ModelArts

Image Management

Edição 01

Data 2024-09-14

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Marcas registadas e permissões

e outras marcas registadas da Huawei são marcas registadas da Huawei Technologies Co., Ltd. Todos as outras marcas registadas e os nomes registados mencionados neste documento são propriedade dos seus respectivos detentores.

Aviso

Os produtos, os serviços e as funcionalidades adquiridos são estipulados pelo contrato estabelecido entre a Huawei Cloud e o cliente. Os produtos, os serviços e as funcionalidades descritos neste documento, no todo ou em parte, podem não estar dentro do âmbito de aquisição ou do âmbito de uso. Salvo especificação em contrário no contrato, todas as declarações, informações e recomendações neste documento são fornecidas "TAL COMO ESTÃO" sem garantias ou representações de qualquer tipo, sejam expressas ou implícitas.

As informações contidas neste documento estão sujeitas a alterações sem aviso prévio. Foram feitos todos os esforços na preparação deste documento para assegurar a exatidão do conteúdo, mas todas as declarações, informações e recomendações contidas neste documento não constituem uma garantia de qualquer tipo, expressa ou implícita.

Huawei Cloud Computing Technologies Co., Ltd.

Endereço: Huawei Cloud Data Center, Rua Jiaoxinggong
Avenida Qianzhong
Novo Distrito de Gui'an
Guizhou 550029
República Popular da China

Site: https://www.huaweicloud.com/intl/pt-br/

Índice

1 Gerenciamento de imagens

Visão geral

Durante o desenvolvimento e o tempo de execução dos serviços de IA, as dependências complexas do ambiente precisam ser depuradas para a conteinerização. Nas melhores práticas de desenvolvimento de IA no ModelArts as imagens de contêiner são usadas para fornecer ambientes de tempo de execução fixos. Desta forma, as dependências podem ser gerenciadas e os ambientes de tempo de execução podem ser facilmente trocados. Os recursos de contêiner fornecidos pelo ModelArts permitem o desenvolvimento rápido e eficiente de IA e a iteração de experimentos de modelos.

As imagens predefinidas fornecidas pelo ModelArts por padrão têm os seguintes recursos:

- Pronto para uso e específico para o cenário: ambientes dependentes típicos para desenvolvimento de IA são predefinidos nessas imagens para fornecer configurações ideais de software, sistema operacional e rede. Eles foram totalmente testados em hardware para garantir compatibilidade e desempenho ideais.
- Configuração personalizável: as imagens predefinidas são armazenadas no repositório do SWR para que você possa personalizá-las e registrá-las como suas próprias imagens.
- Seguro e confiável: políticas de acesso, controle de permissões de usuário, verificação de vulnerabilidades para software de desenvolvimento e sistema operacional são configurados com base nas melhores práticas de proteção de segurança para garantir a segurança das imagens.

Se você tiver requisitos especiais no mecanismo de aprendizado profundo e na biblioteca de desenvolvimento, use imagens personalizadas do ModelArts para personalizar mecanismos de tempo de execução.

Com base na tecnologia de contêiner, você pode personalizar imagens de contêiner e executá-las no ModelArts. As imagens personalizadas suportam parâmetros de CLI e variáveis de ambiente em formato de texto livre, apresentando alta flexibilidade para uma ampla gama de mecanismos de computação.

Cenários de aplicação de imagens predefinidas

O ModelArts fornece um grupo de imagens predefinidas. Você pode usar uma imagem predefinida para criar uma instância de notebook. Depois de instalar e configurar dependências na instância, crie uma imagem personalizada. Em seguida, você pode usar

diretamente a imagem no ModelArts para trabalhos de treinamento sem qualquer adaptação. Você também pode usar imagens predefinidas para enviar trabalhos de treinamento e criar aplicações de IA.

Recomendamos a versão de imagem predefinida com base em seus requisitos de desenvolvimento e estabilidade da versão. Se o seu desenvolvimento pode ser feito usando as versões predefinidas no ModelArts, por exemplo, MindSpore 1.*X*, use as imagens predefinidas. Elas foram totalmente verificadas e têm muitos pacotes de instalação comumente usados, aliviando você de configurar o ambiente.

Cenários de aplicação de images personalizadas

- **Usar imagens personalizadas em instâncias de notebook**

  Se as imagens predefinidas das instâncias de notebook não puderem atender aos requisitos, você poderá criar uma imagem personalizada instalando e configurando o software e outros dados exigidos pelo ambiente em uma imagem predefinida. Em seguida, use a imagem personalizada para criar novas instâncias de notebook.

- **Usar uma imagem personalizada para criar trabalhos de treinamento**

  Se você desenvolveu um modelo ou script de treinamento localmente, mas o mecanismo de IA usado não é suportado pelo ModelArts crie uma imagem personalizada e carregue-a no SWR. Em seguida, use esta imagem para criar um trabalho de treinamento no ModelArts e use os recursos fornecidos pelo ModelArts para treinar modelos.

- **Usar uma imagem personalizada para criar aplicações de IA**

  Se você desenvolveu um modelo usando um mecanismo de IA que não é suportado pelo ModelArts para usar esse modelo para criar aplicações de IA, faça o seguinte: crie uma imagem personalizada, importe a imagem para ModelArts e use-a para criar aplicações de IA. As aplicações de IA criadas dessa maneira podem ser gerenciadas centralmente e implementadas como serviços.

Serviços de imagem personalizados

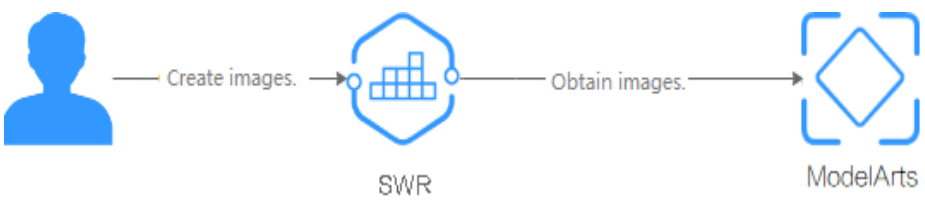
Quando você usa uma imagem personalizada, os seguintes serviços podem estar envolvidos:

- SWR

  O Software Repository for Container (SWR) fornece gerenciamento fácil, seguro e confiável de imagens de contêiner ao longo de seus ciclos de vida, facilitando a implementação de serviços em contêineres. Você pode carregar, baixar e gerenciar imagens de contêiner por meio do console do SWR, das APIs do SWR ou da CLI da comunidade.

  Suas imagens personalizadas devem ser carregadas no SWR. As imagens personalizadas usadas pelo ModelArts para treinamento ou criação de aplicações de IA são obtidas da lista de gerenciamento de serviços do SWR.

  Figura 1-1 Obtenção de imagens

- OBS

O Object Storage Service (OBS) é um serviço de armazenamento em nuvem otimizado para armazenar grandes quantidades de dados. Ele fornece recursos de armazenamento ilimitados, seguros e altamente confiáveis a um custo relativamente baixo.

O ModelArts troca dados com o OBS. Você pode armazenar dados no OBS.

- ECS

  Um Elastic Cloud Server (ECS) é uma unidade de computação básica que consiste nas vCPU, memória, sistema operacional e discos do Elastic Volume Service (EVS). Depois que um ECS é criado, você pode usá-lo na nuvem da mesma forma que você usaria seu PC local ou servidor físico.

  Você pode criar uma imagem personalizada no local ou em um ECS.

  NOTA

  Quando você usa uma imagem personalizada, o ModelArts pode precisar acessar serviços dependentes, como SWR e OBS. A imagem personalizada pode ser usada somente após o acesso ser autorizado. É uma boa prática usar uma agência para autorização. Depois que a agência é configurada, as permissões para acessar serviços dependentes são delegadas ao ModelArts para que o ModelArts possa usar os serviços dependentes e executar operações em seu nome. Para obter detalhes, consulte **Configuração da autorização de acesso (configuração global)**.

2 Uso de uma imagem predefinida

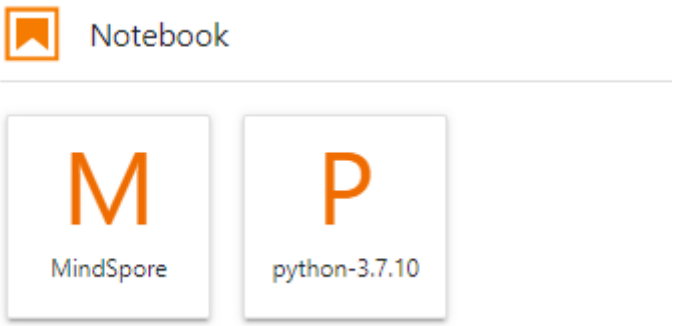
2.1 Imagens predefinidas no notebook

2.1.1 Imagens de base do notebook

Predefinir imagens

As imagens predefinidas no DevEnviron do ModelArts são:

- Pacotes predefinidos típicos: mecanismos de IA baseados no Conda padrão, pacotes de software de análise de dados, como Pandas e Numpy, e software de ferramentas, como CUDA e CUDNN, estão incluídos para atender às suas necessidades.
- Ambientes Conda predefinidos: um ambiente Conda e Conda Python básico (excluindo qualquer mecanismo de IA) são criados para cada imagem predefinida. A figura a seguir mostra o ambiente Conda para uma imagem de MindSpore predefinida.

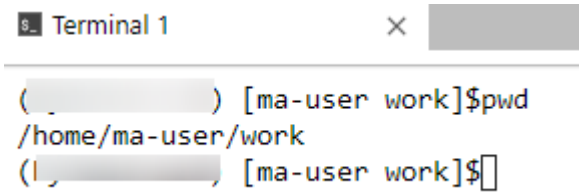


  Selecione um ambiente Conda com base em se o MindSpore é usado para depuração.
- Notebook: uma aplicação da Web que permite codificar na GUI e combinar o código, as equações matemáticas e o conteúdo visualizado em um documento.
- Plug-ins do JupyterLab: permitem mudança de flavor e interrupção de instâncias para melhorar a experiência do usuário. Depois que uma instância de notebook é interrompida, suas CPUs e memória não são mais cobradas.
- SSH remoto: permite que você inicie e depure remotamente uma instância de notebook a partir de um PC local.

- Imagens predefinidas no DevEnviron do ModelArts: depois que essas imagens predefinidas suportam o desenvolvimento da função, as imagens personalizadas criadas com base nessas imagens predefinidas podem ser usadas diretamente para trabalhos de treinamento do ModelArts.

Uma imagem predefinida do ModelArts é iniciada como usuário **ma-user**. O diretório de trabalho padrão de uma instância de notebook acessada é **/home/ma-user/work**.

Crie uma instância e monte o armazenamento persistente para **/home/ma-user/work**. Os dados armazenados somente no diretório **work** são retidos depois que a instância é interrompida e reiniciada. Ao usar um ambiente de desenvolvimento, armazene os dados para persistência em **/home/ma-user/work**.



Criar uma instância de notebook usando uma imagem predefinida

Selecione uma imagem predefinida ao criar uma instância de notebook. Você pode acessar e usar a instância logo após ela ser criada.

1. Faça logon no console de gerenciamento do ModelArts. No painel de navegação à esquerda, escolha **DevEnviron** > **Notebook** para aceder à página da nova versão do **Notebook**.
2. Clique em **Create**. Na página **Create Notebook**, selecione uma imagem pública, configure outros parâmetros e envie a solicitação de criação. Para obter detalhes sobre os parâmetros, consulte **Criação de uma instância de notebook**.
3. Depois que o status da instância do notebook mudar para **Running**, acesse o notebook para usar a imagem criada.

2.1.2 Lista de imagens de base do notebook

O DevEnviron do ModelArts fornece imagens de contêiner do Docker, que podem ser executadas como contêineres predefinidos. Certas imagens predefinidas são construídas em estruturas comuns de mecanismo de IA, como PyTorch, TensorFlow e MindSpore. Essas imagens são nomeadas usando os mecanismos de IA. Além disso, muitos pacotes comuns são predefinidos nessas imagens, aliviando você da instalação do pacote.

Tabela 2-1 Imagens predefinidas x86

Mecanismo de IA	Imagem
PyTorch	pytorch1.8-cuda10.2-cudnn7-ubuntu18.04
	pytorch1.10-cuda10.2-cudnn7-ubuntu18.04
	pytorch1.4-cuda10.1-cudnn7-ubuntu18.04
Tensorflow	tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

Mecanismo de IA	Imagem
	tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04
MindSpore	mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04
	mindspore1.7.0-py3.7-ubuntu18.04
	mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04
	mindspore1.2.0-openmpi2.1.1-ubuntu18.04
Sem mecanismo de IA (imagens de base dedicadas à personalização de imagens)	conda3-cuda10.2-cudnn7-ubuntu18.04
	conda3-ubuntu18.04

2.1.3 Imagem de base do notebook com PyTorch x86

ModelArts fornece as seguintes imagens de base de notebook com PyTorch (x86): **pytorch1.8-cuda10.2-cudnn7-ubuntu18.04**, **pytorch1.10-cuda10.2-cudnn7-ubuntu18.04** e **pytorch1.4-cuda10.1-cudnn7-ubuntu18.04**.

Imagem pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

Tabela 2-2 Descrição de pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
PyTorch 1.8	Sim (cuda 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			torch 1.8.0 torchvision 0.9.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip

Imagem pytorch1.10-cuda10.2-cudnn7-ubuntu18.04

Tabela 2-3 Descrição de pytorch1.10-cuda10.2-cudnn7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
PyTorch 1.10	Sim (cuda 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_10:pytorch_1.10.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221008154718-2b3e39c	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			torch 1.10.2 torchvision 0.11.3 ipykernel 5.3.4 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip

Imagem pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

Tabela 2-4 Descrição de pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
PyTorch 1.4	Sim (cuda 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_4:pytorch_1.4-cuda_10.1-py37-ubuntu_18.04-x86_64-20220926104017-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			torch 1.4.0 torchvision 0.5.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx pandoc python3 rpm screen tar tmux unzip vim wget zip

2.1.4 Imagem de base do notebook com Tensorflow (x86)

O ModelArts fornece as seguintes imagens de base de notebook com Tensorflow (x86): **tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04** e **tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04**

Imagem tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

Tabela 2-5 Descrição de tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
TensorFlow 2.1	Sim (cuda 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926144607-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			tensorflow 2.1.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 opencv-python 4.1.2.30 pandas 1.1.5 pillow 9.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 tensorboard 2.1.1	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

Imagem tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

Tabela 2-6 Descrição de tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
TensorFlow 1.13-gpu	Sim (cuda 10.0)	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_1_13:tensorflow_1.13-cuda_10.0-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			tensorflow-gpu 1.13.1 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.0.1.rc0.ffd1c0c8 numpy 1.17.0 opencv-python 4.1.2.30 pandas 1.1.5 pillow 6.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.2.2 scikit-learn 0.22.1 tornado 6.2	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

2.1.5 Imagem de base do notebook com MindSpore x86

O ModelArts fornece as seguintes imagens de base de notebook MindSpore (x86): **mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04**, **mindspore1.7.0-py3.7-ubuntu18.04**, **mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04** e **mindspore1.2.0-openmpi2.1.1-ubuntu18.04**.

Imagem mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04

Tabela 2-7 Descrição de mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
MindSpore-gpu 1.7.0	Sim (cuda 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			mindspore-gpu 1.7.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.17.0 pandas 1.1.5 pillow 9.1.1 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.1 mindinsight 1.7.0 mindvision 0.1.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

Imagem mindspore1.7.0-py3.7-ubuntu18.04

Tabela 2-8 Descrição de mindspore1.7.0-py3.7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
MindSpore 1.7.0	Nenhum	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220926104017-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			mindspore 1.7.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.6 ma-cau-adapter 1.1.3 ma-cli 1.2.2 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.17.0 pandas 1.1.5 pillow 9.1.1 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.1 mindinsight 1.7.0 mindvision 0.1.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

Imagem mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

Tabela 2-9 Descrição de mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
MindSpore-gpu 1.2.0	Sim (cuda 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-cuda_10.1-ubuntu_18.04-x86_64-20220926104106-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			mindspore-gpu 1.2.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 1.1.5 pillow 6.2.0 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.2.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libcudnn7 libcudnn7-dev libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

Imagem mindspore1.2.0-openmpi2.1.1-ubuntu18.04

Tabela 2-10 Descrição de mindspore1.2.0-openmpi2.1.1-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
MindSpore 1.2.0	Nenhuma	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			mindspore 1.2.0 ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.19.5 pandas 6.2.0 pillow 9.1.1 pip 22.1.2 psutil 5.8.0 PyYAML 5.1 scipy 1.5.2 scikit-learn 0.22.1 tornado 6.2 mindinsight 1.2.0	automake build-essential ca-certificates cmake cpp curl ffmpeg g++ gcc gfortran git git-lfs grep libjpeg-dev:amd64 libjpeg8-dev:amd64 openssh-client openssh-server nginx python3 rpm screen tar tmux unzip vim wget zip

2.1.6 Imagem de base do notebook com imagem dedicada personalizada (x86)

O ModelArts fornece as seguintes imagens de base de notebook com imagens personalizadas (x86): **conda3-cuda10.2-cudnn7-ubuntu18.04** e **conda3-ubuntu18.04**. Essas imagens não têm mecanismos de IA ou pacotes de software relacionados. O tamanho da imagem é de apenas 2 GB a 5 GB. Você pode usar essas imagens como imagens de base e instalar o mecanismo desejado e os pacotes de dependência, melhorando a escalabilidade. Além disso, essas imagens são predefinidas com algumas configurações necessárias para iniciar o ambiente de desenvolvimento. Você pode usar essas imagens depois de instalar os pacotes de software necessários, sem a necessidade de quaisquer adaptações.

Tais imagens são as mais básicas e não possuem nenhum componente instalado. Elas são bastante pequenas para facilitar a personalização da imagem. Se você precisar usar o SDK do OBS, use o SDK do ModelArts para copiar arquivos. Para obter detalhes, consulte **Transferência de arquivos**.

Imagem conda3-cuda10.2-cudnn7-ubuntu18.04

Tabela 2-11 Descrição de conda3-cuda10.2-cudnn7-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
Nenhum	Sim (cuda 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c	Pacote de PyPI	Pacote do Ubuntu

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
			ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.21.6 pandas 1.3.5 pillow 9.2.0 pip 20.3.3 psutil 5.9.1 PyYAML 6.0 scipy 1.7.3 tornado 6.2	automake build-essential ca-certificates cmake cpp curl g++ gcc gfortran grep libcudnn7 libcudnn7-dev nginx python3 rpm tar unzip vim wget zip

Imagem conda3-ubuntu18.04

Tabela 2-12 Descrição de conda3-ubuntu18.04

Mecanismo de IA	Se usar GPUs (Versão de CUDA)	URL	Dependência	
Nenhum	Não	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c Por exemplo: CN North-Beijing4 swr.cn-north-4.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c CN East-Shanghai1 swr.cn-east-3.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c CN South-Guangzhou swr.cn-south-1.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c	Pacote de PyPI	Pacote do Ubuntu
			ipykernel 6.7.0 ipython 7.34.0 jupyter-client 7.3.4 ma-cau 1.1.3 ma-cau-adapter 1.1.3 ma-cli 1.1.5 matplotlib 3.5.1 modelarts 1.4.11 moxing-framework 2.1.0.5d9c87c8 numpy 1.21.6 pandas 1.3.5 pillow 9.2.0 pip 20.3.3 psutil 5.9.1 PyYAML 6.0 scipy 1.7.3 tornado 6.2	automake build-essential ca-certificates cmake cpp curl g++ gcc gfortran grep nginx python3 rpm tar unzip vim wget zip

2.2 Imagem de base de treinamento

2.2.1 Imagens de base de treinamento disponíveis

O ModelArts fornece imagens de base baseadas em aprendizado profundo, como imagens de TensorFlow, PyTorch e MindSpore. Nestas imagens, o software obrigatório para a execução de trabalhos de treinamento foi instalado. Se o software nas imagens de base não puder atender aos seus requisitos de serviço, crie novas imagens com base nas imagens de base e use as novas imagens para criar trabalhos de treinamento.

Imagens de base de treinamento disponíveis

A tabela a seguir lista as imagens de base de treinamento predefinidas do ModelArts.

Tabela 2-13 Imagens de base de treinamento do ModelArts

Mecanismo	Versão
PyTorch	pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
TensorFlow	tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
Horovod	horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64
	horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
MPI	mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

NOTA

Os mecanismos de IA suportados variam dependendo das regiões.

2.2.2 Imagem de base de treinamento (PyTorch)

Esta seção descreve as imagens predefinidas do PyTorch.

Versão do mecanismo: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Endereço da imagem: swr.{region}.myhuaweicloud.com/aip/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-roma-20220309171256-40adcc1
- Tempo de criação da imagem: 20220309171256 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- cuda: 10.2.89
- cudnn: 7.6.5.32
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages
- As versões de alguns pacotes de terceiros:

```
Cython 0.27.3
dask 2022.2.0
```



```
easydict 1.9
enum34 1.1.10
torch 1.8.0
Flask 1.1.1
grpcio 1.44.0
gunicorn 20.1.0
idna 3.3
torchtext 0.5.0
imageio 2.16.0
imgaug 0.4.0
lxml 4.8.0
matplotlib 3.5.1
torchvision 0.9.0
mmcv 1.2.7
numba 0.47.0
numpy 1.21.5
opencv-python 4.1.2.30
toml 0.10.2
pandas 1.1.5
Pillow 9.0.1
pip 21.2.2
protobuf 3.19.4
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 6.0
requests 2.27.1
scikit-image 0.19.2
...
```

- Versões anteriores: nenhuma

2.2.3 Imagem de base de treinamento (TensorFlow)

Esta seção descreve as imagens predefinidas do TensorFlow.

Versão de mecanismo: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Endereço da imagem: swr.{region}.myhuaweicloud.com/aip/tensorflow_2_1:train-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- Tempo de criação da imagem: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages
- As versões de alguns pacotes de terceiros:

```
Cython 0.29.21
dask 2021.9.0
easydict 1.9
enum34 1.1.10
tensorflow 2.1.0
Flask 1.1.1
grpcio 1.40.0
gunicorn 20.1.0
idna 3.2
tensorflow-estimator 2.1.0
imageio 2.9.0
imgaug 0.4.0
lxml 4.6.3
matplotlib 3.4.3
```



```
termcolor 1.1.0
scikit-image 0.18.3
numba 0.47.0
numpy 1.17.0
opencv-python 4.1.2.30
tifffile 2021.8.30
pandas 1.1.5
Pillow 6.2.0
pip 21.0.1
protobuf 3.17.3
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
requests 2.26.0
...
```

- Versões anteriores: nenhuma

2.2.4 Imagem de base de treinamento (Horovod)

Esta seção descreve as imagens predefinidas do Horovod.

Versão do mecanismo 1: horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Endereço da imagem: swr.{region}.myhuaweicloud.com/aip/horovod_tensorflow:train-horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- Tempo de criação da imagem: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/lib/python3.7/site-packages
- As versões de alguns pacotes de terceiros:

```
Cython 0.29.21
dask 2021.9.0
easydict 1.9
enum34 1.1.10
horovod 0.20.0
Flask 1.1.1
grpcio 1.40.0
gunicorn 20.1.0
idna 3.2
tensorboard 2.1.1
imageio 2.9.0
imgaug 0.4.0
lxml 4.6.3
matplotlib 3.4.3
tensorflow-gpu 2.1.0
tensorboardX 2.0
numba 0.47.0
numpy 1.17.0
opencv-python 4.1.2.30
toml 0.10.2
pandas 1.1.5
Pillow 6.2.0
pip 21.0.1
protobuf 3.17.3
```



```
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
requests 2.26.0
scikit-image 0.18.3
...
```

- Versões anteriores: nenhuma

Versão do mecanismo 2: horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Endereço da imagem: swr.{region}.myhuaweicloud.com/aip/horovod_pytorch:train-horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20210912152543-1e0838d
- Tempo de criação da imagem: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- cuda: 11.1.1
- cudnn: 8.0.5.39
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/lib/python3.7/site-packages
- As versões de alguns pacotes de terceiros:

```
Cython 0.27.3
dask 2021.9.0
easydict 1.9
enum34 1.1.10
horovod 0.22.1
Flask 1.1.1
grpcio 1.40.0
gunicorn 20.1.0
idna 3.2
mmcv 1.2.7
imageio 2.9.0
imgaug 0.4.0
lxml 4.6.3
matplotlib 3.4.3
torch 1.8.0
tensorboardX 2.0
numba 0.47.0
numpy 1.17.0
opencv-python 4.1.2.30
torchtext 0.5.0
pandas 1.1.5
Pillow 6.2.0
pip 21.0.1
protobuf 3.17.3
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
requests 2.26.0
scikit-image 0.18.3
torchvision 0.9.0
...
```

- Versões anteriores: nenhuma

2.2.5 Imagem de base de treinamento (MPI)

Esta seção descreve imagens de mindspore_1.3.0 predefinidas.

Versão do mecanismo: mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

- Endereço da imagem: swr.{region}.myhuaweicloud.com/aip/mindspore_1_3_0:train-mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-roma-20211104202338-f258e59
- Tempo de criação da imagem: 20211104202338(yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/lib/python3.7/site-packages
- As versões de alguns pacotes de terceiros:

```
requests 2.26.0
dask 2021.9.0
easydict 1.9
enum34 1.1.10
mindspore-gpu 1.3.0
Flask 1.1.1
grpcio 1.41.1
gunicorn 20.1.0
idna 3.3
PyYAML 5.1
imageio 2.10.1
imgaug 0.4.0
lxml 4.6.4
matplotlib 3.4.2
psutil 5.8.0
scikit-image 0.18.3
numba 0.47.0
numpy 1.17.0
opencv-python 4.5.2.54
tifffile 2021.11.2
pandas 1.1.5
Pillow 8.4.0
pip 21.0.1
protobuf 3.17.3
scikit-learn 0.22.1
...
```

- Versões anteriores: nenhuma

2.2.6 Início do treinamento com uma imagem predefinida

2.2.6.1 PyTorch

O ModelArts fornece várias estruturas de IA para diferentes mecanismos. Quando você usa esses mecanismos para o treinamento do modelo, os comandos de inicialização durante o treinamento precisam ser adaptados de acordo. Esta seção apresenta como fazer adaptações ao mecanismo do PyTorch.

Princípio de inicialização do PyTorch

Especificações e número de nós

Neste caso, **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** é usada como exemplo para descrever como alocar recursos de ModelArts para trabalhos de nó único e distribuídos.

Para um trabalho de nó único (executado em apenas um nó), o ModelArts inicia um contêiner de treinamento que usa exclusivamente os recursos no nó.

Para um trabalho distribuído (executando em mais de um nó), há tantos workers quanto os nós selecionados durante a criação do trabalho. Cada worker é alocado com os recursos de computação da especificação selecionada. Por exemplo, existem **2** nós de computação, dois workers serão iniciados e cada worker possui os recursos de computação de **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB**.

Comunicação de rede

- Para um trabalho de nó único, nenhuma comunicação de rede é necessária.
- Para um trabalho distribuído, as comunicações de rede são necessárias em nós e entre nós.

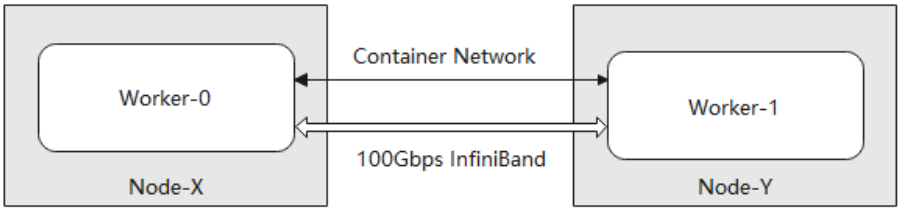
Em nós

NVLink e memória compartilhada são usados para comunicação.

Entre os nós

Se houver mais de um nó de computação, o treinamento distribuído do PyTorch será iniciado. A figura a seguir mostra as comunicações de rede entre workers no treinamento distribuído do PyTorch. Os workers podem se comunicar usando a rede de contêineres e um InfiniBand de 100 Gbit/s ou NIC de RoCE. NICs de RoCE são descritas especificamente para determinadas especificações. Os contêineres podem se comunicar por meio de nomes de domínio do DNS, o que é adequado para comunicação ponto a ponto em pequena escala que exige desempenho médio da rede. As NICs de InfiniBand e RoCE são adequados para trabalhos de treinamento distribuídos usando comunicação coletiva que exigem rede de alto desempenho.

Figura 2-1 Comunicações de rede para treinamento distribuído



Comandos de inicialização

O serviço de treinamento usa o interpretador python padrão na imagem do trabalho para iniciar o script de treinamento. Para obter o interpretador python, execute o comando **which python**. O diretório de trabalho durante a inicialização é **/home/ma-user/user-job-dir/**<*The code directory name*>, que é o diretório retornado executando **pwd** ou **os.getcwd()** em python.

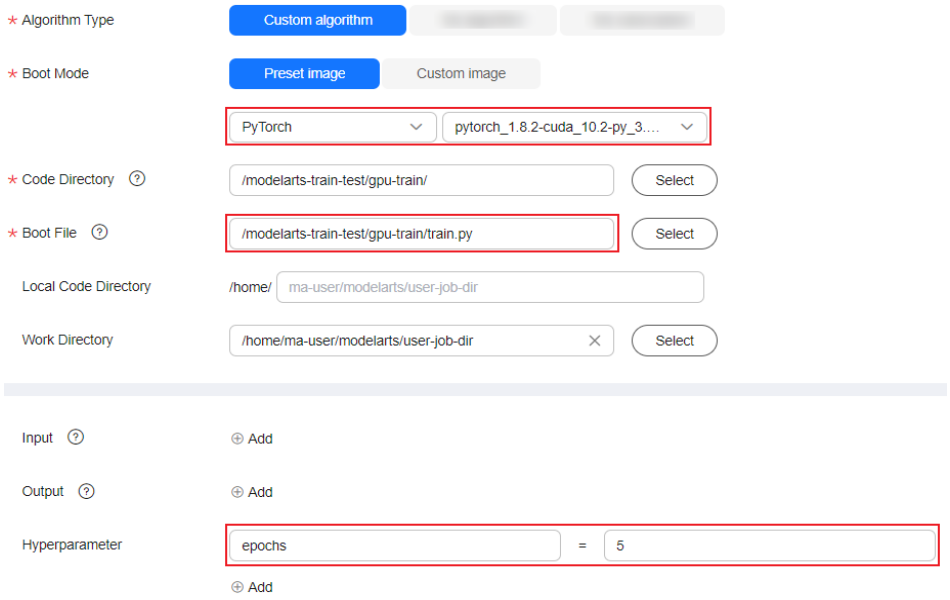
- **Comando de inicialização para nó único de placa única**

  ```
  python <Relative path of the startup file> <Job parameters>
  ```

  – *Relative path of the startup file*: caminho do arquivo de inicialização relativo a **/home/ma-user/user-job-dir/**<*The code directory name*>

– *Job parameters*: parâmetros configurados para um trabalho de treinamento

Figura 2-2 Criar um trabalho de treinamento



Configure os parâmetros consultando a figura acima. Em seguida, execute o seguinte comando no fundo do console:

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --epochs 5
```

- **Comando de inicialização para nó único de multi placa**

  ```
  python <Relative path of the startup file> --init_method "tcp://${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:${port}" <Job parameters>
  ```

  – *Relative path of the startup file*: caminho do arquivo de inicialização relativo a / **home/ma-user/user-job-dir/**<*The code directory name*>

  – *${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}*: nome de domínio do contêiner onde o worker-0 está localizado. Para obter detalhes, consulte **Variáveis de ambiente padrão**.

  – *port*: porta de comunicação padrão do contêiner em que o work-0 está localizado

  – *Job parameters*: parâmetros configurados para um trabalho de treinamento

Figura 2-3 Criar um trabalho de treinamento

Configure os parâmetros consultando a figura acima. Em seguida, execute o seguinte comando no fundo do console:

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --init_method
"tcp://${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:${port}" --epochs 5
```

- **Comando de inicialização para vários nós de multi-placa**

  python *<Relative path of the startup file>* **--init_method "tcp://***${MA_VJ_NAME}-${MA_TASK_NAME}***-0.${MA_VJ_NAME}:$***{port}" --***rank* **<rank_id> --world_size** *<node_num> <Job parameters>*

  – *Relative path of the startup file*: caminho do arquivo de inicialização relativo a / **home/ma-user/user-job-dir/**<*The code directory name*>

  – ${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}: nome de domínio do contêiner onde o worker-0 está localizado. Para obter detalhes, consulte **Variáveis de ambiente padrão**.

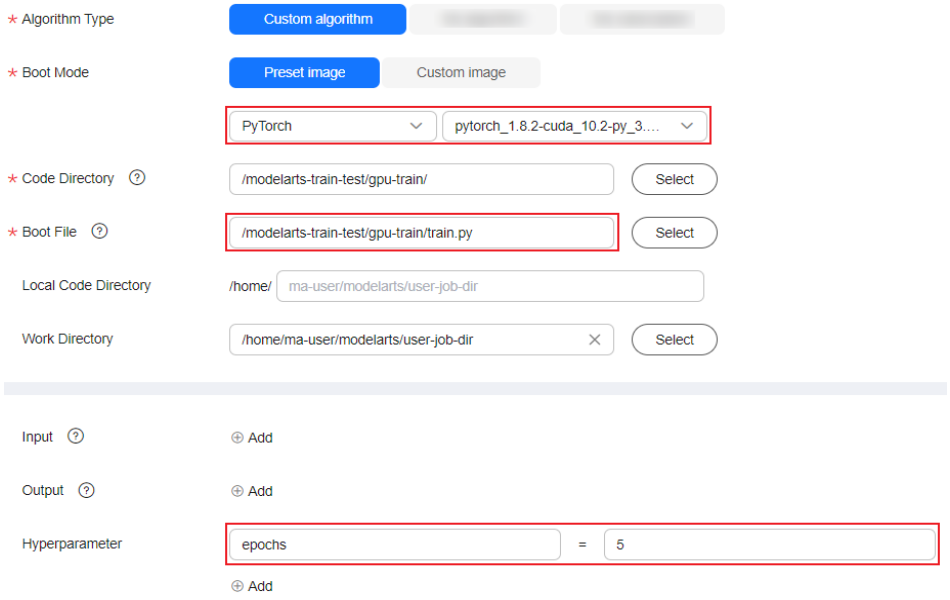
  – *port*: porta de comunicação padrão do contêiner em que o work-0 está localizado

  – *rank*: número de série do worker

  – *node_num*: número de workers

  – *Job parameters*: parâmetros configurados para um trabalho de treinamento

Figura 2-4 Criar um trabalho de treinamento



Configure os parâmetros consultando a figura acima. Em seguida, execute o seguinte comando no fundo do console:

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --init_method
"tcp://${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:${port}" --rank "$
{rank_id}" --world_size "${node_num}" --epochs 5
```

2.2.6.2 TensorFlow

O ModelArts fornece várias estruturas de IA para diferentes mecanismos. Quando você usa esses mecanismos para o treinamento do modelo, os comandos de inicialização durante o treinamento precisam ser adaptados de acordo. Esta seção apresenta como fazer adaptações ao mecanismo do TensorFlow.

Princípio de inicialização do TensorFlow

Especificações e número de nós

Neste caso, **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** é usada como exemplo para descrever como alocar recursos de ModelArts para trabalhos de nó único e distribuídos.

Para um trabalho de nó único (executado em apenas um nó), o ModelArts inicia um contêiner de treinamento que usa exclusivamente os recursos no nó.

Para um trabalho distribuído (executando em mais de um nó), o ModelArts inicia um servidor de parâmetros (PS) e um trabalhador no mesmo nó. O PS possui os recursos de computação da **CPU: 36 cores | Memory: 256 GB** e o worker possui **GPU: 8 xNVIDIA-V100 | CPU: 36 cores | Memory: 256 GB**.

Somente recursos de CPU e memória são alocados para um PS, enquanto um worker também pode possuir placas de aceleração (exceto para especificações de CPU puras). Neste exemplo, cada worker possui oito placas de aceleração NVIDIA V100. Se um PS e um worker forem iniciados no mesmo nó, os recursos de disco serão compartilhados por ambas as partes.

Comunicação de rede

- Para um trabalho de nó único, nenhuma comunicação de rede é necessária.
- Para um trabalho distribuído, as comunicações de rede são necessárias em nós e entre nós.

Em nós

Um PS e um worker podem se comunicar em nós por meio de uma rede de contêineres ou de uma rede de host.

- Uma rede de contêineres é usada quando você executa um trabalho de treinamento em nós usando recursos compartilhados.
- Quando você executa um trabalho de treinamento em nós usando um pool dedicado, a rede host é usada se o nó for configurado com NICs de RoCE, e a rede de contêiner é usada se o nó for configurado com NICs de InfiniBand.

Entre os nós

Para um trabalho distribuído, um PS e um worker podem se comunicar entre nós. O ModelArts fornece NICs de InfiniBand e RoCE com uma largura de banda de até 100 Gbit/s.

Comandos de inicialização

Por padrão, o serviço de treinamento usa o interpretador python na imagem do trabalho para iniciar o script de treinamento. Para obter o interpretador python, execute o comando **which python**. O diretório de trabalho durante a inicialização é **/home/ma-user/user-job-dir/**<*The code directory name*>, que é o diretório retornado executando **pwd** ou **os.getcwd()** em python.

- **Comando de inicialização para nó único de placa única**

  ```
  python <Relative path of the startup file> <Job parameters>
  ```

  – *Relative path of the startup file*: caminho do arquivo de inicialização relativo a **/home/ma-user/user-job-dir/**<*The code directory name*>

  – *Job parameters*: parâmetros de execução configurados para um trabalho de treinamento

Figura 2-5 Criar um trabalho de treinamento

Configure os parâmetros consultando a figura acima. Em seguida, execute o seguinte comando no fundo do console:

```
python /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --epochs 5
```

- **Comando de inicialização para trabalhos distribuídos**

  ```
  python --task_index $VC_TASK_INDEX --PS_hosts $TF_PS_HOSTS --worker_hosts $TF_WORKER_HOSTS --job_name $MA_TASK_NAME <Relative path of the startup file> <Job parameters>
  ```

  – *VC_TASK_INDEX*: número de série da tarefa, por exemplo, **0/1/2**.

  – *TF_PS_HOSTS*: matriz de endereços de nós PS, por exemplo, **[***xx***-PS-0.***xx***:***TCP_PORT***,***xx***-PS-1.***xx***:***TCP_PORT***]**. O valor de **TCP_PORT** é uma porta aleatória que varia de 5.000 a 10.000.

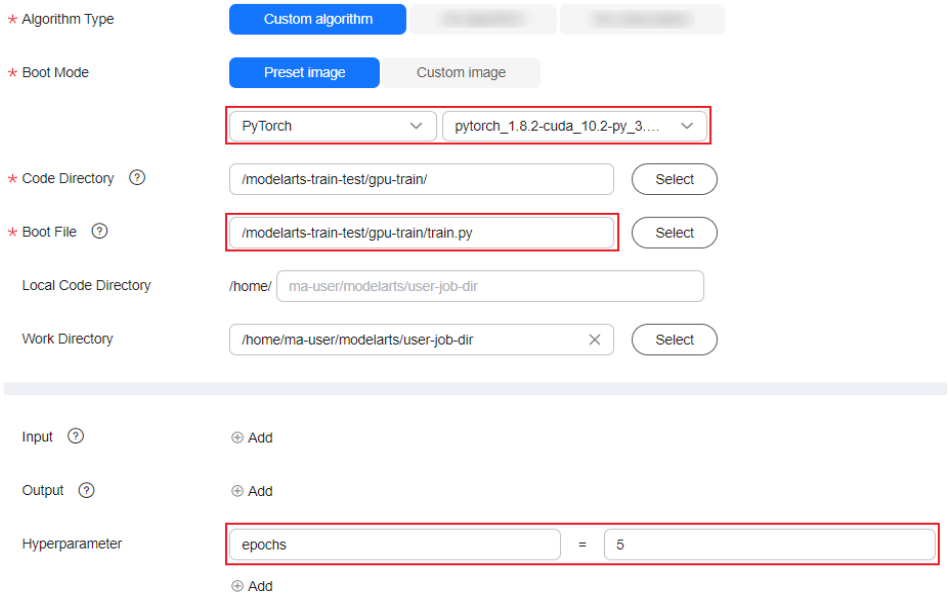
  – *TF_WORKER_HOSTS*: matriz de endereços de nós de trabalho, por exemplo, **[***xx***-worker-0.***xx***:***TCP_PORT***,***xx***-worker-1.***xx***:***TCP_PORT***]**. O valor de **TCP_PORT** é uma porta aleatória que varia de 5.000 a 10.000.

  – *MA_TASK_NAME*: nome da tarefa, que pode ser PS ou work.

  – *Relative path of the startup file*: caminho do arquivo de inicialização relativo a **/home/ma-user/user-job-dir/**<*The code directory name*>

  – *Job parameters*: parâmetros de execução configurados para um trabalho de treinamento

Figura 2-6 Criar um trabalho de treinamento



Configure os parâmetros consultando a figura acima. Em seguida, execute o seguinte comando no fundo do console:

```
python --task_index "$VC_TASK_INDEX" --PS_hosts "$TF_PS_HOSTS" --worker_hosts "$TF_WORKER_HOSTS" --job_name "$MA_TASK_NAME"
 /home/ma-user/modelarts/user-job-dir/gpu-train/train.py --epochs 5
```

2.2.6.3 Horovod/MPI/MindSpore-GPU

O ModelArts fornece várias estruturas de IA para diferentes mecanismos. Quando você usa esses mecanismos para o treinamento do modelo, os códigos do algoritmo durante o treinamento precisam ser adaptados de acordo. Esta seção apresenta como fazer adaptações para o mecanismo Horovod/MPI/MindSpore-GPU.

Princípio de inicialização de Horovod/MPI/MindSpore-GPU

Especificações e número de nós

Neste caso, **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** é usada como exemplo para descrever como alocar recursos de ModelArts para trabalhos de nó único e distribuídos.

Para um trabalho de nó único (executado em apenas um nó), o ModelArts inicia um contêiner de treinamento que usa exclusivamente os recursos no nó.

Para um trabalho distribuído (executando em mais de um nó), há tantos workers quanto os nós selecionados durante a criação do trabalho. Cada worker é alocado com os recursos de computação da especificação selecionada. Por exemplo, se houver **2** nós de computação, dois workers serão iniciados e cada worker possui os recursos de computação de **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB**.

Comunicação de rede

- Para um trabalho de nó único, nenhuma comunicação de rede é necessária.
- Para um trabalho distribuído, as comunicações de rede são necessárias em nós e entre nós.

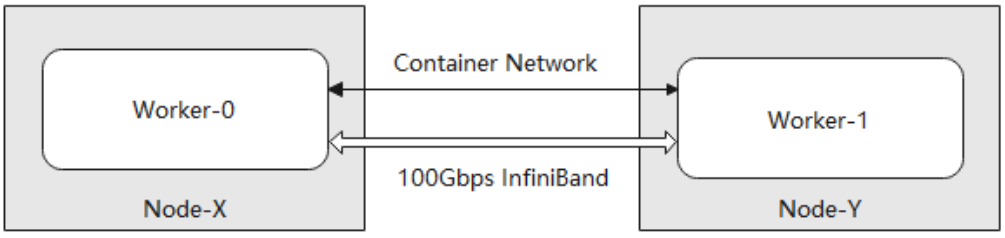
Em nós

NVLink e memória compartilhada são usados para comunicação.

Entre os nós

Se houver mais de um nó de computação, o treinamento distribuído do PyTorch será iniciado. A figura a seguir mostra as comunicações de rede entre workers no treinamento distribuído do PyTorch. Os workers podem se comunicar usando a rede de contêineres e uma NIC de InfiniBand ou de RoCE de 100 Gbit/s. NICs de RoCE são descritas especificamente para determinadas especificações. Os contêineres podem se comunicar por meio de nomes de domínio do DNS, o que é adequado para comunicação ponto a ponto em pequena escala que exige desempenho médio da rede. As NICs de InfiniBand e RoCE são adequados para trabalhos de treinamento distribuídos usando comunicação coletiva que exigem rede de alto desempenho.

Figura 2-7 Comunicações de rede para treinamento distribuído



Comandos de inicialização

Por padrão, o serviço de treinamento usa o interpretador python na imagem do trabalho para iniciar o script de treinamento. Para obter o interpretador python, execute o comando **which python**. O diretório de trabalho durante a inicialização é **/home/ma-user/user-job-dir/**<*The*

code directory name>, que é o diretório retornado executando **pwd** ou **os.getcwd()** em python.

Comandos de inicialização

```
mpirun \
-np ${OPENMPI_NP} \
-hostfile ${OPENMPI_HOST_FILE_PATH} \
-mca plm_rsh_args "-p ${SSHD_PORT}" \
-tune ${TUNE_ENV_FILE} \
${OPENMPI_BIND_ARGS} \
${OPENMPI_X_ARGS} \
${OPENMPI_MCA_ARGS} \
${OPENMPI_EXTRA_ARGS} \
python <Relative path of the startup file> <Job parameters>
```

- *OPENMPI_NP*: número de processos iniciados pelo **mpirun**. O valor padrão é o número de GPUs multiplicado pelo número de nós. Não modifique este valor.
- *OPENMPI_HOST_FILE_PATH*: valor do **hostfile**. Não modifique este valor.
- *SSHD_PORT*: porta para logon SSH. Não modifique este valor.
- *TUNE_ENV_FILE*: variáveis de ambiente de work-0. Transmita as seguintes variáveis de ambiente para outros nós de trabalho do trabalho de treinamento atual.
  - **env** com o prefixo **MA_**
  - **env** com o prefixo **SHARED_**
  - **env** com o prefixo **S3_**
  - **env** de **PATH**
  - **env** com o prefixo **VC_WORKER_**
  - **env** com o prefixo **SCC**
  - **env** com o prefixo **CRED**

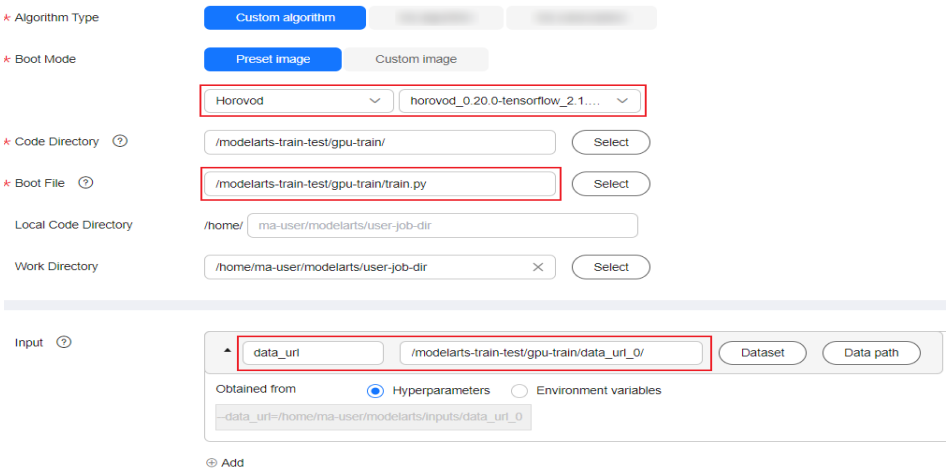
    ```
    env|grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED'|grep -v '=
    $'> ${TUNE_ENV_FILE}
    ```
- *OPENMPI_BIND_ARGS*: processo de pinning com o comando **mpirun cpu**. As configurações padrão são as seguintes:

  ```
  OPENMPI_BIND_ARGS="-bind-to none -map-by slot"
  ```
- *OPENMPI_X_ARGS*: parâmetros **-x** do comando **mpirun**. As configurações padrão são as seguintes:

  ```
  OPENMPI_X_ARGS="-x LD_LIBRARY_PATH -x HOROVOD_MPI_THREADS_DISABLE=1 -x
  NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=ib0,bond0,eth0 -x
  NCCL_SOCKET_FAMILY=AF_INET -x NCCL_IB_DISABLE=0"
  ```
- *OPENMPI_X_ARGS*: parâmetros **-mca** do comando **mpirun**. As configurações padrão são as seguintes:

  ```
  OPENMPI_MCA_ARGS="-mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn
  true"
  ```
- *OPENMPI_EXTRA_ARGS*: parâmetros passados para o **mpirun**. O valor padrão está vazio.
- *Relative path of the startup file*: caminho do arquivo de inicialização relativo a / **home/ma-user/user-job-dir/**<*The code directory name*>
- *Job parameters*: parâmetros em execução de configurados para um trabalho de treinamento

Figura 2-8 Criar um trabalho de treinamento



Configure os parâmetros consultando a figura acima. Em seguida, execute o seguinte comando no fundo do console:

```
mpirun \
-np ${np} \
-hostfile ${OPENMPI_HOST_FILE_PATH} \
-mca plm_rsh_args "-p ${SSHD_PORT}" \
-tune ${TUNE_ENV_FILE} \
${OPENMPI_BIND_ARGS} \
${OPENMPI_X_ARGS} \
${OPENMPI_MCA_ARGS} \
${OPENMPI_EXTRA_ARGS} \
python /home/ma-user/user-job-dir/gpu-train/train.py --datasets=obs://modelarts-train-test/gpu-train/data_url_0
```

NOTA

Se você estiver usando um mecanismo Horovod, MPI ou MindSpore-GPU para treinamento de modelo, os comandos de inicialização para trabalhos de nó único e trabalhos distribuídos serão os mesmos.

2.3 Imagens da base de inferência

2.3.1 Imagens de base de inferência disponíveis

A inferência de ModelArts fornece uma série de imagens de base. Você pode criar imagens personalizadas com base nessas imagens de base para implementar serviços de inferência.

x86 (CPU/GPU)

Tabela 2-14 TensorFlow

Versão do mecanismo de IA	Ambiente de tempo de execução	URI
2.1.0	CPU GPU (CUDA 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817
1.15.5	CPU GPU (CUDA 11.4)	swr.{region_id}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18
2.6.0	CPU GPU (CUDA 11.2)	swr.{region_id}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18

Tabela 2-15 PyTorch

Versão do mecanismo de IA	Ambiente de tempo de execução	URI
1.8.0	CPU GPU (CUDA 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817
1.8.2	CPU GPU (CUDA 11.1)	swr.{region_id}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18

Tabela 2-16 MindSpore

Versão do mecanismo de IA	Ambiente de tempo de execução	URI
1.7.0	CPU	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76

Versão do mecanismo de IA	Ambiente de tempo de execução	URI
1.7.0	GPU (CUDA 10.1)	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
1.7.0	GPU (CUDA 11.1)	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76

2.3.2 Imagens de base de inferência com TensorFlow (CPU/GPU)

O ModelArts fornece as seguintes imagens de base de inferência com TensorFlow (CPU/GPU):

- **Versão de mecanismo 1: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **Versão de mecanismo 2: tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64**
- **Versão de mecanismo 3: tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64**

Versão de mecanismo 1: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Caminho da imagem: swr.{region}.myhuaweicloud.com/atelier/tensorflow_2_1:tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20221121111529-d65d817
- Tempo de criação da imagem: 20220713110657 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages
- Certos pacotes de instalação do pip:

```
Cython                          0.29.21
easydict                        1.9
Flask                           2.0.1
grpcio                          1.47.0
gunicorn                        20.1.0
h5py                            3.7.0
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.1
matplotlib                      3.5.1
moxing-framework                2.1.0.5d9c87c8
numpy                           1.19.5
opencv-python                   4.1.2.30
pandas                          1.1.5
Pillow                          9.2.0
pip                             22.1.2
protobuf                        3.20.1
```



```
psutil                                  5.8.0
PyYAML                                  5.1
requests                                2.27.1
scikit-learn                            0.22.1
scipy                                   1.5.2
sklearn                                 0.0
tensorboard                             2.1.1
tensorboardX                            2.0
tensorflow                              2.1.0
tensorflow-estimator                    2.1.0
wheel                                   0.37.1
zipp                                    3.8.0
...
```

- Certos pacotes de instalação do apt:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
```



```
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

Versão de mecanismo 2: tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64

- Endereço da imagem: swr.{region}.myhuaweicloud.com/aip/tensorflow_1_15:tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64-20220524162601-50d6a18
- Tempo de criação da imagem: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 20.04.4 LTS
- CUDA: 11.4.3
- cuDNN: 8.2.4.15
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/TensorFlow-1.15.5/bin/python, python 3.8.13
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/TensorFlow-1.15.5/lib/python3.8/site-packages
- Certos pacotes de instalação do pip:

```
Cython 0.29.21
psutil 5.9.0
matplotlib 3.5.1
protobuf 3.20.1
tensorflow 1.15.5+nv
Flask 2.0.1
grpcio 1.46.1
gunicorn 20.1.0
Pillow 9.0.1
tensorboard 1.15.0
PyYAML 6.0
pip 22.0.4
lxml 4.7.1
numpy 1.18.5
tensorflow-estimator 1.15.1
...
```

- Certos pacotes de instalação do apt:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
```



```
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

Versão de mecanismo 3: tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64

- Endereço da imagem: swr.{region}.myhuaweicloud.com/aip/tensorflow_2_6:tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- Tempo de criação da imagem: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- CUDA: 11.2.0
- cuDNN: 8.1.1.33
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/lib/python3.7/site-packages
- Certos pacotes de instalação do pip:

```
Cython 0.29.21
requests 2.27.1
easydict 1.9
tensorboardX 2.0
tensorflow 2.6.0
Flask 2.0.1
grpcio 1.46.1
gunicorn 20.1.0
idna 3.3
```



```
tensorflow-estimator 2.9.0
pandas 1.1.5
Pillow 9.0.1
lxml 4.8.0
matplotlib 3.5.1
scikit-learn 0.22.1
psutil 5.8.0
PyYAML 5.1
numpy 1.17.0
opencv-python 4.1.2.30
protobuf 3.20.1
pip 21.2.2
...
```

- Certos pacotes de instalação do apt:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
```



```
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

2.3.3 Imagens de base de inferência com PyTorch (CPU/GPU)

O ModelArts fornece as seguintes imagens de base de inferência com PyTorch (CPU/GPU):

- **Versão do mecanismo 1: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**
- **Versão do mecanismo 2: pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

Versão do mecanismo 1: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Caminho da imagem: swr.{region_id}.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20221118143845-d65d817
- Tempo de criação da imagem: 20220713110657 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- CUDA: 10.2.89
- cuDNN: 7.6.5.32
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages
- Certos pacotes de instalação do pip:

```
Cython                          0.27.3
easydict                        1.9
Flask                           2.0.1
fonttools                       4.34.4
gunicorn                        20.1.0
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.1
matplotlib                      3.5.1
mmcv                            1.2.7
moxing-framework                2.1.0.5d9c87c8
numpy                           1.19.5
opencv-python                   4.1.2.30
pandas                          1.1.5
Pillow                          9.2.0
pip                             22.1.2
protobuf                        3.20.1
psutil                          5.8.0
PyYAML                          5.1
requests                        2.27.1
scikit-learn                    0.22.1
scipy                           1.5.2
sklearn                         0.0
tensorboard                     2.1.1
tensorboardX                    2.0
torch                           1.8.0
torchtext                       0.5.0
torchvision                     0.9.0
tornado                         6.2
tqdm                            4.64.0
traitlets                       5.3.0
```



```
typing_extensions              4.3.0
urllib3                        1.26.10
watchdog                       2.0.0
wcwidth                        0.2.5
Werkzeug                       2.1.2
wheel                          0.37.1
yapf                           0.32.0
zipp                           3.8.0
...
```

- Certos pacotes de instalação do apt:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
```



```
zip
zlib1g-dev
...
```

Versão do mecanismo 2: pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- Caminho da imagem: swr.{region}.myhuaweicloud.com/aip/pytorch_1_8:pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220524162601-50d6a18
- Tempo de criação da imagem: 20220524162601 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/lib/python3.7/site-packages
- Certos pacotes de instalação do pip:

  ```
  Cython 0.27.3
  mmcv 1.2.7
  easydict 1.9
  tensorboardX 2.0
  torch 1.8.2+cu111
  Flask 2.0.1
  pandas 1.1.5
  gunicorn 20.1.0
  PyYAML 5.1
  torchaudio 0.8.2
  Pillow 9.0.1
  psutil 5.8.0
  lxml 4.8.0
  matplotlib 3.5.1
  torchvision 0.9.2+cu111
  pip 21.2.2
  protobuf 3.20.1
  numpy 1.17.0
  opencv-python 4.1.2.30
  scikit-learn 0.22.1
  ...
  ```

- Certos pacotes de instalação do apt:

  ```
  apt
  ca-certificates
  cmake
  cuda
  curl
  ethtool
  fdisk
  ffmpeg
  g++
  gcc
  git
  gpg
  graphviz
  libsm6
  libxext6
  libopencv-dev
  libxrender-dev
  libatlas3-base
  libnuma-dev
  libcap-dev
  libssl-dev
  liblz-dev
  libbz2-dev
  ```



```
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

2.3.4 Imagens de base de inferência com MindSpore (CPU/GPU)

O ModelArts fornece as seguintes imagens de base de inferência com MindSpore (CPU/GPU):

- **Versão de mecanismo: mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64**
- **Versão do mecanismo 2: mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **Versão do mecanismo 3: mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

Versão de mecanismo: mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64

- Caminho da imagem: swr.{region}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Tempo de criação da imagem: 20220702120711 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Certos pacotes de instalação do pip:


```
cycler                          0.11.0
easydict                        1.9
Flask                           2.0.1
grpcio                          1.47.0
gunicorn                        20.1.0
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.0
matplotlib                      3.5.1
mindinsight                     1.7.0
mindspore                       1.7.0
mindvision                      0.1.0
moxing-framework                2.1.0.5d9c87c8
numpy                           1.17.0
opencv-contrib-python-headless  4.6.0.66
opencv-python-headless          4.6.0.66
pandas                          1.1.5
Pillow                          9.1.1
pip                             22.1.2
protobuf                        3.20.1
psutil                          5.8.0
PyYAML                          5.1
requests                        2.27.1
scikit-learn                    0.22.1
scipy                           1.5.2
setuptools                      62.6.0
sklearn                         0.0
tensorboardX                    2.0
threadpoolctl                   3.1.0
tomli                           2.0.1
tornado                         6.1
tqdm                            4.64.0
traitlets                       5.3.0
treelib                         1.6.1
urllib3                         1.26.9
wheel                           0.37.1
zipp                            3.8.0
...
```

- Certos pacotes de instalação do apt:

```
apt
ca-certificates
cmake
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
```



```
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

Versão do mecanismo 2: mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Caminho da imagem: swr.{region}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Tempo de criação da imagem: 20220702120711 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Certos pacotes de instalação do pip:

```
cycler                          0.11.0
easydict                        1.9
Flask                           2.0.1
grpcio                          1.47.0
gunicorn                        20.1.0
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.0
matplotlib                      3.5.1
mindinsight                     1.7.0
mindspore                       1.7.0
mindvision                      0.1.0
moxing-framework                2.1.0.5d9c87c8
numpy                           1.17.0
opencv-contrib-python-headless  4.6.0.66
opencv-python-headless          4.6.0.66
pandas                          1.1.5
```



```
Pillow                          9.1.1
pip                             22.1.2
protobuf                        3.20.1
psutil                          5.8.0
PyYAML                          5.1
requests                        2.27.1
scikit-learn                    0.22.1
scipy                           1.5.2
setuptools                      62.6.0
sklearn                         0.0
tensorboardX                    2.0
threadpoolctl                   3.1.0
tomli                           2.0.1
tornado                         6.1
tqdm                            4.64.0
traitlets                       5.3.0
treelib                         1.6.1
urllib3                         1.26.9
wheel                           0.37.1
zipp                            3.8.0
...
```

- Certos pacotes de instalação do apt:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
```



```
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```

Versão do mecanismo 3: mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- Caminho da imagem: swr.{region}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Tempo de criação da imagem: 20220702120711 (yyyy-mm-dd-hh-mm-ss)
- Versão do sistema de imagem: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39
- Caminho e versão do interpretador Python: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Caminho de instalação do pacote de terceiros: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Certos pacotes de instalação do pip:

```
cycler                          0.11.0
easydict                        1.9
Flask                           2.0.1
grpcio                          1.47.0
gunicorn                        20.1.0
ipykernel                       6.7.0
Jinja2                          3.0.1
lxml                            4.9.0
matplotlib                      3.5.1
mindinsight                     1.7.0
mindspore                       1.7.0
mindvision                      0.1.0
moxing-framework                2.1.0.5d9c87c8
numpy                           1.17.0
opencv-contrib-python-headless  4.6.0.66
opencv-python-headless          4.6.0.66
pandas                          1.1.5
Pillow                          9.1.1
pip                             22.1.2
protobuf                        3.20.1
psutil                          5.8.0
PyYAML                          5.1
requests                        2.27.1
scikit-learn                    0.22.1
scipy                           1.5.2
setuptools                      62.6.0
sklearn                         0.0
tensorboardX                    2.0
threadpoolctl                   3.1.0
tomli                           2.0.1
tornado                         6.1
tqdm                            4.64.0
traitlets                       5.3.0
```



```
treelib                          1.6.1
urllib3                          1.26.9
wheel                            0.37.1
zipp                             3.8.0
...
```

- Certos pacotes de instalação do apt:

```
apt
ca-certificates
cmake
cuda
curl
ethtool
fdisk
ffmpeg
g++
gcc
git
gpg
graphviz
libsm6
libxext6
libopencv-dev
libxrender-dev
libatlas3-base
libnuma-dev
libcap-dev
libssl-dev
liblz-dev
libbz2-dev
liblzma-dev
libboost-graph-dev
libsndfile1
libcurl4-openssl-dev
libopenblas-base
liblapack3
libopenblas-dev
libprotobuf-dev
libleveldb-dev
libsnappy-dev
libhdf5-serial-dev
liblapacke-dev
libgflags-dev
libgoogle-glog-dev
liblmdb-dev
libatlas-base-dev
librdmacm1
libcap2-bin
libpq-dev
mysql-common
net-tools
nginx
openslide-tools
openssh-client
openssh-server
openssh-sftp-server
openssl
protobuf-compiler
redis-server
redis-tools
rpm
tar
tofrodos
unzip
vim
wget
zip
zlib1g-dev
...
```


3 Uso de imagens personalizadas em instâncias de notebook

3.1 Registro de uma imagem no ModelArts

Depois que uma imagem personalizada for criada, registre-a na página **Image Management** do ModelArts antes de usá-la no notebook.

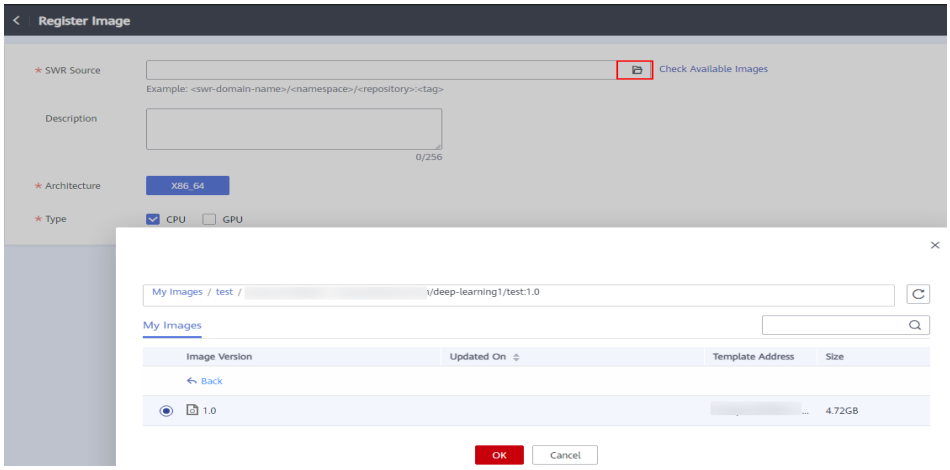
NOTA

Somente os subusuários (usuários do IAM) da conta podem registrar e usar as imagens do SWR se o tipo de imagem for **Private**.

Outros usuários podem registrar e usar imagens do SWR somente quando o tipo de imagem for **Public**.

1. Efetue logon no console de gerenciamento do ModelArts e escolha **Image Management**. Em seguida, clique em **Register**.
2. Configure parâmetros e clique em **Register**.
   - **SWR Source**: selecione uma imagem construída como a fonte da imagem. Você pode copiar o endereço do SWR completo ou clicar em  para selecionar a imagem de destino para registro.

Figura 3-1 Selecionar uma fonte de imagem



– **Architecture** e **Type**: configure-os com base na estrutura real da imagem personalizada.

3. Visualize a imagem registrada na página **Image Management**.

Figura 3-2 Lista de imagens

Criação de uma instância de notebook

Clique no nome da imagem. Na página de detalhes da imagem que aparece, clique em **Create Notebook**. A página para criar uma instância de notebook usando essa imagem é exibida.

Figura 3-3 Página de detalhes da imagem

Sincronizar uma imagem

Depois que a falha da imagem for corrigida, vá para a página de detalhes da imagem. Clique em **Sync** na coluna **Operation** para atualizar o status da imagem.

3.2 Criação de uma imagem personalizada

Você pode criar uma imagem personalizada de qualquer uma das seguintes maneiras:

- Método 1: use uma imagem predefinida de instâncias de notebook para criar uma instância de ambiente de desenvolvimento. Em seguida, instale e configure dependências no ambiente. Após a configuração, use a função de salvar imagem fornecida pelo ambiente de desenvolvimento para salvar a instância em execução como uma imagem de contêiner personalizada. Para mais detalhes, consulte **Salvamento de uma instância de notebook como uma imagem personalizada**.
- Método 2: use imagens de base do ModelArts e modelos de criação de imagens para gravar um Dockerfile e criar sua própria imagem em uma instância de notebook. Em seguida, registre a imagem para criar um novo ambiente de desenvolvimento com base em suas necessidades. Para mais detalhes, consulte **Criação e uso de uma imagem personalizada no notebook**.
- Método 3: use imagens de base do ModelArts ou imagens de terceiros para gravar um Dockerfile em um ECS e reconstruir as imagens de base do Docker ou imagens de terceiros. Isso permite personalizar imagens do Docker que atendam aos **requisitos do ModelArts** e enviar as imagens para o SWR. Para mais detalhes, consulte **Criação de uma imagem personalizada em um ECS e sua utilização no notebook**.

3.3 Salvamento de uma instância de notebook como uma imagem personalizada

3.3.1 Salvamento de uma imagem de ambiente notebook

Para salvar uma imagem de ambiente notebook, faça o seguinte: crie uma instância de notebook usando uma imagem predefinida, instale softwares personalizados e dependências na imagem de base e salve a instância em execução como uma imagem de contêiner.

Na imagem salva, as dependências instaladas são mantidas. Os dados armazenados em **home/ma-user/work** para armazenamento persistente não serão armazenados. Quando você usa o VS Code para desenvolvimento remoto, os plug-ins instalados no servidor são mantidos.

NOTA

As imagens armazenadas em uma instância de notebook não podem ter mais de 35 GB e não pode haver mais de 125 camadas de imagem. Caso contrário, a imagem não poderá ser salva.

Se o erro "The container size (xx) is greater than the threshold (25G)" for relatado quando uma imagem for salva, lidar com o erro consultando **O que fazer se o erro "The container size (xG) is greater than the threshold (25G)" for exibido quando salvar uma imagem?**.

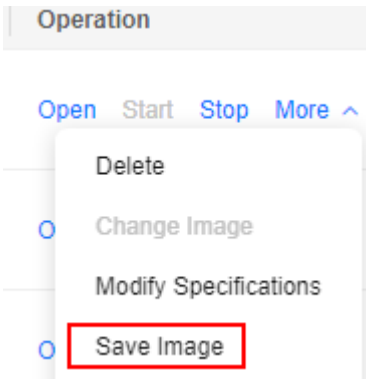
Pré-requisitos

A instância do notebook está no estado **Running**.

Salvar uma imagem

1. Faça logon no console de gerenciamento do ModelArts e escolha **DevEnviron** > **Notebook** no painel de navegação à esquerda para alternar para o notebook da nova versão.
2. Na lista de instâncias do notebook, selecione a instância do notebook de destino e escolha **Save Image** na lista suspensa **More**, na coluna **Operation**. A caixa de diálogo **Save Image** é exibida.

Figura 3-4 Save Image



3. Na caixa de diálogo **Save Image**, configure os parâmetros. Clique em **OK** para salvar a imagem.

   Escolha uma organização na lista suspensa **Organization**. Se nenhuma organização estiver disponível, clique em **Create** à direita para criar uma.

   Os usuários em uma organização podem compartilhar todas as imagens na organização.

4. A imagem será salva como um snapshot e levará cerca de 5 minutos. Durante esse período de tempo, não execute nenhuma operação na instância.

Figura 3-5 Salvar como um snapshot

AVISO

O tempo necessário para salvar uma imagem como um snapshot será contado na duração da execução da instância. Se a duração da execução da instância expirar antes de o snapshot ser salvo, o salvamento da imagem falhará.

5. Depois que a imagem é salva, o status da instância muda para **Running**. Visualize a imagem na página **Image Management**.
6. Clique no nome da imagem para visualizar seus detalhes.

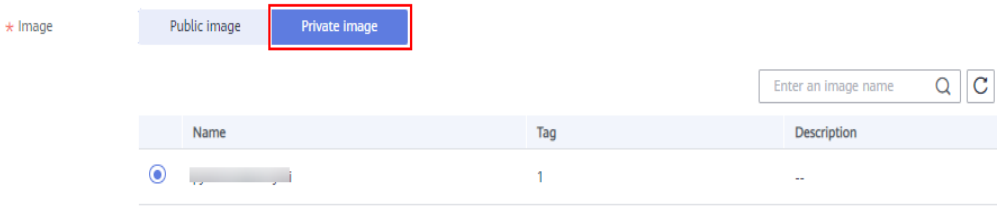
3.3.2 Uso de uma imagem personalizada para criar uma instância de notebook

As imagens salvas de uma instância de notebook podem ser visualizadas na página **Image Management**. Você pode usar essas imagens para criar novas instâncias de notebook, que herdam as configurações de software das instâncias de notebook originais.

Você pode usar um dos seguintes métodos:

Método 1: na página **Create Notebook**, clique em **Private Image** e selecione a imagem salva.

Figura 3-6 Selecionar uma imagem personalizada para criar uma instância de notebook



Método 2: na página **Image Management**, clique na imagem de destino para acessar sua página de detalhes. Em seguida, clique em **Create Notebook**.

3.4 Criação e uso de uma imagem personalizada no notebook

3.4.1 Cenários e processos de aplicação

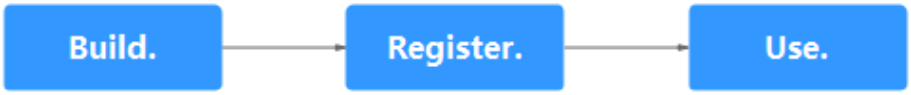
Se as imagens predefinidas não puderem atender aos seus requisitos de serviço, você poderá criar imagens de contêiner com base nas imagens predefinidas para desenvolvimento e treinamento.

Geralmente, você precisará reconstruir o ambiente de desenvolvimento do ModelArts, por exemplo, instalando, atualizando ou desinstalando alguns pacotes. No entanto, a permissão de root é necessária quando certos pacotes são instalados ou atualizados. A instância de notebook em execução não tem a permissão de raiz. Como resultado, você precisa instalar o software que requer a permissão de root na instância do notebook, que está atualmente indisponível no ambiente de desenvolvimento predefinido.

Você precisa gravar um Dockerfile com base em uma imagem pública predefinida para personalizar sua imagem. Em seguida, depure a imagem para que ela possa ser usada no ModelArts. Por fim, registre a imagem no ModelArts para que ela possa ser usada para criar ambientes de desenvolvimento para atender às suas necessidades de serviço.

Este exemplo mostra como usar comandos **ma-cli** na CLI do ModelArts para criar e registrar uma imagem personalizada para desenvolvimento de IA com uma imagem de base do PyTorch. Para obter detalhes, consulte **Comando de criação de imagens ma-cli**. A figura a seguir mostra todo o processo.

Figura 3-7 Criar uma imagem

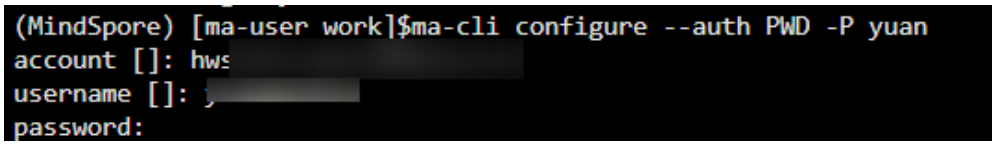


3.4.2 Etapa 1 Criar uma imagem personalizada

Esta seção mostra como criar uma imagem carregando um modelo de criação de imagem e escrevendo um Dockerfile. Certifique-se de ter criado o ambiente de desenvolvimento e aberto um terminal na página **Notebook**. Para obter detalhes sobre Dockerfiles, consulte **Referência do Dockerfile**.

Passo 1 Configure as informações de autenticação, especifique um perfil e digite as informações da conta conforme solicitado. Para obter mais informações sobre autenticação, consulte **Autenticação de ma-cli**.

```
ma-cli configure --auth PWD -P xxx
```



Passo 2 Execute **env|grep -i CURRENT_IMAGE_NAME** para consultar a imagem usada pela instância atual.

```
(PyTorch-1.8) [ma-user work]$env|grep -i CURRENT_IMAGE_NAME
CURRENT_IMAGE_NAME=swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e
```

Passo 3 Crie uma imagem.

1. Obtenha o endereço do SWR da imagem de base.

   CURRENT_IMAGE_NAME=swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

2. Carregue um modelo de criação de imagem.

   Execute o comando **ma-cli image get-template** para consultar o modelo de imagem.

   ```
   (MindSpore) [ma-user work]$ma-cli image get-template
   Template Name                                  Description
   ---------------------------------------------  ------------------------------------------------------------
   upgrade_ascend_mindspore_1.8.1_and_cann_5.1.RC2  Upgrade Ascend MindSpore to 1.8.1 and CANN version to 5.1.RC2
   ```

   ```
   (MindSpore) [ma-user work]$ma-cli image get-template
   Template Name                              Description
   -----------------------------------------  ------------------------------------------------------------------------------------------------------------------------
   customize_from_ubuntu_18.04_to_modelarts   Add ma-user, apt install packages and create a new conda environment with pip based on scratch ubuntu 18.04
   upgrade_current_notebook_apt_packages      Install apt packages like ffmpeg, gcc-8, g++-8 based on current Notebook image
   migrate_3rd_party_image_to_modelarts       General template for migrating your own or open source image to ModelArts
   build_handwritten_number_inference_application  Create a new AI application, used to generate an image to deploy and infer in ModelArts
   update_dli_image_pip_package               Install pip packages based on DLI image
   forward_compat_cuda_11_image_to_modelarts  Migrate and forward compat cuda-11.x to ModelArts by upgrading only user-mode CUDA components
   ```

   Execute o comando **ma-cli image add-template** para carregar o modelo de imagem na pasta especificada. O caminho padrão é onde o comando atual está localizado. Por exemplo, carregue o modelo de criação de imagem **upgrade_current_notebook_apt_packages**.

   ```
   ma-cli image add-template upgrade_current_notebook_apt_packages
   ```

   ```
   (PyTorch-1.8) [ma-user work]$ma-cli image add-template upgrade_current_notebook_apt_packages
   [ OK ] Successfully add configuration template [ upgrade_current_notebook_apt_packages ] under folder [ /home/ma-user/work/.ma/upgrade_current_notebook_apt_packages ]
   ```

3. Modifique um Dockerfile.

   O Dockerfile neste exemplo é modificado com base na imagem de base do PyTorch pytorch1.8-cuda10.2-cudnn7-ubuntu18.04, o modelo de imagem upgrade_current_notebook_apt_packages é carregado, e GCC e G++ são atualizados.

   Depois que o modelo de imagem for carregado, o Dockerfile será carregado automaticamente em **.ma/upgrade_current_notebook_apt_packages**. O conteúdo é o seguinte e você pode modificá-lo com base em suas necessidades.

   ```
   FROM swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64-20220926104358-041ba2e

   # Set proxy to download internet resources
   ENV HTTP_PROXY=http://proxy.modelarts.com:80 \
       http_proxy=http://proxy.modelarts.com:80 \
       HTTPS_PROXY=http://proxy.modelarts.com:80 \
       https_proxy=http://proxy.modelarts.com:80

   USER root

   # Config apt source which can accelerate the apt package download speed. Also install ffmpeg and gcc-8 in root mode
   RUN cp -f /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 update && \
       apt -y install ffmpeg && \
       apt install -y --no-install-recommends gcc-8 g++-8 && apt-get autoremove -y && \
   ```



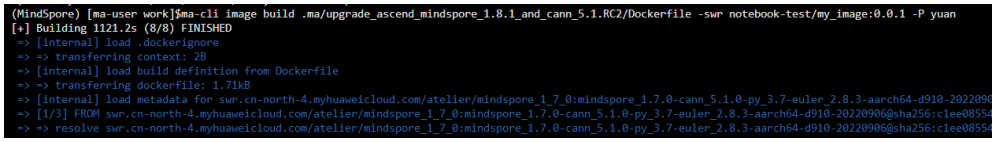
```
    update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --
slave /usr/bin/g++ g++ /usr/bin/g++-8

# ModelArts requires ma-user as the default user to start image
USER ma-user
```

4. Crie uma imagem.

   Execute o comando **ma-cli image build** para criar uma imagem com o Dockerfile. Para obter mais informações, consulte **Criação de uma imagem no notebook do ModelArts**.

   ```
   ma-cli image build .ma/upgrade_current_notebook_apt_packages/Dockerfile -swr
   notebook-test/my_image:0.0.1 -P XXX
   ```

   O Dockerfile é armazenado em **.ma/upgrade_current_notebook_apt_package/Dockerfile** e a nova imagem é armazenada em **notebook-test/my_image:0.0.1** no SWR. **XXX** indica o perfil especificado para autenticação.

----Fim

3.4.3 Etapa 2 Registrar uma nova imagem

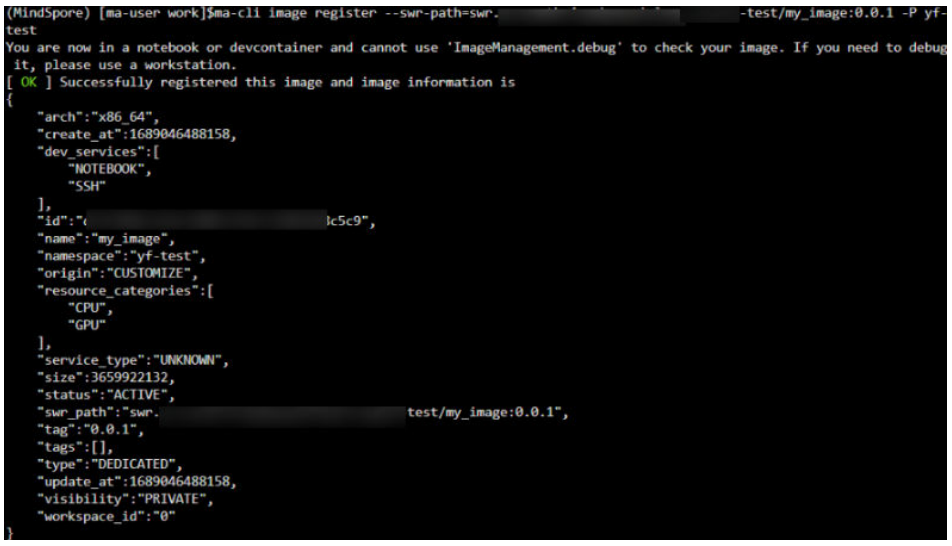
Depois que uma imagem é depurada, registre-a no gerenciamento de imagens do ModelArts para que a imagem possa ser usada no ModelArts.

Use um dos seguintes métodos para registrar a imagem com o ModelArts:

- **Método 1**: execute o comando **ma-cli image register** para registrar uma imagem. Em seguida, as informações da imagem registrada são retornadas, incluindo ID da imagem e nome, conforme mostrado na figura a seguir. Para obter mais informações, consulte **Registro de imagens do SWR com o gerenciamento de imagens do ModelArts**.

  ```
  ma-cli image register --swr-path=swr.ap-southeast-1.myhuaweicloud.com/
  notebook-test/my_image:0.0.1 -P XXX
  ```

  Figura 3-8 Imagem registrada

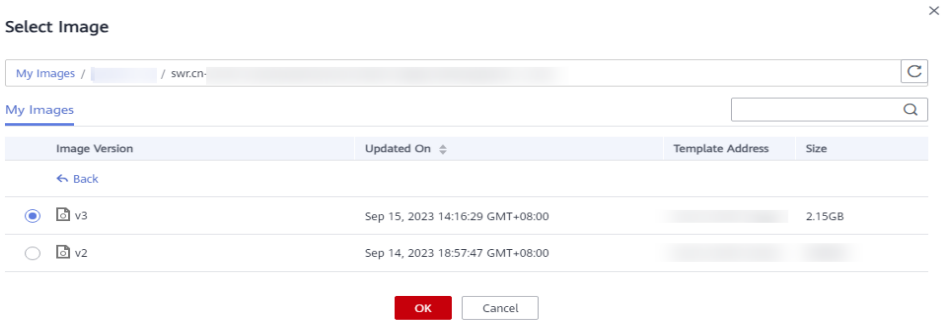
- **Método 2**: registre a imagem no console de gerenciamento do ModelArts.

  Faça logon no console de gerenciamento do ModelArts. No painel de navegação à esquerda, selecione **Image Management**. A página **Image Management** é exibida.

  Clique em **Register**. Paste the complete SWR address, or click  to select a private image from SWR for registration, as shown in **Figura 3-9**.

  Selecione a arquitetura e o tipo com base nos requisitos do site. A arquitetura e o tipo devem ser os mesmos da fonte da imagem.

Figura 3-9 Selecionar uma imagem



3.4.4 Etapa 3 Usar uma nova imagem para criar um ambiente de desenvolvimento

Procedimento

Depois que uma imagem é registrada, ela fica disponível para criação de ambiente de desenvolvimento. Você pode fazer logon no console de gerenciamento do ModelArts, escolher **DevEnviron** > **Notebook** e selecionar a imagem durante a criação.

3.5 Criação de uma imagem personalizada em um ECS e sua utilização no notebook

3.5.1 Cenários e processos de aplicação

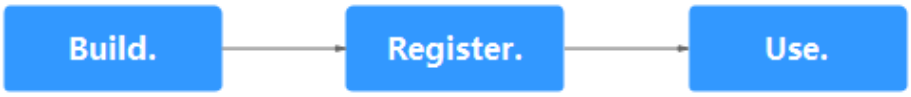
Geralmente, você precisará reconstruir o ambiente de desenvolvimento do ModelArts, por exemplo, instalando, atualizando ou desinstalando alguns pacotes. No entanto, a permissão de root é necessária quando certos pacotes são instalados ou atualizados. A instância de notebook em execução não tem a permissão de raiz. Como resultado, você precisa instalar o software que requer a permissão de root na instância do notebook, que está atualmente indisponível no ambiente de desenvolvimento predefinido.

Você pode gravar um Dockerfile com base em uma imagem base predefinida ou imagem de terceiros para personalizar sua imagem. Em seguida, você pode registrar a imagem para criar um novo ambiente de desenvolvimento com base em suas necessidades.

Esta seção descreve como instalar o PyTorch FFmpeg 3 e GCC 8 em uma imagem do Ubuntu para criar um novo ambiente de desenvolvimento de IA.

A figura a seguir mostra todo o processo.

Figura 3-10 Criar e depurar uma imagem



3.5.2 Etapa 1 Preparar um servidor de Docker e configurar um ambiente

Prepare um servidor com o Docker ativado. Se esse servidor não estiver disponível, crie um ECS, compre um EIP e instale o software necessário nele. A criação, a depuração e o registro subsequentes da imagem são todos executados neste servidor.

O ModelArts fornece scripts do Ubuntu para você instalar o Docker mais facilmente.

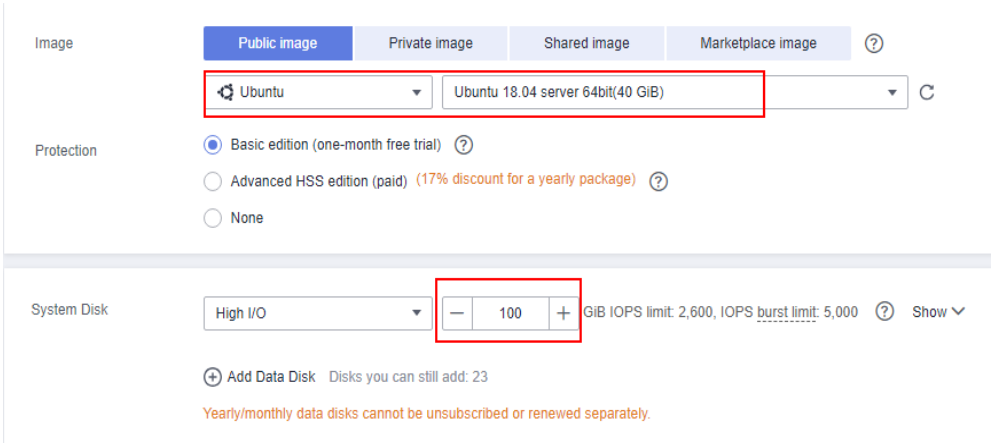
NOTA

As operações no servidor Linux local são as mesmas do ECS. Para mais detalhes, veja este caso.

Criar um ECS

- Efetue logon no console do ECS e clique em **Buy ECS**. Selecione uma imagem pública (uma imagem do Ubuntu 18.04 é recomendada) e defina o disco do sistema para 100 GiB. Para obter mais detalhes, consulte **Compra e logon em um ECS de Linux**.

  Figura 3-11 Selecionar uma imagem e um disco

- Adquira um EIP e vincule-o ao ECS. Para obter detalhes, consulte **Configuração de rede**.

Configurar um ECS

1. Execute o seguinte comando no ECS de Docker para baixar o script de instalação:

   ```
   wget https://cnnorth4-modelarts-sdk.obs.cn-north-4.myhuaweicloud.com/
   modelarts/custom-image-build/install_on_ubuntu1804.sh
   ```


> **NOTA**
>
> Somente scripts do Ubuntu são suportados.

2. Execute o seguinte comando no ECS de Docker para configurar o ambiente:

```
bash install_on_ubuntu1804.sh
```

Figura 3-12 Configurado


```
source /etc/profile
```

O script de instalação é executado para:

a. Instale o Docker.

b. Se o ECS de Docker for executado em GPUs, instale o nvidia-docker2 para montar as GPUs no contêiner do Docker.

3.5.3 Etapa 2 Criar uma imagem personalizada

Esta seção descreve como editar um Dockerfile, usá-lo para criar uma imagem e usar a imagem criada para criar uma instância de notebook. Para obter detalhes sobre como editar um Dockerfile, consulte **Referência do Dockerfile**.

Pré-requisitos

Você preparou um servidor de Docker referindo-se a **Etapa 1 Preparar um servidor de Docker e configurar um ambiente**.

Consultar imagens de base (pular esta etapa para imagens de terceiros)

Para obter detalhes sobre imagens de base do ModelArts, consulte **Lista de imagens de base do notebook**. Verifique o URL da imagem na seção correspondente com base no tipo de mecanismo da imagem predefinida.

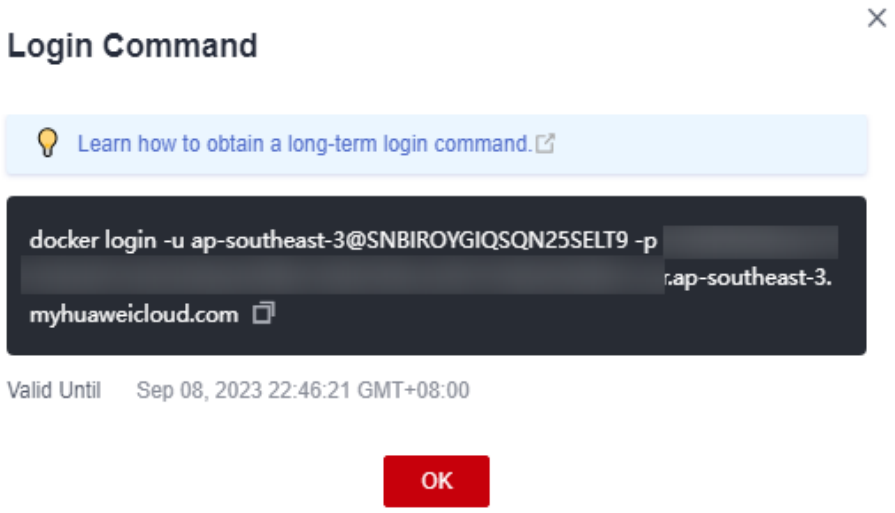
Criar uma imagem

1. Acesse o SWR.

   a. Efetue logon no console do SWR.

   b. No painel de navegação à esquerda, escolha **Dashboard** e clique em **Generate Login Command** no canto superior direito. Na página exibida, copie o comando de logon.

Figura 3-13 Obter o comando de logon



NOTA

- O período de validade do comando de logon gerado é de 24 horas. Para obter um comando de logon válido de longo prazo, consulte **Obtenção de um comando de logon com validade de longo prazo**. Depois de obter um comando de logon válido de longo prazo, seus comandos de logon temporários ainda serão válidos enquanto estiverem em seus períodos de validade.
- O nome de domínio no final do comando de logon é o endereço do repositório de imagens. Registre o endereço para uso posterior.

c. Execute o comando de logon na máquina em que o mecanismo de contêiner está instalado.

A mensagem "Login Succeeded" será exibida após um logon bem-sucedido.

2. Puxe uma imagem de base ou imagem de terceiros. O seguinte usa uma imagem de terceiros como exemplo.

```
docker pull swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/ubuntu:18.04
#Your organization name and image
```

3. Compile um Dockerfile.

   Execute o comando **vim** para criar um Dockerfile. Se uma imagem de base do ModelArts for usada, consulte **Dockerfile em uma imagem de base do ModelArts** para obter detalhes sobre o Dockerfile.

   Se uma imagem de terceiros for usada, adicione o usuário **ma-user** cujo **UID** é **1000** e o grupo de usuários **ma-group** cujo **GID** é **100**. Para mais detalhes, consulte **Dockerfile em uma imagem de base não de ModelArts**.

   Nesse caso, PyTorch 1.8, FFmpeg 3 e GCC 8 serão instalados em uma imagem do Ubuntu para construir uma imagem de IA.

4. Crie uma imagem.

   Execute o comando **docker build** para criar uma nova imagem a partir do Dockerfile. A descrição dos parâmetros do comando é a seguinte:

   – **-t** especifica o novo caminho da imagem, incluindo informações de região, nome da organização, nome da imagem e versão. Defina este parâmetro com base no cenário da vida real. Use um endereço do SWR completo para depuração e registro.

- **-f** especifica o nome do Dockerfile. Defina este parâmetro com base no cenário real.
- **.** no final especifica que o contexto é o diretório atual. Defina este parâmetro com base no cenário real.

```
docker build -t swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/
pytorch_1_8:v1 -f Dockerfile .
```

Figura 3-14 Imagem criada

Dockerfile em uma imagem de base do ModelArts

Execute o comando **vim** para criar um Dockerfile. Se a imagem de base for fornecida pelo ModelArts, o conteúdo do Dockerfile será o seguinte:

```
FROM swr.ap-southeast-1.myhuaweicloud.com/atelier/notebook2.0-pytorch-1.4-kernel-
cp37:3.3.3-release-v1-20220114

USER root
# section1: config apt source
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak && \
    echo -e "deb http://repo.huaweicloud.com/ubuntu/ bionic main restricted\ndeb
http://repo.huaweicloud.com/ubuntu/ bionic-updates main restricted\ndeb http://
repo.huaweicloud.com/ubuntu/ bionic universe\ndeb http://repo.huaweicloud.com/
ubuntu/ bionic-updates universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic
multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates multiverse
\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-backports main restricted
universe multiverse\ndeb http://repo.huaweicloud.com/ubuntu bionic-security main
restricted\ndeb http://repo.huaweicloud.com/ubuntu bionic-security universe\ndeb
http://repo.huaweicloud.com/ubuntu bionic-security multiverse" > /etc/apt/
sources.list && \
    apt-get update
# section2: install ffmpeg and gcc
RUN apt-get -y install ffmpeg && \
    apt -y install gcc-8 g++-8 && \
    update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 80 --
slave /usr/bin/g++ g++ /usr/bin/g++-8 && \
    rm $HOME/.pip/pip.conf
USER ma-user
# section3: configure conda source and pip source
RUN echo -e "channels:\n  - defaults\nshow_channel_urls: true
\ndefault_channels:\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r\n  - https://
mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2\ncustom_channels:\n  conda-
forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  msys2: https://
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  bioconda: https://
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  menpo: https://
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  pytorch: https://
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  pytorch-lts: https://
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud\n  simpleitk: https://
mirrors.tuna.tsinghua.edu.cn/anaconda/cloud" > $HOME/.condarc && \
    echo -e "[global]\nindex-url = https://pypi.tuna.tsinghua.edu.cn/simple
\n[install]\ntrusted-host = https://pypi.tuna.tsinghua.edu.cn" > $HOME/.pip/
pip.conf
# section4: create a conda environment(only support python=3.7) and install
pytorch1.8
RUN source /home/ma-user/anaconda3/bin/activate && \
    conda create -y --name pytorch_1_8 python=3.7 && \
    conda activate pytorch_1_8 && \
    pip install torch==1.8.0 torchvision==0.9.0 torchaudio==0.8.0 && \
    conda deactivate
```


Dockerfile em uma imagem de base não de ModelArts

Se uma imagem de terceiros for usada, adicione o usuário **ma-user** cujo **UID** é **1000** e o grupo de usuários **ma-group** cujo **GID** é **100** ao Dockerfile. Se UID 1000 ou GID 100 na imagem de base tiver sido usado por outro usuário ou grupo de usuários, exclua o usuário ou grupo de usuários. **O usuário e o grupo de usuários foram adicionados ao Dockerfile nesse caso. Você pode usá-los diretamente.**

> NOTA
>
> Você só precisa definir o usuário **ma-user** cujo **UID** é **1000** e o grupo de usuários **ma-group** cujo **GID** é **100** e conceder as permissões de leitura, gravação e execução no diretório de destino ao usuário **ma-user**.

Execute o comando **vim** para criar um Dockerfile e adicione uma imagem de terceiros (não de ModelArts) como a imagem de base, por exemplo, ubuntu 18.04. O conteúdo do Dockerfile é o seguinte:

```
# Replace it with the actual image version.
FROM ubuntu:18.04
# Set the user ma-user whose UID is 1000 and the user group ma-group whose GID is
10
USER root
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid:
1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100
does not exist" && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
    fi && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
    fi && \
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -
s /bin/bash ma-user && \
# Grant the read, write, and execute permissions on the target directory to the
user ma-user.
chmod -R 750 /home/ma-user

#Configure the APT source and install the ZIP and Wget tools (required for
installing conda).
RUN mv /etc/apt/sources.list /etc/apt/sources.list.bak && \
    echo "deb http://repo.huaweicloud.com/ubuntu/ bionic main restricted\ndeb
http://repo.huaweicloud.com/ubuntu/ bionic-updates main restricted\ndeb http://
repo.huaweicloud.com/ubuntu/ bionic universe\ndeb http://repo.huaweicloud.com/
ubuntu/ bionic-updates universe\ndeb http://repo.huaweicloud.com/ubuntu/ bionic
multiverse\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-updates multiverse
\ndeb http://repo.huaweicloud.com/ubuntu/ bionic-backports main restricted
universe multiverse\ndeb http://repo.huaweicloud.com/ubuntu bionic-security main
restricted\ndeb http://repo.huaweicloud.com/ubuntu bionic-security universe\ndeb
http://repo.huaweicloud.com/ubuntu bionic-security multivers e" > /etc/apt/
sources.list && \
apt-get update && \
apt-get install -y zip wget

#Modifying the system Configuration of the image (required for creating the Conda
environment)
RUN rm /bin/sh && ln -s /bin/bash /bin/sh

#Switch to user ma-user , download miniconda from the Tsinghua repository, and
install miniconda in /home/ma-user.
USER ma-user
RUN cd /home/ma-user/ && \
    wget --no-check-certificate https://mirrors.tuna.tsinghua.edu.cn/anaconda/
miniconda/Miniconda3-4.6.14-Linux-x86_64.sh && \
    bash Miniconda3-4.6.14-Linux-x86_64.sh -b -p /home/ma-user/anaconda3 && \
```



```
    rm -rf Miniconda3-4.6.14-Linux-x86_64.sh

#Configure the conda and pip sources
RUN mkdir -p /home/ma-user/.pip && \
    echo -e "channels:\n  - defaults\nshow_channel_urls: true
\ndefault_channels:\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
\n  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r\n  - https://
mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2" > /home/ma-user/.condarc && \
    echo -e "[global]\nindex-url = https://pypi.tuna.tsinghua.edu.cn/simple
\n[install]\ntrusted-host = https://pypi.tuna.tsinghua.edu.cn" > /home/ma-
user/.pip/pip.conf

#Create the conda environment and install the Python third-party package. The
ipykernel package is mandatory for starting a kernel.
RUN source /home/ma-user/anaconda3/bin/activate && \
    conda create -y --name pytorch_1_8 python=3.7 && \
    conda activate pytorch_1_8 && \
    pip install torch==1.8.1 torchvision==0.9.1 && \
    pip install ipykernel==6.7.0 && \
    conda init bash && \
    conda deactivate

#Install FFmpeg and GCC
USER root
RUN apt-get -y install ffmpeg && \
    apt -y install gcc-8 g++-8
```

3.5.4 Etapa 3 Registrar uma nova imagem

Depois que uma imagem é depurada, registre-a no gerenciamento de imagens do ModelArts para que a imagem possa ser usada no ModelArts.

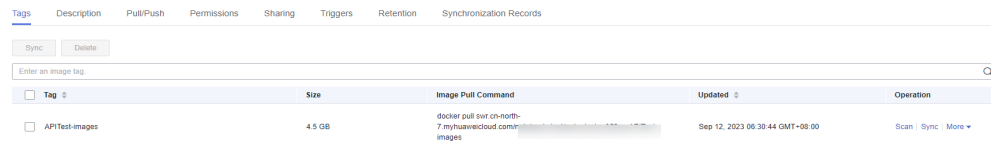
1. Envie a imagem para SWR.

   Faça logon no SWR primeiro. Para obter detalhes, consulte **Logon no SWR**. Execute o seguinte comando para enviar a imagem:

   ```
   docker push swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/pytorch_1_8:v1
   ```

   A imagem fica então disponível no SWR.

   Figura 3-15 Enviar a imagem para SWR

2. Registre uma imagem.

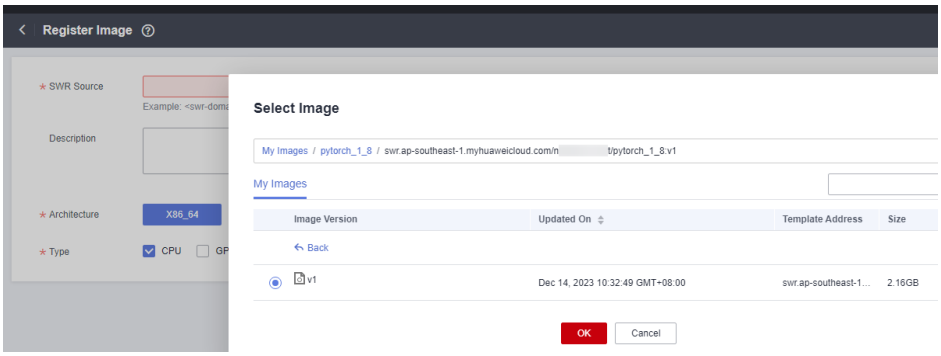
   Registrar uma imagem no console do ModelArts

   Efetue logon no console do ModelArts. No painel de navegação à esquerda, selecione **Image Management** para acessar a página de gerenciamento de imagens. Clique em **Register**. Defina **SWR Source** para a imagem enviada para SWR na **Etapa 1**. Clique em  para selecionar uma imagem existente que você deseja registrar, conforme mostrado em **Figura 3-16**.

   NOTA

   Ao registrar uma nova imagem, certifique-se de que a arquitetura e o tipo sejam iguais aos da origem da imagem. Caso contrário, a criação falha.

Figura 3-16 Registrar uma imagem

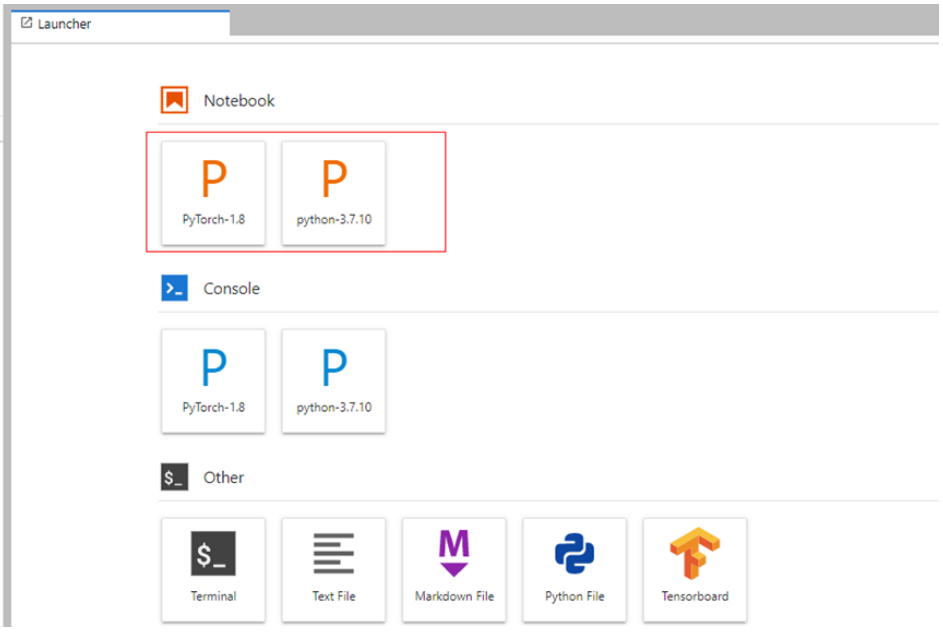


3.5.5 Etapa 5 Criar e iniciar um ambiente de desenvolvimento

Procedimento

1. Depois que a imagem for criada, faça logon no console do ModelArts, vá para a guia de notebook e escolha a imagem registrada em **Etapa 3 Registrar uma nova imagem** para criar um ambiente de desenvolvimento.
2. Vá para a lista do notebook, clique em **Open** para iniciar o ambiente de desenvolvimento criado.

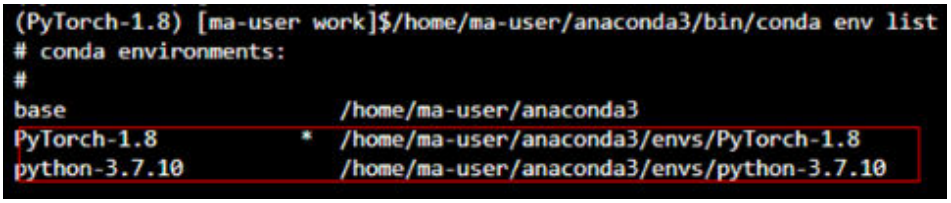
Figura 3-17 Abrir um ambiente de desenvolvimento



3. Abra um terminal para verificar o ambiente da conda. Para mais informações sobre a conda, consulte o **site oficial**.

   Cada kernel no ambiente de desenvolvimento é essencialmente um ambiente conda instalado em **/home/ma-user/anaconda3/**. Execute o comando **/home/ma-user/anaconda3/bin/conda env list** para verificar o ambiente conda.

Figura 3-18 Verificar o ambiente da conda



4 Uso de uma imagem personalizada para treinar modelos (treinamento de modelo)

4.1 Visão geral

Os algoritmos inscritos e as imagens predefinidas podem ser usados na maioria dos cenários de treinamento. Em determinados cenários, o ModelArts permite criar imagens personalizadas para treinar modelos.

A personalização de uma imagem requer uma compreensão profunda dos contêineres. Use esse método somente se os algoritmos inscritos e as imagens predefinidas não puderem atender aos seus requisitos. Imagens personalizadas só podem ser usadas para treinar modelos no ModelArts após serem carregadas no Software Repository for Container (SWR).

Você pode usar imagens personalizadas para treinamento no ModelArts de uma das seguintes maneiras:

- Usar uma imagem predefinida com personalização

  Se você usar uma imagem predefinida para criar um trabalho de treinamento e precisar modificar ou adicionar algumas dependências de software com base na imagem predefinida, poderá personalizar a imagem predefinida. Nesse caso, selecione uma imagem predefinida e escolha **Customize** na caixa de listagem suspensa da versão da estrutura.
- Usar uma imagem personalizada

  Você pode criar uma imagem com base nas especificações de imagem do ModelArts, selecionar sua própria imagem e configurar o diretório de código (opcional) e o comando de inicialização para criar um trabalho de treinamento.

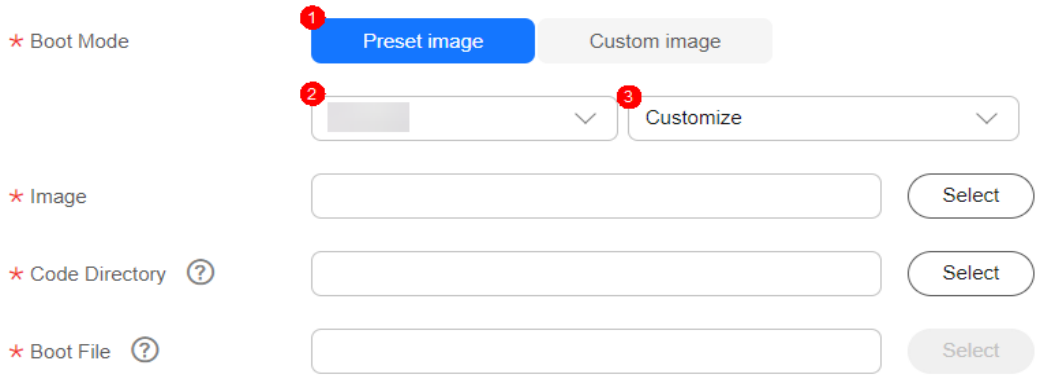
  NOTA

  Quando você usa uma imagem personalizada para criar um trabalho de treinamento, o comando de inicialização deve ser executado no diretório **/home/ma-user**. Caso contrário, o trabalho de treinamento pode ser executado de forma anormal.

Usar uma imagem predefinida com personalização

A única diferença entre este método e a criação de um trabalho de treinamento totalmente baseado em uma imagem predefinida é que você deve selecionar uma imagem. Você pode criar uma imagem personalizada com base em uma imagem predefinida.

Figura 4-1 Criar um algoritmo usando uma imagem predefinida com personalização

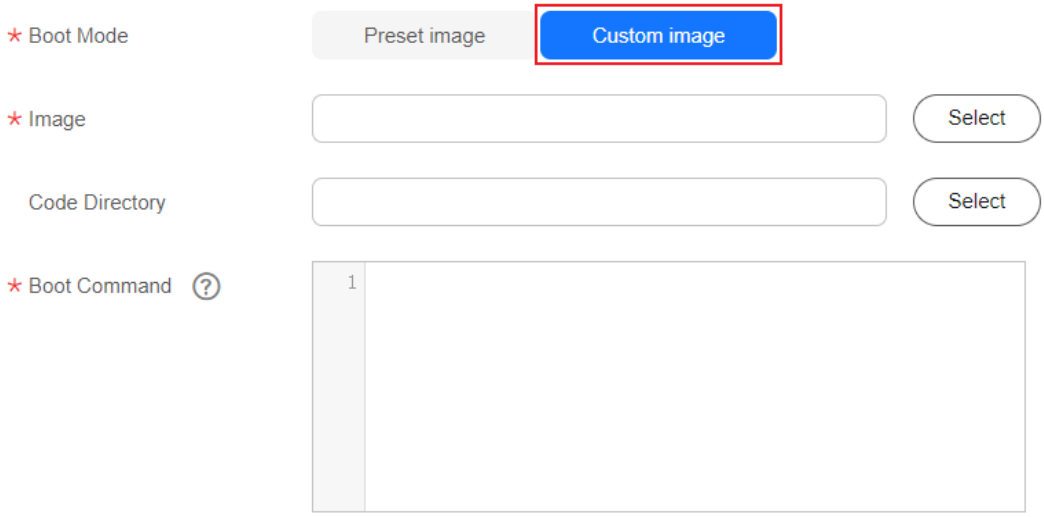


O processo deste método é o mesmo que o de criar um trabalho de treinamento com base em uma imagem predefinida. Por exemplo:

- O sistema injeta automaticamente variáveis de ambiente.
  - PATH=${MA_HOME}/anaconda/bin:${PATH}
  - LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
  - PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
- O arquivo de inicialização selecionado será iniciado automaticamente usando comandos de Python. Certifique-se de que o ambiente Python está correto. A variável de ambiente PATH é injetada automaticamente. Execute os seguintes comandos para verificar a versão do Python para o trabalho de treinamento:
  - export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
  - docker run --rm {image} $(which python) -V
- O sistema adiciona automaticamente hiperparâmetros associados à imagem predefinida.

Usar uma imagem personalizada

Figura 4-2 Criação de um algoritmo usando uma imagem personalizada



Para obter detalhes sobre como usar imagens personalizadas suportadas pelo treinamento, consulte **Uso de uma imagem personalizada para criar um trabalho de treinamento baseado em CPU ou GPU**.

Se todas as imagens usadas forem personalizadas, faça o seguinte para usar um ambiente Conda especificado para iniciar o treinamento:

As tarefas de treinamento não são executadas em um shell. Portanto, você não tem permissão para executar o comando **conda activate** para ativar um ambiente Conda especificado. Neste caso, use outros métodos para começar a treinar.

Por exemplo, Conda em sua imagem personalizada é instalado no diretório **/home/ma-user/anaconda3**, o ambiente Conda é **python-3.7.10** e o script de treinamento é armazenado em **/home/ma-user/modelarts/user-job-dir/code/train.py**. Use um ambiente Conda especificado para iniciar o treinamento de uma das seguintes maneiras:

- Método 1: configure as variáveis de ambiente **DEFAULT_CONDA_ENV_NAME** e **ANACONDA_DIR** corretas para a imagem.

  Execute o comando **python** para iniciar o script de treinamento. O seguinte mostra um exemplo:

  ```
  python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```

- Método 2: use o caminho absoluto do Python do ambiente Conda.

  Execute o comando **/home/ma-user/anaconda3/envs/python-3.7.10/bin/python** para iniciar o script de treinamento. O seguinte mostra um exemplo:

  ```
  /home/ma-user/anaconda3/envs/python-3.7.10/bin/python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```

- Método 3: configure a variável de ambiente path.

  Configure o diretório bin do ambiente Conda especificado na variável de ambiente path. Execute o comando **python** para iniciar o script de treinamento. O seguinte mostra um exemplo:

  ```
  export PATH=/home/ma-user/anaconda3/envs/python-3.7.10/bin:$PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```


- Método 4: execute o comando **conda run -n**.

  Execute o comando **/home/ma-user/anaconda3/bin/conda run -n python-3.7.10** para executar o treinamento. O seguinte mostra um exemplo:

  ```
  /home/ma-user/anaconda3/bin/conda run -n python-3.7.10 python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```

  NOTA

  Se houver um erro indicando que o arquivo .so não está disponível no diretório **$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib**, adicione o diretório ao **LD_LIBRARY_PATH** e coloque o seguinte comando antes do comando de inicialização anterior:

  ```
  export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH;
  ```

  Por exemplo, o exemplo de comando de inicialização usado no método 1 é o seguinte:

  ```
  export LD_LIBRARY_PATH=$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib:$LD_LIBRARY_PATH; python /home/ma-user/modelarts/user-job-dir/code/train.py
  ```

4.2 Exemplo: criar uma imagem personalizada para treinamento

4.2.1 Exemplo: criar uma imagem personalizada para treinamento (PyTorch + CPU/GPU)

Esta seção descreve como criar uma imagem e usá-la para treinamento na plataforma ModelArts. O mecanismo de IA usado para o treinamento é o PyTorch e os recursos são CPUs ou GPUs.

NOTA

Esta seção se aplica somente aos trabalhos de treinamento da nova versão.

Cenários

Neste exemplo, crie uma imagem personalizada gravando um Dockerfile em um host de Linux x86_64 executando o sistema operacional Ubuntu 18.04.

Objetivo: crie e instale imagens de contêiner dos seguintes softwares e use as imagens e CPUs/GPUs para treinamento no ModelArts.

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

Procedimento

Antes de usar uma imagem personalizada para criar um trabalho de treinamento, você precisa estar familiarizado com o Docker e ter experiência em desenvolvimento. O seguinte é o procedimento detalhado:

1. **Pré-requisitos**

2. **Etapa 1 Criar um bucket e uma pasta do OBS**
3. **Etapa 2 Preparar o script de treinamento e carregá-lo no OBS**
4. **Etapa 3 Preparar um host**
5. **Etapa 4 Criar uma imagem personalizada**
6. **Etapa 5 Carregar a imagem para o SWR**
7. **Etapa 6 Criar um trabalho de treinamento no ModelArts**

Pré-requisitos

Você registrou um ID da Huawei e ativou os serviços da Huawei Cloud, e a conta não está em atraso ou congelada.

Etapa 1 Criar um bucket e uma pasta do OBS

Crie um bucket e pastas no OBS para armazenar o conjunto de dados de amostra e o código de treinamento. **Tabela 4-1** lista as pastas a serem criadas. Substitua o nome do bucket e os nomes da pasta no exemplo por nomes reais.

Para obter detalhes sobre como criar um bucket e uma pasta do OBS, consulte **Criação de um bucket** e **Criação de uma pasta**.

Verifique se o diretório do OBS que você usa e o ModelArts estão na mesma região.

Tabela 4-1 Pasta a criar

Nome	Descrição
obs://test-modelarts/pytorch/demo-code/	Armazena o script de treinamento.
obs://test-modelarts/pytorch/log/	Armazene arquivos de log de treinamento.

Etapa 2 Preparar o script de treinamento e carregá-lo no OBS

Prepare o script de treinamento **pytorch-verification.py** e faça o upload para a pasta **obs://test-modelarts/pytorch/demo-code/** do bucket do OBS.

O arquivo **pytorch-verification.py** contém as seguintes informações:

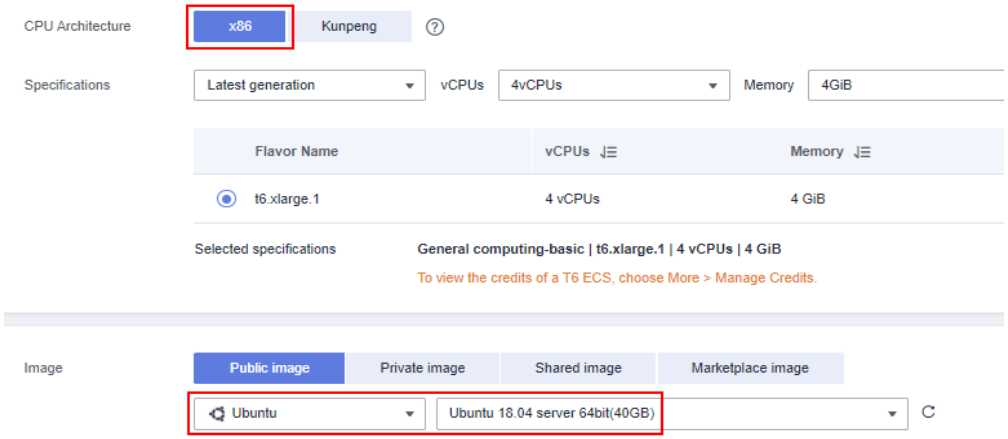
```
import torch
import torch.nn as nn

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)
```

Etapa 3 Preparar um host

Obtenha um servidor Linux x86_64 executando o Ubuntu 18.04. Um ECS ou seu PC local servirão.

Para obter detalhes sobre como comprar um ECS, consulte **Compra e logon em um ECS Linux**. Selecione uma imagem pública. Uma imagem do Ubuntu 18.04 é recomendada.

Figura 4-3 Criar um ECS usando uma imagem pública (x86)



Etapa 4 Criar uma imagem personalizada

Crie uma imagem de contêiner com as seguintes configurações e use a imagem para criar um trabalho de treinamento no ModelArts:

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

Esta seção descreve como gravar um Dockerfile para criar uma imagem personalizada.

1. Instale o Docker.

   O seguinte usa o sistema operacional Linux x86_64 como um exemplo para descrever como obter um pacote de instalação do Docker. Para obter mais detalhes sobre como instalar o Docker, consulte os **documentos oficiais do Docker**.

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

   Se o comando **docker images** é executado, o Docker foi instalado. Nesse caso, pule essa etapa.

2. Execute o seguinte comando para verificar a versão do mecanismo de Docker:

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

   As seguintes informações são exibidas:

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

   NOTA

   Use o mecanismo de Docker da versão anterior ou posterior para criar uma imagem personalizada.

3. Crie uma pasta chamada **context**.

   ```
   mkdir -p context
   ```

4. Obtenha o arquivo **pip.conf**. Neste exemplo, a fonte pip fornecida pelo Huawei Mirrors é usada, que é a seguinte:


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

NOTA

No Huawei Mirrors **https://mirrors.huaweicloud.com/home**, procure **pypi** para obter o arquivo pip.conf.

5. Baixe os seguintes arquivos .whl de: https://download.pytorch.org/whl/torch_stable.html:
   - 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

   NOTA

   O código de URL do símbolo + é %2B. Ao procurar um arquivo no site acima, substitua o símbolo + no nome do arquivo por %2B.

   Por exemplo, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

6. Baixe o arquivo de instalação Miniconda3-py37_4.12.0-Linux-x86_64.sh (Python 3.7.13) de **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.
7. Armazene o arquivo de origem pip, o arquivo torch*.whl e o arquivo de instalação do Miniconda3 na pasta **context**, que é a seguinte:

   ```
   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. Grave a imagem de contêiner de Dockerfile.

   Crie um arquivo vazio chamado **Dockerfile** na pasta **context** e copie o seguinte conteúdo para o arquivo:

   ```
   # The host must be connected to the public network for creating a container
   image.

   # Base container image at 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

   # The default user of the base container image is root.
   # USER root

   # Use the PyPI configuration provided by Huawei Mirrors.
   RUN mkdir -p /root/.pip/
   COPY pip.conf /root/.pip/pip.conf

   # Copy the installation files to the /tmp directory in the base container
   image.
   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
   ```



```
# Install Miniconda3 to the /home/ma-user/miniconda3 directory of the base
container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/
miniconda3

# Install torch*.whl using the default Miniconda3 Python environment in /
home/ma-user/miniconda3/bin/pip.
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

# Create the final container image.
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# Install vim and cURL in Huawei Mirrors.
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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 of the base container image exists. User ma-
user can directly use it.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the
directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-
user/miniconda3

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to avoid log loss.
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# Set the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user
```

Para obter detalhes sobre como gravar um Dockerfile, consulte os **documentos oficiais do Docker**.

9. Verifique se o Dockerfile foi criado. O seguinte mostra a pasta **context**:

```
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. Crie a imagem do contêiner. Execute o comando a seguir no diretório em que o Dockerfile está armazenado para criar a imagem de contêiner **pytorch:1.8.1-cuda11.1**:

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

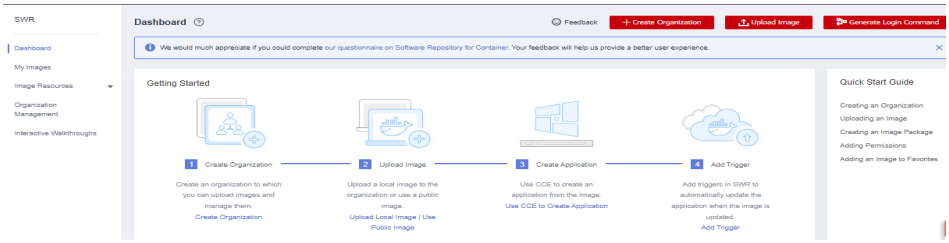
As seguintes informações de log exibidas durante a criação da imagem indicam que a imagem foi criada.

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

Etapa 5 Carregar a imagem para o SWR

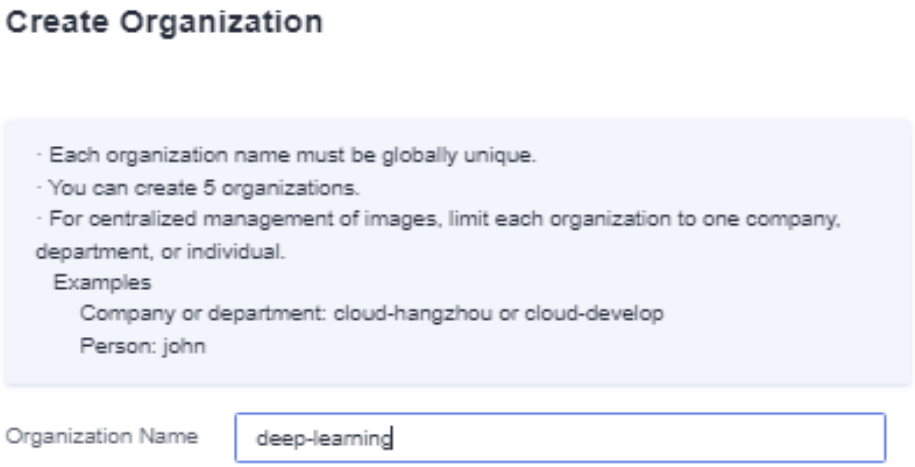
1. Faça logon no console do SWR e selecione a região de destino.

Figura 4-4 Console do SWR



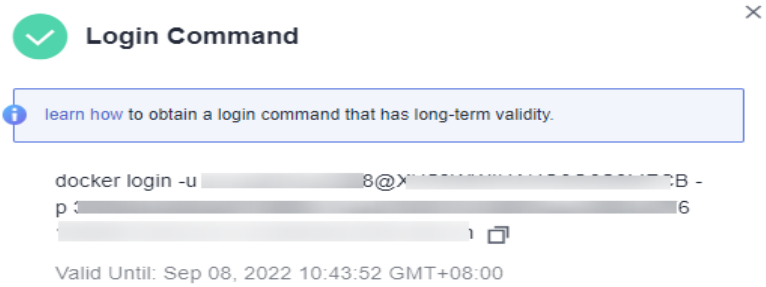
2. Clique em **Create Organization** no canto superior direito e insira um nome de organização para criar uma organização. Personalize o nome da organização. Substitua o nome da organização **deep-learning** nos comandos subsequentes pelo nome real da organização.

Figura 4-5 Criar uma organização



3. Clique em **Generate Login Command** no canto superior direito para obter um comando de logon.

Figura 4-6 Comando de logon



4. Efetue logon no ambiente local como o usuário **root** e digite o comando logon.
5. Carregue a imagem para o SWR.
   a. Execute o seguinte comando para marcar a imagem carregada:

```
#Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
```

   b. Execute o seguinte comando para carregar a imagem:


```
#Replace the region and domain information with the actual values, and
replace the organization name deep-learning with your custom value.
sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-
cuda11.1
```

6. Depois que a imagem for carregada, escolha **My Images** no painel de navegação à esquerda do console do SWR para exibir as imagens personalizadas carregadas.

Etapa 6 Criar um trabalho de treinamento no ModelArts

1. Faça logon no console de gerenciamento do ModelArts e verifique se a autorização de acesso foi configurada para sua conta. Para obter detalhes, consulte **Configuração da autorização da agência**. Se você tiver sido autorizado usando chaves de acesso, limpe a autorização e configure a autorização da agência.
2. No painel de navegação, escolha **Training Management** > **Training Jobs**. A lista de trabalhos de treinamento é exibida por padrão.
3. Na página **Create Training Job**, defina os parâmetros necessários e clique em **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: imagem criada em **Etapa 5 Carregar a imagem para o SWR**.
   - **Code Directory**: diretório onde o arquivo de script de inicialização é armazenado no OBS, por exemplo, **obs://test-modelarts/pytorch/demo-code/**. O código de treinamento é baixado automaticamente para o diretório **${MA_JOB_DIR}/demo-code** do contêiner de treinamento. **demo-code** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Boot Command**: **/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py**. **demo-code** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Resource Pool**: **Public resource pools**
   - **Resource Type**: selecione **CPU** ou **GPU**.
   - **Persistent Log Saving**: ativado
   - **Job Log Path**: defina este parâmetro como o caminho do OBS para armazenar logs de treinamento, por exemplo, **obs://test-modelarts/pytorch/log/**.
4. Verifique os parâmetros do trabalho de treinamento e clique em **Submit**.
5. Aguarde até que o trabalho de treinamento seja concluído.

   Depois que um trabalho de treinamento é criado, as operações como baixa de imagem de contêiner, baixa de diretório de código e execução de comandos de inicialização são executadas automaticamente no back-end. Geralmente, a duração do treinamento varia de dezenas de minutos a várias horas, dependendo do procedimento de treinamento e dos recursos selecionados. Depois que o trabalho de treinamento é executado, o log semelhante ao seguinte é emitido.

Figura 4-7 Executar logs de trabalhos de treinamento com especificações de GPU

```
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')

```

Figura 4-8 Executar logs de trabalhos de treinamento com especificações de CPU

```
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]])

```

4.2.2 Exemplo: criar uma imagem personalizada para treinamento (MPI + CPU/GPU)

Esta seção descreve como criar uma imagem e usá-la para treinamento na plataforma ModelArts. O mecanismo de IA usado para o treinamento é MPI e os recursos são CPUs ou GPUs.

NOTA

Esta seção se aplica somente aos trabalhos de treinamento da nova versão.

Cenários

Neste exemplo, crie uma imagem personalizada gravando um Dockerfile em um host de Linux x86_64 executando o sistema operacional Ubuntu 18.04.

Objetivo: crie e instale imagens de contêiner dos seguintes softwares e use as imagens e CPUs/GPUs para treinamento no ModelArts.

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

Procedimento

Antes de usar uma imagem personalizada para criar um trabalho de treinamento, familiarize-se com o Docker e tenha experiência em desenvolvimento. O seguinte é o procedimento detalhado:

1. **Pré-requisitos**
2. **Etapa 1 Criar um bucket e uma pasta do OBS**
3. **Etapa 2 Preparar os arquivos de script e carregá-los para o OBS**
4. **Etapa 3 Preparar um servidor de imagens**
5. **Etapa 4 Criar uma imagem personalizada**
6. **Etapa 5 Carregar a imagem para o SWR**
7. **Etapa 6 Criar um trabalho de treinamento no ModelArts**

Pré-requisitos

Você registrou um ID da Huawei e ativou os serviços da Huawei Cloud, e a conta não está em atraso ou congelada.

Etapa 1 Criar um bucket e uma pasta do OBS

Crie um bucket e pastas no OBS para armazenar o conjunto de dados de amostra e o código de treinamento. **Tabela 4-2** lista as pastas a serem criadas. Substitua o nome do bucket e os nomes da pasta no exemplo por nomes reais.

Para obter detalhes sobre como criar um bucket e uma pasta do OBS, consulte **Criação de um bucket** e **Criação de uma pasta**.

Verifique se o diretório do OBS que você usa e o ModelArts estão na mesma região.

Tabela 4-2 Pasta a criar

Nome	Descrição
obs://test-modelarts/mpi/demo-code/	Armazena o script de inicialização de MPI e o arquivo de script de treinamento.
obs://test-modelarts/mpi/log/	Armazene arquivos de log de treinamento.

Etapa 2 Preparar os arquivos de script e carregá-los para o OBS

Prepare o script de inicialização do MPI **run_mpi.sh** e o script de treinamento **mpi-verification.py** e carregue-os na pasta **obs://test-modelarts/mpi/demo-code/** do bucket do OBS.

- O conteúdo do script de inicialização do MPI **run_mpi.sh** é o seguinte:

```
#!/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")

    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
```

NOTA

O script **run_mpi.sh** usa terminações de linha LF. Se as terminações de linha CRLF forem usadas, a execução do trabalho de treinamento falhará e o erro "$'\r': command not found" será exibido nos logs.

- O conteúdo do script de treinamento **mpi-verification.py** é o seguinte:

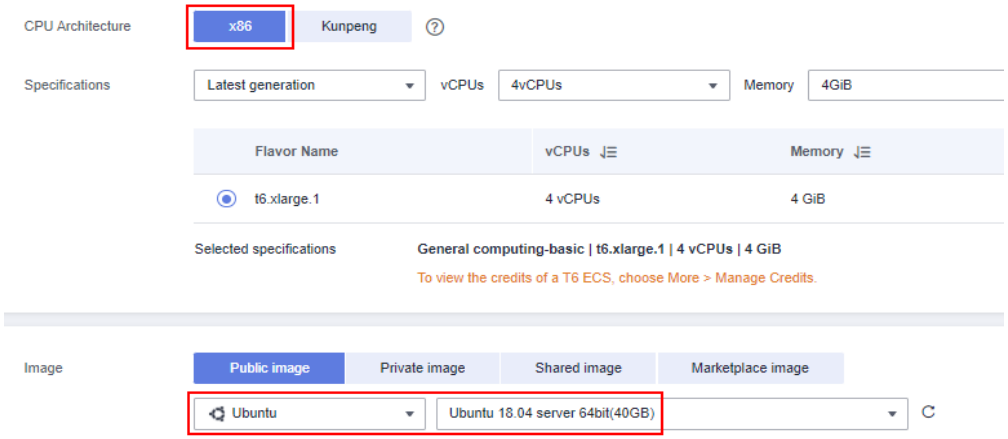
```
import os
import socket

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'])
```

Etapa 3 Preparar um servidor de imagens

Obtenha um servidor Linux x86_64 executando o Ubuntu 18.04. Um ECS ou seu PC local servirão.

Para obter detalhes sobre como comprar um ECS, consulte **Compra e logon em um ECS Linux**. Selecione uma imagem pública. Uma imagem do Ubuntu 18.04 é recomendada.

Figura 4-9 Criar um ECS usando uma imagem pública (x86)



Etapa 4 Criar uma imagem personalizada

Objetivo: crie e instale imagens de contêiner do software a seguir e use o serviço de treinamento do ModelArts para executar as imagens.

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

A seguir, descrevemos como criar uma imagem personalizada gravando um Dockerfile.

1. Instale o Docker.

   O seguinte usa o sistema operacional Linux x86_64 como um exemplo para descrever como obter um pacote de instalação do Docker. Para obter detalhes, consulte **documentos oficiais do Docker**. Execute os seguintes comandos para instalar o Docker:

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


Se o comando **docker images** é executado, o Docker foi instalado. Nesse caso, pule essa etapa.

2. Verifique a versão do mecanismo do Docker. Execute o seguinte comando:
```
docker version | grep -A 1 Engine
```
As seguintes informações são exibidas:
```
 Engine:
  Version:          18.09.0
```

NOTA

Recomendamos que você use mecanismo do Docker desta versão ou posterior para criar uma imagem personalizada.

3. Crie uma pasta chamada **context**.
```
mkdir -p context
```
4. Baixe o arquivo de instalação do Miniconda3.

   Baixe o arquivo de instalação do Miniconda3 py37 4.12.0 (Python 3.7.13) de **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

5. Baixe o arquivo de instalação do openmpi 3.0.0.

   Baixe o arquivo editado de openmpi 3.0.0 usando Horovod v0.22.1 de **https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz**.

6. Armazene os arquivos de Miniconda3 e openmpi 3.0.0 na pasta **context**. O seguinte mostra a pasta **context**:
```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```
7. Grave o Dockerfile da imagem do contêiner.

   Crie um arquivo vazio chamado **Dockerfile** na pasta **context** e grave o seguinte conteúdo para o arquivo:
```
# The host must be connected to the public network for creating a container image.

# Basic container image at 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

# The default user of the basic container image is root.
# USER root

# Copy the Miniconda3 (Python 3.7.13) installation files to the /tmp directory of the basic container image.
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

# Install Miniconda3 to the /home/ma-user/miniconda3 directory of the basic container image.
# 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

# Create the final container image.
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# Install vim, cURL, net-tools, and the SSH tool in Huawei Mirrors.
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

# Install the Open MPI 3.0.0 file written using Horovod v0.22.1.
# 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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 of the basic container image exists. User ma-
user can directly use it.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the
directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-
user/miniconda3

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to avoid log loss.
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# Set the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure sshd to support SSH password-free login.
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
```

Para obter detalhes sobre como gravar um Dockerfile, consulte **documentos oficiais do Docker**.

8. Verifique se o Dockerfile foi criado. O seguinte mostra a pasta **context**:

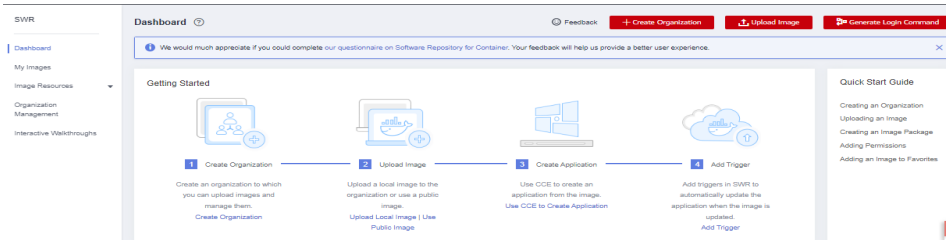
```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```


9. Crie a imagem do contêiner. Execute o comando a seguir no diretório em que o Dockerfile está armazenado para criar a imagem de contêiner **mpi:3.0.0-cuda11.1**:
```
docker build . -t mpi:3.0.0-cuda11.1
```
As seguintes informações de log exibidas durante a criação da imagem indicam que a imagem foi criada.
```
naming to docker.io/library/mpi:3.0.0-cuda11.1
```

Etapa 5 Carregar a imagem para o SWR

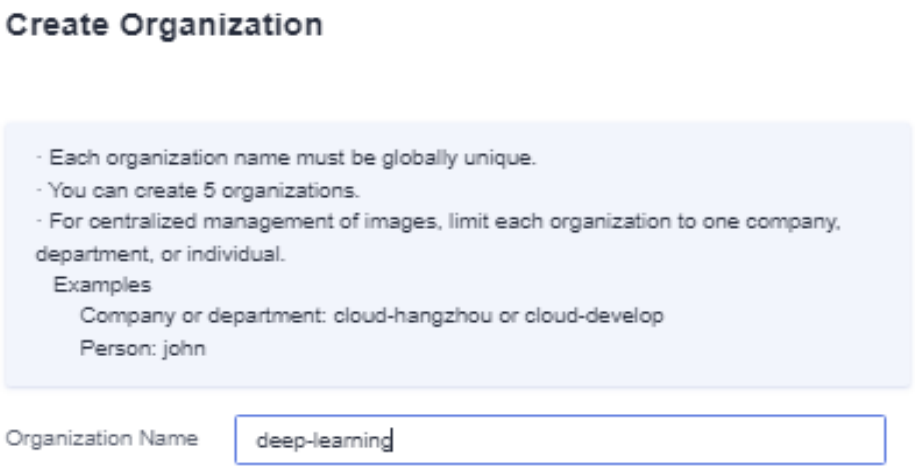
1. Faça logon no console do SWR e selecione a região de destino.

Figura 4-10 Console do SWR



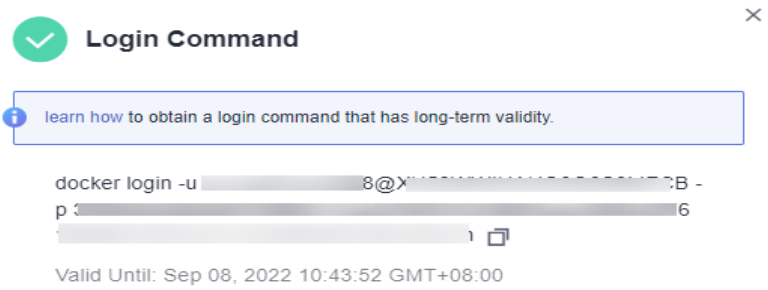
2. Clique em **Create Organization** no canto superior direito e insira um nome de organização para criar uma organização. Personalize o nome da organização. Substitua o nome da organização **deep-learning** nos comandos subsequentes pelo nome real da organização.

Figura 4-11 Criar uma organização



3. Clique em **Generate Login Command** no canto superior direito para obter um comando de logon.

Figura 4-12 Comando de logon



4. Efetue logon no ambiente local como o usuário **root** e digite o comando de logon.
5. Carregue a imagem para o SWR.
   a. Execute o seguinte comando para marcar a imagem carregada:
   ```
   #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
   sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
   b. Execute o seguinte comando para carregar a imagem:
   ```
   #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
   sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
6. Depois que a imagem for carregada, escolha **My Images** no painel de navegação esquerdo do console do SWR para exibir as imagens personalizadas enviadas.

   swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1 é o URL da imagem personalizada do SWR.

Etapa 6 Criar um trabalho de treinamento no ModelArts

1. Faça logon no console de gerenciamento do ModelArts, verifique se a autorização de acesso foi configurada para sua conta. Para obter detalhes, consulte **Configuração da autorização da agência**. Se você tiver sido autorizado usando chaves de acesso, limpe a autorização e configure a autorização da agência.
2. Faça logon no console de gerenciamento do ModelArts. No painel de navegação à esquerda, escolha **Training Management** > **Training Jobs (New)**.
3. Na página **Create Training Job**, defina os parâmetros necessários e clique em **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - **Image path**: **swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1**
   - **Code Directory**: caminho do OBS para o script de inicialização, por exemplo, **obs://test-modelarts/mpi/demo-code/**.
   - **Boot Command**: **bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/mpi-verification.py**
   - **Environment Variable**: adicione **MY_SSHD_PORT = 38888**.
   - **Resource Pool**: **Public resource pools**
   - **Resource Type**: selecione **GPU**.
   - **Compute Nodes**: digite **1** ou **2**.

- **Persistent Log Saving**: ativado
- **Job Log Path**: defina este parâmetro como o caminho do OBS para armazenar logs de treinamento, por exemplo, **obs://test-modelarts/mpi/log/**.

4. Verifique os parâmetros do trabalho de treinamento e clique em **Submit**.
5. Aguarde até que o trabalho de treinamento seja concluído.

   Depois que um trabalho de treinamento é criado, as operações como baixa de imagem de contêiner, baixa de diretório de código e execução de comandos de inicialização são executadas automaticamente no back-end. Geralmente, a duração do treinamento varia de dezenas de minutos a várias horas, dependendo do procedimento de treinamento e dos recursos selecionados. Depois que o trabalho de treinamento é executado, o log semelhante ao seguinte é emitido.

   Figura 4-13 Execute logs do worker-0 com um nó de computação e especificações de GPU

```
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
```

   Defina **Compute Nodes** como **2** e execute o trabalho de treinamento. **Figura 4-14** e **Figura 4-15** mostram as informações do log.

   Figura 4-14 Execute logs do worker-0 com dois nós de computação e especificações de GPU

```
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
```

Figura 4-15 Execute logs do worker-1 com dois nós de computação e especificações de GPU

```
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
```

4.2.3 Exemplo: criar uma imagem personalizada para treinamento (Horovod-PyTorch e GPUs)

Esta seção descreve como criar uma imagem e usá-la para treinamento no ModelArts. O mecanismo de IA usado na imagem é Horovod 0.22.1 + PyTorch 1.8.1, e os recursos usados para treinamento são GPUs.

NOTA

Esta seção se aplica somente aos trabalhos de treinamento da nova versão.

Cenário

Neste exemplo, grave um Dockerfile para criar uma imagem personalizada em um servidor Linux x86_64 executando o Ubuntu 18.04.

Objetivo: crie e instale imagens de contêiner dos seguintes softwares e use as imagens e CPUs/GPUs para treinamento no ModelArts.

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

Procedimento

Antes de usar uma imagem personalizada para criar um trabalho de treinamento, você precisa estar familiarizado com o Docker e ter experiência em desenvolvimento.

1. **Pré-requisitos**
2. **Etapa 1 Criar um bucket e uma pasta do OBS**
3. **Etapa 2 Preparar o script de treinamento e carregá-lo no OBS**
4. **Etapa 3 Preparar um servidor**
5. **Etapa 4 Criar uma imagem personalizada**
6. **Etapa 5 Carregar a imagem para o SWR**
7. **Etapa 6 Criar um trabalho de treinamento no ModelArts**

Pré-requisitos

Você registrou uma conta da Huawei Cloud. A conta não pode estar em atraso ou congelada.

Etapa 1 Criar um bucket e uma pasta do OBS

Crie um bucket e pastas no OBS para armazenar o conjunto de dados de amostra e o código de treinamento. **Tabela 4-3** lista as pastas a serem criadas. Substitua o nome do bucket e os nomes da pasta no exemplo por nomes reais.

Para obter detalhes sobre como criar um bucket e uma pasta do OBS, consulte **Criação de um bucket** e **Criação de uma pasta**.

Verifique se o diretório do OBS que você usa e o ModelArts estão na mesma região.

Tabela 4-3 Pasta a criar

Nome	Descrição
obs://test-modelarts/pytorch/demo-code/	Armazena o script de treinamento.
obs://test-modelarts/pytorch/log/	Armazene arquivos de log de treinamento.

Etapa 2 Preparar o script de treinamento e carregá-lo no OBS

Obtenha scripts de treinamento **pytorch_synthetic_benchmark.py** e **run_mpi.sh** e carregue para **obs://test-modelarts/horovod/demo-code/** no bucket do OBS.

pytorch_synthetic_benchmark.py é o seguinte:

```
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()
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
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 é o seguinte:

```
#!/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
            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
```



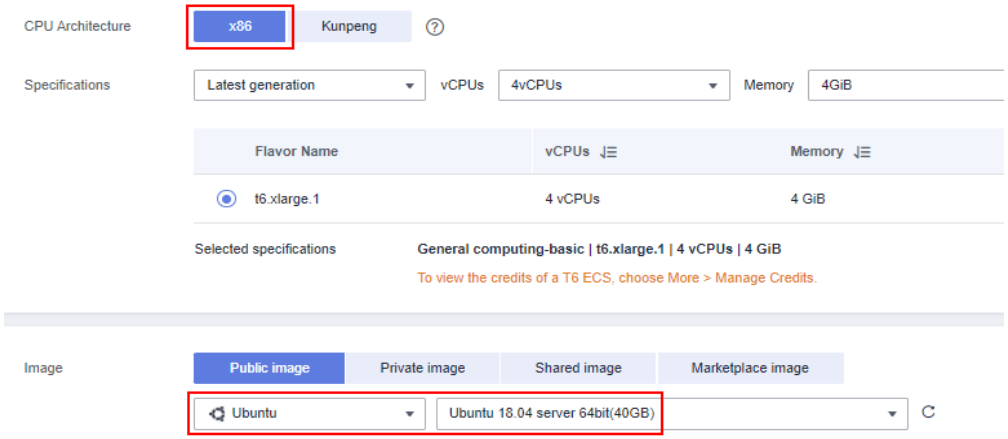
```
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

Etapa 3 Preparar um servidor

Obtenha um servidor Linux x86_64 executando o Ubuntu 18.04. Um ECS ou seu PC local servirão.

Para obter detalhes sobre como comprar um ECS, consulte **Compra e logon em um ECS Linux**. Selecione uma imagem pública. Uma imagem do Ubuntu 18.04 é recomendada.

Figura 4-16 Criar um ECS usando uma imagem pública (x86)



Etapa 4 Criar uma imagem personalizada

Crie uma imagem de contêiner com as seguintes configurações e use a imagem para criar um trabalho de treinamento no ModelArts:

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

A seguir, descrevemos como criar uma imagem personalizada gravando um Dockerfile.

1. Instale o Docker.

   O seguinte usa o sistema operacional Linux x86_64 como um exemplo para descrever como obter um pacote de instalação do Docker. Para obter mais detalhes sobre como instalar o Docker, consulte os **documentos oficiais do Docker**. Execute os seguintes comandos para instalar o Docker:

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


Se o comando **docker images** é executado, o Docker foi instalado. Nesse caso, pule essa etapa.

2. Verifique a versão do mecanismo do Docker. Execute o seguinte comando:
```
docker version | grep -A 1 Engine
```
As seguintes informações são exibidas:
```
 Engine:
  Version:          18.09.0
```
NOTA

Use o mecanismo Docker da versão anterior ou posterior para criar uma imagem personalizada.

3. Crie uma pasta chamada **context**.
```
mkdir -p context
```
4. Obtenha o arquivo **pip.conf**. Neste exemplo, a fonte pip fornecida pelo Huawei Mirrors é usada, que é a seguinte:
```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```
NOTA

Para obter **pip.conf**, vá para Huawei Mirrors em **https://mirrors.huaweicloud.com/home** e procure **pypi**.

5. Baixe o arquivo de código-fonte do Horovod.

Baixe **horovod-0.22.1.tar.gz** de https://pypi.org/project/horovod/0.22.1/#files.

6. Baixe os arquivos torch*.whl.

Baixe os seguintes arquivos .whl de: https://download.pytorch.org/whl/torch_stable.html:

- 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

NOTA

O código de URL do sinal de mais (+) é %2B. Ao pesquisar arquivos nos sites anteriores, substitua o sinal de mais (+) no nome do arquivo por %2B, por exemplo, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

7. Baixe o arquivo de instalação do Miniconda3.

Baixe o arquivo de instalação do Miniconda3 py37 4.12.0 (Python 3.7.13) de https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh.

8. Grave a imagem de contêiner de Dockerfile.

Crie um arquivo vazio chamado **Dockerfile** na pasta **context** e copie o seguinte conteúdo para o arquivo:
```
# The server on which the container image is created must access the Internet.

# Base container image at 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

# Install CMake obtained from Huawei Mirrors.
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

# The default user of the base container image is root.
# USER root

# Use the PyPI configuration obtained from Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container
image.
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
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base
container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/
miniconda3

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# 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 && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# Environment variables required for building Horovod with PyTorch
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

# Install the .whl files using default Miniconda3 Python environment /home/ma-
user/miniconda3/bin/pip.
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

# Build and install horovod-0.22.1.tar.gz using default Miniconda3 Python
environment /home/ma-user/miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/horovod-0.22.1.tar.gz

# Create the container image.
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
```



```
# Install the vim, cURL, net-tools, MLNX_OFED, and SSH tools obtained from
Huawei Mirrors.
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

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# 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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the basic container image. User ma-
user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the
directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-
user/miniconda3

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure sshd to support SSH password-free login.
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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    PYTHONUNBUFFERED=1
```

Para obter detalhes sobre como gravar um Dockerfile, consulte os **documentos oficiais do Docker**.

9. Baixe o pacote de instalação do MLNX_OFED.

   Vá para **Linux Drivers**. Na guia **Download**, escolha os pacotes de instalação em **Current Versions** e **Archive Versions**. Neste exemplo, escolha **Archive Versions**, defina **Version** como **5.4-3.5.8.0-LTS**, **OS Distribution** para **Ubuntu**, **OS Distribution Version** para **Ubuntu 18.04**, **Architecture** como **x86_64** e baixe o pacote de instalação **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

10. Baixe **openmpi-3.0.0-bin.tar.gz**.

    Baixe **openmpi-3.0.0-bin.tar.gz** de https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz.

11. Armazene o arquivo de fonte pip, o arquivo torch*.whl e o arquivo de instalação do Miniconda3 na pasta **context**, que é a seguinte:

```
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. Crie a imagem do contêiner. Execute o comando a seguir no diretório em que o Dockerfile está armazenado para criar a imagem de contêiner **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
```

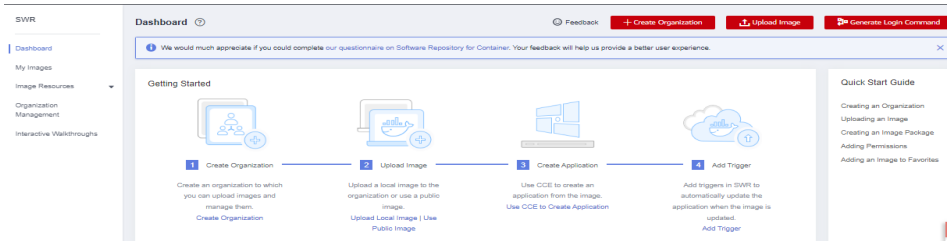
    O log a seguir mostra que a imagem foi criada.

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

Etapa 5 Carregar a imagem para o SWR

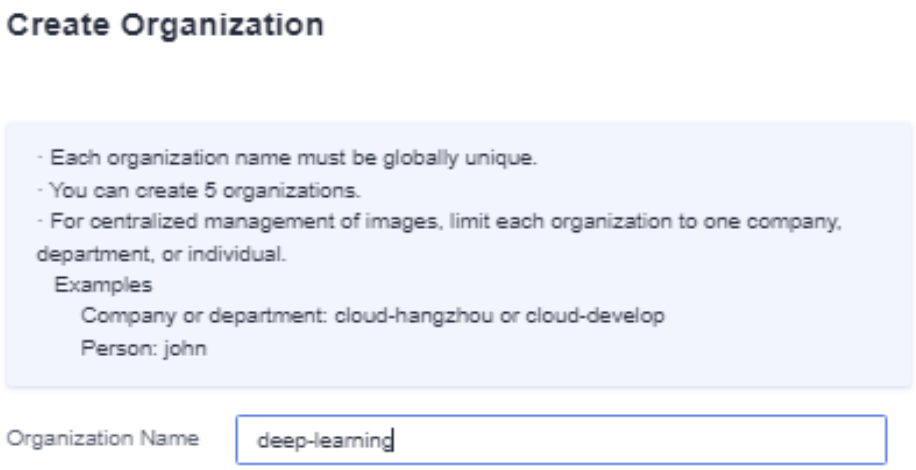
1. Faça logon no console do SWR e selecione a região de destino.

   Figura 4-17 Console do SWR



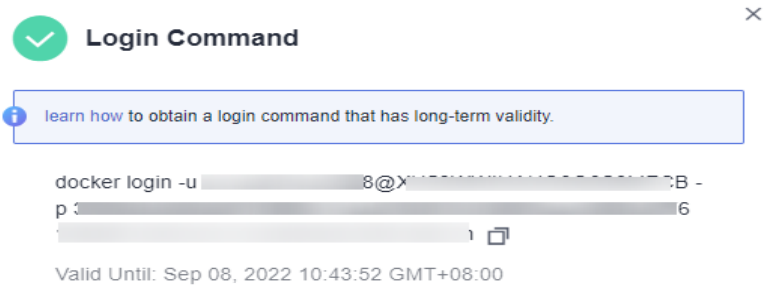
2. Clique em **Create Organization** no canto superior direito e insira um nome de organização para criar uma organização. Personalize o nome da organização. Substitua o nome da organização **deep-learning** nos comandos subsequentes pelo nome real da organização.

Figura 4-18 Criar uma organização



3. Clique em **Generate Login Command** no canto superior direito para obter um comando de logon.

Figura 4-19 Comando de logon



4. Efetue logon no ambiente local como o usuário **root** e digite o comando logon.
5. Carregue a imagem para o SWR.
   a. Marque a imagem carregada.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   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. Execute o seguinte comando para carregar a imagem:
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker push swr.{region-id}.{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
   ```
6. Depois que a imagem for carregada, escolha **My Images** no painel de navegação à esquerda do console do SWR para exibir as imagens personalizadas carregadas.

Etapa 6 Criar um trabalho de treinamento no ModelArts

1. Faça logon no console de gerenciamento do ModelArts, verifique se a autorização de acesso foi configurada para sua conta. Para obter detalhes, consulte **Configuração da autorização da agência**. Se você tiver sido autorizado usando chaves de acesso, limpe a autorização e configure a autorização da agência.
2. No painel de navegação, escolha **Training Management** > **Training Jobs**. A lista de trabalhos de treinamento é exibida por padrão.

3. Clique em **Create Training Job**. Na página exibida, configure os parâmetros e clique em **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: imagem criada em **Etapa 5 Carregar a imagem para o SWR**.
   - **Code Directory**: diretório onde o arquivo de script de inicialização é armazenado no OBS, por exemplo, **obs://test-modelarts/pytorch/demo-code/**. O código de treinamento é baixado automaticamente para o diretório **${MA_JOB_DIR}/demo-code** do contêiner de treinamento. **demo-code** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Boot Command**: **bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/pytorch_synthetic_benchmark.py**. **demo-code** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Environment Variable**: clique em **Add Environment Variable** e adicione a variável de ambiente **MY_SSHD_PORT=38888**.
   - **Resource Pool**: selecione **Public resource pools**.
   - **Resource Type**: selecione **GPU**.
   - **Compute Nodes**: 1 ou 2
   - **Persistent Log Saving**: ativado
   - **Job Log Path**: caminho do OBS para logs de treinamento armazenados, por exemplo, **obs://test-modelarts/pytorch/log/**
4. Confirme as configurações do trabalho de treinamento e clique em **Submit**.
5. Aguarde até que o trabalho de treinamento seja criado.

   Depois que você enviar a solicitação de criação de trabalho, o sistema executará automaticamente operações no back-end, como baixar a imagem do contêiner e o diretório de código e executar o comando de inicialização. Um trabalho de treinamento requer um certo período de tempo para a execução. A duração varia de dezenas de minutos a várias horas, variando dependendo da lógica do serviço e dos recursos selecionados. Depois que o trabalho de treinamento é executado, o log semelhante ao seguinte é emitido.

Figura 4-20 Executar logs de trabalhos de treinamento com especificações de GPU (um nó de computação)

```
58 Iter #0: 342.4 img/sec per GPU
59 Iter #1: 342.7 img/sec per GPU
60 Iter #2: 342.8 img/sec per GPU
61 Iter #3: 342.5 img/sec per GPU
62 Iter #4: 342.9 img/sec per GPU
63 Iter #5: 342.8 img/sec per GPU
64 Iter #6: 342.9 img/sec per GPU
65 Iter #7: 343.0 img/sec per GPU
66 Iter #8: 342.6 img/sec per GPU
67 Iter #9: 342.7 img/sec per GPU
68 Img/sec per GPU: 342.7 +-0.3
69 Total img/sec on 1 GPU(s): 342.7 +-0.3
```

Figura 4-21 Executar logs de trabalhos de treinamento com especificações de GPU (dois nós de computação)

```
84 Iter #0: 115.1 img/sec per GPU
85 Iter #1: 123.5 img/sec per GPU
86 Iter #2: 115.7 img/sec per GPU
87 Iter #3: 117.4 img/sec per GPU
88 Iter #4: 120.6 img/sec per GPU
89 Iter #5: 126.9 img/sec per GPU
90 Iter #6: 119.4 img/sec per GPU
91 Iter #7: 118.4 img/sec per GPU
92 Iter #8: 122.0 img/sec per GPU
93 Iter #9: 118.2 img/sec per GPU
94 Img/sec per GPU: 119.7 +-6.8
95 Total img/sec on 2 GPU(s): 239.4 +-13.5
```

4.2.4 Exemplo: criar uma imagem personalizada para treinamento (MindSpore e GPUs)

Esta seção descreve como criar uma imagem e usá-la para treinamento no ModelArts. O mecanismo de IA usado na imagem é o MindSpore e os recursos usados para o treinamento são as GPUs.

NOTA

Esta seção se aplica somente aos trabalhos de treinamento da nova versão.

Cenário

Neste exemplo, grave um Dockerfile para criar uma imagem personalizada em um servidor Linux x86_64 executando o Ubuntu 18.04.

Crie uma imagem de contêiner com as seguintes configurações e use a imagem para criar um trabalho de treinamento com GPU no ModelArts:

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

Procedimento

Antes de usar uma imagem personalizada para criar um trabalho de treinamento, você precisa estar familiarizado com o Docker e ter experiência em desenvolvimento.

- **Pré-requisitos**
- **Etapa 1 Criar um bucket e uma pasta do OBS**
- **Etapa 2 Criar um conjunto de dados e carregá-lo para o OBS**
- **Etapa 3 Preparar o script de treinamento e carregá-lo no OBS**
- **Etapa 4 Preparar um servidor**
- **Etapa 5 Criar uma imagem personalizada**
- **Etapa 6 Carregar a imagem para o SWR**
- **Etapa 7 Criar um trabalho de treinamento no ModelArts**

Pré-requisitos

Você registrou uma conta da Huawei Cloud. A conta não pode estar em atraso ou congelada.

Etapa 1 Criar um bucket e uma pasta do OBS

Crie um bucket e pastas no OBS para armazenar o conjunto de dados de amostra e o código de treinamento. **Tabela 4-4** lista as pastas a serem criadas. Substitua o nome do bucket e os nomes da pasta no exemplo por nomes reais.

Para obter detalhes, consulte **Criação de um bucket** e **Criação de uma pasta**.

Certifique-se de que o OBS e o ModelArts estejam na mesma região.

Tabela 4-4 Pastas obrigatórias do OBS

Pasta	Descrição
obs://test-modelarts/mindspore-gpu/resnet/	Armazena o script de treinamento.
obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/	Armazena arquivos de conjunto de dados.

Pasta	Descrição
obs://test-modelarts/mindspore-gpu/output/	Armazena arquivos de saída de treinamento.
obs://test-modelarts/mindspore-gpu/log/	Armazene arquivos de log de treinamento.

Etapa 2 Criar um conjunto de dados e carregá-lo para o OBS

Vá para http://www.cs.toronto.edu/~kriz/cifar.html, baixe o pacote **CIFAR-10 da versão binária (adequado para programas em C)**, descomprima-o e faça o upload dos dados descompactados para o diretório **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/** no bucket do OBS.

Etapa 3 Preparar o script de treinamento e carregá-lo no OBS

Obtenha o arquivo ResNet e o script **run_mpi.sh** e faça o upload para **obs://test-modelarts/mindspore-gpu/resnet/** no bucket do OBS.

Baixe o arquivo ResNet do **https://gitee.com/mindspore/models/tree/r1.8/official/cv/resnet**.

run_mpi.sh é o seguinte:

```
#!/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
            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

exit $RET_CODE
```

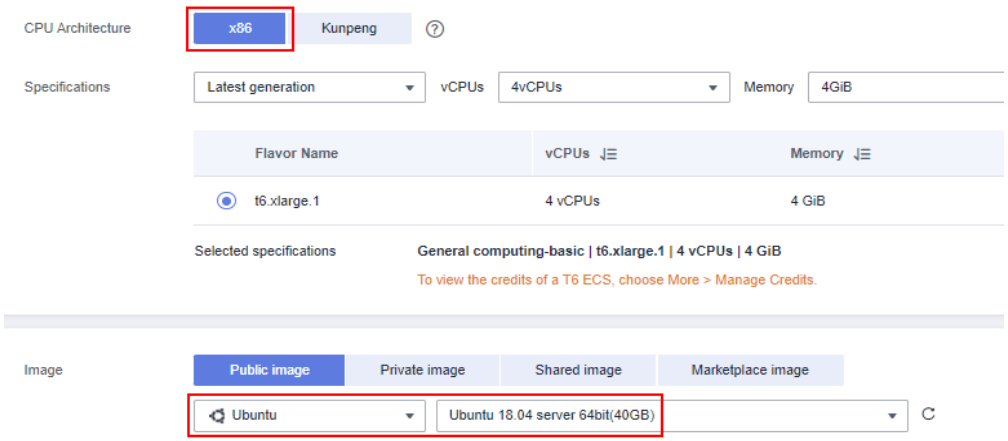
A pasta **obs://test-modelarts/mindspore-gpu/resnet/** contém os arquivos **resnet** e **run_mpi.sh**.

Etapa 4 Preparar um servidor

Obtenha um servidor Linux x86_64 executando o Ubuntu 18.04. Um ECS ou seu PC local servirão.

Para obter detalhes sobre como comprar um ECS, consulte **Compra e logon em um ECS Linux**. Selecione uma imagem pública. Uma imagem do Ubuntu 18.04 é recomendada.

Figura 4-22 Criar um ECS usando uma imagem pública (x86)



Etapa 5 Criar uma imagem personalizada

Crie uma imagem de contêiner com as seguintes configurações e use a imagem para criar um trabalho de treinamento no ModelArts:

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

Esta seção descreve como gravar um Dockerfile para criar uma imagem personalizada.

1. Instale o Docker.

   O seguinte usa o Linux x86_64 como um exemplo para descrever como obter um pacote de instalação do Docker. Para obter mais detalhes sobre como instalar o Docker, consulte os **documentos oficiais do Docker**. Execute o seguinte comando para instalar o Docker:

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

   Se o comando **docker images** puder ser executado, o Docker foi instalado. Nesse caso, pule essa etapa.

2. Verifique a versão do mecanismo do Docker. Execute o seguinte comando:

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

   As seguintes informações são exibidas:

   ```
   Engine:
    Version:          18.09.0
   ```

   NOTA

   Use o mecanismo Docker da versão anterior ou posterior para criar uma imagem personalizada.

3. Crie uma pasta chamada **context**.

   ```
   mkdir -p context
   ```

4. Obtenha o arquivo **pip.conf**. Neste exemplo, a fonte pip fornecida pelo Huawei Mirrors é usada, que é a seguinte:

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

   NOTA

   Para obter **pip.conf**, mude para Huawei Mirrors **https://mirrors.huaweicloud.com/home** e procure **pypi**.

5. Baixe **mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl** do **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**.

6. Baixe o arquivo de instalação do Miniconda3.

   Baixe **Miniconda3-py37_4.12.0-Linux-x86_64.sh** de https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh.

7. Grave a imagem de contêiner de Dockerfile.

   Crie um arquivo vazio chamado **Dockerfile** na pasta **context** e copie o seguinte conteúdo para o arquivo:

   ```
   # The server on which the container image is created must access the Internet.

   # Base container image at 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

   # The default user of the base container image is root.
   # USER root

   # Use the PyPI configuration obtained from Huawei Mirrors.
   RUN mkdir -p /root/.pip/
   COPY pip.conf /root/.pip/pip.conf
   ```



```
# Copy the installation files to the /tmp directory in the base container
image.
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
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base
container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/
miniconda3

# Install the whl file using default Miniconda3 Python environment /home/ma-
user/miniconda3/bin/pip.
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

# Create the container image.
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

# Install the vim, cURL, net-tools, MLNX_OFED, and SSH tools obtained from
Huawei Mirrors.
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

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# 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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the basic container image. User ma-
user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user
```



```
# Copy the /home/ma-user/miniconda3 directory from the builder stage to the
directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-
user/miniconda3

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure sshd to support SSH password-free login.
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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
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
```

Para obter detalhes sobre como gravar um Dockerfile, consulte os **documentos oficiais do Docker**.

8. Baixe **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

   Vá para https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/, clique em **Download**, defina **Version** para **5.4-3.5.8.0-LTS**, **OSDistributionVersion** para **Ubuntu 18.04** e **Architecture** para **x86_64** e baixe **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Baixe **openmpi-3.0.0-bin.tar.gz**.

   Baixe **openmpi-3.0.0-bin.tar.gz** do https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz.

10. Armazene o arquivo de instalação do Dockerfile e do Miniconda3 na pasta **context**, que é a seguinte:

    ```
    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. Crie a imagem do contêiner. Execute o comando a seguir no diretório em que o Dockerfile está armazenado para criar a imagem de contêiner **mindspore:1.8.1-ofed-cuda11.1**:

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

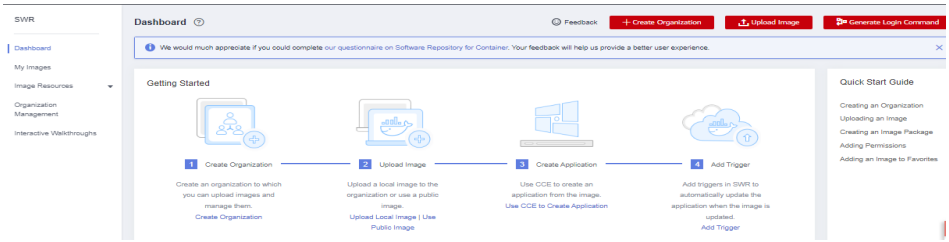
    O log a seguir mostra que a imagem foi criada.

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


Etapa 6 Carregar a imagem para o SWR

1. Faça logon no console do SWR e selecione a região de destino.

   Figura 4-23 Console do SWR

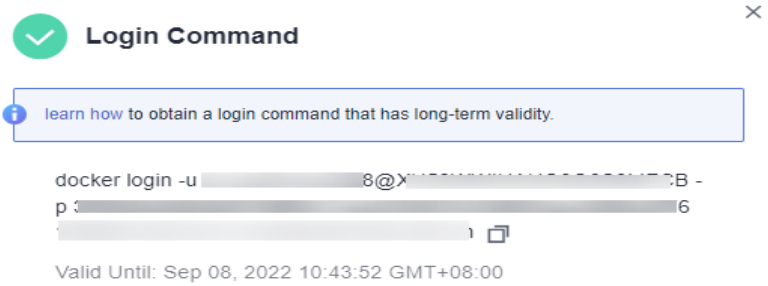
2. Clique em **Create Organization** no canto superior direito e insira um nome de organização para criar uma organização. Personalize o nome da organização. Substitua o nome da organização **deep-learning** nos comandos subsequentes pelo nome real da organização.

   Figura 4-24 Criar uma organização

3. Clique em **Generate Login Command** no canto superior direito para obter um comando de logon.

   Figura 4-25 Comando de logon

4. Efetue logon no ambiente local como o usuário **root** e digite o comando logon.
5. Carregue a imagem para o SWR.
   a. Marque a imagem carregada.

   ```
   # Replace the region, domain, as well as organization name deep-learning
   with the actual values.
   ```



```
sudo docker tag mindspore:1.8.1-ofed-cuda11.1 swr.{region-id}.{domain}/
deep-learning/mindspore:1.8.1-ofed-cuda11.1
```

b. Execute o seguinte comando para carregar a imagem:

```
# Replace the region, domain, as well as organization name deep-learning
with the actual values.
sudo docker push swr.{region-id}.{domain}/deep-learning/mindspore:1.8.1-
ofed-cuda11.1
```

6. Depois que a imagem for carregada, escolha **My Images** no painel de navegação à esquerda do console do SWR para exibir as imagens personalizadas carregadas.

Etapa 7 Criar um trabalho de treinamento no ModelArts

1. Faça logon no console de gerenciamento do ModelArts, verifique se a autorização de acesso foi configurada para sua conta. Para obter detalhes, consulte **Configuração da autorização da agência**. Se você tiver sido autorizado usando chaves de acesso, limpe a autorização e configure a autorização da agência.
2. No painel de navegação, escolha **Training Management** > **Training Jobs**. A lista de trabalhos de treinamento é exibida por padrão.
3. Clique em **Create Training Job**. Na página exibida, configure os parâmetros e clique em **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: imagem criada em **Etapa 6 Carregar a imagem para o SWR**.
   - **Code Directory**: diretório onde o arquivo de script de inicialização é armazenado no OBS, por exemplo, **obs://test-modelarts/mindspore-gpu/resnet/**. O código de treinamento é baixado automaticamente para o diretório **${MA_JOB_DIR}/resnet** do contêiner de treinamento. **resnet** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Boot Command**: **bash ${MA_JOB_DIR}/resnet/run_mpi.sh python ${MA_JOB_DIR}/resnet/train.py**. **resnet** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Training Input**: clique em **Add Training Input**. Digite **data_path** para o nome, selecione o caminho do OBS para o conjunto de dados de destino, por exemplo, **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/**, e defina **Obtained from** para **Hyperparameters**.
   - **Training Output**: clique em **Add Training Output**. Digite **output_path** para o nome, selecione um caminho OBS para armazenar saídas de treinamento, por exemplo, **obs://test-modelarts/mindspore-gpu/output/**, e defina **Obtained from** para **Hyperparameters** e **Predownload** para **No**.
   - **Hyperparameters**: clique em **Add Hyperparameter** e adicione os seguintes hiperparâmetros:
     - run_distribute=True
     - device_num=1 (Defina esse parâmetro com base no número de GPUs nos flavors de instância.)
     - device_target=GPU
     - epoch_size=2
   - **Environment Variable**: clique em **Add Environment Variable** e adicione a variável de ambiente **MY_SSHD_PORT=38888**.
   - **Resource Pool**: selecione **Public resource pools**.

- **Resource Type**: selecione **GPU**.
- **Compute Nodes**: 1 ou 2
- **Persistent Log Saving**: ativado
- **Job Log Path**: caminho do OBS para logs de treinamento armazenados, por exemplo, **obs://test-modelarts/mindspore-gpu/log/**

4. Confirme as configurações do trabalho de treinamento e clique em **Submit**.
5. Aguarde até que o trabalho de treinamento seja criado.

   Depois que você enviar a solicitação de criação de trabalho, o sistema executará automaticamente operações no back-end, como baixar a imagem do contêiner e o diretório de código e executar o comando de inicialização. Um trabalho de treinamento requer um certo período de tempo para a execução. A duração varia de dezenas de minutos a várias horas, variando dependendo da lógica do serviço e dos recursos selecionados. Depois que o trabalho de treinamento é executado, o log semelhante ao seguinte é emitido.

   Figura 4-26 Executar logs de trabalhos de treinamento com especificações de GPU (um nó de computação)

```
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
```

   Figura 4-27 Executar logs de trabalhos de treinamento com especificações de GPU (dois nós de computação)

```
epoch: 1 step: 937, loss is 1.8482083
epoch: 1 step: 937, loss is 1.342748
Train epoch time: 488492.170 ms, per step time: 521.336 ms
Train epoch time: 488490.528 ms, per step time: 521.335 ms
epoch: 2 step: 937, loss is 0.9150252
Train epoch time: 180200.654 ms, per step time: 192.317 ms
epoch: 2 step: 937, loss is 0.9933052
Train epoch time: 180199.969 ms, per step time: 192.316 ms
172.16.0.45 slots=1
```

4.2.5 Exemplo: criar uma imagem personalizada para treinamento (TensorFlow e GPUs)

Esta seção descreve como criar uma imagem e usá-la para treinamento no ModelArts. O mecanismo de IA usado na imagem é o TensorFlow, e os recursos usados para o treinamento são as GPUs.

> **NOTA**
>
> Esta seção se aplica somente aos trabalhos de treinamento da nova versão.

Cenário

Neste exemplo, grave um Dockerfile para criar uma imagem personalizada em um servidor Linux x86_64 executando o Ubuntu 18.04.

Crie uma imagem de contêiner com as seguintes configurações e use a imagem para criar um trabalho de treinamento com GPU no ModelArts:

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

Procedimento

Antes de usar uma imagem personalizada para criar um trabalho de treinamento, você precisa estar familiarizado com o Docker e ter experiência em desenvolvimento.

1. **Pré-requisitos**
2. **Etapa 1 Criar um bucket e uma pasta do OBS**
3. **Etapa 2 Criar um conjunto de dados e carregá-lo para o OBS**
4. **Etapa 3 Preparar o script de treinamento e carregá-lo no OBS**
5. **Etapa 4 Preparar um servidor**
6. **Etapa 5 Criar uma imagem personalizada**
7. **Etapa 6 Carregar a imagem para o SWR**
8. **Etapa 7 Criar um trabalho de treinamento no ModelArts**

Pré-requisitos

Você registrou uma conta da Huawei Cloud. A conta não pode estar em atraso ou congelada.

Etapa 1 Criar um bucket e uma pasta do OBS

Crie um bucket e pastas no OBS para armazenar o conjunto de dados de amostra e o código de treinamento. **Tabela 4-5** lista as pastas a serem criadas. Substitua o nome do bucket e os nomes da pasta no exemplo por nomes reais.

Para obter detalhes sobre como criar um bucket e uma pasta do OBS, consulte **Criação de um bucket** e **Criação de uma pasta**.

Verifique se o diretório do OBS que você usa e o ModelArts estão na mesma região.

Tabela 4-5 Pastas obrigatórias do OBS

Pasta	Descrição
obs://test-modelarts/tensorflow/code/	Armazena o script de treinamento.
obs://test-modelarts/tensorflow/data/	Armazena arquivos de conjunto de dados.
obs://test-modelarts/tensorflow/log/	Armazene arquivos de log de treinamento.

Etapa 2 Criar um conjunto de dados e carregá-lo para o OBS

Faça o download do **mnist.npz** do **https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz** e faça o upload para **obs://test-modelarts/tensorflow/data/** no bucket do OBS.

Etapa 3 Preparar o script de treinamento e carregá-lo no OBS

Obtenha o script de treinamento **mnist.py** e envie-o para **obs://test-modelarts/tensorflow/code/** no bucket do OBS.

mnist.py é o seguinte:

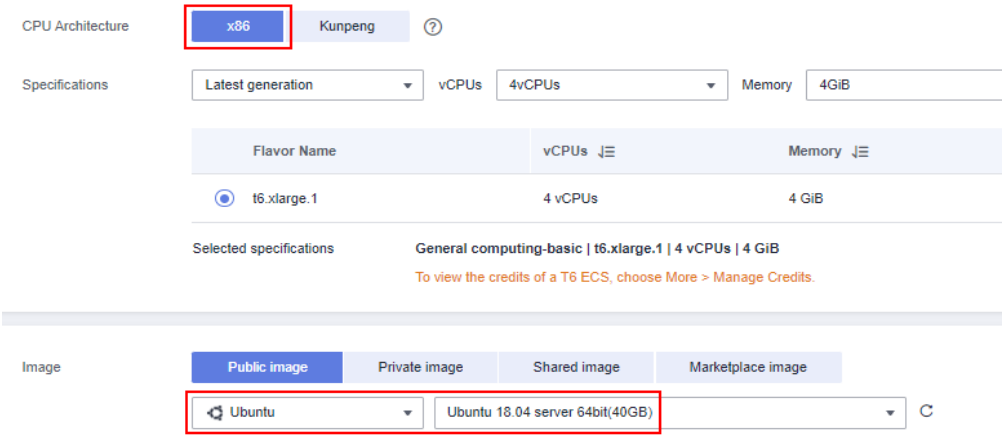
```
import argparse
import tensorflow as tf

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)
])

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)
```

Etapa 4 Preparar um servidor

Obtenha um servidor Linux x86_64 executando o Ubuntu 18.04. Um ECS ou seu PC local servirão.

Para obter detalhes sobre como comprar um ECS, consulte **Compra e logon em um ECS Linux**. Selecione uma imagem pública. Uma imagem do Ubuntu 18.04 é recomendada.

Figura 4-28 Criar um ECS usando uma imagem pública (x86)



Etapa 5 Criar uma imagem personalizada

Crie uma imagem de contêiner com as seguintes configurações e use a imagem para criar um trabalho de treinamento no ModelArts:

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

A seguir, descrevemos como criar uma imagem personalizada gravando um Dockerfile.

1. Instale o Docker.

   O seguinte usa o sistema operacional Linux x86_64 como um exemplo para descrever como obter um pacote de instalação do Docker. Para obter mais detalhes sobre como instalar o Docker, consulte os **documentos oficiais do Docker**. Execute os seguintes comandos para instalar o Docker:

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

   Se o comando **docker images** é executado, o Docker foi instalado. Nesse caso, pule essa etapa.

2. Verifique a versão do mecanismo do Docker. Execute o seguinte comando:

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

   As seguintes informações são exibidas:

   ```
   Engine:
    Version:          18.09.0
   ```

   > **NOTA**
   >
   > Use o mecanismo Docker da versão anterior ou posterior para criar uma imagem personalizada.

3. Crie uma pasta chamada **context**.

   ```
   mkdir -p context
   ```

4. Obtenha o arquivo **pip.conf**. Neste exemplo, a fonte pip fornecida pelo Huawei Mirrors é usada, que é a seguinte:

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



```
trusted-host = repo.huaweicloud.com
timeout = 120
```

> NOTA
>
> Para obter **pip.conf**, vá para Huawei Mirrors em https://mirrors.huaweicloud.com/home e procure **pypi**.

5. Baixe **tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl**.

   Baixe **tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl** de **https://pypi.org/project/tensorflow-gpu/2.10.0/#files**.

6. Baixe o arquivo de instalação do Miniconda3.

   Baixe o arquivo de instalação do Miniconda3 py37 4.12.0 (Python 3.7.13) de https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh.

7. Grave a imagem de contêiner de Dockerfile.

   Crie um arquivo vazio chamado **Dockerfile** na pasta **context** e copie o seguinte conteúdo para o arquivo:

```
# The server on which the container image is created must access the Internet.

# Base container image at 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

# The default user of the base container image is root.
# USER root

# Use the PyPI configuration obtained from Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
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
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install the TensorFlow .whl file using default Miniconda3 Python environment /home/ma-user/miniconda3/bin/pip.
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

# Create the container image.
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
```



```
# Install the vim, cURL, net-tools, and MLNX_OFED tools obtained from Huawei
Mirrors.
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-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

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the base container image. User ma-
user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Copy the /home/ma-user/miniconda3 directory from the builder stage to the
directory with the same name in the current container image.
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-
user/miniconda3

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
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
```

Para obter detalhes sobre como gravar um Dockerfile, consulte os **documentos oficiais do Docker**.

8. Baixe **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

   Vá para **Linux Drivers**. Na guia **Download**, defina **Version** para **5.4-3.5.8.0-LTS**, **OS Distribution Version** para **Ubuntu 18.04**, **Architecture** para **x86_64** e baixe **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Armazene o arquivo de instalação do Dockerfile e do Miniconda3 na pasta **context**, que é a seguinte:

```
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. Crie a imagem do contêiner. Execute o comando a seguir no diretório em que o Dockerfile está armazenado para criar a imagem de contêiner **tensorflow:2.10.0-ofed-cuda11.2**:


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

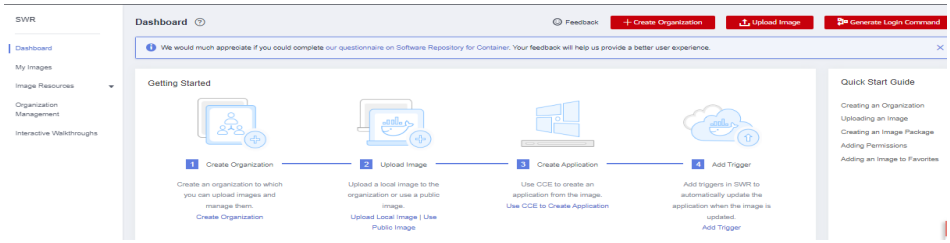
O log a seguir mostra que a imagem foi criada.

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

Etapa 6 Carregar a imagem para o SWR

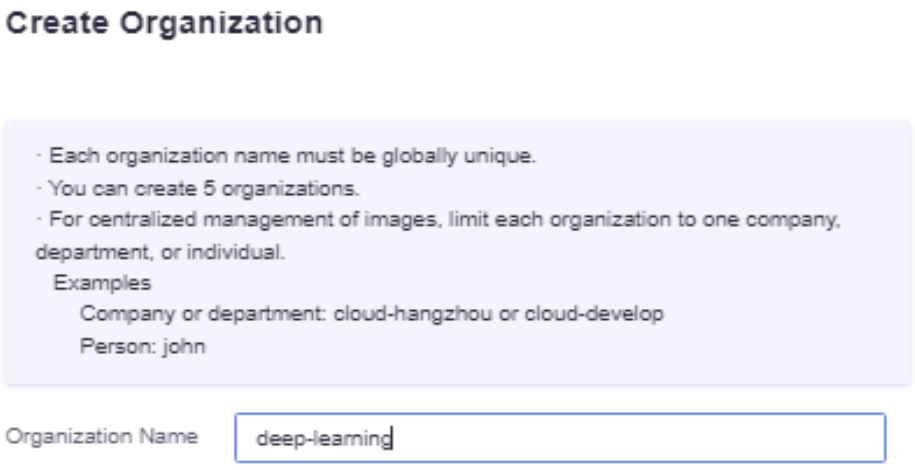
1. Faça logon no console do SWR e selecione a região de destino.

Figura 4-29 Console do SWR



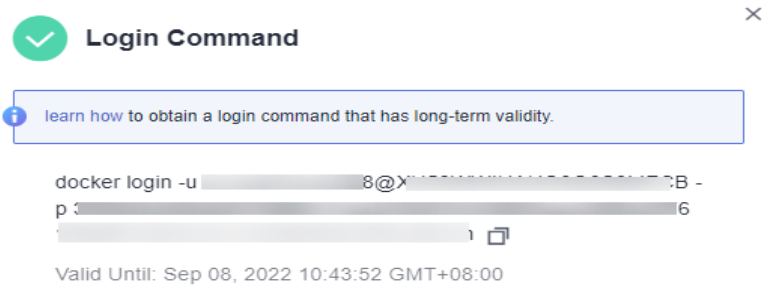
2. Clique em **Create Organization** no canto superior direito e insira um nome de organização para criar uma organização. Personalize o nome da organização. Substitua o nome da organização **deep-learning** nos comandos subsequentes pelo nome real da organização.

Figura 4-30 Criar uma organização



3. Clique em **Generate Login Command** no canto superior direito para obter um comando de logon.

Figura 4-31 Comando de logon



4. Efetue logon no ambiente local como o usuário **root** e digite o comando logon.
5. Carregue a imagem para o SWR.
   a. Marque a imagem carregada.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
   ```
   b. Execute o seguinte comando para carregar a imagem:
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker push swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
   ```
6. Depois que a imagem for carregada, escolha **My Images** no painel de navegação à esquerda do console do SWR para exibir as imagens personalizadas carregadas.

Etapa 7 Criar um trabalho de treinamento no ModelArts

1. Faça logon no console de gerenciamento do ModelArts, verifique se a autorização de acesso foi configurada para sua conta. Para obter detalhes, consulte **Configuração da autorização da agência**. Se você tiver sido autorizado usando chaves de acesso, limpe a autorização e configure a autorização da agência.
2. No painel de navegação, escolha **Training Management** > **Training Jobs**. A lista de trabalhos de treinamento é exibida por padrão.
3. Clique em **Create Training Job**. Na página exibida, configure os parâmetros e clique em **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: imagem criada em **Etapa 5 Criar uma imagem personalizada**.
   - **Code Directory**: diretório onde o arquivo de script de inicialização é armazenado no OBS, por exemplo, **obs://test-modelarts/tensorflow/code/**. O código de treinamento é baixado automaticamente para o diretório **${MA_JOB_DIR}/code** do contêiner de treinamento. **code** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Boot Command**: **python ${MA_JOB_DIR}/code/mnist.py**. **code** (personalizável) é o diretório de último nível do caminho do OBS.
   - **Training Input**: clique em **Add Training Input**. Digite **data_path** para o nome, selecione o caminho do OBS para **mnist.npz**, por exemplo, **obs://test-modelarts/tensorflow/data/mnist.npz** e defina **Obtained from** para **Hyperparameters**.
   - **Resource Pool**: selecione **Public resource pools**.
   - **Resource Type**: selecione **GPU**.
   - **Compute Nodes**: Digite **1**.
   - **Persistent Log Saving**: ativado
   - **Job Log Path**: caminho do OBS para logs de treinamento armazenados, por exemplo, **obs://test-modelarts/mindspore-gpu/log/**
4. Confirme as configurações do trabalho de treinamento e clique em **Submit**.
5. Aguarde até que o trabalho de treinamento seja criado.

   Depois que você enviar a solicitação de criação de trabalho, o sistema executará automaticamente operações no back-end, como baixar a imagem do contêiner e o diretório de código e executar o comando de inicialização. Um trabalho de treinamento

requer um certo período de tempo para a execução. A duração varia de dezenas de minutos a várias horas, variando dependendo da lógica do serviço e dos recursos selecionados. Depois que o trabalho de treinamento é executado, o log semelhante ao seguinte é emitido.

Figura 4-32 Executar logs de trabalhos de treinamento com especificações de GPU

```
    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
```

4.3 Preparação de uma imagem de treinamento

4.3.1 Especificações para imagens personalizadas para trabalhos de treinamento

Ao usar um modelo e um script de treinamento desenvolvidos localmente para criar uma imagem personalizada, verifique se a imagem está em conformidade com as especificações definidas pelo ModelArts.

Especificações

- Use o Ubuntu 18.04 para imagens personalizadas para caso as versões não sejam compatíveis.
- Não utilize uma imagem personalizada com mais de 15 GB. O tamanho não deve exceder metade do espaço do mecanismo de contêiner do pool de recursos. Caso contrário, a hora de início do trabalho de treinamento será afetada.

  O espaço do mecanismo de contêiner do pool de recursos públicos do ModelArts é de 50 GB. Por padrão, o espaço do mecanismo de contêiner do pool de recursos dedicados também é de 50 GB. Você pode personalizar o espaço do mecanismo de contêiner ao criar um pool de recursos dedicados.
- O **uid** do usuário padrão de uma imagem personalizada deve ser **1000**.

- O driver da GPU ou do Ascend não pode ser instalado em uma imagem personalizada. Quando você seleciona recursos da GPU para executar trabalhos de treinamento, o ModelArts coloca automaticamente o driver da GPU no diretório **/usr/local/nvidia** no ambiente de treinamento. Quando você seleciona recursos do Ascend para executar trabalhos de treinamento, o ModelArts coloca automaticamente o driver do Ascend no diretório **/usr/local/Ascend/driver**.
- Imagens personalizadas baseadas em x86 ou Arm podem ser executadas apenas com especificações correspondentes à sua arquitetura.
  - Execute o seguinte comando para verificar a arquitetura da CPU de uma imagem personalizada:
    ```
    docker inspect {Custom image path} | grep Architecture
    ```
    A seguir, a saída do comando para uma imagem personalizada baseada em Arm:
    ```
    "Architecture": "arm64"
    ```
  - Se o nome de uma especificação contiver **Arm**, esta especificação é uma arquitetura de CPU baseada em Arm.

  - Se o nome de uma especificação não contém **Arm**, esta especificação é uma arquitetura de CPU baseada em x86.

- O ModelArts não suporta baixa de pacotes de instalação de código aberto. Instale os pacotes de dependência exigidos pelo trabalho de treinamento na imagem personalizada.

4.3.2 Migração de uma imagem para o treinamento do ModelArts

Esta seção descreve como migrar uma imagem para o gerenciamento de treinamento.

1. Adicione o grupo de usuários padrão **ma-group** (**gid = 100**) do gerenciamento de treinamento para a imagem.

   NOTA

   Se o grupo de usuários cujo **gid** é **100** já existe, a mensagem de erro "groupadd: GID '100' already exists" pode ser exibida. Você pode usar o comando **cat /etc/group | grep 100** 100 para verificar se o grupo de usuários cujo GID é 100 existe.

   Se o grupo de usuários cujo **gid** é **100** já existir, ignore esta etapa e exclua o comando **RUN groupadd ma-group -g 100** do Dockerfile.

2. Adicione o usuário padrão **ma-user** (**uid = 1000**) do gerenciamento de treinamento para a imagem.

   NOTA

   Se o usuário cujo **uid** é **1000** já existir, a mensagem de erro "useradd: UID 1000 is not unique" pode ser exibida. Você pode usar o comando **cat /etc/passwd | grep 1000** para verificar se o usuário cujo UID é 1000 existe.

   Se o usuário cujo **uid** é **1000** já existe, ignore esta etapa e exclua o comando **RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user** do Dockerfile.

3. Modifique as permissões em arquivos na imagem para permitir que **ma-user** cujo **uid** é **1000** leia e grave os arquivos.

Você pode modificar uma imagem consultando o Dockerfile a seguir para que a imagem esteja em conformidade com as especificações para imagens personalizadas do gerenciamento de treinamento da nova versão.

```
FROM {An existing image}

USER root

# If the user group whose GID is 100 already exists, delete the groupadd command.
RUN groupadd ma-group -g 100
# If the user whose UID is 1000 already exists, delete the useradd command.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# Modify the permissions on image files so that user ma-user whose UID is 1000
can read and write the files.
RUN chown -R ma-user:100 {Path to the Python software package}

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PYTHONUNBUFFERED=1

# Configure the default user and working directory of the container image.
USER ma-user
WORKDIR /home/ma-user
```

Depois de editar o Dockerfile, execute o seguinte comando para criar uma nova imagem:

```
docker build -f Dockerfile . -t {New image}
```

Carregue a imagem nova para o SWR. Para mais detalhes, consulte **Como acessar o SWR e carregar imagens para ele?**

4.3.3 Uso de uma imagem de base para criar uma imagem de treinamento

O ModelArts fornece imagens de base baseadas em aprendizado profundo, como imagens de TensorFlow, PyTorch e MindSpore. Nestas imagens, o software obrigatório para a execução de trabalhos de treinamento foi instalado. Se o software nas imagens de base não puder atender aos seus requisitos de serviço, crie novas imagens com base nas imagens de base e use as novas imagens para criar trabalhos de treinamento.

Procedimento

Execute as seguintes operações para criar uma imagem usando uma imagem de base de treinamento:

1. Instale o Docker. Se o comando **docker images** é executado, o Docker foi instalado. Nesse caso, pule essa etapa.

   O seguinte usa o Linux x86_64 como um exemplo para descrever como obter o pacote de instalação do Docker. Execute o seguinte comando para instalar o Docker:

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

2. Crie uma pasta chamada **context**.

   ```
   mkdir -p context
   ```

3. Obtenha o arquivo **pip.conf**.

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


4. Crie uma nova imagem com base em uma imagem de base de treinamento fornecida pelo ModelArts. Salve o Dockerfile editado na pasta **context**. Para obter detalhes sobre como obter uma imagem de base de treinamento, consulte **Imagens de base de treinamento disponíveis**.

```
FROM {Path to the training base image provided by ModelArts}

# Configure pip.
RUN mkdir -p /home/ma-user/.pip/
COPY --chown=ma-user:ma-group pip.conf /home/ma-user/.pip/pip.conf

# Configure the preset environment variables of the container image.
# Add the Python interpreter path to the PATH environment variable.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PATH=${ANACONDA_DIR}/envs/${ENV_NAME}/bin:$PATH \
    PYTHONUNBUFFERED=1

RUN /home/ma-user/anaconda/bin/pip install --no-cache-dir numpy
```

5. Execute o seguinte comando no diretório em que o Dockerfile está armazenado para criar uma imagem de contêiner, por exemplo, **training:v1**:

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

6. Carregue a imagem nova para o SWR. Para mais detalhes, consulte **Como acessar o SWR e carregar imagens para ele?**.
7. Use a imagem personalizada para criar um trabalho de treinamento no ModelArts. Para mais detalhes, consulte **Uso de uma imagem personalizada para criar um trabalho de treinamento baseado em CPU ou GPU**.

4.3.4 Instalação de MLNX_OFED em uma imagem de contêiner

Cenários

A NIC da Mellanox Technologies foi configurada em servidores de GPU do ModelArts para suportar Acesso remoto direto à memória (RDMA). Como resultado, você pode instalar MLNX_OFED na imagem do contêiner, o que permitirá que a **NCCL** aproveite a NIC e aumente a eficiência da comunicação entre nós.

Depois que esta NIC é habilitada para NCCL, NET/IB é usado para a comunicação entre nós. Se essa NIC não estiver habilitada, NET/Socket será usado para comunicação entre nós. NET/IB é melhor que NET/Socket em termos de latência e largura de banda.

Tabela 4-6 Instalação de NIC e MLNX_OFED da Mellanox Technologies em servidores de GPU do ModelArts

Modelo da GPU	NIC da Mellanox Technologies	Versão de MLNX_OFED instalada	Versão de MLNX_OFED recomendada para imagem de contêiner
Vnt1	ConnectX-5	4.3-1.0.1.0/4.5-1.0.1.0	4.9-6.0.6.0-LTS
Ant8/ Ant1	ConnectX-6 Dx	5.5-1.0.3.2	5.8-2.0.3.0-LTS

Instalação de MLNX_OFED

Tome a imagem do contêiner de Ubuntu18.04 como um exemplo. O Dockerfile para instalar o MLNX_OFED 4.9-6.0.6.0-LTS é o seguinte:

> **NOTA**
>
> O host usado para baixar arquivos e criar imagens de contêiner usando um Dockerfile deve ser capaz de se conectar à rede pública.

```
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

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 --no-install-recommends -y lsb-core curl && \
    curl -k -o /tmp/MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64.tgz https://
content.mellanox.com/ofed/MLNX_OFED-4.9-6.0.6.0/MLNX_OFED_LINUX-4.9-6.0.6.0-
ubuntu18.04-x86_64.tgz && \
    cd /tmp && \
    tar xzf MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64.tgz && \
    cd MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64 && \
    ./mlnxofedinstall --user-space-only --without-fw-update --without-neohost-
backend --force && \
    rm /tmp/MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64.tgz && \
    rm -rf /tmp/MLNX_OFED_LINUX-4.9-6.0.6.0-ubuntu18.04-x86_64 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf
```

Crie uma imagem de contêiner consultando este exemplo de comando:

```
docker build -f Dockerfile . -t nvidia/cuda:mlnx-ofed-4.9-11.1.1-runtime-
ubuntu18.04
```

Depois que a imagem de contêiner for criada, execute o seguinte comando para obter a versão de MLNX_OFED na imagem de contêiner:

```
docker run -ti --rm nvidia/cuda:mlnx-ofed-4.9-11.1.1-runtime-ubuntu18.04
ofed_info | head -n 1
```

A saída do comando é a seguinte:

```
MLNX_OFED_LINUX-4.9-6.0.6.0 (OFED-4.9-6.0.6):
```

4.4 Criação de um algoritmo usando uma imagem personalizada

Seus algoritmos desenvolvidos localmente ou algoritmos desenvolvidos usando outras ferramentas podem ser enviados para ModelArts para gerenciamento unificado.

Entradas para criar um algoritmo

Você pode criar um algoritmo usando uma imagem personalizada no ModelArts de uma das seguintes maneiras:

- Entrada 1: no console do ModelArts, escolha **Algorithm Management** > **My algorithms**. Em seguida, crie um algoritmo e use-o em trabalhos de treinamento ou publique-o no Galeria de IA.
- Entrada 2: no console do ModelArts, escolha **Training Management** > **Training Jobs** e clique em **Create Training Job** para criar um algoritmo personalizado e enviar um trabalho de treinamento. Para mais detalhes, consulte **Uso de uma imagem personalizada para criar um trabalho de treinamento baseado em CPU ou GPU**.

Parâmetros para criar um algoritmo

Tabela 4-7 Parâmetros para criar um algoritmo

Parâmetro	Descrição
Boot Mode	Selecione **Custom images**. Este parâmetro é obrigatório.
Image	URL de uma imagem do SWR. Este parâmetro é obrigatório. ● Imagens privadas ou imagens compartilhadas: clique em **Select** à direita para selecionar uma imagem do SWR. Certifique-se de que a imagem foi carregada no SWR. Para mais detalhes, consulte **Como acessar o SWR e carregar imagens para ele?**. ● Imagens públicas: você também pode inserir manualmente o caminho da imagem no formato de "<Organization to which your image belongs>/<Image name>" no SWR. Não contenha o nome de domínio (swr.<region>.xxx.com) no caminho porque o sistema adicionará automaticamente o nome de domínio ao caminho. Por exemplo: `modelarts-job-dev-image/pytorch_1_8:train-pytorch_1.8.0-cuda_10.2-py_3.7-euleros_2.10.1-x86_64-8.1.1`
Code Directory	Caminho do OBS para armazenar o código de treinamento. Este parâmetro é opcional. Tome o caminho do OBS **obs://obs-bucket/training-test/demo-code** como um exemplo. O conteúdo no caminho do OBS será baixado automaticamente para **${MA_JOB_DIR}/demo-code** no contêiner de treinamento, e **demo-code** (personalizável) é o diretório de último nível do caminho do OBS.
Boot Command	Comando para inicializar uma imagem. Este parâmetro é obrigatório. O comando de inicialização será executado automaticamente após o download do diretório de código. ● Se o script de inicialização de treinamento for um arquivo .py, **train.py**, por exemplo, o comando de inicialização pode ser **python ${MA_JOB_DIR}/demo-code/train.py**. ● Se o script de inicialização de treinamento for um arquivo .sh, **main.sh**, por exemplo, o comando de inicialização pode ser **bash ${MA_JOB_DIR}/demo-code/main.sh**. Ponto e vírgula (;) e e comercial (&&) podem ser usados para combinar vários comandos de inicialização, mas não há suporte para quebras de linha. **demo-code** (personalizável) no comando de inicialização é o diretório de último nível do caminho do OBS.

Configuring Pipelines

Um algoritmo predefinido baseado em imagem obtém dados de um bucket ou conjunto de dados do OBS para treinamento do modelo. A saída de treinamento é armazenada em um bucket do OBS. Os parâmetros de entrada e saída no código do algoritmo devem ser analisados para permitir a troca de dados entre ModelArts e OBS. Para obter detalhes sobre como desenvolver código para treinamento em ModelArts, consulte **Desenvolvimento de um script personalizado**.

Ao usar uma imagem predefinida para criar um algoritmo, configure os pipelines de entrada e saída.

- Configurações de entrada

  Tabela 4-8 Configurações de entrada

Parâmetro	Descrição
Parameter Name	Defina o nome com base no parâmetro de entrada de dados no código do algoritmo. O parâmetro caminho do código deve ser o mesmo que o parâmetro de entrada de treinamento analisado no código do algoritmo. Caso contrário, o código do algoritmo não pode obter os dados de entrada. Por exemplo, se você usar **argparse** no código do algoritmo para analisar **data_url** na entrada de dados, defina o parâmetro de entrada de dados como **data_url** ao criar o algoritmo.
Description	Descrição personalizável do parâmetro de entrada,
Obtained from	Fonte do parâmetro de entrada. É possível selecionar **Hyperparameters** (padrão) ou **Environment variables**.
Constraints	Se os dados são obtidos de um caminho de armazenamento ou de um conjunto de dados do ModelArts. Se você selecionar o conjunto de dados do ModelArts como a fonte de dados, as seguintes restrições serão adicionadas: ● **Labeling Type**: para obter detalhes, consulte **Criação de um trabalho de rotulagem**. ● **Data Format**, que pode ser **Default**, **CarbonData** ou ambos. **Default** indica o formato do manifesto. ● **Data Segmentation**: disponível apenas para classificação de imagens, detecção de objetos, classificação de texto e conjuntos de dados de classificação de som. Os valores possíveis são **Segmented dataset**, **Dataset not segmented** e **Unlimited**. Para obter detalhes, consulte **Publicação de uma versão de dados**.
Add	Várias fontes de entrada de dados são permitidas.

- Configurações de saída

Tabela 4-9 Configurações de saída

Parâmetro	Descrição
Parameter Name	Defina o nome com base no parâmetro de saída de dados no código do algoritmo. O parâmetro caminho do código deve ser o mesmo que o parâmetro de saída do treinamento analisado no código do algoritmo. Caso contrário, o código do algoritmo não pode obter o caminho de saída. Por exemplo, se você usar **argparse** no código do algoritmo para analisar **train_url** na saída de dados, defina o parâmetro de saída de dados como **train_url** ao criar o algoritmo.
Description	Descrição personalizável do parâmetro de saída,
Obtained from	Fonte do parâmetro de saída. É possível selecionar **Hyperparameters** (padrão) ou **Environment variables**.
Add	Vários caminhos de saída de dados são permitidos.

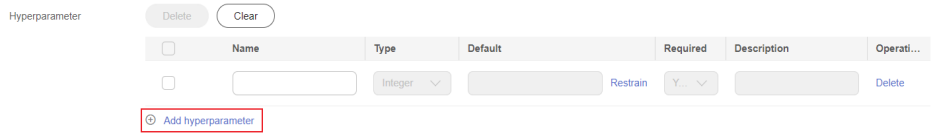
Definir hiperparâmetros

Quando você usa uma imagem predefinida para criar um algoritmo, o ModelArts permite personalizar hiperparâmetros para visualizá-los ou modificá-los a qualquer momento. Depois que os hiperparâmetros são definidos, eles são exibidos no comando de inicialização e transferidos para o arquivo de inicialização como parâmetros da CLI.

1. Importe os hiperparâmetros.

   Você pode clicar em **Add hyperparameter** para adicionar manualmente hiperparâmetros.

   Figura 4-33 Adicionar hiperparâmetros

2. Edite os hiperparâmetros.

   Para mais detalhes, consulte **Tabela 4-10**.

   Tabela 4-10 Hiperparâmetros

Parâmetro	Descrição
Name	Nome do hiperparâmetro Insira de 1 a 64 caracteres. Somente letras, dígitos, hifens (-) e sublinhados (_) são permitidos.
Type	Tipo do hiperparâmetro, que pode ser **String**, **Integer**, **Float** ou **Boolean**

Parâmetro	Descrição
Default	Valor padrão do hiperparâmetro, que é usado para trabalhos de treinamento por padrão
Constraints	Clique em **Restrain**. Em seguida, defina o intervalo do valor padrão ou valor enumerado na caixa de diálogo exibida.
Required	Selecione **Yes** ou **No**. ● Se você selecionar **No**, poderá excluir o hiperparâmetro na página de criação de trabalho de treinamento ao usar esse algoritmo para criar um trabalho de treinamento. ● Se você selecionar **Yes**, não será possível excluir o hiperparâmetro na página de criação de trabalho de treinamento ao usar esse algoritmo para criar um trabalho de treinamento.
Description	Descrição do hiperparâmetro Somente letras, dígitos, espaços, hifens (-), sublinhados (_), vírgulas (,) e pontos (.) são permitidos.

Adicionar restrições de treinamento

Você pode adicionar restrições de treinamento do algoritmo com base em suas necessidades.

- **Resource Type**: selecione os tipos de recursos necessários.
- **Multicard Training**: escolha se deseja oferecer suporte ao treinamento com vários cartões.
- **Distributed Training**: escolha se deseja oferecer suporte ao treinamento distribuído.

Visualização do ambiente de tempo de execução

Ao criar um algoritmo, clique na seta em 

no canto inferior direito da página para saber o caminho do diretório de código, do arquivo de inicialização e dos dados de entrada e saída no contêiner de treinamento.

Procedimento de acompanhamento

Depois que um algoritmo é criado, use-o para criar um trabalho de treinamento. Para mais detalhes, consulte **Uso de uma imagem personalizada para criar um trabalho de treinamento baseado em CPU ou GPU**.

4.5 Uso de uma imagem personalizada para criar um trabalho de treinamento baseado em CPU ou GPU

O treinamento do modelo é um processo iterativo de otimização. Por meio do gerenciamento unificado de treinamento, você pode selecionar de forma flexível algoritmos, dados e hiperparâmetros para obter a configuração e o modelo de entrada ideais. Depois de comparar

as métricas entre as versões do trabalho, você pode determinar o trabalho de treinamento mais satisfatório.

Pré-requisitos

- Os dados a serem treinados foram carregados em um diretório do OBS.
- Pelo menos uma pasta vazia para armazenar a saída de treinamento foi criada no OBS.
- Uma imagem personalizada foi criada com base nas especificações do ModelArts. Para obter detalhes sobre as especificações de imagem personalizada, consulte **Especificações para imagens personalizadas para trabalhos de treinamento**.
- A imagem personalizada foi carregada no SWR. Para mais detalhes, consulte **Como acessar o SWR e carregar imagens para ele?**.

Criar um trabalho de treinamento

1. Faça logon no console de gerenciamento do ModelArts. No painel de navegação à esquerda, escolha **Training Management** > **Training Jobs**.
2. Clique em **Create Training Job** e defina parâmetros.

Tabela 4-11 Criar um trabalho de treinamento usando uma imagem personalizada

Parâmetro	Descrição
Algorithm Type	Selecione **Custom algorithm**. Este parâmetro é obrigatório.
Boot Mode	Selecione **Custom image**. Este parâmetro é obrigatório.
Image	Caminho da imagem do contêiner. Este parâmetro é obrigatório. Você pode definir o caminho da imagem do contêiner de uma das seguintes maneiras: ● Para selecionar sua imagem ou uma imagem compartilhada por outras pessoas, clique em **Select** à direita e selecione uma imagem de contêiner para treinamento. A imagem necessária deve ser carregada no SWR com antecedência. ● Para selecionar uma imagem pública, insira o endereço da imagem pública no SWR. Digite o caminho da imagem no formato "Organization name/Image name:Version name". Não contenha o nome de domínio (swr.<region>.xxx.com) no caminho porque o sistema adicionará automaticamente o nome de domínio ao caminho. Por exemplo, se o endereço do SWR de uma imagem pública for **swr.<region>.xxx.com/test-image/tensorflow2_1_1:1.1.1**, digite **test-images/tensorflow2_1_1:1.1.1**.

Parâmetro	Descrição
Code Directory	Selecione o diretório do OBS onde o arquivo de código de treinamento está armazenado. Se a imagem personalizada não contiver código de treinamento, você precisará definir esse parâmetro. Se a imagem personalizada contiver código de treinamento, você não precisa definir esse parâmetro. ● Carregue o código para o bucket do OBS antecipadamente. O tamanho total dos arquivos no diretório não pode exceder 5 GB, o número de arquivos não pode exceder 1000 e a profundidade da pasta não pode exceder 32. ● O arquivo de código de treinamento é baixado automaticamente para o diretório **${MA_JOB_DIR}/demo-code** do contêiner de treinamento quando o trabalho de treinamento é iniciado. **demo-code** é o diretório do OBS de último nível para armazenar o código. Por exemplo, se **Code Directory** estiver definido como **/test/code**, o arquivo de código de treinamento será baixado para o diretório **${MA_JOB_DIR}/code** do contêiner de treinamento.
User ID	ID de usuário para executar o contêiner. Recomenda-se o valor padrão 1000. Se o UID precisa ser especificado, seu valor deve estar dentro do intervalo especificado. Os intervalos de UID de diferentes pools de recursos são os seguintes: ● Pool de recursos públicos: 1000 a 65535 ● Pool de recursos dedicados: 0 a 65535

Boot Command	Comando para inicializar uma imagem. Este parâmetro é obrigatório.
	Quando um trabalho de treinamento está em execução, o comando de inicialização é executado automaticamente após o download do diretório de código.
	● Se o script de inicialização de treinamento for um arquivo .py, **train.py**, por exemplo, o comando de inicialização é o seguinte.
	`python ${MA_JOB_DIR}/demo-code/train.py`
	● Se o script de inicialização de treinamento for um arquivo .sh, **main.sh**, por exemplo, o comando de inicialização será o seguinte.
	`bash ${MA_JOB_DIR}/demo-code/main.sh`
	Você pode usar ponto-e-vírgula (;) e E comercial (&&) para combinar vários comandos. **demo-code** no comando é o diretório do OBS de último nível onde o código é armazenado. Substitua-o pelo real.

Parâmetro	Descrição
Local Code Directory	Especifique o diretório local de um contêiner de treinamento. Quando um treinamento é iniciado, o sistema baixa automaticamente o diretório de código para esse diretório. O diretório de código local padrão é **/home/ma-user/modelarts/user-job-dir**. Este parâmetro é opcional.
Work Directory	Durante o treinamento, o sistema executa automaticamente o comando **cd** para executar o arquivo de inicialização neste diretório.

Tabela 4-12 Parâmetros para criar um trabalho de treinamento

Parâmetro	Sub parâmetro	Descrição

Input	Parameter	O código do algoritmo lê os dados de entrada de treinamento com base no nome do parâmetro de entrada.
		Defina esse parâmetro como **data_url**, que é o mesmo que o parâmetro para analisar os dados de entrada no código de treinamento. Você pode definir vários parâmetros de entrada de treinamento. O nome de cada parâmetro de entrada de treinamento deve ser exclusivo.
		Por exemplo, se você usar **argparse** no código de treinamento para analisar **data_url** na entrada de dados, defina o nome do parâmetro da entrada de treinamento como **data_url**.
		`import argparse`
		`# Create a parsing task.`
		`parser = argparse.ArgumentParser(description="train mnist", formatter_class=argparse.ArgumentDefaultsHelpFormatter)`
		`# Add parameters.`
		`parser.add_argument('--train_url', type=str, help='the path model saved')`
		`parser.add_argument('--data_url', type=str, help='the training data')`
		`# Parse the parameters.`
		`args, unknown = parser.parse_known_args()`

Parâmetro	Sub parâmetro	Descrição
	Dataset	Clique em **Dataset** e selecione o conjunto de dados de destino e sua versão na lista de conjuntos de dados do ModelArts. Quando o trabalho de treinamento é iniciado, o ModelArts baixa automaticamente os dados no caminho de entrada para o contêiner de treinamento. **NOTA** O gerenciamento de dados do ModelArts está sendo atualizado e é invisível para usuários que não usaram o gerenciamento de dados. Recomenda-se que os novos usuários armazenem seus dados de treinamento em buckets do OBS.
	Data path	Clique em **Data path** e selecione o caminho de armazenamento para os dados de entrada de treinamento de um bucket do OBS. Quando o trabalho de treinamento é iniciado, o ModelArts baixa automaticamente os dados no caminho de entrada para o contêiner de treinamento.

	How to Obtain	O seguinte usa a entrada de treinamento **data_path** como um exemplo.
		● Se você selecionar **Hyperparameters**, use este código para obter os dados:
		`import argparse`
		`parser = argparse.ArgumentParser()`
		`parser.add_argument('--data_path')`
		`args, unknown = parser.parse_known_args()`
		`data_path = args.data_path`
		● Se você selecionar **Environment variables**, use este código para obter os dados:
		`import os`
		`data_path = os.getenv("data_path", "")`

Output	Parameter	O código do algoritmo lê os dados de saída de treinamento com base no nome do parâmetro de saída. Defina esse parâmetro como **train_url**, que é o mesmo que o parâmetro para analisar os dados de saída no código de treinamento. Você pode definir vários parâmetros de saída de treinamento. O nome de cada parâmetro de saída de treinamento deve ser exclusivo.

Parâmetro	Sub parâmetro	Descrição
	Data path	Clique em **Data path** e selecione o caminho de armazenamento para os dados de saída de treinamento de um bucket do OBS. Durante o treinamento, o sistema sincroniza automaticamente os arquivos do diretório de código local do contêiner de treinamento com o caminho dos dados. **NOTA** O caminho de dados só pode ser um caminho do OBS. Para evitar problemas com o armazenamento de dados, escolha um diretório vazio como caminho de dados.

	How to Obtain	O seguinte usa a saída de treinamento **train_url** como um exemplo.
		● Se você selecionar **Hyperparameters**, use este código para obter os dados:
		`import argparse`
		`parser = argparse.ArgumentParser()`
		`parser.add_argument('--train_url')`
		`args, unknown = parser.parse_known_args()`
		`train_url = args.train_url`
		● Se você selecionar **Environment variables**, use este código para obter os dados:
		`import os`
		`train_url = os.getenv("train_url", "")`

	Predownload	Indica se deve ser feito o download prévio dos arquivos no diretório de saída para um diretório local. ● Se você definir **Predownload** como **No**, o sistema não fará o download dos arquivos no caminho de dados de saída do treinamento para um diretório local do contêiner de treinamento quando o trabalho de treinamento for iniciado. ● Se você definir **Predownload** como **Yes**, o sistema fará download automático dos arquivos no caminho de dados de saída de treinamento para um diretório local do contêiner de treinamento quando o trabalho de treinamento for iniciado. Quanto maior o tamanho do arquivo, maior o tempo de download. Para evitar excesso de tempo de treinamento, remova todos os arquivos desnecessários do diretório de código local do contêiner de treinamento o mais rápido possível. Para usar **treinamento resumível e treinamento incremental**, **Download** deve ser selecionado.
Hyperparameters	-	Usado para ajuste de treinamento. Este parâmetro é opcional.

Parâmetro	Sub parâmetro	Descrição
Environment Variable	-	Adicione variáveis de ambiente com base nos requisitos de serviço. Para obter detalhes sobre variáveis de ambiente predefinidas no contêiner de treinamento, consulte **Visualização de variáveis de ambiente de um contêiner de treinamento**.
Auto Restart	-	Número de tentativas para um trabalho de treinamento com falha. Se esse parâmetro estiver habilitado, um trabalho de treinamento com falha será automaticamente entregue novamente e executado. Na página de detalhes do trabalho de treinamento, você pode exibir o número de tentativas de um trabalho de treinamento com falha. ● Esta função está desativada por padrão. ● Se você ativar essa função, defina o número de tentativas. O valor varia de **1** a **3** e não pode ser alterado.

3. Selecione um flavor de instância. O intervalo de valores dos parâmetros de treinamento é consistente com as restrições das imagens personalizadas existentes. Selecione um pool de recursos públicos ou um pool de recursos dedicados, conforme necessário. Para obter detalhes sobre os parâmetros, consulte **Criação de um trabalho de treinamento**.
4. Clique em **Submit** para criar o trabalho de treinamento.

   Leva um período de tempo para criar um trabalho de treinamento.

   Para exibir o status em tempo real de um trabalho de treinamento, vá para a lista de trabalhos de treinamento e clique no nome do trabalho de treinamento. Na página de detalhes do trabalho de treinamento exibida, exiba as informações básicas do trabalho de treinamento. Para obter detalhes, consulte **Exibição de detalhes do trabalho de treinamento**.

4.6 Processo de solução de problemas

Sintoma

Falhou em um trabalho de treinamento usando uma imagem personalizada.

Método de localização

1. Determine a fonte da imagem.
   - Verifique se a imagem de base da imagem personalizada é do ModelArts. Use uma imagem de base fornecida pelo ModelArts para criar uma imagem personalizada. Para obter detalhes, consulte **Uso de uma imagem de base para criar uma imagem de treinamento**.
   - Se a imagem for de um terceiro, verifique com o criador da imagem personalizada como usar essa imagem.

2. Determine o tamanho da imagem personalizada.

   Não utilize uma imagem personalizada com mais de 15 GB. O tamanho não deve exceder metade do espaço do mecanismo de contêiner do pool de recursos. Caso contrário, a hora de início do trabalho de treinamento será afetada.

   O espaço do mecanismo de contêiner do pool de recursos públicos do ModelArts é de 50 GB. Por padrão, o espaço do mecanismo de contêiner do pool de recursos dedicados também é de 50 GB. Você pode personalizar o espaço do mecanismo de contêiner ao criar um pool de recursos dedicados.

3. Determine o tipo de erro.
   - Se uma mensagem de erro for exibida indicando que não foi possível encontrar um arquivo, consulte **Mensagem de erro "No such file or directory" exibida em logs do trabalho de treinamento**.
   - Se uma mensagem de erro for exibida indicando que um pacote não pôde ser encontrado, consulte **Mensagem de erro "No module named .*" exibida em logs do trabalho de treinamento**.
   - Ocorreu um erro no script de inicialização do Ascend ou no script de inicialização.

     Verifique se o script é obtido a partir do site oficial e se o script é usado estritamente seguindo as instruções fornecidas em documentos oficiais. Por exemplo, verifique se o nome e o caminho do script estão corretos.
   - A versão do driver é incompatível com o driver subjacente.

     Antes de atualizar o driver de uma imagem personalizada, verifique se a versão atualizada é suportada pelo driver subjacente. **Obtenha as versões de driver suportadas**.
   - Você não tem permissão para acessar um arquivo.

     A causa possível é que o usuário da imagem personalizada é diferente daquele do contêiner de trabalho. Nesse caso, modifique o Dockerfile.

     ```
     RUN if id -u ma-user > /dev/null 2>&1 ; \
     then echo 'The ModelArts user already exists.' ; \
     else echo 'The ModelArts user does not exist.' && \
     groupadd ma-group -g 1000 && \
     useradd -d /home/ma-user -m -u 1000 -g 1000 -s /bin/bash ma-user ; fi && \
     \
     chmod 770 /home/ma-user && \
     chmod 770 /root && \
     usermod -a -G root ma-user
     ```
   - Para outros problemas, procure soluções em **casos de falha de treinamento**.

Resumo e sugestões

Antes de usar uma imagem personalizada para trabalhos de treinamento, crie a imagem seguindo as **especificações de imagem personalizadas** que também fornece exemplos de ponta a ponta para sua referência.

5 Uso de uma imagem personalizada para criar aplicações de IA para implementação de inferência

5.1 Especificações de imagem personalizada para criar aplicações de IA

Ao criar uma imagem personalizada usando um modelo desenvolvido localmente, certifique-se de que a imagem esteja em conformidade com as especificações do ModelArts.

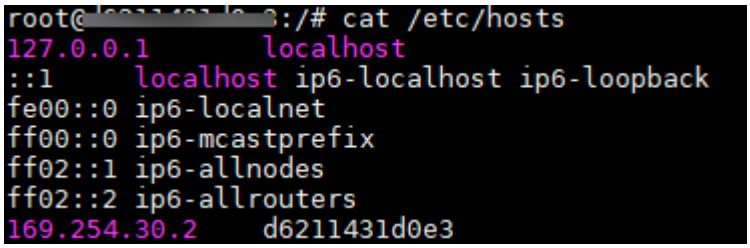
- Nenhum código malicioso é permitido.
- Uma imagem personalizada não pode ter mais de 50 GB.
- **APIs externas**

  Defina a API de serviço externo para uma imagem personalizada. A API de inferência deve ser a mesma que o URL definido pelo **apis** em **config.json**. Em seguida, a API de serviço externo pode ser acessada diretamente quando a imagem é iniciada. Segue-se um exemplo de acesso a uma imagem de MNIST. A imagem contém um modelo treinado usando um conjunto de dados de MNIST e pode identificar dígitos manuscritos. **listen_ip** indica o endereço IP do contêiner. Você pode iniciar uma imagem personalizada para obter o endereço IP do contêiner.

  – Exemplo de solicitação

  ```
  curl -X POST \ http://{Listening IP address}:8080/ \ -F images=@seven.jpg
  ```

  Figura 5-1 Exemplo de obtenção de **listen_ip**

  – Exemplo de resposta

  ```
  {"mnist_result": 7}
  ```


- **(Opcional) API de verificação de integridade**

  Se os serviços não forem interrompidos durante uma atualização contínua, a API de verificação de integridade deve ser configurada em **config.json** para ModelArts. A API de verificação de integridade retorna o estado de integridade de um serviço quando o serviço está sendo executado corretamente ou um erro quando o serviço se torna defeituoso.

  AVISO

  A API de verificação de integridade deve ser configurada para uma atualização contínua sem acertos.

  A seguir, mostramos uma API de verificação de integridade de exemplo:

  – URI
  ```
  GET /health
  ```
  – Exemplo de solicitação: curl -X GET \ http://*{Listening IP address}*:8080/health
  – Exemplo de resposta
  ```
  {"health": "true"}
  ```
  – Código de status

  Tabela 5-1 Código de status

Código de status	Mensagem	Descrição
200	OK	Solicitação enviada.

- **Saída do arquivo de log**

  Configure a saída padrão para que os logs possam ser exibidos corretamente.

- **Arquivo de inicialização de imagem**

  Para implementar um serviço em lote, defina o arquivo de inicialização de uma imagem como **/home/run.sh** e use CMD para definir o caminho de inicialização padrão. A seguir está um exemplo de Dockerfile:

  CMD ["sh", "/home/run.sh"]

- **Dependências de imagem**

  Para implementar um serviço em lote, instale pacotes de dependência como Python, JRE/JDK e ZIP na imagem.

- **(Opcional) Atualização contínua sem acertos**

  Para garantir que os serviços não sejam interrompidos durante uma atualização contínua, defina HTTP **keep-alive** como **200**. Por exemplo, Gunicorn não suporta Keep-alive por padrão. Para garantir uma atualização contínua sem acertos, instale Gevent e configure **--keep-alive 200 -k gevent** na imagem. As configurações de parâmetros variam dependendo da estrutura de serviço. Defina os parâmetros conforme necessário.

- **(Opcional) Saída graciosa de um contêiner**

  Para garantir que os serviços não sejam interrompidos durante uma atualização contínua, o sistema deve capturar sinais SIGTERM no contêiner e aguardar 60s antes de sair do contêiner. Se a duração for inferior a 60s antes da saída graciosa, os serviços podem ser interrompidos durante a atualização rolante. Para garantir a execução ininterrupta do

serviço, o sistema sai do contêiner depois que o sistema recebe sinais SIGTERM e processa todas as solicitações recebidas. Toda a duração não é superior a 90s. A seguir, mostra o exemplo **run.sh**:

```
#!/bin/bash
gunicorn_pid=""

handle_sigterm() {
  echo "Received SIGTERM, send SIGTERM to $gunicorn_pid"
  if [ $gunicorn_pid != "" ]; then
      sleep 60
      kill -15 $gunicorn_pid  # Transfer SIGTERM signals to the Gunicorn
process.
      wait $gunicorn_pid           # Wait until the Gunicorn process stops.
  fi
}

trap handle_sigterm TERM
```

5.2 Criação de uma imagem personalizada e uso dela para criar uma aplicação de IA

Se você quiser usar um mecanismo de IA não suportado pelo ModelArts crie uma imagem personalizada para o mecanismo, importe a imagem para ModelArts e use a imagem para criar aplicações de IA. Esta seção descreve como usar uma imagem personalizada para criar uma aplicação de IA e implementar a aplicação como um serviço em tempo real.

O processo é o seguinte:

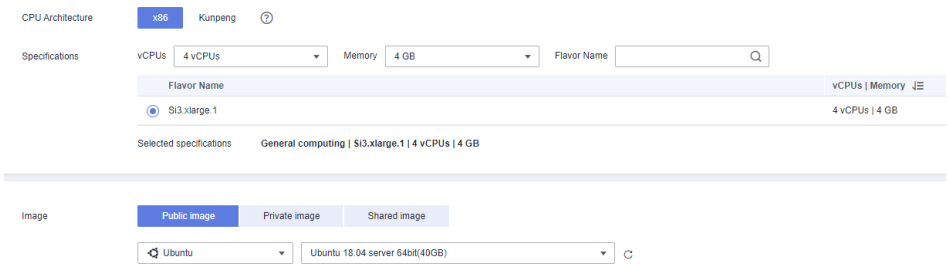
1. **Construir uma imagem localmente**: crie um pacote de imagem personalizado localmente. Para obter detalhes, consulte **Especificações de imagens personalizadas para a criação de uma aplicação de IA**.
2. **Verificar a imagem localmente e carregá-la para o SWR**: verifique as APIs da imagem personalizada e carregue a imagem personalizada para o SWR.
3. **Usar a imagem personalizada para criar uma aplicação de IA**: importe a imagem para o gerenciamento de aplicações de IA da ModelArts.
4. **Implementar a aplicação de IA como um serviço em tempo real**: implemente o modelo como um serviço em tempo real.

Construir uma imagem localmente

Esta seção usa um host de Linux x86_x64 como exemplo. Você pode comprar um ECS com as mesmas especificações ou usar um host local existente para criar uma imagem personalizada.

Para obter detalhes sobre como adquirir um ECS, consulte **Compra e logon em um ECS Linux**. Ao criar o ECS, selecione uma imagem pública do Ubuntu 18.04.

Figura 5-2 Criar um ECS usando uma imagem pública x86



1. Após fazer logon no host, instale o Docker. Para obter detalhes, consulte **documentos oficiais do Docker**. Como alternativa, execute os seguintes comandos para instalar o Docker:

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

2. Obtenha a imagem de base. Ubuntu 18.04 é usado neste exemplo.

   ```
   docker pull ubuntu:18.04
   ```

3. Crie a pasta **self-define-images** e edite **Dockerfile** e **test_app.py** na pasta da imagem personalizada. No código de exemplo, o código da aplicação é executado na estrutura do Flask.

   A estrutura do arquivo é a seguinte:

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

   - **Dockerfile**

     ```
     From ubuntu:18.04
     # Configure the HUAWEI CLOUD source and install Python, Python3-PIP, and
     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

     # Copy the application code to the image.
     COPY test_app.py /opt/test_app.py

     # Specify the boot command of the image.
     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():
     ```



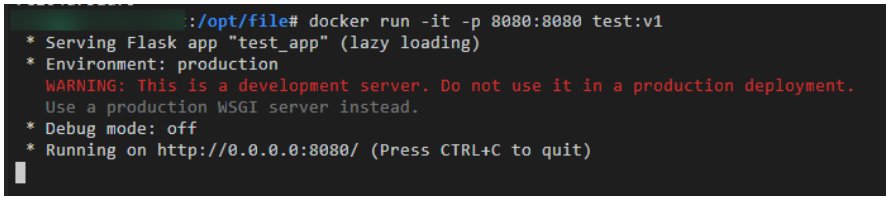
```
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@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. Alterne para a pasta **self-define-images** e execute o seguinte comando para criar uma imagem personalizada **test:v1**:
   ```
   docker build -t test:v1 .
   ```
5. Execute **docker images** para exibir a imagem personalizada que você criou.

Verificar a imagem localmente e carregá-la para o SWR

1. Execute o seguinte comando no ambiente local para iniciar a imagem personalizada:
   ```
   docker run -it -p 8080:8080 test:v1
   ```

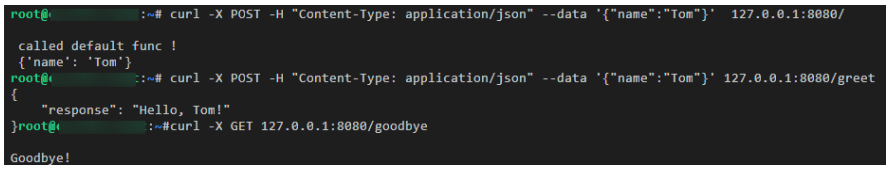
   Figura 5-3 Iniciar uma imagem personalizada

2. Abra outro terminal e execute os seguintes comandos para testar as funções das três APIs da imagem personalizada:
   ```
   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
   ```

   Se forem exibidas informações semelhantes às seguintes, a verificação da função é bem-sucedida.

   Figura 5-4 Testar funções da API

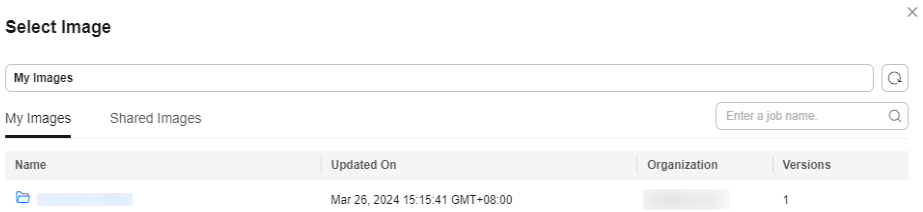
3. Carregue a imagem personalizada para o SWR. Para obter detalhes, consulte **Como carregar imagens para o SWR?**
4. Veja a imagem carregada na página **My Images** > **Private Images** do console do SWR.

Usar a imagem personalizada para criar uma aplicação de IA

Importe um metamodelo. Para obter detalhes, consulte **Criação e importação de uma imagem de modelo**. Os parâmetros principais são os seguintes:

- **Meta Model Source**: selecione **Container image**.
  - **Container Image Path**: selecione a imagem privada criada.

    Figura 5-5 Imagem privada criada

  - **Container API**: protocolo e número de porta para iniciar um modelo. Certifique-se de que o protocolo e o número da porta sejam os mesmos fornecidos na imagem personalizada.
  - **Image Replication**: indica se a imagem do modelo deve ser copiada para ModelArts na imagem do contêiner. Este parâmetro é opcional.
  - **Health Check**: verifica o estado de saúde de um modelo. Este parâmetro é opcional. Esse parâmetro é configurável somente quando a API de verificação de integridade é configurada na imagem personalizada. Caso contrário, a criação da aplicação de IA falhará.
- **APIs**: APIs de uma imagem personalizada. Este parâmetro é opcional. As APIs do modelo devem estar em conformidade com as especificações do ModelArts. Para obter detalhes, consulte **Especificações para editar um arquivo de configuração de modelo**.

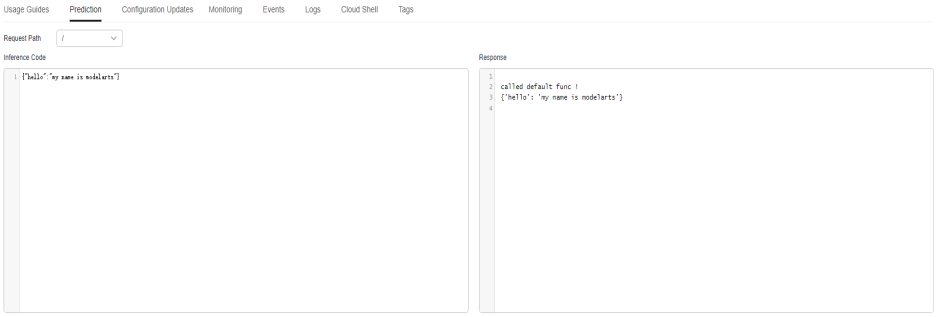
  O arquivo de configuração é o seguinte:

```
[{
        "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"
        }
    }
]
```


Implementar a aplicação de IA como um serviço em tempo real

1. Implemente a aplicação de IA como um serviço em tempo real. Para obter detalhes, consulte **Implementação como serviço em tempo real**.
2. Veja os detalhes sobre o serviço em tempo real.
3. Acesse o serviço em tempo real na página de guia **Prediction**.

Figura 5-6 Acessar um serviço em tempo real



6 Perguntas frequentes

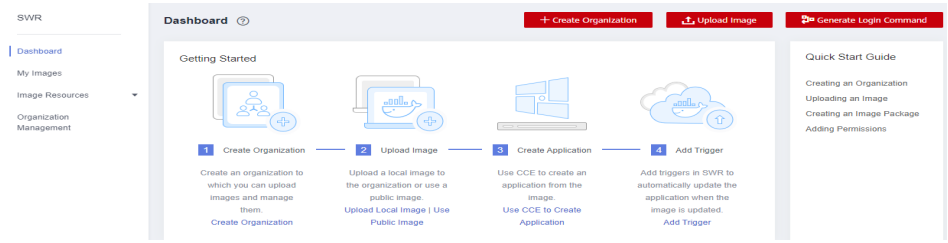
6.1 Como acessar o SWR e carregar imagens para ele?

Esta seção descreve como fazer logon no SWR e carregar imagens para ele.

Etapa 1 Efetue logon no SWR

1. Faça logon no console do SWR e selecione a região de destino.

   Figura 6-1 Console do SWR

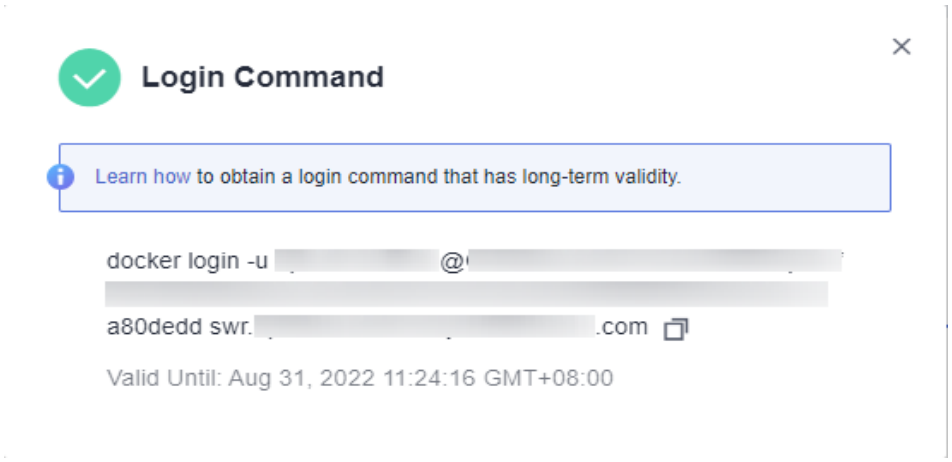
   

2. Clique em **Create Organization** no canto superior direito e insira um nome de organização para criar uma organização. **deep-learning** é usado como exemplo. Substitua-o nos comandos subsequentes pelo nome real da organização.

Figura 6-2 Criar uma organização

3. Clique em **Generate Login Command** no canto superior direito para obter um comando de logon.

Figura 6-3 Comando de logon



4. Efetue logon no ECS como usuário **root** e insira o comando logon.

Figura 6-4 Comando de logon executado no ECS

Etapa 2 Carregar imagens para o SWR

Esta seção descreve como carregar uma imagem para o SWR.

1. Faça logon no SWR e marque a imagem a ser carregada. Substitua o nome da organização **deep-learning** no comando a seguir pelo nome real da organização obtido na etapa 1.
```
sudo docker tag tf-1.13.2:latest swr.xxx.com/deep-learning/tf-1.13.2:latest
```
2. Execute o seguinte comando para carregar a imagem:
```
sudo docker push swr.xxx.com/deep-learning/tf-1.13.2:latest
```


Figura 6-5 Carregar uma imagem

3. Depois que a imagem for carregada, escolha **My Images** no painel de navegação à esquerda do console do SWR para exibir as imagens personalizadas carregadas.

Figura 6-6 Imagem personalizada carregada

swr.xxx.com/deep-learning/tf-1.13.2:latest é o URL do SWR da imagem personalizada.

6.2 Como configurar variáveis de ambiente para uma imagem?

Em um Dockerfile, use a instrução ENV para configurar variáveis de ambiente. Para obter detalhes, consulte **Referência do Dockerfile**.

6.3 Como usar o Docker para iniciar uma imagem salva usando uma instância de notebook?

Uma imagem salva usando uma instância de notebook contém o parâmetro **Entrypoint**, como mostrado em **Entrypoint**. O arquivo executável ou o comando especificado no parâmetro **Entrypoint** substitui o comando de inicialização padrão da imagem. A entrada de comando no parâmetro **Entrypoint** não está predefinida na imagem. Quando você executa **docker run** no ambiente local para iniciar a imagem, uma mensagem de erro é exibida, indicando que a tarefa de criação do contêiner falha porque o arquivo ou diretório de inicialização não foi encontrado, conforme mostrado em **Figura 6-8**

Para evitar esse erro, configure o parâmetro **--entrypoint** para substituir o programa especificado em **Entrypoint**. Use o arquivo de inicialização ou o comando especificado pelo parâmetro **--entrypoint** para iniciar a imagem. Exemplo:


```
docker run -it -d --entrypoint /bin/bash image:tag
```

Figura 6-7 Entrypoint

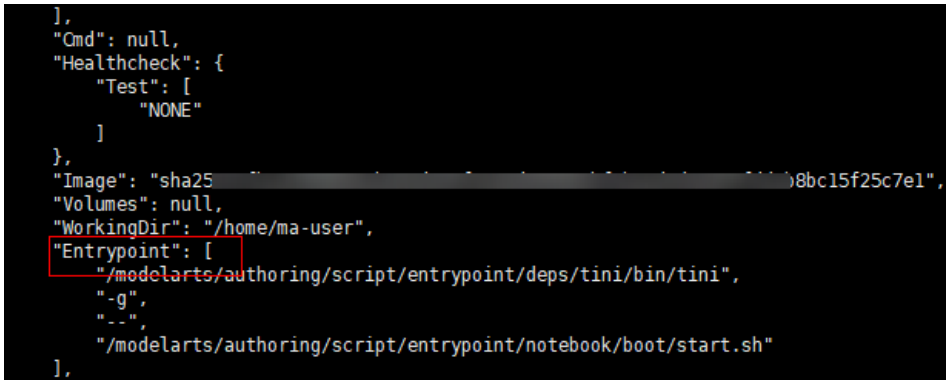
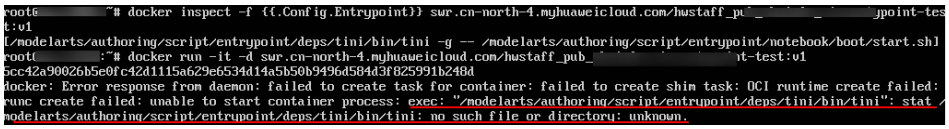


Figura 6-8 Erro relatado quando uma imagem está sendo iniciada



6.4 Como configurar uma fonte de Conda em um ambiente de desenvolvimento de notebook?

Você pode instalar as dependências de desenvolvimento no Notebook conforme necessário. As ferramentas de gerenciamento de pacotes pip e Conda podem ser usadas para instalar dependências regulares. A fonte pip foi configurada e pode ser usada para instalação, enquanto a fonte de Conda requer configuração adicional.

Esta seção descreve como configurar a fonte de Conda em uma instância de notebook.

Configurar a fonte de Conda

O software Conda foi pré-definido em imagens. For details, see **https://mirror.tuna.tsinghua.edu.cn/help/anaconda/**.

Comandos comuns de Conda

Para obter detalhes sobre todos os comandos do Conda, consulte **os documentos oficiais do Conda**. A tabela a seguir lista apenas comandos comuns.

Tabela 6-1 Comandos comuns de Conda

Descrição	Comando
Obter ajuda on-line.	`conda --help` `conda update --help # Obtain help for a command, for example, update.`

Descrição	Comando
Ver a versão de Conda.	`conda -V`
Atualizar Conda.	`conda update conda  # Update Conda.` `conda update anaconda # Update Anaconda.`

Gerenciar ambientes.	`conda env list  # Show all virtual environments.`
	`conda info -e # Show all virtual environments.`
	`conda create -n myenv python=3.7 # Create an environment named myenv with Python version 3.7.`
	`conda activate myenv  # Activate the myenv environment.`
	`conda deactivate  # Disable the current environment.`
	`conda remove -n myenv --all # Delete the myenv environment.`
	`conda create -n newname --clone oldname # Clone the old environment to the new environment.`

Gerenciar pacotes.	`conda list  # Check the packages that have been installed in the current environment.`
	`conda list  -n myenv  # Specify the packages installed in the myenv environment.`
	`conda search numpy # Obtain all information of the numpy package.`
	`conda search numpy=1.12.0 --info  # View the information of NumPy 1.12.0.`
	`conda install numpy pandas  # Concurrently install the NumPy and Pandas packages.`
	`conda install numpy=1.12.0  # Install NumPy of a specified version.`
	`# The install, update, and remove commands use -n to specify an environment, and the install and update commands use -c to specify a source address.`
	`conda install -n myenv numpy  # Install the numpy package in the myenv environment.`
	`conda install -c https://conda.anaconda.org/anaconda numpy  # Install NumPy using https://conda.anaconda.org/anaconda.`
	`conda update numpy pandas   # Concurrently update the NumPy and Pandas packages.`
	`conda remove numpy pandas   # Concurrently uninstall the NumPy and Pandas packages.`
	`conda update --all # Update all packages in the current environment.`

Limpar Conda.	`conda clean -p      # Delete useless packages.`
	`conda clean -t      # Delete compressed packages.`
	`conda clean -y --all # Delete all installation packages and clear caches.`

Salvar como uma imagem

Após instalar as bibliotecas externas, salve o ambiente usando a função de salvar imagem fornecida pelo notebook do ModelArts da nova versão. Você pode salvar uma instância de notebook em execução como uma imagem personalizada com um clique para uso futuro. Depois que os pacotes de dependência são instalados em uma instância de notebook, é uma boa prática salvar a instância como uma imagem para evitar que os pacotes de dependência sejam perdidos. Para mais detalhes, consulte **Salvamento de uma imagem de ambiente notebook**.

6.5 Quais são as versões de software suportadas para uma imagem personalizada?

Se sua imagem personalizada usa bibliotecas de software como NCCL, CUDA e OFED, verifique se as bibliotecas de software atendem aos seguintes requisitos de versão:

- NCCL 2.7.8 ou mais recente
- OFED MLNX_OFED_LINUX-5.4-3.1.0.0 ou mais recente
- A versão de CUDA precisa ser adaptada à versão do driver da GPU do pool de recursos dedicados. Para obter a versão do driver da GPU, acesse a página de detalhes do pool de recursos dedicados.

7 Histórico de modificações

Data de lançamento	Descrição
2023-10-01	Adição do seguinte conteúdo: ● Adição de **Início do treinamento com uma imagem predefinida**.
07/09/2023	Adição do seguinte conteúdo: ● Adição de **Quais são as versões de software suportadas para uma imagem personalizada?**. Modificação do seguinte conteúdo: ● Alteração do nome do manual de "Uso de imagens personalizadas" para "Gerenciamento de imagens". ● Ajuste do conteúdo em "Imagens predefinidas". Combinação do ambiente de desenvolvimento e das imagens de base de treinamento e inferência em **Uso de uma imagem predefinida**.

