ModelArts

Gestión de imágenes

Edición	01
Fecha	2024-09-14

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Marcas registradas y permisos

El logotipo HUAWEI y otras marcas registradas de Huawei pertenecen a Huawei Technologies Co., Ltd.
Todas las demás marcas registradas y los otros nombres comerciales mencionados en este documento son propiedad de sus respectivos titulares.

Aviso

Es posible que la totalidad o parte de los productos, las funcionalidades y/o los servicios que figuran en el presente documento no se encuentren dentro del alcance de un contrato vigente entre Huawei Cloud y el cliente. Las funcionalidades, los productos y los servicios adquiridos se limitan a los estipulados en el respectivo contrato. A menos que un contrato especifique lo contrario, ninguna de las afirmaciones, informaciones ni recomendaciones contenidas en el presente documento constituye garantía alguna, ni expresa ni implícita.

Huawei está permanentemente preocupada por la calidad de los contenidos de este documento; sin embargo, ninguna declaración, información ni recomendación aquí contenida constituye garantía alguna, ni expresa ni implícita. La información contenida en este documento se encuentra sujeta a cambios sin previo aviso.

Huawei Cloud Computing Technologies Co., Ltd.

Dirección: Huawei Cloud Data Center Jiaoxinggong Road
Avenida Qianzhong
Nuevo distrito de Gui'an
Gui Zhou, 550029
República Popular China

Sitio web: https://www.huaweicloud.com/intl/es-us/

Índice

1 Gestión de imágenes....1

2 Uso de una imagen preestablecida....4

2.1 Imágenes preestablecidas en notebook....4
2.1.1 Imágenes de base de notebook....4
2.1.2 Lista de imágenes base de notebook....5
2.1.3 Imagen de base de notebook con PyTorch (x86)....6
2.1.4 Imagen de base de notebook con Tensorflow (x86)....13
2.1.5 Imagen de base de notebook con MindSpore (x86)....17
2.1.6 Imagen de base de notebook con imagen dedicada personalizada (x86)....25
2.2 Imagen de base de entrenamiento....27
2.2.1 Imágenes de base de entrenamiento disponibles....28
2.2.2 Entrenamiento de imagen de base (PyTorch)....28
2.2.3 Entrenamiento de imagen de base (TensorFlow)....29
2.2.4 Entrenamiento de imagen de base (Horovod)....30
2.2.5 Entrenamiento de imagen de base (MPI)....32
2.2.6 Inicio del entrenamiento con una imagen preestablecida....32
2.2.6.1 PyTorch....32
2.2.6.2 TensorFlow....36
2.2.6.3 Horovod/MPI/MindSpore-GPU....39
2.3 Imágenes de base de inferencia....42
2.3.1 Imágenes de base de inferencia disponibles....42
2.3.2 Imágenes de base de inferencia con TensorFlow (CPU/GPU)....43
2.3.3 Imágenes base de inferencia con PyTorch (CPU/GPU)....48
2.3.4 Imágenes base de inferencia con MindSpore (CPU/GPU)....51

3 Uso de imágenes personalizadas en instancias de notebook....58

3.1 Registro de una imagen en ModelArts....58
3.2 Creación de una imagen personalizada....59
3.3 Guardar una instancia de Notebook como una imagen personalizada....60
3.3.1 Guardar una imagen de entorno de notebook....60
3.3.2 Uso de una imagen personalizada para crear una instancia de notebook....61
3.4 Creación y uso de una imagen personalizada en notebook....62
3.4.1 Escenarios de aplicación y proceso....62

3.4.2 Paso 1 Crear una imagen personalizada..... 62
3.4.3 Paso 2 Registrar una nueva imagen..... 64
3.4.4 Paso 3 Usar una nueva imagen para crear un entorno de desarrollo..... 65
3.5 Creación de una imagen personalizada en un ECS y su uso en notebook..... 66
3.5.1 Escenarios de aplicación y proceso..... 66
3.5.2 Paso 1 Preparar un servidor de Docker y configurar un entorno..... 66
3.5.3 Paso 2 Crear una imagen personalizada..... 67
3.5.4 Paso 3 Registrar una nueva imagen..... 71
3.5.5 Paso 5 Crear e iniciar un entorno de desarrollo..... 72

4 Uso de una imagen personalizada para entrenar modelos (entrenamiento de modelos) 74

4.1 Descripción general..... 74
4.2 Ejemplo: creación de una imagen personalizada para entrenamiento..... 77
4.2.1 Ejemplo: creación de una imagen personalizada para el entrenamiento (PyTorch + CPU/GPU)..... 77
4.2.2 Ejemplo: creación de una imagen personalizada para entrenamiento (MPI + CPU/GPU)..... 85
4.2.3 Ejemplo: creación de una imagen personalizada para entrenamiento (Horovod-PyTorch y GPU)..... 94
4.2.4 Ejemplo: creación de una imagen personalizada para entrenamiento (MindSpore y GPU)..... 106
4.2.5 Ejemplo: creación de una imagen personalizada para entrenamiento (TensorFlow y GPU)..... 116
4.3 Preparación de una imagen de entrenamiento..... 124
4.3.1 Especificaciones a las Imágenes personalizadas para trabajos de entrenamiento..... 124
4.3.2 Migración de una imagen al entrenamiento de ModelArts..... 125
4.3.3 Uso de una imagen de base para crear una imagen de entrenamiento..... 126
4.3.4 Instalación de MLNX_OFED en una imagen de contenedor..... 127
4.4 Creación de un algoritmo mediante una imagen personalizada..... 128
4.5 Uso de una imagen personalizada para crear un trabajo de entrenamiento basado en CPU o GPU..... 132
4.6 Proceso de solución de problemas..... 138

5 Uso de una imagen personalizada para crear aplicaciones de IA para el despliegue de inferencia..... 140

5.1 Especificaciones de imágenes personalizadas para crear aplicaciones de IA..... 140
5.2 Creación de una imagen personalizada y su uso para crear una aplicación de IA..... 142

6 Preguntas frecuentes..... 147

6.1 ¿Cómo puedo iniciar sesión en SWR y cargar imágenes en él?..... 147
6.2 ¿Cómo configuro variables de entorno para una imagen?..... 149
6.3 ¿Cómo uso Docker para iniciar una imagen guardada con una instancia de notebook?..... 149
6.4 ¿Cómo configuro un origen de Conda en un entorno de desarrollo de notebook?..... 150
6.5 ¿Cuáles son las versiones de software admitidas para una imagen personalizada?..... 152

7 Cambios de modificaciones..... 153

1 Gestión de imágenes

Descripción general

Durante el desarrollo y el tiempo de ejecución de los servicios de IA, las dependencias de entorno complejas deben depurarse para la contenedorización. En las mejores prácticas de desarrollo de IA de ModelArts las imágenes de contenedores se utilizan para proporcionar entornos de tiempo de ejecución fijos. De esta manera, las dependencias se pueden gestionar y los entornos de tiempo de ejecución se pueden cambiar fácilmente. Los recursos de contenedores proporcionados por ModelArts permiten el desarrollo rápido y eficiente de IA y la iteración de experimentos de modelos.

Las imágenes preestablecidas proporcionadas por ModelArts por defecto tienen las siguientes características:

- Listo para usar y específico para escenarios: Los entornos dependientes típicos para el desarrollo de IA están preestablecidos en estas imágenes para proporcionar configuraciones óptimas de software, SO y red. Se han probado completamente en el hardware para garantizar una compatibilidad y un rendimiento óptimos.
- Configuración personalizable: las imágenes preestablecidas se almacenan en el repositorio de SWR para que pueda personalizarlas y registrarlas como sus propias imágenes.
- Seguro y confiable: las políticas de acceso, el control de permisos de usuario, el escaneo de vulnerabilidades para software de desarrollo y el sistema operativo se configuran en función de las mejores prácticas para reforzar la seguridad para garantizar la seguridad de las imágenes.

Si tiene los requisitos especiales en el motor de aprendizaje profundo y la biblioteca de desarrollo, puede usar imágenes personalizadas de ModelArts para personalizar los motores de tiempo de ejecución.

En función de la tecnología de contenedores, puede personalizar las imágenes de contenedores y ejecutarlas en ModelArts. Las imágenes personalizadas admiten parámetros de CLI y variables de entorno en formato de texto libre, con una alta flexibilidad para una amplia gama de motores de cómputo.

Escenarios de aplicación de imágenes preestablecidas

ModelArts ofrece un grupo de imágenes preestablecidas. Puede utilizar una imagen preestablecida para crear una instancia de notebook. Después de instalar y configurar

dependencias en la instancia, cree una imagen personalizada. A continuación, puede utilizar directamente la imagen de ModelArts para los trabajos de entrenamiento sin ninguna adaptación. También puede usar imágenes preestablecidas para enviar trabajos de entrenamiento y crear aplicaciones de IA.

Recomendamos la versión de imagen preestablecida en función de sus requisitos de desarrollo y la estabilidad de la versión. Si su desarrollo se puede realizar con las versiones preestablecidas de ModelArts, por ejemplo, MindSpore 1.*X*, utilice las imágenes preestablecidas. Se han verificado completamente y tienen muchos paquetes de instalación de uso común, lo que le libera de la configuración del entorno.

Escenarios de aplicación de imágenes personalizadas

- **Usar imágenes personalizadas en las instancias de notebook**

  Si las imágenes preestablecidas de las instancias de notebook no pueden cumplir los requisitos, puede crear una imagen personalizada instalando y configurando el software y otros datos requeridos por el entorno en una imagen preestablecida. A continuación, utilice la imagen personalizada para crear nuevas instancias de notebook.

- **Usar una imagen personalizada para crear trabajos de entrenamiento**

  Si ha desarrollado un modelo o script de entrenamiento localmente pero ModelArts no admite el motor de IA, cree una imagen personalizada y súbala a SWR. A continuación, utilice esta imagen para crear un trabajo de entrenamiento sobre ModelArts y utilice los recursos proporcionados por ModelArts para entrenar modelos.

- **Usar una imagen personalizada para crear aplicaciones de IA**

  Si ha desarrollado un modelo utilizando un motor de IA que no es compatible con ModelArts, para utilizar este modelo para crear aplicaciones de IA, haga lo siguiente: Cree una imagen personalizada, importe la imagen a ModelArts y utilícela para crear aplicaciones de IA. Las aplicaciones de IA creadas de esta manera pueden gestionarse de forma centralizada y desplegarse como servicios.

Servicios de imágenes personalizadas

Cuando se utiliza una imagen personalizada, pueden estar involucrados los siguientes servicios:

- SWR

  Software Repository for Container (SWR) proporciona una gestión fácil, segura y confiable de las imágenes de contenedores a lo largo de su ciclo de vida, facilitando el despliegue de las aplicaciones en contenedores. Puede cargar, descargar y gestionar las imágenes de contenedores a través de la consola de SWR, las API de SWR o la CLI de la comunidad.

  Sus imágenes personalizadas deben subirse a SWR. Las imágenes personalizadas utilizadas por ModelArts para entrenar o crear aplicaciones de IA se obtienen de la lista de gestión de servicios de SWR.

Figura 1-1 Obtención de imágenes

- OBS

  Object Storage Service (OBS) es un servicio de almacenamiento en la nube optimizado para almacenar las cantidades masivas de datos. Proporciona capacidades de almacenamiento ilimitadas, seguras y altamente confiables con un costo relativamente bajo.

  ModelArts intercambia datos con OBS. Puede almacenar datos en OBS.

- ECS

  Un Elastic Cloud Server (ECS) es una unidad informática básica que consta de vCPUs, memoria, sistema operativo (SO) y discos de Elastic Volume Service (EVS). Después de crear un ECS, puede usarlo de manera similar a cómo usaría su PC local o servidor físico.

  Puede crear una imagen personalizada en las instalaciones o en un ECS.

NOTA

Cuando utiliza una imagen personalizada, es posible que ModelArts deba acceder a los servicios dependientes, como SWR y OBS. La imagen personalizada solo se puede utilizar una vez autorizado el acceso. Es una buena práctica utilizar una delegación para la autorización. Una vez configurada la delegación, los permisos para acceder a los servicios dependientes se delegan en ModelArts para que ModelArts pueda utilizar los servicios dependientes y realizar operaciones en los recursos en su nombre. Para obtener más información, véase **Configuración de la autorización de acceso (configuración global)**.

2 Uso de una imagen preestablecida

Imágenes preestablecidas en notebook
Imagen de base de entrenamiento
Imágenes de base de inferencia

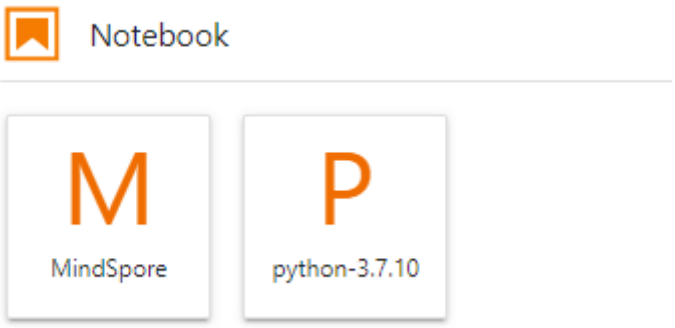
2.1 Imágenes preestablecidas en notebook

2.1.1 Imágenes de base de notebook

Imágenes preestablecidas

Las imágenes preestablecidas en el DevEnviron de ModelArts mantiene:

- Paquetes preestablecidos típicos: se incluyen motores de IA basados en Conda estándar, paquetes de software de análisis de datos como Pandas y Numpy, y software de herramientas como CUDA y CUDNN para satisfacer sus necesidades.
- Entornos de Conda preestablecidos: Se crean un entorno de Conda y Conda Python básico (excluyendo cualquier motor de IA) para cada imagen preestablecida. La siguiente figura muestra el entorno de Conda para una imagen MindSpore preestablecida.

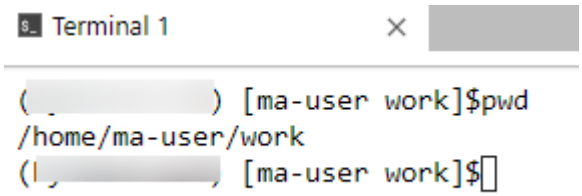


  Seleccione un entorno de Conda basado en si se utiliza MindSpore para la depuración.
- Notebook: una aplicación web que permite codificar en la GUI y combinar el código, las ecuaciones matemáticas y el contenido visualizado en un documento.

- Complementos de JupyterLab: permiten cambiar la variante y la detención de la instancia para mejorar la experiencia del usuario. Después de detener una instancia de notebook, sus CPU y memoria ya no se facturan.
- SSH remoto: le permite iniciar y depurar remotamente una instancia de notebook desde un PC local.
- Imágenes preestablecidas en ModelArts DevEnviron: Después de que estas imágenes preestablecidas admitan el desarrollo de funciones, las imágenes personalizadas creadas a partir de estas imágenes preestablecidas se pueden utilizar directamente para los trabajos de entrenamiento de ModelArts.

Una imagen preestablecida ModelArts se inicia como el usuario **ma-user**. **/home/ma-user/work** es el directorio predeterminado de trabajo de una instancia de notebook a la que se accede.

Cree una instancia y monte el almacenamiento persistente en **/home/ma-user/work**. Los datos almacenados solo en el directorio **work** se conservan después de detener y reiniciar la instancia. Cuando utilice un entorno de desarrollo, almacene los datos de persistencia en **/home/ma-user/work**.



Creación de una instancia de notebook con una imagen predefinida

Seleccione una imagen preestablecida al crear una instancia de notebook. Puede acceder a la instancia y utilizarla justo después de crearla.

1. Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación de la izquierda, seleccione **DevEnviron** > **Notebook** para acceder a la página de **Notebook** de la nueva versión.
2. Haga clic en **Create**. En la página **Create Notebook**, seleccione una imagen pública, configure otros parámetros y envíe la solicitud de creación. Para obtener más información sobre los parámetros, véase **Creación de una instancia de notebook**.
3. Después de que el estado de la instancia de notebook cambie a **Running**, acceda al notebook para usar la imagen creada.

2.1.2 Lista de imágenes base de notebook

ModelArts DevEnviron proporciona imágenes de contenedor de Docker, que se pueden ejecutar como los contenedores preestablecidos. Ciertas imágenes preestablecidas se construyen en marcos comunes de motores de IA como PyTorch, TensorFlow y MindSpore. Estas imágenes se nombran con los motores de IA. Además, muchos paquetes comunes están preestablecidos en estas imágenes, lo que lo libera de la instalación del paquete.

Tabla 2-1 Imágenes de x86 predefinidas

Motor con IA	Imagen
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
	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
Sin motor de IA (imágenes base dedicadas para la personalización de imágenes)	conda3-cuda10.2-cudnn7-ubuntu18.04
	conda3-ubuntu18.04

2.1.3 Imagen de base de notebook con PyTorch (x86)

ModelArts ofrece las siguientes imágenes de base de notebook con tecnología PyTorch (x86): **pytorch1.8-cuda10.2-cudnn7-ubuntu18.04**, **pytorch1.10-cuda10.2-cudnn7-ubuntu18.04**, and **pytorch1.4-cuda10.1-cudnn7-ubuntu18.04**.

Imagen de pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

Tabla 2-2 Descripción de pytorch1.8-cuda10.2-cudnn7-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
PyTorch 1.8	Sí (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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
				wget zip

Imagen de pytorch1.10-cuda10.2-cudnn7-ubuntu18.04

Tabla 2-3 pytorch1.10-cuda10.2-cudnn7-ubuntu18.04 description

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
PyTorch 1.10	Sí (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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
				wget zip

Imagen de pytorch1.4-cuda10.1-cudnn7-ubuntu18.04

Tabla 2-4 pytorch1.4-cuda10.1-cudnn7-ubuntu18.04 description

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
PyTorch 1.4	Sí (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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
				wget zip

2.1.4 Imagen de base de notebook con Tensorflow (x86)

ModelArts ofrece las siguientes imágenes de base para notebook con Tensorflow (x86): **tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04** y **tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04**

Image de tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

Tabla 2-5 Descripción de tensorflow2.1-cuda10.1-cudnn7-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
TensorFlow 2.1	Sí (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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
				zip

Image de tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

Tabla 2-6 Descripción de tensorflow1.13-cuda10.0-cudnn7-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
TensorFlow 1.13-gpu	Sí (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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
				zip

2.1.5 Imagen de base de notebook con MindSpore (x86)

ModelArts ofrece las siguientes imágenes de base para notebook con 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** y **mindspore1.2.0-openmpi2.1.1-ubuntu18.04**.

Imagen de mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04

Tabla 2-7 Descripción de mindspore1.7.0-cuda10.1-py3.7-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
MindSpore-gpu 1.7.0	Sí (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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
				zip

Imagen de mindspore1.7.0-py3.7-ubuntu18.04

Tabla 2-8 Descripción de mindspore1.7.0-py3.7-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
MindSpore 1.7.0	No	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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Imagen de mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

Tabla 2-9 Descripción de mindspore1.2.0-cuda10.1-cudnn7-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
MindSpore-gpu 1.2.0	Sí (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	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
				zip

Imagen de mindspore1.2.0-openmpi2.1.1-ubuntu18.04

Tabla 2-10 Descripción de mindspore1.2.0-openmpi2.1.1-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
MindSpore 1.2.0	No	swr.{region_id}.myhuaweicloud.com/atelier/mindspore_1_2_0:mindspore_1.2.0-py_3.7-ubuntu_18.04-x86_64-20220926104106-041ba2e	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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 Imagen de base de notebook con imagen dedicada personalizada (x86)

ModelArts ofrece las siguientes imágenes de base para notebook con las imágenes personalizadas (x86): **conda3-cuda10.2-cudnn7-ubuntu18.04** y **conda3-ubuntu18.04**. Estas imágenes no tienen motores de IA ni paquetes de software relacionados. El tamaño de la imagen es de solo 2 GB a 5 GB. Puede utilizar estas imágenes como imágenes base e instalar el motor y los paquetes de dependencias que desee, lo que mejora la escalabilidad. Además, estas imágenes están preestablecidas con algunas configuraciones necesarias para iniciar el entorno de desarrollo. Puede utilizar estas imágenes después de instalar los paquetes de software necesarios, sin necesidad de ninguna adaptación.

Estas imágenes son las más básicas y no tienen ningún componente instalado. Son lo suficientemente pequeños para facilitar la personalización de la imagen. Si necesita usar el SDK de OBS, use el SDK de ModelArts en su lugar para copiar archivos. Para obtener más información, véase **Transferencia de archivos**.

Imagen de conda3-cuda10.2-cudnn7-ubuntu18.04

Tabla 2-11 Descripción de conda3-cuda10.2-cudnn7-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
Ninguno	Sí (cuda 10.2)	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:cuda_10.2-ubuntu_18.04-x86_64-20221008154718-2b3e39c	Paquete de PyPI	Paquete de Ubuntu

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
			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

Imagen de conda3-ubuntu18.04

Tabla 2-12 Descripción de conda3-ubuntu18.04

Motor con IA	Si se deben usar GPU (Versión de CUDA)	URL	Dependencia	
Ninguno	No	swr.{region_id}.myhuaweicloud.com/atelier/user_defined_base:ubuntu_18.04-x86_64-20221008154718-2b3e39c For example: 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	Paquete de PyPI	Paquete de 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 Imagen de base de entrenamiento

2.2.1 Imágenes de base de entrenamiento disponibles

ModelArts proporciona las imágenes de base impulsadas por el aprendizaje profundo, como imágenes de TensorFlow, de PyTorch y de MindSpore. En estas imágenes, se ha instalado el software obligatorio para ejecutar los trabajos de entrenamiento. Si el software de las imágenes de base no puede cumplir con los requisitos de servicio, cree nuevas imágenes basadas en las imágenes de base y utilice las nuevas imágenes para crear trabajos de entrenamiento.

Imágenes de base de entrenamiento disponibles

La siguiente tabla enumera las imágenes de ModelArts de base de entrenamiento preestablecidas.

Tabla 2-13 Imágenes de base de entrenamiento de ModelArts

Motor	Versión
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

Los motores de IA admitidos varían según las regiones.

2.2.2 Entrenamiento de imagen de base (PyTorch)

Esta sección describe las imágenes de PyTorch preestablecidas.

Versión del motor: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Dirección de la imagen: 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
- Tiempo de creación de la imagen: 20220309171256 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- cuda: 10.2.89
- cudnn: 7.6.5.32
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages

- Las versiones de algunos paquetes de terceros:

```
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
...
```

- Versiones anteriores: ninguna

2.2.3 Entrenamiento de imagen de base (TensorFlow)

Esta sección describe las imágenes de TensorFlow preestablecidas.

Versión del motor: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Dirección de la imagen: 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
- Fecha y hora de creación de la imagen: 20210912152543(aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages
- Las versiones de algunos paquetes de terceros:

```
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
...
```

- Versiones anteriores: ninguna

2.2.4 Entrenamiento de imagen de base (Horovod)

Esta sección describe las imágenes de Horovod preestablecidas.

Versión del motor 1: horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Dirección de la imagen: 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
- Image creation time: 20210912152543(yyyy-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/bin/python, python 3.7.10
- Ruta de instalación del paquete de terceros: /home/ma-user/anaconda3/envs/horovod_0.20.0-tensorflow_2.1.0/lib/python3.7/site-packages
- Las versiones de algunos paquetes de terceros:

```
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
...
```

- Versiones anteriores: ninguna

Versión del motor 2: horovod_0.22.1-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Dirección de la imagen: 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
- Fecha y hora de creación de la imagen: 20210912152543 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- cuda: 11.1.1
- cudnn: 8.0.5.39
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/bin/python, python 3.7.10
- Ruta de instalación del paquete de terceros: /home/ma-user/anaconda3/envs/horovod-0.22.1-pytorch-1.8.0/lib/python3.7/site-packages
- Las versiones de algunos paquetes de terceros:

```
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
...
```

- Versiones anteriores: ninguna

2.2.5 Entrenamiento de imagen de base (MPI)

Esta sección describe las imágenes preestablecidas de mindspore_1.3.0.

Versión del motor: mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64

- Dirección de la imagen: 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
- Fecha y hora de creación de la imagen: 20211104202338(aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- cuda: 10.1.243
- cudnn: 7.6.5.32
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/bin/python, python 3.7.10
- Ruta de instalación del paquete de terceros: /home/ma-user/anaconda3/envs/MindSpore-1.3.0-gpu/lib/python3.7/site-packages
- Las versiones de algunos paquetes de terceros:

```
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
...
```

- Versiones anteriores: ninguna

2.2.6 Inicio del entrenamiento con una imagen preestablecida

2.2.6.1 PyTorch

ModelArts ofrece múltiples marcos de IA para diferentes motores. Cuando se utilizan estos motores para el entrenamiento de modelos, los comandos de arranque durante el entrenamiento deben adaptarse en consecuencia. Esta sección describe cómo realizar adaptaciones al motor de PyTorch.

Principio de inicio de PyTorch

Especificaciones y número de nodos

En este caso, se utiliza **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** como ejemplo para describir cómo asignar recursos de ModelArts para trabajos de nodo singular y distribuidos.

Para un trabajo de nodo singular (que solo se ejecuta en un nodo), ModelArts inicia un contenedor de entrenamiento que utiliza exclusivamente los recursos del nodo.

Para un trabajo distribuido (que se ejecuta en más de un nodo), hay tantos trabajadores como nodos que se seleccionan durante la creación del trabajo. A cada trabajador se le asignan los recursos de cómputo de la especificación seleccionada. Por ejemplo, hay **2** nodos de cómputo, se iniciarán dos trabajadores y cada trabajador posee los recursos de cómputo de **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB**.

Comunicación de red

- Para un trabajo de nodo singular, no se requiere la comunicación de red.
- Para un trabajo distribuido, se requieren comunicaciones de red dentro de los nodos y entre nodos.

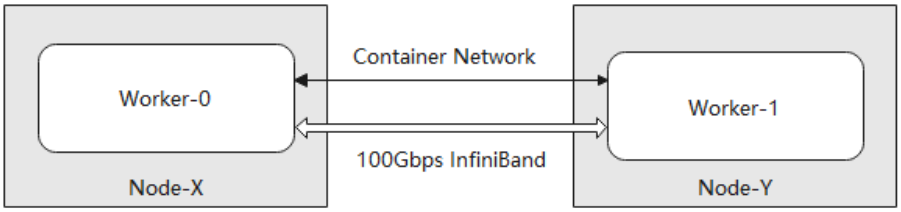
Dentro de los nodos

Se utilizan NVLink y la memoria compartida para la comunicación.

Entre los nodos

Si hay más de un nodo de cómputo, se iniciará el entrenamiento distribuido de PyTorch. La siguiente figura muestra las comunicaciones de red entre los trabajadores en el entrenamiento distribuido de PyTorch. Los trabajadores pueden comunicarse entre sí por la red de contenedores y una NIC de 100 Gbit/s de InfiniBand o de RoCE. Las NIC de RoCE se describen específicamente para ciertas especificaciones. Los contenedores pueden comunicarse con nombres de dominio de DNS, lo que es adecuado para la comunicación punto a punto a pequeña escala que requiere un rendimiento de red medio. Las NIC de InfiniBand y de RoCE son adecuadas para trabajos de entrenamiento distribuidos mediante comunicación colectiva que requieren una red de alto rendimiento.

Figura 2-1 Comunicaciones de red para el entrenamiento distribuido



Comandos de arranque

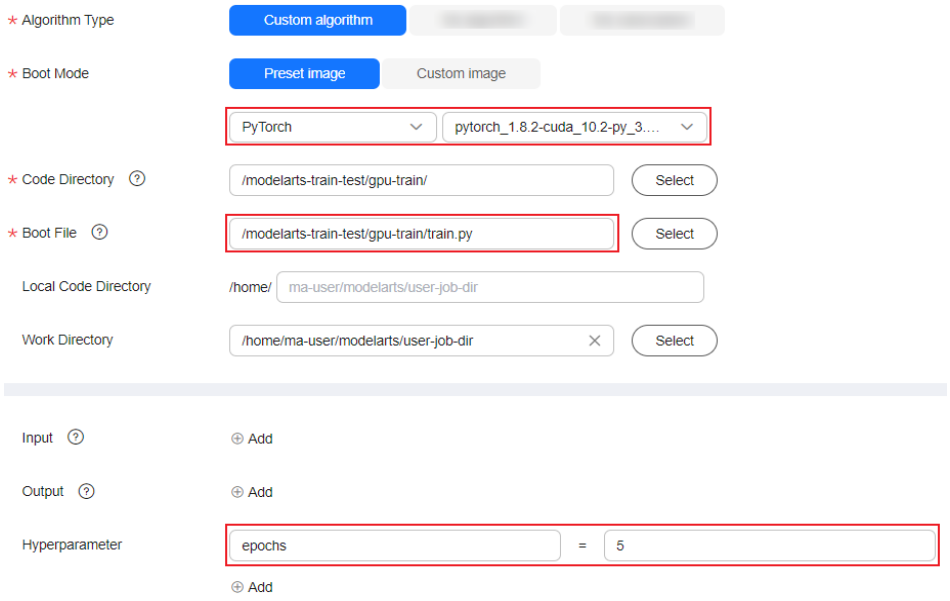
El servicio de entrenamiento utiliza el intérprete de python predeterminado en la imagen del trabajo para iniciar el script de entrenamiento. Para obtener el intérprete de python, ejecute el comando **which python**. El directorio de trabajo durante el inicio es **/home/ma-user/user-job-dir/**<*The code directory name*>, que es el directorio devuelto al ejecutar **pwd** u **os.getcwd()** en python.

- **Comando de arranque para la tarjeta única del nodo singular**

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

  - *Relative path of the startup file*: ruta del archivo de inicio relativo a **/home/ma-user/user-job-dir/***<The code directory name>*
  - *Job parameters*: Parámetros configurados para un trabajo de entrenamiento

Figura 2-2 Creación de un trabajo de entrenamiento



Configure los parámetros de acuerdo con la figura anterior. Luego, ejecute el siguiente comando en segundo plano de la consola:

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

- **Comando de arranque para múltiples tarjetas del nodo singular**

```
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*: ruta del archivo de inicio relativo a **/home/ma-user/user-job-dir/***<The code directory name>*
  - *${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}*: nombre de dominio del contenedor donde se encuentra el worker-0. Para obtener más información, véase **Variables de entorno predeterminadas**.
  - *port*: puerto de comunicación predeterminado del contenedor donde se encuentra el worker-0
  - *Job parameters*: parámetros configurados para un trabajo de entrenamiento

Figura 2-3 Creación de un trabajo de entrenamiento

Configure los parámetros de acuerdo con la figura anterior. Luego, ejecute el siguiente comando en segundo plano de la consola:

```
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 arranque para múltiples tarjetas de múltiples nodos**

  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*: ruta del archivo de inicio relativo a **/home/ma-user/user-job-dir/***<The code directory name>*

  – ${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}: nombre de dominio del contenedor donde se encuentra el worker-0. Para obtener más información, véase **Variables de entorno predeterminadas**.

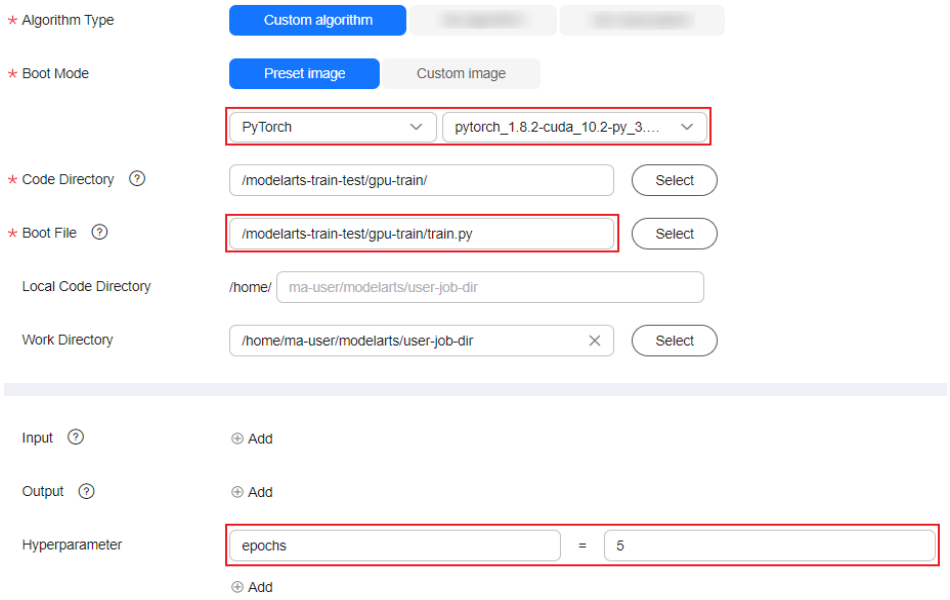
  – *port*: puerto de comunicación predeterminado del contenedor donde se encuentra el worker-0

  – *rank*: número de serie del trabajador

  – *node_num*: número de trabajadores

  – *Job parameters*: parámetros configurados para un trabajo de entrenamiento

Figura 2-4 Creación de un trabajo de entrenamiento



Configure los parámetros de acuerdo con la figura anterior. Luego, ejecute el siguiente comando en segundo plano de la consola:

```
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

ModelArts ofrece múltiples marcos de IA para diferentes motores. Cuando se utilizan estos motores para el entrenamiento de modelos, los comandos de arranque durante el entrenamiento deben adaptarse en consecuencia. Esta sección describe cómo realizar adaptaciones al motor de TensorFlow.

Principio de inicio de TensorFlow

Especificaciones y número de nodos

En este caso, se utiliza **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** como ejemplo para describir cómo asignar recursos de ModelArts para trabajos de nodo singular y distribuidos.

Para un trabajo de nodo singular (que solo se ejecuta en un nodo), ModelArts inicia un contenedor de entrenamiento que utiliza exclusivamente los recursos del nodo.

Para un trabajo distribuido (que se ejecuta en más de un nodo), ModelArts inicia un servidor de parámetros (PS) y un trabajador en el mismo nodo. El PS posee los recursos de cómputo de **CPU: 36 cores | Memory: 256 GB** y el trabajador posee **GPU: 8 xNVIDIA-V100 | CPU: 36 cores | Memory: 256 GB**.

Solo los recursos de CPU y memoria se asignan a un PS, mientras que un trabajador también puede poseer tarjetas de aceleración (excepto para las especificaciones de CPU puras). En este ejemplo, cada trabajador posee ocho tarjetas de aceleración de NVIDIA V100. Si un PS y un trabajador se inician en el mismo nodo, los recursos de disco son compartidos por ambas partes.

Comunicación de red

- Para un trabajo de nodo singular, no se requiere la comunicación de red.
- Para un trabajo distribuido, se requieren comunicaciones de red dentro de los nodos y entre nodos.

Dentro de los nodos

Un PS y un trabajador pueden comunicarse en nodos con una red de contenedor o una red host.

- Se utiliza una red de contenedor cuando se ejecuta un trabajo de entrenamiento en nodos que utilizan los recursos compartidos.
- Cuando ejecuta un trabajo de entrenamiento en nodos que utilizan un grupo dedicado, se utiliza la red host si el nodo está configurado con NIC de RoCE y la red de contenedor si el nodo está configurado con NIC de InfiniBand.

Entre los nodos

Para un trabajo distribuido, un PS y un trabajador pueden comunicarse entre nodos. ModelArts proporciona las NIC de InfiniBand y de RoCE con un ancho de banda de hasta 100 Gbit/s.

Comandos de arranque

De forma predeterminada, el servicio de entrenamiento utiliza el intérprete de Python en la imagen del trabajo para iniciar el script de entrenamiento. Para obtener el intérprete python, ejecute el comando **which python**. El directorio de trabajo durante el inicio es **/home/ma-user/user-job-dir/**<*The code directory name*>, que es el directorio devuelto al ejecutar **pwd** u **os.getcwd()** en python.

- **Comando de arranque para la tarjeta única del nodo singular**

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

  - *Relative path of the startup file*: ruta del archivo de inicio relativo a **/home/ma-user/user-job-dir/**<*The code directory name*>
  - *Job parameters*: parámetros de ejecución configurados para un trabajo de entrenamiento

Figura 2-5 Creación de un trabajo de entrenamiento

Configure los parámetros de acuerdo con la figura anterior. Luego, ejecute el siguiente comando en segundo plano de la consola:

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

- **Comando de arranque para trabajos distribuidos**

  ```
  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 serie de la tarea, por ejemplo, **0/1/2**.

  – *TF_PS_HOSTS*, matriz de direcciones de nodos PS, por ejemplo, **[***xx***-PS-0.***xx***:***TCP_PORT***,***xx***-PS-1.***xx***:***TCP_PORT***]**. El valor de **TCP_PORT** es un puerto aleatorio que va de 5,000 a 10,000.

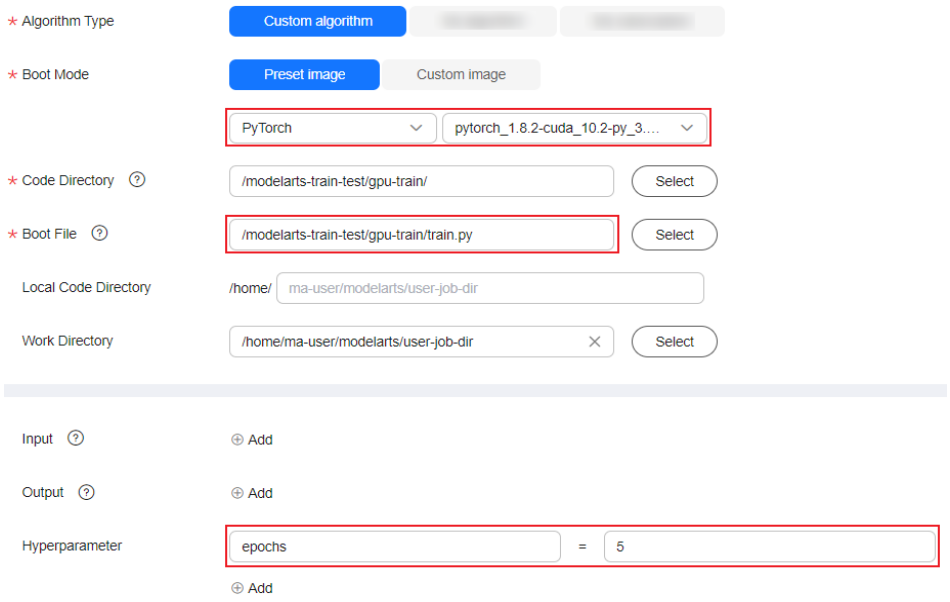
  – *TF_WORKER_HOSTS*, matriz de direcciones de nodos de trabajo, por ejemplo, **[***xx***-worker-0.***xx***:***TCP_PORT***,***xx***-worker-1.***xx***:***TCP_PORT***]**. El valor de **TCP_PORT** es un puerto aleatorio que va de 5,000 a 10,000.

  – *MA_TASK_NAME*: nombre de la tarea, que puede ser PS o trabajador.

  – *Relative path of the startup file*: ruta del archivo de inicio relativo a **/home/ma-user/user-job-dir/**<*The code directory name*>

  – *Job parameters*: parámetros de ejecución configurados para un trabajo de entrenamiento

Figura 2-6 Creación de un trabajo de entrenamiento



Configure los parámetros de acuerdo con la figura anterior. Luego, ejecute el siguiente comando en segundo plano de la consola:

```
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

ModelArts ofrece múltiples marcos de IA para diferentes motores. Cuando se utilizan estos motores para el entrenamiento de modelos, los códigos de algoritmos durante el entrenamiento deben adaptarse en consecuencia. Esta sección introduce cómo hacer adaptaciones al motor de Horovod/MPI/MindSpore-GPU.

Principio de arranque de Horovod/MPI/MindSpore-GPU

Especificaciones y número de nodos

En este caso, se utiliza **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB** como ejemplo para describir cómo asignar recursos de ModelArts para trabajos de nodo singular y distribuidos.

Para un trabajo de nodo singular (que solo se ejecuta en un nodo), ModelArts inicia un contenedor de entrenamiento que utiliza exclusivamente los recursos del nodo.

Para un trabajo distribuido (que se ejecuta en más de un nodo), hay tantos trabajadores como nodos que se seleccionan durante la creación del trabajo. A cada trabajador se le asignan los recursos de cómputo de la especificación seleccionada. Por ejemplo, si hay **2** nodos de cómputo, se iniciarán dos trabajadores y cada trabajador posee los recursos de cómputo de **GPU: 8 × NVIDIA-V100 | CPU: 72 cores | Memory: 512 GB**.

Comunicación de red

- Para un trabajo de nodo singular, no se requiere la comunicación de red.
- Para un trabajo distribuido, se requieren comunicaciones de red dentro de los nodos y entre nodos.

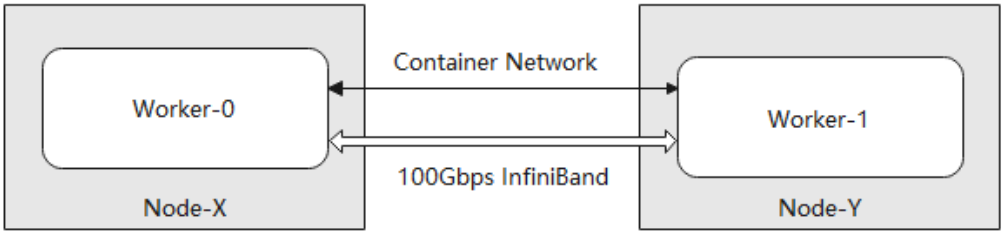
Dentro de los nodos

Se utilizan NVLink y la memoria compartida para la comunicación.

Entre los nodos

Si hay más de un nodo de cómputo, se iniciará el entrenamiento distribuido de PyTorch. La siguiente figura muestra las comunicaciones de red entre los trabajadores en el entrenamiento distribuido de PyTorch. Los trabajadores pueden comunicarse entre sí por la red de contenedores y una NIC de 100 Gbit/s de InfiniBand o de RoCE. Las NIC de RoCE se describen específicamente para ciertas especificaciones. Los contenedores pueden comunicarse con nombres de dominio de DNS, lo que es adecuado para la comunicación punto a punto a pequeña escala que requiere un rendimiento de red medio. Las NIC de InfiniBand y de RoCE son adecuadas para trabajos de entrenamiento distribuidos mediante comunicación colectiva que requieren una red de alto rendimiento.

Figura 2-7 Comunicaciones de red para el entrenamiento distribuido



Comandos de arranque

De forma predeterminada, el servicio de entrenamiento utiliza el intérprete de Python en la imagen del trabajo para iniciar el script de entrenamiento. Para obtener el intérprete python, ejecute el comando **which python**. El directorio de trabajo durante el inicio es **/home/ma-user/user-job-dir/**<*The code directory name*>, que es el directorio devuelto al ejecutar **pwd** u **os.getcwd()** en python.

Comandos de arranque

```
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 procesos iniciados por **mpirun**. El valor predeterminado es el número de GPU multiplicado por el número de nodos. No modifique este valor.
- *OPENMPI_HOST_FILE_PATH*: valor de **hostfile**. No modifique este valor.
- *SSHD_PORT*: Puerto para el inicio de sesión de SSH. No modifique este valor.
- *TUNE_ENV_FILE*: variables de entorno del worker-0. Transmita las siguientes variables de entorno a otros nodos de trabajo del trabajo de entrenamiento actual.
  - **env** con el prefijo **MA_**
  - **env** con el prefijo **SHARED_**

  - **env** con el prefijo **S3_**
  - **env** de **PATH**
  - **env** con el prefijo **VC_WORKER_**
  - **env** con el prefijo **SCC**
  - **env** con el prefijo **CRED**

    ```
    env|grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED'|grep -v '=
    $'> ${TUNE_ENV_FILE}
    ```

- *OPENMPI_BIND_ARGS*: anclar el proceso con el comando **mpirun cpu**. La configuración predeterminada es la siguiente:

  ```
  OPENMPI_BIND_ARGS="-bind-to none -map-by slot"
  ```

- *OPENMPI_X_ARGS*: **-x** los parámetros del comando **mpirun**. La configuración predeterminada es la siguiente:

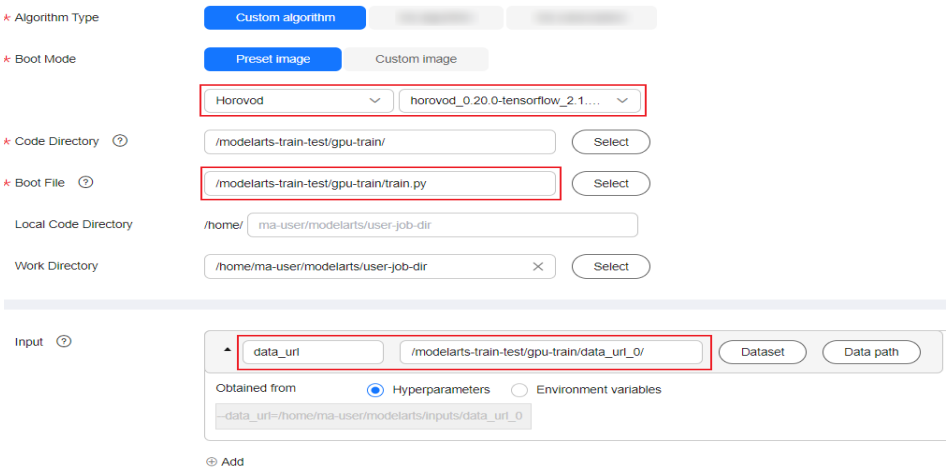
  ```
  OPENMPI_X_ARGS="-x LD_LIBRARY_PATH -x HOROVOD_MPI_THREADS_DISABLE=1 -x
  NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=ib0,bond0,eth0 -x
  NCCL_SOCKET_FAMILY=AF_INET -x NCCL_IB_DISABLE=0"
  ```

- *OPENMPI_X_ARGS*: **-mca** los parámetros del comando **mpirun**. La configuración predeterminada es la siguiente:

  ```
  OPENMPI_MCA_ARGS="-mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn
  true"
  ```

- *OPENMPI_EXTRA_ARGS*: parámetros pasados a **mpirun**. El valor predeterminado es vacío.
- *Relative path of the startup file*: ruta del archivo de inicio relativo a **/home/ma-user/user-job-dir/**<*The code directory name*>
- *Job parameters*: parámetros de ejecución configurados para un trabajo de entrenamiento

Figura 2-8 Creación de un trabajo de entrenamiento



Configure los parámetros de acuerdo con la figura anterior. Luego, ejecute el siguiente comando en segundo plano de la consola:

```
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

Si utiliza un motor de Horovod, MPI o MindSpore-GPU para el entrenamiento de modelos, los comandos de arranque para trabajos de nodo único y trabajos distribuidos son los mismos.

2.3 Imágenes de base de inferencia

2.3.1 Imágenes de base de inferencia disponibles

La inferencia de ModelArts proporciona una serie de imágenes base. Puede crear las imágenes personalizadas basadas en estas imágenes base para desplegar los servicios de inferencia.

x86 (CPU/GPU)

Tabla 2-14 TensorFlow

Versión del motor de IA	Entorno de tiempo de ejecución	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

Tabla 2-15 PyTorch

Versión del motor de IA	Entorno de tiempo de ejecución	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

Versión del motor de IA	Entorno de tiempo de ejecución	URI
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

Tabla 2-16 MindSpore

Versión del motor de IA	Entorno de tiempo de ejecución	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
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 Imágenes de base de inferencia con TensorFlow (CPU/GPU)

ModelArts ofrece las siguientes imágenes de base de inferencia con TensorFlow (CPU/GPU):

- **Versión del motor 1: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **Versión del motor 2: tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64**
- **Versión del motor 3: tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64**

Versión del motor 1: tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Ruta de la imagen: 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
- Fecha y hora de creación de la imagen: 20220713110657 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/TensorFlow-2.1/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/TensorFlow-2.1/lib/python3.7/site-packages

- Ciertos paquetes de instalación de 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
...
```

- Ciertos paquetes de instalación de 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
...
```

Versión del motor 2: tensorflow_1.15.5-cuda_11.4-py_3.8-ubuntu_20.04-x86_64

- Ruta de la imagen: 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
- Fecha y hora de creación de la imagen: 20220524162601 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 20.04.4 LTS
- CUDA: 11.4.3
- cuDNN: 8.2.4.15
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/TensorFlow-1.15.5/bin/python, python 3.8.13
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/TensorFlow-1.15.5/lib/python3.8/site-packages
- Ciertos paquetes de instalación de 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
  ...
  ```

- Ciertos paquetes de instalación de 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
...
```

Versión del motor 3: tensorflow_2.6.0-cuda_11.2-py_3.7-ubuntu_18.04-x86_64

- Ruta de la imagen: 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
- Fecha y hora de creación de la imagen: 20220524162601 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- CUDA: 11.2.0

- cuDNN: 8.1.1.33
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/TensorFlow-2.6.0/lib/python3.7/site-packages
- Ciertos paquetes de instalación de 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
...
```

- Ciertos paquetes de instalación de 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 Imágenes base de inferencia con PyTorch (CPU/GPU)

ModelArts ofrece las siguientes imágenes de base de inferencia con PyTorch (CPU/GPU):

- **Versión del motor 1: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64**
- **Versión del motor 2: pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

Versión del motor 1: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64

- Ruta de la imagen: 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
- Fecha y hora de creación de la imagen: 20220713110657 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- CUDA: 10.2.89
- cuDNN: 7.6.5.32
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages
- Ciertos paquetes de instalación de 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
...
```

- Ciertos paquetes de instalación de 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
...
```

Versión del motor 2: pytorch_1.8.2-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- Ruta de la imagen: 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
- Fecha y hora de creación de la imagen: 20220524162601 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/PyTorch-1.8.2/lib/python3.7/site-packages
- Ciertos paquetes de instalación de 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
  ...
  ```

- Ciertos paquetes de instalación de 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 Imágenes base de inferencia con MindSpore (CPU/GPU)

ModelArts ofrece las siguientes imágenes de base de inferencia con MindSpore (CPU/GPU):

- **Versión del motor 1: mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64**
- **Versión del motor 2: mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64**
- **Versión del motor 3: mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64**

Versión del motor 1: mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64

- Ruta de la imagen: swr.{region}.myhuaweicloud.com/atelier/mindspore_1_7_0:mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64-20220702120711-8590b76
- Fecha y hora de creación de la imagen: 20220702120711 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Ciertos paquetes de instalación de 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
...
```

- Ciertos paquetes de instalación de 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
...
```

Versión del motor 2: mindspore_1.7.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64

- Ruta de la imagen: 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
- Fecha y hora de creación de la imagen: 20220702120711 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- CUDA: 10.1.243
- cuDNN: 7.6.5.32
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages

- Ciertos paquetes de instalación de 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
...
```

- Ciertos paquetes de instalación de 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
...
```

Versión del motor 3: mindspore_1.7.0-cuda_11.1-py_3.7-ubuntu_18.04-x86_64

- Ruta de la imagen: 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
- Fecha y hora de creación de la imagen: 20220702120711 (aaaa-mm-dd-hh-mm-ss)
- Versión del sistema de la imagen: Ubuntu 18.04.4 LTS
- CUDA: 11.1.1
- cuDNN: 8.0.5.39
- Ruta y versión del intérprete de Python: /home/ma-user/anaconda3/envs/MindSpore/bin/python, python 3.7.10
- Ruta de instalación de paquetes de terceros: /home/ma-user/anaconda3/envs/MindSpore/lib/python3.7/site-packages
- Ciertos paquetes de instalación de 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
...
```

- Ciertos paquetes de instalación de 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 imágenes personalizadas en instancias de notebook

Registro de una imagen en ModelArts

Creación de una imagen personalizada

Guardar una instancia de Notebook como una imagen personalizada

Creación y uso de una imagen personalizada en notebook

Creación de una imagen personalizada en un ECS y su uso en notebook

3.1 Registro de una imagen en ModelArts

Después de crear una imagen personalizada, regístrela en la página de **Image Management** de ModelArts antes de usarla en notebook.

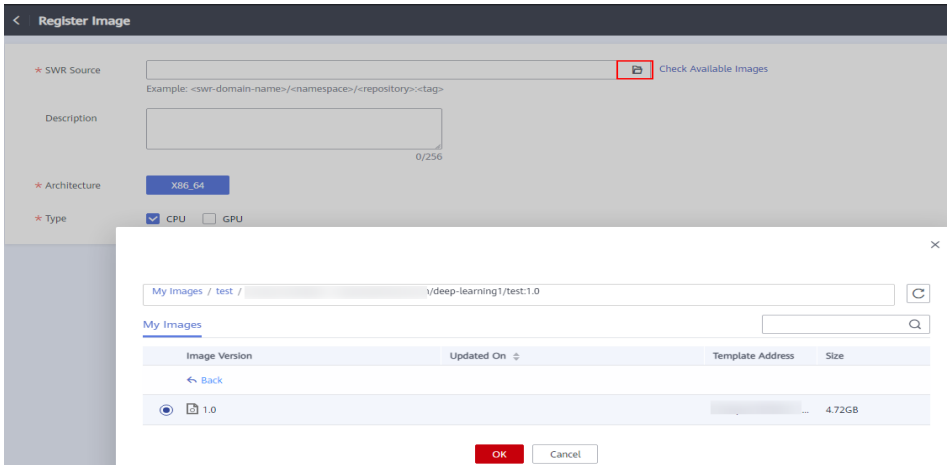
NOTA

Solo los subusuarios (usuarios de IAM) de la cuenta pueden registrar y utilizar las imágenes de SWR si el tipo de imagen es **Private**.

Otros usuarios pueden registrar y utilizar las imágenes de SWR solo cuando el tipo de imagen es **Public**.

1. Inicie sesión en la consola de gestión de ModelArts y seleccione **Image Management**. Luego, haga clic en **Register**.
2. Configure los parámetros y haga clic en **Register**.
   - **SWR Source**: seleccione una imagen generada como origen de la imagen. Puede copiar la dirección de SWR completa o hacer clic en  para seleccionar la imagen de destino para el registro.

Figura 3-1 Selección de un origen de imagen



– **Architecture** y **Type**: configúrelos en función del marco real de la imagen personalizada.

3. Vea la imagen registrada en la página **Image Management**.

Figura 3-2 Lista de imágenes

Creación de una instancia de notebook

Haga clic en el nombre de la imagen. En la página de detalles de la imagen que aparece, haga clic en **Create Notebook**. Se muestra la página para crear una instancia de notebook con esta imagen.

Figura 3-3 Página de detalles de imagen

Sincronización de una imagen

Una vez rectificada la falla de la imagen, vaya a la página de detalles de la imagen. Haga clic en **Sync** en la columna **Operation** para actualizar el estado de la imagen.

3.2 Creación de una imagen personalizada

Puede crear una imagen personalizada de cualquiera de las siguientes maneras:

- Método 1: Utilice una imagen preestablecida de instancias de notebook para crear una instancia de entorno de desarrollo. Luego, instale y configure las dependencias en el entorno. Después de la configuración, utilice la función de guardado de imágenes proporcionada por el entorno de desarrollo para guardar la instancia en ejecución como

una imagen de contenedor personalizada. Para más detalles, véase **Guardar una instancia de Notebook como una imagen personalizada**.

- Método 2: Utilice imágenes de base de ModelArts y plantillas de creación de imágenes para escribir un Dockerfile y crear su propia imagen en una instancia de notebook. A continuación, registre la imagen para crear un nuevo entorno de desarrollo basado en sus necesidades. Para más detalles, véase **Creación y uso de una imagen personalizada en notebook**.
- Método 3: Uilice las imágenes de base de ModelArts o las imágenes de terceros para escribir un Dockerfile en un ECS y reconstruirlas Esto le permite personalizar las imágenes de Docker que cumplan con **los requisitos de ModelArts** y enviar las imágenes a SWR. Para más detalles, véase **Creación de una imagen personalizada en un ECS y su uso en notebook**.

3.3 Guardar una instancia de Notebook como una imagen personalizada

3.3.1 Guardar una imagen de entorno de notebook

Para guardar una imagen de entorno de notebook, haga lo siguiente: Cree una instancia de notebook con una imagen preestablecida, instale el software personalizado y las dependencias en la imagen base y guarde la instancia en ejecución como una imagen de contenedor.

En la imagen guardada, se conservan las dependencias instaladas. Los datos almacenados en **home/ma-user/work** para el almacenamiento permanente no se almacenarán. Cuando se utiliza VS Code para el desarrollo remoto, se conservan los complementos instalados en el servidor.

NOTA

Las imágenes almacenadas en una instancia de notebook no pueden superar los 35 GB y no puede haber más de 125 capas de imágenes. De lo contrario, la imagen no se puede guardar.

Si se informa del error "El tamaño del contenedor (xx) es mayor que el umbral (25G)" cuando se guarda una imagen, trate el error según **¿Qué hago si aparece el error "El tamaño del contenedor (xG) es mayor que el umbral (25G)" al guardar una imagen?**.

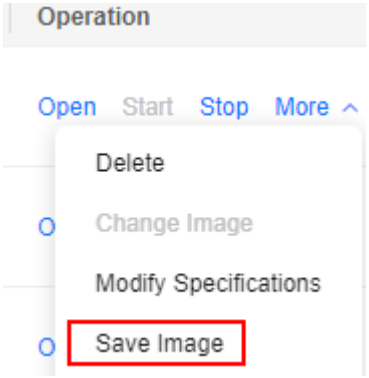
Requisitos previos

La instancia de notebook está en estado **Running**.

Guardar una imagen

1. Inicie sesión en la consola de gestión de ModelArts y elija **DevEnviron** > **Notebook** en el panel de navegación de la izquierda para cambiar a notebook de la nueva versión.
2. En la lista de instancias de notebook, seleccione la instancia de notebook de destino y elija **Save Image** en la lista desplegable **More** de la columna **Operation**. Aparecerá el cuadro de diálogo **Save Image**.

Figura 3-4 Guardar imagen



3. En el cuadro de diálogo **Save Image**, configure los parámetros. Haga clic en **OK** para guardar la imagen.

   Elija una organización de la lista desplegable **Organization**. Si no hay ninguna organización disponible, haga clic en **Create** a la derecha para crear una.

   Los usuarios de una organización pueden compartir todas las imágenes de la organización.

4. La imagen se guardará como una instantánea y tardará unos 5 minutos. Durante este período de tiempo, no realice ninguna operación en la instancia.

   Figura 3-5 Guardar como instantánea

   AVISO

   El tiempo necesario para guardar una imagen como instantánea se contará en la duración de ejecución de la instancia. Si la duración de ejecución de la instancia vence antes de guardar la instantánea, se producirá un error al guardar la imagen.

5. Después de guardar la imagen, el estado de la instancia cambia a **Running**. Vea la imagen en la página **Image Management**.
6. Haga clic en el nombre de la imagen para ver sus detalles.

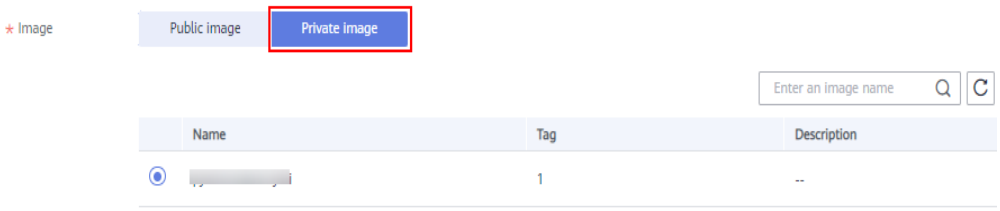
3.3.2 Uso de una imagen personalizada para crear una instancia de notebook

Las imágenes guardadas de una instancia de notebook se pueden ver en la página **Image Management**. Puede utilizar estas imágenes para crear nuevas instancias de notebook, que heredan las configuraciones de software de las instancias de notebook originales.

Puede utilizar cualquiera de los siguientes métodos:

Método 1: En la página **Create Notebook**, haga clic en **Private Image** y seleccione la imagen guardada.

Figura 3-6 Selección de una imagen personalizada para crear una instancia de notebook



Método 2: En la página **Image Management**, haga clic en la imagen de destino para acceder a su página de detalles. A continuación, haga clic en **Create Notebook**.

3.4 Creación y uso de una imagen personalizada en notebook

3.4.1 Escenarios de aplicación y proceso

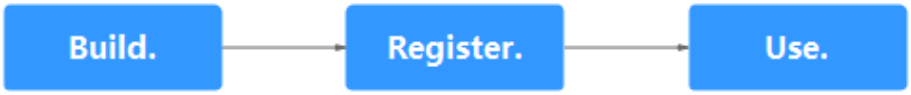
Si las imágenes preestablecidas no cumplen con los requisitos de servicio, puede crear imágenes de contenedor basadas en las imágenes preestablecidas para el desarrollo y el entrenamiento.

En general, necesitará reconstruir el entorno de desarrollo de ModelArts, por ejemplo, instalando, actualizando o desinstalando algunos paquetes. Sin embargo, el permiso de root es necesario cuando se instalan o actualizan determinados paquetes. La instancia de notebook en ejecución no tiene el permiso de root. Como resultado, debe instalar el software que requiere el permiso de root en la instancia de notebook, que actualmente no está disponible en el entorno de desarrollo preestablecido.

Necesita escribir un Dockerfile basado en una imagen pública preestablecida para personalizar su imagen. Luego, depure la imagen para que ModelArts pueda usarla. Por último, registre la imagen con ModelArts para que pueda utilizarse para crear entornos de desarrollo que cumplan con sus requisitos de servicio.

Este ejemplo muestra cómo usar comandos **ma-cli** en la CLI de ModelArts para crear y registrar una imagen personalizada para el desarrollo de IA con una imagen de base de PyTorch. Para obtener más información, véase **Comando de creación de imágenes ma-cli**. La siguiente figura muestra todo el proceso.

Figura 3-7 Creación de una imagen



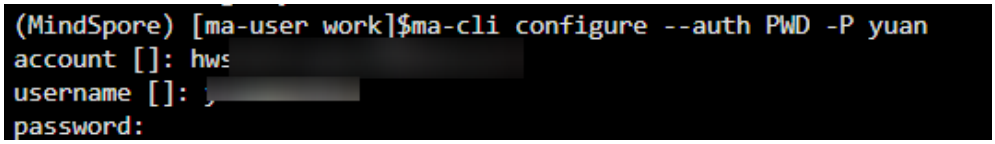
3.4.2 Paso 1 Crear una imagen personalizada

Esta sección muestra cómo crear una imagen cargando una plantilla de creación de imagen y escribiendo un Dockerfile. Asegúrese de haber creado el entorno de desarrollo y de haber

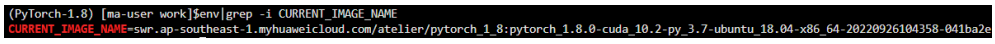
abierto una terminal en la página **Notebook**. Para obtener más información sobre Dockerfiles, véase la **Referencia de Dockerfile**.

Paso 1 Configure la información de autenticación, especifique un perfil e ingrese la información de la cuenta según se solicite. Para obtener más información sobre la autenticación, véase **Autenticación de ma-cli**.

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



Paso 2 Ejecute **env|grep -i CURRENT_IMAGE_NAME** para consultar la imagen utilizada por la instancia actual.



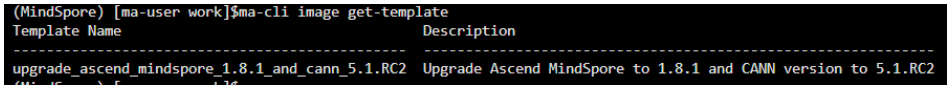
Paso 3 Cree una imagen.

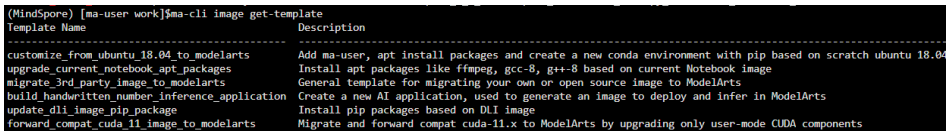
1. Obtenga la dirección de SWR de la imagen base.

   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. Cargar una plantilla de creación de imágenes.

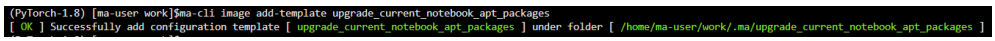
   Ejecute el comando **ma-cli image get-template** para consultar la plantilla de imagen.





   Ejecute el comando **ma-cli image add-template** para cargar la plantilla de imagen en la carpeta especificada. La ruta predeterminada es donde se encuentra el comando actual. Por ejemplo, cargue la plantilla de creación de imágenes **upgrade_current_notebook_apt_packages**.

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



3. Modifique un Dockerfile.

   El Dockerfile en este ejemplo se modifica en función de la imagen de base de PyTorch pytorch1.8-cuda10.2-cudnn7-ubuntu18.04, se carga la plantilla de imagen upgrade_current_notebook_apt_packages y se actualizan GCC y G++.

   Después de cargar la plantilla de imagen, se carga automáticamente el Dockerfile en **.ma/upgrade_current_notebook_apt_packages**. El contenido es el siguiente y puede modificarlo según sus necesidades.

```
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 \
```



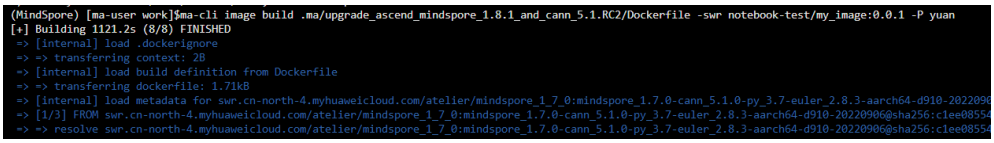
```
    HTTPS_PROXY=http://proxy.modelarts.com:80 \
    https_proxy=http://proxy.modelarts.com:80

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. Construya una imagen.

   Ejecute el comando **ma-cli image build** para crear una imagen con el Dockerfile. Para obtener más información, véase **Creación de una imagen en el notebook de ModelArts**.

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

   El Dockerfile se almacena en **.ma/upgrade_current_notebook_apt_package/Dockerfile** y la nueva imagen se almacena en **notebook-test/my_image:0.0.1** en SWR. **XXX** indica el perfil especificado para la autenticación.

----Fin

3.4.3 Paso 2 Registrar una nueva imagen

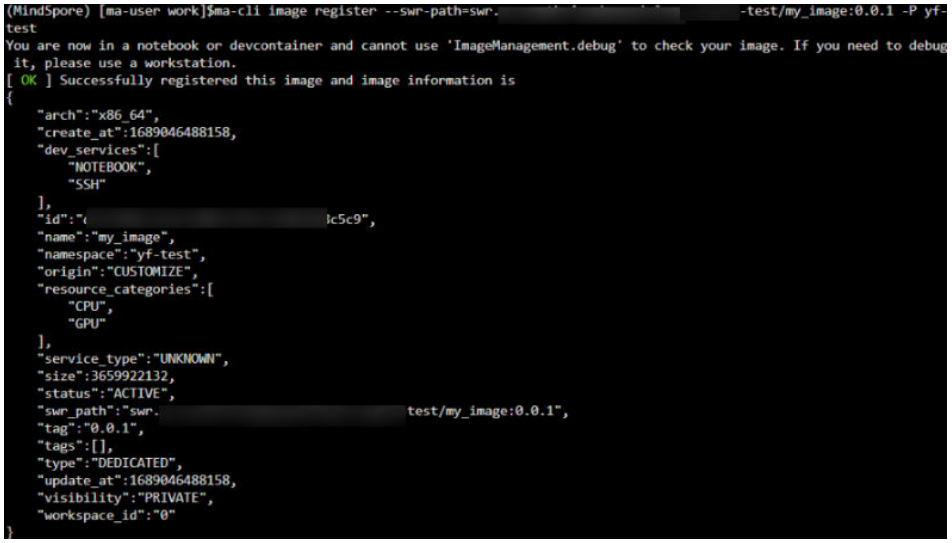
Después de depurar una imagen, regístrela en la gestión de imágenes de ModelArts para que la imagen se pueda usar en ModelArts.

Para registrar la imagen con ModelArts, utilice uno de los métodos siguientes:

- **Method 1**: Ejecute el comando **ma-cli image register** para registrar una imagen. Luego, se devuelve la información de la imagen registrada, que incluye el ID y el nombre de la imagen, como se muestra en la siguiente figura. Para obtener más información, véase **Registro de imágenes de SWR con la gestión de imágenes de 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 Imagen registrada



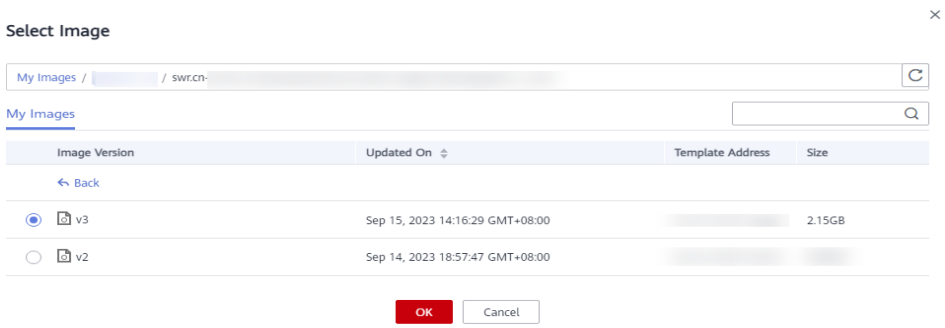
- **Método 2**: Registre la imagen en la consola de gestión de ModelArts.

  Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación de la izquierda, seleccione **Image Management**. Se muestra la página **Image Management**.

  Haga clic en **Register**. Pegue la dirección de SWR completa o haga clic en para seleccionar una imagen privada de SWR para registrarla, como se muestra en **Figura 3-9**.

  Seleccione la arquitectura y el tipo según los requisitos del sitio. La arquitectura y el tipo deben ser iguales a los de la fuente de la imagen.

Figura 3-9 Selección de una imagen



3.4.4 Paso 3 Usar una nueva imagen para crear un entorno de desarrollo

Procedimiento

Después de registrar una imagen, está disponible para la creación del entorno de desarrollo. Puede iniciar sesión en la consola de gestión de ModelArts, elegir **DevEnviron** > **Notebook** y seleccionar la imagen durante la creación.

3.5 Creación de una imagen personalizada en un ECS y su uso en notebook

3.5.1 Escenarios de aplicación y proceso

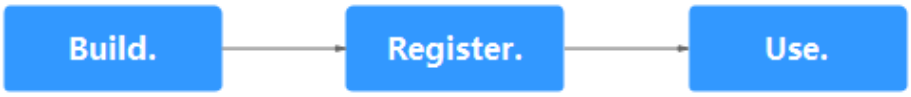
En general, necesitará reconstruir el entorno de desarrollo de ModelArts, por ejemplo, instalando, actualizando o desinstalando algunos paquetes. Sin embargo, el permiso de root es necesario cuando se instalan o actualizan determinados paquetes. La instancia de notebook en ejecución no tiene el permiso de root. Como resultado, debe instalar el software que requiere el permiso de root en la instancia de notebook, que actualmente no está disponible en el entorno de desarrollo preestablecido.

Puede escribir un Dockerfile basado en una imagen de base preestablecida o en una imagen de terceros para personalizar su imagen. A continuación, puede registrar la imagen para crear un nuevo entorno de desarrollo basado en sus necesidades.

Esta sección describe cómo instalar PyTorch 1.8, FFmpeg 3 y GCC 8 en una imagen de Ubuntu para crear un nuevo entorno de desarrollo de IA.

La siguiente figura muestra todo el proceso.

Figura 3-10 Creación y depuración de una imagen



3.5.2 Paso 1 Preparar un servidor de Docker y configurar un entorno

Prepare un servidor con Docker habilitado. Si no hay un servidor de este tipo disponible, cree un ECS, compre una EIP e instale el software requerido en él. La creación, depuración y registro de imágenes posteriores se realizan en este servidor.

ModelArts proporciona scripts de Ubuntu para que instale Docker más fácilmente.

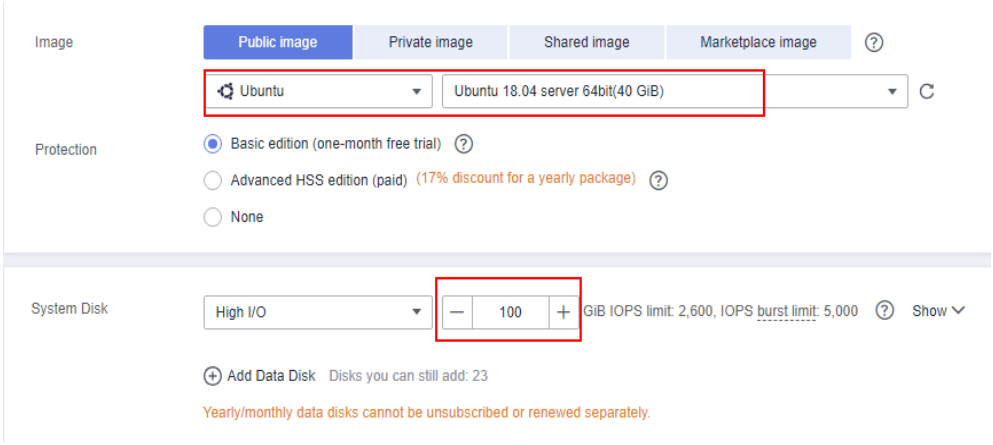
NOTA

Las operaciones en el servidor de Linux local son las mismas que las del ECS. Para más detalles, véase este caso.

Creación de un ECS

- Inicie sesión en la consola de ECS y haga clic en **Buy ECS**. Seleccione una imagen pública (se recomienda una imagen de Ubuntu 18.04) y configure el disco del sistema en 100 GiB. Para obtener más detalles, véase **Compra e inicio de sesión en un ECS de Linux**.

Figura 3-11 Selección de una imagen y un disco



- Compre una EIP y vincúlela al ECS. Para obtener más detalles, véase **Configuración de red**.

Configuración de un ECS

1. Ejecute el siguiente comando en el ECS de Docker para descargar el script de instalación:

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

> **NOTA**
>
> Solo se admiten los scripts de Ubuntu.

2. Ejecute el siguiente comando en el ECS de Docker para configurar el entorno:

```
bash install_on_ubuntu1804.sh
```

Figura 3-12 Configurado


```
source /etc/profile
```

El script de instalación se ejecuta en:

a. Instale Docker.

b. Si el ECS de Docker se ejecuta en GPU, instale nvidia-docker2 para montar las GPU en el contenedor de Docker.

3.5.3 Paso 2 Crear una imagen personalizada

Esta sección describe cómo editar un Dockerfile, usarlo para crear una imagen y usar la imagen creada para crear una instancia de notebook. Para obtener detalles sobre cómo editar un Dockerfile, véase la **Referencia de Dockerfile**.

Requisitos previos

Ha preparado un servidor de Docker según **Paso 1 Preparar un servidor de Docker y configurar un entorno**.

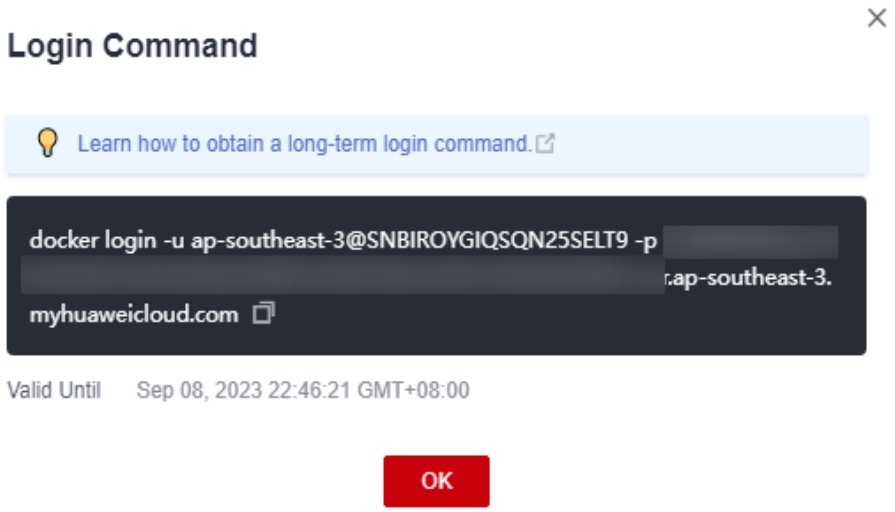
Consulta de imágenes base (omita este paso para imágenes de terceros)

Para obtener más información sobre las imágenes de base de ModelArts, véase la **Lista de imágenes de base de notebook**. Verifique la URL de la imagen en la sección correspondiente según el tipo de motor de la imagen preestablecida.

Creación de una imagen

1. Acceda a SWR.
   a. Inicie sesión en la consola de SWR.
   b. En el panel de navegación de la izquierda, elija **Dashboard** y haga clic en **Generate Login Command** en el extremo superior derecho. En la página mostrada, copie el comando de inicio de sesión.

   Figura 3-13 Obtención del comando de inicio de sesión

   NOTA

   - El período de validez del comando de inicio de sesión generado es de 24 horas. Para obtener un comando de inicio de sesión válido a largo plazo, vea **Obtención de un comando de inicio de sesión con validez a largo plazo**. Después de obtener un comando de inicio de sesión válido a largo plazo, los comandos de inicio de sesión temporales seguirán siendo válidos mientras estén en sus períodos de validez.
   - El nombre de dominio al final del comando de inicio de sesión es la dirección del repositorio de imágenes. Registre la dirección para uso posterior.

   c. Ejecute el comando de inicio de sesión en la máquina donde está instalado el motor de contenedor.

   El mensaje "Login Succeeded" aparecerá en pantalla cuando el inicio de sesión sea exitoso.

2. Extraiga una imagen base o una imagen de terceros. A continuación se utiliza una imagen de terceros como ejemplo.
   ```
   docker pull swr.ap-southeast-1.myhuaweicloud.com/notebook-xxx/ubuntu:18.04
   #Your organization name and image
   ```

3. Compile un Dockerfile.

Ejecute el comando **vim** para crear un Dockerfile. Si se utiliza una imagen base de ModelArts, véase **Dockerfile en una imagen base de ModelArts** para obtener detalles sobre el Dockerfile.

Si se utiliza una imagen de terceros, agregue el usuario **ma-user** cuyo **UID** es **1000** y el grupo de usuarios **ma-group** cuyo **GID** es **100**. Para más detalles, véase **Dockerfile en una imagen base que no es ModelArts**.

En este caso, se instalarán PyTorch 1.8, FFmpeg 3 y GCC 8 en una imagen de Ubuntu para crear una imagen de IA.

4. Construya una imagen.

   Ejecute el comando **docker build** para crear una nueva imagen a partir del Dockerfile. La descripción de los parámetros del comando es la siguiente:

   - **-t** especifica la nueva ruta de acceso de la imagen, incluida la información de la región, el nombre de la organización, el nombre de la imagen y la versión. Configure este parámetro según el escenario de la vida real. Utilice una dirección de SWR completa para la depuración y el registro.
   - **-f** especifica el nombre de Dockerfile. Configure este parámetro según el escenario de la vida real.
   - **.** al final especifica que el contexto es el directorio actual. Configure este parámetro según el escenario de la vida real.

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

   Figura 3-14 Imagen creada

Dockerfile en una imagen base de ModelArts

Ejecute el comando **vim** para crear un Dockerfile. Si ModelArts proporciona la imagen base, el contenido del Dockerfile es el siguiente:

```
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 en una imagen base que no es ModelArts

Si se utiliza una imagen de terceros, agregue el usuario **ma-user** cuyo **UID** es **1000** y el grupo de usuarios **ma-group** cuyo **GID** es **100** al Dockerfile. Si el UID 1000 o el GID 100 de la imagen base ha sido utilizado por otro usuario o grupo de usuarios, elimine el usuario o grupo de usuarios. **El usuario y el grupo de usuarios se han agregado al Dockerfile en este caso. Se pueden usar directamente.**

> **NOTA**
>
> Solo necesita establecer el usuario **ma-user** cuyo **UID** es **1000** y el grupo de usuarios **ma-group** cuyo **GID** es **100** y conceda la lectura, escritura, y ejecute permisos en el directorio de destino para el usuario **ma-user**.

Ejecute el comando **vim** para crear un Dockerfile y agregar una imagen de terceros (no ModelArts) como imagen base, por ejemplo, ubuntu 18.04. El contenido del Dockerfile es el siguiente:

```
# 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 Paso 3 Registrar una nueva imagen

Después de depurar una imagen, regístrela en la gestión de imágenes de ModelArts para que la imagen se pueda usar en ModelArts.

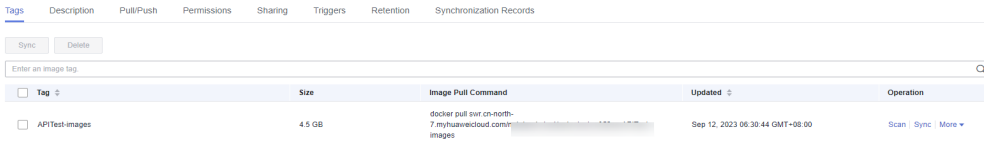
1. Empuje la imagen a SWR.

   Primero inicie sesión en SWR. Para obtener más detalles, véase **Inicie sesión en SWR**. Ejecute el siguiente comando para insertar la imagen:

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

   La imagen está entonces disponible en SWR.

Figura 3-15 Empujar la imagen a SWR



2. Registre una imagen.

   Registro de una imagen en la consola de ModelArts

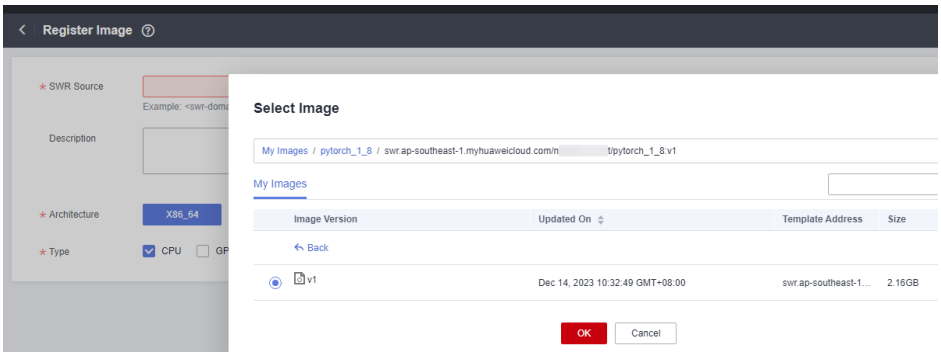
   Inicie sesión en la consola de ModelArts. En el panel de navegación de la izquierda, seleccione **Image Management** para acceder a la página de gestión de imágenes. Haga clic en **Register**. Configure **SWR Source** en la imagen insertada en SWR en el **Paso 1**.

   Haga clic en  para seleccionar una imagen existente que desea registrar, como se muestra en **Figura 3-16**.

   NOTA

   Cuando registre una nueva imagen, asegúrese de que la arquitectura y el tipo sean los mismos que los del origen de la imagen. De lo contrario, la creación falla.

Figura 3-16 Registro de una imagen

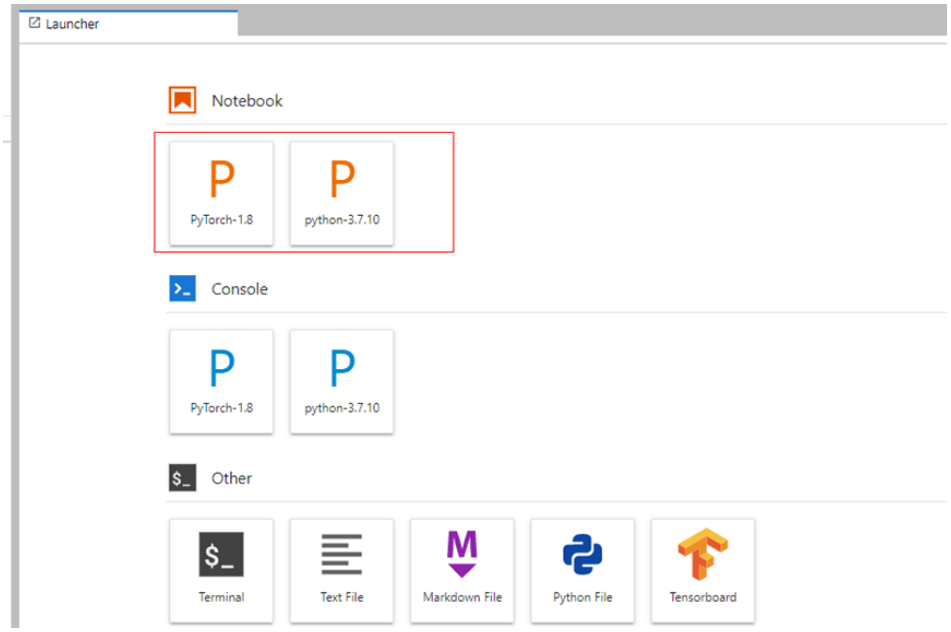


3.5.5 Paso 5 Crear e iniciar un entorno de desarrollo

Procedimiento

1. Una vez creada la imagen, inicie sesión en la consola de ModelArts, vaya a la pestaña de notebook y elija la imagen registrada en **Paso 3 Registrar una nueva imagen** para crear un entorno de desarrollo.
2. Vaya a la lista de notebook, haga clic en **Open** para iniciar el entorno de desarrollo creado.

Figura 3-17 Apertura de un entorno de desarrollo



3. Abra un terminal para verificar el entorno de conda. Para obtener más información sobre conda, véase el **sitio web oficial**.

   Cada núcleo del entorno de desarrollo es esencialmente un entorno conda instalado en **/home/ma-user/anaconda3/**. Ejecute el comando **/home/ma-user/anaconda3/bin/conda env list** para verificar el entorno de la conda.

Figura 3-18 Comprobación del entorno de conda

```
(PyTorch-1.8) [ma-user work]$/home/ma-user/anaconda3/bin/conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8           *  /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10            /home/ma-user/anaconda3/envs/python-3.7.10
```


4 Uso de una imagen personalizada para entrenar modelos (entrenamiento de modelos)

Descripción general

Ejemplo: creación de una imagen personalizada para entrenamiento

Preparación de una imagen de entrenamiento

Creación de un algoritmo mediante una imagen personalizada

Uso de una imagen personalizada para crear un trabajo de entrenamiento basado en CPU o GPU

Proceso de solución de problemas

4.1 Descripción general

Los algoritmos suscritos y las imágenes preestablecidas se pueden utilizar en la mayoría de los escenarios de entrenamiento. En ciertos escenarios de ModelArts le permite crear las imágenes personalizadas para entrenar modelos.

La personalización de una imagen requiere una comprensión profunda de los contenedores. Utilice este método solo si los algoritmos suscritos y las imágenes preestablecidas no pueden cumplir con sus requisitos. Las imágenes personalizadas se pueden usar para entrenar modelos de ModelArts solo después de que se suban al Software Repository for Container (SWR).

Puede utilizar imágenes personalizadas para el entrenamiento de ModelArts de cualquiera de las siguientes maneras:

- Uso de una imagen preestablecida con personalización

  Si utiliza una imagen preestablecida para crear un trabajo de entrenamiento y necesita modificar o agregar algunas dependencias de software basadas en la imagen preestablecida, puede personalizar la imagen preestablecida. En este caso, seleccione una imagen preestablecida y elija **Customize** en el cuadro de lista desplegable de versiones del marco.
- Uso de una imagen personalizada

Puede crear una imagen basada en las especificaciones de imagen ModelArts, seleccionar su propia imagen y configurar el directorio de código (opcional) y el comando boot para crear un trabajo de entrenamiento.

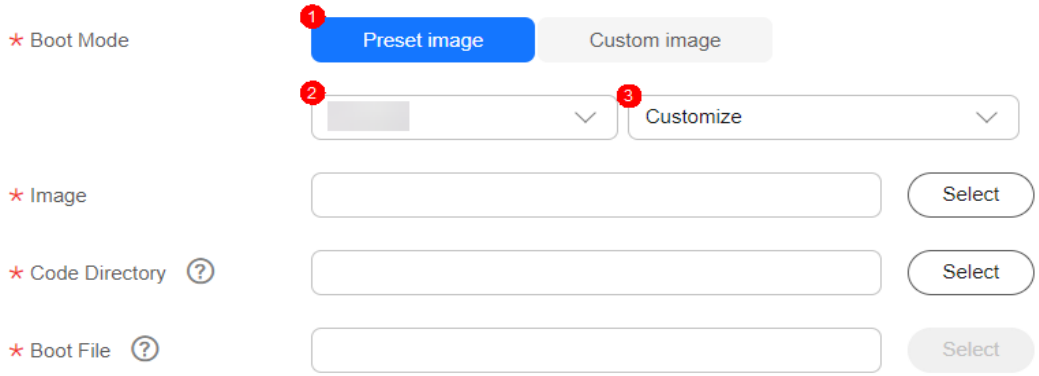
NOTA

Cuando se utiliza una imagen personalizada para crear un trabajo de entrenamiento, el comando de arranque se debe ejecutar en el directorio **/home/ma-user**. De lo contrario, el trabajo de entrenamiento puede ejecutarse de manera anormal.

Uso de una imagen preestablecida con personalización

Entre este método y crear un trabajo de entrenamiento totalmente basado en una imagen preestablecida, la única diferencia es que debe seleccionar una imagen. Puede crear una imagen personalizada basada en una imagen preestablecida.

Figura 4-1 Crear un algoritmo usando una imagen preestablecida con personalización

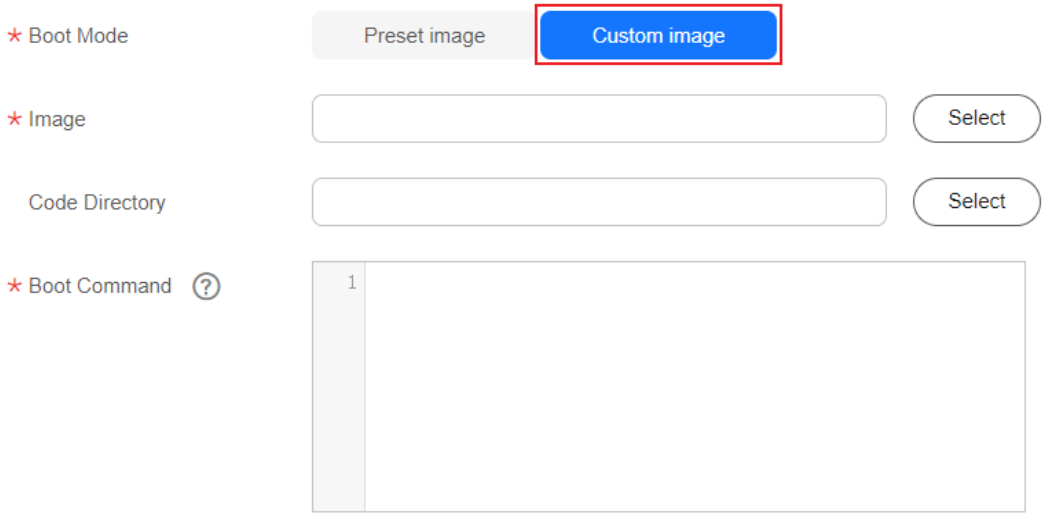


El proceso de este método es el mismo que el de crear un trabajo de entrenamiento basado en una imagen preestablecida. Por ejemplo:

- El sistema inyecta automáticamente variables de entorno.
  - PATH=${MA_HOME}/anaconda/bin:${PATH}
  - LD_LIBRARY_PATH=${MA_HOME}/anaconda/lib:${LD_LIBRARY_PATH}
  - PYTHONPATH=${MA_JOB_DIR}:${PYTHONPATH}
- El archivo de arranque seleccionado se iniciará automáticamente con los comandos de Python. Asegúrese de que el entorno de Python sea correcto. La variable de entorno de PATH se inyecta automáticamente. Ejecute los siguientes comandos para comprobar la versión de Python para el trabajo de entrenamiento:
  - export MA_HOME=/home/ma-user; docker run --rm {image} ${MA_HOME}/anaconda/bin/python -V
  - docker run --rm {image} $(which python) -V
- El sistema agrega automáticamente hiperparámetros asociados con la imagen preestablecida.

Uso de una imagen personalizada

Figura 4-2 Creación de un algoritmo mediante una imagen personalizada



Para obtener más información sobre cómo utilizar imágenes personalizadas compatibles con el entrenamiento, véase **Uso de una imagen personalizada para crear un trabajo de entrenamiento basado en CPU o GPU**.

Si todas las imágenes usadas están personalizadas, haga lo siguiente para usar un entorno de Conda especificado para comenzar el entrenamiento:

Los trabajos de entrenamiento no se ejecutan en un shell. Por lo tanto, no se le permite ejecutar el comando **conda activate** para activar un entorno de Conda especificado. En este caso, utilice otros métodos para iniciar el entrenamiento.

Por ejemplo, Conda en la imagen personalizada está instalado en el directorio **/home/ma-user/anaconda3**, el entorno de Conda es **python-3.7.10** y el script de entrenamiento está almacenado en **/home/ma-user/modelarts/user-job-dir/code/train.py**. Utilice un entorno de Conda especificado para iniciar el entrenamiento de una de las siguientes maneras:

- Método 1: Configure las variables de entorno **DEFAULT_CONDA_ENV_NAME** y **ANACONDA_DIR** correctas para la imagen.

  Ejecute el comando **python** para iniciar el script de entrenamiento. A continuación se muestra un ejemplo:

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

- Método 2: Utilice la ruta absoluta del entorno de Conda de Python.

  Ejecute el comando **/home/ma-user/anaconda3/envs/python-3.7.10/bin/python** para iniciar el script de entrenamiento. A continuación se muestra un ejemplo:

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

- Método 3: Configurar la variable de entorno de ruta.

  Configure el directorio bin del entorno de Conda especificado en la variable de entorno de ruta. Ejecute el comando **python** para iniciar el script de entrenamiento. A continuación se muestra un ejemplo:

  ```
  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: Ejecute el comando **conda run -n**.

  Ejecute el comando **/home/ma-user/anaconda3/bin/conda run -n python-3.7.10** para ejecutar el entrenamiento. A continuación se muestra un ejemplo:

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

  NOTA

  Si hay un error que indica que el archivo .so no está disponible en el directorio **$ANACONDA_DIR/envs/$DEFAULT_CONDA_ENV_NAME/lib**, agregue el directorio a **LD_LIBRARY_PATH** y coloque el siguiente comando antes del comando de arranque anterior:

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

  Por ejemplo, el comando de arranque de ejemplo utilizado en el método 1 es el siguiente:

  ```
  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 Ejemplo: creación de una imagen personalizada para entrenamiento

4.2.1 Ejemplo: creación de una imagen personalizada para el entrenamiento (PyTorch + CPU/GPU)

En esta sección se describe cómo crear una imagen y utilizarla para entrenamiento en la plataforma de ModelArts. El motor de IA utilizado para el entrenamiento es PyTorch y los recursos son CPU o GPU.

NOTA

Esta sección solo se aplica a los trabajos de entrenamiento de la nueva versión.

Escenarios

En este ejemplo, cree una imagen personalizada escribiendo un Dockerfile en un host de Linux x86_64 que ejecute el sistema operativo Ubuntu 18.04.

Objetivo: crear e instalar imágenes de contenedor del siguiente software y utilizar las imágenes y las CPU/GPU para entrenamiento en ModelArts.

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

Procedimiento

Antes de utilizar una imagen personalizada para crear un trabajo de entrenamiento, debe estar familiarizado con Docker y tener experiencia en desarrollo. El procedimiento detallado es el siguiente:

1. **Requisitos previos**

2. **Paso 1 Crear un bucket de OBS y una carpeta**
3. **Paso 2 Preparar el script de entrenamiento y cargarlo en OBS**
4. **Paso 3 Preparar un host**
5. **Paso 4 Crear una imagen personalizada**
6. **Paso 5 Cargar una imagen en SWR**
7. **Paso 6 Crear un trabajo de entrenamiento en ModelArts**

Requisitos previos

Ha creado un ID de Huawei y ha habilitado los servicios en Huawei Cloud. Además, la cuenta no está en mora ni congelada.

Paso 1 Crear un bucket de OBS y una carpeta

Cree un bucket y unas carpetas en OBS para almacenar la muestra de conjunto de datos y el código de entrenamiento. **Tabla 4-1** enumera las carpetas que se van a crear. En el ejemplo, el nombre del bucket y los nombres de las carpetas junto con los nombres reales.

Para obtener más información sobre cómo crear un bucket de OBS y una carpeta, véase **Creación de un bucket** y **Creación de una carpeta**.

Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región.

Tabla 4-1 Carpeta para crear

Nombre	Descripción
obs://test-modelarts/pytorch/demo-code/	Almacena el script de entrenamiento.
obs://test-modelarts/pytorch/log/	Almacena los archivos de log de entrenamiento.

Paso 2 Preparar el script de entrenamiento y cargarlo en OBS

Prepare el script de entrenamiento **pytorch-verification.py** y cárguelo en la carpeta **obs://test-modelarts/pytorch/demo-code/** del bucket del OBS.

El archivo **pytorch-verification.py** contiene la siguiente información:

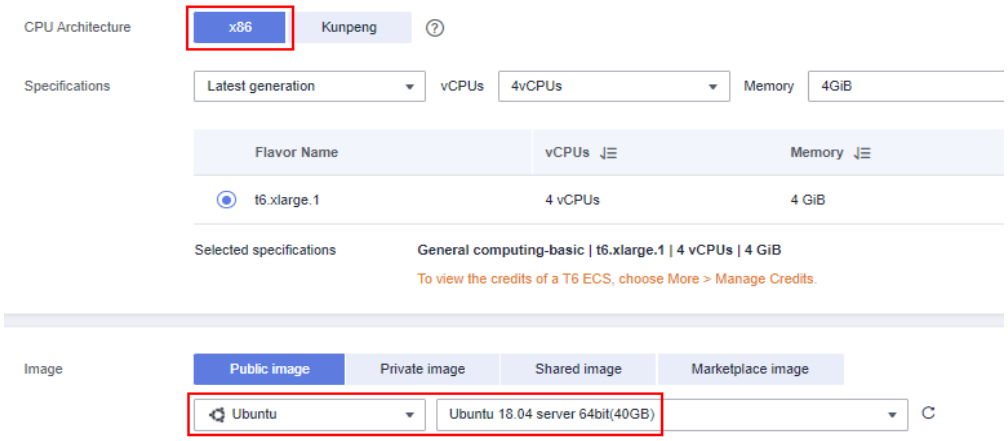
```
import torch
import torch.nn as nn

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)
```


Paso 3 Preparar un host

Obtener un servidor de Linux x86_64 que ejecute Ubuntu 18.04. Un ECS o su PC local servirán.

Para obtener más información sobre cómo comprar un ECS, véase **Compra e inicio de sesión en un ECS de Linux**. Seleccione una imagen pública. Se recomienda una imagen de Ubuntu 18.04.

Figura 4-3 Creación de un ECS con una imagen pública (x86)



Paso 4 Crear una imagen personalizada

Cree una imagen de contenedor con las siguientes configuraciones y utilice la imagen para crear un trabajo de entrenamiento en ModelArts:

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

En esta sección se describe cómo escribir un Dockerfile para crear una imagen personalizada.

1. Instale Docker.

   A continuación se utiliza Linux x86_64 OS como ejemplo para describir cómo obtener el paquete de instalación de Docker. Para obtener más detalles sobre cómo instalar Docker, consulte los **documentos oficiales de Docker**.

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

   Si se ejecuta el comando **docker images**, se ha instalado Docker. Si es así, omita este paso.

2. Ejecute el siguiente comando para verificar la versión del Docker Engine:

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

   Se muestra la siguiente información:

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


> **NOTA**
>
> Utilice el motor de Docker de la versión anterior o posterior para crear una imagen personalizada.

3. Cree una carpeta denominada **context**.

```
mkdir -p context
```

4. Obtenga el archivo **pip.conf**. En este ejemplo, se utiliza el origen pip proporcionado por Huawei Mirrors, que es el siguiente:

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

> **NOTA**
>
> En Huawei Mirrors **https://mirrors.huaweicloud.com/home**, busque **pypi** para obtener el archivo pip.conf.

5. Descargue los siguientes archivos .whl desde 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**
>
> El código de URL del símbolo + es %2B. Cuando busque un archivo en el sitio web anterior, reemplace el símbolo + en el nombre del archivo por %2B.
>
> Por ejemplo, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

6. Descargue el archivo de instalación Miniconda3-py37_4.12.0-Linux-x86_64.sh (Python 3.7.13) desde **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.
7. Almacene el archivo de origen de pip, el archivo torch*.whl y el archivo de instalación de Miniconda3 en la carpeta **context**, que es la siguiente:

```
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. Escriba la imagen del contenedor Dockerfile.

   Cree un archivo vacío denominado **Dockerfile** en la carpeta **context** y copie el siguiente contenido en el archivo:

```
# 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 obtener detalles sobre cómo escribir un Dockerfile, consulte los **documentos oficiales de Docker**.

9. Verifique que se haya creado el Dockerfile. A continuación se muestra la carpeta **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. Cree la imagen de contenedor. Ejecute el siguiente comando en el directorio donde se almacena el Dockerfile para crear la imagen de contenedor **pytorch:1.8.1-cuda11.1**:

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

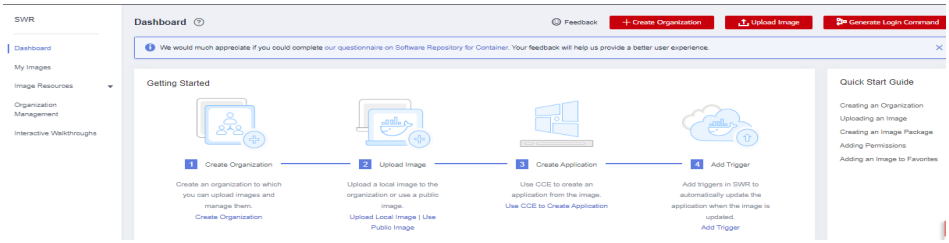
La siguiente información de log mostrada durante la creación de la imagen indica que la imagen se ha creado.

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

Paso 5 Cargar una imagen en SWR

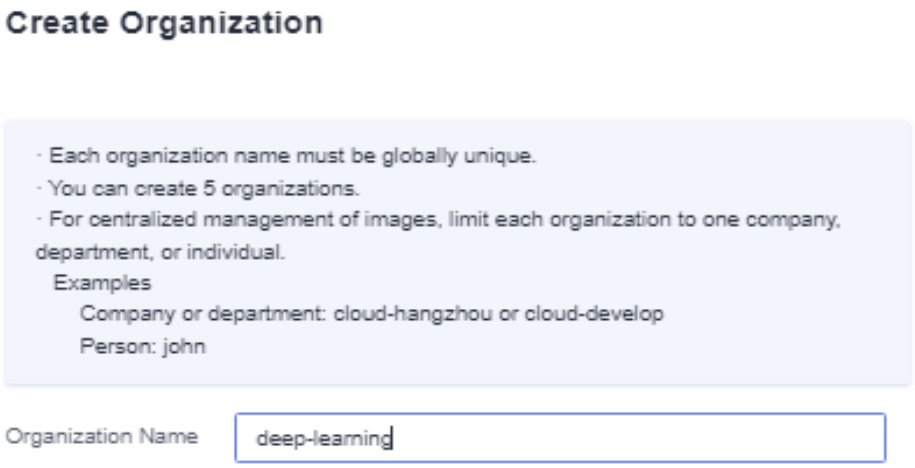
1. Inicie sesión en la consola de SWR y seleccione la región de destino.

Figura 4-4 Consola de SWR



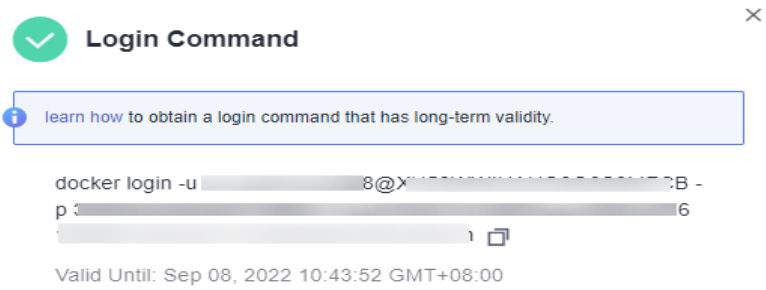
2. Haga clic en **Create Organization** en la esquina superior derecha e introduzca un nombre de organización para crear una organización. Personalice el nombre de la organización. Sustituya el nombre de la organización **deep-learning** en comandos posteriores con el nombre real de la organización.

Figura 4-5 Creación de una organización



3. Haga clic en **Generate Login Command** en la esquina superior derecha para obtener un comando de acceso.

Figura 4-6 Comando de acceso



4. Inicie sesión en el entorno local como usuario **root** e ingrese el comando de inicio de sesión.
5. Cargue la imagen en SWR.
   a. Ejecute el siguiente comando para etiquetar la imagen cargada:
   ```
   #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. Ejecute el siguiente comando para subir la imagen:
   ```
   #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. Después de cargar la imagen, elija **My Images** en el panel de navegación a la izquierda de la consola de SWR para ver las imágenes personalizadas cargadas.

Paso 6 Crear un trabajo de entrenamiento en ModelArts

1. Inicie sesión en la consola de gestión de ModelArts y compruebe si se ha configurado la autorización de acceso para su cuenta. Para obtener más información, véase **Configuración de la autorización de la delegación**. Si ha sido autorizado mediante claves de acceso, borre la autorización y configure la autorización de la delegación.
2. En el panel de navegación, seleccione **Training Management** > **Training Jobs**. La lista de trabajos de entrenamiento se muestra de forma predeterminada.
3. En la página **Create Training Job**, configure los parámetros necesarios y haga clic en **Submit**.
   - **Created By**: **Algoritmos personalizados**
   - **Boot Mode**: **Imágenes personalizadas**
   - Image path: imagen creada en **Paso 5 Cargar una imagen en SWR**.
   - **Code Directory**: directorio donde se almacena el archivo de script de arranque en OBS. Por ejemplo, **obs://test-modelarts/pytorch/demo-code/**. El código de entrenamiento se descarga automáticamente en el directorio **${MA_JOB_DIR}/demo-code** del contenedor de entrenamiento. **demo-code** (personalizable) es el directorio de último nivel de la ruta del OBS.
   - **Boot Command**: **/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py**. **demo-code** (personalizable) es el directorio de último nivel de la ruta del OBS.
   - **Recurso grupo**: **Grupos de recursos públicos**
   - **Resource Type**: seleccione **CPU** o **GPU**.

- **Persistent Log Saving**: habilitado
- **Job Log Path**: Establezca este parámetro en la ruta de OBS para almacenar logs de entrenamiento, por ejemplo, **obs://test-modelarts/pytorch/log/**.

4. Verifique los parámetros del trabajo de entrenamiento y haga clic en **Submit**.
5. Espere hasta que finalice el trabajo de entrenamiento.

   Después de crear un trabajo de entrenamiento, las operaciones como la descarga de imágenes de contenedor, la descarga de directorios de código y la ejecución de comandos de arranque se realizan automáticamente en el backend. Por lo general, la duración del entrenamiento oscila entre decenas de minutos y varias horas, dependiendo del procedimiento de entrenamiento y de los recursos seleccionados. Una vez ejecutado el trabajo de entrenamiento, se muestra un log similar al siguiente.

Figura 4-7 Ejecutar logs de trabajos de entrenamiento con especificaciones 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 Run logs of training jobs with CPU specifications

```
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 Ejemplo: creación de una imagen personalizada para entrenamiento (MPI + CPU/GPU)

En esta sección se describe cómo crear una imagen y utilizarla para entrenamiento en la plataforma de ModelArts. El motor de IA utilizado para el entrenamiento es MPI y los recursos son CPU o GPU.

NOTA

Esta sección solo se aplica a los trabajos de entrenamiento de la nueva versión.

Escenarios

En este ejemplo, cree una imagen personalizada escribiendo un Dockerfile en un host de Linux x86_64 que ejecute el sistema operativo Ubuntu 18.04.

Objetivo: crear e instalar imágenes de contenedor del siguiente software y utilizar las imágenes y las CPU/GPU para entrenamiento en ModelArts.

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

Procedimiento

Antes de usar una imagen personalizada para crear un trabajo de entrenamiento, familiarícese con Docker y tenga experiencia en desarrollo. El procedimiento detallado es el siguiente:

1. **Requisitos previos**
2. **Paso 1 Crear un bucket de OBS y una carpeta**
3. **Paso 2 Preparar los archivos y cargarlos en OBS**
4. **Paso 3 Preparar un servidor de imágenes**
5. **Paso 4 Crear una imagen personalizada**
6. **Paso 5 Cargar una imagen en SWR**
7. **Paso 6 Crear un trabajo de entrenamiento en ModelArts**

Requisitos previos

Ha creado un ID de Huawei y ha habilitado los servicios en Huawei Cloud. Además, la cuenta no está en mora ni congelada.

Paso 1 Crear un bucket de OBS y una carpeta

Cree un bucket y unas carpetas en OBS para almacenar la muestra de conjunto de datos y el código de entrenamiento. **Tabla 4-2** enumera las carpetas que se van a crear. En el ejemplo, el nombre del bucket y los nombres de las carpetas junto con los nombres reales.

Para obtener más información sobre cómo crear un bucket de OBS y una carpeta, véase **Creación de un bucket** y **Creación de una carpeta**.

Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región.

Tabla 4-2 Carpeta para crear

Nombre	Descripción
obs://test-modelarts/mpi/demo-code/	Almacena el script de arranque de MPI y el archivo de script de entrenamiento.
obs://test-modelarts/mpi/log/	Almacena los archivos de log de entrenamiento.

Paso 2 Preparar los archivos y cargarlos en OBS

Prepare el script de inicio de MPI **run_mpi.sh** y el script de entrenamiento **mpi-verification.py** y cárguelos en la carpeta **obs://test-modelarts/mpi/demo-code/** del bucket de OBS.

- El contenido del script de inicio de MPI **run_mpi.sh** es el siguiente:

```
#!/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

El script **run_mpi.sh** utiliza finales de línea LF. Si se utilizan finales de línea CRLF, la ejecución del trabajo de entrenamiento fallará y se mostrará el error "$'\r': command not found" en los logs.

- El contenido del script de entrenamiento **mpi-verification.py** es el siguiente:

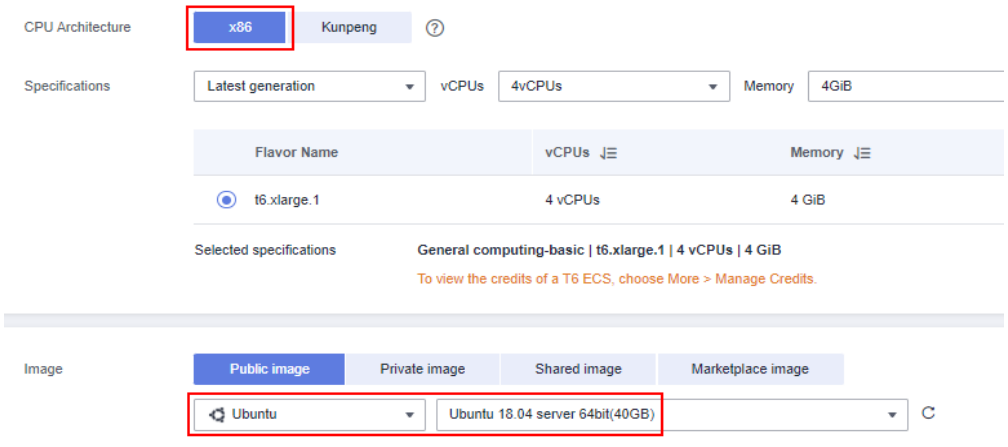
```
import os
import socket

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'])
```

Paso 3 Preparar un servidor de imágenes

Obtener un servidor de Linux x86_64 que ejecute Ubuntu 18.04. Un ECS o su PC local servirán.

Para obtener más información sobre cómo comprar un ECS, véase **Compra e inicio de sesión en un ECS de Linux**. Seleccione una imagen pública. Se recomienda una imagen de Ubuntu 18.04.

Figura 4-9 Creación de un ECS con una imagen pública (x86)



Paso 4 Crear una imagen personalizada

Objetivo: crear e instalar imágenes de contenedor del siguiente software y utilizar el servicio de entrenamiento de ModelArts para ejecutarlas.

- ubuntu-18.04
- cuda-11.1

- python-3.7.13
- openmpi-3.0.0

A continuación se describe cómo crear una imagen personalizada escribiendo un Dockerfile.

1. Instale Docker.

   A continuación se utiliza Linux x86_64 OS como ejemplo para describir cómo obtener un paquete de instalación de Docker. Para obtener más detalles, consulte los **documentos oficiales de Docker**. Ejecute los siguientes comandos para instalar Docker:

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

   Si se ejecuta el comando **docker images**, se ha instalado Docker. Si es así, omita este paso.

2. Verifique la versión del motor de Docker. Ejecute el siguiente comando:

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

   Se muestra la siguiente información:

   ```
   Engine:
    Version:          18.09.0
   ```

   NOTA

   Se recomienda utilizar el Docker Engine de esta versión o posterior para crear una imagen personalizada.

3. Cree una carpeta denominada **context**.

   ```
   mkdir -p context
   ```

4. Descargue el archivo de instalación de Miniconda3.

   Descargue el archivo de instalación de Miniconda3 py37 4.12.0 (Python 3.7.13) desde **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

5. Descargue el archivo de instalación de openmpi 3.0.0.

   Descargue el archivo de openmpi 3.0.0 editado con Horovod v0.22.1 desde **https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz**.

6. Guarde los archivos de Miniconda3 y de openmpi 3.0.0 en la carpeta **context**. A continuación se muestra la carpeta **context**:

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

7. Escriba el Dockerfile de la imagen de contenedor.

   Cree un archivo vacío denominado **Dockerfile** en la carpeta **context** y escriba el siguiente contenido en el archivo:

   ```
   # 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 obtener detalles sobre cómo escribir un Dockerfile, consulte los **Documentos oficiales de Docker**.

8. Verifique que se haya creado el Dockerfile. A continuación se muestra la carpeta **context**:

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

9. Cree la imagen de contenedor. Ejecute el siguiente comando en el directorio donde se almacena Dockerfile para crear la imagen de contenedor **mpi:3.0.0-cuda11.1**:

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

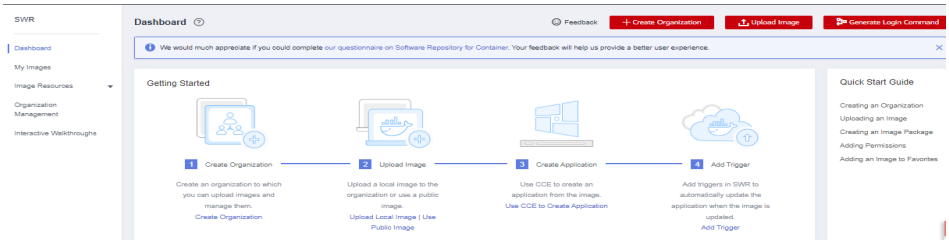
La siguiente información de log mostrada durante la creación de la imagen indica que la imagen se ha creado.

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

Paso 5 Cargar una imagen en SWR

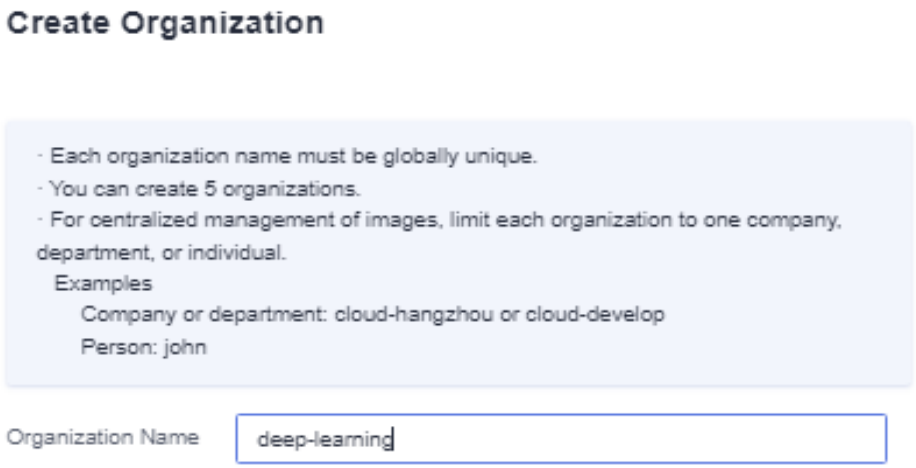
1. Inicie sesión en la consola de SWR y seleccione la región de destino.

Figura 4-10 Consola de SWR



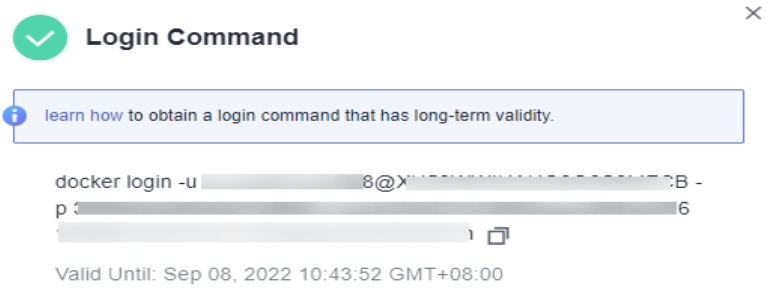
2. Haga clic en **Create Organization** en la esquina superior derecha e introduzca un nombre de organización para crear una organización. Personalice el nombre de la organización. Sustituya el nombre de la organización **deep-learning** en comandos posteriores con el nombre real de la organización.

Figura 4-11 Creación de una organización



3. Haga clic en **Generate Login Command** en la esquina superior derecha para obtener un comando de acceso.

   Figura 4-12 Comando de acceso

4. Inicie sesión en el entorno local como usuario **root** e ingrese el comando de inicio de sesión.
5. Cargue la imagen en SWR.
   a. Ejecute el siguiente comando para etiquetar la imagen cargada:
   ```
   #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. Ejecute el siguiente comando para subir la imagen:
   ```
   #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. Después de cargar la imagen, seleccione **My Images** en el panel de navegación izquierdo de la consola de SWR para ver las imágenes personalizadas cargadas.

   swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1 es la dirección URL de SWR de la imagen personalizada.

Paso 6 Crear un trabajo de entrenamiento en ModelArts

1. Inicie sesión en la consola de gestión de ModelArts y compruebe si se ha configurado la autorización de acceso para su cuenta. Para obtener más información, véase **Configuración de la autorización de la delegación**. Si ha sido autorizado mediante claves de acceso, borre la autorización y configure la autorización de la delegación.
2. Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación izquierdo, seleccione **Training Management** > **Training Jobs (New)**.
3. En la página **Create Training Job**, configure los parámetros y haga clic en **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Ruta de acceso a la imagen: **swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1**
   - **Code Directory**: ruta de acceso de OBS al script de inicio, por ejemplo, **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**: Agregue **MY_SSHD_PORT = 38888**.

- **Resource Pool**: **Public resource pools**
- **Resource Type**: seleccione **GPU**.
- **Compute Nodes**: escriba **1** o **2**.
- **Persistent Log Saving**: habilitado
- **Job Log Path**: Establezca este parámetro en la ruta de OBS para almacenar logs de entrenamiento, por ejemplo, **obs://test-modelarts/mpi/log/**.

4. Verifique los parámetros del trabajo de entrenamiento y haga clic en **Submit**.
5. Espere hasta que finalice el trabajo de entrenamiento.

   Después de crear un trabajo de entrenamiento, las operaciones como la descarga de imágenes de contenedor, la descarga de directorios de código y la ejecución de comandos de arranque se realizan automáticamente en el backend. Por lo general, la duración del entrenamiento oscila entre decenas de minutos y varias horas, dependiendo del procedimiento de entrenamiento y de los recursos seleccionados. Una vez ejecutado el trabajo de entrenamiento, se muestra un log similar al siguiente.

   Figura 4-13 Ejecutar logs de worker-0 con un nodo de cómputo y especificaciones 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
```

   Configure **Compute Nodes** en **2** y ejecute el trabajo de entrenamiento. **Figura 4-14** y **Figura 4-15** muestran la información del log.

   Figura 4-14 Ejecutar logs de worker-0 con dos nodos de cómputo y especificaciones 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 Ejecutar logs de worker-1 con dos nodos de cómputo y especificaciones 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 Ejemplo: creación de una imagen personalizada para entrenamiento (Horovod-PyTorch y GPU)

En esta sección se describe cómo crear una imagen y utilizarla para los entrenamientos en ModelArts. El motor de IA utilizado en la imagen es Horovod 0.22.1 + PyTorch 1.8.1 y los recursos utilizados para el entrenamiento son GPU.

NOTA

Esta sección solo se aplica a los trabajos de entrenamiento de la nueva versión.

Escenario

En este ejemplo, escriba un Dockerfile para crear una imagen personalizada en un servidor Linux x86_64 que ejecute Ubuntu 18.04.

Objetivo: crear e instalar imágenes de contenedor del siguiente software y utilizar las imágenes y las CPU/GPU para entrenamiento en ModelArts.

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

Procedimiento

Antes de utilizar una imagen personalizada para crear un trabajo de entrenamiento, debe estar familiarizado con Docker y tener experiencia en desarrollo.

1. **Requisitos previos**
2. **Paso 1 Crear un bucket de OBS y una carpeta**
3. **Paso 2 Preparar el script de entrenamiento y cargarlo en OBS**
4. **Paso 3 Preparar un servidor**
5. **Paso 4 Crear una imagen personalizada**
6. **Paso 5 Cargar la imagen en SWR**
7. **Paso 6 Crear un trabajo de entrenamiento en ModelArts**

Requisitos previos

Ha creado una cuenta en Huawei Cloud. La cuenta no está en mora ni congelada.

Paso 1 Crear un bucket de OBS y una carpeta

Cree un bucket y unas carpetas en OBS para almacenar la muestra de conjunto de datos y el código de entrenamiento. **Tabla 4-3** enumera las carpetas que se van a crear. En el ejemplo, el nombre del bucket y los nombres de las carpetas junto con los nombres reales.

Para obtener más información sobre cómo crear un bucket de OBS y una carpeta, véase **Creación de un bucket** y **Creación de una carpeta**.

Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región.

Tabla 4-3 Carpeta para crear

Nombre	Descripción
obs://test-modelarts/pytorch/demo-code/	Almacena el script de entrenamiento.
obs://test-modelarts/pytorch/log/	Almacena los archivos de log de entrenamiento.

Paso 2 Preparar el script de entrenamiento y cargarlo en OBS

Obtenga scripts de entrenamiento **pytorch_synthetic_benchmark.py** y **run_mpi.sh** y cárguelos a **obs://test-modelarts/horovod/demo-code/** en el bucket de OBS.

pytorch_synthetic_benchmark.py es el siguiente:

```
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 es el siguiente:

```
#!/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
```



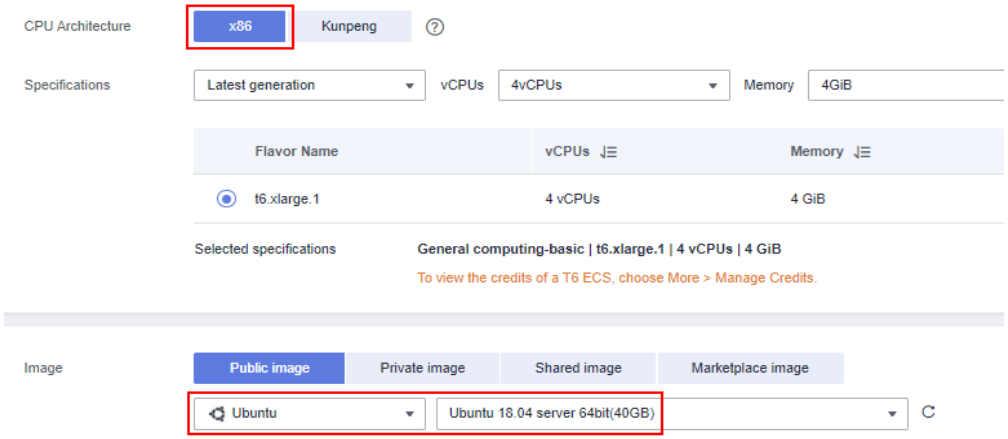
```
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

Paso 3 Preparar un servidor

Obtener un servidor de Linux x86_64 que ejecute Ubuntu 18.04. Un ECS o su PC local servirán.

Para obtener más información sobre cómo comprar un ECS, véase **Compra e inicio de sesión en un ECS de Linux**. Seleccione una imagen pública. Se recomienda una imagen de Ubuntu 18.04.

Figura 4-16 Creación de un ECS con una imagen pública (x86)



Paso 4 Crear una imagen personalizada

Cree una imagen de contenedor con las siguientes configuraciones y utilice la imagen para crear un trabajo de entrenamiento en ModelArts:

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

A continuación se describe cómo crear una imagen personalizada escribiendo un Dockerfile.

1. Instale Docker.

   A continuación se utiliza Linux x86_64 OS como ejemplo para describir cómo obtener el paquete de instalación de Docker. Para obtener detalles sobre cómo instalar Docker, consulte los **documentos oficiales de Docker**. Ejecute los siguientes comandos para instalar Docker:


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

Si se ejecuta el comando **docker images**, se ha instalado Docker. Si es así, omita este paso.

2. Verifique la versión del motor de Docker. Ejecute el siguiente comando:

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

Se muestra la siguiente información:

```
 Engine:
  Version:          18.09.0
```

NOTA

Utilice el motor de Docker de la versión anterior o posterior para crear una imagen personalizada.

3. Cree una carpeta denominada **context**.

```
mkdir -p context
```

4. Obtenga el archivo **pip.conf**. En este ejemplo, se utiliza el origen pip proporcionado por Huawei Mirrors, que es el siguiente:

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

NOTA

Para obtener **pip.conf**, acceda a Huawei Mirrors en **https://mirrors.huaweicloud.com/home** y busque **pypi**.

5. Descargue el archivo de código de Horovod de fuente.

Descargue **horovod-0.22.1.tar.gz** desde https://pypi.org/project/horovod/0.22.1/#files.

6. Descargue los archivos torch*.whl.

Descargue los siguientes archivos .whl desde 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

El código URL del signo más (+) es %2B. Cuando busque archivos en los sitios web anteriores, sustituya el signo más (+) del nombre del archivo por %2B. Por ejemplo, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

7. Descargue el archivo de instalación de Miniconda3.

Descargue el archivo de instalación de Miniconda3 py37 4.12.0 (Python 3.7.13) desde https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh.

8. Escriba la imagen del contenedor Dockerfile.

Cree un archivo vacío denominado **Dockerfile** en la carpeta **context** y copie el siguiente contenido en el archivo:

```
# 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 obtener detalles sobre cómo escribir un Dockerfile, consulte los **documentos oficiales de Docker**.

9. Descargue el paquete de instalación de MLNX_OFED.

   Vaya a **Linux Drivers**. En la ficha **Download**, seleccione los paquetes de instalación de **Current Versions** y **Archive Versions**. En este ejemplo, seleccione **Archive Versions**, configure **Version** en **5.4-3.5.8.0-LTS**, **OS Distribution** en **Ubuntu**, **OS Distribution Version** en **Ubuntu 18.04**, **Architecture** en **x86_64** y descargue el paquete de instalación **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

10. Descarga de **openmpi-3.0.0-bin.tar.gz**.

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

11. Almacene el archivo de origen de pip, el archivo torch*.whl y el archivo de instalación de Miniconda3 en la carpeta **context**, que es la siguiente:

```
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. Cree la imagen de contenedor. Ejecute el siguiente comando en el directorio donde se almacena el Dockerfile para crear la imagen de contenedor **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
```

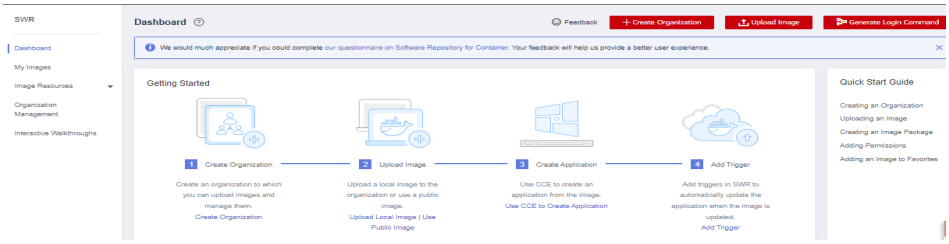
    El siguiente log muestra que se ha creado la imagen.

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

Paso 5 Cargar la imagen en SWR

1. Inicie sesión en la consola de SWR y seleccione la región de destino.

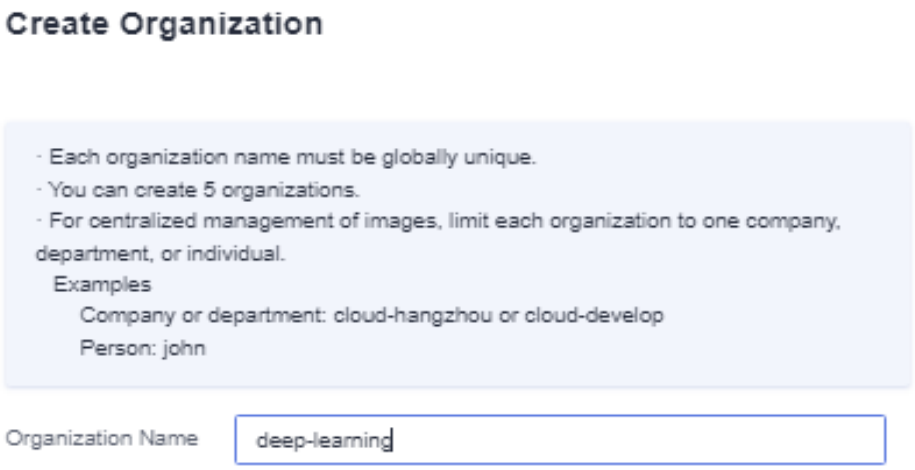
   Figura 4-17 Consola de SWR



2. Haga clic en **Create Organization** en la esquina superior derecha e introduzca un nombre de organización para crear una organización. Personalice el nombre de la

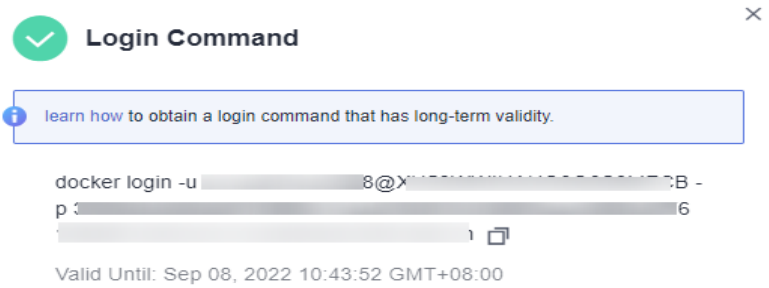
organización. Sustituya el nombre de la organización **deep-learning** en comandos posteriores con el nombre real de la organización.

Figura 4-18 Creación de una organización



3. Haga clic en **Generate Login Command** en la esquina superior derecha para obtener un comando de acceso.

Figura 4-19 Comando de acceso



4. Inicie sesión en el entorno local como usuario **root** e ingrese el comando de inicio de sesión.
5. Cargue la imagen en SWR.
   a. Etiquete la imagen cargada.
   ```
   # 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. Ejecute el siguiente comando para subir la imagen:
   ```
   # 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. Después de cargar la imagen, elija **My Images** en el panel de navegación a la izquierda de la consola de SWR para ver las imágenes personalizadas cargadas.

Paso 6 Crear un trabajo de entrenamiento en ModelArts

1. Inicie sesión en la consola de gestión de ModelArts y compruebe si se ha configurado la autorización de acceso para su cuenta. Para obtener más información, véase

Configuración de la autorización de la delegación. Si ha sido autorizado mediante claves de acceso, borre la autorización y configure la autorización de la delegación.

2. En el panel de navegación, seleccione **Training Management** > **Training Jobs**. La lista de trabajos de entrenamiento se muestra de forma predeterminada.
3. Haga clic en **Create Training Job**. En la página que aparece en pantalla, configure los parámetros y haga clic en **Next**.
   - **Created By**: **Algoritmos personalizados**
   - **Boot Mode**: **Imágenes personalizadas**
   - Image path: imagen creada en **Paso 5 Cargar la imagen en SWR**.
   - **Code Directory**: directorio donde se almacena el archivo de script de arranque en OBS. Por ejemplo, **obs://test-modelarts/pytorch/demo-code/**. El código de entrenamiento se descarga automáticamente en el directorio **${MA_JOB_DIR}/demo-code** del contenedor de entrenamiento. **demo-code** (personalizable) es el directorio de último nivel de la ruta del OBS.
   - **Boot Command**: **bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/pytorch_synthetic_benchmark.py**. **demo-code** (personalizable) es el directorio de último nivel de la ruta del OBS.
   - **Environment Variable**: Haga clic en **Add Environment Variable** y agregue la variable de entorno **MY_SSHD_PORT=38888**.
   - **Resource Pool**: seleccione **Public resource pools**.
   - **Resource Type**: seleccione **GPU**.
   - **Compute Nodes**: 1 o 2
   - **Persistent Log Saving**: habilitado
   - **Job Log Path**: ruta del OBS a los logs de entrenamiento almacenados, por ejemplo, **obs://test-modelarts/pytorch/log/**
4. Confirme las configuraciones del trabajo de entrenamiento y haga clic en **Submit**.
5. Espere hasta que se cree el trabajo de entrenamiento.

   Después de enviar la solicitud de creación de trabajo, el sistema realizará automáticamente operaciones en el backend, como descargar la imagen del contenedor y el directorio de código y ejecutar el comando de arranque. Un trabajo de entrenamiento requiere un cierto período de tiempo para ejecutarse. La duración oscila entre decenas de minutos y varias horas, dependiendo de la lógica del servicio y de los recursos seleccionados. Una vez ejecutado el trabajo de entrenamiento, se muestra un log similar al siguiente.

Figura 4-20 Ejecutar logs de trabajos de entrenamiento con especificaciones de GPU (un nodo de cómputo)

```
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 Ejecutar logs de trabajos de entrenamiento con especificaciones de GPU (dos nodos de cómputo)

```
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 Ejemplo: creación de una imagen personalizada para entrenamiento (MindSpore y GPU)

En esta sección se describe cómo crear una imagen y utilizarla para los entrenamientos en ModelArts. El motor de IA utilizado en la imagen es MindSpore y los recursos utilizados para el entrenamiento son las GPU.

NOTA

Esta sección solo se aplica a los trabajos de entrenamiento de la nueva versión.

Escenario

En este ejemplo, escriba un Dockerfile para crear una imagen personalizada en un servidor Linux x86_64 que ejecute Ubuntu 18.04.

Cree una imagen contenedora con las siguientes configuraciones y utilícela para crear un trabajo de entrenamiento basado en GPU en ModelArts:

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

Procedimiento

Antes de utilizar una imagen personalizada para crear un trabajo de entrenamiento, debe estar familiarizado con Docker y tener experiencia en desarrollo.

- **Requisitos previos**
- **Paso 1 Crear un bucket de OBS y una carpeta**
- **Paso 2 Crear un conjunto de datos y cargarlo en OBS**
- **Paso 3 Preparar el script de entrenamiento y cargarlo en OBS**
- **Paso 4 Preparar un servidor**
- **Paso 5 Crear una imagen personalizada**
- **Paso 6 Cargar la imagen en SWR**
- **Paso 7 Crear un trabajo de entrenamiento en ModelArts**

Requisitos previos

Ha creado una cuenta en Huawei Cloud. La cuenta no está en mora ni congelada.

Paso 1 Crear un bucket de OBS y una carpeta

Cree un bucket y unas carpetas en OBS para almacenar la muestra de conjunto de datos y el código de entrenamiento. **Tabla 4-4** enumera las carpetas que se van a crear. En el ejemplo, el nombre del bucket y los nombres de las carpetas junto con los nombres reales.

Para obtener más información, véase **Creación de un bucket** y **Creación de una carpeta**.

Asegúrese de que OBS y ModelArts se encuentren en la misma región.

Tabla 4-4 Carpetas de OBS requeridas

Carpeta	Descripción
obs://test-modelarts/mindspore-gpu/resnet/	Almacena el script de entrenamiento.
obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/	Almacena archivos de conjuntos de datos.

Carpeta	Descripción
obs://test-modelarts/mindspore-gpu/output/	Almacena archivos de salida de entrenamiento.
obs://test-modelarts/mindspore-gpu/log/	Almacena archivos de log de entrenamiento.

Paso 2 Crear un conjunto de datos y cargarlo en OBS

Vaya a http://www.cs.toronto.edu/~kriz/cifar.html, descargue el paquete **CIFAR-10 binary version (suitable for C programs)**, descomprímalo y cargue los datos de decomposición en el directorio **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/** del bucket del OBS.

Paso 3 Preparar el script de entrenamiento y cargarlo en OBS

Obtenga el archivo de ResNet y el script **run_mpi.sh** y cárguelos en **obs://test-modelarts/mindspore-gpu/ResNet/** en el bucket de OBS.

Descargue el archivo de ResNet desde **https://gitee.com/mindspore/models/tree/r1.8/official/cv/resnet**.

run_mpi.sh es el siguiente:

```
#!/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
```

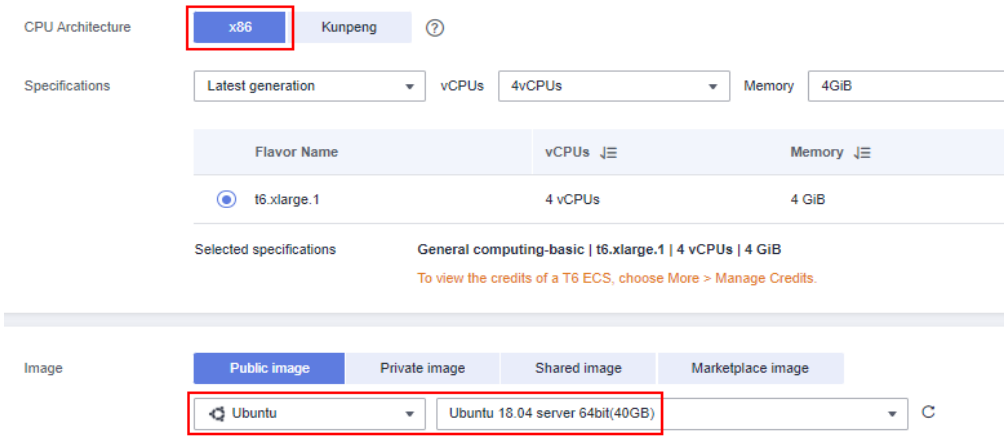
La carpeta **obs://test-modelarts/mindspore-gpu/resnet/** contiene los archivos **resnet** y **run_mpi.sh**.

Paso 4 Preparar un servidor

Obtener un servidor de Linux x86_64 que ejecute Ubuntu 18.04. Un ECS o su PC local servirán.

Para obtener más información sobre cómo comprar un ECS, véase **Compra e inicio de sesión en un ECS de Linux**. Seleccione una imagen pública. Se recomienda una imagen de Ubuntu 18.04.

Figura 4-22 Creación de un ECS con una imagen pública (x86)



Paso 5 Crear una imagen personalizada

Cree una imagen de contenedor con las siguientes configuraciones y utilice la imagen para crear un trabajo de entrenamiento en ModelArts:

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4

- mindspore gpu-1.8.1

En esta sección se describe cómo escribir un Dockerfile para crear una imagen personalizada.

1. Instale Docker.

   A continuación se utiliza Linux x86_64 como ejemplo para describir cómo obtener un paquete de instalación de Docker. Para obtener más detalles sobre cómo instalar Docker, consulte los **documentos oficiales de Docker**. Ejecute el siguiente comando para instalar Docker:

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

   Si se puede ejecutar el comando **docker images**, Docker se ha instalado. Si es así, omita este paso.

2. Verifique la versión del motor de Docker. Ejecute el siguiente comando:

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

   Se muestra la siguiente información:

   ```
   Engine:
    Version:          18.09.0
   ```

   NOTA

   Utilice el motor de Docker de la versión anterior o posterior para crear una imagen personalizada.

3. Cree una carpeta denominada **context**.

   ```
   mkdir -p context
   ```

4. Obtenga el archivo **pip.conf**. En este ejemplo, se utiliza el origen pip proporcionado por Huawei Mirrors, que es el siguiente:

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

   NOTA

   Para obtener **pip.conf**, pase a Huawei Mirrors **https://mirrors.huaweicloud.com/home** y busque **pypi**.

5. Descargue **mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl** desde **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. Descargue el archivo de instalación de Miniconda3.

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

7. Escriba la imagen del contenedor Dockerfile.

   Cree un archivo vacío denominado **Dockerfile** en la carpeta **context** y copie el siguiente contenido en el archivo:

   ```
   # 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 obtener detalles sobre cómo escribir un Dockerfile, consulte los **documentos oficiales de Docker**.

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

   Vaya a https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/ y haga clic en **Download**, configure **Version** en **5.4-3.5.8.0-LTS**, **OSDistributionVersion** en **Ubuntu 18.04** y **Architecture** en **x86_64** y descargue **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Descarga de **openmpi-3.0.0-bin.tar.gz**.

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

10. Almacene el archivo de instalación de Dockerfile y de Miniconda3 en la carpeta **context**, que es la siguiente:

    ```
    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. Cree la imagen de contenedor. Ejecute el siguiente comando en el directorio donde se almacena Dockerfile para crear la imagen de contenedor **mindspore:1.8.1-ofed-cuda11.1**:

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

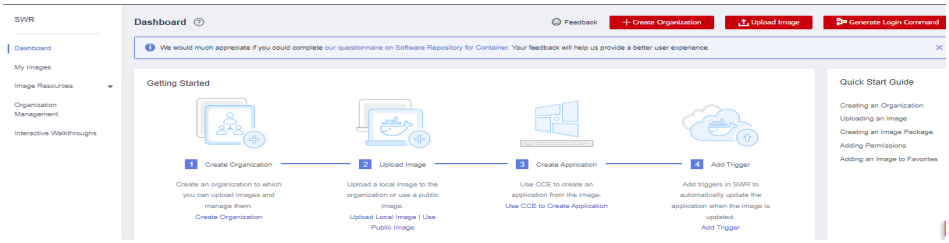

El siguiente log muestra que se ha creado la imagen.

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

Paso 6 Cargar la imagen en SWR

1. Inicie sesión en la consola de SWR y seleccione la región de destino.

   Figura 4-23 Consola de SWR

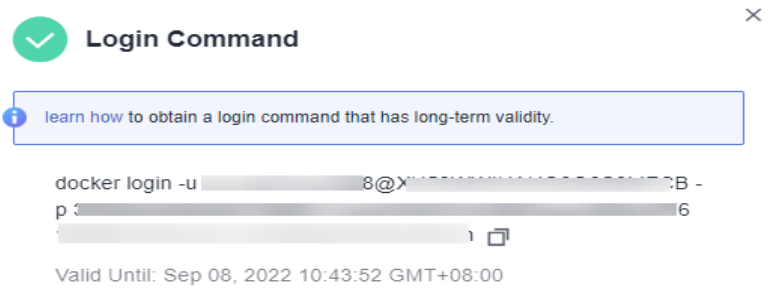
2. Haga clic en **Create Organization** en la esquina superior derecha e introduzca un nombre de organización para crear una organización. Personalice el nombre de la organización. Sustituya el nombre de la organización **deep-learning** en comandos posteriores con el nombre real de la organización.

   Figura 4-24 Creación de una organización

3. Haga clic en **Generate Login Command** en la esquina superior derecha para obtener un comando de acceso.

   Figura 4-25 Comando de acceso

4. Inicie sesión en el entorno local como usuario **root** e ingrese el comando de inicio de sesión.

5. Cargue la imagen en SWR.

   a. Etiquete la imagen cargada.

   ```
   # 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. Ejecute el siguiente comando para subir la imagen:

   ```
   # 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. Después de cargar la imagen, elija **My Images** en el panel de navegación a la izquierda de la consola de SWR para ver las imágenes personalizadas cargadas.

Paso 7 Crear un trabajo de entrenamiento en ModelArts

1. Inicie sesión en la consola de gestión de ModelArts y compruebe si se ha configurado la autorización de acceso para su cuenta. Para obtener más información, véase **Configuración de la autorización de la delegación**. Si ha sido autorizado mediante claves de acceso, borre la autorización y configure la autorización de la delegación.
2. En el panel de navegación, seleccione **Training Management** > **Training Jobs**. La lista de trabajos de entrenamiento se muestra de forma predeterminada.
3. Haga clic en **Create Training Job**. En la página que aparece en pantalla, configure los parámetros y haga clic en **Next**.
   - **Created By**: **Algoritmos personalizados**
   - **Boot Mode**: **Imágenes personalizadas**
   - Image path: imagen creada en **Paso 6 Cargar la imagen en SWR**.
   - **Code Directory**: directorio donde se almacena el archivo de script de arranque en OBS, por ejemplo, **obs://test-modelarts/mindspore-gpu/resnet/**. El código de entrenamiento se descarga automáticamente en el directorio **${MA_JOB_DIR}/resnet** del contenedor de entrenamiento. **resnet** (personalizable) es el directorio de último nivel de la ruta de OBS.
   - **Boot Command**: **bash ${MA_JOB_DIR}/resnet/run_mpi.sh python ${MA_JOB_DIR}/resnet/train.py**. **resnet** (personalizable) es el directorio de último nivel de la ruta de OBS.
   - **Training Input**: haga clic en **Add Training Input**. Escriba **data_path** como nombre, seleccione la ruta de OBS al conjunto de datos de destino, por ejemplo, **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/** y configure **Obtained from** como **Hyperparameters**.
   - **Training Output**: haga clic en **Add Training Output**. Escriba **output_path** como nombre, seleccione una ruta de OBS para almacenar salidas de entrenamiento, por ejemplo **obs://test-modelarts/mindspore-gpu/output/** y configure **Obtained from** como **Hyperparameters** y **Predownload** como **No**.
   - **Hyperparameters**: haga clic en **Add Hyperparameter** y agregue los siguientes hiperparámetros:
     - run_distribute=True
     - device_num=1 (Set this parameter based on the number of GPUs in the instance flavors.)
     - device_target=GPU

    - epoch_size=2
  - **Environment Variable**: Haga clic en **Add Environment Variable** y agregue la variable de entorno **MY_SSHD_PORT=38888**.
  - **Resource Pool**: seleccione **Public resource pools**.
  - **Resource Type**: seleccione **GPU**.
  - **Compute Nodes**: 1 o 2
  - **Persistent Log Saving**: habilitado
  - **Job Log Path**: ruta de OBS a los logs de entrenamiento almacenados, por ejemplo, **obs://test-modelarts/mindspore-gpu/log/**
4. Confirme las configuraciones del trabajo de entrenamiento y haga clic en **Submit**.
5. Espere hasta que se cree el trabajo de entrenamiento.

   Después de enviar la solicitud de creación de trabajo, el sistema realizará automáticamente operaciones en el backend, como descargar la imagen del contenedor y el directorio de código y ejecutar el comando de arranque. Un trabajo de entrenamiento requiere un cierto período de tiempo para ejecutarse. La duración oscila entre decenas de minutos y varias horas, dependiendo de la lógica del servicio y de los recursos seleccionados. Una vez ejecutado el trabajo de entrenamiento, se muestra un log similar al siguiente.

   Figura 4-26 Ejecutar logs de trabajos de entrenamiento con especificaciones de GPU (un nodo de cómputo)

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

   Figura 4-27 Ejecutar logs de trabajos de entrenamiento con especificaciones de GPU (dos nodos de cómputo)

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

4.2.5 Ejemplo: creación de una imagen personalizada para entrenamiento (TensorFlow y GPU)

En esta sección se describe cómo crear una imagen y utilizarla para los entrenamientos en ModelArts. El motor de IA utilizado en la imagen es TensorFlow y los recursos utilizados para el entrenamiento son las GPU.

NOTA

Esta sección solo se aplica a los trabajos de entrenamiento de la nueva versión.

Escenario

En este ejemplo, escriba un Dockerfile para crear una imagen personalizada en un servidor Linux x86_64 que ejecute Ubuntu 18.04.

Cree una imagen contenedora con las siguientes configuraciones y utilícela para crear un trabajo de entrenamiento basado en GPU en ModelArts:

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

Procedimiento

Antes de utilizar una imagen personalizada para crear un trabajo de entrenamiento, debe estar familiarizado con Docker y tener experiencia en desarrollo.

1. **Requisitos previos**
2. **Paso 1 Crear un bucket de OBS y una carpeta**
3. **Paso 2 Crear un conjunto de datos y cargarlo en OBS**
4. **Paso 3 Preparar el script de entrenamiento y cargarlo en OBS**
5. **Paso 4 Preparar un servidor**
6. **Paso 5 Crear una imagen personalizada**
7. **Paso 6 Cargar la imagen en SWR**
8. **Paso 7 Crear un trabajo de entrenamiento en ModelArts**

Requisitos previos

Ha creado una cuenta en Huawei Cloud. La cuenta no está en mora ni congelada.

Paso 1 Crear un bucket de OBS y una carpeta

Cree un bucket y unas carpetas en OBS para almacenar la muestra de conjunto de datos y el código de entrenamiento. **Tabla 4-5** enumera las carpetas que se van a crear. En el ejemplo, el nombre del bucket y los nombres de las carpetas junto con los nombres reales.

Para obtener más información sobre cómo crear un bucket de OBS y una carpeta, véase **Creación de un bucket** y **Creación de una carpeta**.

Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región.

Tabla 4-5 Carpetas de OBS requeridas

Carpeta	Descripción
obs://test-modelarts/tensorflow/code/	Almacena el script de entrenamiento.
obs://test-modelarts/tensorflow/data/	Almacena archivos de conjuntos de datos.
obs://test-modelarts/tensorflow/log/	Almacena archivos de log de entrenamiento.

Paso 2 Crear un conjunto de datos y cargarlo en OBS

Descargue **mnist.npz** desde **https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz** y cárguelo en **obs://test-modelarts/tensorflow/data/** en el bucket de OBS.

Paso 3 Preparar el script de entrenamiento y cargarlo en OBS

Obtenga el script de entrenamiento **mnist.py** y cárguelo en **obs://test-modelarts/tensorflow/code/** en el bucket de OBS.

mnist.py es el siguiente:

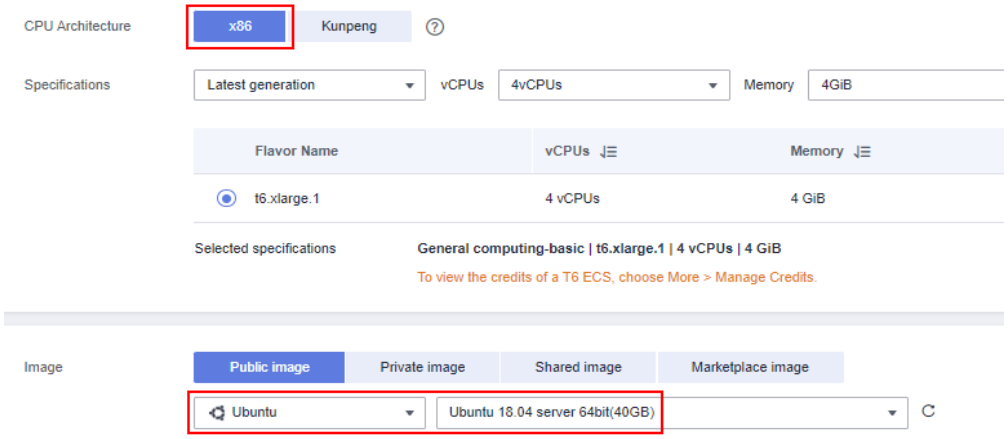
```
import argparse
import tensorflow as tf

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)
```

Paso 4 Preparar un servidor

Obtener un servidor de Linux x86_64 que ejecute Ubuntu 18.04. Un ECS o su PC local servirán.

Para obtener más información sobre cómo comprar un ECS, véase **Compra e inicio de sesión en un ECS de Linux**. Seleccione una imagen pública. Se recomienda una imagen de Ubuntu 18.04.

Figura 4-28 Creación de un ECS con una imagen pública (x86)



Paso 5 Crear una imagen personalizada

Cree una imagen de contenedor con las siguientes configuraciones y utilice la imagen para crear un trabajo de entrenamiento en ModelArts:

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

A continuación se describe cómo crear una imagen personalizada escribiendo un Dockerfile.

1. Instale Docker.

   A continuación se utiliza Linux x86_64 OS como ejemplo para describir cómo obtener el paquete de instalación de Docker. Para obtener detalles sobre cómo instalar Docker, consulte los **documentos oficiales de Docker**. Ejecute los siguientes comandos para instalar Docker:

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

   Si se ejecuta el comando **docker images**, se ha instalado Docker. Si es así, omita este paso.

2. Verifique la versión del motor de Docker. Ejecute el siguiente comando:

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

   Se muestra la siguiente información:

   ```
   Engine:
    Version:          18.09.0
   ```

   NOTA

   Utilice el motor de Docker de la versión anterior o posterior para crear una imagen personalizada.

3. Cree una carpeta denominada **context**.

   ```
   mkdir -p context
   ```

4. Obtenga el archivo **pip.conf**. En este ejemplo, se utiliza el origen pip proporcionado por Huawei Mirrors, que es el siguiente:

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



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

NOTA

Para obtener **pip.conf**, acceda a Huawei Mirrors en https://mirrors.huaweicloud.com/home y busque **pypi**.

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

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

6. Descargue el archivo de instalación de Miniconda3.

   Descargue el archivo de instalación de Miniconda3 py37 4.12.0 (Python 3.7.13) desde https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh.

7. Escriba la imagen del contenedor Dockerfile.

   Cree un archivo vacío denominado **Dockerfile** en la carpeta **context** y copie el siguiente contenido en el archivo:

```
# 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 obtener detalles sobre cómo escribir un Dockerfile, consulte los **documentos oficiales de Docker**.

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

   Vaya a **Linux Drivers**. En la ficha **Download**, configure **Version** en **5.4-3.5.8.0-LTS**, **OS Distribution Version** en **Ubuntu 18.04**, **Architecture** en **x86_64** y descargue **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Almacene el archivo de instalación de Dockerfile y de Miniconda3 en la carpeta **context**, que es la siguiente:

```
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. Cree la imagen de contenedor. Ejecute el siguiente comando en el directorio donde se almacena el Dockerfile para crear la imagen de contenedor **tensorflow:2.10.0-ofed-cuda11.2**:


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

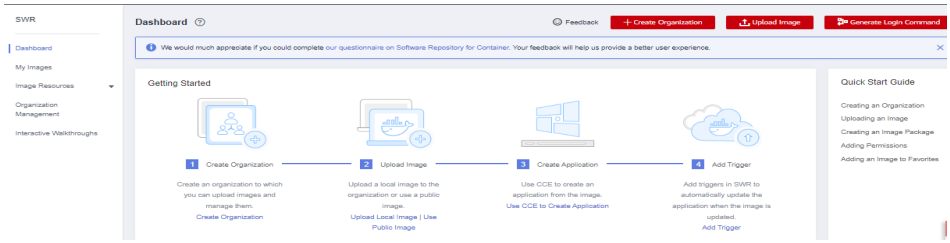
El siguiente log muestra que se ha creado la imagen.

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

Paso 6 Cargar la imagen en SWR

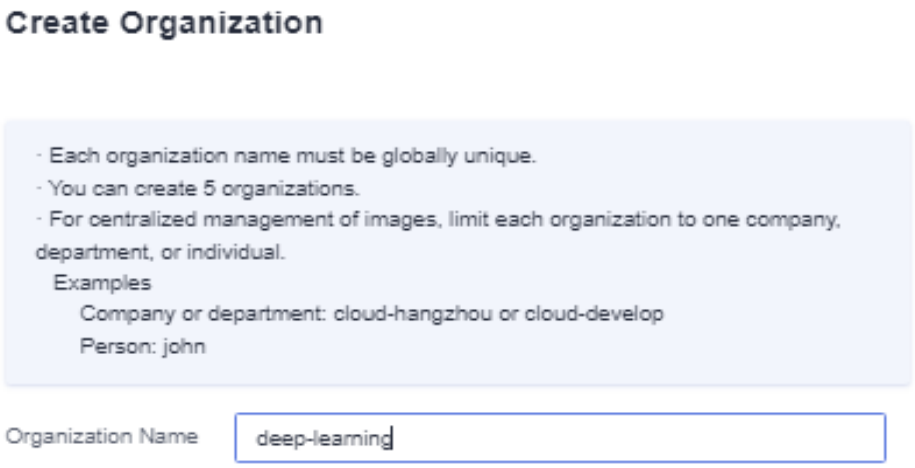
1. Inicie sesión en la consola de SWR y seleccione la región de destino.

Figura 4-29 Consola de SWR



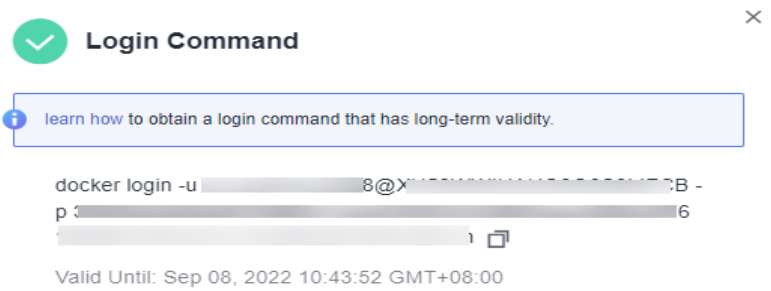
2. Haga clic en **Create Organization** en la esquina superior derecha e introduzca un nombre de organización para crear una organización. Personalice el nombre de la organización. Sustituya el nombre de la organización **deep-learning** en comandos posteriores con el nombre real de la organización.

Figura 4-30 Creación de una organización



3. Haga clic en **Generate Login Command** en la esquina superior derecha para obtener un comando de acceso.

Figura 4-31 Comando de acceso



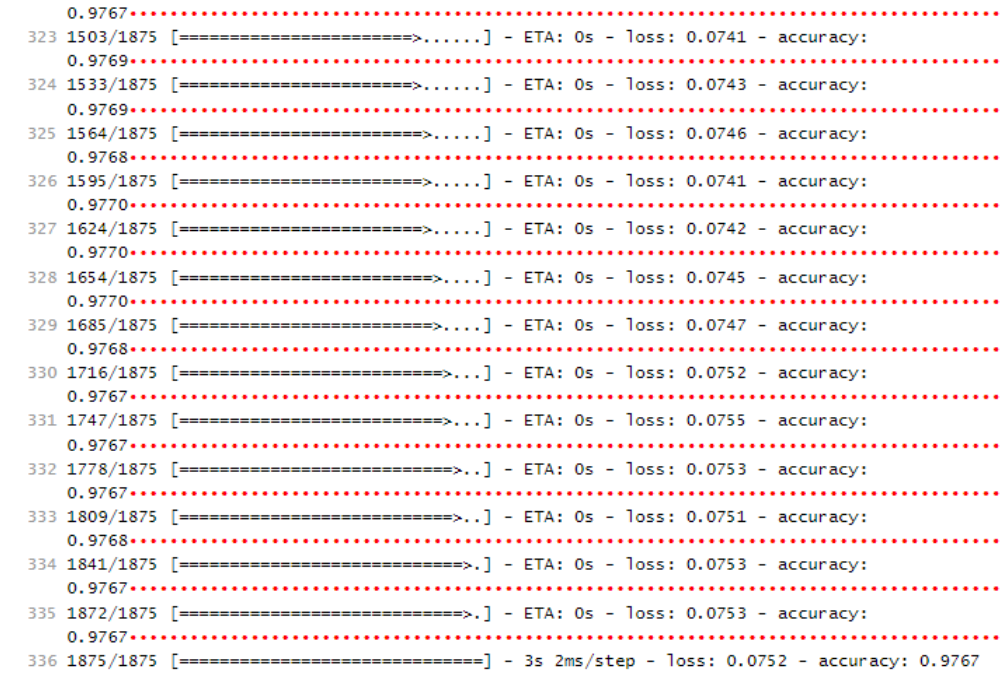
4. Inicie sesión en el entorno local como usuario **root** e ingrese el comando de inicio de sesión.
5. Cargue la imagen en SWR.
   a. Etiquete la imagen cargada.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
   ```
   b. Ejecute el siguiente comando para subir la imagen:
   ```
   # 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. Después de cargar la imagen, elija **My Images** en el panel de navegación a la izquierda de la consola de SWR para ver las imágenes personalizadas cargadas.

Paso 7 Crear un trabajo de entrenamiento en ModelArts

1. Inicie sesión en la consola de gestión de ModelArts y compruebe si se ha configurado la autorización de acceso para su cuenta. Para obtener más información, véase **Configuración de la autorización de la delegación**. Si ha sido autorizado mediante claves de acceso, borre la autorización y configure la autorización de la delegación.
2. En el panel de navegación, seleccione **Training Management** > **Training Jobs**. La lista de trabajos de entrenamiento se muestra de forma predeterminada.
3. Haga clic en **Create Training Job**. En la página que aparece en pantalla, configure los parámetros y haga clic en **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: imagen creada en **Paso 5 Crear una imagen personalizada**.
   - **Code Directory**: directorio donde se almacena el archivo de script de inicio en OBS, por ejemplo, **obs://test-modelarts/tensorflow/code/**. El código de entrenamiento se descarga automáticamente en el directorio **${MA_JOB_DIR}/code** del contenedor de entrenamiento. **code** (personalizable) es el directorio de último nivel de la ruta del OBS.
   - **Boot Command**: **python ${MA_JOB_DIR}/code/mnist.py**. **code** (personalizable) es el directorio de último nivel de la ruta del OBS.
   - **Training Input**: haga clic en **Add Training Input**. Escriba **data_path** como nombre, seleccione la ruta del OBS a **mnist.npz**. Por ejemplo, **obs://test-modelarts/tensorflow/data/mnist.npz** y configure **Obtained from** como **Hyperparameters**.
   - **Resource Pool**: seleccione **Public resource pools**.
   - **Resource Type**: seleccione **GPU**.
   - **Compute Nodes**: Ingresar **1**.
   - **Persistent Log Saving**: habilitado
   - **Job Log Path**: ruta de OBS a los logs de entrenamiento almacenados, por ejemplo, **obs://test-modelarts/mindspore-gpu/log/**
4. Confirme las configuraciones del trabajo de entrenamiento y haga clic en **Submit**.
5. Espere hasta que se cree el trabajo de entrenamiento.

Después de enviar la solicitud de creación de trabajo, el sistema realizará automáticamente operaciones en el backend, como descargar la imagen del contenedor y el directorio de código y ejecutar el comando de arranque. Un trabajo de entrenamiento requiere un cierto período de tiempo para ejecutarse. La duración oscila entre decenas de minutos y varias horas, dependiendo de la lógica del servicio y de los recursos seleccionados. Una vez ejecutado el trabajo de entrenamiento, se muestra un log similar al siguiente.

Figura 4-32 Ejecutar logs de trabajos de entrenamiento con especificaciones de GPU



4.3 Preparación de una imagen de entrenamiento

4.3.1 Especificaciones a las Imágenes personalizadas para trabajos de entrenamiento

Cuando utilice un modelo desarrollado localmente y un script de entrenamiento para crear una imagen personalizada, asegúrese de que la imagen personalizada cumple las especificaciones definidas por ModelArts.

Especificaciones

- Utilice Ubuntu 18.04 para las imágenes personalizadas en caso de que las versiones no sean compatibles.
- No utilice una imagen personalizada de más de 15 GB. El tamaño no debe exceder la mitad del espacio del motor del contenedor del grupo de recursos. De lo contrario, la hora de inicio del trabajo de entrenamiento se ve afectada.

  El espacio del motor de contenedor del grupo de recursos público de ModelArts es de 50 GB. Por defecto, el espacio del motor del contenedor del recurso de grupo dedicado también es de 50 GB. Puede personalizar el espacio del motor del contenedor al crear un grupo de recursos dedicado.

- El **uid** del usuario predeterminado de una imagen personalizada debe ser **1000**.
- El controlador de GPU o de Ascend no se puede instalar en una imagen personalizada. Al seleccionar recursos de GPU para ejecutar trabajos de entrenamiento, ModelArts coloca automáticamente el controlador de GPU en el directorio **/usr/local/nvidia** en el entorno de entrenamiento. Cuando selecciona recursos de Ascend para ejecutar trabajos de entrenamiento, ModelArts coloca automáticamente el controlador de Ascend en el directorio **/usr/local/Ascend/driver**.
- Las imágenes personalizadas basadas en x86 o Arm pueden ejecutarse solo con las especificaciones correspondientes a su arquitectura.
  - Ejecute el siguiente comando para comprobar la arquitectura de CPU de una imagen personalizada:
    ```
    docker inspect {Custom image path} | grep Architecture
    ```
    A continuación se muestra el resultado del comando para una imagen personalizada basada en Arm:
    ```
    "Architecture": "arm64"
    ```
  - Si el nombre de una especificación contiene **Arm**, esta especificación es una arquitectura de CPU basada en Arm.

  - Si el nombre de una especificación no contiene **Arm**, esta especificación es una arquitectura de CPU basada en x86.

- ModelArts no admite la descarga de paquetes de instalación del código abierto. Instale los paquetes de dependencias requeridos por el trabajo de entrenamiento en la imagen personalizada.

4.3.2 Migración de una imagen al entrenamiento de ModelArts

Esta sección describe cómo migrar una imagen a la gestión de entrenamiento.

1. Agregue el grupo de usuarios predeterminado **ma-group** (**gid = 100**) de la gestión de entrenamiento para la imagen.

   NOTA

   Si ya existe el grupo de usuarios cuyo **gid** es **100**, puede aparecer en pantalla el mensaje de error "groupadd: GID '100' ya existe". Puede utilizar el comando **cat /etc/group | grep 100** para verificar si existe el grupo de usuarios cuyo GID es 100.

   Si ya existe el grupo de usuarios cuyo **gid** es **100**, omita este paso y elimine el comando **RUN groupadd ma-group -g 100** del Dockerfile.

2. Agregue el usuario predeterminado **ma-user** (**uid = 1000**) de la gestión de entrenamiento para la imagen.

   NOTA

   Si ya existe el usuario cuyo **uid** es **1000**, puede aparecer en pantalla el mensaje de error "useradd: UID 1000 no es único". Puede utilizar el comando **cat /etc/passwd | grep 1000** para verificar si existe el usuario cuyo UID es 1000.

   Si ya existe el usuario cuyo **uid** es **1000**, omita este paso y elimine el comando **RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user** del Dockerfile.

3. Modifique los permisos de los archivos de la imagen para permitir que **ma-user** cuyo **uid** sea **1000** lea y escriba los archivos.

Puede modificar una imagen haciendo referencia al siguiente Dockerfile para que la imagen cumpla con las especificaciones para las imágenes personalizadas de la gestión de entrenamiento de la nueva versión.

```
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
```

Después de editar el Dockerfile, ejecute el siguiente comando para crear una nueva imagen:

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

Cargue la nueva imagen en SWR. Para más detalles, véase **¿Cómo puedo iniciar sesión en SWR y cargar imágenes en él?**

4.3.3 Uso de una imagen de base para crear una imagen de entrenamiento

ModelArts proporciona las imágenes de base impulsadas por el aprendizaje profundo, como imágenes de TensorFlow, de PyTorch y de MindSpore. En estas imágenes, se ha instalado el software obligatorio para ejecutar los trabajos de entrenamiento. Si el software de las imágenes de base no puede cumplir con los requisitos de servicio, cree nuevas imágenes basadas en las imágenes de base y utilice las nuevas imágenes para crear trabajos de entrenamiento.

Procedimiento

Realice las siguientes operaciones para crear una imagen con una imagen de base de entrenamiento:

1. Instale Docker. Si se ejecuta el comando **docker images**, se ha instalado Docker. Si es así, omita este paso.

   A continuación se utiliza Linux x86_64 como ejemplo para describir cómo obtener el paquete de instalación de Docker. Ejecute el siguiente comando para instalar Docker:

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

2. Cree una carpeta denominada **context**.

   ```
   mkdir -p context
   ```

3. Obtenga el archivo **pip.conf**.


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

4. Cree una nueva imagen basada en una imagen de base de entrenamiento proporcionada por ModelArts. Guarde el Dockerfile editado en la carpeta **context**. Para más detalles sobre cómo obtener una imagen de base de entrenamiento, véase **Imágenes de base de entrenamiento disponibles**.

```
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. Ejecute el siguiente comando en el directorio donde se almacena el Dockerfile para crear una imagen de contenedor, por ejemplo, **training:v1**:

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

6. Cargue la nueva imagen en SWR. Para más detalles, véase **¿Cómo puedo iniciar sesión en SWR y cargar imágenes en él?**.
7. Utilice la imagen personalizada para crear un trabajo de entrenamiento para ModelArts. Para más detalles, véase **Uso de una imagen personalizada para crear un trabajo de entrenamiento basado en CPU o GPU**.

4.3.4 Instalación de MLNX_OFED en una imagen de contenedor

Escenarios

La NIC de Mellanox Technologies se ha configurado en los servidores de GPU de ModelArts para admitir Remote Direct Memory Access (RDMA). Como resultado, puede instalar MLNX_OFED en la imagen de contenedor, lo que permite a **NCCL** aprovechar la NIC y mejorar la eficiencia de la comunicación entre los nodos.

Después de habilitar esta NIC para NCCL, NET/IB se utiliza para la comunicación entre los nodos. Si esta NIC no está habilitada, se utiliza NET/Socket para la comunicación entre los nodos. NET/IB es mejor que NET/Socket en términos de latencia y ancho de banda.

Tabla 4-6 Instalación de NIC de Mellanox Technologies y MLNX_OFED en servidores de GPU de ModelArts

Modelo de GPU	NIC de Mellanox Technologies	Versión de MLNX_OFED instalada	Versión de MLNX_OFED recomendada para la imagen de contenedor
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

Instalación de MLNX_OFED

Tome la imagen de contenedor de Ubuntu18.04 como ejemplo. El Dockerfile para instalar MLNX_OFED 4.9-6.0.6.0-LTS es el siguiente:

> **NOTA**
>
> El host que se utiliza para descargar archivos y crear imágenes de contenedor usando un Dockerfile debe ser capaz de conectarse a la red 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
```

Cree una imagen de contenedor según este ejemplo de comando:

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

Una vez creada la imagen del contenedor, ejecute el siguiente comando para obtener la versión de MLNX_OFED en la imagen de contenedor:

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

La salida de comandos es la siguiente:

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

4.4 Creación de un algoritmo mediante una imagen personalizada

Sus algoritmos desarrollados localmente o desarrollados utilizando otras herramientas se pueden cargar en ModelArts para una gestión unificada.

Entradas para crear un algoritmo

Puede crear un algoritmo usando una imagen personalizada de ModelArts de cualquiera de las siguientes maneras:

- Entrada 1: En la consola de ModelArts, elija **Algorithm Management** > **My algorithms**. Luego, cree un algoritmo y utilícelo en los trabajos de entrenamiento o publíquelo en AI Gallery.
- Entrada 2: En la consola de ModelArts, elija **Training Management** > **Training Jobs** y haga clic en **Create Training Job** para crear un algoritmo personalizado y enviar un trabajo de entrenamiento. Para más detalles, véase **Uso de una imagen personalizada para crear un trabajo de entrenamiento basado en CPU o GPU**.

Parámetros para crear un algoritmo

Tabla 4-7 Parámetros para crear un algoritmo

Parámetro	Descripción
Boot Mode	Seleccione **Custom images**. Este parámetro es obligatorio.
Image	URL de una imagen de SWR. Este parámetro es obligatorio. ● Imágenes privadas o imágenes compartidas: Haga clic en **Select** a la derecha para seleccionar una imagen de SWR. Asegúrese de que la imagen se ha cargado a SWR. Para más detalles, véase **¿Cómo puedo iniciar sesión en SWR y cargar imágenes en él?**. ● Imágenes públicas: También puede introducir manualmente la ruta de la imagen en el formato "<Organización a la que pertenece la imagen>/<Nombre de la imagen>" en SWR. No contenga el nombre de dominio (swr.<region>.xxx.com) en la ruta porque el sistema agregará automáticamente el nombre de dominio a la ruta. Por ejemplo: `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	Ruta de OBS para almacenar el código de entrenamiento. Este parámetro es opcional. Tome la ruta de OBS **obs://obs-bucket/training-test/demo-code** como ejemplo. El contenido de la ruta de OBS se descargará automáticamente a **${MA_JOB_DIR}/demo-code** en el contenedor de entrenamiento, y **demo-code** (personalizable) es el directorio de último nivel de la ruta de OBS.
Boot Command	Comando para arrancar una imagen. Este parámetro es obligatorio. El comando de arranque se ejecutará automáticamente después de descargar el directorio de código. ● Si el script de arranque de entrenamiento es un archivo .py, **train.py** por ejemplo, el comando boot puede ser **python ${MA_JOB_DIR}/demo-code/train.py**. ● Si el script de arranque de entrenamiento es un archivo .sh, **main.sh** por ejemplo, el comando boot puede ser **bash ${MA_JOB_DIR}/demo-code/main.sh**. Los puntos y coma (;) y ampersands (&&) se pueden usar para combinar varios comandos de arranque, pero no se admiten saltos de línea. **demo-code** (personalizable) en el comando de arranque es el directorio del último nivel de la ruta OBS.

Configuración de tuberías

Un algoritmo basado en imágenes preestablecidas obtiene datos de un bucket de OBS o conjunto de datos para el entrenamiento del modelo. La salida de entrenamiento se almacena en un bucket de OBS. Los parámetros de entrada y salida del código del algoritmo deben analizarse para permitir el intercambio de datos entre ModelArts y OBS. Para obtener más información sobre cómo desplegar el código para entrenamientos en ModelArts, consulte **Despliegue de un script personalizado**.

Cuando utilice una imagen preestablecida para crear un algoritmo, configure las canalizaciones de entrada y salida.

- Configuraciones de entrada

  Tabla 4-8 Configuraciones de entrada

Parámetro	Descripción
Parameter Name	Establezca el nombre basado en el parámetro de entrada de datos en su código de algoritmo. El parámetro de ruta de acceso del código debe ser el mismo que el parámetro de entrada de entrenamiento analizado en el código del algoritmo. De lo contrario, el código de algoritmo no puede obtener los datos de entrada. Por ejemplo, si utiliza **argparse** en el código del algoritmo para analizar **data_url** en la entrada de datos, configure el parámetro de entrada de datos en **data_url** al crear el algoritmo.
Description	Descripción personalizable del parámetro de entrada,
Obtained from	Fuente del parámetro de entrada. Puede seleccionar **Hyperparameters** (predeterminado) o **Environment variables**.
Constraints	Si los datos se obtienen de una ruta de almacenamiento o de un conjunto de datos de ModelArts. Si selecciona el conjunto de datos de ModelArts como el origen de datos, se agregan las siguientes restricciones: ● **Labeling Type**: para obtener más información, véase **Creación de un trabajo de etiquetado**. ● **Data Format**, que puede ser **Default**, **CarbonData** o ambos. **Default** indica el formato de manifiesto. ● **Data Segmentation**: solo está disponible para los conjuntos de datos de clasificación de imágenes, de detección de objetos, de clasificación de texto y de clasificación de sonido. Los valores posibles son **Segmented dataset**, **Dataset not segmented** y **Unlimited**. Para obtener más información, consulte **Publicación de una versión de datos**.
Add	Se permiten múltiples fuentes de entrada de datos.

- Configuraciones de salida

Tabla 4-9 Configuraciones de salida

Parámetro	Descripción
Parameter Name	Establezca el nombre basado en el parámetro de salida de datos en su código de algoritmo. El parámetro de ruta de acceso del código debe ser el mismo que el parámetro de salida de entrenamiento analizado en el código del algoritmo. De lo contrario, el código de algoritmo no puede obtener la ruta de salida. Por ejemplo, si utiliza **argparse** en el código del algoritmo para analizar **train_url** en la salida de datos, establezca el parámetro de salida de datos en **train_url** al crear el algoritmo.
Description	Descripción personalizable del parámetro de salida,
Obtained from	Fuente del parámetro de salida. Puede seleccionar **Hyperparameters** (predeterminado) o **Environment variables**.
Add	Se permiten múltiples rutas de salida de datos.

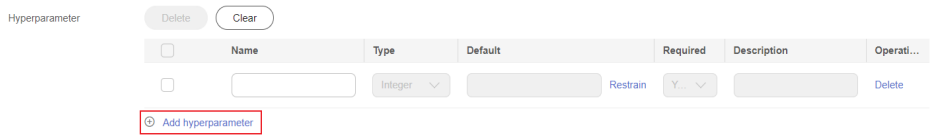
Definición de Hiperparámetros

Cuando se utiliza una imagen preestablecida para crear un algoritmo, ModelArts le permite personalizar los hiperparámetros para que pueda verlos o modificarlos en cualquier momento. Después de definir los hiperparámetros, se muestran en el comando de inicio y se transfieren al archivo de inicio como parámetros de CLI.

1. Importar hiperparámetros.

   Puede hacer clic en **Add hyperparameter** para agregar manualmente hiperparámetros.

   Figura 4-33 Adición de hiperparámetros

2. Editar hiperparámetros.

   Para más detalles, véase **Tabla 4-10**.

Tabla 4-10 Hiperparámetros

Parámetro	Descripción
Name	Nombre de hiperparámetro. Introduzca de 1 a 64 caracteres. Solo se permiten letras, dígitos, guiones medios (-) y guiones bajos (_).
Type	Tipo del hiperparámetro, que puede ser **String**, **Integer**, **Float** o **Boolean**
Default	Valor predeterminado del hiperparámetro, que se utiliza para los trabajos de entrenamiento de forma predeterminada

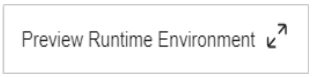
Parámetro	Descripción
Constraints	Haga clic en **Restrain**. A continuación, establezca el rango del valor predeterminado o el valor enumerado en el cuadro de diálogo que se muestra.
Required	Seleccione **Yes** o **No**. ● Si selecciona **No**, puede eliminar el hiperparámetro en la página de creación de trabajos de entrenamiento cuando utilice este algoritmo para crear un trabajo de entrenamiento. ● Si selecciona **Yes**, no puede suprimir el hiperparámetro de la página de creación de trabajos de entrenamiento cuando utilice este algoritmo para crear un trabajo de entrenamiento.
Description	Descripción del hiperparámetro Solo se permiten letras, dígitos, espacios, guiones (-), guiones bajos (_), comas (,) y puntos (.).

Adición de restricciones de entrenamiento

Puede agregar restricciones de entrenamiento del algoritmo en función de sus necesidades.

- **Resource Type**: Seleccione los tipos de recursos necesarios.
- **Multicard Training**: Seleccione si desea admitir el entrenamiento de varias tarjetas.
- **Distributed Training**: Seleccione si desea admitir el entrenamiento distribuido.

Vista previa del entorno en tiempo de ejecución

Al crear un algoritmo, haga clic en la flecha de 

en la esquina inferior derecha de la página para conocer la ruta del directorio de código, el archivo de arranque y los datos de entrada y salida en el contenedor de entrenamiento.

Procedimiento posterior

Después de crear un algoritmo, úselo para crear un trabajo de entrenamiento. Para más detalles, véase **Uso de una imagen personalizada para crear un trabajo de entrenamiento basado en CPU o GPU**.

4.5 Uso de una imagen personalizada para crear un trabajo de entrenamiento basado en CPU o GPU

El entrenamiento de modelos es un proceso de optimización iterativo. A través de la gestión de entrenamiento unificada, puede seleccionar de forma flexible algoritmos, datos e hiperparámetros para obtener la configuración y el modelo de entrada óptimos. Después de comparar métricas entre las versiones de trabajo, puede determinar el trabajo de entrenamiento más satisfactorio.

Requisitos previos

- Los datos que se van a entrenar se han subido a un directorio de OBS.
- Se ha creado al menos una carpeta vacía para almacenar la salida de entrenamiento en OBS.
- Se ha creado una imagen personalizada basada en las especificaciones de ModelArts. Para obtener detalles sobre las especificaciones de la imagen personalizada, véase **Especificaciones a las Imágenes personalizadas para trabajos de entrenamiento**.
- La imagen personalizada se ha subido a SWR. Para más detalles, véase **¿Cómo puedo iniciar sesión en SWR y cargar imágenes en él?**.

Creación de un trabajo de entrenamiento

1. Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación izquierdo, elija **Training Management** > **Training Jobs**.
2. Haga clic en **Create Training Job** y configure los parámetros.

Tabla 4-11 Creación de un trabajo de entrenamiento con una imagen personalizada

Parámetro	Descripción
Algorithm Type	Seleccione **Custom algorithm**. Este parámetro es obligatorio.
Boot Mode	Seleccione **Custom image**. Este parámetro es obligatorio.
Image	Ruta de la imagen del contenedor. Este parámetro es obligatorio. Puede establecer la ruta de acceso de la imagen del contenedor de cualquiera de las siguientes maneras: ● Para seleccionar su imagen o una imagen compartida con otros, haga clic en **Select** a la derecha y seleccione una imagen de contenedor para el entrenamiento. La imagen requerida debe cargarse en SWR de antemano. ● Para seleccionar una imagen pública, introduzca la dirección de la imagen pública en SWR. Introduzca la ruta de la imagen en el formato "Nombre de la organización/Nombre de la imagen:Nombre de la versión". No contenga el nombre de dominio (swr.<region>.xxx.com) en la ruta porque el sistema agregará automáticamente el nombre de dominio a la ruta. Por ejemplo, si la dirección de SWR de una imagen pública es **swr.<region>.xxx.com/test-image/tensorflow2_1_1:1.1.1**, ingrese **test-images/tensorflow2_1_1:1.1.1**.

Parámetro	Descripción
Code Directory	Seleccione el directorio de OBS donde se almacena el archivo de código de entrenamiento. Si la imagen personalizada no contiene código de entrenamiento, debe establecer este parámetro. Si la imagen personalizada contiene código de entrenamiento, no es necesario establecer este parámetro. ● Cargue el código al bucket de OBS de antemano. El tamaño total de los archivos del directorio no puede superar los 5 GB, el número de archivos no puede superar los 1000 y la profundidad de la carpeta no puede superar los 32. ● El archivo de código de entrenamiento se descarga automáticamente en el directorio **${MA_JOB_DIR}/demo-code** del contenedor de entrenamiento cuando se inicia el trabajo de entrenamiento. **demo-code** es el directorio de OBS de último nivel para almacenar el código. Por ejemplo, si **Code Directory** se establece en **/test/code**, el archivo de código de entrenamiento se descarga en el directorio **${MA_JOB_DIR}/code** del contenedor de entrenamiento.
User ID	ID de usuario para ejecutar el contenedor. Se recomienda el valor predeterminado 1000. Si es necesario especificar el UID, su valor debe estar dentro del rango especificado. Los rangos de UID de los diferentes grupos de recursos son los siguientes: ● Grupo de recursos público: 1000 a 65535 ● Grupo de recursos dedicado: 0 a 65535

Boot Command	Comando para arrancar una imagen. Este parámetro es obligatorio.
	Cuando se está ejecutando un trabajo de entrenamiento, el comando de arranque se ejecuta automáticamente después de descargar el directorio de código.
	● Si el script de arranque de entrenamiento es un archivo .py, por ejemplo, **train.py**, el comando de arranque es el siguiente.
	`python ${MA_JOB_DIR}/demo-code/train.py`
	● Si el script de arranque de entrenamiento es un archivo .sh, **main.sh** por ejemplo, el comando de arranque es el siguiente.
	`bash ${MA_JOB_DIR}/demo-code/main.sh`
	Puede usar punto y coma (;) y ampersands (&&) para combinar varios comandos. **demo-code** en el comando es el directorio de OBS de último nivel donde se almacena el código. Reemplácelo por el actual.

Parámetro	Descripción
Local Code Directory	Especifique el directorio local de un contenedor de entrenamiento. Cuando se inicia un entrenamiento, el sistema descarga automáticamente el directorio de códigos en este directorio. **/home/ma-user/modelarts/user-job-dir** es el directorio de código local predeterminado. Este parámetro es opcional.
Work Directory	Durante el entrenamiento, el sistema ejecuta automáticamente el comando **cd** para ejecutar el archivo de inicio en este directorio.

Tabla 4-12 Parámetros para crear un trabajo de entrenamiento

Parámetro	Subparámetro	Descripción

Input	Parameter	El código del algoritmo lee los datos de entrada de entrenamiento según el nombre del parámetro de entrada.
		Configure este parámetro como **data_url**, que es el mismo que el parámetro para analizar los datos de entrada en el código de entrenamiento. Puede establecer varios parámetros de entrada de entrenamiento. El nombre de cada parámetro de entrada de entrenamiento debe ser único.
		Por ejemplo, si utiliza **argparse** en el código de entrenamiento para analizar **data_url** en la entrada de datos, establezca el nombre del parámetro de la entrada del entrenamiento en **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()`

Parámetro	Subparámetro	Descripción
	Dataset	Haga clic en **Dataset** y seleccione el conjunto de datos de destino y su versión en la lista del conjunto de datos de ModelArts. Cuando se inicia el trabajo de entrenamiento, ModelArts descarga automáticamente los datos en la ruta de entrada al contenedor de entrenamiento. **NOTA** La gestión de datos de ModelArts se está actualizando y es invisible para los usuarios que no han utilizado la gestión de datos. Se recomienda que los nuevos usuarios almacenen sus datos de entrenamiento en buckets de OBS.
	Data path	Haga clic en **Data path** y seleccione la ruta de almacenamiento para los datos de entrada de entrenamiento de un bucket de OBS. Cuando se inicia el trabajo de entrenamiento, ModelArts descarga automáticamente los datos en la ruta de entrada al contenedor de entrenamiento.

	How to Obtain	A continuación se utiliza la entrada de entrenamiento **data_path** como ejemplo.
		● Si selecciona **Hyperparameters**, utilice este código para obtener los datos:
		`import argparse`
		`parser = argparse.ArgumentParser()`
		`parser.add_argument('--data_path')`
		`args, unknown = parser.parse_known_args()`
		`data_path = args.data_path`
		● Si selecciona **Environment variables**, utilice este código para obtener los datos:
		`import os`
		`data_path = os.getenv("data_path", "")`

Output	Parameter	El código del algoritmo lee los datos de salida de entrenamiento según el nombre del parámetro de salida. Configure este parámetro como **train_url** que es el mismo que el parámetro para analizar los datos de salida en el código de entrenamiento. Puede establecer varios parámetros de salida de entrenamiento. El nombre de cada parámetro de salida de entrenamiento debe ser único.
	Data path	Haga clic en **Data path** y seleccione la ruta de almacenamiento para los datos de salida de entrenamiento de un bucket de OBS. Durante el entrenamiento, el sistema sincroniza automáticamente los archivos del directorio de código local del contenedor de entrenamiento con la ruta de datos. **NOTA** La ruta de datos solo puede ser una ruta de OBS. Para evitar problemas con el almacenamiento de datos, elija un directorio vacío como ruta de acceso de datos.

Parámetro	Subparámetro	Descripción

	How to Obtain	A continuación se utiliza la salida de entrenamiento **train_url** como ejemplo.
		● Si selecciona **Hyperparameters**, utilice este código para obtener los datos:
		`import argparse`
		`parser = argparse.ArgumentParser()`
		`parser.add_argument('--train_url')`
		`args, unknown = parser.parse_known_args()`
		`train_url = args.train_url`
		● Si selecciona **Environment variables**, utilice este código para obtener los datos:
		`import os`
		`train_url = os.getenv("train_url", "")`

	Predownload	Indica si se deben descargar previamente los archivos del directorio de salida a un directorio local. ● Si establece **Predownload** en **No**, el sistema no descarga los archivos de la ruta de datos de salida de entrenamiento a un directorio local del contenedor de entrenamiento cuando se inicia el trabajo de entrenamiento. ● Si establece **Predownload** en **Yes**, el sistema descarga automáticamente los archivos de la ruta de datos de salida de entrenamiento a un directorio local del contenedor de entrenamiento cuando se inicia el trabajo de entrenamiento. Cuanto mayor sea el tamaño del archivo, mayor será el tiempo de descarga. Para evitar un tiempo de entrenamiento excesivo, elimine todos los archivos innecesarios del directorio de código local del contenedor de entrenamiento lo antes posible. Para utilizar **los entrenamientos reanudable e incremental**, se debe seleccionar **Download**.
Hyperparameters	-	Se utiliza para el ajuste de entrenamiento. Este parámetro es opcional.
Environment Variable	-	Agregue variables de entorno según los requisitos de servicio. Para obtener más información sobre las variables de entorno predefinidas en el contenedor de entrenamiento, véase **Consulta de variables de entorno de un contenedor de entrenamiento**.

Parámetro	Subparámetro	Descripción
Auto Restart	-	Número de reintentos para un trabajo de entrenamiento fallido. Si este parámetro está habilitado, un trabajo de entrenamiento fallido se volverá a entregar y ejecutará automáticamente. En la página de detalles del trabajo de entrenamiento, puede ver el número de reintentos de un trabajo de entrenamiento fallido. ● Esta función está deshabilitada por defecto. ● Si habilita esta función, establezca el número de reintentos. El valor oscila de **1** a **3** y no se puede cambiar.

3. Seleccione una variante de instancia. El rango de valores de los parámetros de entrenamiento es coherente con las restricciones de las imágenes personalizadas existentes. Seleccione un grupo de recursos público o dedicado según sea necesario. Para obtener más información sobre los parámetros, véase **Creación de un trabajo de entrenamiento**.
4. Haga clic en **Submit** para crear el trabajo de entrenamiento.

   Se necesita un período de tiempo para crear un trabajo de entrenamiento.

   Para ver el estado en tiempo real de un trabajo de entrenamiento, vaya a la lista de los trabajos de entrenamiento y haga clic en su nombre. En la página de detalles del trabajo de entrenamiento que se muestra, vea la información básica del trabajo de entrenamiento. Para obtener más información, véase **Consulta de detalles del trabajo de entrenamiento**.

4.6 Proceso de solución de problemas

Síntoma

Error en un trabajo de entrenamiento con una imagen personalizada.

Método de localización

1. Determine el origen de la imagen.
   - Verifique si la imagen base de la imagen personalizada es de ModelArts. Utilice una imagen base proporcionada por ModelArts para crear una imagen personalizada. Para obtener más información, véase **Uso de una imagen base para crear una imagen de entrenamiento**.
   - Si la imagen es de un tercero, consulte con el creador de la imagen personalizada para saber cómo usarla.
2. Determine el tamaño de la imagen personalizada.

   No utilice una imagen personalizada de más de 15 GB. El tamaño no debe exceder la mitad del espacio del motor del contenedor del grupo de recursos. De lo contrario, la hora de inicio del trabajo de entrenamiento se ve afectada.

   El espacio del motor de contenedor del grupo de recursos público de ModelArts es de 50 GB. Por defecto, el espacio del motor del contenedor del recurso de grupo dedicado

también es de 50 GB. Puede personalizar el espacio del motor del contenedor al crear un grupo de recursos dedicado.

3. Determine el tipo de error.
   - Si aparece un mensaje de error que indica que no se ha encontrado un archivo, véase **Mensaje de error "No hay ningún archivo o directorio" en logs de trabajo de entrenamiento**.
   - Si aparece un mensaje de error que indica que no se ha encontrado un paquete, véase **Mensaje de error "No hay módulo con nombre .*" en logs de trabajos de entrenamiento**.
   - Se produjo un error en el script de inicio de Ascend o en el script de inicialización.

     Verifique si el script se ha obtenido del sitio web oficial y si el script se utiliza estrictamente siguiendo las instrucciones proporcionadas en los documentos oficiales. Por ejemplo, verifique si el nombre y la ruta del script son correctos.
   - La versión del controlador no es compatible con el controlador subyacente.

     Antes de actualizar el controlador de una imagen personalizada, compruebe si la versión actualizada es compatible con el controlador subyacente. **Obtenga las versiones de controlador admitidas**.
   - No tiene permiso para acceder a un archivo.

     La posible causa es que el usuario de la imagen personalizada es diferente al del contenedor de trabajos. En este caso, modifique el 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 otros problemas, busque soluciones en **casos de fallas de entrenamiento**.

Resumen y sugerencias

Antes de usar una imagen personalizada para los trabajos de entrenamiento, cree la imagen siguiendo las **especificaciones de imagen personalizada**, que también proporciona ejemplos de extremo a extremo para su referencia.

5 Uso de una imagen personalizada para crear aplicaciones de IA para el despliegue de inferencia

Especificaciones de imágenes personalizadas para crear aplicaciones de IA

Creación de una imagen personalizada y su uso para crear una aplicación de IA

5.1 Especificaciones de imágenes personalizadas para crear aplicaciones de IA

Al crear una imagen personalizada utilizando un modelo desarrollado localmente, asegúrese de que la imagen cumple con las especificaciones de ModelArts.

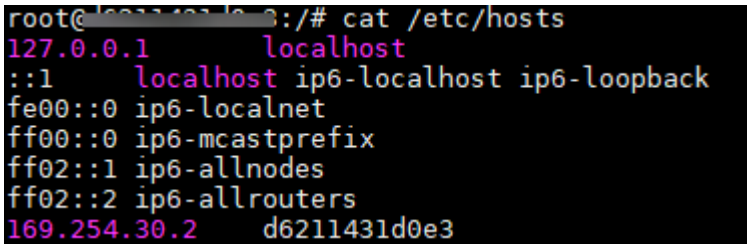
- No se permite ningún código malicioso.
- Una imagen personalizada no puede superar los 50 GB.
- **API externas**

  Establezca la API de servicio externa para una imagen personalizada. La API de inferencia debe ser la misma que el URL definida por **apis** en **config.json**. A continuación, se puede acceder directamente a la API de servicio externa cuando se inicia la imagen. A continuación se muestra un ejemplo de acceso a una imagen MNIST. La imagen contiene un modelo entrenado usando un conjunto de datos de MNIST y puede identificar los dígitos escritos a mano. **listen_ip** indica la dirección IP del contenedor. Puede iniciar una imagen personalizada para obtener la dirección IP del contenedor.

  – Ejemplo de solicitud

    ```
    curl -X POST \ http://{Listening IP address}:8080/ \ -F images=@seven.jpg
    ```


Figura 5-1 Ejemplo de obtención de **listen_ip**



– Ejemplo de respuesta

```
{"mnist_result": 7}
```

- **(Opcional) API de comprobación de estado**

  Si los servicios no se deben interrumpir durante una actualización sucesiva, la API de comprobación de estado debe configurarse en **config.json** para ModelArts. La API de la prueba de estado devuelve la situación de estado de un servicio cuando se está ejecutando correctamente, o un error cuando se vuelve defectuoso.

  AVISO

  La API de comprobación de estado debe configurarse para una actualización continua sin problemas.

  A continuación se muestra un ejemplo de API de comprobación de estado:

  – URI

  ```
  GET /health
  ```

  – Ejemplo de solicitud: curl -X GET \ http://*{Listening IP address}*:8080/health

  – Ejemplo de respuesta

  ```
  {"health": "true"}
  ```

  – Código de estado

  Tabla 5-1 Código de estado

Código de estado	Mensaje	Descripción
200	OK	Solicitud enviada

- **Salida del archivo de log**

  Configure la salida estándar para que los logs se puedan mostrar correctamente.

- **Archivo de arranque de imagen**

  Para desplegar un servicio por lotes, establezca el archivo de inicio de una imagen en **/home/run.sh** y use CMD para establecer la ruta de inicio predeterminada. El siguiente es un ejemplo de Dockerfile:

  CMD ["sh", "/home/run.sh"]

- **Dependencias de imagen**

  Para desplegar un servicio por lotes, instale los paquetes de dependencias como Python, JRE/JDK y ZIP en la imagen.

- **(Opcional) Actualización continua sin hit**

Para asegurarse de que los servicios no se interrumpen durante una actualización sucesiva, establezca HTTP **keep-alive** en **200**. Por ejemplo, Gunicorn no es compatible con keep-alive de forma predeterminada. Para garantizar una actualización continua sin problemas, instale Gevent y configure **--keep-alive 200 -k gevent** en la imagen. La configuración de los parámetros varía en función del marco de servicio. Configure los parámetros según sea necesario.

- **(Opcional) Salida con gracia de un contenedor**

  Para garantizar que los servicios no se interrumpan durante una mejora continua, el sistema debe capturar las señales SIGTERM en el contenedor y esperar 60 segundos antes de salir del contenedor con gracia. Si la duración es inferior a 60 segundos antes de la salida elegante, los servicios pueden interrumpirse durante la actualización continua. Para garantizar un funcionamiento no interrumpido del servicio, el sistema sale del contenedor después de que recibe las señales SIGTERM y procesa todas las solicitudes recibidas. La duración total no es superior a los 90 segundos. A continuación se muestra el ejemplo **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 Creación de una imagen personalizada y su uso para crear una aplicación de IA

Si desea utilizar un motor de IA que no es compatible con ModelArts, cree una imagen personalizada para el motor, importe la imagen a ModelArts y use la imagen para crear aplicaciones de IA. Esta sección describe cómo utilizar una imagen personalizada para crear una aplicación de IA e implementarla como un servicio en tiempo real.

El proceso es el siguiente:

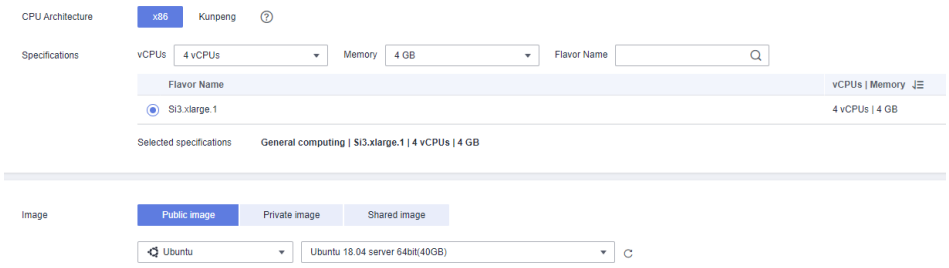
1. **Construcción de una imagen localmente**: cree un paquete de imágenes personalizadas localmente. Para obtener más información, consulte las **Especificaciones de imágenes personalizadas para crear aplicaciones de AI**.
2. **Verificación de la imagen localmente y su carga a SWR**: Verificar las API de la imagen personalizada y subir la imagen personalizada a SWR.
3. **Uso de la imagen personalizada para crear una aplicación de IA**: Importar la imagen a la gestión de aplicaciones de IA de ModelArts.
4. **Despliegue de la aplicación de IA como servicio en tiempo real**: Desplegar el modelo como un servicio en tiempo real.

Construcción de una imagen localmente

Esta sección utiliza un host de Linux x86_x64 como ejemplo. Puede comprar un ECS de las mismas especificaciones o utilizar un host local existente para crear una imagen personalizada.

Para más información sobre cómo comprar un ECS, véase **Compra e inicio de sesión en un ECS de Linux**. Al crear el ECS, seleccione una imagen pública de Ubuntu 18.04.

Figura 5-2 Creación de un ECS utilizando una imagen pública de x86



1. Después de iniciar sesión en el host, instale Docker. Para obtener más información, consulte los **documentos oficiales de Docker**. Como alternativa, ejecute los siguientes comandos para instalar Docker:
   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```
2. Obtenga la imagen de base. Se utiliza Ubuntu 18.04 en este ejemplo.
   ```
   docker pull ubuntu:18.04
   ```
3. Cree la carpeta **self-define-images** y edite **Dockerfile** y **test_app.py** en la carpeta de la imagen personalizada. En el código de ejemplo, el código de aplicación se ejecuta en el marco de Flask.

   La estructura de archivos es la siguiente:
   ```
   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__)
   ```



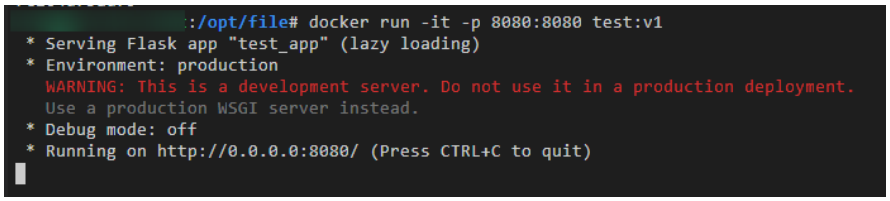
```
@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, 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. Cambie a la carpeta **self-define-images** y ejecute el siguiente comando para crear **test:v1** de imágenes personalizadas:
   ```
   docker build -t test:v1 .
   ```
5. Ejecute **docker images** para ver la imagen personalizada que ha creado.

Verificación de la imagen localmente y su carga a SWR

1. Ejecute el siguiente comando en el entorno local para iniciar la imagen personalizada:
   ```
   docker run -it -p 8080:8080 test:v1
   ```

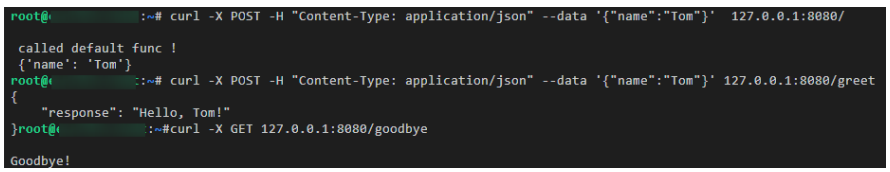
   Figura 5-3 Inicio de una imagen personalizada

2. Abra otro terminal y ejecute los siguientes comandos para probar las funciones de las tres API de la imagen 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
   ```

   Si se muestra la información similar a la siguiente, la verificación de la función se realiza correctamente.

   Figura 5-4 Prueba de las funciones de API

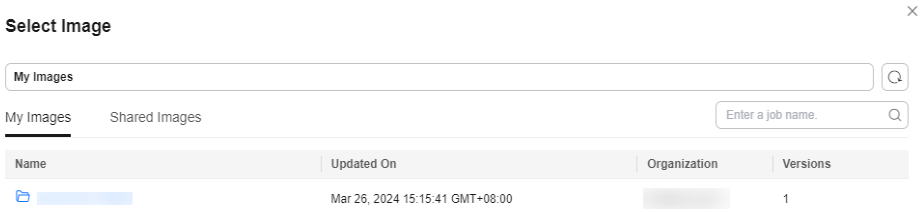
3. Cargue la imagen personalizada en SWR. Para obtener más información, véase **¿Cómo puedo subir imágenes a SWR?**
4. Vea la imagen cargada en la página **My Images** > **Private Images** de la consola de SWR.

Uso de la imagen personalizada para crear una aplicación de IA

Importe un modelo meta. Para obtener más información, véase **Creación e importación de una imagen de modelo**. Los parámetros clave son los siguientes:

- **Meta Model Source**: Seleccione **Container image**.
  - **Container Image Path**: Seleccione la imagen privada creada.

    Figura 5-5 Imagen privada creada

  - **Container API**: Protocolo y número de puerto para iniciar un modelo. Asegúrese de que el protocolo y el número de puerto son los mismos que los proporcionados en la imagen personalizada.
  - **Image Replication**: indica si se debe copiar en ModelArts la imagen de modelo de la imagen de contenedor. Este parámetro es opcional.
  - **Health Check**: comprueba el estado de salud de un modelo. Este parámetro es opcional. Este parámetro es configurable solo cuando la API de comprobación de estado está configurada en la imagen personalizada. De lo contrario, la creación de la aplicación de IA va a fallar.
- **APIs**: API de una imagen personalizada. Este parámetro es opcional. Las API de modelo deben cumplir con las especificaciones de ModelArts. Para obtener más detalles, consulte **Especificaciones para editar un archivo de configuración de modelo**.

  El archivo de configuración es el siguiente:

```
[{
        "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",
```

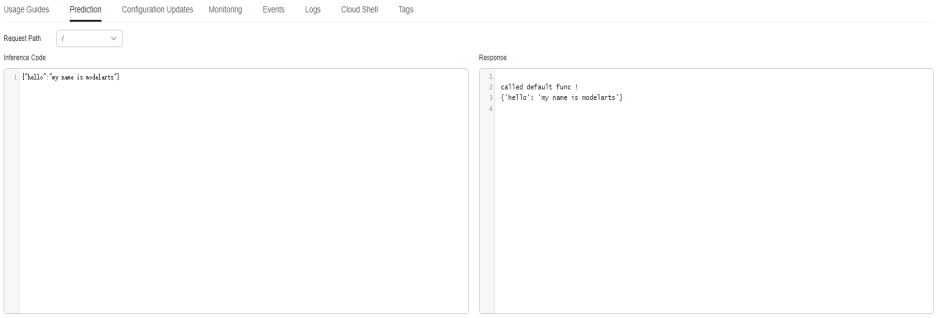


```
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    }
]
```

Despliegue de la aplicación de IA como servicio en tiempo real

1. Despliegue la aplicación de IA como un servicio en tiempo real. Para obtener más información, véase **Despliegue como servicio en tiempo real**.
2. Vea los detalles sobre el servicio en tiempo real.
3. Acceda al servicio en tiempo real en la página de ficha **Prediction**.

Figura 5-6 Acceso a un servicio en tiempo real



6 Preguntas frecuentes

¿Cómo puedo iniciar sesión en SWR y cargar imágenes en él?

¿Cómo configuro variables de entorno para una imagen?

¿Cómo uso Docker para iniciar una imagen guardada con una instancia de notebook?

¿Cómo configuro un origen de Conda en un entorno de desarrollo de notebook?

¿Cuáles son las versiones de software admitidas para una imagen personalizada?

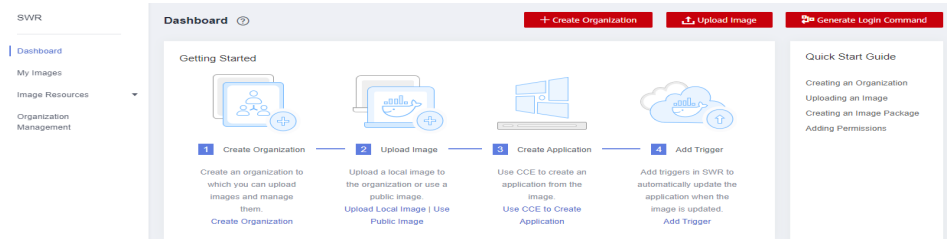
6.1 ¿Cómo puedo iniciar sesión en SWR y cargar imágenes en él?

Esta sección describe cómo iniciar sesión en SWR y cargar imágenes en él.

Paso 1 Iniciar sesión en SWR

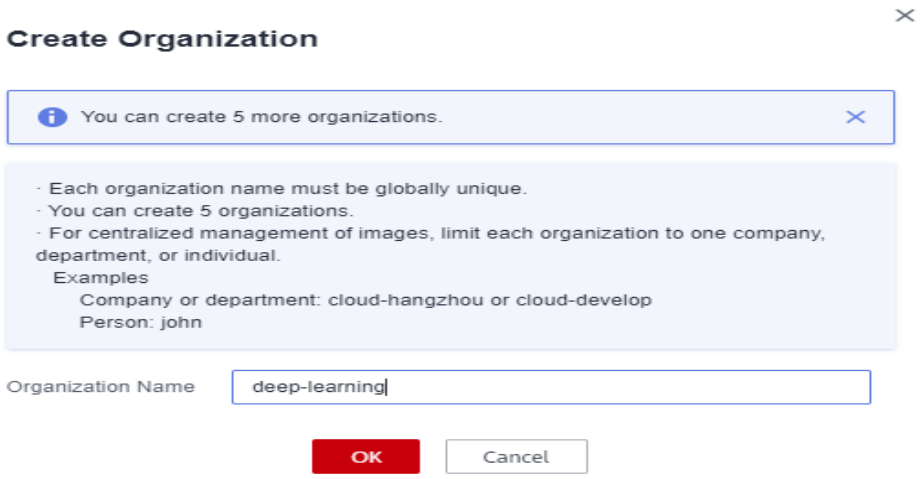
1. Inicie sesión en la consola de SWR y seleccione la región de destino.

   Figura 6-1 Consola de SWR

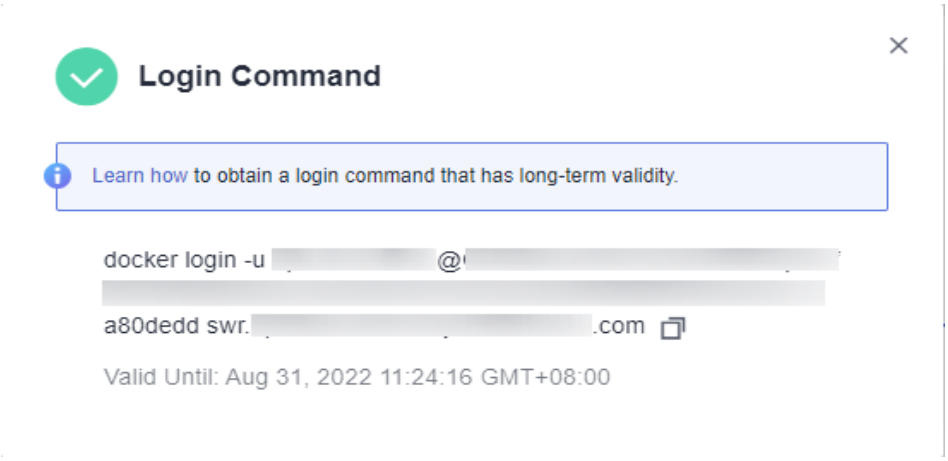
2. Haga clic en **Create Organization** en la esquina superior derecha e introduzca un nombre de organización para crear una organización. Se utiliza **deep-learning** como ejemplo. Sustitúyalo en los comandos siguientes por el nombre real de la organización.

Figura 6-2 Creación de una organización



3. Haga clic en **Generate Login Command** en la esquina superior derecha para obtener un comando de acceso.

Figura 6-3 Comando de acceso



4. Inicie sesión en el ECS como usuario **root** e ingrese el comando de acceso.

Figura 6-4 Comando de acceso ejecutado en el ECS

Paso 2 Cargar imágenes en SWR

Esta sección describe cómo subir una imagen a SWR.

1. Inicie sesión en SWR y etiquete la imagen que se va a cargar. Reemplace el nombre de organización **deep-learning** en el siguiente comando con el nombre de organización real obtenido en el paso 1.
```
sudo docker tag tf-1.13.2:latest swr.xxx.com/deep-learning/tf-1.13.2:latest
```
2. Ejecute el siguiente comando para subir la imagen:
```
sudo docker push swr.xxx.com/deep-learning/tf-1.13.2:latest
```


Figura 6-5 Carga de una imagen

3. Después de cargar la imagen, elija **My Images** en el panel de navegación a la izquierda de la consola de SWR para ver las imágenes personalizadas cargadas.

Figura 6-6 Imagen personalizada cargada

swr.xxx.com/deep-learning/tf-1.13.2:latest es la URL de SWR de la imagen personalizada.

6.2 ¿Cómo configuro variables de entorno para una imagen?

En un Dockerfile, utilice la instrucción ENV para configurar las variables de entorno. Para obtener más información, consulte la **referencia de Dockerfile**.

6.3 ¿Cómo uso Docker para iniciar una imagen guardada con una instancia de notebook?

Una imagen guardada con una instancia de notebook contiene el parámetro **Entrypoint**, como se muestra en **Entrypoint**. El archivo ejecutable o comando especificado en el parámetro **Entrypoint** sobrescribe el comando de inicio predeterminado de la imagen. La entrada del comando en el parámetro **Entrypoint** no está preestablecida en la imagen. Cuando se ejecuta **docker run** en el entorno local para iniciar la imagen, aparece un mensaje de error que indica que la tarea de creación del contenedor falla porque no se encuentra el archivo o directorio de arranque, como se muestra en **Figura 6-8**.

Para evitar este error, configure el parámetro **--entrypoint** para que sobrescriba el programa especificado en **Entrypoint**. Utilice el archivo de arranque o el comando especificado por el parámetro **--entrypoint** para iniciar la imagen. Por ejemplo:


```
docker run -it -d --entrypoint /bin/bash image:tag
```

Figura 6-7 Entrypoint

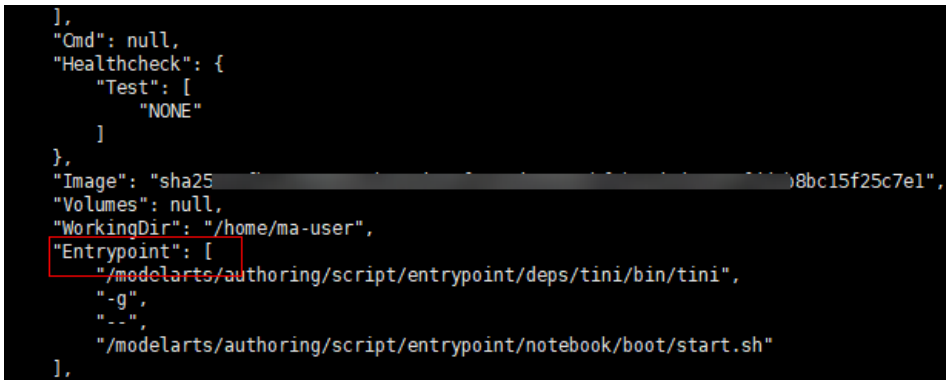
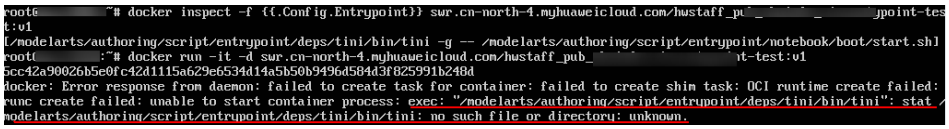


Figura 6-8 Error reportado cuando se está iniciando una imagen



6.4 ¿Cómo configuro un origen de Conda en un entorno de desarrollo de notebook?

Puede instalar las dependencias de desarrollo en Notebook según lo necesite. Las herramientas de gestión de paquetes pip y Conda se pueden usar para instalar las dependencias regulares. El origen pip se ha configurado y se puede utilizar para la instalación, mientras que el origen de Conda requiere más configuración.

Esta sección describe cómo configurar el origen de Conda en una instancia de notebook.

Configuración del origen de Conda

El software Conda ha sido preestablecido en imágenes.For details, see **https://mirror.tuna.tsinghua.edu.cn/help/anaconda/**.

Comandos comunes de Conda

Para más detalles sobre todos los comandos de Conda, consulte los **documentos oficiales de Conda**. En la siguiente tabla solo se enumeran los comandos comunes.

Tabla 6-1 Comandos comunes de Conda

Descripción	Comando
Obtener la ayuda en línea.	`conda --help` `conda update --help # Obtain help for a command, for example, update.`

Descripción	Comando
Consultar la versión de Conda.	`conda -V`
Actualizar Conda.	`conda update conda  # Update Conda.` `conda update anaconda # Update Anaconda.`

Gestionar los entornos.	`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.`

Gestionar los paquetes.	`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.`

Eliminar Conda.	`conda clean -p      # Delete useless packages.`
	`conda clean -t      # Delete compressed packages.`
	`conda clean -y --all # Delete all installation packages and clear caches.`

Guardar como una imagen

Después de instalar las librerías externas, guarde el entorno utilizando la función de guardado de imágenes proporcionada por notebook de ModelArts de la nueva versión. Puede guardar una instancia de notebook en ejecución como una imagen personalizada con un clic singular para su uso en el futuro. Después de instalar los paquetes de dependencias en una instancia de notebook, es una buena práctica guardar la instancia como una imagen para evitar que se pierdan los paquetes de dependencias. Para más detalles, véase **Guardar una imagen de entorno de notebook**.

6.5 ¿Cuáles son las versiones de software admitidas para una imagen personalizada?

Si la imagen personalizada utiliza bibliotecas de software como NCCL, CUDA y OFED, asegúrese de que las bibliotecas de software cumplan los siguientes requisitos de versión:

- NCCL 2.7.8 o posterior
- OFED MLNX_OFED_LINUX-5.4-3.1.0.0 o posterior
- La versión de CUDA debe adaptarse a la versión del controlador de GPU del grupo de recursos dedicado. Para obtener la versión del controlador de GPU, vaya a la página de detalles del grupo de recursos dedicado.

7 Cambios de modificaciones

Fecha de lanzamiento	Descripción
2023-10-01	Agregado el siguiente contenido: ● Agregado **Inicio del entrenamiento con una imagen preestablecida**.
2023-09-07	Agregado el siguiente contenido: ● Agregado **¿Cuáles son las versiones de software admitidas para una imagen personalizada?**. Modificado el siguiente contenido: ● Cambiado el nombre del manual de "Uso de imágenes personalizadas" a "Gestión de imágenes". ● Ajustado el contenido en "Imágenes preestablecidas". Combinado el entorno de desarrollo y las imágenes de base de entrenamiento e inferencia en **Uso de una imagen preestablecida**.

