221225812407 docs/s, 25.14771264931429 MB/s).
Processed 78100 documents (1950.9776825402894 docs/s, 25.140942950000856 MB/s).
Processed 78200 documents (1949.7230206179488 docs/s, 25.122084117198362 MB/s).
Processed 78300 documents (1951.864504443268 docs/s, 25.149179100644623 MB/s).
Processed 78400 documents (1953.915315616835 docs/s, 25.171079961861405 MB/s).
Processed 78500 documents (1953.3149970708835 docs/s, 25.159395115131833 MB/s).
Processed 78600 documents (1947.1849552182766 docs/s, 25.116209914586644 MB/s).
Processed 78700 documents (1949.2702646176806 docs/s, 25.144594511179115 MB/s).
Processed 78800 documents (1951.3745402099773 docs/s, 25.178304942726875 MB/s).
Processed 78900 documents (1952.9719405469252 docs/s, 25.193807775649628 MB/s).
Processed 79000 documents (1950.0766282677068 docs/s, 25.172163738301247 MB/s).
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
```

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

Step2 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 2-68 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
   ```

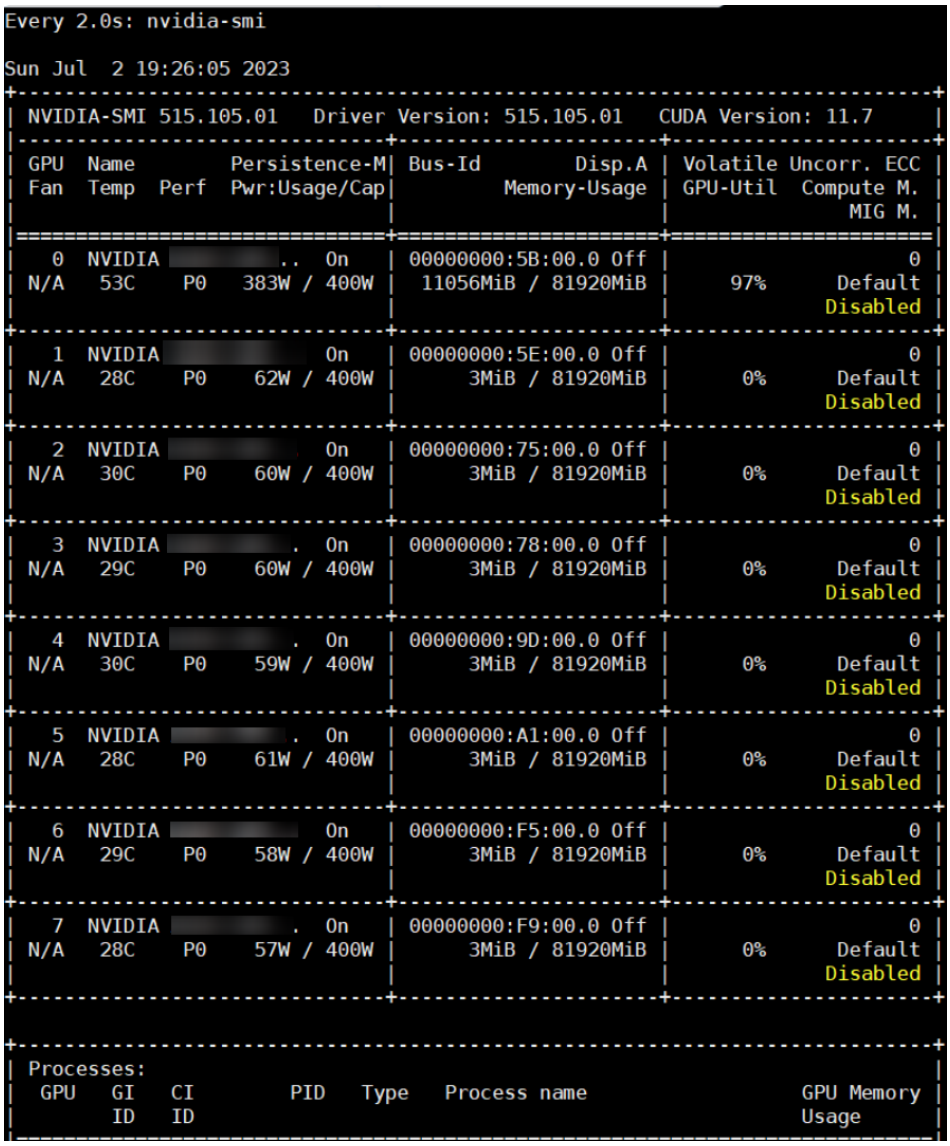
   如果显示如下信息， 表示模型训练完成。

   图 2-69 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 2-70 GPU 利用率



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 2-71 模型 checkpoint

```
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed# ls /workspace/Megatron-DeepSpeed/checkpoints/gpt2
iter_0000500  iter_0001500  iter_0002500  iter_0003500  iter_0004500  latest_checkpointed_iteration.txt
iter_0001000  iter_0002000  iter_0003000  iter_0004000  iter_0005000
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
```

----结束

Step3 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：

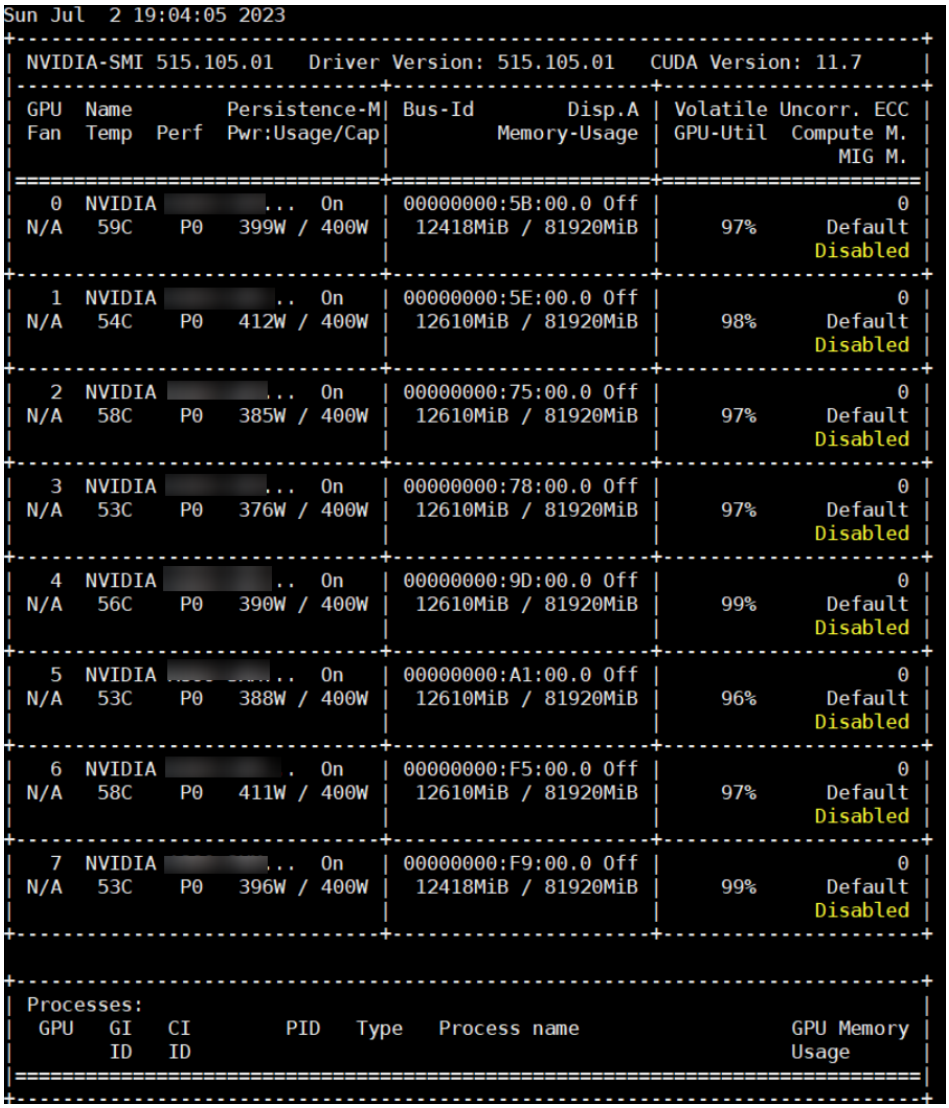
```
global_batch_size = 64
micro_batch_size = 4
data_parallel_size = 8
```

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 2-72 GPU 利用率



----结束

Step4 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 2-73 生成文本完成信息

```
Loading extension module fused_mix_prec_layer_norm_cuda...
>>> done with compiling and loading fused kernels. Compilation time: 1.622 seconds
building GPT model ...
 loading checkpoint from checkpoints/gpt2 at iteration 5000
 checkpoint version 3.0
  successfully loaded checkpoint from checkpoints/gpt2 at iteration 5000
Avg s/batch: 20.582671880722046
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
root@megatron-deepspeed-0001:/workspace/Megatron-DeepSpeed#
```

3. 查看模型生成的文本文件。

```
cat unconditional_samples.json
```

回显信息如下：

图 2-74 文件信息

步骤2 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。

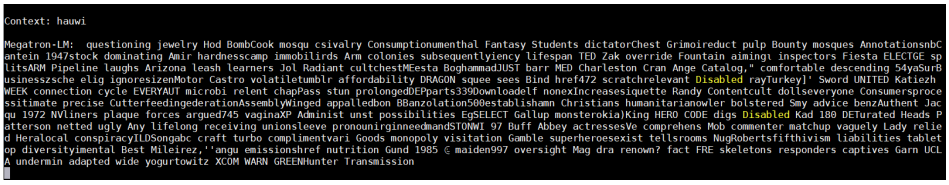
```
bash interactive_text.sh
```

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 2-75 模型输出文本信息



----结束

2.4.2 启动/停止实例

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“操作”列的“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“操作”列的“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

2.4.3 同步裸金属服务器状态

当用户在云服务器页面修改了裸金属服务器状态后，可同步其状态至ModelArts的弹性节点Server实例。

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，单击操作列的“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

2.4.4 释放资源

针对不再使用的弹性裸金属实例，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**。

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 释放弹性节点Server实例。

- 删除按需计费的弹性节点Server。

  在弹性节点Server列表中，单击操作列的“删除”，在弹出的确认对话框中，确认信息无误，然后单击“确定”，完成删除操作。

- 退订包年/包月的弹性节点Server。

  1、在弹性节点Server列表中，单击操作列的“退订”，跳转至费用中心的“退订资源”页面。

  2、根据界面提示，确认需要退订的资源，并选择退订原因。

  3、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。

  4、单击“退订”，再次根据界面信息确认要退订的资源。

  5、再次单击“退订”，完成包年/包月资源的退订操作。

----结束

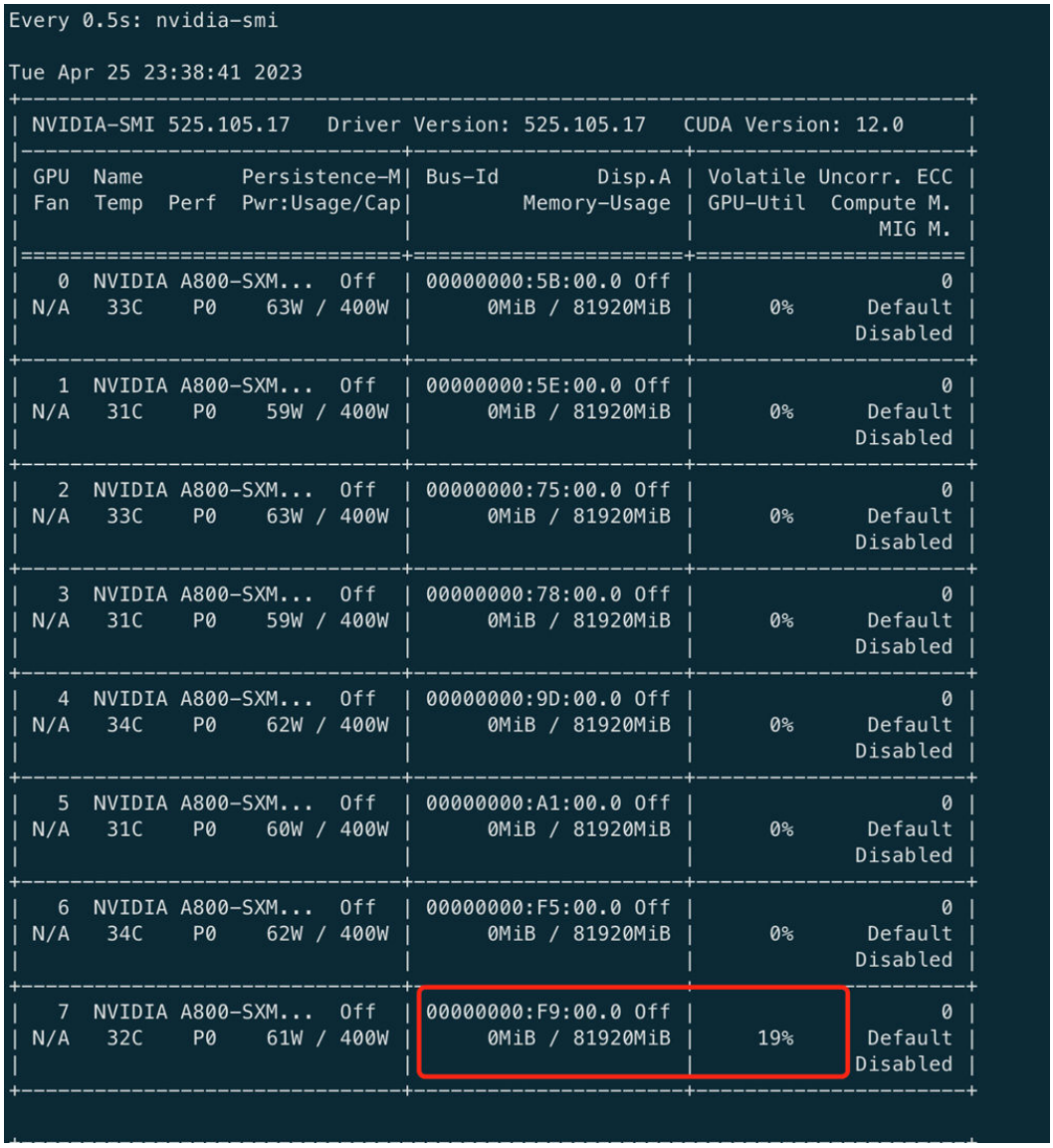
2.5 FAQ

2.5.1 GPU A 系列裸金属服务器没有任务，GPU 被占用问题

问题现象

服务器没有任务，但GPU显示被占用。

图 2-76 显卡运行状态



处理方法

```
nvidia-smi -pm 1
```

2.5.2 GPU A 系列裸金属服务器无法获取显卡问题解决方法

问题现象

在A系列裸金属服务器上使用PyTorch一段时间后，出现获取显卡失败的现象，报错如下：

```
> torch.cuda.is_available()
/usr/local/lib/python3.8/dist-packages/torch/cuda/__init__.py:107: UserWarning: CUDA initialization: Unexpected error from cudaGetDeviceCount(). Did you run some cuda functions before calling NumCudaDevices() that might have already set an error? Error 802: system not yet initialized (Triggered internally at ../c10/cuda/CUDAFunctions.cpp:109.)
  return torch._C._cuda_getDeviceCount() > 0
False
```

原因分析

Error 802原因为缺少fabricmanager，可能由于以下原因导致nvidia-fabricmanager.servcie不工作：

- 可能系统资源不足、如内存不足、内存泄露。
- 硬件故障、如IB网络或者GPU互联设备故障等。
- 没安装nvidia-fabricmanager组件或被误卸载。

处理方法

- 若未安装fabricmanager，则需安装改组件。
- 若已安装fabricmanager，运行以下命令重启fabricmanager.service。
  ```
  systemctl restart  nvidia-fabricmanager.service
  ```

  说明

  建议您进一步定位出nvidia-fabricmanager不工作原因，避免该问题再次发生。

2.5.3 GPU A 系列裸金属服务器 RoCE 带宽不足问题解决方法

问题现象

GP Ant8支持RoCE网卡， Ubuntu20.04场景，在进行nccl-tests时，总线带宽理论峰值可达90GB/s，但实际测试下来的结果只有35GB/s。

原因分析

“nv_peer_mem”是一个Linux内核模块，它允许支持P2P(Peer-to-Peer)的NVIDIA GPU直接进行内存访问(DMA)。这意味着数据可以直接在多个GPU之间传输，而无需经过CPU或系统内存，这可以显著降低延迟并提高带宽。

所以既然nccl-tests能正常测试， 但是达不到预期，可能是nv_peer_mem异常。

处理方法

- 查看nv_peer_mem是否已安装。
  ```
  dpkg -i | grep peer
  ```
  若未安装则需要安装，安装方法参考**装机指导**。 若已安装则进入下一检测项。
- 查看该软件是否已经加载至内核。
  ```
  lsmod | grep peer
  ```
  若没有则需要重新加载至内核，执行如下命令进行加载：
  ```
  /etc/init.d/nv_peer_mem start
  ```
  如果执行失败，可能是未加载nv_peer_mem.conf至/etc/infiniband/中或nv_peer_mem不在/etc/init.d/中。

  若找不到相关文件的问题，可以搜索相关文件在哪里，然后复制到指定目录，例如可执行如下命令：
  ```
  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/
  ```

2.5.4 GPU 裸金属服务器更换 NVIDIA 驱动后执行 nvidia-smi 提示 Failed to initialize NVML

问题现象

华为云裸金属服务器，NVIDIA驱动卸载后重新安装。

（1）已卸载原有版本NVIDIA驱动和CUDA版本，且已安装新版本的NVIDIA驱动和CUDA版本

（2）执行nvidia-smi失败，提示**Failed to initialize NVML: Driver/library version mismatch**

处理方法

执行命令：lsmod | grep nvidia，查看内核中是否残留旧版nvidia，显示如下：

```
nvidia_uvm            634880  8
nvidia_drm             53248  0
nvidia_modeset        790528  1 nvidia_drm
nvidia              12312576  86 nvidia_modeset,nvidia_uvm
```

卸载nvidia相关模块

```
sudo rmmod nvidia_drm
sudo rmmod nvidia_modeset
sudo rmmod nvidia_uvm
```

最后卸载nvidia

```
sudo rmmod nvidia
```

如果遇到“rmmod: ERROR: Module nvidia is in use”，执行以下命令：

```
sudo lsof /dev/nvidia*
```

并再次验证：

```
lsmod | grep nvidia
```

再次执行“nvidia-smi”，显示成功，且是最新安装的NVIDIA驱动。

2.5.5 训练速度突然下降以及执行 nvidia-smi 卡顿的解决方法

问题现象

在高性能8卡GPU的裸金属上的训练任务突然变慢，以前1个epoch约2小时执行完成，最近1个epoch需要2天才能执行完成，并且执行“nvidia-smi”也明显变很卡顿。

原因分析

根据现象描述可能出现了nvidia-smi D+进程。 "D+"表示进程状态为"Uninterruptible Sleep (usually IO)"，即进程正在等待I/O操作完成，此时无法被中断。

在正常情况下，nvidia-smi进程通常只会短暂地出现D+状态，因为它们是由内核控制的，该进程处于等待I/O操作完成的状态，可能是在读取或写入GPU相关的数据，这是正常的操作。但是，如果该进程一直处于"D+"状态，可能表明出现了I/O操作阻塞或其他问题，这可能导致系统死锁或其他问题。

如果想构造nvidia-smi D+进程，可以死循环一直执行nvidia-smi体验D+进程带来的系统不稳定性， 如：

```
#!/bin/bash

while true; do
nvidia-smi &
sleep 1   # 可以根据需要调整间隔时间
done
```

然后执行查看产生nvidia D+进程

```
ps -aux | grep -w D+
```

结果如下：

图 2-77 nvidia D+进程

```
root@devserver-aa3b:~# ps -aux | grep -w D+
root      147504 64.1  0.0  20736 12412 pts/2    D+   17:44   0:40 nvidia-smi
root      147595 87.0  0.0  34160 29528 pts/2    D+   17:44   0:05 nvidia-smi
root      147603 76.7  0.0  20736 14728 ttyS0    D+   17:44   0:03 nvidia-smi
root      147604 83.0  0.0  20736 14780 pts/2    D+   17:44   0:03 nvidia-smi
root      147606 85.3  0.0  20736 14920 pts/2    D+   17:45   0:02 nvidia-smi
root      147609 83.5  0.0  20736 14844 pts/2    D+   17:45   0:01 nvidia-smi
root      147612  0.0  0.0   6300  2612 pts/0    S+   17:45   0:00 grep --color=auto -w D+
root      147614  0.0  0.0  20736 14840 pts/2    D+   17:45   0:00 nvidia-smi
root      147616  0.0  0.0  20736 14824 pts/2    D+   17:45   0:00 nvidia-smi
root      147619  0.0  0.0   4556   516 pts/2    D+   17:45   0:00 nvidia-smi
root@devserver-aa3b:~#
```

此时可以观察你的训练任务或者执行“nvidia-smi”等命令，几乎是卡顿无法执行，因为内核IO已经阻塞， 无法执行相关GPU命令，只能尝试释放D+进程。

处理方法

“nvidia-smi”是一个NVIDIA GPU监视器命令行工具，用于查看GPU的使用情况和性能指标，可以帮助用户进行GPU优化和故障排除。 但是建议在业务软件或训练算法中，避免频繁使用“nvidia-smi”命令功能获取相关信息，存在锁死的风险。出现D+进程后可以尝试如下方法：

- 方法1： 可以根据ps -aux查到的进程号， 使用kill -9强制关闭进程。
  ```
  sudo kill -9 <进程ID>
  ```
- 方法2： 如果方法1执行后无法消除D+进程， 请尝试重启服务器。

2.5.6 如何将 Ubuntu20.04 内核版本从低版本升级至 5.4.0-144-generic

场景描述

Ubuntu20.04内核版本从低版本升级至5.4.0-144-generic。

操作指导

步骤1 检查当前内核版本。

```
uname -r
```

步骤2 升级内核


```
apt-get install linux-headers-5.4.0-144-generic linux-image-5.4.0-144-generic
grub-mkconfig -o /boot/efi/EFI/ubuntu/grub.cfg
reboot
```

第一条命令为安装Linux内核头文件和内核镜像，其中版本为5.4.0-144-generic。

第二条命令为重新生成GRUB引导程序的配置文件，用于在启动计算机时加载操作系统， 命令将使用新安装的内核镜像更新GRUB的配置文件，以便在下次启动时加载新的内核。

----结束

2.5.7 如何禁止 Ubuntu 20.04 内核自动升级

场景描述

在Ubuntu 20.04每次内核升级后，系统需要重新启动以加载新内核。如果您已经安装了自动更新功能，则系统将自动下载和安装可用的更新，这可能导致系统在不经意间被重启，若使用的软件依赖于特定版本的内核，那么当系统自动更新到新的内核版本时，可能会出现兼容性问题。在使用Ubuntu20.04时，建议手动控制内核的更新。

说明

禁用自动更新可能会导致您的系统变得不安全，因为您需要手动安装重要的安全补丁。在禁用自动更新之前，请确保您已了解其中的风险。

操作步骤

在Ubuntu 20.04上禁止内核自动升级，步骤如下：

步骤1 禁用unattended-upgrades。

“unattended-upgrades”是一个用于安装安全更新的软件包。要禁用它，首先打开“/etc/apt/apt.conf.d/20auto-upgrades”文件：

```
vi /etc/apt/apt.conf.d/20auto-upgrades
```

将其中的“Unattended-Upgrade "1"; ”改为“Unattended-Upgrade "0";”以禁用自动更新，然后保存文件并退出。

步骤2 将当前内核版本锁定。

要禁止特定的内核版本更新，你可以使用“apt-mark”命令将其锁定。

首先，检查你当前的内核版本：

```
uname -r
```

例如，如果你的内核版本是“5.4.0-42-generic”，你需要锁定所有与此版本相关的软件包。可执行以下命令：

```
sudo apt-mark hold linux-image-5.4.0-42-generic linux-headers-5.4.0-42-generic linux-modules-5.4.0-42-generic linux-modules-extra-5.4.0-42-generic
```

步骤3 禁用自动更新

要禁用所有自动更新，首先打开“/etc/apt/apt.conf.d/10periodic”文件：

```
vi /etc/apt/apt.conf.d/10periodic
```

修改文件以将所有选项设置为“0”：

```
APT::Periodic::Update-Package-Lists "0";
APT::Periodic::Download-Upgradeable-Packages "0";
APT::Periodic::AutocleanInterval "0";
APT::Periodic::Unattended-Upgrade "0";
```

保存文件并退出。

执行完以上步骤后，您的Ubuntu 20.04系统将不会自动升级内核。

----结束

2.5.8 如何设置 SSH 免密登录

本文介绍如何设置SSH免密登录。

操作步骤

步骤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
```

步骤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
```

步骤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

2.5.9 GPU 裸金属服务器使用 EulerOS 内核误升级解决方案

问题现象

GP Vnt1裸金属服务器，操作系统为EulerOS 2.9(基于CentOS制作的Linux发行版)，经常遇到服务器重启后，操作系统内核无故升级，导致系统上原安装的nvidia-driver等软件无法使用，只能卸载重新安装。

原因分析

分析EulerOS内核是如何在不知情的情况下升级的：

1. 首先查看当前操作系统内核。

   ```
   [root@Server-ddff ~]# uname -r
   4.18.0-147.5.1.6.h934.eulerosv2r9.x86_64
   ```

2. 一般执行如下升级命令，就会导致自动下载和安装高级内核版本。

   ```
   yum update -y
   ```

   执行后查看当前可用内核，发现已经新增了内核h998：

```
[root@Server-ddff ~]#
[root@Server-ddff ~]# cat /boot/grub2/grub.cfg |grep "menuentry "
menuentry 'EulerOS (4.18.0-147.5.1.6.h998.eulerosv2r9.x86_64) 2.0 (SP9x86_64)' --class euleros --
class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-4.18.0-147.5.
1.6.h934.eulerosv2r9.x86_64-advanced-f6aefacb-f2d3-4809-b708-6ad0357037f5' {
menuentry 'EulerOS (4.18.0-147.5.1.6.h934.eulerosv2r9.x86_64) 2.0 (SP9x86_64)' --class euleros --
class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-4.18.0-147.5.
1.6.h934.eulerosv2r9.x86_64-advanced-f6aefacb-f2d3-4809-b708-6ad0357037f5' {
menuentry 'EulerOS (0-rescue) 2.0 (SP9x86_64)' --class euleros --class gnu-linux --class gnu --class os
--unrestricted $menuentry_id_option 'gnulinux-0-rescue-advanced-f6aefacb-f2d3-4809-b708-6ad
0357037f5' {
[root@Server-ddff ~]#
```

3. 查看假如reboot（尚未reboot）后默认选择的内核版本：

```
[root@Server-ddff ~]# grub2-editenv list
saved_entry=EulerOS (4.18.0-147.5.1.6.h998.eulerosv2r9.x86_64) 2.0 (SP9x86_64)
boot_success=0
[root@Server-ddff ~]#
```

发现默认系统内核已经变为h998，reboot后就会生效。 此时若重启那么内核版本就被升级了。

处理方法

下文中假设当前服务器的内核版本是为4.18.0-147.5.1.6.h934.eulerosv2r9.x86_64，介绍如何避免操作系统内核自动升级。

1. 操作系统内核升级生效，必然需要服务器重启， 因此重启reboot前需要查看当前默认选择的内核版本：

```
[root@Server-ddff ~]# grub2-editenv list
saved_entry=EulerOS (4.18.0-147.5.1.6.h998.eulerosv2r9.x86_64) 2.0 (SP9x86_64)
boot_success=0
[root@Server-ddff ~]#
```

如上发现reboot后内核为4.18.0-147.5.1.6.h998.eulerosv2r9.x86_64，和当前内核版本h934不一致，则需要重新设置内核版本与当前版本一致。

2. 查看当前内核版本，并且锁定reboot后默认启动的内核版本，执行如下命令：

```
grub2-set-default 'EulerOS (4.18.0-147.5.1.6.h934.eulerosv2r9.x86_64) 2.0 (SP9x86_64)'
```

3. 执行后查看默认启动的内核版本是否和上述设置的相同：

```
[root@Server-ddff ~]# grub2-editenv list
saved_entry=EulerOS (4.18.0-147.5.1.6.h934.eulerosv2r9.x86_64) 2.0 (SP9x86_64)
boot_success=0
[root@Server-ddff ~]#
```

发现和当前内核一致，因此即使reboot也不会更改服务器的内核版本。

若希望升级指定的操作系统内核，也可以执行grub2-set-default进行设置默认启动内核版本。但操作系统内核升级可能带来的问题。例如在操作系统内核4.18.0-147.5.1.6.h934.eulerosv2r9.x86_64 下安装的nvidia-driver-515，由于执行了yum update并reboot服务器， 发现再次执行nvidia命令时报错：

```
[root@Server-ddff ~]# nvidia-smi
NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver. Make sure that the latest
NVIDIA driver is installed and running.
[root@Server-ddff ~]#
```

此时只能安装nvidia-driver-515以及配套的cuda版本，安装方法可以参考**GP Vnt1裸金属服务器EulerOS 2.9安装NVIDIA 515+CUDA 11.7**。

2.5.10 Atlas800 训练服务器硬件指南

场景描述

本文提供Atlas800训练服务器硬件相关指南，包括三维视图、备件信息、HCCL常用方法以及网卡配置信息。

Atlas 800 训练服务器三维视图

Atlas 800 训练服务器（型号9000）是基于华为鲲鹏920+Snt9处理器的AI训练服务器，实现完全自主可控，广泛应用于深度学习模型开发和AI训练服务场景，可单击**此处**查看硬件三维视图。

Atlas 800 训练服务器 HCCN Tool

Atlas 800 训练服务器 1.0.11 HCCN Tool接口参考主要介绍集群网络工具hccn_tool对外接口说明，包括配置RoCE网卡的IP、网关，配置网络检测对象IP和查询LLDP信息等。

Atlas 800 训练服务器备件查询助手

备件查询助手可以帮助你查询服务器的所有部件、规格描述，数量等详细信息。

打开网站后请输入SN编码“2102313LNR10P5100077”， 若失效可以提工单至华为云ModelArts查询。

Atlas 800 训练服务器的网卡配置问题

1. 机头网卡配置是什么?

   有以下两类网卡：

   - 四个2*100GE网卡，为RoCE网卡，插在NPU板。
   - 一个4*25GE/10GE，为Hi1822网卡，插在主板上的。

2. ifconfig能看到的网卡信息吗

   能看到主板上的网卡信息，即VPC分配的私有IP。若要看RoCE网卡的命令需要执行“hccn_tools”命令查看，参考**Atlas 800 训练服务器 1.0.11 HCCN Tool接口参考**中的指导。

3. NPU上的网卡在哪里可以看到， 会健康检查吗?

   8*NPU的网卡为机头上配置的四个2*100GE网卡。华为云有网卡健康状态监控机制。

2.5.11 GP Vnt1 裸金属服务器用 PyTorch 报错 CUDA initialization:CUDA unknown error

问题现象

在Vnt1 GPU裸金属服务器（Ubuntu18.04系统），根据**GP Vnt1裸金属服务器Ubuntu 18.04安装NVIDIA 470+CUDA 11.4**安装NVIDIA 470+CUDA 11.4后使用“nvidia-smi”和“nvcc - V”显示正确的安装信息，然后使用Pytorch下述命令验证cuda有效性：

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

显示报错：

```
UserWarning: CUDA initialization: CUDA unknown error - this may be due to an incorrectly set up environment, e.g. changing env variable CUDA_VISIBLE_DEVICES after program start. Setting the available devices to be zero. (Triggered internally at  /pytorch/c10/cuda/CUDAFunctions.cpp:100.)
  return torch._C._cuda_getDeviceCount() > 0
```

原因分析

nvidia-modprobe是一个Linux工具，用于在系统中加载NVIDIA驱动程序及其相关的内核模块。在Linux系统上安装NVIDIA显卡驱动后，需要通过“nvidia-modprobe”命令来加载相应的内核模块，以便让显卡驱动正常工作。

通常情况下，在安装NVIDIA驱动时，会自动执行“nvidia-modprobe”命令，将必要的内核模块加载到系统中。但有时候也可能需要手动执行该命令。例如，在更新了NVIDIA驱动后，需要重新加载新版本的内核模块才能使变更生效。

此外，若使用了多个NVIDIA显卡，每个显卡都需要加载相应的内核模块才能正常工作。在这种情况下，也需要手动执行“nvidia-modprobe”命令来加载所有必要的内核模块。

处理方法

- 方案1：操作系统内核重新加载nvidia_uvm。
  ```
  sudo rmmod nvidia_uvm
  sudo modprobe nvidia_uvm
  ```
- 方案2：安装nvidia-modprobe。
  ```
  apt-get install nvidia-modprobe
  ```

2.5.12 使用 SFS 盘出现报错 rpc_check_timeout:939 callbacks suppressed

问题现象

弹性文件服务（Scalable File Service，SFS）提供按需扩展的高性能文件存储（NAS），可以在裸金属服务器中通过网络协议挂载使用，SFS支持NFS和CIFS的网络协议。在使用裸金属服务器时， 将数据放在SFS盘中， 并发建立多个NFS链接、并发的读写数据、做大模型训练。 但有时候会出现读取速度变慢的现象，并且SFS提示报错"rpc_check_timeout:939 callbacks suppressed"。

原因分析

根据SFS客户端日志分析出现问题的时间点发现，SFS盘连接的客户端个数较多，在问题的时间点并发读取数据，I/O超高；当前SFS服务端的机制是：当SFS盘的性能到上限时，就会IO排队。IO排队造成处理时间超过 1 分钟时，客户端内核会打印"rpc_check_timeout:939 callbacks suppressed"日志。这个日志只是说明某个IO处理时间超过 1 分钟了，不会造成数据丢失。客户端有重试机制，等峰值过去后，所有IO最终都会正确处理。所以理论上，出现该错误日志， 并不会造成数据丢失， 只是SFS客户端I/O速度变慢或卡顿，但最终会争取处理。

处理方法

- 结合当前购买的SFS盘性能规划业务， 建议不要运行到性能上限。
- 可以购买多个SFS Turbo实例分担业务压力， 或者更换高性能的SFS盘。

- 一个SFS实例容量建议不要太大，建议以同样的成本换成购买多个SFS实例。

2.5.13 GPU 裸金属服务器无法 Ping 通的解决方案

问题现象

在华为云使用GPU裸金属服务器时， 服务器绑定EIP（华为云弹性IP服务）后，出现无法ping通弹性公网IP现象。

原因分析

1. 查看当前GPU裸金属服务器的安全组的入方向规则的配置，发现仅开通了TCP协议的22端口。
2. ping命令是一种基于ICMP协议（Internet Control Message Protocol）的网络诊断工具，利用ICMP协议向目标主机发送数据包并接收返回的数据包来判断网络连接质量。当安全组的入方向规则中没有包含ICMP协议，就会出现ping不通的问题。

处理方法

在当前安全组的入方向规则中添加一条规则，基本协议选择ICMP协议，详细配置如下表所示，添加规则步骤请参考**添加安全组规则**。

表 2-26 入方向规则

方向	协议/应用	端口	源地址
入方向	ICMP	全部	0.0.0.0/0

华为云安全组支持的协议参考可参考下表。

表 2-27 入方向规则

协议	端口	说明
协议	端口	说明
FTP	21	FTP服务上传和下载文件。
SSH	22	远程连接Linux弹性云服务器。
Telnet	23	使用Telnet协议访问网站。
SMTP	25	SMTP服务器所开放的端口，用于发送邮件。 基于安全考虑，TCP 25端口出方向默认被封禁，申请解封请参考**TCP 25端口出方向无法访问时怎么办？** 。
HTTP	80	使用HTTP协议访问网站。
POP3	110	使用POP3协议接受邮件。

协议	端口	说明
IMAP	143	使用IMAP协议接受邮件。
HTTPS	443	使用HTTPS协议访问网站。
SQL Server	1433	SQL Server的TCP端口，用于供SQL Server对外提供服务。
SQL Server	1434	SQL Server的TCP端口，用于返回SQLServer使用了哪个TCP/IP端口。
Oracle	1521	Oracle通信端口，弹性云服务器上部署了Oracle SQL需要放行的端口。
MySQL	3306	MySQL数据库对外提供服务的端口。
Windows Server Remote Desktop Services	3389	Windows远程桌面服务端口，通过这个端口可以连接Windows弹性云服务器。
代理	8080	8080端口常用于WWW代理服务，实现网页浏览，实现网页浏览。如果您使用8080端口，访问网站或使用代理服务器时，需要在IP地址后面加上：8080。安装Apache Tomcat服务后，默认服务端口为8080。
NetBIOS	137、138、139	NetBIOS协议常被用于Windows文件、打印机共享和Samba。 ● 137、138：UDP端口，通过网上邻居传输文件时使用的端口。 ● 139：通过这个端口进入的连接试图获得NetBIOS/SMB服务。

2.5.14 华为云 BMS GO SDK 和 Python 脚本实现裸金属服务器的操作系统切换

场景描述

华为云BMS裸金属服务器在停机状态可以切换操作系统，本文介绍以下两种切换操作系统的方式：

- 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。

使用 BMS Go SDK 的方式切换操作系统

```
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

```
# -*- 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")
```

裸金属服务器操作系统切换状态说明

- 切换中， 服务器状态为“切换操作系统中”。

  图 2-78 切换中服务器状态

- 切换后，服务器状态为“运行中”。

  图 2-79 切换后服务器状态

2.5.15 使用 GPU A 系列裸金属服务器有哪些注意事项?

使用华为云A系列裸金属服务器时有如下注意事项：

1. nvidia-fabricmanager版本号必须和nvidia-driver版本号保持一致，可参考**GPAnt8裸金属服务器Ubuntu 20.04安装NVIDIA 515+CUDA 11.7**中的安装nvidia-fabricmanager方法。
2. NCCL必须和CUDA版本相匹配，可单击**此处**可查看配套关系和安装方法。
3. 使用该裸金属服务器制作自定义镜像时， 必须清除残留文件，请参考**清理文件**。

2.5.16 华为云 CCE 集群纳管 GPU 裸金属服务器由于 CloudInit 导致纳管失败的解决方案

问题现象

创建出3台GPU裸金属服务器，使用A节点制作镜像，用于在CCE纳管裸金属服务器时，使用该镜像，但是纳管后发现服务器A纳管失败，剩下两台服务器纳管成功。

原因分析

在CCE纳管过程中，需要通过cloudinit userdata机制拉取cce-agent，但是在服务器上查看没有拉cce-agent的动作，理论上该动作是cloudinit中的脚本在创建时自动执行的，可能是由于安装脚本没有注入userdata或者注入了但未执行。

经查看是由于userdata未执行，可能原因为服务器A制作镜像时没有清理残留目录导致，即：

镜像里面“/var/lib/cloud/instances”残留了制作镜像机器（后面称模板机）的实例ID信息，如果制作镜像不清理“/var/lib/cloud/*”就会导致用该镜像再重装模板机时，cloud-init根据残留目录（含实例ID）判断已经执行过一次，进而不会再执行user-data里面的脚本。

而使用该镜像的服务器B和C，由于实例ID信息和镜像中残留的服务器A实例ID不同，就会执行user-data，所以CCE能纳管成功。

处理方法

制作镜像前，清理“/var/lib/cloud/”目录下的所有信息，请参考**清理临时文件**步骤对文件进行清理，然后再制作镜像。CCE重新纳管服务器A时， 使用最新制作的镜像即可。

2.5.17 GPU A 系列裸金属服务器使用 CUDA cudaGetDeviceCount()提示 CUDA initializat 失败

问题现象

在A系列GPU裸金属服务器上，系统环境是ubuntu20.04+nvidia515+cuda11.7，使用Pytorch2.0时出现如下错误：

```
CUDA initialization: Unexpected error from cudaGetDeviceCount()
```

原因分析

经过对裸金属服务器排查，发现nvidia-drvier和cuda都已安装，并且正常运行。nvidia-fabricmanager服务可以使单节点GPU卡间互联，在多卡GPU机器上，出现这种问题可能是nvidia-fabricmanger异常导致。

1. 执行以下命令，查看NVIDIA和CUDA的版本，以及nvidia-fabricmanager的状态。
   ```
   systemctl status nvidia-fabricmanager
   ```
2. 发现nvidia-fabricmanager的服务为failed状态，尝试重新启动nvidia-fabricmanager失败，且提示以下信息：
   ```
   nvidia-fabricmanager.service failed because the control process exited with error code
   ```
3. 通过命令查看nvidia-fabricmanager的版本，发现nvidia-fabricmanager版本与当前NVIDIA驱动版本不一致。
   ```
   dpkg -l | grep nvidia-fabricmanager
   ```
4. 卸载并重新安装正确版本的nvidia-fabricmanager，验证CUDA成功。

处理方法

1. 查看nvidia-fabricmanager的版本，若nvidia-fabricmanager版本与当前NVIDIA驱动版本不一致则执行下述命令卸载nvidia-fabricmanager。
   ```
   dpkg -l | grep nvidia-fabricmanager
   # 若有nvidia-fabricmanager软件，将其卸载
   # 若无nvidia-fabricmanager软件，请跳过此命令
   sudo apt-get autoremove --purge nvidia-fabricmanager-版本
   ```
2. 安装与NVIDIA驱动版本号相等的nvidia-fabricmanager（以515.105.01举例）。
   ```
   version=515.105.01
   main_version=$(echo $version | awk -F '.' '{print $1}')
   apt-get update
   apt-get -y install nvidia-fabricmanager-${main_version}=${version}-*
   ```
3. 启动nvidia-fabricmanager，确保其服务状态为RUNNING。
   ```
   systemctl enable nvidia-fabricmanager
   systemctl start nvidia-fabricmanager
   systemctl status nvidia-fabricmanager
   ```
4. 重新验证cuda状态， 输出为True。
   ```
   import torch
   print(torch.cuda.is_available())
   ```

2.5.18 GPU A 系列裸金属服务器节点内 NVLINK 带宽性能测试方法（Pytorch 模式）

场景描述

本文指导如何进行节点内NVLINK带宽性能测试，适用的环境为：Ant8或者Ant1 GPU裸金属服务器， 且服务器中已经安装相关GPU驱动软件，以及Pytorch2.0。

GPU A系列裸金属服务器，单台服务器GPU间是走NVLINK，可以通过相关命令查询GPU拓扑模式：

```
nvidia-smi topo -m
```

图 2-80 查询 GPU 拓扑模式

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	NIC0	NIC1	NIC2	NIC3	NIC4	NIC5	NIC6	NIC7	Affinity
GPU0	X	NV8	NV8	NV8	NV8	NV8	NV8	NV8	PXB	PXB	NODE	NODE	SYS	SYS	SYS	SYS	
GPU1	NV8	X	NV8	NV8	NV8	NV8	NV8	NV8	PXB	PXB	NODE	NODE	SYS	SYS	SYS	SYS	
GPU2	NV8	NV8	X	NV8	NV8	NV8	NV8	NV8	NODE	NODE	PXB	PXB	SYS	SYS	SYS	SYS	
GPU3	NV8	NV8	NV8	X	NV8	NV8	NV8	NV8	NODE	NODE	PXB	PXB	SYS	SYS	SYS	SYS	
GPU4	NV8	NV8	NV8	NV8	X	NV8	NV8	NV8	SYS	SYS	SYS	SYS	PXB	PXB	NODE	NODE	
GPU5	NV8	NV8	NV8	NV8	NV8	X	NV8	NV8	SYS	SYS	SYS	SYS	PXB	PXB	NODE	NODE	
GPU6	NV8	NV8	NV8	NV8	NV8	NV8	X	NV8	SYS	SYS	SYS	SYS	NODE	NODE	PXB	PXB	
GPU7	NV8	NV8	NV8	NV8	NV8	NV8	NV8	X	SYS	SYS	SYS	SYS	NODE	NODE	PXB	PXB	
NIC0	PXB	PXB	NODE	NODE	SYS	SYS	SYS	SYS	X	PIX	NODE	NODE	SYS	SYS	SYS	SYS	
NIC1	PXB	PXB	NODE	NODE	SYS	SYS	SYS	SYS	PIX	X	NODE	NODE	SYS	SYS	SYS	SYS	
NIC2	NODE	NODE	PXB	PXB	SYS	SYS	SYS	SYS	NODE	NODE	X	PIX	SYS	SYS	SYS	SYS	
NIC3	NODE	NODE	PXB	PXB	SYS	SYS	SYS	SYS	NODE	NODE	PIX	X	SYS	SYS	SYS	SYS	
NIC4	SYS	SYS	SYS	SYS	PXB	PXB	NODE	NODE	SYS	SYS	SYS	SYS	X	PIX	NODE	NODE	
NIC5	SYS	SYS	SYS	SYS	PXB	PXB	NODE	NODE	SYS	SYS	SYS	SYS	PIX	X	NODE	NODE	
NIC6	SYS	SYS	SYS	SYS	NODE	NODE	PXB	PXB	SYS	SYS	SYS	SYS	NODE	NODE	X	PIX	
NIC7	SYS	SYS	SYS	SYS	NODE	NODE	PXB	PXB	SYS	SYS	SYS	SYS	NODE	NODE	PIX	X	

操作步骤

步骤1 使用以下脚本测得GPU服务器内NVLINK带宽性能。

```
import torch
import numpy as np

device = torch.device("cuda")

n_gpus = 8
data_size = 1024 * 1024 * 1024  # 1 GB

speed_matrix = np.zeros((n_gpus, n_gpus))

for i in range(n_gpus):
    for j in range(i + 1, n_gpus):
        print(f"Testing communication between GPU {i} and GPU {j}...")
        with torch.cuda.device(i):
            data = torch.randn(data_size, device=device)
            torch.cuda.synchronize()
        with torch.cuda.device(j):
            result = torch.randn(data_size, device=device)
            torch.cuda.synchronize()
        with torch.cuda.device(i):
            start = torch.cuda.Event(enable_timing=True)
            end = torch.cuda.Event(enable_timing=True)
            start.record()
            result.copy_(data)
            end.record()
            torch.cuda.synchronize()
            elapsed_time_ms = start.elapsed_time(end)
        transfer_rate = data_size / elapsed_time_ms * 1000 * 8 / 1e9
        speed_matrix[i][j] = transfer_rate
        speed_matrix[j][i] = transfer_rate

print(speed_matrix)
```


步骤2 以Ant8 GPU裸金属服务器为例， 其理论GPU卡间带宽为：NVIDIA*NVLink*Bridge for 2GPUS: 400GB/s。使用上述测试脚本测得带宽性能进行如下分析。

- 正常模式-NVLINK全互通，带宽约为370GB。基本符合预期，且证明Ant GPU裸金属服务器内部GPU间确实走NVLINK模式，且完全互联。

图 2-81 正常模式带宽性能

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7
GPU0	0.00	307.55	369.55	369.00	369.00	368.83	368.59	369.19
GPU1	307.55	0.00	369.26	370.37	368.07	368.78	369.01	370.36
GPU2	369.55	369.26	0.00	368.15	370.90	370.48	370.84	370.95
GPU3	369.00	370.37	368.15	0.00	368.05	370.16	370.42	370.38
GPU4	369.00	368.07	370.90	368.05	0.00	368.02	370.01	370.97
GPU5	368.83	368.78	370.48	370.16	368.02	0.00	369.75	369.99
GPU6	368.59	369.01	370.84	370.42	370.01	369.75	0.00	367.77
GPU7	369.19	370.36	370.95	370.38	370.97	369.99	367.77	0.00
服务器1，116.204.125.186								

- 异常模式-NVLINK部分互通，出现带宽波动较大的情况。如下图中GPU0和GPU4之间带宽远低于理论值， 存在问题。

图 2-82 异常模式带宽性能

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7
GPU0	0.00	367.18	368.41	369.04	**24.45**	368.85	368.73	368.82
GPU1	367.18	0.00	369.53	370.42	370.34	370.53	370.81	370.95
GPU2	368.41	369.53	0.00	370.37	370.78	370.81	370.63	370.92
GPU3	369.04	370.42	370.37	0.00	370.19	370.47	370.70	370.74
GPU4	**24.45**	370.34	370.78	370.19	0.00	370.25	370.49	370.53
GPU5	368.85	370.53	370.81	370.47	370.25	0.00	370.36	370.13
GPU6	368.73	370.81	370.63	370.70	370.49	370.36	0.00	368.94
GPU7	368.82	370.95	370.92	370.74	370.53	370.13	368.94	0.00

出现这种现象， 可尝试重装nvidia/cuda/nvidia-fabricmanager， 重装后再测试又恢复到了正式模式，GPU0和GPU4之间带宽恢复到370GB/s。

可能原因如下，仅供参考：

a. 驱动程序问题：可能是由于驱动程序没有正确安装或配置，导致NVLINK带宽受限。重新安装nvidia驱动、CUDA和nvidia-fabricmanager等软件后，驱动程序可能已经正确配置，从而解决了这个问题。

b. 硬件问题：如果GPU之间的NVLINK连接存在硬件故障，那么这可能会导致带宽受限。重新安装软件后，重启系统，可能触发了某种硬件自检或修复机制，从而恢复了正常的带宽。

c. 系统负载问题：最初测试GPU卡间带宽时，可能存在其他系统负载，如进程、服务等，这些负载会占用一部分网络带宽，从而影响NVLINK带宽的表现。重新安装软件后，这些负载可能被清除，从而使NVLINK带宽恢复正常。

----结束

2.5.19 裸金属服务器 Euler OS 升级 NetworkManager-config-server 导致 SSH 链接故障解决方案

问题现象

裸金属服务器EulerOS 2.8系统下，使用yum update -y命令，导致软件NetworkManagre-config-server升级到高版本，出现SSH链接故障无法访问。

原因分析

1. 查看yum命令历史，发现执行了“yum update -y”，“yum update -y”命令是用于在Linux操作系统上更新软件包的命令。其中，选项-y表示在更新时自动确认所有提示信息，而不需要手动输入“y”确认。

   请注意，使用此命令将会检查您系统中已安装的软件包并更新至最新版本。

   图 2-83 yum 命令历史

   ```
   [root@devserver-ei-cto-office-cn9-4 ~]# yum history
   ID     | Command line             | Date and time    | Action(s)      | Altered
   -------------------------------------------------------------------------------
        9 | install lrzsz -y         | 2023-05-27 11:35 | Install        |    1
        8 | install -y vim gcc gcc-c | 2023-05-23 19:28 | Install        |   70
        7 | update -y                | 2023-05-23 19:23 | I, U           |  437 E<
        6 | install cloud-utils-grow | 2020-02-17 17:48 | Install        |    1 >
        5 | install gcc perl strace  | 2020-02-17 17:46 | Install        |    3  <
        4 | install python2-requests | 2020-02-17 17:17 | Install        |    8 >
        3 | install python2-six      | 2020-02-17 17:17 | Install        |    1
        2 | install cloud-init       | 2020-02-17 17:14 | Install        |   22
        1 |                          | 2020-02-17 16:24 | Install        |  809 EE
   [root@devserver-ei-cto-office-cn9-4 ~]#
   ```

2. 查看NetworkManager配置：

   ```
   NetworkManager --print-config
   ```

   配置内容如下：

   ```
   # NetworkManager configuration: /etc/NetworkManager/NetworkManager.conf (lib: 00-server.conf)

   [main]
   # plugins=ifcfg-rh,ibft
   # rc-manager=symlink
   # auth-polkit=true
   # dhcp=dhclient
   no-auto-default=*
   ignore-carrier=*

   [logging]
   # backend=journal
   # audit=false
   ```

   发现“no-auto-default=*”是打开的状态，“no-auto-default=*”含义是关闭DH Client，无法使用DHCP获取IP。正常情况下裸金属服务器这个参数是被注释的状态。

   - 当服务器有网卡配置文件， NetworkManager.service实现将VPC子网分配的私有IP写入网卡配置文件中。NetworkManager.service会优先读取网卡配置文件中的IP设置为主机IP， 此时无论DH Cient是否关闭，服务器都可以获取分配IP。
   - 当服务器没有网卡配置文件时，DH Client开启，此时服务器会分配私有IP。如果关闭DH Client，则服务器无法获取私有IP。

图 2-84 查看 NetworkManager 配置

```
[root@xb-test- ~]# ll /etc/sysconfig/network-scripts/
total 224
-rw-r--r--. 1 root root   254 Jul 22  2019 ifcfg-lo
-rwxr-xr-x. 1 root root  2062 Jul 22  2019 ifdown
-rwxr-xr-x. 1 root root   649 Jul 22  2019 ifdown-bnep
-rwxr-xr-x. 1 root root  6423 Jul 22  2019 ifdown-eth
-rwxr-xr-x. 1 root root   772 Jul 22  2019 ifdown-ippp
-rwxr-xr-x. 1 root root  4540 Jul 22  2019 ifdown-ipv6
lrwxrwxrwx. 1 root root    11 Jul 22  2019 ifdown-isdn -> ifdown-ippp
-rwxr-xr-x. 1 root root  2068 Jul 22  2019 ifdown-post
-rwxr-xr-x. 1 root root   873 Jul 22  2019 ifdown-routes
-rwxr-xr-x. 1 root root  1462 Jul 22  2019 ifdown-sit
-rwxr-xr-x. 1 root root  1625 Jul 22  2019 ifdown-Team
-rwxr-xr-x. 1 root root  1560 Jul 22  2019 ifdown-TeamPort
-rwxr-xr-x. 1 root root  1466 Jul 22  2019 ifdown-tunnel
-rwxr-xr-x. 1 root root  5413 Jul 22  2019 ifup
-rwxr-xr-x. 1 root root 12206 Jul 22  2019 ifup-aliases
-rwxr-xr-x. 1 root root   909 Jul 22  2019 ifup-bnep
-rwxr-xr-x. 1 root root 13656 Jul 22  2019 ifup-eth
-rwxr-xr-x. 1 root root 12078 Jul 22  2019 ifup-ippp
-rwxr-xr-x. 1 root root 11895 Jul 22  2019 ifup-ipv6
lrwxrwxrwx. 1 root root     9 Jul 22  2019 ifup-isdn -> ifup-ippp
-rwxr-xr-x. 1 root root   654 Jul 22  2019 ifup-plip
-rwxr-xr-x. 1 root root  1068 Jul 22  2019 ifup-plusb
-rwxr-xr-x. 1 root root  5004 Jul 22  2019 ifup-post
-rwxr-xr-x. 1 root root  2004 Jul 22  2019 ifup-routes
-rwxr-xr-x. 1 root root  3307 Jul 22  2019 ifup-sit
-rwxr-xr-x. 1 root root  1759 Jul 22  2019 ifup-Team
-rwxr-xr-x. 1 root root  1880 Jul 22  2019 ifup-TeamPort
-rwxr-xr-x. 1 root root  2825 Jul 22  2019 ifup-tunnel
-rwxr-xr-x. 1 root root  1840 Jul 22  2019 ifup-wireless
-rwxr-xr-x. 1 root root  5425 Jul 22  2019 init.ipv6-global
-rw-r--r--. 1 root root 20603 Jul 22  2019 network-functions
-rw-r--r--. 1 root root 31025 Jul 22  2019 network-functions-ipv6
[root@xb-test- ~]# NetworkManager --print-config
# NetworkManager configuration: /etc/NetworkManager/NetworkManager.conf (lib: 00-server.conf)

[main]
# plugins=ifcfg-rh,ibft
# rc-manager=symlink
# auth-polkit=true
# dhcp=dhclient
no-auto-default=*
ignore-carrier=*

[logging]
# backend=journal
# audit=false
[root@xb-test- ~]#
```

图 2-85 查看网络配置

```
[root@xb-test- ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp69s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether fa:16:3e:7c:d4:d0 brd ff:ff:ff:ff:ff:ff
3: enp189s0f0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether 8c:2a:8e:9a:8d:72 brd ff:ff:ff:ff:ff:ff
4: enp189s0f1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether 8c:2a:8e:9a:8d:73 brd ff:ff:ff:ff:ff:ff
5: enp189s0f2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether 8c:2a:8e:9a:8d:74 brd ff:ff:ff:ff:ff:ff
6: enp189s0f3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether 8c:2a:8e:9a:8d:75 brd ff:ff:ff:ff:ff:ff
[root@xb-test- ~]# ifconfig
enp189s0f0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 8c:2a:8e:9a:8d:72  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp189s0f1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 8c:2a:8e:9a:8d:73  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp189s0f2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 8c:2a:8e:9a:8d:74  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp189s0f3: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 8c:2a:8e:9a:8d:75  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp69s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether fa:16:3e:7c:d4:d0  txqueuelen 1000  (Ethernet)
        RX packets 12  bytes 780 (780.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@xb-test- ~]#
```

命令“yum update -y”或“yum update NetworkManagre-config-server”，都会将NetworkManagre-config-server软件升级，高版本的NetworkManagre-config-server会将参数no-auto-default=*是打开的状态，又因当前镜像无网卡配置文件导致ip获取不到，从而使得SSH连接失败。

处理方法

在Euler2.8操作系统，NetworkManagre-config-server是一个无用的软件包，无需安装 。执行以下命令卸载NetworkManagre-config-server，并重启NetworkManager服务，重新尝试SSH连接，验证网络是否恢复。

```
# 卸载 NetworkManagre-config-server
rpm -e NetworkManager-config-server
# 重启 NetworkManager 服务
systemctl restart NetworkManager
```