ModelArts

最佳实践

文档版本 01

发布日期 2024-04-30

华为云计算技术有限公司

版权所有 © 华为云计算技术有限公司 2024。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

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。

目 录

 版权所有 © 华为云计算技术有限公司

 版权所有 © 华为云计算技术有限公司

1 官方案例列表

在最佳实践文档中，提供了针对多种场景、多种AI引擎的ModelArts案例，方便您通过如下案例快速了解使用ModelArts完成AI开发的流程和操作。

配置 ModelArts 使用权限（基础教程）

样例	对应功能	场景	说明
场景描述	IAM权限配置、全局配置	为子用户配置权限	当一个华为云账号下需创建多个IAM用户（即子用户）时，可参考此样例，为IAM用户赋予使用ModelArts所需的权限。避免IAM用户因权限问题导致使用时出现异常。

模型训练-自定义算法样例列表（高阶教程）

表 1-1 自定义算法样例列表

样例	镜像	对应功能	场景	说明
使用自定义算法构建模型（手写数字识别）	PyTorch	自定义算法	手写数字识别	使用用户自己的算法，训练得到手写数字识别模型，并部署后进行预测。
示例：从0到1制作自定义镜像并用于训练（PyTorch+CPU/GPU）	PyTorch	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。

版权所有 © 华为云计算技术有限公司

样例	镜像	对应功能	场景	说明
示例：从0到1制作自定义镜像并用于训练（MPI+CPU/GPU）	MPI	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。
示例：从0到1制作自定义镜像并用于训练（Horovod-PyTorch+GPU）	Horovod-PyTorch	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Horovod-PyTorch，训练使用的资源是GPU。
示例：从0到1制作自定义镜像并用于训练（MindSpore+GPU）	MindSpore	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MindSpore，训练使用的资源是GPU。
示例：从0到1制作自定义镜像并用于训练（Tensorflow+GPU）	Tensorflow	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Tensorflow，训练使用的资源是GPU。

版权所有 © 华为云计算技术有限公司

2 权限管理

2.1 ModelArts 权限管理基本概念

ModelArts作为一个完备的AI开发平台，支持用户对其进行细粒度的权限配置，以达到精细化资源、权限管理之目的。这类特性在大型企业用户的使用场景下很常见，但对个人用户则显得复杂而意义不足，所以建议个人用户在使用ModelArts时，参照**个人用户快速配置ModelArts访问权限**来进行初始权限设置。

说明

您是否需要阅读本文档?

如果下述问题您的任何一个回答为“是”，则需要阅读此文档

- 您是企业用户，且
  - 存在多个部门，且需要限定不同部门的用户只能访问其专属资源、功能
  - 存在多种角色（如管理员、算法开发者、应用运维），希望限制不同角色只能使用特定功能
  - 逻辑上存在多套“环境”且相互隔离（如开发环境、预生产环境、生产环境），并限定不同用户在不同环境上的操作权限
  - 其他任何需要对特定子用户（组）做出特定权限限制的情况
- 您是个人用户，但已经在IAM创建多个子用户，且期望限定不同子用户所能使用的ModelArts功能、资源不同
- 希望了解ModelArts的权限控制能力细节，期望理解其概念和实操方法

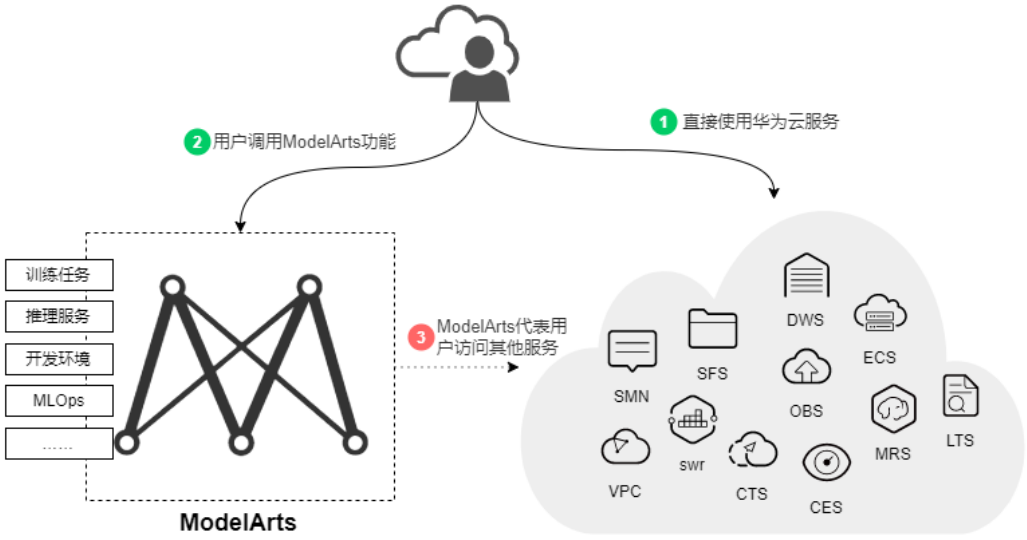
ModelArts的大部分权限管理能力均基于统一身份认证服务（Identity and Access Management，简称IAM）来实现，在您继续往下阅读之前，强烈建议您先行熟悉**IAM基本概念**，如果能完整理解IAM的所有概念，将更加有助于您理解本文档。

为了支持客户对ModelArts的权限做精细化控制，提供了3个方面的能力来支撑，分别是：权限、委托和工作空间。下面分别讲解。

版权所有 © 华为云计算技术有限公司

理解 ModelArts 的权限与委托

图 2-1 权限管理抽象



ModelArts与其他服务类似，对外暴露的每个功能，都通过IAM的权限来进行控制。比如，用户（此处指IAM子用户，而非租户）希望在ModelArts创建训练作业，则该用户必须拥有 "modelarts:trainJob:create" 的权限才可以完成操作（无论界面操作还是API调用）。关于如何给一个用户赋权（准确讲是需要先将用户加入用户组，再面向用户组赋权），可以参考IAM的文档《**权限管理**》。

而ModelArts还有一个特殊的地方在于，为了完成AI计算的各种操作，AI平台在任务执行过程中需要访问用户的其他服务，典型的就是训练过程中，需要访问OBS读取用户的训练数据。在这个过程中，就出现了ModelArts“代表”用户去访问其他云服务的情形。从安全角度出发，ModelArts代表用户访问任何云服务之前，均需要先获得用户的授权，而这个动作就是一个“委托”的过程。用户授权ModelArts再代表自己访问特定的云服务，以完成其在ModelArts平台上执行的AI计算任务。

综上，对于**图1 权限管理抽象**可以做如下解读：

- 用户访问任何云服务，均是通过标准的IAM权限体系进行访问控制。用户首先需要具备相关云服务的权限（根据您具体使用的功能不同，所需的相关服务权限多寡亦有差异）。
- **权限**：用户使用ModelArts的任何功能，亦需要通过IAM权限体系进行正确权限授权。
- **委托**：ModelArts上的AI计算任务执行过程中需要访问其他云服务，此动作需要获得用户的委托授权。

ModelArts 权限管理

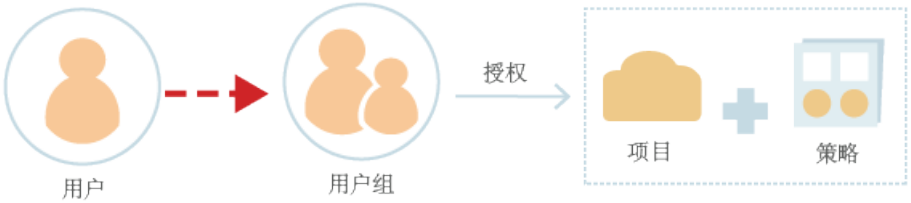
默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于授予的权限对云服务进行操作。

版权所有 © 华为云计算技术有限公司

⚠ 注意

ModelArts部署时通过物理区域划分，为项目级服务，授权时“选择授权范围方案”可以选择“指定区域项目资源”，如果授权时指定了区域对应的项目，则该权限仅对此项目生效；简单的做法是直接选择“所有资源”。

ModelArts也支持企业项目，所以选择授权范围方案时，也可以指定企业项目。具体操作参见《**创建用户组并授权**》。



IAM在对用户组授权的时候，并不是直接将具体的某个权限进行赋权，而是需要先将权限加入到“策略”当中，再把策略赋给用户组。为了方便用户的权限管理，各个云服务都提供了一些预置的“系统策略”供用户直接使用。如果预置的策略不能满足您的细粒度权限控制要求，则可以通过“自定义策略”来进行精细控制。

表2-1列出了ModelArts的所有预置系统策略。

表 2-1 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限	系统策略

通常来讲，只给管理员开通“ModelArts FullAccess”，如果不需要太精细的控制，直接给所有用户开通“ModelArts CommonOperations”即可满足大多数小团队的开发场景诉求。如果您希望通过自定义策略做深入细致的权限控制，请阅读**ModelArts的IAM权限控制详解**。

版权所有 © 华为云计算技术有限公司

说明

ModelArts的权限不会凌驾于其他服务的权限之上，当您给用户进行ModelArts赋权时，系统不会自动对其他相关服务的相关权限进行赋权。这样做的好处是更加安全，不会出现预期外的“越权”，但缺点是，您必须同时给用户赋予不同服务的权限，才能确保用户可以顺利完成某些ModelArts操作。

举例，如果用户需要用OBS中的数据进行训练，当已经为IAM用户配置ModelArts训练权限时，仍需同时为其配置对应的OBS权限（读、写、列表），才可以正常使用。其中OBS的列表权限用于支持用户从ModelArts界面上选择要进行训练的数据路径；读权限主要用于数据的预览以及训练任务执行时的数据读取；写权限则是为了保存训练结果和日志。

- 对于个人用户或小型组织，一个简单做法是为IAM用户配置“作用范围”为“全局级服务”的“Tenant Administrator”策略，这会使用户获得除了IAM以外的所有用户权限。在获得便利的同时，由于用户的权限较大，会存在相对较大的安全风险，需谨慎使用。（对于个人用户，其默认IAM账号就已经属于admin用户组，且具备Tenant Administrator权限，无需额外操作）
- 当您需要限制用户操作，仅为ModelArts用户配置OBS相关的最小化权限项，具体操作请参见**OBS权限管理**。对于其他云服务，也可以进行精细化权限控制，具体请参考对应的云服务文档。

ModelArts 委托授权

前文已经介绍，ModelArts在执行AI计算任务过程中，需要“代表”用户去访问其他云服务，而此动作需要提前获得用户的授权。在IAM权限体系下，此类授权动作是通过“委托”来完成。

关于委托的基本概念及操作可以参考对应的IAM文档《**委托其他云服务管理资源**》。

为了简化用户的委托授权操作，ModelArts增加了自动配置委托授权的支持，用户仅需在ModelArts控制台的“全局配置”页面中，为自己或特定用户配置委托即可。

说明

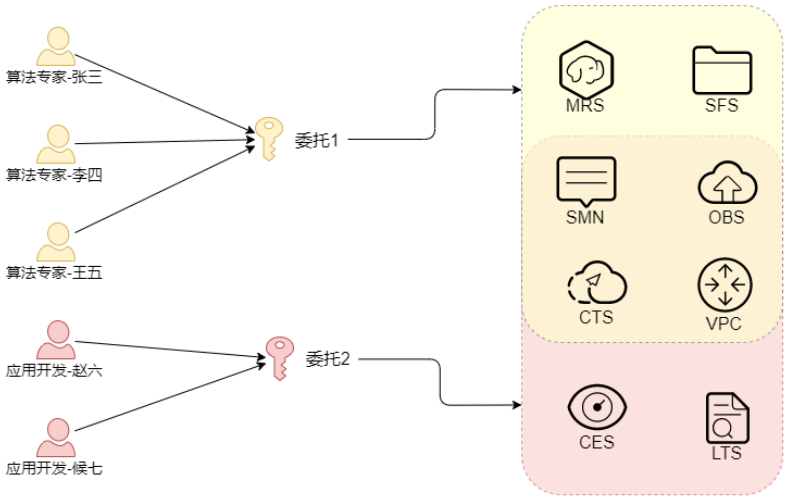
- 只有具备IAM委托管理权限的用户才可以进行此项操作，通常是IAM admin用户组的成员才具备此权限。
- 目前ModelArts的委托授权操作是分区域操作的，这意味着您需要在每个您所用到的区域均执行委托授权操作。

在ModelArts控制台的“全局配置”页面，单击“添加授权”后，系统会引导您为特定用户或所有用户进行委托配置，通常默认会创建一个名为“modelarts_agency_<用户名>_随机ID”的委托条目。在权限配置的区域，您可以选择ModelArts提供的预置配置，也可以自定义选择您所授权的策略。当然如果这两种形态对于您的诉求均过于粗犷，您也可以直接在IAM管理页面里创建完全由您进行精细化配置的委托（需要委托给ModelArts服务），然后在此页面的委托选择里使用“已有委托”“”（而非“新增委托”）。

至此，您应该已经发现了一个细节，ModelArts在使用委托时，是将其与用户进行关联的，用户与委托的关系是多对1的关系。这意味着，如果两个用户需要配置的委托一致，那么不需要为每个用户都创建一个独立的委托项，只需要将两个用户都“指向”同一个委托项即可。

版权所有 © 华为云计算技术有限公司

图 2-2 用户与委托对应关系



说明

每个用户必须关联委托才可以使用ModelArts，但即使委托所赋之权限不足，在API调用之初也不会报错，只有到系统具体使用到该功能时，才会发生问题。例如，用户在创建训练任务时打开了“消息通知”，该功能依赖SMN委托授权，但只有训练任务运行过程中，真正需要发送消息时，系统才会“出错”，而有些错误系统会选择“忽略”，另一些错误则可能导致任务直接失败。当您做深入的“权限最小化”限制时，请确保您在ModelArts上将要执行的操作仍旧有足够的权限。

严格授权模式

严格授权模式是指在IAM中创建的子用户必须由账号管理员显式在IAM中授权，才能访问ModelArts服务，管理员用户可以通过授权策略为普通用户精确添加所需使用的ModelArts功能的权限。

相对的，在非严格授权模式下，子用户不需要显式授权就可以使用ModelArts，管理员需要在IAM上为子用户配置Deny策略来禁止子用户使用ModelArts的某些功能。

账号的管理员用户可以在“全局配置”页面修改授权模式。

须知

如无特殊情况，建议优先使用严格授权模式。在严格授权模式下，子用户要使用ModelArts的功能都需经过授权，可以更精确的控制子用户的权限范围，达成权限最小化的安全策略。

用工作空间限制资源访问

工作空间是ModelArts面向企业客户提供的一个高阶功能，用于进一步将用户的资源划分在多个**逻辑隔离**的空间中，并支持以空间维度进行访问的权限限定。目前工作空间功能是“受邀开通”状态，作为企业用户您可以通过您对口的技术支持经理申请开通。

在开通工作空间后，系统会默认为您创建一个“default”空间，您之前所创建的所有资源，均在该空间下。当您创建新的工作空间之后，相当于您拥有了一个新的

版权所有 © 华为云计算技术有限公司

“ModelArts分身”，您可以通过菜单栏的左上角进行工作空间的切换，不同工作空间中的工作互不影响。

创建工作空间时，必须绑定一个企业项目。多个工作空间可以绑定到同一个企业项目，但一个工作空间**不可以**绑定多个企业项目。借助工作空间，您可以对不同用户的资源访问和权限做更加细致的约束，具体为如下两种约束：

- 只有被授权的用户才能访问特定的工作空间（在创建、管理工作空间的页面进行配置），这意味着，像数据集、算法等AI资产，均可以借助工作空间做访问的限制。
- 在前文提到的权限授权操作中，如果“选择授权范围方案”时设定为“指定企业项目资源”，那么该授权仅对绑定至该企业项目的工作空间生效。

说明

- 工作空间的约束与权限授权的约束是叠加生效的，意味着对于一个用户，必须同时拥有工作空间的访问权和训练任务的创建权限（且该权限覆盖至当前的工作空间），他才可以在这个空间里提交训练任务。
- 对于已经开通企业项目但没有开通工作空间的用户，其所有操作均相当于在“default”企业项目里进行，请确保对应权限已覆盖了名为default的企业项目。
- 对于未开通企业项目的用户，不受上述约束限制。

本章小结

对于ModelArts的权限管理，总结了如下几条关键点：

- 如果您是个人用户，则不需要考虑细粒度权限问题，您的账户默认具备使用ModelArts的所有权限。
- ModelArts平台的所有功能均通过IAM体系进行了权限管控，您可以通过标准的IAM**授权**动作，来对特定用户进行精细化的权限管控。
- 对于所有用户（包括个人用户），需要完成对ModelArts的**委托**授权（ModelArts > 全局配置 > 添加授权），才能使用特定的功能，否则会造成您的操作出现不可预期的错误。
- 对于开通了企业项目的用户，可以进一步申请开通ModelArts的**工作空间**，通过组合使用基础授权和工作空间，来达成更加复杂的权限控制目的。

2.2 权限控制方式

2.2.1 IAM

介绍ModelArts所有功能涉及到的IAM权限配置。

IAM 权限简介

如果您需要对华为云云服务平台上购买创建的ModelArts资源，为企业中的员工设置不同的访问权限，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全的控制华为云云服务资源的访问。如果华为账号已经能满足您的要求，不需要通过IAM对用户进行权限管理，您可以跳过本章节，不影响您使用ModelArts服务的其他功能。

版权所有 © 华为云计算技术有限公司

IAM是华为云云服务平台提供权限管理的基础服务，无需付费即可使用，您只需要为您账号中的资源进行付费。

通过IAM，您可以通过授权控制他们对华为云云服务资源的访问范围。例如您的员工中有负责软件开发的人员，您希望他们拥有ModelArts的使用权限，但是不希望他们拥有删除ModelArts等高危操作的权限，那么您可以使用IAM进行权限分配，通过授予用户仅能使用ModelArts，但是不允许删除ModelArts的权限，控制他们对ModelArts资源的使用范围。

关于IAM的详细介绍，请参见**IAM产品介绍**。

角色与策略权限管理

ModelArts服务支持角色与策略授权。默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略或角色，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于被授予的权限对云服务进行操作。

ModelArts部署时通过物理区域划分，为项目级服务。授权时，“授权范围”需要选择“指定区域项目资源”，然后在指定区域（如亚太-曼谷）对应的项目（ap-southeast-2）中设置相关权限，并且该权限仅对此项目生效；如果“授权范围”选择“所有资源”，则该权限在所有区域项目中都生效。访问ModelArts时，需要先切换至授权区域。

如**表2-2**所示，包括了ModelArts的所有系统策略权限。如果系统预置的ModelArts权限，不满足您的授权要求，可以创建自定义策略，可参考**策略JSON格式字段介绍**。

表 2-2 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限。	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限。	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限。	系统策略

ModelArts对其他云服务有依赖关系，因此在ModelArts控制台的各项功能需要配置相应的服务权限后才能正常查看或使用，依赖服务及其预置的权限如下。

表 2-3 ModelArts 控制台依赖服务的角色或策略

控制台功能	依赖服务	需配置角色/策略
数据管理	对象存储服务OBS	OBS Administrator
	数据湖探索DLI	DLI FullAccess
	MapReduce服务MRS	MRS Administrator

版权所有 © 华为云计算技术有限公司

控制台功能	依赖服务	需配置角色/策略
	数据仓库服务 GaussDB(DWS)	DWS Administrator
	云审计服务CTS	CTS Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
开发环境	对象存储服务OBS	OBS Administrator
	凭据管理服务CSMS	CSMS ReadOnlyAccess
	云审计服务CTS	CTS Administrator
	弹性云服务器ECS	ECS FullAccess
	容器镜像服务SWR	SWR Administrator
	弹性文件服务SFS	SFS Turbo FullAccess
	应用运维管理服务 AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
训练管理	对象存储服务OBS	OBS Administrator
	消息通知服务SMN	SMN Administrator
	云审计服务CTS	CTS Administrator
	弹性文件服务SFS Turbo	SFS Turbo ReadOnlyAccess
	容器镜像服务SWR	SWR Administrator
	应用运维管理服务 AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Workflow	对象存储服务OBS	OBS Administrator
	云审计服务CTS	CTS Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
自动学习	对象存储服务OBS	OBS Administrator

版权所有 © 华为云计算技术有限公司

控制台功能	依赖服务	需配置角色/策略
	云审计服务CTS	CTS Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI应用管理	对象存储服务OBS	OBS Administrator
	企业项目管理服务EPS	EPS FullAccess
	云审计服务CTS	CTS Administrator
	容器镜像服务SWR	SWR Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
部署上线	对象存储服务OBS	OBS Administrator
	云监控服务CES	CES ReadOnlyAccess
	消息通知服务SMN	SMN Administrator
	企业项目管理服务EPS	EPS FullAccess
	云审计服务CTS	CTS Administrator
	云日志服务LTS	LTS FullAccess
	虚拟私有云VPC	VPC FullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI Gallery	对象存储服务OBS	OBS Administrator
	云审计服务CTS	CTS Administrator
	容器镜像服务SWR	SWR Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
专属资源池	云审计服务CTS	CTS Administrator
	云容器引擎CCE	CCE Administrator
	裸金属服务器BMS	BMS FullAccess
	镜像服务IMS	IMS FullAccess
	数据加密服务DEW	DEW KeypairReadOnlyAccess
	虚拟私有云VPC	VPC FullAccess
	弹性云服务器ECS	ECS FullAccess
	弹性文件服务SFS	SFS Turbo FullAccess

版权所有 © 华为云计算技术有限公司

控制台功能	依赖服务	需配置角色/策略
	对象存储服务OBS	OBS Administrator
	应用运维管理服务 AOM	AOM FullAccess
	标签管理服务TMS	TMS FullAccess
	AI开发平台ModelArts	ModelArts FullAccess
	费用中心	BSS Administrator

如果系统预置的权限，不满足您的授权要求，可以创建自定义策略。自定义策略中可以添加的授权项（Action）请参考**ModelArts资源权限项**。

目前华为云云服务平台支持以下两种方式创建自定义策略：

- 可视化视图创建自定义策略：无需了解策略语法，按可视化视图导航栏选择云服务、操作、资源、条件等策略内容，可自动生成策略。
- JSON视图创建自定义策略：可以在选择策略模板后，根据具体需求编辑策略内容；也可以直接在编辑框内编写JSON格式的策略内容。

具体创建步骤请参见：**创建自定义策略**。下面为您介绍常用的ModelArts自定义策略样例。

- 示例1：授权镜像管理的权限。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:image:register",
                "modelarts:image:listGroup"
            ]
        }
    ]
}
```

- 示例2：拒绝用户创建、更新、删除专属资源池。

  拒绝策略需要同时配合其他策略使用，否则没有实际作用。用户被授予的策略中，一个授权项的作用如果同时存在Allow和Deny，则遵循**Deny优先原则**。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "swr:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
```


版权所有 © 华为云计算技术有限公司

```
            "smn:*:*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "modelarts:pool:create",
            "modelarts:pool:update",
            "modelarts:pool:delete"
        ],
        "Effect": "Deny"
    }
  ]
}
```

- 示例3：多个授权项策略。

  一个自定义策略中可以包含多个授权项，且除了可以包含本服务的授权项外，还可以包含其他服务的授权项，可以包含的其他服务必须跟本服务同属性，即都是项目级服务或都是全局级服务。多个授权语句策略描述如下：

```
{
  "Version": "1.1",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "modelarts:service:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lts:logs:list"
        ]
    }
  ]
}
```

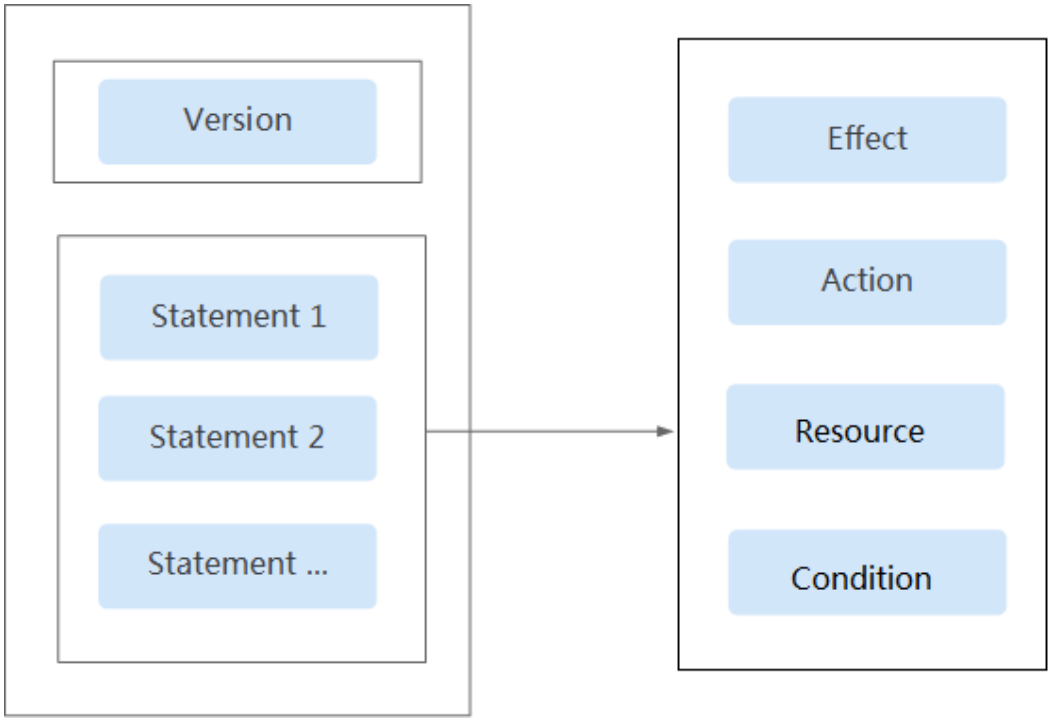
策略 JSON 格式字段介绍

策略结构

策略结构包括Version（策略版本号）和Statement（策略权限语句）两部分，其中Statement可以有多个，表示不同的授权项。

版权所有 © 华为云计算技术有限公司

图 2-3 策略结构



策略参数

下面介绍策略参数详细说明。了解策略参数后，您可以根据场景自定义策略。具体可以参考文档**自定义策略使用样例**。

表 2-4 策略参数说明

参数		含义	值
Version		策略的版本。	1.1：代表基于策略的访问控制。
Statement：策略的授权语句	Effect：作用	定义Action中的操作权限是否允许执行。	• Allow：允许执行。 • Deny：不允许执行。 **说明** 当同一个Action的Effect既有Allow又有Deny时，遵循Deny优先的原则。
	Action：授权项	操作权限。	格式为“服务名:资源类型:操作”。授权项支持通配符号*，通配符号*表示所有。 示例： "modelarts:notebook:list"：表示查看Notebook实例列表权限，其中modelarts为服务名，notebook为资源类型，list为操作。 您可以在对应服务“API参考”资料中查看该服务所有授权项。

版权所有 © 华为云计算技术有限公司

参数		含义	值
	Condition：条件	使策略生效的特定条件，包括**条件键**和**运算符**。	格式为“条件运算符:{条件键：[条件值1,条件值2]}”。 如果您设置多个条件，同时满足所有条件时，该策略才生效。 示例: "StringEndWithIfExists":{"g:UserName":["specialCharacter"]}：表示当用户输入的用户名以"specialCharacter"结尾时该条statement生效。
	Resource: 资源类型	策略所作用的资源。	格式为“服务名:<region>:<account-id>:资源类型:资源路径”，资源类型支持通配符号*，通配符号*表示所有。 **说明** ModelArts的授权不支持指定具体资源路径。

ModelArts 资源类型

管理员可以按ModelArts的资源类型选择授权范围。ModelArts支持的资源类型如下表：

表 2-5 ModelArts 资源类型（角色与策略授权）

资源类型	说明
notebook	开发环境的Notebook实例
exemlProject	自动学习项目
exemlProjectInf	自动学习项目的在线推理服务
exemlProjectTrain	自动学习项目的训练作业
exemlProjectVersion	自动学习项目的版本
workflow	Workflow项目
pool	专属资源池
network	专属资源池网络连接
trainJob	训练作业
trainJobLog	训练作业的运行日志
trainJobInnerModel	系统预置模型
model	模型
service	在线服务
nodeservice	边缘服务

版权所有 © 华为云计算技术有限公司

资源类型	说明
workspace	工作空间
dataset	数据集
dataAnnotation	数据集的标注信息
aiAlgorithm	训练算法
image	镜像
devserver	弹性裸金属

ModelArts 资源权限项

参考《 ModelArts API参考 》中的权限策略和授权项。

- **数据管理权限**
- **开发环境权限**
- **训练作业权限**
- **模型管理权限**
- **服务管理权限**

2.2.2 委托和依赖

功能依赖

功能依赖策略项

您在使用ModelArts服务中开发算法、管理训练作业过程中，需要和其他云服务交互，比如需要在提交训练作业时选择指定数据集OBS路径和日志存储OBS路径。因此管理员在为用户配置细粒度授权策略时，需要同时配置依赖的权限项，用户才能使用完整的功能。

说明

- 如果您使用根用户（与账户同名的缺省子用户）使用ModelArts，根用户默认拥有所有权限，不再需要单独授权。
- 请用户确保当前用户具备委托授权中包含的依赖策略项权限。例如，用户给ModelArts的委托需要授权SWR Admin权限，需要保证用户本身具备SWR Admin权限。

表 2-6 基本配置

业务场景	依赖的服务	依赖策略项	支持的功能
全局配置	IAM	iam:users:listUsers	查询用户列表（仅管理员需要）
基本功能	IAM	iam:tokens:assume	使用委托获取用户临时认证凭据（必需）

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
基本功能	BSS	bss:balance:view	在ModelArts控制台创建资源后，页面展示账号当前余额

表 2-7 管理工作空间

业务场景	依赖的服务	依赖策略项	支持的功能
工作空间	IAM	iam:users:listUsers	按用户进行工作空间授权
	ModelArts	modelarts:*:*delete*	删除工作空间时，同时清理空间内的资源

版权所有 © 华为云计算技术有限公司

表 2-8 管理开发环境 Notebook

业务场景	依赖的服务	依赖策略项	支持的功能
开发环境实例生命周期管理	ModelArts	modelarts:notebook:create modelarts:notebook:list modelarts:notebook:get modelarts:notebook:update modelarts:notebook:delete modelarts:notebook:start modelarts:notebook:stop modelarts:notebook:updateStopPolicy modelarts:image:delete modelarts:image:list modelarts:image:create modelarts:image:get modelarts:pool:list modelarts:tag:list modelarts:network:get aom:metric:get aom:metric:list aom:alarm:list	实例的启动、停止、创建、删除、更新等依赖的权限。
动态挂载存储配置	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	动态挂载存储配置。
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket	

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
镜像管理	ModelArts	modelarts:image:register modelarts:image:listGroup	在镜像管理中注册和查看镜像。
保存镜像	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 开发环境运行的实例，保存成镜像。 • 使用自定义镜像创建开发环境Notebook实例。
使用SSH功能	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	为开发环境Notebook实例配置登录密钥。
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list	
挂载SFS Turbo盘	SFS Turbo	SFS Turbo FullAccess	子用户对SFS目录的读写操作权限。专属池Notebook实例挂载SFS（公共池不支持），且挂载的SFS不是当前子用户创建的。
查看所有实例	ModelArts	modelarts:notebook:listAllNotebooks	ModelArts开发环境界面上，查询所有用户的实例列表，适用于给开发环境的实例管理员配置该权限。
	IAM	iam:users:listUsers	

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
VSCode插件（本地）/ PyCharm Toolkit（本地）	ModelArts	modelarts:notebook:listAllNotebooks modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJob:update modelarts:trainJobVersion:delete modelarts:trainJob:get modelarts:trainJob:logExport modelarts:workspace:getQuotas（如果开通了**工作空间**功能，则需要配置此权限。）	从本地VSCode连接云上的Notebook实例、提交训练作业等。

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	
	IAM	iam:projects:listProjects	从本地PyCharm查询IAM项目列表，完成连接配置。

版权所有 © 华为云计算技术有限公司

表 2-9 管理训练作业

业务场景	依赖的服务	依赖策略项	支持的功能
训练管理	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	创建训练作业和查看训练日志。
		modelarts:workspace:getQuota	查询工作空间配额。如果开通了**工作空间**功能，则需要配置此权限。
		modelarts:tag:list	在训练作业中使用标签管理服务TMS。
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	使用配置的委托授权项。
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	在训练作业中使用SFS Turbo。
	SWR	swr:repository:listTags swr:repository:getRepository swr:repository:listRepositories	使用自定义镜像运行训练作业。
	SMN	smn:topic:publish smn:topic:list	通过SMN通知训练作业状态变化事件。

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	使用OBS桶中的数据集运行训练作业。

表 2-10 使用 Workflow

业务场景	依赖的服务	依赖策略项	支持的功能
使用数据集	ModelArts	modelarts:dataset:getDataset modelarts:dataset:createDataset modelarts:dataset:createDatasetVersion modelarts:dataset:createImportTask modelarts:dataset:updateDataset modelarts:processTask:createProcessTask modelarts:processTask:getProcessTask modelarts:dataset:listDatasets	在工作流中使用ModelArts数据集

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
管理AI应用	ModelArts	modelarts:model:list modelarts:model:get modelarts:model:create modelarts:model:delete modelarts:model:update	在工作流中管理ModelArts AI应用
部署上线	ModelArts	modelarts:service:get modelarts:service:create modelarts:service:update modelarts:service:delete modelarts:service:getLogs	在工作流中管理ModelArts在线服务
训练作业	ModelArts	modelarts:trainJob:get modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJobVersion:list modelarts:trainJobVersion:create modelarts:trainJob:delete modelarts:trainJobVersion:delete modelarts:trainJobVersion:stop	在工作流中管理ModelArts训练作业
工作空间	ModelArts	modelarts:workspace:get modelarts:workspace:getQuotas	在工作流中使用ModelArts工作空间

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	在工作流中使用OBS数据
工作流运行	IAM	iam:users:listUsers（查询用户列表） iam:agencies:getAgency（查询指定委托详情） iam:tokens:assume（获取委托Token）	在工作流运行时，调用ModelArts其他服务
集成DLI	DLI	dli:jobs:get（查询作业详情） dli:jobs:list_all（查询作业列表） dli:jobs:create（创建新作业）	在工作流中集成DLI

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
集成MRS	MRS	mrs:job:get（查询作业详情） mrs:job:submit（创建并执行作业） mrs:job:list（查询作业列表） mrs:job:stop（停止作业） mrs:job:batchDelete（批量删除作业） mrs:file:list（查询文件列表）	在工作流中集成MRS

表 2-11 管理 AI 应用

业务场景	依赖的服务	依赖策略项	支持的功能
管理AI应用	SWR	swr:repository:deleteRepository swr:repository:deleteTag swr:repository:getRepository swr:repository:listTags	从自定义镜像导入 从OBS导入时使用自定义引擎

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	从OBS导入模型 模型转换指定OBS路径

表 2-12 管理部署上线

业务场景	依赖的服务	依赖策略项	支持的功能
在线服务	LTS	lts:logs:list（查询日志列表）	查询和展示LTS日志

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:GetBucketPolicy （获取桶策略） obs:bucket:HeadBucket （获取桶元数据） obs:bucket:ListAllMyBuckets （获取桶列表） obs:bucket:PutBucketPolicy （设置桶策略） obs:bucket:DeleteBucketPolicy （删除桶策略）	服务运行时容器挂载外部存储卷
批量服务	OBS	obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:bucket:CreateBucket（创建桶） obs:bucket:ListBucket（列举桶内对象） obs:bucket:ListAllMyBuckets（获取桶列表）	创建批量服务，批量推理。
边缘服务	CES	ces:metricData:list（查询指标数据）	查看服务的监控指标
	IEF	ief:deployment:delete（删除应用部署）	管理边缘服务

版权所有 © 华为云计算技术有限公司

表 2-13 管理数据集

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据集和标注	OBS	obs:bucket:ListBucket（列举桶内对象） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:GetBucketAcl（获取桶ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:bucket:GetBucketPolicy（获取桶策略） obs:bucket:PutBucketPolicy（设置桶策略） obs:bucket:DeleteBucketPolicy（删除桶策略） obs:bucket:PutBucketCORS（设置桶的CORS配置、删除桶的CORS配置） obs:bucket:GetBucketCORS（获取桶的CORS配置） obs:object:PutObjectAcl（设置对象ACL）	管理OBS中的数据集 标注OBS数据 创建数据管理作业
管理表格数据集	DLI	dli:database:displayAllDatabases dli:database:displayAllTables dli:table:describe_table	在数据集中管理DLI数据
管理表格数据集	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail	在数据集中管理DWS数据
管理表格数据集	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:cluster:get	在数据集中管理MRS数据

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
智能标注	ModelArts	modelarts:service:list modelarts:model:list modelarts:model:get modelarts:model:create modelarts:trainJobInnerModel:list modelarts:workspace:get modelarts:workspace:list	使用智能标注
团队标注	IAM	iam:projects:listProjects（查询租户项目） iam:users:listUsers（查询用户列表） iam:agencies:createAgency（创建委托） iam:quotas:listQuotasForProject（查询指定项目的配额）	管理标注团队

表 2-14 资源管理

业务场景	依赖的服务	依赖策略项	支持的功能
资源池管理	BSS	bss:coupon:view bss:order:view bss:balance:view bss:discount:view bss:renewal:view bss:bill:view bss:contract:update bss:order:pay bss:unsubscribe:update bss:renewal:update bss:order:update	资源池的创建、续费、退订等与计费相关的功能。依赖权限需要配置在IAM项目视图中。
	CCE	cce:cluster:list cce:cluster:get	获取CCE集群列表、集群详情、集群证书等信息。依赖权限需要配置在IAM项目视图中。

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
	KMS	kms:cmk:list kms:cmk:getMaterial	获取用户创建的密钥对列表信息。依赖权限需要配置在IAM项目视图中。
	AOM	aom:metric:get	获取资源池的监控数据。依赖权限需要配置在IAM项目视图中。
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion	获取AI诊断日志。依赖权限需要配置在IAM项目视图中。
	ECS	ecs:availabilityZones:list	查询可用区列表。依赖权限需要配置在IAM项目视图中。

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	依赖策略项	支持的功能
网络管理	VPC	vpc:routes:create vpc:routes:list vpc:routes:get vpc:routes:delete vpc:peerings:create vpc:peerings:accept vpc:peerings:get vpc:peerings:delete vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:vpcs:create vpc:vpcs:list vpc:vpcs:get vpc:vpcs:delete vpc:subnets:create vpc:subnets:get vpc:subnets:delete vpcep:endpoints:list vpcep:endpoints:create vpcep:endpoints:delete vpcep:endpoints:get vpc:ports:create vpc:ports:get vpc:ports:update vpc:ports:delete vpc:networks:create vpc:networks:get vpc:networks:update vpc:networks:delete	ModelArts网络资源创建和删除、VPC网络打通。依赖权限需要配置在IAM项目视图中。
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	用户的网络和SFS Turbo资源打通。依赖权限需要配置在IAM项目视图中。

版权所有 © 华为云计算技术有限公司

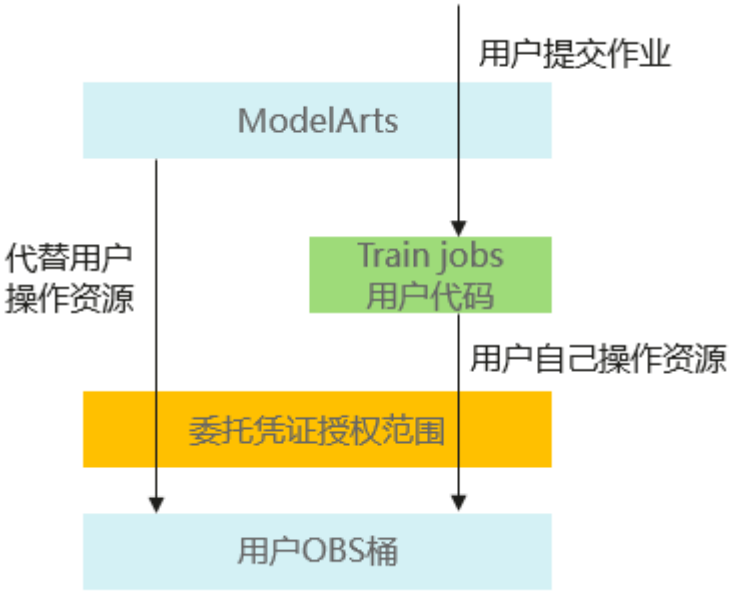
业务场景	依赖的服务	依赖策略项	支持的功能
边缘资源池	IEF	ief:node:list ief:group:get ief:application:list ief:application:get ief:node:listNodeCert ief:node:get ief:IEFInstance:get ief:deployment:list ief:group:listGroupInstanceState ief:IEFInstance:list ief:deployment:get ief:group:list	边缘池增删改查管理

委托授权

用户在使用ModelArts服务运行作业过程中，为了简化用户的操作，ModelArts后台可以代替用户完成一些工作，如训练作业启动前自动下载用户OBS桶中的数据集到作业空间、自动转储训练作业日志到用户OBS桶中。

ModelArts服务不会保存用户的Token认证凭据，在后台异步作业中操作用户的资源（如OBS桶）前，需要用户通过IAM委托向ModelArts显式授权，ModelArts在需要时使用用户的委托获取临时认证凭据用于操作用户资源，见“**添加授权**”。

图 2-4 委托授权



版权所有 © 华为云计算技术有限公司

如图2-4所示，用户向ModelArts授权后，ModelArts使用委托授权的临时凭证访问和操作用户资源，协助用户自动化一些繁琐和耗时的操作。同时，委托凭证会同步到用户的作业中（Notebook实例和训练作业），客户在作业中可以使用委托凭证自行访问自己的资源。

在ModelArts服务中委托授权有两种方式：

1、一键式委托授权

ModelArts提供了一键式自动授权功能，用户可以在ModelArts的全局配置功能中，快速完成委托授权，由ModelArts为用户自动创建委托并配置到ModelArts服务中。

这种方式为保证使用业务过程中有足够的权限，基于依赖服务的预置系统策略指定授权范围，创建的委托的权限比较大，基本覆盖了依赖服务的全部权限。如果您需要对委托授权的权限范围进行精确控制，请使用第二种方式。

2、定制化委托授权

管理员在IAM中为不同用户创建不同的委托授权策略，再到ModelArts中为用户配置已创建好的委托。管理员在IAM中为用户创建委托时，根据用户的实际权限范围为委托指定最小权限范围，控制用户在使用ModelArts过程中可访问的资源内容。具体参考**配置ModelArts基本使用权限**。

委托授权的越权风险

可以看到用户的委托授权是独立的，理论上用户的委托授权范围是可以超出用户自身用户组的授权策略的授权范围，如果配置不当就会出现用户越权的问题。

为了控制委托授权的越权风险，ModelArts服务的全局配置功能要求只有租户管理员才能为用户配置委托，由管理员保证委托授权的安全性。

委托授权的最小化

管理员在配置委托授权时，应严格控制授权的范围。

ModelArts为用户异步自动化完成作业的准备、清理等操作，所需的委托授权内容是基础授权范围。如果用户只使用ModelArts的部分功能，管理员可以依据委托授权表格的说明屏蔽不使用的基础权限项。相反地，如果用户需要在作业中使用基础授权范围外的资源权限，管理员也可以为用户在委托授权中增加新的权限项。总之，委托授权的范围应该基于实际业务场景所需权限范围来进行定制，保持委托授权范围的最小化。

委托基础授权范围

当您需要定制委托授权的权限列表时，请参考下面表格，根据实际业务选择授权项。

版权所有 © 华为云计算技术有限公司

表 2-15 开发环境基础委托授权

业务场景	依赖的服务	委托授权项	说明	配置建议
JupyterLab	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	通过ModelArts的Notebook，在JupyterLab中使用OBS上传下载数据。	建议配置。
开发环境监控功能	AOM	aom:alarm:put	调用AOM的接口，获取Notebook相关的监控数据和事件，展示在ModelArts的Notebook中。	建议配置。

表 2-16 训练作业基础委托授权

业务场景	依赖的服务	委托授权项	说明
训练作业	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	训练作业启动前下载数据、模型、代码。 训练作业运行中上传日志、模型。

表 2-17 部署上线基础委托授权

业务场景	依赖的服务	委托授权项	说明
在线服务	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	在线服务配置LTS日志上报

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	委托授权项	说明
批量服务	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	运行批量服务
边缘服务	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	通过IEF部署边缘服务

表 2-18 数据管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
数据集和数据标注	OBS	obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:object:PutObjectAcl obs:bucket:ListBucket obs:bucket:HeadBucket obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:GetBucketPolicy obs:bucket:PutBucketPolicy obs:bucket:DeleteBucketPolicy obs:bucket:PutBucketCORS obs:bucket:GetBucketCORS	管理OBS桶中的数据集
数据标注	ModelArts推理	modelarts:service:get modelarts:service:create modelarts:service:update	基于ModelArts推理进行智能数据标注

版权所有 © 华为云计算技术有限公司

表 2-19 专属资源池管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
资源池网络管理（新版本）	VPC	vpc:routes:create vpc:routes:list vpc:routes:get vpc:routes:delete vpc:peerings:create vpc:peerings:accept vpc:peerings:get vpc:peerings:delete vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:vpcs:create vpc:vpcs:list vpc:vpcs:get vpc:vpcs:delete vpc:subnets:create vpc:subnets:get vpc:subnets:delete vpcep:endpoints:list vpcep:endpoints:create vpcep:endpoints:delete vpcep:endpoints:get vpc:ports:create vpc:ports:get vpc:ports:update vpc:ports:delete vpc:networks:create vpc:networks:get vpc:networks:update vpc:networks:delete	ModelArts网络资源创建和删除、VPC网络打通。此处依赖权限需要配置在IAM项目视图中。
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	用户的网络和SFS Turbo资源打通。依赖权限需要配置在IAM项目视图中。

版权所有 © 华为云计算技术有限公司

业务场景	依赖的服务	委托授权项	说明
资源池管理	BSS	bss:coupon:view bss:order:view bss:balance:view bss:discount:view bss:renewal:view bss:bill:view bss:contract:update bss:order:pay bss:unsubscribe:update bss:renewal:update bss:order:update	资源池的创建、续费、退订等与计费相关的功能。依赖权限需要配置在IAM项目视图中。
资源池管理	ECS	ecs:availabilityZones:list	查询可用区列表。依赖权限需要配置在IAM项目视图中。

2.2.3 工作空间

ModelArts的用户需要为不同的业务目标开发算法、管理和部署模型，此时可以创建多个工作空间，把不同应用开发过程的输出内容划分到不同工作空间中，便于管理和使用。

工作空间支持3种访问控制：

- PUBLIC：租户（主账号和所有子账号）内部公开访问。
- PRIVATE：仅创建者和主账号可访问。
- INTERNAL：创建者、主账号、指定IAM子账号可访问当授权类型为INTERNAL时需要指定可访问的子账号的账号名，可选择多个。

每个账号每个IAM项目都会分配1个默认工作空间，默认工作空间的访问控制为PUBLIC。

通过工作空间的访问控制能力，可限制仅允许部分人访问对应的工作空间。通过此功能可实现类似如下场景：

- **教育场景**：老师可给每个学生分配1个INTERNAL的工作空间并且限制该工作空间被指定学生访问，这样可使得学生可独立完成在ModelArts上的实验。
- **企业场景**：管理者可创建用于生产任务的工作空间并限制仅让运维人员使用，用于日常调试的工作空间并限制仅让开发人员使用。通过这种方式让不同的企业角色只能在指定工作空间下使用资源。

目前工作空间功能是“受邀开通”状态，作为企业用户您可以通过您对口的技术支持申请开通。

版权所有 © 华为云计算技术有限公司

2.3 典型场景配置实践

2.3.1 个人用户快速配置 ModelArts 访问权限

ModelArts使用过程中涉及到OBS、SWR等服务交互，需要用户配置委托授权，允许ModelArts访问这些依赖服务。如果没有授权，ModelArts的部分功能将不能正常使用。

约束与限制

- 只有主账号可以使用委托授权，可以为当前账号授权，也可以为当前账号下的所有IAM用户授权。
- 多个IAM用户或账号，可使用同一个委托。
- 一个账号下，最多可创建50个委托。
- 对于首次使用ModelArts新用户，请直接新增委托即可。一般用户新增普通用户权限即可满足使用要求。如果有精细化权限管理的需求，可以自定义权限按需设置。
- 如果未获得委托授权，当打开“访问授权”页面时，ModelArts会提醒您当前用户未配置授权，需联系此IAM用户的管理员账号进行委托授权。

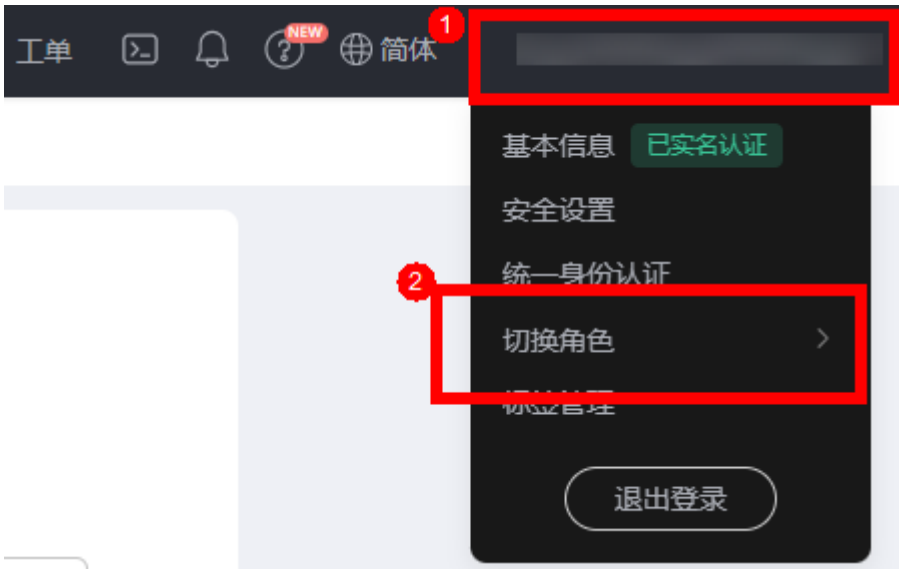
添加授权

1. 登录ModelArts管理控制台，在左侧导航栏选择“全局配置”，进入“全局配置”页面。
2. 单击“添加授权”，进入“访问授权”配置页面，根据参数说明进行配置。

表 2-20 参数说明

参数	说明
“授权对象类型”	包括IAM子用户、联邦用户、委托用户和所有用户。 • IAM子用户：由主账号在IAM中创建的用户，是服务的使用人员，具有独立的身份凭证（密码和访问密钥），根据账号授予的权限使用资源。IAM子用户相关介绍请参见**IAM用户介绍**。 • 联邦用户：又称企业虚拟用户。联邦用户相关介绍请参见**联邦身份认证**。 • 委托用户：IAM中创建的一个委托。IAM创建委托相关介绍请参见**创建委托**。 • 所有用户：该选项表示会将委托的权限授权到当前账号下的所有子账号、包括未来创建的子账号，授权范围较大，需谨慎使用。个人用户选择“所有用户”即可。

版权所有 © 华为云计算技术有限公司

参数	说明
“授权对象”	“授权对象类型”选择“所有用户”时不涉及此参数。 • IAM子用户：选择指定的IAM子用户，给指定的IAM子用户配置委托授权。 **图 2-5** 选择 IAM 子用户  • 联邦用户：输入联邦用户的用户名或用户ID。 **图 2-6** 选择联邦用户  • 委托用户：选择委托名称。使用账号A创建一个权限委托，在此处将该委托授权给账号B拥有的委托。在使用账号B登录控制台时，可以在控制台右上角的个人账号切换角色到账号A，使用账号A的委托权限。 **图 2-7** 委托用户切换角色 
“委托选择”	• 已有委托：列表中如果已有委托选项，则直接选择一个可用的委托为上述选择的用户授权。单击委托名称查看该委托的权限详情。 • 新增委托：如果没有委托可选，可以在新增委托中创建委托权限。对于首次使用ModelArts的用户，需要新增委托。
“新增委托 > 委托名称”	系统自动创建委托名称，用户可以手动修改。

版权所有 © 华为云计算技术有限公司

参数	说明
“新增委托 > 授权方式”	• 角色授权：IAM最初提供的一种根据用户的工作职能定义权限的粗粒度授权机制。该机制以服务为粒度，提供有限的服务相关角色用于授权。由于华为云各服务之间存在业务依赖关系，因此给用户授予角色时，可能需要一并授予依赖的其他角色，才能正确完成业务。角色并不能满足用户对精细化授权的要求，无法完全达到企业对权限最小化的安全管控要求。 • 策略授权：IAM最新提供的一种细粒度授权的能力，可以精确到具体服务的操作、资源以及请求条件等。基于策略的授权是一种更加灵活的授权方式，能够满足企业对权限最小化的安全管控要求。 角色与策略相关介绍请参考**权限基本概念**。
“新增委托 > 权限配置 > 普通用户”	普通用户包括用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练任务的创建和管理等。一般用户选择此项即可。 可以单击“查看权限列表”，查看普通用户权限。
“新增委托 > 权限配置 > 自定义”	如用户有精细化权限管理的需求，可使用自定义模式灵活按需配置ModelArts创建的委托权限。可以根据实际需要在权限列表中勾选要配置的权限。

3. 然后勾选“我已经详细阅读并同意《ModelArts服务声明》”，单击“创建”，即可完成委托配置。

查看授权的权限列表

用户可以在“全局配置”页面的授权列表中，查看已经配置的委托授权内容。单击授权内容列的“查看权限”，可以查看该授权的权限详情。

图 2-8 查看权限

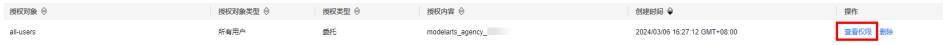
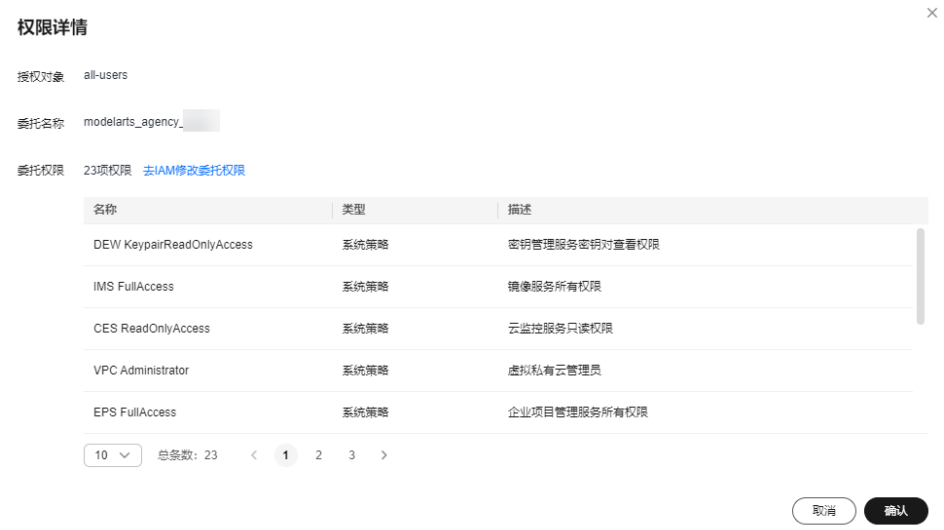


图 2-9 普通用户权限列表



版权所有 © 华为云计算技术有限公司

2.3.2 配置 ModelArts 基本使用权限

2.3.2.1 场景描述

ModelArts作为顶层服务，其部分功能依赖于其他服务的访问权限。本章节主要介绍对于IAM子用户使用ModelArts时，如何根据需要开通的功能配置子用户相应权限。

权限列表

子用户的权限，由主用户来控制，主用户通过IAM的权限配置功能设置用户组的权限，从而控制用户组内的子用户的权限。此处的授权列表均按照ModelArts和其他服务的系统预置策略来举例。

表 2-21 服务授权列表

待授权的服务	授权说明	IAM权限设置	是否必选
ModelArts	授予子用户使用ModelArts服务的权限。 ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。	ModelArts CommonOperations	必选
	如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。	ModelArts FullAccess	可选 ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。
OBS对象存储服务	授予子用户使用OBS服务的权限。ModelArts的数据管理、开发环境、训练作业、模型推理部署均需要**通过OBS进行数据中转**。	OBS OperateAccess	必选
SWR容器镜像仓库	授予子用户使用SWR服务权限。ModelArts的**自定义镜像功能**依赖镜像服务SWR FullAccess权限。	SWR OperateAccess	必选
密钥管理服务	当子用户使用ModelArts **Notebook的SSH远程功能**时，需要配置子用户密钥管理服务的使用权限。	KMS CMKFullAccess	可选

版权所有 © 华为云计算技术有限公司

待授权的服务	授权说明	IAM权限设置	是否必选
IEF智能边缘平台	授予子用户智能边缘平台使用权限，ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。	Tenant Administrator	可选
CES云监控	授予子用户使用CES云监控服务的权限。通过CES云监控可以查看ModelArts的在线服务和对应模型负载运行状态的整体情况，并设置监控告警。	CES FullAccess	可选
SMN消息服务	授予子用户使用SMN消息服务的权限。SMN消息通知服务配合CES监控告警功能一起使用。	SMN FullAccess	可选
VPC虚拟私有云	子用户在创建ModelArts的专属资源池过程中，如果需要开启自定义网络配置，需要配置VPC权限。	VPC FullAccess	可选
SFS弹性文件服务	授予子用户使用SFS服务的权限，ModelArts的专属资源池中可以挂载SFS系统作为开发环境或训练的存储。	SFS Turbo FullAccess SFS FullAccess	可选

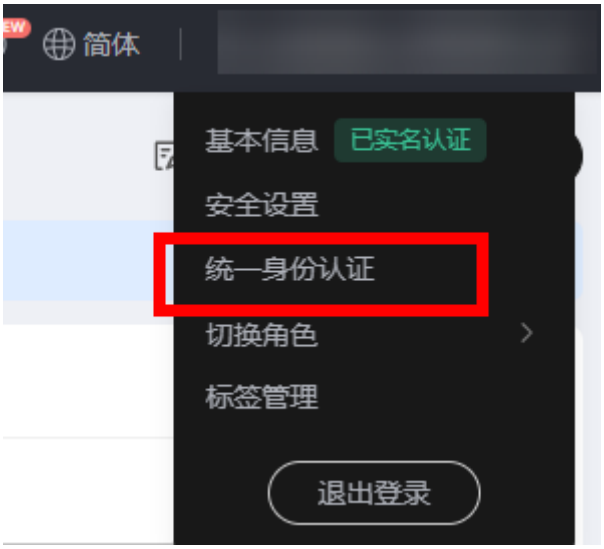
2.3.2.2 Step1 创建用户组并加入用户

主用户账号下面可以创建多个子用户，并对子用户的权限进行分组管理。此步骤介绍如何创建用户组、子用户、并将子用户加入用户组中。

1. 主用户登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

版权所有 © 华为云计算技术有限公司

图 2-10 统一身份认证



2. 创建用户组。在左侧菜单栏中，选择“用户组”。单击右上角“创建用户组”，在“用户组名称”中填入“用户组02”，然后单击“确定”完成用户组创建。

   创建完成后，返回用户组列表。通过用户组管理，将已有子用户加入到用户组中。如果没有子用户账号，可以创建子用户并加入用户组。

3. 创建子用户账号并加入用户组。在IAM左侧菜单栏中，选择“用户”，单击右上角“创建用户”，在“创建用户”页面中，添加多个用户。

   请根据界面提示，填写必选参数，然后单击“下一步”。

4. 在“加入用户组”步骤中，选择“用户组02”，然后单击“创建用户”。

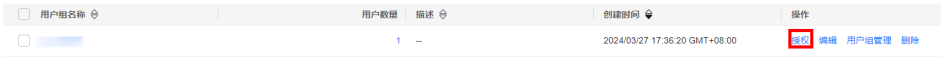
   系统将逐步创建好前面设置的2个用户。

2.3.2.3 Step2 为用户配置云服务使用权限

主用户为子用户授予ModelArts、OBS等云服务的使用权限后，子用户才可以使用这些云服务。此步骤介绍如何为用户组中的所有子用户授予使用ModelArts、OBS、SWR等各类云服务的权限。

1. 主用户在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子用户配置权限。

   图 2-11 为用户组授权



2. 配置授权前，请先了解ModelArts各模块使用到的最小权限要求，如表2-21所示。
3. 配置ModelArts使用权限。在搜索框搜索ModelArts。ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。

   选择说明如下：

   - ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。
   - 如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。

版权所有 © 华为云计算技术有限公司

4. 配置OBS使用权限。搜索OBS，勾选“ OBS Administrator”。ModelArts训练作业中需要依赖OBS作为数据中转站，需要配置OBS的使用权限。
5. 配置SWR使用权限。搜索SWR，勾选“SWR FullAccess”。ModelArts的自定义镜像功能依赖镜像服务SWR FullAccess权限。
6. （可选）配置密钥管理权限。如果需要使用ModelArts Notebook的SSH访问功能，依赖密钥管理权限。搜索DEW，勾选“DEW KeypairFullAccess”。

   此处需要注意以下Region配置的是DEW密钥管理权限：华北-北京一、华北-北京四、华东-上海一、华东-上海二、华南-广州、西南-贵阳一、中国-香港、亚太-新加坡。其他Region配置的是KMS密钥管理权限。本示例中使用“华南-广州”Region举例，所以需要配置DEW密钥管理权限。
7. （可选）配置智能边缘平台使用权限。ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。

   注意：Tenant Administrator权限比较大，包含全部云服务的管理权限，而不仅是使用ModelArts服务。请谨慎配置。
8. （可选）配置CES云监控和SMN消息通知使用权限。ModelArts推理部署的在线服务详情页面内有调用次数详情，单击可查看该在线服务的调用次数随时间详细分布的情况。如果想进一步通过CES云监控查看ModelArts的在线服务和对应模型负载运行状态的整体情况，需要给子用户授予CES权限。

   如果只是查看监控，给子用户授予CES ReadOnlyAccess权限即可。

   如果还需要在CES上设置监控告警，则需要再加上CES FullAccess权限，以及SMN消息通知权限。
9. （可选）配置VPC权限。如果用户在创建专属资源池过程中，需要开启自定义网络配置，此处需要授予用户VPC权限。
10. （可选）配置SFS和SFS Turbo权限。如果用户在专属资源池中挂载SFS系统作为开发环境或训练的存储时，需要授予使用权限。
11. 单击左上角的“查看已选”，确认已勾选的权限。
12. 再单击“下一步”，设置最小授权范围。单击“指定区域项目资源”，勾选待授权使用的区域，单击“确定”。
13. 提示授权成功，查看授权信息，单击“完成”。此处的授权生效需要15-30分钟。

2.3.2.4 Step3 为用户配置 ModelArts 的委托访问授权

配置完IAM权限之后，需要在ModelArts页面为子用户设置ModelArts访问授权，允许ModelArts访问OBS、SWR、IEF等依赖服务。

此方式只允许主用户为子用户进行配置。因此，本示例中，管理员账号需为所有用户完成访问授权的配置。

1. 使用主用户的账号登录ModelArts服务管理控制台。请注意选择左上角的区域。
2. 在左侧导航栏单击“全局配置”，进入“全局配置”页面。
3. 单击“添加授权”。在“授权”页面，在“授权对象类型”下面选择“所有用户”，选择“新增委托 ”，为该主用户下面的所有子用户配置委托访问授权。
   - 普通用户：普通用户的委托权限包括了用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练任务的创建和管理等。一般用户选择此项即可。
   - 自定义：如果对用户有更精细化的权限管理需求，可使用自定义模式灵活按需配置ModelArts创建的委托权限。可以根据实际需在权限列表中勾选要配置的权限。

版权所有 © 华为云计算技术有限公司

4. 勾选“我已经详细阅读并同意《ModelArts服务声明》”，单击“创建”，完成委托授权配置。

2.3.2.5 Step4 测试用户权限

由于4中的权限需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。

2. 验证ModelArts权限。
   a. 在左上角选择区域，区域需与授权配置中的区域相同。
   b. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，界面未提示权限不足，表明ModelArts的使用权限和委托授权配置成功。

      如果提示“需获取依赖服务的授权”，说明未配置ModelArts委托访问授权，请参考**Step3 为用户配置ModelArts的委托访问授权**，使用主用户为子用户配置ModelArts委托访问授权。
   c. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，单击“创建”，如果可以正常打开创建页面，说明具备ModelArts的操作权限。

      您也可以尝试其他功能，例如“训练管理>训练作业”等，如能正常打开创建页面，即可正常使用ModelArts。
3. 验证OBS权限。
   a. 在左上角的服务列表中，选择OBS服务，进入OBS管理控制台。
   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。
4. 验证SWR权限。
   a. 在左上角的服务列表中，选择SWR服务，进入SWR管理控制台。
   b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。
5. 依次验证其他可选权限。
6. 验证结束，当前用户同时具备ModelArts部分功能的操作权限，可正常开始使用ModelArts服务。

2.3.3 管理员和开发者权限分离

对于中小规模团队，管理员希望对ModelArts资源进行主导分配，全局控制，而对于普通开发者只需关注自己实例的生命周期控制。对于开发者账号，一般不会具有te_admin的权限，相应的权限也需要主账号进行统一配置。本章节以使用Notebook进行项目开发为例，通过自定义策略配置实现管理员和开发者分离。

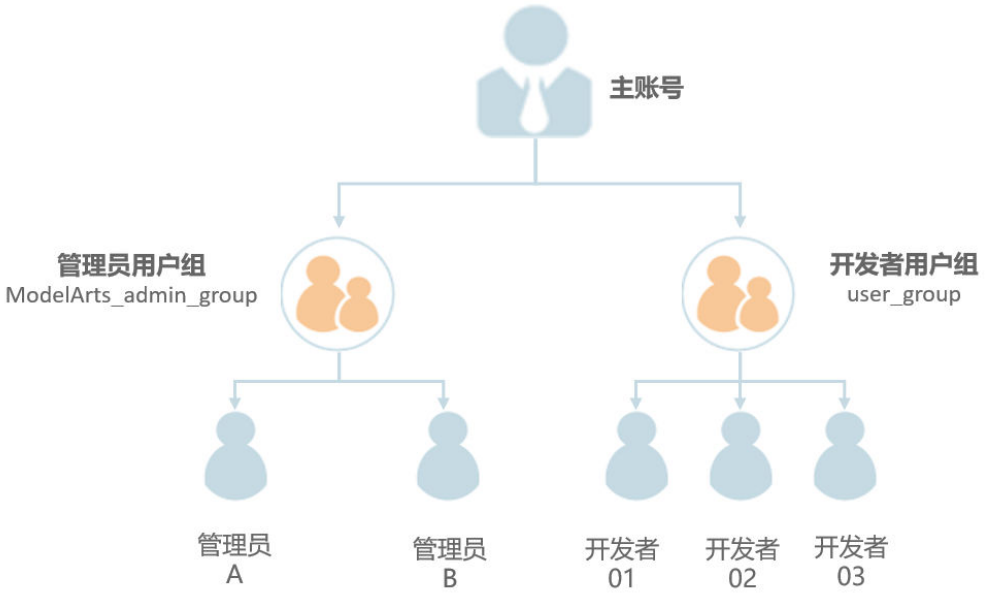
场景描述

以使用Notebook进行项目开发为例，管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。

普通开发者使用开发环境，只需关注对自己Notebook实例的操作权限，包括对自己实例的创建、启动、停止、删除等权限以及周边依赖服务的权限。普通开发者不需要ModelArts专属资源池的操作权限，也不需要查看其他用户的Notebook实例。

版权所有 © 华为云计算技术有限公司

图 2-12 账号关系示意图



配置管理员权限

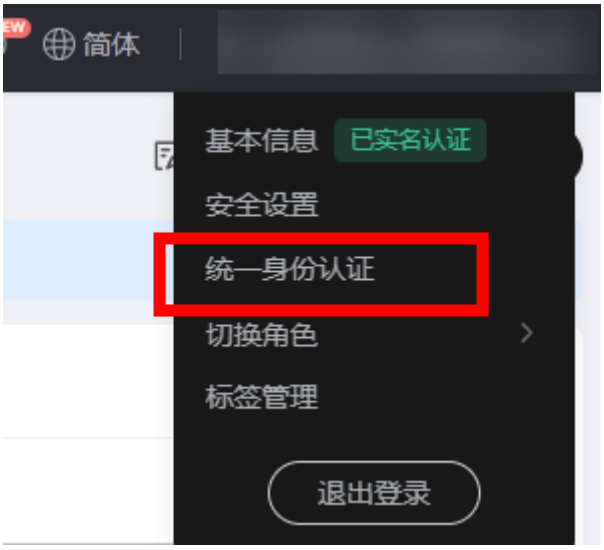
管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。可以通过以下配置流程实现管理员权限配置。

步骤1 使用主账号创建一个管理员用户组ModelArts_admin_group，将管理员账号加入用户组ModelArts_admin_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用管理员账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

图 2-13 登录控制台



2. 创建自定义策略1，赋予用户IAM和OBS服务权限。在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，在

版权所有 © 华为云计算技术有限公司

“策略名称”中填入“Policy1_IAM_OBS”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

自定义策略“Policy1_IAM_OBS”的具体内容如下，赋予用户IAM和OBS操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:users:listUsers",
                "iam:projects:listProjects",
                "obs:object:PutObject",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:bucket:HeadBucket",
                "obs:object:DeleteObject",
                "obs:bucket:CreateBucket",
                "obs:bucket:ListBucket"
                ]
        }
    ]
}
```

3. 重复**步骤2.2**创建自定义策略2，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。“策略名称”为“Policy2_AllowOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy2_AllowOperation”的具体内容如下，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:serverKeypairs:list",
                "ecs:serverKeypairs:get",
                "ecs:serverKeypairs:delete",
                "ecs:serverKeypairs:create",
                "swr:repository:getNamespace",
                "swr:repository:listNamespaces",
                "swr:repository:deleteTag",
                "swr:repository:getRepository",
                "swr:repository:listTags",
                "swr:instance:createTempCredential",
                "mrs:cluster:get",
                "modelarts:*:*"
            ]
        }
    ]
}
```

步骤3 将**步骤2**创建的自定义策略授权给管理员用户组ModelArts_admin_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称ModelArts_admin_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”和“Policy2_AllowOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给管理员用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录ModelArts的管理控制台，在左侧导航栏单击“全局配置”，进入“全局配置”页面。

版权所有 © 华为云计算技术有限公司

2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择管理员的账号，选择“新增委托”，“权限配置”选择“普通用户”。管理员不做权限控制，此处默认使用普通用户委托即可。
3. 勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，单击“创建”。

步骤5 测试管理员用户权限。

1. 使用管理员用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts控制台的左侧导航栏中，选择“专属资源池”，单击创建，未提示权限不足，表明管理员用户的权限配置成功。

----结束

配置开发者权限

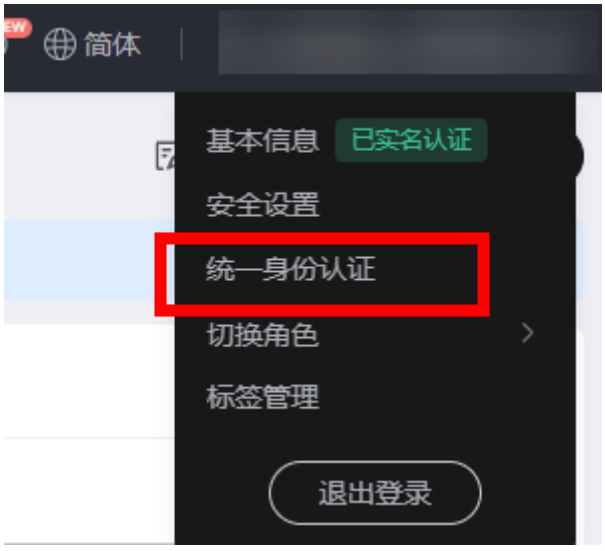
开发者权限需要通过IAM的细粒度授权控制实现，可以通过以下配置流程实现开发者权限配置。

步骤1 使用主账号创建一个开发者用户组user_group，将开发者账号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用主账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

   图 2-14 登录控制台



2. 创建自定义策略3，拒绝用户操作ModelArts专属资源池并拒用户查看其他用户的Notebook。

   在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy3_DenyOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy3_DenyOperation”的具体内容如下，可以直接复制粘贴。

```
{
    "Version": "1.1",
```


版权所有 © 华为云计算技术有限公司

```
  "Statement": [
        {
         "Effect": "deny",
         "Action": [
             "modelarts:pool:create",
             "modelarts:pool:update",
             "modelarts:pool:delete",
      "modelarts:notebook:listAllNotebooks"
          ]

      }
   ]
}
```

步骤3 将自定义策略授权给开发者用户组user_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”、“Policy2_AllowOperation”和“Policy3_DenyOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给开发者用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录ModelArts的管理控制台，在左侧导航栏单击“全局配置”，进入“全局配置”页面。
2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“新增委托”，“委托名称”设置为“ma_agency_develop_user”，“权限配置”选择“自定义”，“权限名称”勾选“OBS Administrator”。开发者用户只需要配置OBS的委托授权即可，允许开发者用户在使用Notebook时，与OBS服务交互。
3. 单击“创建”。
4. 在“全局配置”页面，再次单击“添加授权”，进入“访问授权”页面，为其他开发者用户配置委托。

   “授权对象类型”选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“已有委托”，“委托名称”勾选上一步创建的“ma_agency_develop_user”，

步骤5 测试开发者用户权限。

1. 使用user_group用户组中任意一个子用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts左侧菜单栏中，选择“专属资源池”，单击创建，界面未提示权限不足，表明开发者用户的权限配置成功。

----结束

2.3.4 查看所有子账号的 Notebook 实例

当子用户被授予“listAllNotebooks”和“listUsers”权限时，在Notebook页面上，单击“查看所有”，可以看到IAM项目下所有子用户创建的Notebook实例 。

说明

配置该权限后，可以查看子用户的Notebook，也可以在Notebook中访问子用户的OBS、SWR等。

版权所有 © 华为云计算技术有限公司

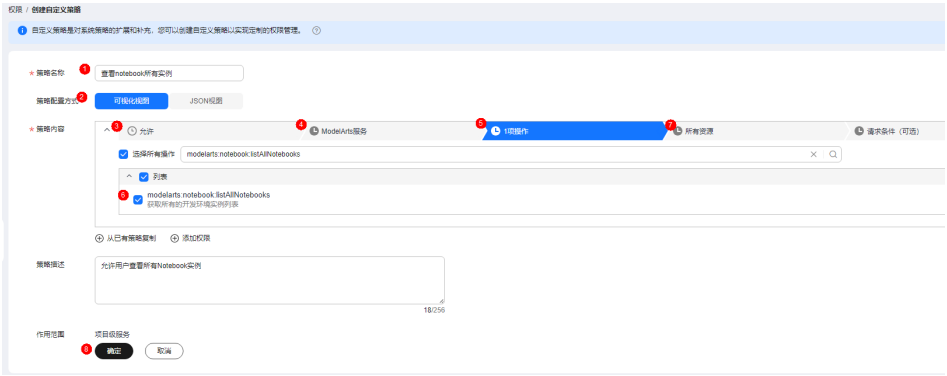
给子账号配置查看所有 Notebook 实例的权限

1. 使用主用户账号登录ModelArts管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，需要设置两条策略。

   策略1：设置查看Notebook所有实例，如图2-15所示，单击“确定”。

   - “策略名称”：设置自定义策略名称，例如：查看Notebook所有实例。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:notebook:listAllNotebooks并选中，所有资源选择默认值。

   图 2-15 创建自定义策略

   策略2：设置查看Notebook实例创建者信息的策略。

   - “策略名称”：设置自定义策略名称，例如：查看所有子用户信息。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索IAM服务并选中，操作列中搜索关键词iam:users:listUsers并选中，所有资源选择默认值。
3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的两条自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限查看该用户组内成员创建的所有Notebook实例。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

子用户启动其他用户的 SSH 实例

子用户可以看到所有用户的Notebook实例后，如果要通过SSH方式远程连接其他用户的Notebook实例，需要将SSH密钥对更新成自己的，否则会报错ModelArts.6786。更新密钥对具体操作请参见**修改Notebook SSH远程连接配置**。

具体的错误信息提示：ModelArts.6789: 在ECS密钥对管理中找不到指定的ssh密钥对xxx，请更新密钥对并重试。

版权所有 © 华为云计算技术有限公司

2.3.5 使用 Cloud Shell 登录训练容器

使用场景

允许用户使用ModelArts控制台提供的Cloud Shell登录运行中的训练容器。

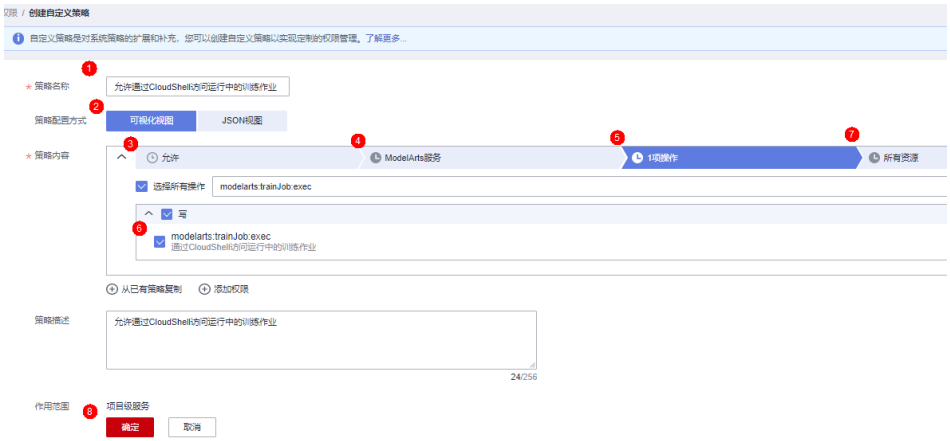
约束限制

仅专属资源池均支持使用Cloud Shell，且训练作业必须处于“运行中”状态。

前提条件：给子账号配置允许使用 Cloud Shell 的权限

1. 使用主用户账号登录华为云的管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”按如下要求设置完成后单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：允许通过Cloud Shell访问运行中的训练作业。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:trainJob:exec并选中，所有资源选择默认值。

   图 2-16 创建自定义策略

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限通过Cloud Shell登录运行中的训练作业容器。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

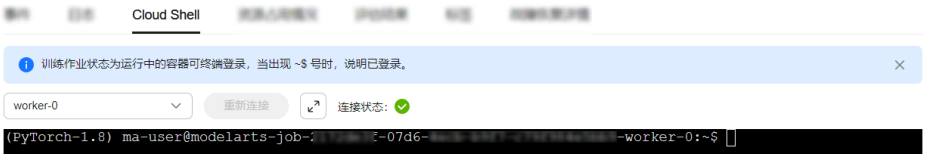
如何使用 Cloud Shell

1. 参考**前提条件：给子账号配置允许使用Cloud Shell的权限**，完成配置。
2. 在ModelArts管理控制台的左侧导航栏中选择“训练管理 > 训练作业”。

版权所有 © 华为云计算技术有限公司

3. 在训练作业列表中，单击作业名称进入训练作业详情页面。
4. 在训练作业详情页面，单击“Cloud Shell”页签，登录训练容器。
   连接成功后，Cloud Shell界面提示如下。

图 2-17 Cloud Shell 界面



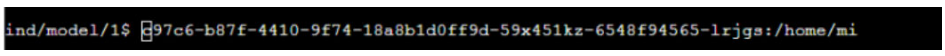
当作业处于非运行状态或权限不足时会导致无法使用Cloud Shell，请根据提示定位原因即可。

图 2-18 报错提示

说明

部分用户登录Cloud Shell界面时，可能会出现路径显示异常情况，此时在Cloud Shell中单击回车键即可恢复正常。

图 2-19 路径异常



2.3.6 限制用户使用公共资源池

本章节介绍如何控制ModelArts用户权限，限制用户使用ModelArts公共资源池的资源创建训练作业、创建开发环境实例，部署推理服务等。

场景介绍

对于ModelArts专属资源池的用户，不允许使用公共资源池创建训练作业、创建Notebook实例或者部署推理服务时，可以通过权限控制限制用户使用公共资源池。

涉及配置的自定义权限策略项如下;

- modelarts:notebook:create：此策略项表示创建Notebook实例。
- modelarts:trainJob:create：此策略项表示创建训练作业。

版权所有 © 华为云计算技术有限公司

- modelarts:service:create：此策略项表示创建推理服务。

给子账号配置权限：限制使用公共资源池

1. 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略，单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：不允许用户使用公共资源池创建。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：拒绝，云服务中搜索“ModelArts”服务并选中，“操作”中查找**写**操作“modelarts:trainJob:create”、“modelarts:notebook:create”和“modelarts:service:create”并选中。“所有资源”选择“默认值”。“请求条件”中单击“添加条件”，设置“条件键”为“modelarts:poolType”，“运算符”为“StringEquals”，“值”为“public”。

     JSON视图的策略内容如下：

     ```
     {
        "Version": "1.1",
        "Statement": [
           {
              "Effect": "Deny",
              "Action": [
                 "modelarts:trainJob:create",
                 "modelarts:notebook:create",
                 "modelarts:service:create"
              ],
              "Condition": {
                 "StringEquals": {
                    "modelarts:poolType": [
                       "public"
                    ]
                 }
              }
           }
        ]
     }
     ```

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤**2**创建的两条自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限查看该用户组内成员创建的所有Notebook实例。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

4. 在用户的委托授权中同步增加此策略，避免在租户面通过委托token突破限制。

   在统一身份认证服务页面的左侧导航中选择委托，找到该用户组在ModelArts上使用的委托名称，单击右侧的“修改”操作，选择“授权记录”页签，单击“授权”，选中上一步创建的自定义策略“不允许用户使用公共资源池”，单击“下一步”，选择允许使用的资源区域，单击“确定”。

 版权所有 © 华为云计算技术有限公司

验证

使用子账号用户登录ModelArts控制台，选择“训练管理 > 训练作业”，单击“创建训练作业”，在创建训练页面，资源池规格只能选择专属资源池。

使用子账号用户登录ModelArts控制台，选择“开发环境 > Notebook”，单击“创建”，在创建Notebook页面，资源池规格只能选择专属资源池。

使用子账号用户登录ModelArts控制台，选择“部署上线 > 在线服务”，单击“部署”，在部署服务页面，资源池规格只能选择专属资源池。

2.3.7 给子用户配置文件夹级的 SFS Turbo 访问权限

场景描述

本文介绍如何配置文件夹级的SFS Turbo访问权限，实现在ModelArts中访问挂载的SFS Turbo时，只允许子用户访问特定的SFS Turbo文件夹内容。

前提条件

- 需要在ModelArts控制台打开严格授权模式，单击“全局配置 > 启用严格模式”。
- 如果打开严格模式前没有为子用户配置过ModelArts权限，开启严格授权模式后可能会导致子用户无法使用ModelArts功能，请参考**快速配置ModelArts访问权限**为子用户配置权限。

操作步骤

步骤1 使用主用户账号登录管理控制台，鼠标放在右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：ma_sfs_turbo。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "<modelarts_action>"
      ],
      "Condition": {
        "StringEquals": {
          "modelarts:sfsId": [
            "<your_ssf_id>"
          ],
          "modelarts:sfsPath": [
            "<sfs_path>"
          ],
          "modelarts:sfsOption": [
            "<sfs_option>"
          ]
        }
      }
    }
```


版权所有 © 华为云计算技术有限公司

```
    ]
}
```

以上代码中的"*<modelarts_action>*"、"*<your_ssf_id>*"、"*<sfs_path>*"、"*<sfs_option>*"，需要根据您的业务需求替换为实际的参数，各参数含义如下。

表 2-22 参数解释

参数	参数解释
Action	表示在何种场景下授予SFS Turbo文件夹访问权限。 ● 创建开发环境实例：modelarts:trainJob:create ● 创建训练作业：modelarts:notebook:create 支持填写多种Action，例如： `"Action": [ "modelarts:trainJob:create", "modelarts:notebook:create" ],`
modelarts:sfsId	SFS Turbo的ID，在SFS Turbo详情页查看。支持填写多个ID，例如： `"modelarts:sfsId": [ "0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41" ],`

modelarts:sfsPath	需要进行权限配置的SFS Turbo文件夹路径。支持填写多个路径，例如：
	`"modelarts:sfsPath": [ "/path1", "/path2/path2-1" ],`
	如果sfsId中填写了多个ID，则sfsPath会应用于所有sfsId。例如以下代码含义为：为"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"的"/path1"和"/path2/path2-1"配置访问权限，同时也为"2a70da1e-ea87-4ee4-ae1e-55df846e7f41"的"/path1"和"/path2/path2-1"配置访问权限。
	`"modelarts:sfsId": [ "0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41" ], "modelarts:sfsPath": [ "/path1", "/path2/path2-1" ],`

版权所有 © 华为云计算技术有限公司

参数	参数解释

modelarts:sfsOption	设置用户对于SFS Turbo文件夹的权限类型，支持填写以下参数：
	● 仅读权限：readonly
	● 读写权限：readwrite
	如果需要在一个自定义策略中添加多个不同的sfsOption，需要“Statement”中新增JSON结构体，例如：
	{
	"Version": "1.1",
	"Statement": [
	{
	"Effect": "Allow",
	"Action": [
	"modelarts:notebook:create"
	],
	"Condition": {
	"StringEquals": {
	"modelarts:sfsId": [
	"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
	],
	"modelarts:sfsPath": [
	"/path1"
	],
	"modelarts:sfsOption": [
	"readonly"
	]
	}
	}
	},
	{
	"Effect": "Allow",
	"Action": [
	"modelarts:notebook:create"
	],
	"Condition": {
	"StringEquals": {
	"modelarts:sfsId": [
	"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
	],
	"modelarts:sfsPath": [
	"/path2"
	],
	"modelarts:sfsOption": [
	"readwrite"
	]
	}
	}
	}
	]
	}

步骤3 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤4 给用户组授权策略。在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子用户配置权限。勾选**步骤2**中创建的“ma_sfs_turbo”策略。单击“下一步”和“确定”。

步骤5 在已有的ModelArts委托权限中，追加IAM ReadOnlyAccess权限。

1. 在ModelArts管理控制台，单击“全局配置”，在对应委托的操作列，单击“查看权限 > 去IAM修改委托权限”。

版权所有 © 华为云计算技术有限公司

2. 在新页面中，单击“授权记录 > 授权”，搜索“IAM ReadOnlyAccess”，勾选后单击“下一步”并单击“确认”。

步骤6 验证权限是否配置成功。

登录子用户账号，在创建训练作业/创建Notebook时，仅能看到配置的SFS Turbo文件夹，则表示权限配置成功。

----结束

2.4 FAQ

2.4.1 使用 ModelArts 时提示“权限不足”，如何解决?

当您使用ModelArts时如果提示权限不足，请您按照如下指导对相关服务和用户进行授权，并对用户权限进行检查操作。

由于ModelArts的使用权限依赖OBS服务的授权，您需要为用户授予OBS的系统权限。

- 如果您需要授予用户关于OBS的所有权限和ModelArts的基础操作权限，请参见**配置基础操作权限**。
- 如果您需要对用户使用OBS和ModelArts的权限进行精细化管理，进行自定义策略配置，请参见**创建ModelArts自定义策略**。

配置基础操作权限

使用ModelArts的基本功能，您需要为用户配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，由于ModelArts依赖OBS权限，您还需要为用户授予“作用范围”为“全局级服务”的“OBS Administrator”策略。

具体操作步骤如下：

步骤1 创建用户组。

登录IAM管理控制台，单击“用户组>创建用户组”。在“创建用户组”界面，输入“用户组名称”单击“确定”。

步骤2 配置用户组权限。

在用户组列表中，单击步骤1新建的用户组右侧的“授权”，在用户组“授权”页面，您需要配置的权限如下：

1. 配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，如下图所示，然后单击“确定”完成授权。

   说明

   区域级项目授权后只在授权区域生效，如果需要所有区域都生效，则所有区域都需要进行授权操作。

2. 配置“作用范围”为“全局级服务”的“OBS Administrator”权限，然后单击“确定”完成授权。

步骤3 **创建用户并加入用户组**。

在IAM控制台创建用户，并将其加入步骤1中创建的用户组。

 版权所有 © 华为云计算技术有限公司

步骤4 用户登录并验证权限。

新创建的用户登录控制台，切换至授权区域，验证权限：

- 在“服务列表”中选择ModelArts，进入ModelArts主界面，选择不同类型的专属资源池，在页面单击“创建”，如果无法进行创建（当前权限仅包含ModelArts CommonOperations），表“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择除ModelArts外（假设当前策略仅包含ModelArts CommonOperations）的任一服务，如果提示权限不足，表示“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理>数据集>创建数据 > 集”，如果可以成功访问对应的OBS路径，表示全局级服务的“OBS Administrator”已生效。

----结束

创建 ModelArts 自定义策略

如果系统预置的ModelArts权限不满足您的授权要求，或者您需要管理用户操作OBS的操作权限，可以创建自定义策略。更多关于创建自定义策略操作和参数说明请参见**创建自定义策略**。

目前华为云支持可视化视图创建自定义策略和JSON视图创建自定义策略，本章节将使用JSON视图方式的策略，以为ModelArts用户授予开发环境的使用权限并且配置ModelArts用户OBS相关的最小化权限项为例，指导您进行自定义策略配置。

说明

如果一个自定义策略中包含多个服务的授权语句，这些服务必须是同一属性，即都是全局级服务或者项目级服务。

由于OBS为全局服务，ModelArts为项目级服务，所以需要创建两条“作用范围”别为“全局级服务”以及“项目级服务”的自定义策略，然后将两条策略同时授予用户。

1. 创建ModelArts相关OBS的最小化权限的自定义策略。

   登录IAM控制台，在“权限管理>权限”页面，单击“创建自定义策略”。参数配置说明如下：

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts依赖的OBS权限自定义策略样例**，如果您需要了解更多关于OBS的系统权限，请参见**OBS权限管理**。

2. 创建ModelArts开发环境的使用权限的自定义策略。参数配置说明如下：

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts开发环境使用权限的自定义策略样例**，ModelArts自定义策略中可以添加的授权项（Action）请参见**《ModelArts API参考》>权限策略和授权项**。
   - 如果您需要对除ModelArts和OBS之外的其它服务授权，IAM支持服务的所有策略请参见**权限策略**。

3. 在IAM控制台**创建用户组并授权**。

   在IAM控制台创建用户组之后，将步骤**1**中创建的自定义策略授权给该用户组。

4. **创建用户并加入用户组**。

版权所有 © 华为云计算技术有限公司

在IAM控制台创建用户，并将其加入3中创建的用户组。

5. 用户登录并验证权限。

   新创建的用户登录控制台，切换至授权区域，验证权限：

   - 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理 > 数据集”，如果无法进行创建（当前仅包含开发环境的使用权限），表示仅为ModelArts用户授予开发环境的使用权限已生效。
   - 在“服务列表”中选择除ModelArts，进入ModelArts主界面，单击“开发环境>Notebook>创建”，如果可以成功访问“存储配置”项对应的OBS路径，表示为用户配置的OBS相关权限已生效。

ModelArts 依赖的 OBS 权限自定义策略样例

如下示例为ModelArts依赖OBS服务的最小化权限项，包含OBS桶和OBS对象的权限。授予示例中的权限您可以通过ModelArts正常访问OBS不受限制。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "obs:bucket:ListAllMybuckets",
                "obs:bucket:HeadBucket",
                "obs:bucket:ListBucket",
                "obs:bucket:GetBucketLocation",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:object:PutObject",
                "obs:object:DeleteObject",
                "obs:object:DeleteObjectVersion",
                "obs:object:ListMultipartUploadParts",
                "obs:object:AbortMultipartUpload",
                "obs:object:GetObjectAcl",
                "obs:object:GetObjectVersionAcl",
                "obs:bucket:PutBucketAcl",
                "obs:object:PutObjectAcl"
            ],
            "Effect": "Allow"
        }
    ]
}
```

ModelArts 开发环境使用权限的自定义策略样例

```
{
    "Version": "1.1",
    "Statement": [

        {
            "Effect": "Allow",
            "Action": [
                "modelarts:notebook:list",
                "modelarts:notebook:create" ,
                "modelarts:notebook:get" ,
                "modelarts:notebook:update" ,
                "modelarts:notebook:delete" ,
                "modelarts:notebook:action" ,
                "modelarts:notebook:access"
            ]
        }
    ]
}
```


版权所有 © 华为云计算技术有限公司

3 开发环境

3.1 基于 SFS 创建、迁移和管理 Conda 虚拟环境

本文介绍了如何将Notebook的Conda环境迁移到SFS磁盘上。这样重启Notebook实例后，Conda环境不会丢失。

步骤如下:

1. **创建新的虚拟环境并保存到SFS目录**
2. **克隆原有的虚拟环境到SFS盘**
3. **重新启动镜像激活SFS盘中的虚拟环境**
4. **保存并共享虚拟环境**

前提条件

创建一个Notebook，“资源类型”选择“专属资源池”，“存储配置”选择“SFS弹性文件服务器”，打开terminal。

创建新的虚拟环境并保存到 SFS 目录

创建新的conda虚拟环境。

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-new-env python=3.7.10 -y
```

查看现有的conda虚拟环境，此时可能出现新创建的虚拟环境的名称为空的情况。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
                      /home/ma-user/work/envs/user_conda/sfs-new-env
```

添加新创建的虚拟环境到conda env。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
```

查看现有的conda虚拟环境，此时新的虚拟环境已经能够正常显示，可以直接通过名称进行虚拟环境的切换。

 版权所有 © 华为云计算技术有限公司

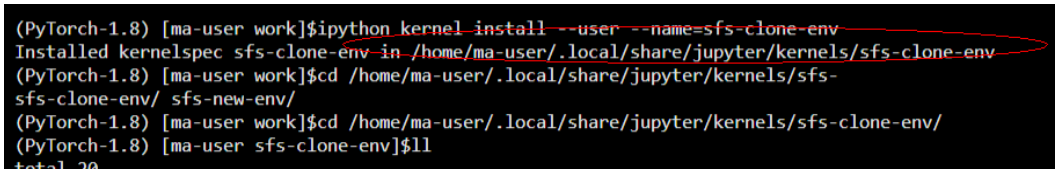
```
# shell
conda env list
conda activate sfs-new-env
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）。

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-new-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-new-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-new-env”为举例，请以用户实际的安装路径为准。

图 3-1 安装路径回显



刷新JupyterLab页面，可以看到新的kernel。

说明

重启Notebook后kernel需要重新注册。

克隆原有的虚拟环境到 SFS 盘

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-clone-env --clone PyTorch-1.8 -
y
Source:      /home/ma-user/anaconda3/envs/PyTorch-1.8
Destination: /home/ma-user/work/envs/user_conda/sfs-clone-env
Packages: 20
Files: 39687
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate /home/ma-user/work/envs/user_conda/sfs-clone-env
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

查看新创建的clone虚拟环境，如果出现新创建的虚拟环境的名称为空的情况，可以参考**添加新创建到虚拟环境到conda env**。

```
# shell
conda env list
# conda environments:
#
```


版权所有 © 华为云计算技术有限公司

```
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10            /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env            /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env           *  /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-clone-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-clone-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-clone-env”为举例，请以用户实际的安装路径为准。

刷新JupyterLab页面，可以看到新的kernel。

重新启动镜像激活 SFS 盘中的虚拟环境

方法一，直接使用完整conda env路径。

```
# shell
conda activate /home/ma-user/work/envs/user_conda/sfs-new-env
```

方法二，先添加虚拟环境到conda env，然后使用名称激活。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
conda activate sfs-new-env
```

方法三，直接使用完成虚拟环境中的python或者pip。

```
# shell
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/pip list
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/python -V
```

保存并共享虚拟环境

将要迁移的虚拟环境打包。

```
# shell
pip install conda-pack
conda pack -n sfs-clone-env -o sfs-clone-env.tar.gz --ignore-editable-packages
Collecting packages...
Packing environment at '/home/ma-user/work/envs/user_conda/sfs-clone-env' to 'sfs-clone-env.tar.gz'
[########################################] | 100% Completed |  3min 33.9s
```

解压到SFS目录。

```
# shell

mkdir /home/ma-user/work/envs/user_conda/sfs-tar-env
tar -zxvf sfs-clone-env.tar.gz -C /home/ma-user/work/envs/user_conda/sfs-tar-env
```

查看现有的conda虚拟环境。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
```


版权所有 © 华为云计算技术有限公司

```
PyTorch-1.8           *  /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10            /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env            /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
sfs-tar-env              /home/ma-user/work/envs/user_conda/sfs-tar-env
test-env                 /home/ma-user/work/envs/user_conda/test-env
```


版权所有 © 华为云计算技术有限公司

4 模型训练

4.1 使用自定义算法构建模型（手写数字识别）

本文为用户提供如何将本地的自定义算法通过简单的代码适配，实现在ModelArts上进行模型训练与部署的全流程指导。

场景描述

本案例用于指导用户使用PyTorch1.8实现手写数字图像识别，示例采用的数据集为MNIST官方数据集。

通过学习本案例，您可以了解如何在ModelArts平台上训练作业、部署推理模型并预测的完整流程。

操作流程

开始使用如下样例前，请务必按**准备工作**指导完成必要操作。

1. **Step1 准备训练数据**：下载MNIST数据集。
2. **Step2 准备训练文件和推理文件**：编写训练与推理代码。
3. **Step3 创建OBS桶并上传文件**：创建OBS桶和文件夹，并将数据集和训练脚本，推理脚本，推理配置文件上传到OBS中。
4. **Step4 创建训练作业**：进行模型训练。
5. **Step5 推理部署**：训练结束后，将生成的模型导入ModelArts用于创建AI应用，并将AI应用部署为在线服务。
6. **Step6 预测结果**：上传一张手写数字图片，发起预测请求获取预测结果。
7. **Step7 清除资源**：运行完成后，停止服务并删除OBS中的数据，避免不必要的扣费。

准备工作

- 已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

版权所有 © 华为云计算技术有限公司

a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“全局配置”，进入“全局配置”页面，单击“添加授权”。

b. 在弹出的“访问授权”窗口中，

授权对象类型：所有用户

委托选择：新增委托

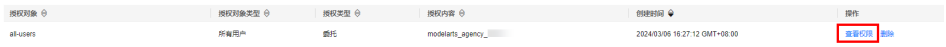
权限配置：普通用户

选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

图 4-1 配置委托访问授权

c. 完成配置后，在ModelArts控制台的全局配置列表，可查看到此账号的委托配置信息。

图 4-2 查看委托配置信息



Step1 准备训练数据

本案例使用的数据是MNIST数据集，您可以从**MNIST官网**下载数据集至本地，以下4个文件均要下载。

图 4-3 MNIST 数据集

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

- “train-images-idx3-ubyte.gz”：训练集的压缩包文件，共包含60000个样本。
- “train-labels-idx1-ubyte.gz”：训练集标签的压缩包文件，共包含60000个样本的类别标签。
- “t10k-images-idx3-ubyte.gz”：验证集的压缩包文件，共包含10000个样本。
- “t10k-labels-idx1-ubyte.gz”：验证集标签的压缩包文件，共包含10000个样本的类别标签。

版权所有 © 华为云计算技术有限公司

说明

如果单击MNIST官网链接后提示输入登录信息，可在浏览器中直接粘贴此链接免登录：http://yann.lecun.com/exdb/mnist/。

出现登录提示是由于浏览器使用了https的方式打开了该链接，使用http的方式打开则无需登录信息。

Step2 准备训练文件和推理文件

针对此案例，ModelArts提供了需使用的训练脚本、推理脚本和推理配置文件。请参考如下文件内容。

说明

粘贴“.py”文件代码时，请直接新建“.py”文件，否则会可能出现“SyntaxError: 'gbk' codec can't decode byte 0xa4 in position 324: illegal multibyte sequence”报错。

在本地电脑中创建训练脚本“train.py”，内容如下：

```
# base on https://github.com/pytorch/examples/blob/main/mnist/main.py

from __future__ import print_function

import os
import gzip
import codecs
import argparse
from typing import IO, Union

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

import shutil


# 定义网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

 版权所有 © 华为云计算技术有限公司

```
# 模型训练，设置模型为训练模式，加载训练数据，计算损失函数，执行梯度下降
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


# 模型验证，设置模型为验证模式，加载验证数据，计算损失函数和准确率
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


# 以下为pytorch mnist
# https://github.com/pytorch/vision/blob/v0.9.0/torchvision/datasets/mnist.py
def get_int(b: bytes) -> int:
    return int(codecs.encode(b, 'hex'), 16)


def open_maybe_compressed_file(path: Union[str, IO]) -> Union[IO, gzip.GzipFile]:
    """Return a file object that possibly decompresses 'path' on the fly.
       Decompression occurs when argument `path` is a string and ends with '.gz' or '.xz'.
    """
    if not isinstance(path, torch._six.string_classes):
        return path
    if path.endswith('.gz'):
        return gzip.open(path, 'rb')
    if path.endswith('.xz'):
        return lzma.open(path, 'rb')
    return open(path, 'rb')


SN3_PASCALVINCENT_TYPEMAP = {
    8: (torch.uint8, np.uint8, np.uint8),
    9: (torch.int8, np.int8, np.int8),
    11: (torch.int16, np.dtype('>i2'), 'i2'),
    12: (torch.int32, np.dtype('>i4'), 'i4'),
    13: (torch.float32, np.dtype('>f4'), 'f4'),
    14: (torch.float64, np.dtype('>f8'), 'f8')
}
```


版权所有 © 华为云计算技术有限公司

```
def read_sn3_pascalvincent_tensor(path: Union[str, IO], strict: bool = True) -> torch.Tensor:
    """Read a SN3 file in "Pascal Vincent" format (Lush file 'libidx/idx-io.lsh').
       Argument may be a filename, compressed filename, or file object.
    """
    # read
    with open_maybe_compressed_file(path) as f:
        data = f.read()
    # parse
    magic = get_int(data[0:4])
    nd = magic % 256
    ty = magic // 256
    assert 1 <= nd <= 3
    assert 8 <= ty <= 14
    m = SN3_PASCALVINCENT_TYPEMAP[ty]
    s = [get_int(data[4 * (i + 1): 4 * (i + 2)]) for i in range(nd)]
    parsed = np.frombuffer(data, dtype=m[1], offset=(4 * (nd + 1)))
    assert parsed.shape[0] == np.prod(s) or not strict
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)


def read_label_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 1)
    return x.long()


def read_image_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 3)
    return x


def extract_archive(from_path, to_path):
    to_path = os.path.join(to_path, os.path.splitext(os.path.basename(from_path))[0])
    with open(to_path, "wb") as out_f, gzip.GzipFile(from_path) as zip_f:
        out_f.write(zip_f.read())
# --- 以上为pytorch mnist
# --- end


# raw mnist 数据处理
def convert_raw_mnist_dataset_to_pytorch_mnist_dataset(data_url):
    """
    raw

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz

    processed

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz
        MNIST/raw
            train-images-idx3-ubyte
            train-labels-idx1-ubyte
            t10k-images-idx3-ubyte
            t10k-labels-idx1-ubyte
        MNIST/processed
            training.pt
```


版权所有 © 华为云计算技术有限公司

```
        test.pt
    """
    resources = [
        "train-images-idx3-ubyte.gz",
        "train-labels-idx1-ubyte.gz",
        "t10k-images-idx3-ubyte.gz",
        "t10k-labels-idx1-ubyte.gz"
    ]

    pytorch_mnist_dataset = os.path.join(data_url, 'MNIST')

    raw_folder = os.path.join(pytorch_mnist_dataset, 'raw')
    processed_folder = os.path.join(pytorch_mnist_dataset, 'processed')

    os.makedirs(raw_folder, exist_ok=True)
    os.makedirs(processed_folder, exist_ok=True)

    print('Processing...')

    for f in resources:
        extract_archive(os.path.join(data_url, f), raw_folder)

    training_set = (
        read_image_file(os.path.join(raw_folder, 'train-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 'train-labels-idx1-ubyte'))
    )
    test_set = (
        read_image_file(os.path.join(raw_folder, 't10k-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 't10k-labels-idx1-ubyte'))
    )
    with open(os.path.join(processed_folder, 'training.pt'), 'wb') as f:
        torch.save(training_set, f)
    with open(os.path.join(processed_folder, 'test.pt'), 'wb') as f:
        torch.save(test_set, f)

    print('Done!')


def main():
    # 定义可以接收的训练作业运行参数
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')

    parser.add_argument('--data_url', type=str, default=False,
                        help='mnist dataset path')
    parser.add_argument('--train_url', type=str, default=False,
                        help='mnist model path')

    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=14, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True,
                        help='For Saving the current Model')
    args = parser.parse_args()
```


版权所有 © 华为云计算技术有限公司

```
    use_cuda = not args.no_cuda and torch.cuda.is_available()

    torch.manual_seed(args.seed)

    # 设置使用 GPU 还是 CPU 来运行算法
    device = torch.device("cuda" if use_cuda else "cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    # 定义数据预处理方法
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

    # 将 raw mnist 数据集转换为 pytorch mnist 数据集
    convert_raw_mnist_dataset_to_pytorch_mnist_dataset(args.data_url)

    # 分别创建训练和验证数据集
    dataset1 = datasets.MNIST(args.data_url, train=True, download=False,
                       transform=transform)
    dataset2 = datasets.MNIST(args.data_url, train=False, download=False,
                       transform=transform)

    # 分别构建训练和验证数据迭代器
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    # 初始化神经网络模型并复制模型到计算设备上
    model = Net().to(device)
    # 定义训练优化器和学习率策略，用于梯度下降计算
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    # 训练神经网络，每一轮进行一次验证
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    # 保存模型与适配 ModelArts 推理模型包规范
    if args.save_model:

        # 在 train_url 训练参数对应的路径内创建 model 目录
        model_path = os.path.join(args.train_url, 'model')
        os.makedirs(model_path, exist_ok = True)

        # 按 ModelArts 推理模型包规范，保存模型到 model 目录内
        torch.save(model.state_dict(), os.path.join(model_path, 'mnist_cnn.pt'))

        # 复制推理代码与配置文件到 model 目录内
        the_path_of_current_file = os.path.dirname(__file__)
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/customize_service.py'),
os.path.join(model_path, 'customize_service.py'))
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/config.json'), os.path.join(model_path,
'config.json'))

if __name__ == '__main__':
    main()
```

在本地电脑中创建推理脚本“customize_service.py”，内容如下：

 版权所有 © 华为云计算技术有限公司

```
import os
import log
import json

import torch.nn.functional as F
import torch.nn as nn
import torch
import torchvision.transforms as transforms

import numpy as np
from PIL import Image

from model_service.pytorch_model_service import PTServingBaseService

logger = log.getLogger(__name__)

# 定义模型预处理
infer_transformation = transforms.Compose([
    transforms.Resize(28),
    transforms.CenterCrop(28),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# 模型推理服务
class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # 调用父类构造方法
        super(PTVisionService, self).__init__(model_name, model_path)

        # 调用自定义函数加载模型
        self.model = Mnist(model_path)

        # 加载标签
        self.label = [0,1,2,3,4,5,6,7,8,9]

    # 接收request数据，并转换为模型可以接受的输入格式
    def _preprocess(self, data):
        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # 灰度处理
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    # 将推理的结果进行后处理，得到预期的输出格式，该结果就是最终的返回值
    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    # 对于输入数据进行前向推理，得到推理结果
    def _inference(self, data):

        result = {}
```

 版权所有 © 华为云计算技术有限公司

```
        for k, v in data.items():
            result[k] = self.model(v)

        return result

# 定义网络
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def Mnist(model_path, **kwargs):
    # 生成网络
    model = Net()

    # 加载模型
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))

    # CPU 或者 GPU 映射
    model.to(device)

    # 声明为推理模式
    model.eval()

    return model
```

在本地电脑中推理配置文件“config.json”，内容如下：

```
{
    "model_algorithm": "image_classification",
    "model_type": "PyTorch",
    "runtime": "pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64"
}
```

Step3 创建 OBS 桶并上传文件

将上一步中的数据和代码文件、推理代码文件与推理配置文件，从本地上传到OBS桶中。在ModelArts上运行训练作业时，需要从OBS桶中读取数据和代码文件。

1. 登录OBS管理控制台，按照如下示例创建OBS桶和文件夹。创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

版权所有 © 华为云计算技术有限公司

```
{OBS桶}                  # OBS对象桶，用户可以自定义名称，例如：test-modelarts-xx
    -{OBS文件夹}          # OBS文件夹，自定义名称，此处举例为pytorch
       - mnist-data      # OBS文件夹，用于存放训练数据集，可以自定义名称，此处举例为mnist-data
       - mnist-code      # OBS文件夹，用于存放训练脚本train.py，可以自定义名称，此处举例为mnist-code
          - infer        # OBS文件夹，用于存放推理脚本customize_service.py和配置文件config.json
       - mnist-output    # OBS文件夹，用于存放训练输出模型，可以自定义名称，此处举例为mnist-output
```

注意

- 创建的OBS桶所在区域和后续使用ModelArts必须在同一个区域Region，否则会导致训练时找不到OBS桶。具体操作可参见**查看OBS桶与ModelArts是否在同一区域**。
- 创建OBS桶时，桶的存储类别请勿选择“归档存储”，归档存储的OBS桶会导致模型训练失败。

2. 上传**Step1 准备训练数据**下载的MNIST数据集压缩包文件到OBS中。上传文件至OBS的操作指导请参见**上传对象**。

注意

- 上传数据到OBS中时，请不要加密，否则会导致训练失败。
- 文件无需解压，直接上传压缩包至OBS中即可。

3. 上传训练脚本“train.py”到“mnist-code”文件夹中。
4. 上传推理脚本“customize_service.py”和推理配置文件“config.json”到“infer”文件中。

Step4 创建训练作业

1. 登录ModelArts管理控制台，选择和OBS桶相同的区域。
2. 在“全局配置”中检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
3. 在左侧导航栏选择“训练管理 > 训练作业”进入训练作业页面，单击“创建训练作业”。
4. 填写创建训练作业相关信息。
   - “创建方式”：选择“自定义算法”。
   - “启动方式”：选择“预置框架”，下拉框中选择PyTorch，pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64。
   - “代码目录”：选择已创建的OBS代码目录路径，例如“/test-modelarts-xx/pytorch/mnist-code/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “启动文件”：选择代码目录下上传的训练脚本“train.py”。
   - “输入”：单击“增加训练输入”，设置训练输入的“参数名称”为“data_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-data/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “输出”：单击“增加训练输出”，设置训练输出的“参数名称”为“train_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-

版权所有 © 华为云计算技术有限公司

xx/pytorch/mnist-output/”（test-modelarts-xx需替换为您的OBS桶名称）。预下载至本地目录选择“不下载”。

- “资源类型”：选择GPU单卡的规格，如“GPU: 1*NVIDIA-V100(16GB) | CPU: 8 核 64GB”。如果有免费GPU规格，可以选择免费规格进行训练。
- 其他参数保持默认即可。

说明

本样例代码为单机单卡场景，选择GPU多卡规格会导致训练失败。

5. 单击“提交”，确认训练作业的参数信息，确认无误后单击“确定”。

   页面自动返回“训练作业”列表页，当训练作业状态变为“已完成”时，即完成了模型训练过程。

说明

本案例的训练作业预计运行十分钟。

6. 单击训练作业名称，进入作业详情界面查看训练作业日志信息，观察日志是否有明显的Error信息，如果有则表示训练失败，请根据日志提示定位原因并解决。
7. 在训练详情页左下方单击训练输出路径，如图4-4所示，跳转到OBS目录，查看是否存在model文件夹，且model文件夹中是否有生成训练模型。如果未生成model文件夹或者训练模型，可能是训练输入数据不完整导致，请检查训练数据上传是否完整，并重新训练。

图 4-4 训练输出路径

Step5 推理部署

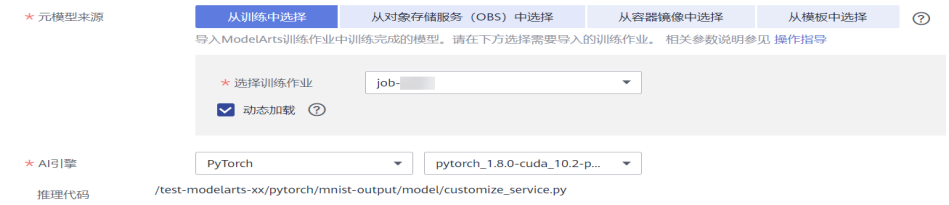
模型训练完成后，可以创建AI应用，将AI应用部署为在线服务。

1. 在ModelArts管理控制台，单击左侧导航栏中的“AI应用管理>AI应用”，进入“我的AI应用”页面，单击“创建”。
2. 在“创建AI应用”页面，填写相关参数，然后单击“立即创建”。

版权所有 © 华为云计算技术有限公司

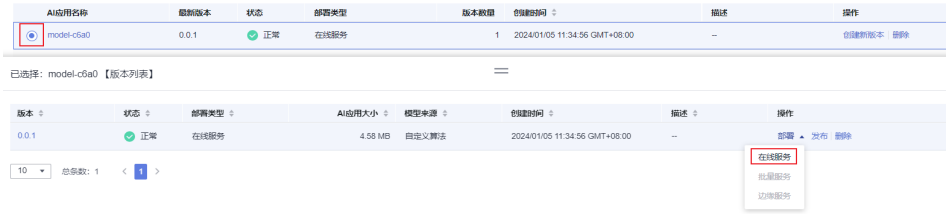
在“元模型来源”中，选择“从训练中选择”页签，选择Step4 创建训练作业中完成的训练作业，勾选“动态加载”。AI引擎的值是系统自动写入的，无需设置。

图 4-5 设置元模型来源



3. 在AI应用列表页面，当AI应用状态变为“正常”时，表示AI应用创建成功。单击AI应用名称左侧的单选按钮，在列表页底部展开“版本列表”，单击操作列“部署>在线服务”，将AI应用部署为在线服务。

图 4-6 部署在线服务



4. 在“部署”页面，参考下图填写参数，然后根据界面提示完成在线服务创建。本案例适用于CPU规格，节点规格需选择CPU。如果有免费CPU规格，可选择免费规格进行部署（每名用户限部署一个免费的在线服务，如果您已经部署了一个免费在线服务，需要先将其删除才能部署新的免费在线服务）。

图 4-7 部署模型

完成服务部署后，返回在线服务页面列表页，等待服务部署完成，当服务状态显示为“运行中”，表示服务已部署成功。

Step6 预测结果

1. 在“在线服务”页面，单击在线服务名称，进入服务详情页面。
2. 单击“预测”页签，请求类型选择“multipart/form-data”，请求参数填写“image”，单击“上传”按钮上传示例图片，然后单击“预测”。

   预测完成后，预测结果显示区域将展示预测结果，根据预测结果内容，可识别出此图片的数字是“2”。

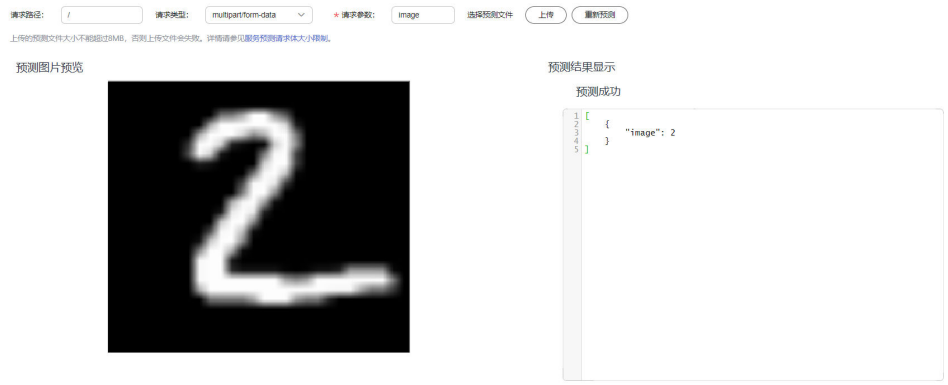
版权所有 © 华为云计算技术有限公司

说明

本案例中使用的MNIST是比较简单的用做demo的数据集，配套算法也是比较简单的用于教学的神经网络算法。这样的数据和算法生成的模型仅适用于教学模式，并不能应对复杂的预测场景。即生成的模型对预测图片有一定范围和要求，预测图片必须和训练集中的图片相似（黑底白字）才可能预测准确。

图 4-8 示例图片

图 4-9 预测结果展示



Step7 清除资源

如果不再需要使用此模型及在线服务，建议清除相关资源，避免产生不必要的费用。

- 在“在线服务”页面，“停止”或“删除”刚创建的在线服务。
- 在“AI应用管理”页面，“删除”刚创建的AI应用。
- 在“训练作业”页面，“删除”运行结束的训练作业。
- 进入OBS，删除本示例使用的OBS桶及文件夹，以及文件夹的文件。

常见问题

- 训练作业一直在等待中（排队）？

  训练作业状态一直在等待中状态表示当前所选的资源池规格资源紧张，作业需要进行排队，请耐心等待。请参考**训练作业一直在等待中（排队）？** 。

- 在ModelArts中选择OBS路径时，找不到已创建的OBS桶?

  请确保创建的桶和ModelArts服务在同一区域，详细操作请参考**查看OBS桶与ModelArts是否在同一个区域**。

4.2 示例：从 0 到 1 制作自定义镜像并用于训练（PyTorch +CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。

版权所有 © 华为云计算技术有限公司

> **说明**
>
> 本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 准备训练脚本并上传至OBS**
4. **Step3 准备镜像主机**
5. **Step4 制作自定义镜像**
6. **Step5 上传镜像至SWR服务**
7. **Step6 在ModelArts上创建训练作业**

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表4-1**所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 4-1 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/pytorch/demo-code/”	用于存储训练脚本文件。

版权所有 © 华为云计算技术有限公司

文件夹名称	用途
“obs://test-modelarts/pytorch/log/”	用于存储训练日志文件。

Step2 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本“pytorch-verification.py”文件，并上传至OBS桶的“obs://test-modelarts/pytorch/demo-code/”文件夹下。

“pytorch-verification.py”文件内容如下：

```
import torch
import torch.nn as nn

x = torch.randn(5, 3)
print(x)

available_dev = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
y = torch.randn(5, 3).to(available_dev)
print(y)
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用Ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤 。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以执行以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker 官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 执行如下命令确认Docker Engine版本。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
   ...
   Engine:
    Version:          18.09.0
   ```


版权所有 © 华为云计算技术有限公司

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。
```
mkdir -p context
```
4. 准备可用的pip源文件pip.conf 。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。
```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

说明

在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi ，也可以查看“pip.conf”文件内容。

5. 下载“torch*.whl ”文件。

在网站“https://download.pytorch.org/whl/torch_stable.html”搜索并下载如下whl文件。

- torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
- torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
- torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

说明

“+”符号的URL编码为“%2B”，在上述网站中搜索目标文件名时，需要将原文件名中的“+”符号替换为“%2B”。

例如“torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl”。

6. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 将上述pip源文件、torch*.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。
```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```
8. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。
```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
```

 版权所有 © 华为云计算技术有限公司

```
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 torch*.whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim和curl 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y vim curl && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见**Docker 官方文档**。

9. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像pytorch:1.8.1-cuda11.1。

```
docker build . -t pytorch:1.8.1-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged pytorch:1.8.1-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。

版权所有 © 华为云计算技术有限公司

2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
   ```
6. 完成镜像上传后，在容器镜像服务控制台的“我的镜像”页面可查看已上传的自定义镜像。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”
   - 镜像地址：**Step5 上传镜像至SWR服务**中创建的镜像。
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 启动命令：“/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 资源池：选择公共资源池
   - 类型：选择GPU或者CPU规格。
   - 永久保存日志：打开
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

版权所有 © 华为云计算技术有限公司

图 4-10 GPU 规格运行日志信息

```
tensor([[-0.4181,  0.8150, -0.2581],
        [-0.6062,  0.5347,  0.1890],
        [ 0.5751,  1.2730, -0.3907],
        [ 0.4812, -0.4064, -0.2753],
        [ 1.0377, -1.1248,  1.2977]])
tensor([[-0.7440, -0.8577, -0.2340],
        [ 0.9569,  0.5516, -1.3350],
        [-1.2878, -0.2791,  0.3486],
        [-1.0997,  0.7627, -0.3188],
        [-1.0865, -1.2626, -0.5900]], device='cuda:0')

```

图 4-11 CPU 规格运行日志信息

```
tensor([[ 0.8945, -0.6946,  0.3807],
        [ 0.6665,  0.3133,  0.8285],
        [-0.5353, -0.1730, -0.5419],
        [ 0.4870,  0.5183,  0.2505],
        [ 0.2679, -0.4996,  0.7919]])
tensor([[ 0.9692,  0.4652,  0.5659],
        [ 2.2032,  1.4157, -0.1755],
        [-0.6296,  0.5466,  0.6994],
        [ 0.2353, -0.0089, -1.9546],
        [ 0.9319,  1.1781, -0.4587]])

```

4.3 示例：从 0 到 1 制作自定义镜像并用于训练（MPI+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

版权所有 © 华为云计算技术有限公司

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 准备脚本文件并上传至OBS中**
4. **Step3 准备镜像主机**
5. **Step4 制作自定义镜像**
6. **Step5 上传镜像至SWR服务**
7. **Step6 在ModelArts上创建训练作业**

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表4-2**所示，示例中的桶名称“test-modelarts” 和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 4-2 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/mpi/demo-code/”	用于存储MPI启动脚本与训练脚本文件。
“obs://test-modelarts/mpi/log/”	用于存储训练日志文件。

Step2 准备脚本文件并上传至 OBS 中

准备本案例所需的MPI启动脚本run_mpi.sh文件和训练脚本mpi-verification.py文件，并上传至OBS桶的“obs://test-modelarts/mpi/demo-code/”文件夹下。

版权所有 © 华为云计算技术有限公司

- MPI启动脚本run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")
```


版权所有 © 华为云计算技术有限公司

```
    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-
p ${MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG -x NCCL_SOCKET_IFNAME -x NCCL_IB_HCA -x NCCL_IB_TIMEOUT -x
NCCL_IB_GID_INDEX -x NCCL_IB_TC \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x PATH -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

说明

“run_mpi.sh”脚本需要以LF作为换行符。使用CRLF作为换行符会导致训练作业运行失败，日志中会打印“$'\r': command not found”的错误信息。

- 训练脚本mpi-verification.py文件内容如下：

```
import os
import socket

if __name__ == '__main__':
    print(socket.gethostname())

    # https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
    print('OMPI_COMM_WORLD_SIZE: ' + os.environ['OMPI_COMM_WORLD_SIZE'])
    print('OMPI_COMM_WORLD_RANK: ' + os.environ['OMPI_COMM_WORLD_RANK'])
    print('OMPI_COMM_WORLD_LOCAL_RANK: ' + os.environ['OMPI_COMM_WORLD_LOCAL_RANK'])
```

 版权所有 © 华为云计算技术有限公司

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker 官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
    Engine:
     Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 下载Miniconda3安装文件。

   使用地址 https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应 python 3.7.13）。

5. 下载openmpi 3.0.0安装文件。

   使用地址https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz，下载 horovod v0.22.1已经编译好的openmpi 3.0.0文件。

6. 将上述Miniconda3安装文件、openmpi 3.0.0文件放置在context文件夹内，context文件夹内容如下。

   ```
   context
   ├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
   └── openmpi-3.0.0-bin.tar.gz
   ```

7. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

   ```
   # 容器镜像构建主机需要连通公网

   # 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
   ```


版权所有 © 华为云计算技术有限公司

```
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 复制 Miniconda3 (python 3.7.13) 安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim / curl / net-tools / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd，使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
```


版权所有 © 华为云计算技术有限公司

```
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

关于Dockerfile文件编写的更多指导内容参见 **Docker 官方文档**。

8. 确认已创建完成Dockerfile 文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

9. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像mpi:3.0.0-cuda11.1。

```
docker build . -t mpi:3.0.0-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
naming to docker.io/library/mpi:3.0.0-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。

   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```

   b. 使用docker push命令上传镜像。

   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```

6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

   “swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在ModelArts管理控制台，左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”

 版权所有 © 华为云计算技术有限公司

- 镜像地址：“swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”
- 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/mpi/demo-code/”
- 启动命令：bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/mpi-verification.py
- 环境变量：添加“MY_SSHD_PORT = 38888”
- 资源池：选择公共资源池
- 类型：选择GPU规格
- 计算节点个数：选择“1”或“2”
- 永久保存日志：打开
- 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mpi/log/”

4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如图4-12所示。

图 4-12 1 个计算节点 GPU 规格 worker-0 运行日志信息

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      60/sshd
tcp6       0      0 :::38888                :::*                    LISTEN      60/sshd
172.16.0.122 slots=1
modelarts-job-8cf8a682-21cb-4d73-9bb3-789cecdc458b-worker-0
OMPI_COMM_WORLD_SIZE: 1
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
```

计算节点个数选择为2，训练作业也可以运行。日志信息如图4-13和图4-14所示。

图 4-13 2 个计算节点 worker-0 运行日志信息

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      61/sshd
tcp6       0      0 :::38888                :::*                    LISTEN      61/sshd
172.16.0.39 slots=1
172.16.0.123 slots=1
Warning: Permanently added '[172.16.0.123]:38888' (RSA) to the list of known hosts.
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-0
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-1
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 1
OMPI_COMM_WORLD_LOCAL_RANK: 0
```


版权所有 © 华为云计算技术有限公司

图 4-14 2 个计算节点 worker-1 运行日志信息

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 1
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      62/sshd
tcp6       0      0 :::38888                :::*                    LISTEN      62/sshd
/home/ma-user/modelarts/user-job-dir/[redacted]e/run_mpi.sh: line 109:    66 Terminated              sleep 365d
```

4.4 示例：从 0 到 1 制作自定义镜像并用于训练（Horovod-PyTorch+GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Horovod 0.22.1 + PyTorch 1.8.1，训练使用的资源是GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- pytorch-1.8.1
- horovod-0.22.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 准备训练脚本并上传至OBS**
4. **Step3 准备镜像主机**
5. **Step4 制作自定义镜像**
6. **Step5 上传镜像至SWR服务**
7. **Step6 在ModelArts上创建训练作业**

前提条件

已注册华为云账号，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

版权所有 © 华为云计算技术有限公司

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表4-3**所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 4-3 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/pytorch/demo-code/”	用于存储训练脚本文件。
“obs://test-modelarts/pytorch/log/”	用于存储训练日志文件。

Step2 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本pytorch_synthetic_benchmark.py和run_mpi.sh文件，并上传至OBS桶的“obs://test-modelarts/horovod/demo-code/”文件夹下。

pytorch_synthetic_benchmark.py文件内容如下：

```
import argparse
import torch.backends.cudnn as cudnn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data.distributed
from torchvision import models
import horovod.torch as hvd
import timeit
import numpy as np

# Benchmark settings
parser = argparse.ArgumentParser(description='PyTorch Synthetic Benchmark',
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--fp16-allreduce', action='store_true', default=False,
                    help='use fp16 compression during allreduce')

parser.add_argument('--model', type=str, default='resnet50',
                    help='model to benchmark')
parser.add_argument('--batch-size', type=int, default=32,
                    help='input batch size')

parser.add_argument('--num-warmup-batches', type=int, default=10,
                    help='number of warm-up batches that don\'t count towards benchmark')
parser.add_argument('--num-batches-per-iter', type=int, default=10,
                    help='number of batches per benchmark iteration')
parser.add_argument('--num-iters', type=int, default=10,
                    help='number of benchmark iterations')

parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')

parser.add_argument('--use-adasum', action='store_true', default=False,
                    help='use adasum algorithm to do reduction')

args = parser.parse_args()
```


版权所有 © 华为云计算技术有限公司

```
args.cuda = not args.no_cuda and torch.cuda.is_available()

hvd.init()

if args.cuda:
    # Horovod: pin GPU to local rank.
    torch.cuda.set_device(hvd.local_rank())

cudnn.benchmark = True

# Set up standard model.
model = getattr(models, args.model)()

# By default, Adasum doesn't need scaling up learning rate.
lr_scaler = hvd.size() if not args.use_adasum else 1

if args.cuda:
    # Move model to GPU.
    model.cuda()
    # If using GPU Adasum allreduce, scale learning rate by local_size.
    if args.use_adasum and hvd.nccl_built():
        lr_scaler = hvd.local_size()

optimizer = optim.SGD(model.parameters(), lr=0.01 * lr_scaler)

# Horovod: (optional) compression algorithm.
compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none

# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(optimizer,
                                     named_parameters=model.named_parameters(),
                                     compression=compression,
                                     op=hvd.Adasum if args.use_adasum else hvd.Average)

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)

# Set up fixed fake data
data = torch.randn(args.batch_size, 3, 224, 224)
target = torch.LongTensor(args.batch_size).random_() % 1000
if args.cuda:
    data, target = data.cuda(), target.cuda()


def benchmark_step():
    optimizer.zero_grad()
    output = model(data)
    loss = F.cross_entropy(output, target)
    loss.backward()
    optimizer.step()


def log(s, nl=True):
    if hvd.rank() != 0:
        return
    print(s, end='\n' if nl else '')


log('Model: %s' % args.model)
log('Batch size: %d' % args.batch_size)
device = 'GPU' if args.cuda else 'CPU'
log('Number of %ss: %d' % (device, hvd.size()))

# Warm-up
log('Running warmup...')
timeit.timeit(benchmark_step, number=args.num_warmup_batches)

# Benchmark
```


版权所有 © 华为云计算技术有限公司

```
log('Running benchmark...')
img_secs = []
for x in range(args.num_iters):
    time = timeit.timeit(benchmark_step, number=args.num_batches_per_iter)
    img_sec = args.batch_size * args.num_batches_per_iter / time
    log('Iter #%d: %.1f img/sec per %s' % (x, img_sec, device))
    img_secs.append(img_sec)

# Results
img_sec_mean = np.mean(img_secs)
img_sec_conf = 1.96 * np.std(img_secs)
log('Img/sec per %s: %.1f +-%.1f' % (device, img_sec_mean, img_sec_conf))
log('Total img/sec on %d %s(s): %.1f +-%.1f' %
    (hvd.size(), device, hvd.size() * img_sec_mean, hvd.size() * img_sec_conf))
```

run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
```


版权所有 © 华为云计算技术有限公司

```
        fi
        sleep 1
    done

    echo "[run_mpi] the sshd of ip ${ip} is up"

    echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
  done

  printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

  echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

  np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

  echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p $
{MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

  # execute mpirun at worker-0
  # mpirun
  mpirun \
    -np ${np} \
    -hostfile ${MY_HOME}/hostfile \
    -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
    -tune ${MY_MPI_TUNE_FILE} \
    -bind-to none -map-by slot \
    -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
    -x HOROVOD_MPI_THREADS_DISABLE=1 \
    -x LD_LIBRARY_PATH \
    -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
    "$@"

  RET_CODE=$?

  if [ $RET_CODE -ne 0 ]; then
    echo "[run_mpi] exec command failed, exited with $RET_CODE"
  else
    echo "[run_mpi] exec command successfully, exited with $RET_CODE"
  fi

  # stop 1...N worker by killing the sleep proc
  sed -i '1d' ${MY_HOME}/hostfile
  if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
    echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

    sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

    mpirun \
    --hostfile ${MY_HOME}/hostfile \
    --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
    --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
    -x PATH -x LD_LIBRARY_PATH \
    pkill sleep \
    > /dev/null 2>&1
  fi

  echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
  echo "[run_mpi] the training log is in worker-0"
  sleep 365d
  echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi
```


版权所有 © 华为云计算技术有限公司

```
exit $RET_CODE
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- pytorch-1.8.1
- horovod-0.22.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
    Engine:
     Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

   说明

   在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi，也可以查看pip.conf文件内容。

5. 下载horovod源码文件。

   进入网站https://pypi.org/project/horovod/0.22.1/#files，下载horovod-0.22.1.tar.gz文件。

版权所有 © 华为云计算技术有限公司

6. 下载torch*.whl文件。

   在网站https://download.pytorch.org/whl/torch_stable.html搜索并下载如下whl文件。

   - torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
   - torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
   - torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

   说明

   “+”符号的URL编码为“%2B”；在上述网站中搜索目标文件名时，需要将原文件名中的“+”符号替换为“%2B”。

   例如“torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl”。

7. 下载Miniconda3安装文件。

   使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

8. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-devel-ubuntu18.04 AS builder

# 安装 cmake 工具（使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y build-essential cmake g++-7 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp
COPY openmpi-3.0.0-bin.tar.gz /tmp
COPY horovod-0.22.1.tar.gz /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
RUN cd /usr/local && \
```


版权所有 © 华为云计算技术有限公司

```
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# 编译 Horovod with PyTorch 所需的环境变量
ENV HOROVOD_NCCL_INCLUDE=/usr/include \
    HOROVOD_NCCL_LIB=/usr/lib/x86_64-linux-gnu \
    HOROVOD_MPICXX_SHOW="/usr/local/openmpi/bin/mpicxx -show" \
    HOROVOD_GPU_OPERATIONS=NCCL \
    HOROVOD_WITH_PYTORCH=1

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 torch*.whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 编译安装
horovod-0.22.1.tar.gz
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/horovod-0.22.1.tar.gz

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# 安装 vim / curl / net-tools / mlnx ofed / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping libfile-find-rule-perl-perl \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    # mlnx ofed
    apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1
libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-
dev flex chrpath libltdl-dev && \
    cd /tmp && \
    tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
    MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --
without-fw-update -q && \
    cd - && \
    rm -rf /tmp/* && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3
```


版权所有 © 华为云计算技术有限公司

```
# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd，使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    PYTHONUNBUFFERED=1
```

关于Dockerfile文件编写的更多指导内容参见**Docker 官方文档**。

9. 下载MLNX_OFED安装包。

   进入**地址**，单击“Download”，在“Current Versions”和“Archive Versions”中选择合适的安装包下载。本文示例中选择“Archive Versions”，“Version”选择“5.4-3.5.8.0-LTS”，“OS Distribution”选择“Ubuntu”，“OS Distribution Version”选择“Ubuntu 18.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

10. 下载openmpi-3.0.0-bin.tar.gz。

    使用地址https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz，下载openmpi-3.0.0-bin.tar.gz文件。

11. 将上述pip源文件、torch*.whl文件、Miniconda3安装文件等放置在context文件夹内，context文件夹内容如下。

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── horovod-0.22.1.tar.gz
├── openmpi-3.0.0-bin.tar.gz
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

12. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1。

```
docker build . -t horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```

    构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。

版权所有 © 华为云计算技术有限公司

2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1 swr.{region-id}.{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.{region-id}.{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
   ```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“下一步”。
   - 创建方式：选择“自定义算法”
   - 镜像来源：选择“自定义”
   - 镜像地址：**Step5 上传镜像至SWR服务**中创建的镜像。
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 启动命令：“bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/pytorch_synthetic_benchmark.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 环境变量：单击“增加环境变量”，增加环境变量：MY_SSHD_PORT=38888。
   - 资源池：选择公共资源池。
   - 资源类型：选择GPU规格。
   - 计算节点个数：1个或者2个。
   - 永久保存日志：打开。
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

 版权所有 © 华为云计算技术有限公司

训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 4-15 GPU 规格运行日志信息（1 个计算节点）

```
Iter #0: 342.4 img/sec per GPU
Iter #1: 342.7 img/sec per GPU
Iter #2: 342.8 img/sec per GPU
Iter #3: 342.5 img/sec per GPU
Iter #4: 342.9 img/sec per GPU
Iter #5: 342.8 img/sec per GPU
Iter #6: 342.9 img/sec per GPU
Iter #7: 343.0 img/sec per GPU
Iter #8: 342.6 img/sec per GPU
Iter #9: 342.7 img/sec per GPU
Img/sec per GPU: 342.7 +-0.3
Total img/sec on 1 GPU(s): 342.7 +-0.3
```

图 4-16 GPU 规格运行日志信息（2 个计算节点）

```
Iter #0: 115.1 img/sec per GPU
Iter #1: 123.5 img/sec per GPU
Iter #2: 115.7 img/sec per GPU
Iter #3: 117.4 img/sec per GPU
Iter #4: 120.6 img/sec per GPU
Iter #5: 126.9 img/sec per GPU
Iter #6: 119.4 img/sec per GPU
Iter #7: 118.4 img/sec per GPU
Iter #8: 122.0 img/sec per GPU
Iter #9: 118.2 img/sec per GPU
Img/sec per GPU: 119.7 +-6.8
Total img/sec on 2 GPU(s): 239.4 +-13.5
```

4.5 示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore+GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MindSpore，训练使用的资源是GPU。

说明

本实践教程仅适用于新版训练作业。

版权所有 © 华为云计算技术有限公司

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

- **前提条件**
- **Step1 创建OBS桶和文件夹**
- **Step2 创建数据集并上传至OBS**
- **Step3 准备训练脚本并上传至OBS**
- **Step4 准备镜像主机**
- **Step5 制作自定义镜像**
- **Step6 上传镜像至SWR服务**
- **Step7 在ModelArts上创建训练作业**

前提条件

已注册华为云账号，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表4-4**所示，示例中的桶名称“test-modelarts” 和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 4-4 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/mindspore-gpu/resnet/”	用于存储训练脚本文件。

版权所有 © 华为云计算技术有限公司

文件夹名称	用途
“obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/”	用于存储数据集文件。
“obs://test-modelarts/mindspore-gpu/output/”	用于存储训练输出文件。
“obs://test-modelarts/mindspore-gpu/log/”	用于存储训练日志文件。

Step2 创建数据集并上传至 OBS

进入网站http://www.cs.toronto.edu/~kriz/cifar.html，下载“CIFAR-10 binary version (suitable for C programs)”，解压后将数据上传至OBS桶的“obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/”文件夹下。

Step3 准备训练脚本并上传至 OBS

准备本案例所需的resnet文件和脚本run_mpi.sh，并上传至OBS桶的“obs://test-modelarts/mindspore-gpu/resnet/”文件夹下。

resnet文件下载地址：**https://gitee.com/mindspore/models/tree/r1.8/official/cv/resnet**

run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}
```


版权所有 © 华为云计算技术有限公司

```
if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p $
{MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"
```


版权所有 © 华为云计算技术有限公司

```
        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

“obs://test-modelarts/mindspore-gpu/resnet/”文件夹下包含resnet文件和run_mpi.sh。

Step4 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step5 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker 官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
    Engine:
     Version:          18.09.0
   ```


版权所有 © 华为云计算技术有限公司

说明

推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

```
mkdir -p context
```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

说明

在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi，也可以查看pip.conf文件内容。

5. 下载mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl文件。

使用网站**https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.8.1/MindSpore/gpu/x86_64/cuda-11.1/mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl**，mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl文件。

6. 下载Miniconda3安装文件。

使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 编写容器镜像Dockerfile文件。

在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-devel-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 mindspore whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl \
    easydict PyYAML

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-cudnn8-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp
```


版权所有 © 华为云计算技术有限公司

```
# 安装 vim / curl / net-tools / mlnx ofed / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping libfile-find-rule-perl-perl \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    # mlnx ofed
    apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1
libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-
dev flex chrpath libltdl-dev && \
    cd /tmp && \
    tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
    MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --
without-fw-update -q && \
    cd - && \
    rm -rf /tmp/* && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd，使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
```


版权所有 © 华为云计算技术有限公司

```
    LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
    PYTHONUNBUFFERED=1
```

关于Dockerfile文件编写的更多指导内容参见**Docker 官方文档**。

8. 下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

   进入地址https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/，单击“Download”，“Version”选择“5.4-3.5.8.0-LTS”，“OSDistributionVersion”选择“Ubuntu 18.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

9. 下载openmpi-3.0.0-bin.tar.gz。

   使用地址https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz，下载openmpi-3.0.0-bin.tar.gz文件。

10. 将上述Dockerfile文件、Miniconda3安装文件等放置在context文件夹内，context文件夹内容如下。

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl
├── openmpi-3.0.0-bin.tar.gz
└── pip.conf
```

11. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像mindspore:1.8.1-ofed-cuda11.1。

```
docker build . -t mindspore:1.8.1-ofed-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged mindspore:1.8.1-ofed-cuda11.1
```

Step6 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker tag mindspore:1.8.1-ofed-cuda11.1 swr.{region-id}.{domain}/deep-learning/mindspore:1.8.1-ofed-cuda11.1
```

   b. 使用docker push命令上传镜像。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker push swr.{region-id}.{domain}/deep-learning/mindspore:1.8.1-ofed-cuda11.1
```

6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

Step7 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。

版权所有 © 华为云计算技术有限公司

3. 在“创建训练作业”页面，填写相关参数信息，然后单击“下一步”。
   - 创建方式：选择“自定义算法”。
   - 镜像来源：选择“自定义”。
   - 镜像地址：**Step6 上传镜像至SWR服务**中创建的镜像。。
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/mindspore-gpu/resnet/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/resnet”目录中，“resnet”为OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 启动命令：“bash ${MA_JOB_DIR}/resnet/run_mpi.sh python ${MA_JOB_DIR}/resnet/train.py”，此处的“resnet”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 训练输入：单击“增加训练输入”，参数名称设置为“data_path”，选择OBS中存放数据集的目录，例如“obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/”，获取方式设置为“超参”。
   - 训练输出：单击“增加训练输出”，参数名称设置为“output_path”，选择OBS存放训练输出的路径，例如：“obs://test-modelarts/mindspore-gpu/output/”，获取方式设置为“超参”，预下载至本地目录设置为“不下载”。
   - 超参：单击“增加超参”，增加如下四个超参：
     - run_distribute=True
     - device_num=1（根据规格中的GPU卡数设置）
     - device_target=GPU
     - epoch_size=2
   - 环境变量：单击“增加环境变量”，增加环境变量：MY_SSHD_PORT=38888。
   - 资源池：选择公共资源池。
   - 资源类型：选择GPU规格。
   - 计算节点个数：1个或者2个。
   - 永久保存日志：打开。
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mindspore-gpu/log/”。
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

图 4-17 GPU 规格运行日志信息（1 个计算节点）

```
epoch: 1 step: 1875, loss is 1.4800076
Train epoch time: 369422.027 ms, per step time: 197.025 ms
epoch: 2 step: 1875, loss is 1.0306032
Train epoch time: 66996.087 ms, per step time: 35.731 ms
```


版权所有 © 华为云计算技术有限公司

图 4-18 GPU 规格运行日志信息（2 个计算节点）

```
187 epoch: 1 step: 937, loss is 1.8482083
188 epoch: 1 step: 937, loss is 1.342748
189 Train epoch time: 488492.170 ms, per step time: 521.336 ms
190 Train epoch time: 488490.528 ms, per step time: 521.335 ms
191 epoch: 2 step: 937, loss is 0.9150252
192 Train epoch time: 180200.654 ms, per step time: 192.317 ms
193 epoch: 2 step: 937, loss is 0.9933052
194 Train epoch time: 180199.969 ms, per step time: 192.316 ms
195 172.16.0.45 slots=1
```

4.6 示例：从 0 到 1 制作自定义镜像并用于训练（Tensorflow+GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Tensorflow，训练使用的资源是GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.2
- python-3.7.13
- mlnx ofed-5.4
- tensorflow gpu-2.10.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 创建数据集并上传至OBS**
4. **Step3 准备训练脚本并上传至OBS**
5. **Step4 准备镜像主机**
6. **Step5 制作自定义镜像**
7. **Step6 上传镜像至SWR服务**
8. **Step7 在ModelArts上创建训练作业**

版权所有 © 华为云计算技术有限公司

前提条件

已注册华为云账号，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表4-5**所示，示例中的桶名称“test-modelarts” 和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 4-5 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/tensorflow/code/”	用于存储训练脚本文件。
“obs://test-modelarts/tensorflow/data/”	用于存储数据集文件。
“obs://test-modelarts/tensorflow/log/”	用于存储训练日志文件。

Step2 创建数据集并上传至 OBS

使用网站**https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz**，下载“mnist.npz”文件并上传至OBS桶的“obs://test-modelarts/tensorflow/data/”文件夹下。

Step3 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本mnist.py，并上传至OBS桶的“obs://test-modelarts/tensorflow/code/”文件夹下。

mnist.py文件内容如下：

```
import argparse
import tensorflow as tf

parser = argparse.ArgumentParser(description='TensorFlow quick start')
parser.add_argument('--data_url', type=str, default="./Data", help='path where the dataset is saved')
args = parser.parse_args()

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data(args.data_url)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
```


版权所有 © 华为云计算技术有限公司

```
])

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
```

Step4 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step5 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker 官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
    Engine:
     Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 准备可用的pip源文件pip.conf。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

 版权所有 © 华为云计算技术有限公司

说明

在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi，也可以查看pip.conf文件内容。

5. 下载tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl文件。

   使用网站https://pypi.org/project/tensorflow-gpu/2.10.0/#files，下载tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl文件。

6. 下载Miniconda3安装文件。

   使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 tensorflow whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl

RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir keras==2.10.0

# 构建最终容器镜像
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# 安装 vim / curl / net-tools / mlnx ofed（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping && \
    # mlnx ofed
    apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1
libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-
```


版权所有 © 华为云计算技术有限公司

```
dev flex chrpath libltdl-dev && \
    cd /tmp && \
    tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
    MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --
without-fw-update -q && \
    cd - && \
    rm -rf /tmp/* && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
    PYTHONUNBUFFERED=1
```

关于Dockerfile文件编写的更多指导内容参见**Docker 官方文档**。

8. 下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

   进入**地址**，单击“Download”，“Version”选择“5.4-3.5.8.0-LTS”，“OSDistributionVersion”选择“Ubuntu 18.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz。

9. 将上述Dockerfile文件、 Miniconda3 安装文件等放置在context文件夹内，context文件夹内容如下。

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
└── tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像tensorflow:2.10.0-ofed-cuda11.2。

```
docker build . -t tensorflow:2.10.0-ofed-cuda11.2
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged tensorflow:2.10.0-ofed-cuda11.2
```

Step6 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。

   a. 使用docker tag命令给上传镜像打标签。

 版权所有 © 华为云计算技术有限公司

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/
tensorflow:2.10.0-ofed-cuda11.2
```

b. 使用docker push命令上传镜像。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker push swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
```

6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。

Step7 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“下一步”。
   - 创建方式：选择“自定义算法”。
   - 镜像来源：选择“自定义”。
   - 镜像地址：**Step5 制作自定义镜像**中创建的镜像。。
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/tensorflow/code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/code”目录中，“code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 启动命令：“python ${MA_JOB_DIR}/code/mnist.py”，此处的“code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 训练输入：单击“增加训练输入”，参数名称设置为“data_path”，选择OBS中存放“mnist.npz”的目录，例如“obs://test-modelarts/tensorflow/data/mnist.npz”，获取方式设置为“超参”。
   - 资源池：选择公共资源池。
   - 资源类型：选择GPU规格。
   - 计算节点个数：1个。
   - 永久保存日志：打开。
   - 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mindspore-gpu/log/”。
4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

 版权所有 © 华为云计算技术有限公司

图 4-19 GPU 规格运行日志信息

```
    0.9767
323 1503/1875 [=========================>......] - ETA: 0s - loss: 0.0741 - accuracy:
    0.9769
324 1533/1875 [=========================>......] - ETA: 0s - loss: 0.0743 - accuracy:
    0.9769
325 1564/1875 [=========================>.....] - ETA: 0s - loss: 0.0746 - accuracy:
    0.9768
326 1595/1875 [=========================>.....] - ETA: 0s - loss: 0.0741 - accuracy:
    0.9770
327 1624/1875 [=========================>.....] - ETA: 0s - loss: 0.0742 - accuracy:
    0.9770
328 1654/1875 [==========================>....] - ETA: 0s - loss: 0.0745 - accuracy:
    0.9770
329 1685/1875 [==========================>....] - ETA: 0s - loss: 0.0747 - accuracy:
    0.9768
330 1716/1875 [==========================>...] - ETA: 0s - loss: 0.0752 - accuracy:
    0.9767
331 1747/1875 [==========================>...] - ETA: 0s - loss: 0.0755 - accuracy:
    0.9767
332 1778/1875 [===========================>..] - ETA: 0s - loss: 0.0753 - accuracy:
    0.9767
333 1809/1875 [===========================>..] - ETA: 0s - loss: 0.0751 - accuracy:
    0.9768
334 1841/1875 [============================>.] - ETA: 0s - loss: 0.0753 - accuracy:
    0.9767
335 1872/1875 [============================>.] - ETA: 0s - loss: 0.0753 - accuracy:
    0.9767
336 1875/1875 [==============================] - 3s 2ms/step - loss: 0.0752 - accuracy: 0.9767
```


版权所有 © 华为云计算技术有限公司

5 推理部署

5.1 从 0-1 制作自定义镜像并创建 AI 应用

针对ModelArts目前不支持的AI引擎，您可以针对该引擎构建自定义镜像，并将镜像导入ModelArts，创建为AI应用。本文详细介绍如何使用自定义镜像完成AI应用的创建，并部署成在线服务。

操作流程如下：

1. **本地构建镜像**：在本地制作自定义镜像包，镜像包规范可参考**创建AI应用的自定义镜像规范**。
2. **本地验证镜像并上传镜像至SWR服务**：验证自定义镜像的API接口功能，无误后将自定义镜像上传至SWR服务。
3. **将自定义镜像创建为AI应用**：将上传至SWR服务的镜像导入ModelArts的AI应用管理。
4. **将AI应用部署为在线服务**：将导入的模型部署上线。

本地构建镜像

以linux x86_x64架构的主机为例，您可以购买相同规格的ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录弹性云服务器**。镜像选择公共镜像，推荐使用ubuntu18.04的镜像。

图 5-1 创建 ECS 服务器-选择 X86 架构的公共镜像

1. 登录主机后，安装Docker，可参考**Docker官方文档**。也可执行以下命令安装docker。

版权所有 © 华为云计算技术有限公司

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```

2. 获取基础镜像。本示例以Ubuntu18.04为例。

```
docker pull ubuntu:18.04
```

3. 新建文件夹“self-define-images”，在该文件夹下编写自定义镜像的“Dockerfile”文件和应用服务代码“test_app.py”。本样例代码中，应用服务代码采用了flask框架。

   文件结构如下所示

```
self-define-images/
   --Dockerfile
   --test_app.py
```

   - “Dockerfile”

```
From ubuntu:18.04
# 配置华为云的源，安装 python、python3-pip 和 Flask
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
  sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
  apt-get update && \
  apt-get install -y python3 python3-pip && \
  pip3 install  --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/
repository/pypi/simple  Flask

# 复制应用服务代码进镜像里面
COPY test_app.py /opt/test_app.py

# 指定镜像的启动命令
CMD python3  /opt/test_app.py
```

   - “test_app.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)
```

4. 进入“self-define-images”文件夹，执行以下命令构建自定义镜像“test:v1”。

```
docker build -t test:v1 .
```

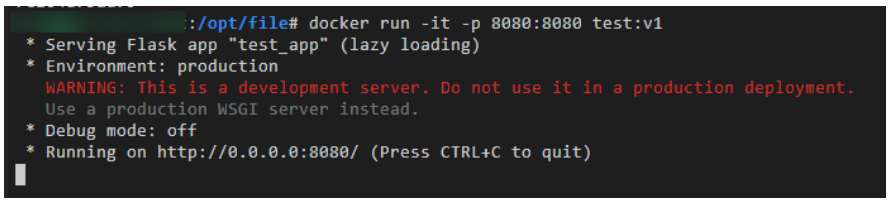
5. 您可以使用“docker images”查看您构建的自定义镜像。

 版权所有 © 华为云计算技术有限公司

本地验证镜像并上传镜像至 SWR 服务

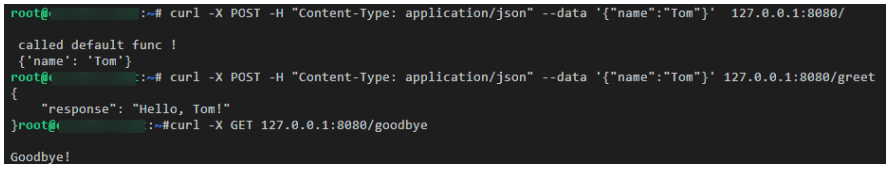
1. 在本地环境执行以下命令启动自定义镜像
```
docker run -it -p 8080:8080 test:v1
```

图 5-2 启动自定义镜像



2. 另开一个终端，执行以下命令验证自定义镜像的三个API接口功能。
```
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
```
如果验证自定义镜像功能成功，结果如下图所示。

图 5-3 校验接口



3. 上传自定义镜像至SWR服务。上传镜像的详细操作可参考**如何登录并上传镜像到SWR**。
4. 完成自定义镜像上传后，您可以在“容器镜像服务>我的镜像>自有镜像”列表中看到已上传镜像。

将自定义镜像创建为 AI 应用

参考**从容器镜像中选择元模型**导入元模型，您需要特别关注以下参数：

- 元模型来源：选择“从容器镜像中选择”
  - 容器镜像所在的路径：选择已制作好的自有镜像

图 5-4 选择已制作好的自有镜像

  - 容器调用接口：指定模型启动的协议和端口号。请确保协议和端口号与自定义镜像中提供的协议和端口号保持一致。
  - 镜像复制：选填，选择是否将容器镜像中的模型镜像复制到ModelArts中。
  - 健康检查：选填，用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致AI应用创建失败。
- apis定义：选填，用于编辑自定义镜像的apis定义。模型apis定义需要遵循ModelArts的填写规范，参见**模型配置文件编写说明**。

版权所有 © 华为云计算技术有限公司

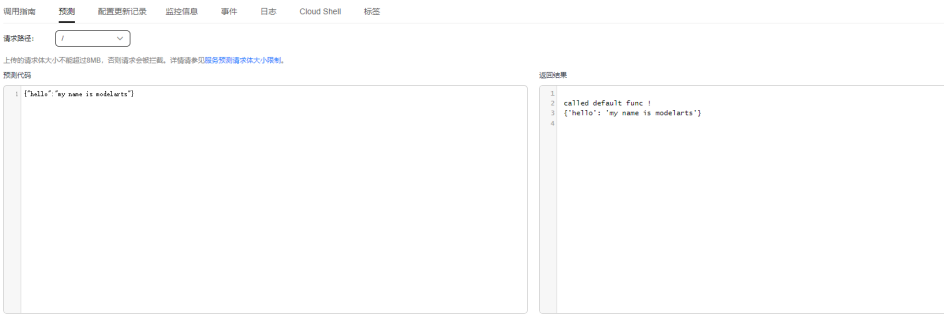
本样例的配置文件如下所示：

```
[{
      "url": "/",
      "method": "post",
      "request": {
         "Content-type": "application/json"
      },
      "response": {
         "Content-type": "application/json"
      }
   },
{
      "url": "/greet",
      "method": "post",
      "request": {
         "Content-type": "application/json"
      },
      "response": {
         "Content-type": "application/json"
      }
   },
{
      "url": "/goodbye",
      "method": "get",
      "request": {
         "Content-type": "application/json"
      },
      "response": {
         "Content-type": "application/json"
      }
   }
]
```

将 AI 应用部署为在线服务

1. 参考**部署为在线服务**将AI应用部署为在线服务。
2. 在线服务创建成功后，您可以在服务详情页查看服务详情。
3. 您可以通过“预测”页签访问在线服务。

图 5-5 访问在线服务



5.2 推理服务访问公网

本章节提供了推理服务访问公网的方法。

应用场景

推理服务访问公网地址的场景，如：

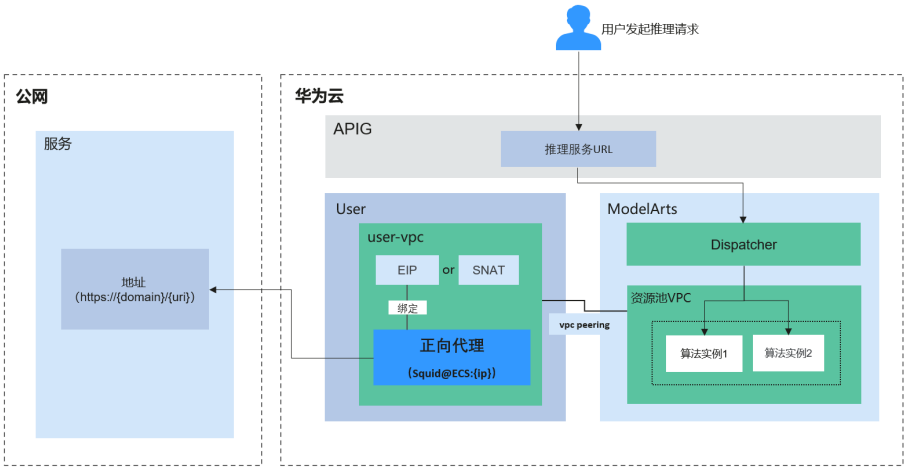
版权所有 © 华为云计算技术有限公司

- 输入图片，先进行公网OCR服务调用，然后进行NLP处理；
- 进行公网文件下载，然后进行分析；
- 分析结果回调给公网服务终端。

方案设计

从推理服务的算法实例内部，访问公网服务地址的方案。如下图所示：

图 5-6 推理服务访问公网



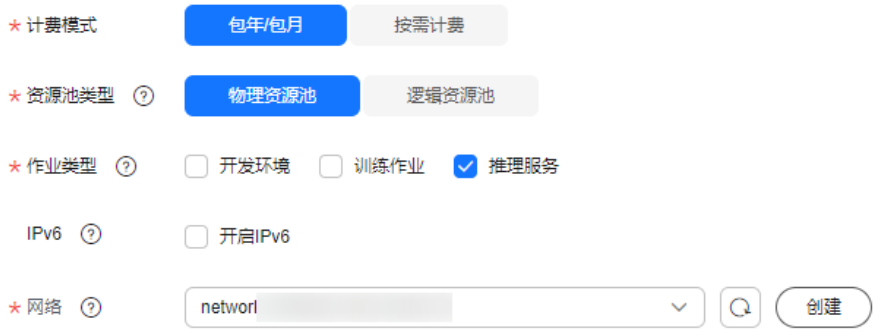
操作事项

- **ModelArts：设置资源池的网络**
- **用户VPC：安装和配置正向代理**
- **算法镜像：设置DNS代理和公网地址调用**

步骤1 ModelArts：设置资源池的网络

专属资源池的创建作业类型包含推理服务，选择的网络需打通VPC网络，如下图所示：

图 5-7 创建专属资源池



版权所有 © 华为云计算技术有限公司

图 5-8 打通 VPC

打通VPC可实现ModelArts资源池和用户VPC的网络打通。打通VPC前需要提前创建好VPC和子网，具体步骤请参考**创建虚拟私有云和子网**。

步骤2 用户VPC：安装和配置正向代理

在安装正向代理前，需要先购买一台弹性云服务器ECS（镜像可选择Ubuntu最新版本），并配置好弹性EIP，然后登录ECS进行正向代理Squid的安装和配置，步骤如下：

1. 如果没有安装Docker，执行以下命令进行Docker安装
   ```
   curl -sSL https://get.daocloud.io/docker | sh
   ```
2. 拉取Squid镜像
   ```
   docker pull ubuntu/squid
   ```
3. 创建主机目录，配置whitelist.conf和squid.conf

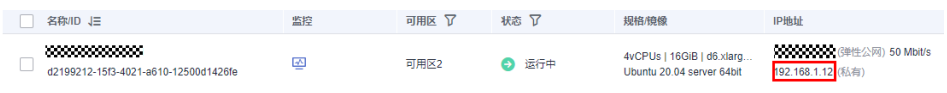
   先创建主机目录：
   ```
   mkdir –p /etc/squid/
   ```
   添加whitelist.conf配置文件，内容为安全控制可访问的地址，如：
   ```
   .apig.cn-east-3.huaweicloudapis.com
   ```
   添加squid.conf配置文件，内容如下：
   ```
   # An ACL named 'whitelist'
   acl whitelist dstdomain '/etc/squid/whitelist.conf'

   # Allow whitelisted URLs through
   http_access allow whitelist

   # Block the rest
   http_access deny all

   # Default port
   http_port 3128
   ```
   然后设置主机目录和配置文件权限如下：
   ```
   chmod 640 -R /etc/squid
   ```
4. 启动squid实例
   ```
   docker run -d --name squid -e TZ=UTC -v /etc/squid:/etc/squid -p 3128:3128 ubuntu/squid:latest
   ```
5. 如果whitelist.conf或squid.conf有更新，则进入容器刷新squid
   ```
   docker exec –it squid bash
   root@{container_id}:/# squid -k reconfigure
   ```

步骤3 算法镜像：设置DNS代理和公网地址调用

1. 设置代理

   在代码中设置代理指向代理服务器私有IP和端口，如下所示：
   ```
   proxies = {
     "http": "http://{proxy_server_private_ip}:3128",
     "https": "http://{proxy_server_private_ip}:3128"
   }
   ```
   服务器私有IP获取如下图所示：

图 5-9 ECS 私有 IP



 版权所有 © 华为云计算技术有限公司

2. 地址调用

   在推理代码中，使用服务URL进行业务请求，如：

   ```
   https://e8a048ce25136addbbac23ce6132a.apig.cn-east-3.huaweicloudapis.com
   ```

----结束

5.3 推理服务端到端运维

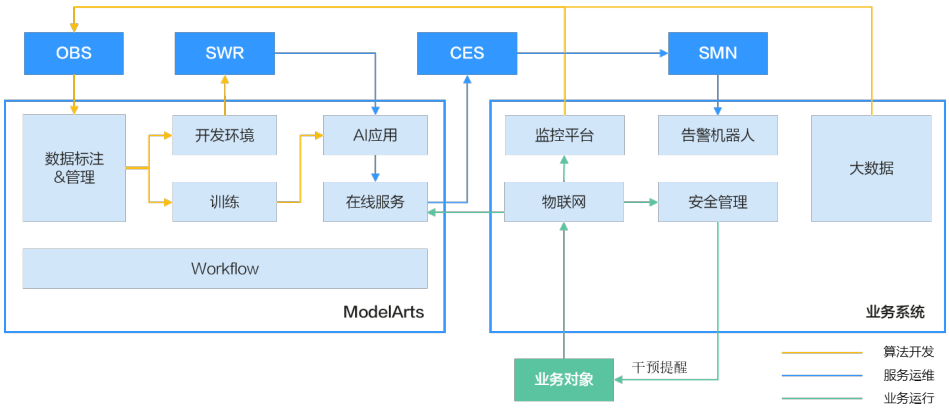
ModelArts推理服务的端到端运维覆盖了算法开发、服务运维和业务运行的整个AI流程。

方案概述

推理服务的端到端运维流程

- 算法开发阶段，先将业务AI数据存放到对象存储服务（OBS）中，接着通过ModelArts数据管理进行标注和版本管理，然后通过训练获得AI模型结果，最后通过开发环境构建AI应用镜像。
- 服务运维阶段，先利用镜像构建AI应用，接着部署AI应用为在线服务，然后可在云监控服务（CES）中获得ModelArts推理在线服务的监控数据，最后可配置告警规则实现实时告警通知。
- 业务运行阶段，先将业务系统对接在线服务请求，然后进行业务逻辑处理和监控设置。

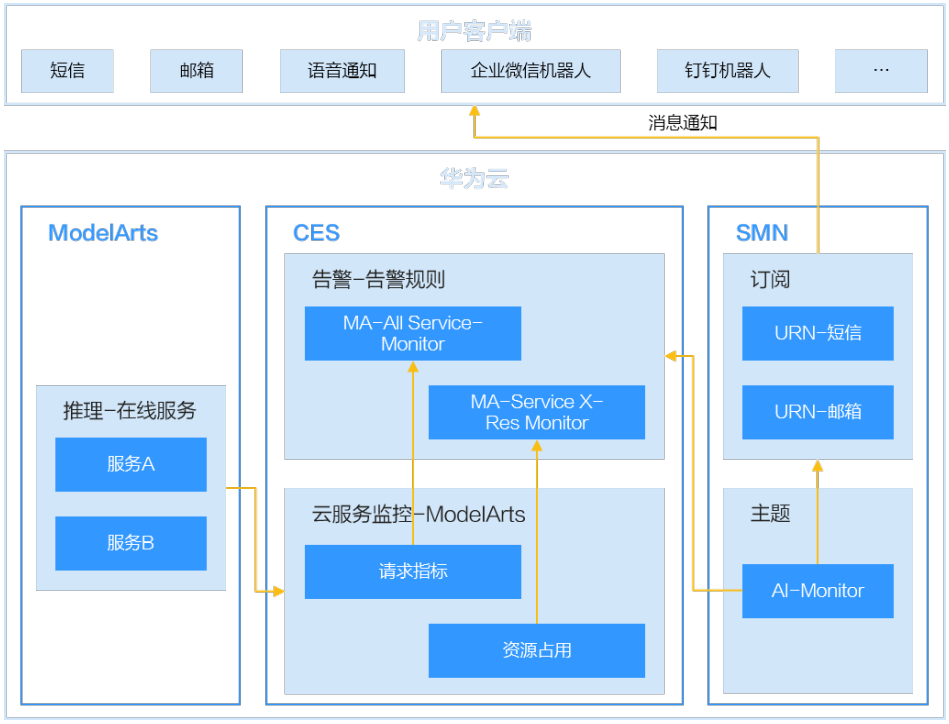
图 5-10 推理服务的端到端运维流程图



整个运维过程会对服务请求失败和资源占用过高的场景进行监控，当超过阈值时发送告警通知。

版权所有 © 华为云计算技术有限公司

图 5-11 监控告警流程图



方案优势

通过端到端的服务运维配置，可方便地查看业务运行高低峰情况，并能够实时感知在线服务的健康状态。

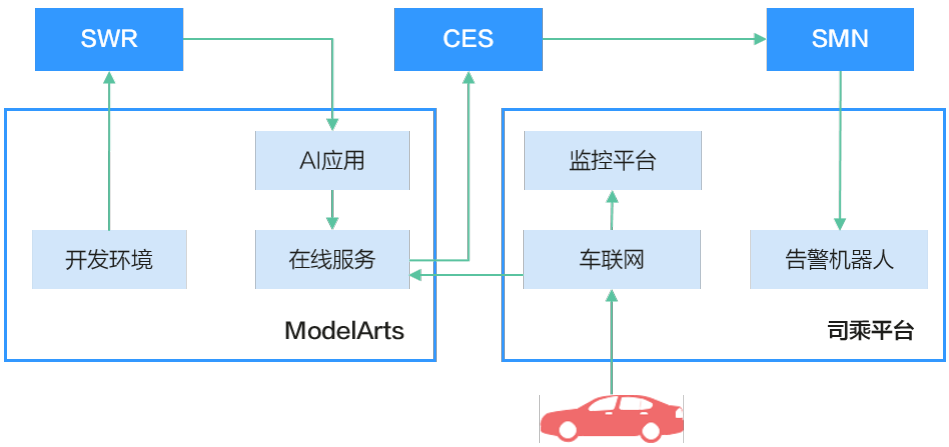
约束限制

端到端服务运维只支持在线服务，因为推理的批量服务和边缘服务无CES监控数据，不支持完整的端到端服务运维设置。

实施步骤

以出行场景的司乘安全算法为例，介绍使用ModelArts进行流程化服务部署和更新、自动化服务运维和监控的实现步骤。

图 5-12 司乘安全算法



版权所有 © 华为云计算技术有限公司

步骤1 将用户本地开发完成的模型，使用自定义镜像在ModelArts构建成AI应用。具体操作请参考**从0-1制作自定义镜像并创建AI应用**。

步骤2 在ModelArts管理控制台，使用创建好的AI应用部署为在线服务。

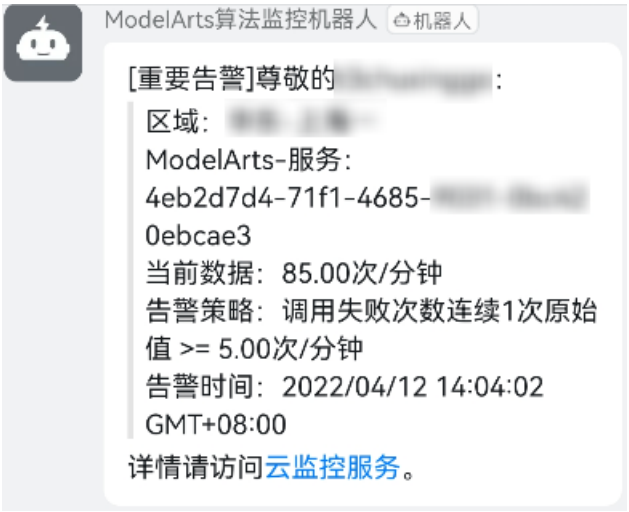
步骤3 登录云监控服务CES管理控制台，设置ModelArts服务的告警规则并配置主题订阅方式发送通知。具体操作请参考**设置告警规则**。

当配置完成后，在左侧导航栏选择“云服务监控 > ModelArts”即可查看在线服务的请求情况和资源占用情况，如下图所示。

图 5-13 查看服务的监控指标

当监控信息触发告警时，主题订阅对象将会收到消息通知。

图 5-14 告警消息通知



----结束

5.4 使用自定义引擎创建 AI 应用

使用自定义引擎创建AI应用，用户可以通过选择自己存储在SWR服务中的镜像作为AI应用的引擎，指定预先存储于OBS服务中的文件目录路径作为模型包，来创建AI应用，轻松地应对ModelArts平台预置引擎无法满足个性化诉求的场景。

ModelArts将自定义引擎类型的AI应用部署为服务时，会先将AI应用相关的SWR镜像下载至集群中，用“uid=1000, gid=100”的用户启动SWR镜像为容器，然后将OBS文件

版权所有 © 华为云计算技术有限公司

下载到容器中的“/home/mind/model”目录下，最后执行SWR镜像中预置的启动命令。ModelArts平台将暴露在容器“8080”端口的服务注册到APIG，用户可以通过提供的APIG（API网关）URL访问到该服务。

自定义引擎创建 AI 应用的规范

使用自定义引擎创建AI应用，用户的SWR镜像、OBS模型包和文件大小需要满足以下规范：

- SWR镜像规范：
  - 镜像必须内置一个用户名为“ma-user”，组名为“ma-group”的普通用户，且必须确保该用户的uid=1000、gid=100。内置用户的dockerfile指令如下：
    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```
  - 明确设置镜像的启动命令。在dockerfile文件中指定cmd，dockerfile指令示例如下：
    ```
    CMD sh /home/mind/run.sh
    ```
    启动入口文件run.sh需要自定义。示例如下：
    ```
    #!/bin/bash

    # 自定义脚本内容
    ...

    # run.sh调用app.py启动服务器，app.py请参考https示例
    python app.py
    ```
  - 提供的服务必须使用https协议， 且暴露在“8080”端口。请参考**https示例**。
  - （可选）在“8080”端口，提供URL路径为“/health”的健康检查服务（健康检查的URL路径必须为 "/health"）。
- OBS模型包规范

  模型包的名字必须为model。模型包规范请参见**模型包规范介绍**。
- 文件大小规范

  当使用公共资源池时，SWR的镜像大小（指下载后的镜像大小，非SWR界面显示的压缩后的镜像大小）和OBS模型包大小总和不大于30G。

https 示例

使用Flask启动https，Webserver代码示例如下：

```
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 ----------")
```


版权所有 © 华为云计算技术有限公司

```
    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))

@app.route('/health', methods=['GET'])
def healthy():
    return "{\"status\": \"OK\"}"

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

在本地机器调试

自定义引擎的规范可以在安装有docker的本地机器上通过以下步骤提前验证：

1. 将自定义引擎镜像下载至本地机器，假设镜像名为custom_engine:v1。
2. 将模型包文件夹复制到本地机器，假设模型包文件夹名字为model。
3. 在模型包文件夹的同级目录下验证如下命令拉起服务：
   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   说明

   该指令无法完全模拟线上，主要是由于-v挂载进去的目录是root权限。在线上，模型文件从OBS下载到/home/mind/model目录之后，文件owner将统一修改为ma-user。
4. 在本地机器上启动另一个终端，执行以下验证指令，得到符合预期的推理结果。
   ```
   curl https://127.0.0.1:8080/${推理服务的请求路径}
   ```

推理部署示例

本节将详细说明以自定义引擎方式创建AI应用的步骤。

1. 创建AI应用并查看AI应用详情

   登录ModelArts管理控制台，进入“AI应用管理>AI应用”页面中，单击“创建AI应用”，进入AI应用创建页面，设置相关参数如下：
   - 元模型来源：选择“从对象存储服务（OBS）中选择”。
   - 选择元模型：从OBS中选择一个模型包。
   - AI引擎：选择“Custom”。
   - 引擎包：从容器镜像中选择一个镜像。

     其他参数保持默认值。

   单击“立即创建”，跳转到AI应用列表页，查看AI应用状态，当状态变为“正常”，AI应用创建成功。

版权所有 © 华为云计算技术有限公司

图 5-15 创建 AI 应用

单击AI应用名称，进入AI应用详情页面，查看AI应用详情信息。

2. 部署服务并查看详情

   在AI应用详情页面，单击右上角“部署>在线服务”，进入服务部署页面，AI应用和版本默认选中，选择合适的“计算节点规格”（例如CPU：2核 8GB），其他参数可保持默认值，单击“下一步”，跳转至服务列表页，当服务状态变为“运行中”，服务部署成功。

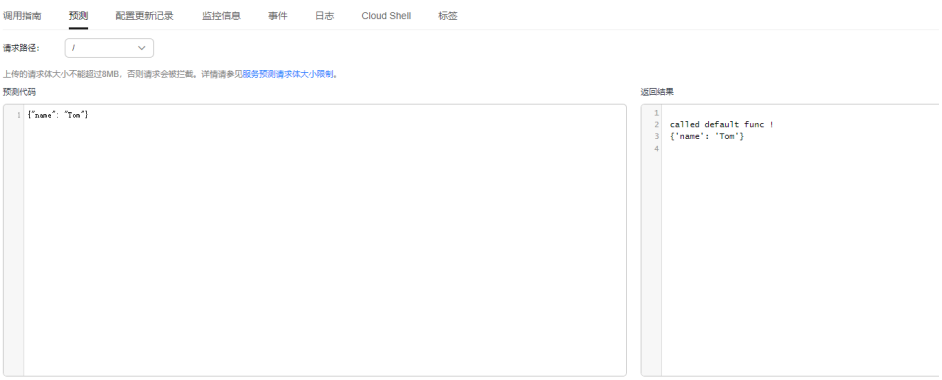
   单击服务名称，进入服务详情页面，查看服务详情信息，单击“日志”页签，查看服务日志信息。

图 5-16 查看服务日志信息

3. 服务预测

   在服务详情页面，单击“预测”页签，进行服务预测。

图 5-17 服务预测



版权所有 © 华为云计算技术有限公司

5.5 使用大模型创建 AI 应用部署在线服务

背景说明

目前大模型的参数量已经达到千亿甚至万亿，随之大模型的体积也越来越大。千亿参数大模型的体积超过200G，在版本管理、生产部署上对平台系统产生了新的要求。例如：导入AI应用管理时，需要支持动态调整租户存储配额；模型加载、启动慢，部署时需要灵活的超时配置；当负载异常重启，模型需要重新加载，服务恢复时间长的问题亟待解决。

为了应对如上诉求，ModelArts推理平台针对性给出解决方案，用于支持大模型场景下的AI应用管理和服务部署。

约束与限制

- 需要申请单个AI应用大小配额和添加使用节点本地存储缓存的白名单
- 需要使用自定义引擎Custom，配置动态加载
- 需要使用专属资源池部署服务
- 专属资源池磁盘空间需大于1T

操作事项

1. **申请扩大AI应用的大小配额和使用节点本地存储缓存白名单**
2. **上传模型数据并校验上传对象的一致性**
3. **创建专属资源池**
4. **创建AI应用**
5. **部署在线服务**

申请扩大 AI 应用的大小配额和使用节点本地存储缓存白名单

服务部署时，默认情况下，动态加载的模型包位于临时磁盘空间，服务停止时已加载的文件会被删除，再次启动时需要重新加载。为了避免反复加载，平台允许使用资源池节点的本地存储空间来加载模型包，并在服务停止和重启时仍有效（通过哈希值保证数据一致性）

使用大模型要求用户采用自定义引擎，并开启动态加载的模式导入模型。基于此，需要执行以下操作：

- 如果模型超过默认配额值，需要提工单申请扩大单个AI应用的大小配额。单个AI应用大小配额默认值为20GB。
- 需要提工单申请添加使用节点本地存储缓存的白名单。

上传模型数据并校验上传对象的一致性

为了动态加载时保证数据完整性，需要在上传模型数据至OBS时，进行上传对象的一致性校验。obsutil、OBS Browser+以及OBS SDK都支持在上传对象时进行一致性校验，您可以根据自己的业务选择任意一种方式进行校验。详见**校验上传对象的一致性**。

版权所有 © 华为云计算技术有限公司

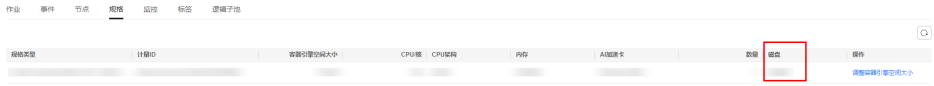
以OBS Browser+为例，如图5-18。使用OBS Browser+上传数据，开启MD5校验，动态加载并使用节点本地的持久化存储时，检查数据一致性。

图 5-18 OBS Browser+配置 MD5 校验

创建专属资源池

使用本地的持久化存储功能，需使用专属资源池，且专属资源池磁盘空间大小必须超过1T。您可以通过专属资源池详情页面，规格页签，查看专属资源池磁盘信息。当服务部署失败，提示磁盘空间不足时，请参考**服务部署、启动、升级和修改时，资源不足如何处理?**

图 5-19 查看专属资源池磁盘信息



创建 AI 应用

使用大模型创建AI应用，选择从对象存储服务（OBS）中导入，需满足以下参数配置：

1. 采用自定义引擎，开启动态加载

   使用大模型要求用户使用自定义引擎，并开启动态加载的模式导入模型。用户可以制作自定义引擎，满足大模型场景下对镜像依赖包、推理框架等的特殊需求。自定义引擎的制作请参考**使用自定义引擎创建AI应用**。

   当用户使用自定义引擎时，默认开启动态加载，模型包与镜像分离，在服务部署时动态将模型加载到服务负载。

2. 配置健康检查

   大模型场景下导入的AI应用，要求配置健康检查，避免在部署时服务显示已启动但实际不可用。

版权所有 © 华为云计算技术有限公司

图 5-20 采用自定义引擎，开启动态加载并配置健康检查示例图

部署在线服务

部署服务时，需满足以下参数配置：

1. 自定义部署超时时间

   大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

2. 添加环境变量

   部署服务时，增加如下环境变量，会将负载均衡的请求亲和策略配置为集群亲和，避免未就绪的服务实例影响预测成功率。

   ```
   MODELARTS_SERVICE_TRAFFIC_POLICY: cluster
   ```

图 5-21 自定义部署超时时间和添加环境变量示例图

建议部署多实例，增加服务可靠性。

5.6 第三方推理框架迁移到推理自定义引擎

背景说明

ModelArts支持第三方的推理框架在ModelArts上部署，本文以TFServing框架、Triton框架为例，介绍如何迁移到推理自定义引擎。

- TensorFlow Serving是一个灵活、高性能的机器学习模型部署系统，提供模型版本管理、服务回滚等能力。通过配置模型路径、模型端口、模型名称等参数，原

版权所有 © 华为云计算技术有限公司

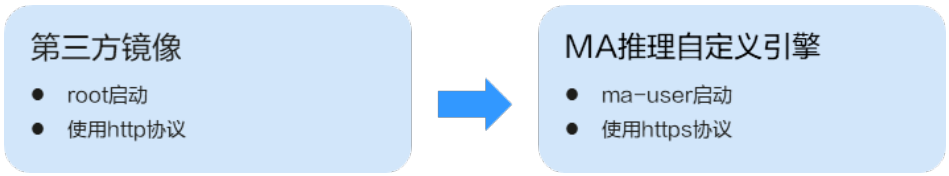
生TFServing镜像可以快速启动提供服务，并支持gRPC和HTTP Restful API的访问方式。

- Triton是一个高性能推理服务框架，提供HTTP/gRPC等多种服务协议，支持TensorFlow、TensorRT、PyTorch、ONNXRuntime等多种推理引擎后端，并且支持多模型并发、动态batch等功能，能够提高GPU的使用率，改善推理服务的性能。

当从第三方推理框架迁移到使用ModelArts推理的AI应用管理和服务管理时，需要对原生第三方推理框架镜像的构建方式做一定的改造，以使用ModelArts推理平台的模型版本管理能力和动态加载模型的部署能力。本案例将指导用户完成原生第三方推理框架镜像到ModelArts推理自定义引擎的改造。自定义引擎的镜像制作完成后，即可以通过AI应用导入对模型版本进行管理，并基于AI应用进行部署和管理服务。

适配和改造的主要工作项如下：

图 5-22 改造工作项



针对不同框架的镜像，可能还需要做额外的适配工作，具体差异请见对应框架的操作步骤。

- **TFServing框架迁移操作步骤**
- **Triton框架迁移操作步骤**

TFServing 框架迁移操作步骤

步骤1 增加用户ma-user。

基于原生"tensorflow/serving:2.8.0"镜像构建，镜像中100的用户组默认已存在，Dockerfile中执行如下命令增加用户ma-user。

```
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

协议转换为https之后，对外暴露的端口从tfserving的8501变为8080。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

2. 准备nginx目录如下：

版权所有 © 华为云计算技术有限公司

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
```

 版权所有 © 华为云计算技术有限公司

```
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8501;
    }
}
```

5. 准备启动脚本。

   说明

   启动前先创建ssl证书，然后启动TFServing的启动脚本。

   启动脚本run.sh示例代码如下：

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText
sh /usr/bin/tf_serving_entrypoint.sh
```

步骤3 修改模型默认路径，支持ModelArts推理模型动态加载。

Dockerfile中执行如下命令修改默认的模型路径。

```
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
```

----结束

完整的Dockerfile参考：

```
FROM tensorflow/serving:2.8.0
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
```

 版权所有 © 华为云计算技术有限公司

```
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

Triton 框架迁移操作步骤

本教程基于nvidia官方提供的nvcr.io/nvidia/tritonserver:23.03-py3镜像进行适配，使用开源大模型llama7b进行推理任务。

步骤1 增加用户ma-user。

Triton镜像中默认已存在id为1000的triton-server用户，需先修改triton-server用户名id后再增加用户ma-user，Dockerfile中执行如下命令。

```
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
```

2. 准备nginx目录如下：

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
```

 版权所有 © 华为云计算技术有限公司

```
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

5. 准备启动脚本run.sh。

 版权所有 © 华为云计算技术有限公司

说明

启动前先创建ssl证书，然后启动Triton的启动脚本。

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText

bash /home/mind/model/triton_serving.sh
```

步骤3 编译安装tensorrtllm_backend。

1. Dockerfile中执行如下命令获取tensorrtllm_backend源码，安装tensorrt、cmake和pytorch等相关依赖，并进行编译安装。

```
# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver
```

2. 准备triton serving的启动脚本triton_serving.sh，llama模型的参考样例如下：

```
MODEL_NAME=llama_7b
MODEL_DIR=/home/mind/model/${MODEL_NAME}
OUTPUT_DIR=/tmp/llama/7B/trt_engines/fp16/1-gpu/
MAX_BATCH_SIZE=1
export LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH}

# build tensorrt_llm engine
cd /opt/tritonserver/tensorrtllm_backend/tensorrt_llm/examples/llama
python build.py --model_dir ${MODEL_DIR} \
                --dtype float16 \
                --remove_input_padding \
                --use_gpt_attention_plugin float16 \
                --enable_context_fmha \
                --use_weight_only \
                --use_gemm_plugin float16 \
                --output_dir ${OUTPUT_DIR} \
```

 版权所有 © 华为云计算技术有限公司

```
            --paged_kv_cache \
            --max_batch_size ${MAX_BATCH_SIZE}

# set config parameters
cd /opt/tritonserver/tensorrtllm_backend
mkdir triton_model_repo
cp all_models/inflight_batcher_llm/* triton_model_repo/ -r

python3 tools/fill_template.py -i triton_model_repo/preprocessing/config.pbtxt tokenizer_dir:${MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:${MAX_BATCH_SIZE},preprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/postprocessing/config.pbtxt tokenizer_dir:${MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:${MAX_BATCH_SIZE},postprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/ensemble/config.pbtxt triton_max_batch_size:${MAX_BATCH_SIZE}
python3 tools/fill_template.py -i triton_model_repo/tensorrt_llm/config.pbtxt triton_max_batch_size:${MAX_BATCH_SIZE},decoupled_mode:False,max_beam_width:1,engine_dir:${OUTPUT_DIR},max_tokens_in_paged_kv_cache:2560,max_attention_window_size:2560,kv_cache_free_gpu_mem_fraction:0.5,exclude_input_in_output:True,enable_kv_cache_reuse:False,batching_strategy:V1,max_queue_delay_microseconds:600

# launch tritonserver
python3 scripts/launch_triton_server.py --world_size 1 --model_repo=triton_model_repo/
while true; do sleep 10000; done
```

部分参数说明：

- MODEL_NAME：HuggingFace格式模型权重文件所在OBS文件夹名称。
- OUTPUT_DIR：通过TensorRT-LLM转换后的模型文件在容器中的路径。

完整的Dockerfile如下：

```
FROM nvcr.io/nvidia/tritonserver:23.03-py3

# add ma-user and install nginx
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user && \
    apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx

# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
```


版权所有 © 华为云计算技术有限公司

```
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver

ADD nginx /etc/nginx
ADD run.sh /home/mind/
CMD /bin/bash /home/mind/run.sh
```

完成镜像构建后，将镜像注册至华为云容器镜像服务SWR中，用于后续在ModelArts上部署推理服务。

步骤4 使用适配后的镜像在ModelArts部署在线推理服务。

1. 在obs中创建model目录，并将triton_serving.sh文件和llama_7b文件夹上传至model目录下，如下图所示。

图 5-23 上传至 model 目录

2. 创建AI应用，源模型来源选择“从对象存储服务（OBS）中选择”，元模型选择至model目录，AI引擎选择Custom，引擎包选择步骤3构建的镜像。

图 5-24 创建 AI 应用

版权所有 © 华为云计算技术有限公司

3. 将创建的AI应用部署为在线服务，大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

图 5-25 部署为在线服务

4. 调用在线服务进行大模型推理，请求路径填写/v2/models/ensemble/infer，调用样例如下：

```
{
  "inputs": [
    {
      "name": "text_input",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": ["what is machine learning"]
    },
    {
      "name": "max_tokens",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [64]
    },
    {
      "name": "bad_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "stop_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "pad_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    },
    {
      "name": "end_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    }
  ],
  "outputs": [
    {
      "name": "text_output"
    }
  ]
}
```


版权所有 © 华为云计算技术有限公司

说明

- "inputs"中"name"为"text_input"的元素代表输入，"data"为具体输入语句，本示例中为"what is machine learning"。
- "inputs"中"name"为"max_tokens"的元素代表输出最大tokens数，"data"为具体数值，本示例中为64。

图 5-26 调用在线服务

----结束

5.7 推理服务支持虚拟私有云（VPC）直连的高速访问通道

背景说明

访问在线服务的实际业务中，用户可能会存在如下需求：

- 高吞吐量、低时延
- TCP或者RPC请求

因此，ModelArts提供了VPC直连的高速访问通道功能以满足用户的需求。

使用VPC直连的高速访问通道，用户的业务请求不需要经过推理平台，而是直接经VPC对等连接发送到实例处理，访问速度更快。

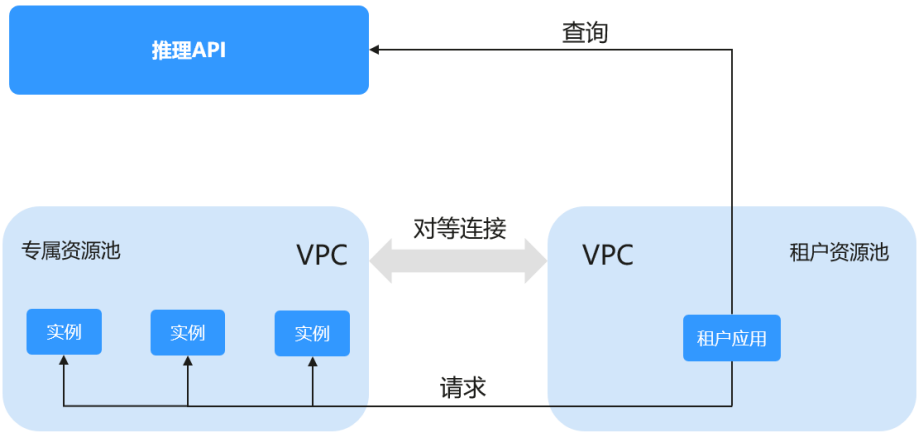
说明

由于请求不经过推理平台，所以会丢失以下功能：

- 认证鉴权
- 流量按配置分发
- 负载均衡
- 告警、监控和统计

版权所有 © 华为云计算技术有限公司

图 5-27 VPC 直连的高速访问通道示意图



准备工作

使用专属资源池部署在线服务，服务状态为“运行中”。

须知

- 需使用新版专属资源池部署服务，详情请参见**ModelArts资源池管理功能全面升级**。
- 只有专属资源池部署的服务才支持VPC直连的高速访问通道。
- VPC直连的高速访问通道，目前只支持访问在线服务。
- 因流量限控，获取在线服务的IP和端口号次数有限制，每个主账号租户调用次数不超过2000次/分钟，每个子账号租户不超过20次/分钟。
- 目前仅支持自定义镜像导入模型，部署的服务支持高速访问通道。

操作步骤

使用VPC直连的高速访问通道访问在线服务，基本操作步骤如下：

1. **将专属资源池的网络打通VPC**
2. **VPC下创建弹性云服务器**
3. **获取在线服务的IP和端口号**
4. **通过IP和端口号直连应用**

步骤1 将专属资源池的网络打通VPC

登录ModelArts控制台，进入“专属资源池 > 弹性集群”找到服务部署使用的专属资源池，单击“名称/ID”，进入资源池详情页面，查看网络配置信息。返回专属资源池列表，选择“网络”页签，找到专属资源池关联的网络，打通VPC。打通VPC网络后，网络列表和资源池详情页面将显示VPC名称，单击后可以跳转至VPC详情页面。

版权所有 © 华为云计算技术有限公司

图 5-28 查找专属资源池

图 5-29 查看网络配置

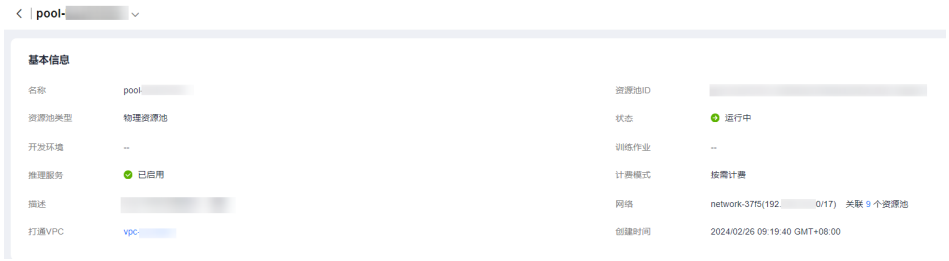


图 5-30 打通 VPC

步骤2 VPC下创建弹性云服务器

登录弹性云服务器ECS控制台，单击右上角“购买弹性云服务器”，进入购买弹性云服务器页面，完成基本配置后单击“下一步：网络配置”，进入网络配置页面，选择**步骤1**中打通的VPC，完成其他参数配置，完成高级配置并确认配置，下发购买弹性云服务器的任务。等待服务器的状态变为“运行中”时，弹性云服务器创建成功。单击“名称/ID”，进入服务器详情页面，查看虚拟私有云配置信息。

图 5-31 购买弹性云服务器时选择 VPC

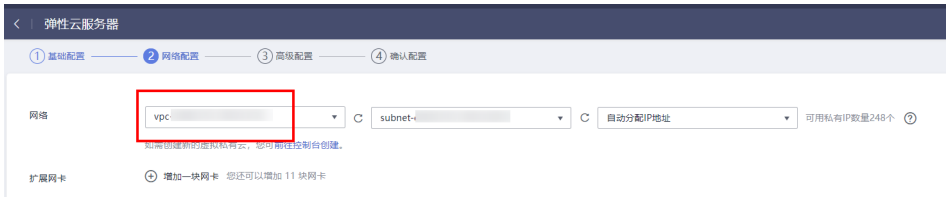
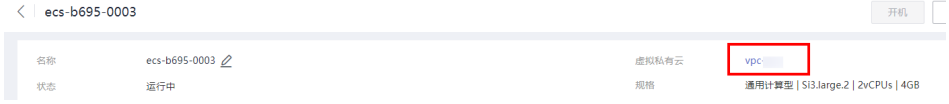


图 5-32 查看虚拟私有云配置信息



步骤3 获取在线服务的IP和端口号

可以通过使用图形界面的软件（以Postman为例）获取服务的IP和端口号，也可以登录弹性云服务器（ECS），创建Python环境运行代码，获取服务IP和端口号。

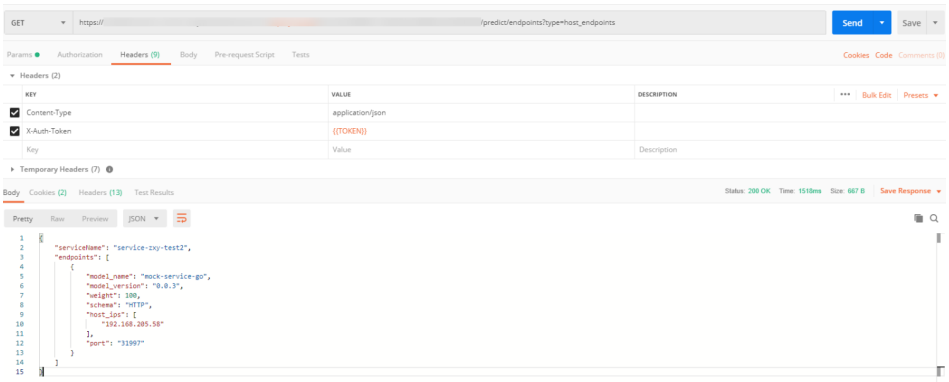
版权所有 © 华为云计算技术有限公司

API接口：

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- 方式一：图形界面的软件获取服务的IP和端口号

  图 5-33 接口返回示例

- 方式二：Python语言获取IP和端口号

  Python代码如下，下述代码中以下参数需要手动修改：

  - project_id：用户项目ID，获取方法请参见**获取项目ID和名称**。
  - service_id：服务ID，在服务详情页可查看。
  - REGION_ENDPOINT：服务的终端节点，查询请参见**终端节点**。

  ```
  def get_app_info(project_id, service_id):
      list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?type=host_endpoints"
      url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id, service_id)
      headers = {'X-Auth-Token': X_Auth_Token}
      response = requests.get(url, headers=headers)
      print(response.content)
  ```

步骤4 通过IP和端口号直连应用

登录弹性云服务器（ECS），可以通过Linux命令行访问在线服务，也可以创建Python环境运行Python代码访问在线服务。schema、 ip、port参数值从**步骤3**获取。

- 执行命令示例如下，直接访问在线服务。

  ```
  curl --location --request POST 'http://192.168.205.58:31997' \
  --header 'Content-Type:  application/json' \
  --data-raw '{"a":"a"}'
  ```

  图 5-34 访问在线服务

- 创建Python环境，运行Python代码访问在线服务。

  ```
  def vpc_infer(schema, ip, port, body):
      infer_url = "{}://{}:{}"
      url = infer_url.format(schema, ip, port)
      response = requests.post(url, data=body)
      print(response.content)
  ```

说明

由于高速通道特性会缺失负载均衡的能力，因此在多实例时需要自主制定负载均衡策略。

----结束

 版权所有 © 华为云计算技术有限公司

5.8 WebSocket 在线服务全流程开发

背景说明

WebSocket是一种网络传输协议，可在单个TCP连接上进行全双工通信，位于OSI模型的应用层。WebSocket协议在2011年由IETF标准化为RFC 6455，后由RFC 7936补充规范。Web IDL中的WebSocket API由W3C标准化。

WebSocket使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握手，两者之间就可以建立持久性的连接，并进行双向数据传输。

前提条件

- 用户需有一定的Java开发经验，熟悉jar打包流程。
- 用户需了解WebSocket协议的基本概念及调用方法。
- 用户需熟悉Docker制作镜像的方法。

约束与限制

- WebSocket协议只支持部署在线服务。
- 只支持自定义镜像导入AI应用部署的在线服务。

准备工作

ModelArts使用WebSocket完成推理需要用户自己准备自定义镜像，该自定义镜像需要在单机环境下能够提供完整的WebSocket服务，如完成WebSocket的握手，client向server发送数据，server向client发送数据等。模型的推理过程在自定义镜像中完成，如下载模型，加载模型，执行预处理，完成推理，拼装响应体等。

操作步骤

WebSocket在线服务开发操作步骤如下：

- **上传镜像至容器镜像服务**
- **使用镜像创建AI应用**
- **使用AI应用部署在线服务**
- **WebSocket在线服务调用**

上传镜像至容器镜像服务

将准备好的本地镜像上传到容器镜像服务（SWR）。上传镜像的详细操作可参考**如何登录并上传镜像到SWR**。

使用镜像创建 AI 应用

1. 登录ModelArts管理控制台，进入“AI应用管理 > AI应用”页面，单击“创建”，跳转至创建AI应用页面。
2. 完成AI应用配置，部分配置如下：

版权所有 © 华为云计算技术有限公司

- 元模型来源：选择“从容器镜像中选择”。
- 容器镜像所在的路径：选择**上传镜像至容器镜像服务**上传的路径。
- 容器调用接口：根据实际情况配置容器调用接口。
- 健康检查：保持默认。如果镜像中配置了健康检查则按实际情况配置健康检查。

图 5-35 AI 应用配置参数

3. 单击“立即创建”，进入AI应用列表页，等AI应用状态变为“正常”，表示AI应用创建成功。

使用 AI 应用部署在线服务

1. 登录ModelArts管理控制台，进入“部署上线 >在线服务”页面，单击“部署”，跳转至在线服务部署页面。
2. 完成服务的配置，部分配置如下：
   - 选择AI应用及版本：选择**使用镜像创建AI应用**创建完成的AI应用及版本
   - 升级为WebSocket：打开开关

图 5-36 升级为 WebSocket

3. 单击“下一步”，确认配置后“提交”，完成在线服务的部署。返回在线服务列表页，查看服务状态变为“运行中”，表示服务部署成功。

版权所有 © 华为云计算技术有限公司

WebSocket 在线服务调用

WebSocket协议本身不提供额外的认证方式。不管自定义镜像里面是ws还是wss，经过ModelArts平台出去的WebSocket协议都是wss的。同时wss只支持客户端对服务端的单向认证，不支持服务端对客户端的双向认证。

可以使用ModelArts提供的以下认证方式：

- **token认证**
- **AK/SK**
- **APP认证**

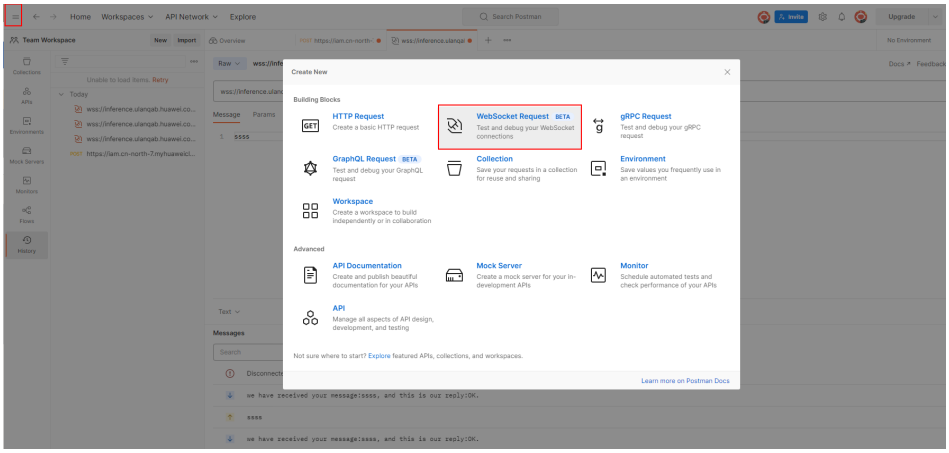
WebSocket服务调用步骤如下（以图形界面的软件Postman进行预测，token认证为例）：

1. **WebSocket连接的建立**
2. **WebSocket客户端和服务端双向传输数据**

步骤1 WebSocket连接的建立

1. 打开Postman（需选择8.5 以上版本，以10.12.0为例）工具，单击左上角 ，选择“File>New”，弹出新建对话框，选择“WebSocket Request”（当前为beta版本）功能：

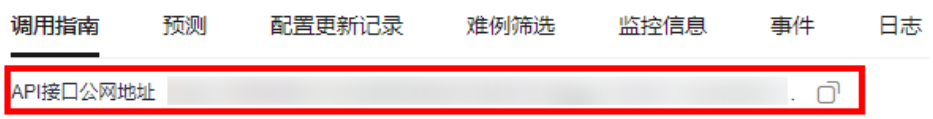
   图 5-37 选择 WebSocket Request 功能

2. 在新建的窗口中填入WebSocket连接信息：

   左上角选择Raw，不要选择Socket.IO（一种WebSocket实现，要求客户端跟服务端都要基于Socket.IO），地址栏中填入从服务详情页“调用指南”页签中获取“API接口调用公网地址”后面的地址。如果自定义镜像中有更细粒度的地址，则在地址后面追加该URL。如果有queryString，那么在params栏中添加参数。在header中添加认证信息（不同认证方式有不同header，跟https的推理服务相同）。选择单击右上的connect按钮，建立WebSocket连接。

   图 5-38 获取 API 接口调用公网地址

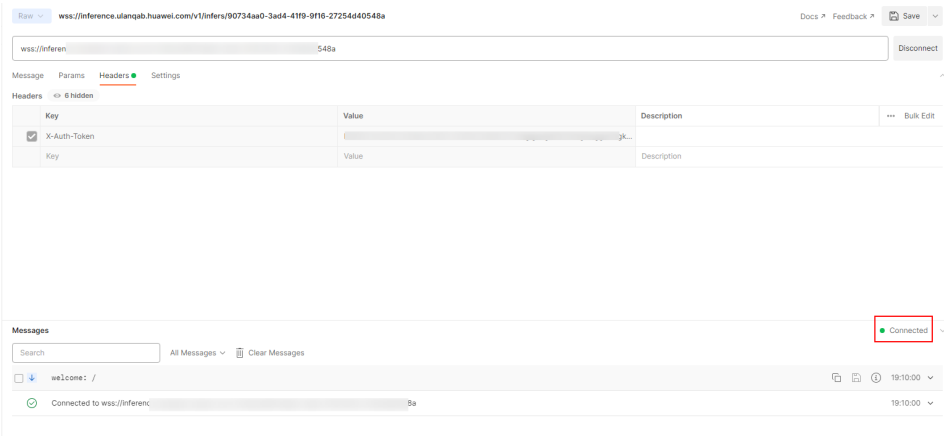
版权所有 © 华为云计算技术有限公司

说明

- 如果信息正确，右下角连接状态处会显示：CONNECTED；
- 如果无法建立连接，如果是401状态码，检查认证信息；
- 如果显示WRONG_VERSION_NUMBER等关键字，检查自定义镜像的端口和ws跟wss的配置是否正确。

连接成功后结果如下：

图 5-39 连接成功



须知

优先验证自定义镜像提供的websocket服务的情况，不同的工具实现的websocket服务会有不同，可能出现连接建立后维持不住，可能出现请求一次后连接就中断需要重新连接的情况，ModelArts平台只保证，未上ModelArts前自定义镜像的websocket的形态跟上了ModelArts平台后的websocket形态相同（除了地址跟认证方式不同）。

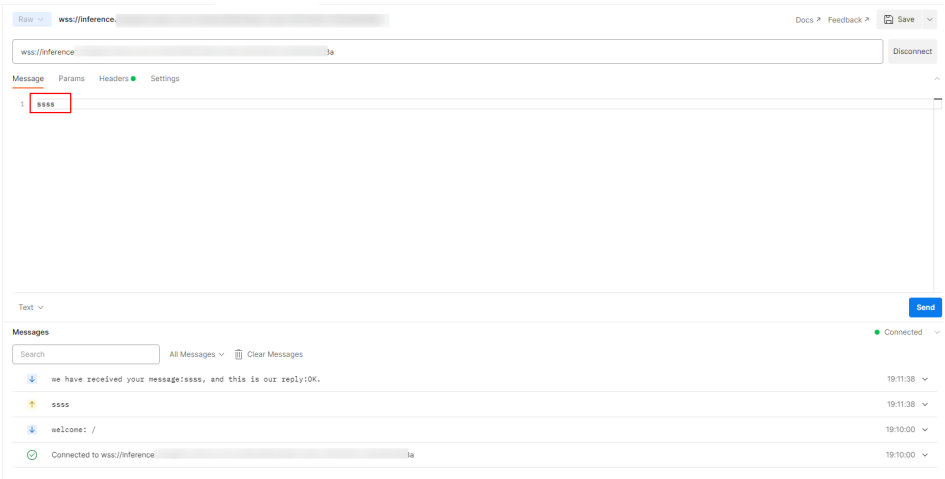
步骤2 WebSocket客户端和服务端双向传输数据

连接建立后，WebSocket使用TCP完成全双工通信。WebSocket的客户端可以往服务端发送数据，客户端有不同的实现，同一种语言也存在不同的lib包的实现，这里不考虑实现的不同种类。

客户端发送的内容在协议的角度不限定格式，Postman支持Text/Json/XML/HTML/Binary，以text为例，在输入框中输入要发送的文本，单击右侧中部的Send按钮即可将请求发往服务端，当文本内容过长，可能会导致postman工具卡住。

版权所有 © 华为云计算技术有限公司

图 5-40 发送数据



----结束

版权所有 © 华为云计算技术有限公司