ModelArts

Despliegue de inferencia

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

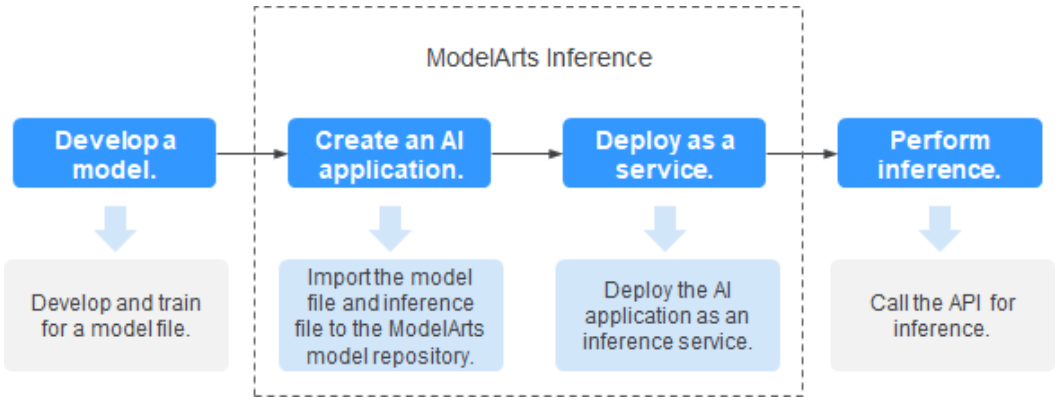
Sitio web: https://www.huaweicloud.com/intl/es-us/

Índice

1 Introducción a la Inferencia

Después de desarrollar un modelo de IA, puede usarlo para crear una aplicación de IA y desplegar rápidamente la aplicación como un servicio de inferencia. Las capacidades de inferencia de IA se pueden integrar en su plataforma de TI llamando a las API.

Figura 1-1 Inferencia



- Desarrollar un modelo: Los modelos se pueden desarrollar en el ModelArts o en su entorno de desarrollo local. Se debe cargar un modelo desarrollado localmente en Huawei Cloud OBS.
- Crear una aplicación de IA: Importe el archivo de modelo y el archivo de inferencia al repositorio de modelos ModelArts y adminístrelos por versión. Utilice estos archivos para crear una aplicación de IA ejecutable.
- Desplegar un servicio: Despliegue la aplicación de IA como instancia de contenedor en el grupo de recursos y registre las API de inferencia a las que se puede acceder externamente.
- Realizar inferencia: Agregue la función de invocar a las API de inferencia a su aplicación para integrar la inferencia de IA en el proceso de servicio.

Despliegue de una aplicación de IA como servicio

Después de crear una aplicación de IA, puede desplegarla como un servicio en la página **Deploy**. ModelArts admite los siguientes tipos de despliegue:

- **Servicio en tiempo real**

Despliegue una aplicación de IA como servicio web con interfaz de usuario de prueba en tiempo real y monitorización compatible.

- **Servicio por lotes**

  Despliegue una aplicación de IA como un servicio por lotes que realiza inferencias en datos por lotes y se detiene automáticamente una vez que se completa el procesamiento de datos.

2 Gestión de aplicaciones de IA

2.1 Introducción a la gestión de aplicaciones de IA

El desarrollo y la optimización de la IA requieren iteraciones y depuración frecuentes. Las modificaciones en los conjuntos de datos, el código de entrenamiento o los parámetros afectan a la calidad de los modelos. Si los metadatos del proceso de desarrollo no se pueden gestionar de forma centralizada, es posible que el modelo óptimo no se reproduzca.

La gestión de aplicaciones de IA de ModelArts le permite importar todos los metamodelos obtenidos con el entrenamiento, los metamodelos cargados a OBS y los metamodelos en imágenes de contenedor. De esta manera, puede gestionar de forma centralizada todas las aplicaciones de IA iteradas y depuradas.

Restricciones

- En un proyecto ExeML, después de implementar un modelo, el modelo se carga automáticamente a la lista de gestión de aplicaciones de IA. Sin embargo, las aplicaciones de IA generadas por ExeML no se pueden descargar y solo se pueden usar para despliegue y lanzamiento.
- Funciones como la creación de aplicaciones de IA, la gestión de versiones de aplicaciones de IA y la conversión de modelos están disponibles de forma gratuita para todos los usuarios.

Escenarios para crear aplicaciones de IA

- **Importado de un trabajo de entrenamiento**: crear un trabajo de entrenamiento en ModelArts y entrenar un modelo. Después de obtener un modelo satisfactorio, úselo para crear una aplicación de IA y desplegar la aplicación como servicios.
- **Importado desde OBS**: Si utiliza un marco general para desarrollar y entrenar un modelo localmente, puede cargar el modelo en un bucket de OBS según las especificaciones del paquete de modelo, importar el modelo desde OBS a ModelArts, y usar el modelo para crear una aplicación de IA para el despliegue de servicios.
- **Importado desde una imagen de contenedor**: Si ModelArts no admite un motor de IA, puede usarlo para crear un modelo, importar el modelo a ModelArts como una imagen personalizada, usar la imagen para crear una aplicación de IA y desplegar la aplicación de IA como servicios.

- **Importado desde una plantilla**: Debido a que las configuraciones de los modelos con las mismas funciones son similares, ModelArts integra las configuraciones de dichos modelos en una plantilla general. Con esta plantilla, puede importar modelos y crear aplicaciones de IA de forma fácil y rápida sin escribir el archivo de configuración **config.json**.

  NOTA

  La importación de un modelo desde una plantilla dejará de estar disponible en breve. Después de desconectarse, puede usar las plantillas para el motor de IA y las configuraciones de modelo eligiendo **OBS**, configurando **AI Engine** en **Custom** e importando su motor de IA personalizado.

Funciones de la gestión de aplicaciones de IA

Tabla 2-1 Funciones de la gestión de aplicaciones de IA

Función admitida	Descripción
Creación de una aplicación de IA	Importar los modelos capacitados a ModelArts y crear las aplicaciones de IA para una gestión centralizada. A continuación proporcionarse la guía de operación para cada método de importación de modelos. ● **Importación de un metamodelo desde un trabajo de entrenamiento** ● **Importación de un metamodelo desde OBS** ● **Importación de un metamodelo desde una imagen de contenedor** ● **Importación de un metamodelo desde una plantilla**
Consulta de detalles sobre una aplicación de IA	Después de crear una aplicación de IA, puede ver su información en la página de detalles.
Gestión de versiones de aplicaciones de IA	Para facilitar el seguimiento y el ajuste del modelo, el ModelArts proporciona la función de gestión de versiones de la aplicación de IA. Puede gestionar aplicaciones de IA por versión.

Motores de IA compatibles para la inferencia de ModelArts

Si importa un modelo desde una plantilla u OBS para crear una aplicación de IA, se admiten los siguientes motores y versiones de IA.

NOTA

- Los entornos de tiempo de ejecución marcados con **recommended** son imágenes de tiempo de ejecución unificadas, que se utilizarán como imágenes de inferencia base convencionales. Los paquetes de instalación de imágenes unificadas son más ricos. Para obtener más información, consulte **Imágenes de inferencia de base**.
- Las imágenes de la versión anterior serán descontinuadas. Utilice imágenes unificadas.
- Las imágenes base que se van a eliminar ya no se mantienen.
- Nombre una imagen unificada en tiempo de ejecución: *<AI engine name and version>* - *<Hardware and version*: CPU, CUDA, or CANN> - *<Python version>* - *<OS version>* - *<CPU architecture>*

Tabla 2-2 Motores de IA compatibles y su tiempo de ejecución

Motor	Tiempo de ejecución	Nota
TensorFlow	python3.6 python2.7 (próximamente no disponible) tf1.13-python3.6-gpu tf1.13-python3.6-cpu tf1.13-python3.7-cpu tf1.13-python3.7-gpu tf2.1-python3.7 (próximamente no disponible) tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 (recomendado)	● TensorFlow 1.8.0 se utiliza en **python2.7** y **python3.6**. ● **python3.6**, **python2.7** y **tf2.1-python3.7** indican que el modelo puede ejecutarse tanto en CPU como en GPU. Para otros valores de tiempo de ejecución, si el sufijo contiene **cpu** o **gpu**, el modelo solo puede ejecutarse en CPU o GPU. ● El tiempo de ejecución predeterminado es **python2.7**.
Spark_MLlib	python2.7 (próximamente no disponible) python3.6 (próximamente no disponible)	● Spark_MLlib 2.3.2 se utiliza en **python2.7** y **python3.6**. ● El tiempo de ejecución predeterminado es **python2.7**. ● **python2.7** y **python3.6** solo se pueden usar para ejecutar modelos en CPU.
Scikit_Learn	python2.7 (próximamente no disponible) python3.6 (próximamente no disponible)	● Scikit_Learn 0.18.1 se utiliza en **python2.7** y **python3.6**. ● El tiempo de ejecución predeterminado es **python2.7**. ● **python2.7** y **python3.6** solo se pueden usar para ejecutar modelos en CPU.
XGBoost	python2.7 (próximamente no disponible) python3.6 (unavailable soon)	● XGBoost 0.80 se utiliza en **python2.7** y **python3.6**. ● El tiempo de ejecución predeterminado es **python2.7**. ● **python2.7** y **python3.6** solo se pueden usar para ejecutar modelos en CPU.

Motor	Tiempo de ejecución	Nota
PyTorch	python2.7 (próximamente no disponible) python3.6 python3.7 pytorch1.4-python3.7 pytorch1.5-python3.7 (unavailable soon) pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 (recomendado)	● PyTorch 1.0 se utiliza en **python2.7**, **python3.6** y **python3.7**. ● **python2.7**, **python3.6**, **python3.7**, **pytorch1.4-python3.7** y **pytorch1.5-python3.7** indican que el modelo puede ejecutarse tanto en CPU como en GPU. ● El tiempo de ejecución predeterminado es **python2.7**.
MindSpore	aarch64 (recomendado)	AArch64 solo puede ejecutarse en chips Snt3.

2.2 Creación de una aplicación de IA

2.2.1 Importación de un metamodelo desde un trabajo de entrenamiento

Puede crear un trabajo de entrenamiento en ModelArts para obtener un modelo satisfactorio. Luego, puede importar el modelo a **AI Application Management** para la administración centralizada. Además, puede desplegar rápidamente el modelo como un servicio.

Restricciones

- Un modelo generado a partir de un trabajo de entrenamiento que utiliza un algoritmo suscrito se puede importar directamente a ModelArts sin necesidad de utilizar el código de inferencia o el archivo de configuración.
- Si el metamodelo es de una imagen de contenedor, asegúrese de que el tamaño del metamodelo cumple con **Restricciones sobre el tamaño de una imagen para importar una aplicación de IA**.

Requisitos previos

- El trabajo de entrenamiento se ha ejecutado correctamente y el modelo se ha almacenado en el directorio de OBS donde se almacena la salida de entrenamiento (el parámetro de entrada es **train_url**).
- Si se genera un modelo a partir de un trabajo de entrenamiento que utiliza un marco de trabajo o una imagen personalizada de uso frecuente, cargue el código de inferencia y el archivo de configuración en el directorio de almacenamiento del modelo haciendo referencia a **Introducción a las especificaciones del paquete modelo**.
- El directorio de OBS seleccionado a usar y ModelArts están en la misma región.

Creación de una aplicación de IA

1. Inicie sesión en la consola de gestión de ModelArts y seleccione **AI Application Management** > **AI Applications** en el panel de navegación izquierdo. Se muestra la página **AI Applications**.
2. Haga clic en **Create** en la esquina superior izquierda.
3. En la página mostrada, establezca los parámetros.
   a. Establezca información básica sobre la aplicación de IA. Para obtener detalles sobre los parámetros, véase **Tabla 2-3**.

Tabla 2-3 Parámetros de la información básica de la aplicación de IA

Parámetro	Descripción
Name	Nombre de la aplicación. El valor puede contener de 1 a 64 caracteres visibles. Solo se permiten letras, dígitos, guiones medios (-) y guiones bajos (_).
Version	Versión de la aplicación de IA a crear. Para la primera importación, el valor predeterminado es **0.0.1**. **NOTA** Después de crear una aplicación de IA, puede **crear las nuevas versiones** con diferentes metamodelos para la optimización.
Description	Breve descripción de una aplicación de IA

   b. Seleccione el origen del metamodelo y establezca los parámetros relacionados. Establezca **Meta Model Source** en **Training job**. Para obtener detalles sobre los parámetros, véase **Tabla 2-4**.

Figura 2-1 Establecer un trabajo de entrenamiento como fuente del metamodelo

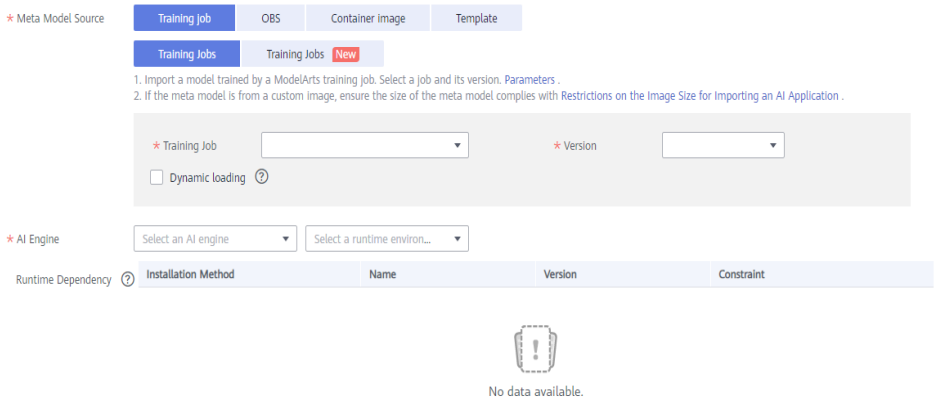


Tabla 2-4 Parámetros de la fuente del metamodelo

Parámetro	Descripción
Meta Model Source	Seleccione **Training Job** > **Training Jobs** o **Training Job** > **Training Jobs (New)**. ● Seleccione un trabajo de entrenamiento que haya completado el entrenamiento con la cuenta corriente y una versión de entrenamiento en las listas desplegables a la derecha de **Training Job** y **Version**, respectivamente. ● **Dynamic loading**: habilitada para un despliegue y una actualización de modelo rápidos. Si se selecciona esta función, los archivos de modelo y las dependencias en tiempo de ejecución solo se extraen durante un despliegue real. Habilite esta función si un archivo de modelo singular tiene más de 5 GB. **NOTA** ModelArts ofrece entrenamiento de modelos de las versiones nuevas y antiguas. La gestión de entrenamiento de la versión anterior solo está disponible para sus usuarios existentes.
AI Engine	Motor de inferencia utilizado por el metamodelo. El motor se empareja automáticamente en función del trabajo de entrenamiento que seleccione.
Inference Code	Configure el código de inferencia para una aplicación de IA. El código se utiliza para personalizar la lógica de procesamiento de inferencia. Muestra la URL del código de inferencia. Puede copiar esta URL directamente.
Runtime Dependency	Enumere las dependencias del modelo seleccionado en el entorno. Por ejemplo, si se utiliza **tensorflow** y el método de instalación es **pip**, la versión debe ser 1.8.0 o posterior.
AI Application Description	Proporcione las descripciones de aplicaciones de IA para ayudar a otros desarrolladores de aplicaciones de IA a comprender y usar mejor sus aplicaciones. Haga clic en **Add AI Application Descripción** y configure el **Document name** y la URL. Se admite un máximo de tres descripciones de aplicaciones de IA.
Deployment Type	Seleccione los tipos de servicio que se pueden desplegar en la aplicación. Al desplegar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services**, solo puede deplegar la aplicación de IA como un servicio en tiempo real después de crearla.

c. Confirme las configuraciones y haga clic en **Create now**. Se crea la aplicación de IA.

En la lista de aplicaciones de IA, puede ver la aplicación de IA creada y su versión. Cuando el estado cambia a **Normal**, la aplicación de IA se crea correctamente. En esta página, puede realizar operaciones como la creación de nuevas versiones y el despliegue rápido de servicios.

Procedimiento posterior

Despliegue de una aplicación de IA como servicio: En la lista de aplicaciones de IA, haga clic en el botón de opción a la izquierda del nombre de la aplicación de IA para mostrar la lista de versiones en la parte inferior de la página de lista. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** para desplegar la aplicación de IA como un tipo de despliegue seleccionado durante la creación de la aplicación de IA.

2.2.2 Importación de un metamodelo desde una plantilla

NOTA

La importación de un modelo desde una plantilla dejará de estar disponible en breve. Después de desconectarse, puede usar las plantillas para el motor de IA y las configuraciones de modelo eligiendo **OBS**, configurando **AI Engine** en **Custom** e importando su motor de IA personalizado.

Debido a que las configuraciones de plantillas con las mismas funciones son similares, el ModelArts integra las configuraciones de tales plantillas en una plantilla común. Mediante el uso de esta plantilla, puede crear fácilmente y rápidamente aplicaciones de IA sin compilar el archivo de configuración **config.json**. Para obtener más información sobre la plantilla, véase **Introducción a las plantillas de modelo**.

Restricciones

- Para obtener detalles sobre las plantillas admitidas, véase **Plantillas admitidas**. Para obtener detalles sobre los modos de entrada y salida de cada plantilla, vea **Modos de entrada y salida admitidos**.
- La creación y gestión de aplicaciones de IA es gratuita.

Requisitos previos

- Asegúrese de haber cargado el modelo en OBS de acuerdo con las especificaciones del paquete de modelo de la plantilla correspondiente.
- El directorio de OBS seleccionado a usar y ModelArts están en la misma región.

Creación de una aplicación de IA

1. Inicie sesión en la consola de gestión ModelArts y elija **AI Application Management** > **AI Applications** en el panel de navegación izquierdo. Se muestra la página **AI Applications**.
2. Haga clic en **Create** en la esquina superior izquierda.
3. En la página mostrada, establezca los parámetros.
   a. Establezca información básica sobre la aplicación de IA. Para obtener detalles sobre los parámetros, véase **Tabla 2-5**.

Tabla 2-5 Parámetros de la información básica de la aplicación de IA

Parámetro	Descripción
Name	Nombre de la aplicación. El valor puede contener de 1 a 64 caracteres visibles. Solo se permiten letras, dígitos, guiones medios (-) y guiones bajos (_).

Parámetro	Descripción
Version	Versión de la aplicación de IA a crear. Para la primera importación, el valor predeterminado es **0.0.1**. **NOTA** Después de crear una aplicación de IA, puede **crear las nuevas versiones** con diferentes metamodelos para la optimización.
Description	Breve descripción de una aplicación de IA

b. Seleccione el origen del metamodelo y establezca los parámetros relacionados. Establezca **Meta Model Source** en **Template**. Para obtener detalles sobre los parámetros, véase **Tabla 2-6**.

Figura 2-2 Establecer una plantilla como origen del metamodelo

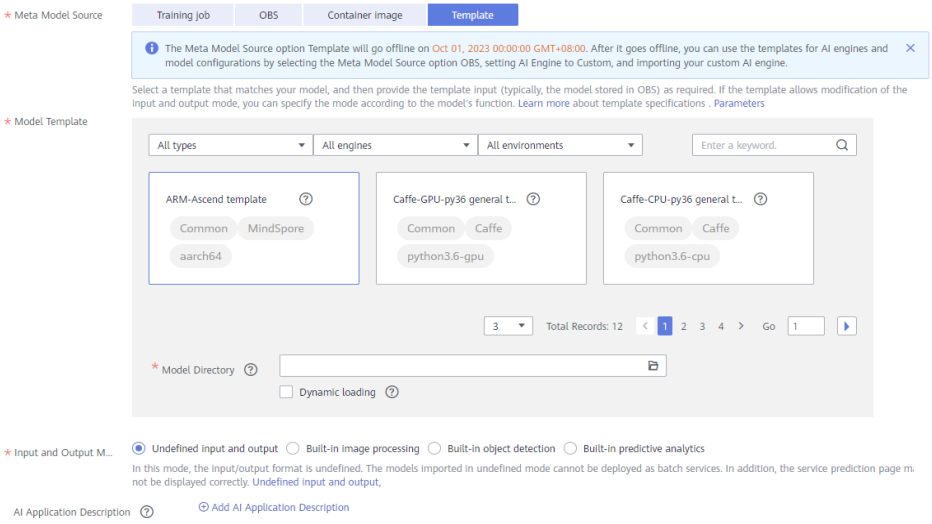


Tabla 2-6 Parámetros de la fuente del metamodelo

Parámetro	Descripción
Model Template	Seleccione una plantilla de la lista de plantillas de ModelArts existente, como **TensorFlow-based image classification template**. ModelArts también proporciona tres criterios de filtro: **Type**, **Engine**, y **Environment**, lo que le ayuda a encontrar rápidamente la plantilla deseada. Si los tres criterios de filtro no pueden cumplir sus requisitos, puede introducir palabras clave para buscar la plantilla de destino. Para obtener detalles sobre las plantillas admitidas, véase **Plantillas admitidas**.

Parámetro	Descripción
Model Directory	Ruta de OBS donde se guarda un modelo. Seleccione una ruta de OBS para almacenar el modelo según los requisitos de entrada de la plantilla de modelo seleccionada. La ruta de OBS no puede contener espacios. De lo contrario, no se puede crear la aplicación de IA. **NOTA** ● Si selecciona un bucket o archivo cifrado, la importación fallará. ● Si se ejecuta un trabajo de entrenamiento varias veces, se generan directorios de versiones diferentes, como V001 y V002, y los modelos generados se almacenan en la carpeta **model** en directorios de versiones diferentes. Al seleccionar los archivos de modelo, especifique la carpeta **model** en el directorio de versión correspondiente.
Dynamic Loading	Rápidamente despliegue y actualización de modelo. Si se selecciona esta función, los archivos de modelo y las dependencias en tiempo de ejecución solo se extraen durante un despliegue real. Seleccione esta función si el tamaño de un solo archivo modelo supera los 5 GB.
Input and Output Mode	Si el modo de entrada y salida predeterminado de la plantilla seleccionada se puede sobrescribir, puede seleccionar un modo de entrada y salida basado en la función de aplicación de IA o en el escenario de aplicación. **Input and Output Mode** es un resumen de la API (**apis**) en **config.json**. Describe la interfaz proporcionada por la aplicación de IA para inferencia externa. Un modo de entrada y salida describe una o más API, y corresponde a una plantilla. Por ejemplo, para **TensorFlow-based image classification template**, **Input and Output Mode** admite **Built-in image processing mode**. Los modos de entrada y salida no se pueden modificar en la plantilla. Por lo tanto, solo puede ver, pero no modificar, el modo de entrada y salida predeterminado de la plantilla en la página. Para obtener detalles sobre los modos de entrada y salida admitidos, véase **Modos de entrada y salida admitidos**.
AI Application Description	Proporcione las descripciones de aplicaciones de IA para ayudar a otros desarrolladores de aplicaciones de IA a comprender y usar mejor sus aplicaciones. Haga clic en **Add AI Application Description** y establezca **Document name** y **URL**. Puede agregar hasta tres descripciones de aplicaciones de IA.
Deployment Type	Seleccione los tipos de servicio que se pueden desplegar en la aplicación. Al desplegar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services**, solo puede deplegar la aplicación de IA como un servicio en tiempo real después de crearla.

c. Compruebe la información y haga clic en **Next**. Se crea la aplicación de IA.

En la lista de aplicaciones de IA, puede ver la aplicación de IA creada y su versión. Cuando el estado cambia a **Normal**, la aplicación de IA se crea correctamente. En esta página, puede realizar operaciones como la creación de nuevas versiones y el despliegue rápido de servicios.

Procedimiento posterior

Despliegue de una aplicación de IA como servicio: En la lista de aplicaciones de IA, haga clic en el botón de opción a la izquierda del nombre de la aplicación de IA para mostrar la lista de versiones en la parte inferior de la página de lista. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** para desplegar la aplicación de IA como un tipo de despliegue seleccionado durante la creación de la aplicación de IA.

2.2.3 Importación de un metamodelo desde OBS

En escenarios donde se utilizan marcos de trabajo de uso frecuente para el desarrollo y el entrenamiento de modelos, puede importar el modelo a ModelArts y usarlo para crear una aplicación de IA para la gestión unificada.

Restricciones

- El modelo importado para crear una aplicación de IA, código de inferencia y archivo de configuración debe cumplir con los requisitos de ModelArts. Para obtener más información, véase **Introducción a las especificaciones del paquete modelo**, **Especificaciones para editar un archivo de configuración de modelo** y **Especificaciones para escribir el código de inferencia de modelo**.
- Si el metamodelo es de una imagen de contenedor, asegúrese de que el tamaño del metamodelo cumple con **Restricciones sobre el tamaño de una imagen para importar una aplicación de IA**.

Requisitos previos

- El modelo ha sido desarrollado y entrenado, y ModelArts admite el tipo y la versión del motor de IA utilizado. Para más detalles, véase **Motores de IA compatibles para la inferencia de ModelArts**.
- El paquete de modelo entrenado, el código de inferencia y el archivo de configuración se han subido a OBS.
- El directorio de OBS seleccionado a usar y ModelArts están en la misma región.

Creación de una aplicación de IA

1. Inicie sesión en la consola de gestión ModelArts y elija **AI Application Management** > **AI Applications** en el panel de navegación izquierdo. Se muestra la página **AI Applications**.
2. Haga clic en **Create** en la esquina superior izquierda.
3. En la página mostrada, establezca los parámetros.
   a. Establezca información básica sobre la aplicación de IA. Para obtener detalles sobre los parámetros, véase **Tabla 2-7**.

Tabla 2-7 Parámetros de la información básica de la aplicación de IA

Parámetro	Descripción
Name	Nombre de la aplicación. El valor puede contener de 1 a 64 caracteres visibles. Solo se permiten letras, dígitos, guiones medios (-) y guiones bajos (_).
Version	Versión de la aplicación de IA a crear. Para la primera importación, el valor predeterminado es **0.0.1**. **NOTA** Después de crear una aplicación de IA, puede **crear las nuevas versiones** con diferentes metamodelos para la optimización.
Description	Breve descripción de una aplicación de IA

b. Seleccione el origen del metamodelo y establezca los parámetros relacionados. Establezca **Meta Model Source** en **OBS**. Para obtener detalles sobre los parámetros, véase **Tabla 2-8**.

Para el metamodelo importado de OBS, edite el código de inferencia y los archivos de configuración siguiendo **las especificaciones del paquete de model** y coloque el código de inferencia y los archivos de configuración en la carpeta **model** que almacena el metamodelo. Si el directorio seleccionado no cumple con las especificaciones del paquete modelo, no se puede crear la aplicación de IA.

Tabla 2-8 Parámetros de la fuente del metamodelo

Parámetro	Descripción
Meta Model	Ruta de OBS para almacenar el metamodelo. La ruta de OBS no puede contener espacios. De lo contrario, no se puede crear la aplicación de IA.
AI Engine	El motor de IA se asocia automáticamente con la ruta de almacenamiento del metamodelo que seleccione. Si **AI Engine** se establece en **Custom**, debe especificar el protocolo y el número de puerto de **Container API** para iniciar el modelo. El protocolo de solicitud debe ser **HTTPS** y el número de puerto debe ser **8080**.

Parámetro	Descripción
Health Check	Comprobación de estado de un modelo. Después de seleccionar un motor de IA que admita la comprobación de estado y el entorno de tiempo de ejecución, se muestra este parámetro. Cuando **AI Engine** se establece en **Custom** debe configurar la comprobación de estado en la imagen. De lo contrario, el despliegue del servicio fallará. ● **Check Mode**: seleccione **HTTP request** o **Command**. Cuando se utiliza un motor personalizado, puede seleccionar **HTTP request** o **Command**. Cuando se utiliza un motor no personalizado, solo se puede seleccionar **HTTP request**. ● **Health Check URL**: Este parámetro se muestra cuando **Check Mode** se establece en **HTTP request**. Ingrese la URL de comprobación de estado. El valor predeterminado es **/health**. ● **Health Check Command**: Este parámetro se muestra cuando **Check Mode** se establece en **Command**. Ingrese el comando de comprobación de estado. ● **Health Check Period**: Ingrese un número entero entre 1 y 2147483647. La unidad es segundo. ● **Delay(seconds)**: especifica el retraso para realizar la comprobación de estado después de iniciar la instancia. Ingrese un número entero entre 0 y 2147483647. ● **Maximum Failures**: Ingrese un número entero entre 1 y 2147483647. Durante el inicio del servicio, si el número de fallos de comprobación de estado consecutivos alcanza el valor especificado, el servicio será anormal. Durante la ejecución del servicio, si el número de fallos de comprobación de estado consecutivos alcanza el valor especificado, el servicio ingresará el estado de alarma. **NOTA** Para usar un motor personalizado para crear una aplicación de IA, asegúrese de que el motor personalizado cumpla con las especificaciones para los motores personalizados. Para obtener más información, véase **Creación de una aplicación de IA con un motor personalizado**. Si la comprobación de estado está configurada para una aplicación de IA, los servicios desplegados que utilizan esta aplicación de IA se detendrán 3 minutos después de recibir la instrucción de parada.
Dynamic Loading	Rápidamente despliegue y actualización de modelo. Si se selecciona, los archivos de modelo y las dependencias en tiempo de ejecución solo se extraen durante un despliegue real. Habilite esta función si un archivo de modelo singular tiene más de 5 GB.
Runtime Dependency	Enumere las dependencias del modelo seleccionado en el entorno. Por ejemplo, si se utiliza **tensorflow** y el método de instalación es **pip**, la versión debe ser 1.8.0 o posterior.

Parámetro	Descripción
AI Application Description	Proporcione las descripciones de aplicaciones de IA para ayudar a otros desarrolladores de aplicaciones de IA a comprender y usar mejor sus aplicaciones. Haga clic en **Add AI Application Description** y establezca **Document name** y **URL**. Puede agregar hasta tres descripciones de aplicaciones de IA.
Configuration File	De forma predeterminada, el sistema asocia el archivo de configuración almacenado en OBS. Después de habilitar esta función, puede ver y editar el archivo de configuración del modelo. **NOTA** Esta función debe ser puesta fuera de línea. Luego, puede modificar la configuración del modelo configurando **AI Engine**, **Runtime Dependency** y **Apis**.
Deployment Type	Seleccione los tipos de servicio que se pueden desplegar en la aplicación. Al desplegar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services** aquí, solo podrá desplegar la aplicación de IA como servicio en tiempo real después de crearla.
API Configuration	Después de habilitar esta función, puede editar las API de RESTful para definir los formatos de entrada y salida de una aplicación de IA. Las API del modelo deben cumplir con las especificaciones de ModelArts. Para obtener más información, véase **Especificaciones para editar un archivo de configuración de modelo**. Para obtener detalles sobre el ejemplo de código, vea **Ejemplo de código de parámetros de apis**.

c. Verifique la información y haga clic en **Create now**. Se crea la aplicación de IA.

En la lista de aplicaciones de IA, puede ver la aplicación de IA creada y su versión. Cuando el estado cambia a **Normal**, la aplicación de IA se crea correctamente. En esta página, puede realizar operaciones como la creación de nuevas versiones y el despliegue rápido de servicios.

Procedimiento posterior

Despliegue de una aplicación de IA como servicio: En la lista de aplicaciones de IA, haga clic en el botón de opción a la izquierda del nombre de la aplicación de IA para mostrar la lista de versiones en la parte inferior de la página de lista. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** para desplegar la aplicación de IA como un tipo de despliegue seleccionado durante la creación de la aplicación de IA.

2.2.4 Importación de un metamodelo desde una imagen de contenedor

Para los motores de IA que no son compatibles con ModelArts puede importar los modelos que compila a ModelArts desde imágenes personalizadas.

Restricciones

- Para obtener más información sobre las especificaciones y la descripción de las imágenes personalizadas, véase **Especificaciones de imagen personalizada para crear aplicaciones con IA**.
- El archivo de configuración debe proporcionarse para un modelo que haya desarrollado y entrenado. El archivo debe cumplir con las especificaciones de ModelArts. Para más detalles, véase **Especificaciones para editar un archivo de configuración de modelo**. Una vez completada la escritura, cargue el archivo en el directorio de OBS especificado.
- Si el metamodelo es de una imagen de contenedor, asegúrese de que el tamaño del metamodelo cumpla con **Restricciones sobre el tamaño de una imagen para importar una aplicación de IA**.

Requisitos previos

El directorio de OBS seleccionado a usar y ModelArts están en la misma región.

Creación de una aplicación de IA

1. Inicie sesión en la consola de gestión ModelArts y elija **AI Application Management** > **AI Applications** en el panel de navegación izquierdo. Se muestra la página **AI Applications**.
2. Haga clic en **Create** en la esquina superior izquierda.
3. En la página mostrada, establezca los parámetros.
   a. Establezca información básica sobre la aplicación de IA. Para obtener detalles sobre los parámetros, véase **Tabla 2-9**.

Tabla 2-9 Parámetros de la información básica de la aplicación de IA

Parámetro	Descripción
Name	Nombre de la aplicación. El valor puede contener de 1 a 64 caracteres visibles. Solo se permiten letras, dígitos, guiones medios (-) y guiones bajos (_).
Version	Versión de la aplicación de IA a crear. Para la primera importación, el valor predeterminado es **0.0.1**. **NOTA** Después de crear una aplicación de IA, puede **crear las nuevas versiones** con diferentes metamodelos para la optimización.
Description	Breve descripción de una aplicación de IA

   b. Seleccione el origen del metamodelo y establezca los parámetros relacionados. Establezca **Meta Model Source** en **Container image**. Para obtener detalles sobre los parámetros, véase **Tabla 2-10**.

Figura 2-3 Establecer una imagen contenedora como fuente del metamodelo

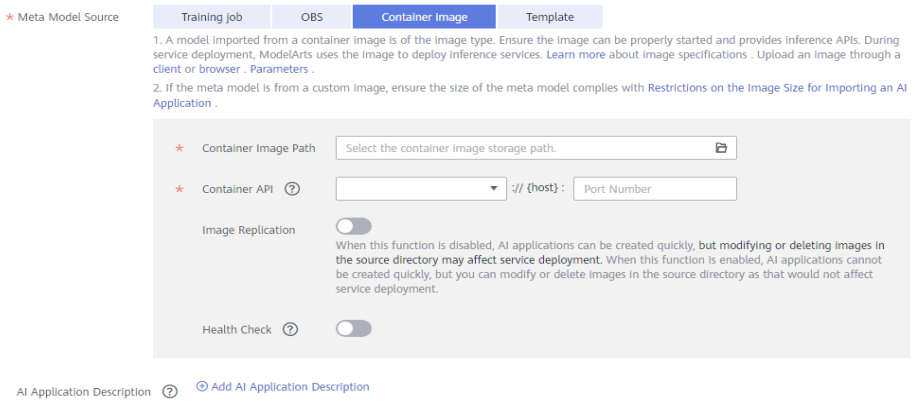


Tabla 2-10 Parámetros de la fuente del metamodelo

Parámetro	Descripción
Container Image Path	Haga clic en para importar la imagen del modelo desde la imagen del contenedor. El modelo es del tipo Imagen y no es necesario usar **swr_location** en el archivo de configuración para especificar la ubicación de la imagen. Para obtener detalles sobre la guía de operación y los requisitos para crear una imagen personalizada, véase **Especificaciones de imagen personalizada para crear aplicaciones con IA**. **NOTA** La imagen del modelo que seleccione se compartirá con el administrador del sistema, así que asegúrese de tener permiso para compartir la imagen (no se admiten imágenes compartidas con otras cuentas). Cuando se despliega un servicio, ModelArts despliega la imagen como un servicio de inferencia. Asegúrese de que la imagen se pueda iniciar correctamente y proporcione una API de inferencia.
Container API	Protocolo y número de puerto para iniciar una aplicación de IA **NOTA** El protocolo de solicitud predeterminado y el número de puerto proporcionado por ModelArts son HTTP y 8080, respectivamente. Establezcalos en función de la imagen personalizada real.

Parámetro	Descripción
Image Replication	Indica si se debe copiar la imagen del modelo en la imagen del contenedor a ModelArts. ● Cuando esta función está deshabilitada, la imagen del modelo no se copia, las aplicaciones de IA se pueden crear rápidamente, pero modificar o eliminar imágenes en el directorio de origen de SWR puede afectar al despliegue del servicio. ● Cuando esta función está habilitada, se copia la imagen del modelo, las aplicaciones de IA no se pueden crear rápidamente, pero puede modificar o eliminar imágenes en el directorio de origen de SWR, ya que eso no afectaría al despliegue del servicio. **NOTA** Debe habilitar esta función si desea utilizar imágenes compartidas por otros usuarios. De lo contrario, las aplicaciones de IA no se crearán.
Health Check	Comprobación de estado en una aplicación de IA. Este parámetro es configurable solo cuando la API de comprobación de estado está configurada en la imagen personalizada. De lo contrario, el despliegue de la aplicación de IA fallará. ● **Check Mode**: seleccione **HTTP request** o **Command**. ● **Health Check URL**: Este parámetro se muestra cuando **Check Mode** se establece en **HTTP request**. Ingrese la URL de comprobación de estado. El valor predeterminado es **/health**. ● **Health Check Command**: Este parámetro se muestra cuando **Check Mode** se establece en **Command**. Ingrese el comando de comprobación de estado. ● **Health Check Period**: Ingrese un número entero entre 1 y 2147483647. La unidad es segundo. ● **Delay(seconds)**: especifica el retraso para realizar la comprobación de estado después de iniciar la instancia. Ingrese un número entero entre 0 y 2147483647. ● **Maximum Failures**: Ingrese un número entero entre 1 y 2147483647. Durante el inicio del servicio, si el número de fallos de comprobación de estado consecutivos alcanza el valor especificado, el servicio será anormal. Durante la ejecución del servicio, si el número de fallos de comprobación de estado consecutivos alcanza el valor especificado, el servicio ingresará el estado de alarma. **NOTA** Si la comprobación de estado está configurada para una aplicación de IA, los servicios desplegados que utilizan esta aplicación de IA se detendrán 3 minutos después de recibir la instrucción de parada.

Parámetro	Descripción
Descripción de la aplicación de AI	Proporcione las descripciones de aplicaciones de IA para ayudar a otros desarrolladores de aplicaciones de IA a comprender y usar mejor sus aplicaciones. Haga clic en **Add AI Application Description** y establezca **Document name** y **URL**. Puede agregar hasta tres descripciones de aplicaciones de IA.
Deployment Type	Seleccione los tipos de servicio que se pueden desplegar en la aplicación. Al desplegar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services**, solo puede deplegar la aplicación de IA como un servicio en tiempo real después de crearla.
Start command	Comando de inicio personalizable de un modelo
Apis	Cuando habilita esta función, puede editar las API de RESTful para definir los formatos de entrada y salida de aplicaciones de IA. Las API del modelo deben cumplir con las especificaciones de ModelArts. Para obtener más información, véase **Especificaciones para editar un archivo de configuración de modelo**. Para obtener detalles sobre el ejemplo de código, vea **Ejemplo de código de parámetros de apis**.

c. Compruebe la información y haga clic en **Next**. Se crea la aplicación de IA.

   En la lista de aplicaciones de IA, puede ver la aplicación de IA creada y su versión. Cuando el estado cambia a **Normal**, la aplicación de IA se crea correctamente. En esta página, puede realizar operaciones como la creación de nuevas versiones y el despliegue rápido de servicios.

Procedimiento posterior

Despliegue de una aplicación de IA como servicio: En la lista de aplicaciones de IA, haga clic en el botón de opción a la izquierda del nombre de la aplicación de IA para mostrar la lista de versiones en la parte inferior de la página de lista. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** para desplegar la aplicación de IA como un tipo de despliegue seleccionado durante la creación de la aplicación de IA.

2.3 Consulta de la lista de aplicaciones de IA

Puede ver todas las aplicaciones de IA creadas en la página de lista de aplicaciones de IA. La página de lista de aplicaciones de IA muestra la siguiente información.

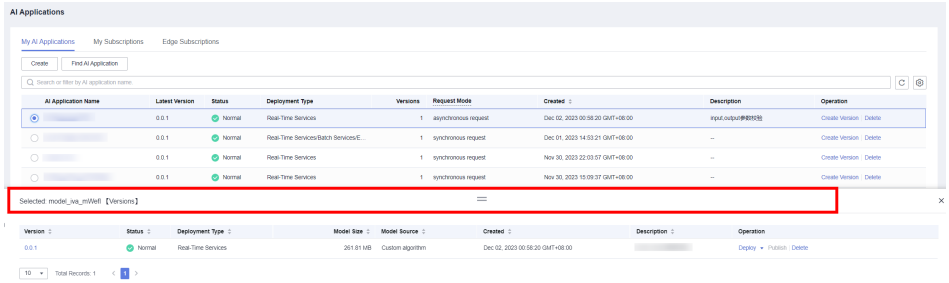
Tabla 2-11 Lista de aplicaciones de IA

Parámetro	Descripción
AI Application Name	Nombre de una aplicación de IA.

Parámetro	Descripción
Latest Version	Última versión de una aplicación de IA.
Status	Estado de una aplicación de IA.
Deployment Type	Tipos de servicios como los que se puede desplegar una aplicación de IA.
Versions	Número de versiones de aplicaciones de IA.
Request Mode	Modo de solicitud de servicios en tiempo real. ● Solicitud síncrona: inferencia única con resultados devueltos de forma síncrona (dentro de los 60 segundos). Este modo es adecuado para imágenes y videos pequeños. ● Solicitud asincrónica: inferencia única con resultados devueltos asincrónicamente (más de 60 segundos). Este modo es adecuado para inferencia de video en tiempo real y videos grandes.
Created	Fecha y hora en la que se crea una aplicación de IA.
Description	Descripción de una aplicación de IA.
Operation	● **Create Version**: Cree una versión de aplicación de IA. La configuración de la última versión se utiliza de forma predeterminada, excepto para la versión. Puede cambiar la configuración de los parámetros. ● **Delete**: Elimine la aplicación de IA. **NOTA** Si una versión de la aplicación de IA se ha desplegado como servicio, debe eliminar el servicio asociado antes de eliminar la versión de la aplicación de IA. No se puede recuperar una aplicación de IA eliminada.

Haga clic en la casilla de verificación situada junto al nombre de la aplicación de IA para mostrar la vista oculta en la parte inferior de la lista, donde puede ver la lista de versiones. (Si la vista no se muestra, haga clic en ^ en la esquina inferior derecha.)

Figura 2-4 Lista de versiones



La lista de versiones muestra la siguiente información.

Tabla 2-12 Lista de versiones

Parámetro	Descripción
Version	Versión actual de una aplicación de IA.
Status	Estado de una aplicación de IA.
Deployment Type	Tipos de servicios como los que se puede desplegar una aplicación de IA.
Model Size	Tamaño de una aplicación de IA.
Model Source	Fuente del modelo de una aplicación de IA.
Created	Fecha y hora en la que se crea una aplicación de IA.
Description	Descripción de una aplicación de IA.
Operation	● **Deploy**: Despliegue una aplicación de IA como servicios en tiempo real, servicios por lotes o servicios perimetrales. ● **Delete**: Elimine una versión de una aplicación de IA.

2.4 Consulta de detalles sobre una aplicación de IA

Después de crear una aplicación de IA, puede ver su información en la página de detalles.

1. Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación de la izquierda, elija **AI Application Management** > **AI Applications**. Se muestra la página **AI Applications**.
2. Haga clic en el nombre de la aplicación de IA de destino. Se muestra la página de detalles de la aplicación.

   En la página de detalles de la aplicación, puede ver la información básica y la precisión del modelo de la aplicación de IA, y cambiar las páginas de fichas para ver más información.

   Tabla 2-13 Información básica sobre una aplicación de IA

Parámetro	Descripción
Name	Nombre de una aplicación de IA
Status	Estado de una aplicación de IA
Version	Versión actual de una aplicación de IA
ID	ID de una aplicación de IA
Description	Haga clic en el botón de edición para agregar la descripción de una aplicación de IA.
Deployment Type	Tipos de servicios que puede desplegar una aplicación de IA

Parámetro	Descripción
Meta Model Source	Origen del metamodelo, que puede ser trabajos de entrenamiento, OBS o imágenes de contenedor.
Training Name	Trabajo de entrenamiento asociado si el metamodelo proviene de un trabajo de entrenamiento. Haga clic en el nombre del trabajo de entrenamiento para ir a su página de detalles.
Training Version	Versión de trabajo de entrenamiento si el metamodelo proviene de un trabajo de entrenamiento de versión antigua.
Storage path of the meta model	Ruta al metamodelo si el metamodelo proviene de OBS.
Container Image Storage Path	Ruta de acceso a la imagen de contenedor si el metamodelo procede de una imagen de contenedor.
AI Engine	Motor de IA si el metamodelo proviene de un trabajo de entrenamiento u OBS.
Engine Package Address	Dirección del paquete del motor si el metamodelo proviene de OBS y el **AI Engine** es **Custom**.
Runtime Environment	Entorno de tiempo de ejecución del que depende el metamodelo si el metamodelo proviene de un trabajo de entrenamiento u OBS y se utiliza un motor de IA preestablecido.
Container API	Protocolo y número de puerto para iniciar la aplicación de IA si el metamodelo proviene de OBS (**AI Engine** es **Custom**) o una imagen de contenedor.
Inference Code	Ruta al código de inferencia si el metamodelo proviene de un trabajo de entrenamiento de versión antigua.
Image Replication	Estado de replicación de imagen si el metamodelo proviene de OBS o una imagen de contenedor.
Dynamic loading	Estado de carga dinámica si el metamodelo proviene de un trabajo de entrenamiento u OBS.
Size	Tamaño de una aplicación de IA
Health Check	Estado de comprobación de estado si el metamodelo proviene de OBS o de una imagen de contenedor. Si la comprobación de estado está habilitada, se muestran los siguientes parámetros: **Check Mode**, **Health Check URL**, **Health Check Period**, **Delay** y **Maximum Failures**.
AI Application Description	Documento de descripción agregado durante la creación de una aplicación de IA.
Instruction Set Architecture	Arquitectura del sistema.

Parámetro	Descripción
Inference Accelerator	Tipo de tarjetas aceleradoras de inferencia.

Tabla 2-14 Página de detalles de una aplicación de IA

Parámetro	Descripción
Model Precision	Retirada de modelos, precisión, precisión y puntuación F1 de una aplicación de IA
Parameter Configuration	Configuración de API, parámetros de entrada y parámetros de salida de una aplicación de IA
Runtime Dependency	La dependencia del modelo en el entorno. Si se ha producido un error al crear un trabajo, edite la dependencia en tiempo de ejecución. Después de guardar la modificación, el sistema utilizará automáticamente la imagen original para crear el trabajo de nuevo.
Events	El progreso de las operaciones clave durante la creación de aplicaciones de IA Los eventos se almacenan durante tres meses y luego se borrarán automáticamente. Para obtener detalles sobre cómo ver los eventos de una aplicación de IA, véase **Consulta de eventos de una aplicación de IA**.
Constraint	Muestra las restricciones del despliegue de servicio, como el modo de solicitud, el comando de inicio y la encriptación de modelo, según la configuración durante la creación de la aplicación de IA. Para las aplicaciones de IA en modo de solicitud asincrónica, se pueden mostrar parámetros como el modo de entrada, el modo de salida, los parámetros de inicio del servicio y los parámetros de configuración de trabajos.
Associated Services	La lista de servicios que se desplegó una aplicación de IA. Haga clic en un nombre de servicio para ir a la página de detalles del servicio.

2.5 Gestión de versiones de aplicaciones de IA

Para facilitar el seguimiento del origen y el ajuste repetido de la aplicación de IA, el ModelArts proporciona la función de gestión de versiones de la aplicación de IA. Puede gestionar los modelos basados en versiones.

Requisitos previos

ModelArts ha creado una aplicación de IA.

Creación de una nueva versión

En la página **AI Application Management** > **AI Applications**, haga clic en **Create Version** en la columna **Operation** de la aplicación de IA de destino. En la página **Create Version**, configure los parámetros. Para más detalles, véase **Creación de una aplicación de IA**. Haga clic en **Create Now**.

Eliminación de una versión

En la página **AI Application Management > AI Applications**, haga clic en el botón de opción a la izquierda del nombre de la aplicación de IA para mostrar la lista de versiones de la aplicación. En la lista de versiones de la aplicación, haga clic en **Delete** en la columna **Operation** para eliminar la versión correspondiente.

NOTA

Si se ha desplegado un servicio para la versión de la aplicación de IA, debe eliminar el servicio asociado antes de eliminar la versión de la aplicación de IA. No se puede recuperar una versión eliminada. Realice esta operación con precaución.

Eliminación de una aplicación de IA

En el panel de navegación, elija **AI Application Management** > **AI Applications**. En la página **AI Applications**, haga clic en **Delete** en la columna **Operation** para eliminar la aplicación de IA de destino.

NOTA

Si se ha desplegado un servicio para la versión de la aplicación de IA, debe eliminar el servicio asociado antes de eliminar la versión de la aplicación de IA. No se puede recuperar una aplicación de IA eliminada. Realice esta operación con precaución.

2.6 Consulta de eventos de una aplicación de IA

Durante la creación de una aplicación de IA, cada evento clave se registra automáticamente. Puede ver los eventos en la página de detalles de la aplicación de IA en cualquier momento.

Esto le ayuda a comprender mejor el proceso de creación de una aplicación de IA y a localizar fallas con mayor precisión cuando ocurre una excepción de tarea. En la siguiente tabla se enumeran los eventos disponibles.

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Normal	El modelo comienza a importarse.	-
Abnormal	Error al crear la imagen.	Localice y rectifique la falla según la información del error. **Preguntas frecuentes**

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Abnormal	La imagen personalizada no admite las dependencias especificadas.	Las dependencias en tiempo de ejecución no se pueden configurar cuando se importa una imagen personalizada. Instale el paquete de dependencias pip en el Dockerfile que se utiliza para crear la imagen. **Preguntas frecuentes**
Abnormal	Solo las imágenes personalizadas admiten **swr_location**.	Elimine el campo **swr_location** del archivo de configuración del modelo **config.json** e inténtelo de nuevo.
Abnormal	La API de comprobación de estado de una imagen personalizada debe ser *xxx*.	Modifique la API de comprobación de estado de la imagen personalizada y vuelva a intentarlo.
Normal	La tarea de creación de imagen se encuentra en estado *xxx*.	-
Abnormal	La etiqueta *xxx* no existe en la imagen *xxx*.	Contacte con el servicio de asistencia técnica.
Abnormal	Existe un valor de parámetro *xxx* no válido en el archivo de configuración del modelo.	Elimine parámetros no válidos del archivo de configuración del modelo y vuelva a intentarlo.
Abnormal	Error al obtener las etiquetas de la imagen *xxx*.	Contacte con el servicio de asistencia técnica.
Abnormal	Error al importar los datos porque *xxx* supera los *xxx* GB.	El tamaño del modelo o la imagen supera el límite superior. Reduzca el modelo o la imagen e impórtelo de nuevo. **Preguntas frecuentes**

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Abnormal	El usuario *xxx* no tiene permiso de OBS obs:object:PutObjectAcl.	El usuario de IAM no tiene el permiso obs:object:PutObjectAcl en OBS. Agregue el permiso de la delegación para el usuario de IAM. **Preguntas frecuentes**
Abnormal	Tiempo fuera de creación de la imagen. La duración del tiempo fuera es de *xxx* minutos.	Hay un límite de tiempo de espera para la creación de imágenes con ImagePacker. Simplifique el código para mejorar la eficiencia. **Preguntas frecuentes**
Normal	Descripción del modelo actualizada.	-
Normal	Dependencias de tiempo de ejecución del modelo no actualizadas.	-
Normal	Dependencias de tiempo de ejecución del modelo actualizadas. Recreación de la imagen.	-
Abnormal	Control de tráfico de SWR activado. Vuelva a intentarlo más tarde.	Control de tráfico de SWR activado. Vuelva a intentarlo más tarde.
Normal	Se está haciendo actualizando el sistema. Vuelva a intentarlo más tarde.	-
Abnormal	Error al obtener la imagen de origen. Ocurrió un error en la autenticación. El token ha caducado.	Contacte con el servicio de asistencia técnica.
Abnormal	Error al obtener la imagen de origen. Compruebe si la imagen existe.	Contacte con el servicio de asistencia técnica.
Normal	Tamaño de imagen de origen calculado.	-
Normal	Imagen de origen compartida.	-
Abnormal	Error al crear la imagen debido al control de tráfico. Vuelva a intentarlo más tarde.	Control de tráfico activado. Vuelva a intentarlo más tarde.
Abnormal	Error al enviar la solicitud de creación de imagen.	Contacte con el servicio de asistencia técnica.

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Abnormal	Error al compartir la imagen de origen. Verifique si la imagen existe o si tiene permiso para compartirla.	Verifique si la imagen existe o si tiene permiso para compartirla.
Normal	El modelo importado.	-
Normal	Archivo de modelo importado.	-
Normal	Tamaño del modelo calculado.	-
Abnormal	Error al importar el modelo.	Para obtener detalles sobre cómo localizar y rectificar la falla, véase **Preguntas frecuentes**.
Abnormal	Error al copiar el archivo de modelo. Compruebe si tiene el permiso de OBS.	Compruebe si tiene el permiso de OBS. **Preguntas frecuentes**
Abnormal	Error al programar la tarea de creación de imagen.	Contacte con el servicio de asistencia técnica.
Abnormal	Error al iniciar la tarea de creación de imagen.	Contacte con el servicio de asistencia técnica.
Abnormal	La imagen de Roman se ha creado pero no se puede compartir con los tenants del recurso.	Contacte con el servicio de asistencia técnica.
Normal	Imagen creada.	-
Normal	Se inició la tarea de creación de imagen.	-
Normal	Se inició la tarea de creación de imagen de entorno.	-
Normal	Solicitud de creación de una imagen de entorno recibida.	-
Normal	Se ha recibido la solicitud para crear una imagen.	-
Normal	Se utiliza una imagen de entorno existente.	-
Abnormal	Error al crear la imagen. Para obtener más información, véase registros de creación de imágenes.	Consulte los logs de compilación para localizar y rectificar la falla. **Preguntas frecuentes**

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Abnormal	Error al crear la imagen debido a un error interno del sistema. Contacte con el servicio de asistencia técnica.	Contacte con el servicio de asistencia técnica.
Abnormal	Error al importar el archivo de modelo *xxx* porque supera los 5 GB.	El tamaño del archivo de modelo *xxx* es superior a 5 GB. Reduzca el tamaño del archivo de modelo e inténtelo de nuevo, o utilice la carga dinámica para importar el archivo de modelo. **Preguntas frecuentes**
Abnormal	Error al crear el bucket de OBS debido a un error interno del sistema. Contacte con el servicio de asistencia técnica.	Contacte con el servicio de asistencia técnica.
Abnormal	Error al calcular el tamaño del modelo. La subruta *xxx* no existe en la ruta *xxx*.	Corrija la ruta secundaria y vuelva a intentar o póngase en contacto con el soporte técnico.
Abnormal	Error al calcular el tamaño del modelo. El modelo del tipo *xxx* no existe en la ruta *xxx*.	Compruebe la ubicación de almacenamiento del modelo del tipo *xxx*, corrija la ruta y vuelva a intentar, o póngase en contacto con el soporte técnico.
Warning	Error al calcular el tamaño del modelo. En la ruta *xxx* hay más de un archivo de modelo *xxx* almacenado.	-

Durante la creación de aplicaciones de IA, los eventos clave se pueden actualizar manual y automáticamente.

Consulta de eventos

1. En el panel de navegación de la consola de gestión de ModelArts, seleccione **AI Application Management** > **AI Applications**. En la lista de aplicaciones de IA, haga clic en el nombre de la aplicación de IA de destino para ir a su página de detalles.
2. Vea los eventos en la pestaña **Events**.

3 Despliegue de una aplicación de IA como servicio

3.1 Despliegue de aplicaciones de IA como servicios en tiempo real

3.1.1 Despliegue como servicio en tiempo real

Después de preparar una aplicación de IA, puede desplegarla como un servicio en tiempo real e invocar al servicio para la predicción.

Restricciones

Un usuario puede desplegar un máximo de 20 servicios en tiempo real.

Requisitos previos

- Se han preparado los datos. Específicamente, ha creado una aplicación de IA en el estado **Normal** de ModelArts.
- La cuenta no está en mora para garantizar los recursos disponibles para la ejecución del servicio.

Procedimiento

1. Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación izquierdo, elija **Service Deployment** > **Real-Time Services**. La lista de servicios en tiempo real se muestra por defecto.
2. En la lista de servicios en tiempo real, haga clic en **Deploy** en el extremo superior izquierdo. Se muestra la página **Deploy**.
3. Establezca parámetros para un servicio en tiempo real.
   a. Establezca información básica acerca del despliegue del modelo. Para obtener detalles sobre los parámetros, véase **Tabla 3-1**.

Tabla 3-1 Parámetros básicos

Parámetro	Descripción
Name	Nombre del servicio en tiempo real. Establezca este parámetro como se le solicite.
Auto Stop	Después de activar este parámetro y establecer el tiempo de parada automática, un servicio se detiene automáticamente a la hora especificada. Si este parámetro está desactivado, un servicio en tiempo real se mantiene en ejecución y facturación. La función puede ayudarlo a evitar la facturación innecesaria. La función de parada automática está activada de forma predeterminada y el valor predeterminado es **1 hour later**. Las opciones son **1 hour later**, **2 hours later**, **4 hours later**, **6 hours later** y **Custom**. Si selecciona **Custom**, puede escribir cualquier entero de 1 a 24 horas en el cuadro de texto de la derecha.
Description	Breve descripción del servicio en tiempo real.

b. Ingrese la información clave, incluidas las configuraciones del grupo de recursos y de la aplicación de IA. Para más detalles, véase **Tabla 3-2**.

Tabla 3-2 Parámetros

Parámetro	Subparámetro	Descripción
Resource Pool	Public Resource Pool	Los recursos de cómputo de CPU/GPU están disponibles para que los seleccione. Los estándares de precios para los grupos de recursos con diferentes variantes son diferentes. Para obtener más información, véase **Detalles de precios de productos**. El grupo de recursos público solo soporta el modo de facturación de pago por uso.

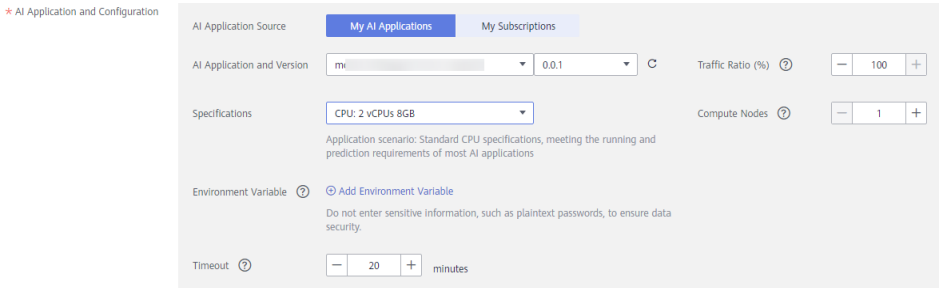
Parámetro	Subparámetro	Descripción
	Dedicated Resource Pool	Seleccione una especificación de las especificaciones del grupo de recursos dedicados. Los grupos físicos con subgrupos lógicos creados no se admiten temporalmente. **NOTA** ● Los datos de los grupos de recursos dedicados de la versión anterior se migrarán gradualmente a los grupos de recursos dedicados de la nueva versión. ● Para los usuarios nuevos y los existentes que han migrado datos de los grupos de recursos dedicados de versiones anteriores a los nuevos, solo hay una entrada para los de versiones nuevas en la consola de gestión de ModelArts. ● Para los usuarios existentes que no han migrado datos de los grupos de recursos dedicados de versiones anteriores a nuevos, hay dos entradas para los grupos en la consola de gestión de ModelArts, donde la entrada marcada con **New** es para la nueva versión. Para obtener más información sobre la nueva versión de los grupos de recursos dedicados, véase **Actualizaciones amplias de las funciones de gestión del grupo de recursos de ModelArts**.
AI Application and Configuration	AI Application Source	Seleccione **My AI Applications** o **My Subscriptions** según sus requisitos.
	AI Application and Version	Seleccione la aplicación de IA y la versión que están en el estado **Normal**.
	Traffic Ratio (%)	Establezca la proporción de tráfico del nodo de instancia actual. Las solicitudes de invocaciones de servicio se asignan a la versión actual en función de esta proporción. Si despliega solo una versión de una aplicación de IA, establezca este parámetro en **100%**. Si selecciona varias versiones para la versión intermedia, asegúrese de que la suma de las relaciones de tráfico de estas versiones sea **100%**.

Parámetro	Subparámetro	Descripción
	Specifications	Seleccione las especificaciones disponibles según la lista mostrada en la consola. Las especificaciones en gris no se pueden utilizar en el entorno actual. Si las especificaciones de los grupos de recursos públicos no están disponibles, no hay ningún grupo de recursos públicos disponible en el entorno actual. En este caso, utilice un grupo de recursos dedicado o póngase en contacto con el administrador para crear un grupo de recursos público. **NOTA** Cuando se utiliza la variante seleccionada para desplegar el servicio, se genera el consumo necesario del sistema. Por lo tanto, los recursos realmente ocupados por el servicio son ligeramente mayores que la variante seleccionada.
	Compute Nodes	Establezca el número de instancias para la versión actual de la aplicación de IA. Si establece el número de nodos en **1**, se utilizará el modo de cómputo independiente. Si establece el número de nodos en un valor mayor que 1, se utilizará el modo de cómputo distribuido. Seleccione un modo de cómputo basado en los requisitos reales.
	Environment Variable	Establezca las variables de entorno e inyéctelas en el pod. Para garantizar la seguridad de los datos, no introduzca la información confidencial como contraseñas de texto sin formato en las variables de entorno.
	Timeout	Tiempo de espera de un único modelo, incluido el tiempo de despliegue y de inicio. El valor predeterminado es 20 minutos. El valor debe estar dentro del rango de 3 a 120.
	Add AI Application Version and Configuration	Si la aplicación de IA seleccionada tiene varias versiones, puede agregar varias versiones y configurar una relación de tráfico. Puede usar la versión intermedia para actualizar sin problemas la versión de la aplicación de IA. **NOTA** Las especificaciones de cómputo gratuito no admiten la versión intermedia de varias versiones.

Parámetro	Subparámetro	Descripción
	Mount Storage	Este parámetro se muestra cuando el grupo de recursos es un grupo de recursos dedicado. Esta función montará un volumen de almacenamiento para calcular nodos (instancias de cómputo) como un directorio local cuando el servicio se está ejecutando. Se recomienda cuando el modelo o los datos de entrada son grandes. Hay dos tipos de volúmenes: sistema de archivos paralelo de OBS y sistema de archivos de SFS. Actualmente, solo se admiten los sistemas de archivos paralelos de OBS. ● Sistema de archivos paralelo de OBS – **Source Path**: seleccione la ruta de almacenamiento del archivo paralelo. No se puede seleccionar un sistema de archivos paralelo de OBS entre regiones. – **Mount Path**: introduzca la ruta de montaje del contenedor, por ejemplo, **/tmp**. – Para evitar excepciones de contenedor, no monte el almacenamiento en un directorio del sistema como / o **/var/run**. – Es una buena práctica montar el contenedor en un directorio vacío. Si el directorio no está vacío, asegúrese de que no haya archivos que afecten al inicio del contenedor en el directorio. De lo contrario, dichos archivos se reemplazarán, lo que provocará errores al iniciar el contenedor y crear la carga de trabajo. – La ruta de montaje debe comenzar con una barra diagonal (/) y puede contener un máximo de 1,024 caracteres, incluidas las letras, los dígitos y los siguientes caracteres especiales: \ _ -. ● Sistema de archivos de SFS (no soportado) **NOTA** El montaje de almacenamiento solo puede ser utilizado por servicios desplegados en un grupo de recursos dedicado.
Traffic Limit	N/A	Número máximo de veces que se puede acceder a un servicio en un segundo. Puede establecer este parámetro según sea necesario.

Parámetro	Subparámetro	Descripción
WebSocket	N/A	Si se debe desplegar un servicio en tiempo real como servicio de WebSocket. Para obtener más información sobre los servicios en tiempo real de WebSocket, véase **Desarrollo de proceso completo de servicios en tiempo real de WebSocket**. **NOTA** ● Esta función solo se admite si la aplicación de IA es compatible con WebSocket y proviene de una imagen de contenedor. ● Una vez habilitada esta función, no se pueden configurar **Traffic Limit** ni **Data Collection**. ● Este parámetro no se puede cambiar una vez desplegado el servicio.
Runtime Log Output	N/A	Esta función está deshabilitada por defecto. Los logs de tiempo de ejecución de los servicios en tiempo real se almacenan solo en el sistema de logs de ModelArts. Puede consultar los logs de tiempo de ejecución en la pestaña **Logs** de la página de detalles del servicio. Si esta función está habilitada, los logs de tiempo de ejecución de los servicios en tiempo real se exportarán y almacenarán en Log Tank Service (LTS). LTS crea automáticamente grupos de logs y flujos de logs y cachés ejecuta logs generados en un plazo de siete días de forma predeterminada. Para obtener más detalles sobre la función de gestión de logs LTS, véase **Log Tank Service**. **NOTA** ● Esto no se puede deshabilitar una vez que está habilitado. ● Se le facturarán las funciones de consulta de logs y almacenamiento de logs proporcionadas por LTS. Para obtener más detalles, véase la sección **Detalles de precios de LTS**.
Application Authentication	Application	Deshabilitada por defecto. Para habilitar esta función, véase **Acceso autenticado mediante una aplicación** para obtener más detalles y configure los parámetros según sea necesario.

Figura 3-1 Configuración de la información de la aplicación de IA



c. (Opcional) Configure los ajustes avanzados.

Tabla 3-3 Configuración avanzada

Parámetro	Descripción
Tags	ModelArts puede trabajar con Tag Management Service (TMS). Al crear tareas que consumen recursos de ModelArts, por ejemplo, trabajos de entrenamiento, configure etiquetas para estas tareas de modo que ModelArts pueda usar etiquetas para gestionar recursos por grupo. Para obtener más información sobre cómo usar etiquetas, véase **¿Cómo usa ModelArts etiquetas para gestionar recursos por grupo?** **NOTA** Puede seleccionar una etiqueta de TMS predefinida de la lista desplegable de etiquetas o personalizar una etiqueta. Las etiquetas predefinidas están disponibles para todos los recursos de servicio que admiten etiquetas. Las etiquetas personalizadas solo están disponibles para los recursos de servicio del usuario que las ha creado.

4. Después de confirmar la información introducida, complete el despliegue del servicio como se le solicite. Por lo general, los trabajos de despliegue de servicio se ejecutan durante un período de tiempo, que puede ser de varios minutos o decenas de minutos, dependiendo de la cantidad de datos y recursos seleccionados.

   NOTA

   Una vez desplegado un servicio en tiempo real, se inicia de inmediato.

   Puede ir a la lista de servicios en tiempo real para comprobar si se ha completado el despliegue del servicio en tiempo real. En la lista de servicios en tiempo real, después de que el estado del servicio recién desplegado cambie de **Deploying** a **Running**, el servicio se despliega correctamente.

3.1.2 Consulta de detalles del servicio

Después de implementar una aplicación de IA como servicio en tiempo real, puede acceder a la página del servicio para ver sus detalles.

1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** > **Real-Time Services**.
2. En la página **Real-Time Services**, haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio.

   Puede ver el nombre del servicio, el estado y otra información. Para más detalles, véase **Tabla 3-4**.

Tabla 3-4 Parámetros de servicio en tiempo real

Parámetro	Descripción
Name	Nombre del servicio en tiempo real.

Parámetro	Descripción
Status	Estado del servicio en tiempo real.
Source	Fuente de aplicación de IA del servicio en tiempo real.
Service ID	ID de servicio en tiempo real
Description	Descripción del servicio, que se puede editar después de hacer clic en el botón de edición en el lado derecho.
Resource Pool	Especificaciones de grupo de recursos utilizadas por el servicio. Si se utiliza el grupo de recursos públicos para el despliegue, no se muestra este parámetro.
Custom Settings	Configuraciones personalizadas basadas en versiones de servicio en tiempo real. Esto permite configuraciones y políticas de distribución de tráfico basadas en versiones. Active esta opción y haga clic en **View Settings** para personalizar la configuración. Para más detalles, véase **Modificación de parámetros personalizados**.
Traffic Limit	Número máximo de veces que se puede acceder a un servicio en un segundo.
Runtime Log Output	Esta función está deshabilitada por defecto. Los logs de tiempo de ejecución de los servicios en tiempo real se almacenan solo en el sistema de logs de ModelArts. Si esta función está habilitada, los logs de tiempo de ejecución de los servicios en tiempo real se exportarán y almacenarán en Log Tank Service (LTS). LTS crea automáticamente grupos de logs y flujos de logs y cachés ejecuta logs generados en un plazo de siete días de forma predeterminada. Para obtener más detalles sobre la función de gestión de logs LTS, véase **Log Tank Service**. **NOTA** ● Esto no se puede deshabilitar una vez que está habilitado. ● Se le facturarán las funciones de consulta de logs y almacenamiento de logs proporcionadas por LTS. Para obtener más detalles, véase la sección **Detalles de precios de LTS**.
WebSocket	Si se debe actualizar al servicio de WebSocket.

3. Cambie entre las pestañas de la página de detalles de un servicio en tiempo real para ver más detalles. Para más detalles, véase **Tabla 3-5**.

Tabla 3-5 Detalles de un servicio en tiempo real

Parámetro	Descripción
Usage Guides	Esta página muestra la URL de la API, la información de la aplicación de IA, los parámetros de entrada y los parámetros de salida. Puede hacer clic en  para copiar la URL de API para invocar al servicio. Si se admite la autenticación de aplicaciones, en las **Guías de uso** puede ver los detalles de URL de la API y de gestión de autorizaciones, incluidos el nombre de la aplicación, AppKey y AppSecret. También puede agregar o cancelar la autorización para una aplicación.
Prediction	Puede realizar la predicción en tiempo real en esta página. Para más detalles, véase **Prueba del servicio desplegado**.
Configuration Updates	Esta página muestra **Current Configurations** y **Update History**. ● **Current Configurations**: nombre, versión, estado, especificaciones de nodo de cómputo, relación de tráfico, número de nodos de cómputo, intervalo de tiempo de espera de despliegue, variables de entorno, montaje de almacenamiento e información de grupo de recursos de aplicación de IA (para servicios desplegados en un grupo de recursos dedicado) ● **Update History**: información histórica de aplicaciones de IA.
Monitoring	Esta página muestra el uso de recursos y las llamadas de aplicaciones de IA. ● **Resource Usage**: incluye las CPU usadas y disponibles, la memoria y la GPU. ● **AI Application Calls**: indica el número de llamadas de aplicación de IA. La recopilación de estadísticas comienza después de que el estado de la aplicación de IA cambie a **Ready**. (Este parámetro no se muestra para los servicios de WebSocket)
Event	Esta página muestra operaciones clave durante el uso del servicio, como el progreso de despliegue del servicio, causas detalladas de excepciones de despliegue y puntos en el tiempo cuando se inicia, para o modifica un servicio. Los eventos se guardan durante un mes y luego se borran automáticamente. Para obtener detalles sobre cómo ver los eventos de un servicio, véase **Consulta de eventos de servicio**.

Parámetro	Descripción
Logs	Esta página muestra la información de log de cada aplicación de IA del servicio. Puede ver los registros generados en los últimos 5 minutos, los últimos 30 minutos, las últimas 1 hora y el segmento de tiempo definido por el usuario. Puede seleccionar la hora de inicio y la hora de finalización al definir el segmento de tiempo. Si esta función está habilitada, se mostrarán los logs almacenados en LTS. Puede hacer clic en **View Complete Logs on LTS** para ver todos los logs. Cumpla las siguientes reglas para buscar logs: ● No introduzca cadenas que contengan los siguientes delimitadores: **,'";=() [] {}@&<>/:\n\t\r**. ● Ingrese palabras clave para la búsqueda exacta. Una palabra clave es una palabra entre dos delimitadores adyacentes. ● Ingrese palabras clave para búsqueda difusa. Por ejemplo, puede escribir **error**, **er?or**, **rro*** o **er*r**. ● Ingrese las frases para la búsqueda exacta. Por ejemplo, **Start to refresh**. ● Antes de habilitar esta función, puede combinar las palabras clave con AND (**&&**) u OR (||). Por ejemplo, **query logs&&erro*** o **query logs||erro***. Una vez habilitada esta función, se pueden combinar palabras clave con **AND** u **OR**. Por ejemplo, **query logs AND erro*** o **query logs OR erro***.
Tags	Etiquetas que se han agregado al servicio. Las etiquetas se pueden agregar, modificar y eliminar. Para obtener más información sobre cómo usar etiquetas, véase **¿Cómo usa ModelArts etiquetas para gestionar recursos por grupo?**

Modificación de parámetros personalizados

Una regla de configuración personalizada consiste en la condición de configuración (**Setting**), la versión de acceso (**Version**) y los parámetros de ejecución personalizados (incluidos **Setting Name** y **Setting Value**).

Puede configurar diferentes ajustes con parámetros de ejecución personalizados para diferentes versiones de un servicio en tiempo real.

Las prioridades de las reglas de configuración personalizadas están en orden descendente. Puede cambiar las prioridades arrastrando la secuencia de reglas de configuración personalizadas.

Una vez coincidente una regla, el sistema ya no coincidirá con las reglas posteriores. Se puede configurar un máximo de 10 reglas de configuración.

Tabla 3-6 Parámetros para **Custom Settings**

Parámetro	Obligatorio	Descripción
Setting	Sí	Expresión de la regla Spring Expression Language (SPEL). Solo se admiten las expresiones igual, coinciden y hashCode del tipo de carácter.
Version	Sí	Versión de acceso para una regla de configuración de servicio personalizada. Cuando se coincide con una regla, se solicita el servicio en tiempo real de la versión.
Setting Name	No	Clave de un parámetro de ejecución personalizado, que consta de un máximo de 128 caracteres. Configure este parámetro si el encabezado del mensaje HTTP se utiliza para llevar parámetros de ejecución personalizados a un servicio en tiempo real.
Setting Value	No	Valor de un parámetro de ejecución personalizado, que consta de un máximo de 256 caracteres. Configure este parámetro si el encabezado del mensaje HTTP se utiliza para llevar parámetros de ejecución personalizados a un servicio en tiempo real.

La configuración personalizada se puede utilizar en los siguientes escenarios:

- Si se despliegan varias versiones de un servicio en tiempo real para la versión intermedia, se pueden utilizar los ajustes personalizados para distribuir el tráfico por usuario.

  Tabla 3-7 Variables incorporadas

Variable incorporada	Descripción
DOMAIN_NAME	Nombre de cuenta que se utiliza para invocar a una solicitud de inferencia
DOMAIN_ID	ID de cuenta que se utiliza para invocar a una solicitud de inferencia
PROJECT_NAME	Nombre del proyecto que se utiliza para invocar a una solicitud de inferencia
PROJECT_ID	ID de proyecto que invoca la solicitud de inferencia
USER_NAME	Nombre de usuario que se utiliza para invocar a una solicitud de inferencia
USER_ID	ID de usuario que se utiliza para invocar a una solicitud de inferencia

La clave Pound (#) indica que se hace referencia a una variable. La string de caracteres coincidentes debe estar entre comillas simples.

```
#{Built-in variable} == 'Character string'
#{Built-in variable} matches 'Regular expression'
```

– Ejemplo 1:

  Si el nombre de cuenta para invocar la solicitud de inferencia es **User A**, la versión especificada coincide.

  ```
  #DOMAIN_NAME == 'User A'
  ```

– Ejemplo 2:

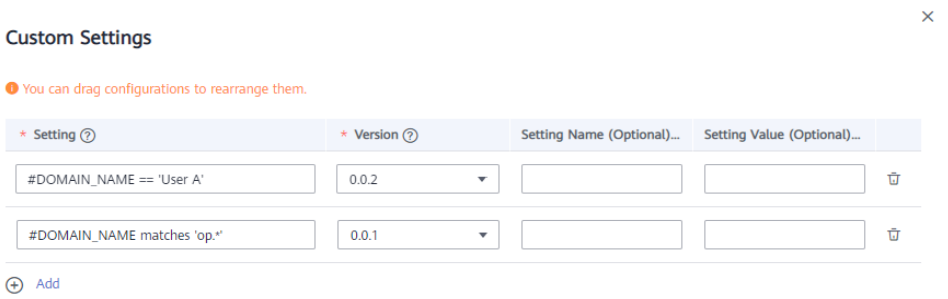
  Si el nombre de cuenta en la solicitud de inferencia comienza con **op**, la versión especificada coincide.

  ```
  #DOMAIN_NAME matches 'op.*'
  ```

Tabla 3-8 Expresiones regulares comunes

Carácter	Descripción
.	Coincide con cualquier carácter, excepto **\n**. Para que coincida con cualquier carácter, incluido **\n**, utilice **(.|\n)**.
*	Coincide con la subexpresión que sigue para cero o varias veces. Por ejemplo, **zo*** puede coincidir con **z** y **zoo**.
+	Coincide con la subexpresión que sigue una o varias veces. Por ejemplo, **zo+** puede coincidir con **zo** y **zoo**, pero no puede igualar **z**.
?	Coincide con la subexpresión que sigue durante cero o una vez. Por ejemplo, **do(es)?** puede igualar **does** o **do** en **does**.
^	Coincide con el inicio de la string de entrada.
$	Coincide con el final de la string de entrada.
{n}	*n* es un entero no negativo que coincide exactamente con el número *n* de apariciones de una expresión. Por ejemplo, **o{2}** no puede coincidir con **o** en **Bob**, pero puede coincidir con dos **o** en **food**.
x|y	Coincide con x o y. Por ejemplo, **z|food** puede coincidir con **z** o **food**, y **(z|f)ood** puede coincidir con **zood** o **food**.
[xyz]	Conjuntos de caracteres, en el que se puede hacer coincidir cualquier carácter. Por ejemplo, **[abc]** puede coincidir con **a** en **plain**.

Figura 3-2 Distribución del tráfico por usuario



- Si se despliegue varias versiones de un servicio en tiempo real para el inicio cerrado, se pueden usar configuraciones personalizadas para acceder a diferentes versiones a través del encabezado.

  Comience con **#HEADER_** para indicar que se hace referencia al encabezado como condición.

  ```
  #HEADER_{key} == '{value}'
  #HEADER_{key} matches '{value}'
  ```

  – Ejemplo 1:

    Si el encabezado de una solicitud HTTP de inferencia contiene una versión y el valor es **0.0.1**, se cumple la condición. De lo contrario, la condición no se cumple.

    ```
    #HEADER_version == '0.0.1'
    ```

  – Ejemplo 2:

    Si el encabezado de una solicitud HTTP de inferencia contiene **testheader** y el valor comienza con **mock**, la regla coincide.

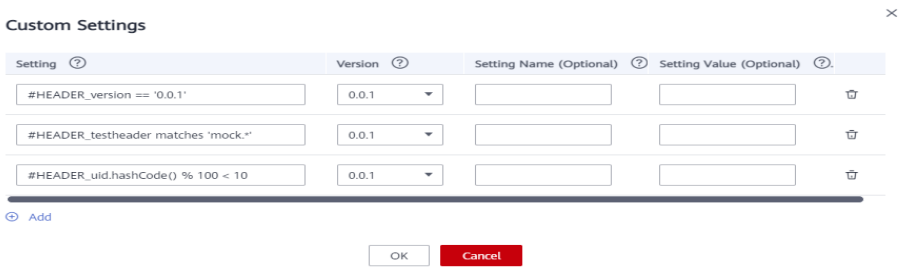
    ```
    #HEADER_testheader matches 'mock.*'
    ```

  – Ejemplo 3:

    Si el encabezado de una solicitud HTTP de inferencia contiene **uid** y el valor del código hash cumple las condiciones descritas en el siguiente algoritmo, la regla coincide.

    ```
    #HEADER_uid.hashCode() % 100 < 10
    ```

    Figura 3-3 Usar el encabezado para acceder a diferentes versiones

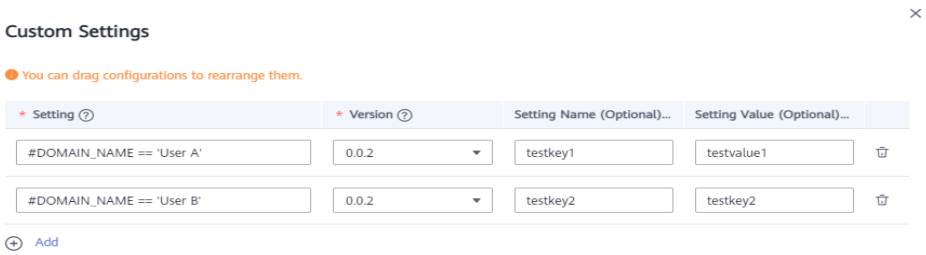
    

- Si una versión de servicio en tiempo real admite diferentes configuraciones de tiempo de ejecución, puede utilizar **Setting Name** y **Setting Value** para especificar parámetros de tiempo de ejecución personalizados para que diferentes usuarios puedan utilizar diferentes configuraciones de ejecución.

  Por ejemplo:

  Cuando el usuario A accede a la aplicación de IA, el usuario utiliza la configuración A. Cuando el usuario B accede a la aplicación de IA, el usuario utiliza la configuración B. Cuando se coincide con una configuración en ejecución, el ModelArts agrega un encabezado a la solicitud y también los parámetros de ejecución personalizados especificados por **Setting Name** y **Setting Value**.

Figura 3-4 Parámetros de ejecución personalizados agregados para una regla de configuración personalizada



3.1.3 Prueba del servicio desplegado

Después de desplegar una aplicación de IA como servicio en tiempo real, puede depurar código o agregar archivos para probarlos en la pestaña **Prediction**. En función de la solicitud de entrada (texto o archivo JSON) definida por la aplicación de IA, el servicio se puede probar de cualquiera de las siguientes maneras:

- **Predicción de texto de JSON**: Si el tipo de entrada de la aplicación de IA del servicio desplegado es texto JSON, es decir, la entrada no contiene archivos, puede introducir el código JSON en la página de ficha **Prediction** para realizar pruebas de servicio.
- **Predicción de archivo**: Si el tipo de entrada de la aplicación de IA del servicio implementado es archivo, incluidas imágenes, audios y vídeos, puede agregar imágenes en la página de ficha **Prediction** para realizar pruebas de servicio.

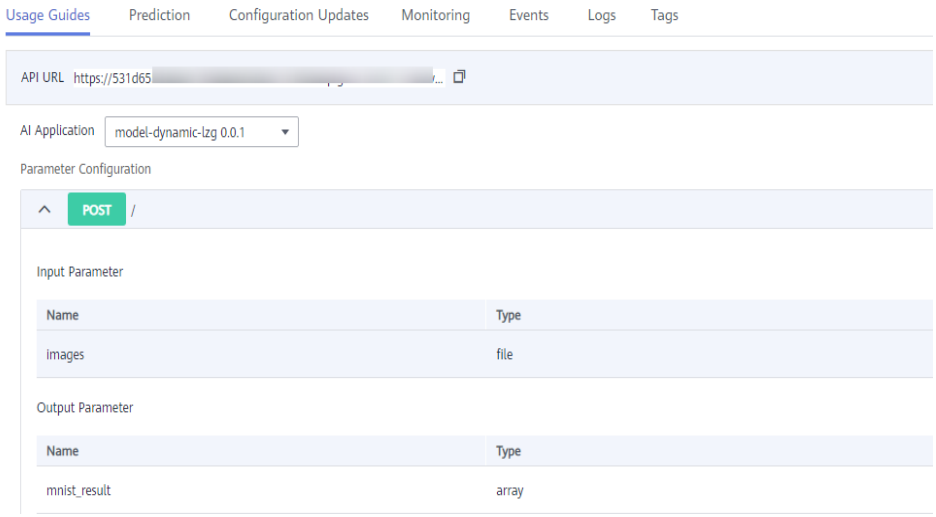
NOTA

- Si el tipo de entrada es imagen, el tamaño de una sola imagen debe ser inferior a 8 MB.
- El tamaño máximo del cuerpo de la solicitud para la predicción de texto de JSON es de 8 MB.
- Debido a la limitación de API Gateway, la duración de una sola predicción no puede superar los 40 s.
- Se admiten los siguientes tipos de imágenes: png, psd, jpg, jpeg, bmp, gif, webp, psd, svg y tiff.
- Si se utilizan variantes de Ascend durante el despliegue de servicio, las imágenes transparentes .png no se pueden predecir porque Ascend solo admite imágenes RGB-3.
- Esta función se utiliza para el comisionamiento. En la producción real, se recomienda invocar a las API. Puede seleccionar **Acceso autenticado mediante un token**, **Acceso autenticado con una AK/SK** o **Acceso autenticado mediante una aplicación** según el modo de autenticación.

Parámetros de entrada

Después de desplegar un servicio, obtenga los parámetros de entrada del servicio en la pestaña **Usage Guides** de la página de detalles del servicio.

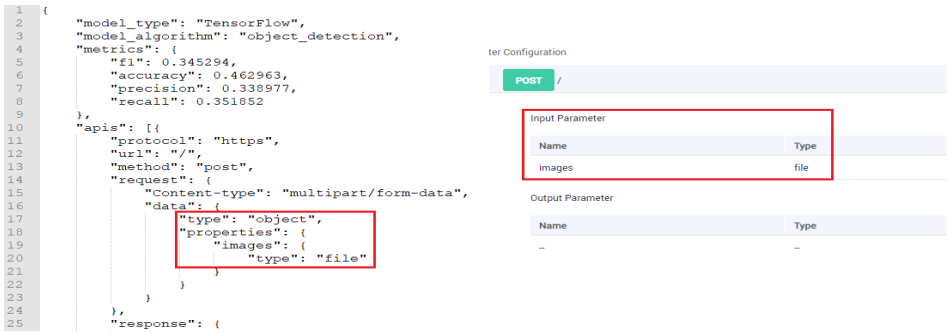
Figura 3-5 Consulta de la página de ficha Usage Guides



Los parámetros de entrada que se muestran en la pestaña **Usage Guides** varían según la fuente de aplicación de IA que seleccione.

- Si su metamodelo proviene de ExeML o de un algoritmo integrado, ModelArts define los parámetros de entrada y salida. Para obtener más información, consulte la página de ficha **Usage Guides**. En la página de la ficha **Prediction**, introduzca el texto o archivo JSON correspondiente para las pruebas de servicio.
- Si utiliza un metamodelo personalizado con el código de inferencia y el archivo de configuración compilados por usted (**Especificaciones para escribir el archivo de configuración del modelo**), ModelArts solo visualiza los datos en la ficha **Usage Guides**. La siguiente figura muestra la asignación entre los parámetros de entrada que se muestran en la página de ficha **Usage Guides** y el archivo de configuración.

  Figura 3-6 Asignación entre el archivo de configuración y las guías de uso



- Si el metamodelo se importa mediante una plantilla de modelo, los parámetros de entrada y salida varían con la plantilla. Para obtener más detalles, véase la descripción en **Introducción a las plantillas de modelo**.

Predicción de texto de JSON

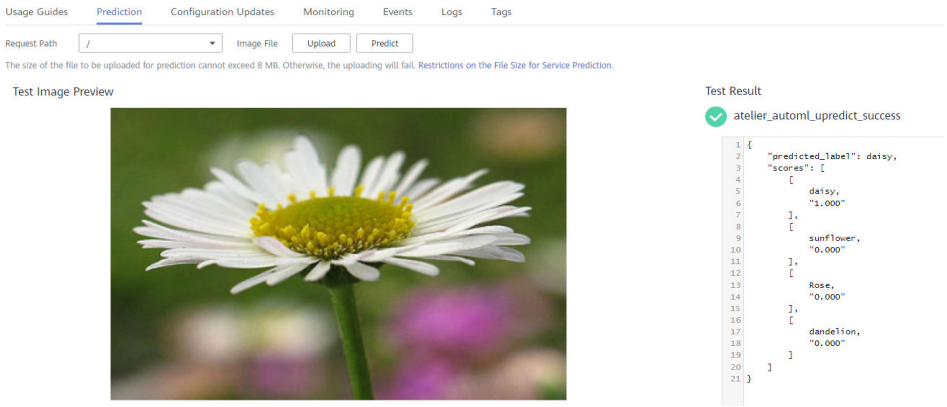
1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** > **Real-Time Services**.

2. En la página **Real-Time Services**, haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio. Ingrese el código de inferencia en la pestaña **Prediction** y haga clic en **Predict** para realizar la predicción.

Predicción de archivo

1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** > **Real-Time Services**.
2. En la página **Real-Time Services**, haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio.On the **Prediction** tab page, click **Upload** and select a test file. After the file is uploaded successfully, click **Predict** to perform a prediction test. In **Figura 3-7**, the label, position coordinates, and confidence score are displayed.

Figura 3-7 Predicción de imágenes



3.1.4 Acceso a los servicios en tiempo real

3.1.4.1 Acceso a un servicio en tiempo real

Si un servicio en tiempo real se encuentra en estado **Running**, el servicio en tiempo real se ha desplegado correctamente. Este servicio proporciona una API de RESTful estándar a la que puede invocar. Antes de integrar la API en el entorno de producción, comisione la API.

Por defecto, se accede a las API de servicios en tiempo real mediante HTTPS. También se admite el acceso basado en WebSocket. Si selecciona **WebSocket** durante el despliegue del servicio en tiempo real, la URL de la API es una dirección WebSocket después de desplegar el servicio. Para obtener más información, véase **Acceso a un servicio en tiempo real con WebSocket**.

ModelArts admite los siguientes métodos de autenticación para acceder a servicios en tiempo real (se utilizan como ejemplo las solicitudes de HTTPS):

- **Acceso autenticado con un token**
- **Acceso autenticado con una AK/SK**
- **Acceso autenticado con una aplicación**

ModelArts le permite invocar a las API para acceder a servicios en tiempo real de las siguientes maneras:

- **Acceso a un servicio en tiempo real (canal de red pública)**
- **Acceso a un servicio en tiempo real (canal de VPC de alta velocidad)**

Cuando se invoca a una API para acceder a un servicio en tiempo real, el tamaño del cuerpo de solicitud de predicción y el tiempo de predicción están sujetos a las siguientes limitaciones:

- El tamaño del cuerpo de una solicitud no puede superar los 12 MB. De lo contrario, la solicitud no se completará.
- Debido a la limitación de API Gateway, la duración de predicción de cada solicitud no excede los 40 segundos.

3.1.4.2 Modo de autenticación

3.1.4.2.1 Acceso autenticado mediante un token

Si un servicio en tiempo real se encuentra en estado **Running**, se ha desplegado correctamente. Este servicio proporciona una API de RESTful estándar para que los usuarios invoquen. Antes de integrar la API en el entorno de producción, comisione la API. Puede utilizar los siguientes métodos para enviar una solicitud de inferencia al servicio en tiempo real:

- **Método 1: Usar software basado en GUI para inferencia (Postman)**. (Se recomiendan Postman para Windows.)
- **Método 2: Ejecutar el comando cURL para enviar una solicitud de inferencia**. (Se recomiendan los comandos curl para Linux.)
- **Método 3: Usar Python para enviar una solicitud de inferencia**.
- **Método 4: Usar Java para enviar una solicitud de inferencia**.

Requisitos previos

Ha obtenido un token de usuario, ruta local al archivo de inferencia, URL del servicio en tiempo real y parámetros de entrada del servicio en tiempo real.

- Para obtener detalles sobre cómo obtener un token de usuario, consulte **Autenticación basada en token**. Las API de servicio en tiempo real generadas por ModelArts no admiten tokens cuyo alcance es dominio. Por lo tanto, es necesario obtener el token cuyo alcance es proyecto.
- La ruta local al archivo de inferencia puede ser una ruta absoluta (por ejemplo, **D:/test.png** para Windows y **/opt/data/test.png** para Linux) o una ruta relativa (por ejemplo, **./test.png**).
- Puede obtener URL del servicio y los parámetros de entrada de un servicio en tiempo real en la página de ficha Usage Guides de su página de detalles del servicio.

  La URL de API es la URL de servicio del servicio en tiempo real. Si se define una ruta para **apis** en el archivo de configuración del modelo, la URL debe ir seguida de la ruta definida por el usuario, por ejemplo, {*URL of the real-time service*}**/predictions/poetry**.

Figura 3-8 Obtención de la URL de la API y los parámetros de entrada de predicción de archivos de un servicio en tiempo real

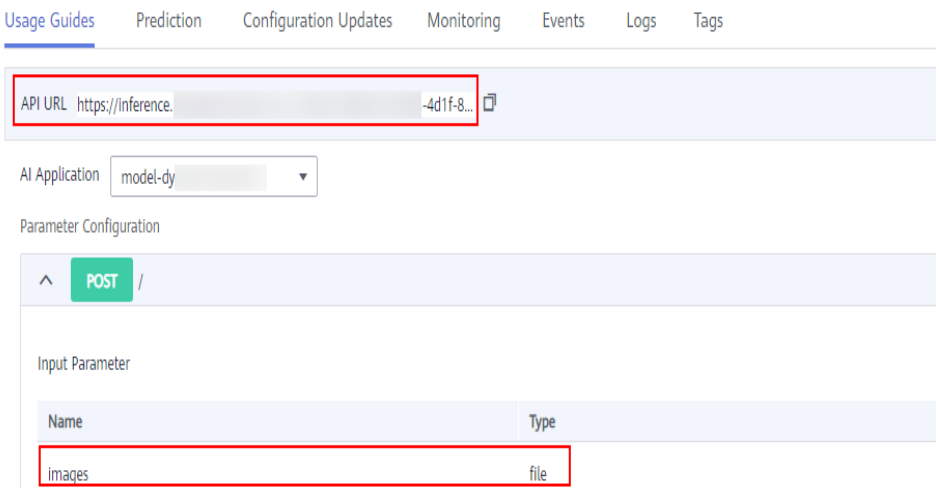
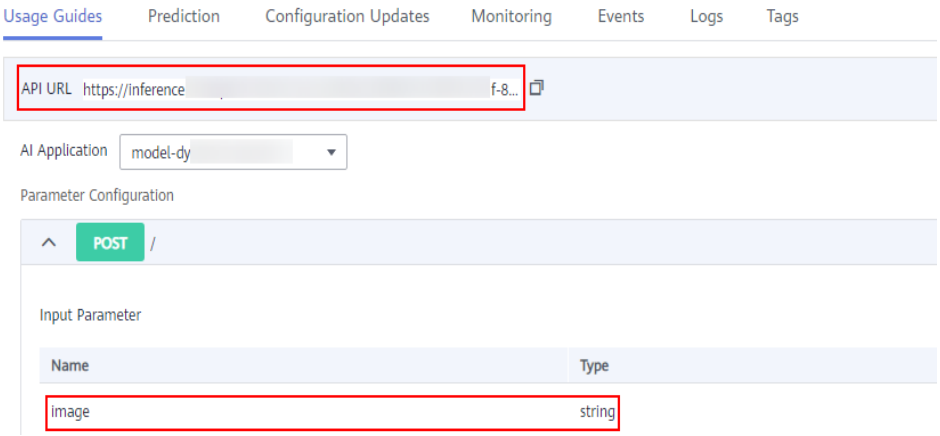


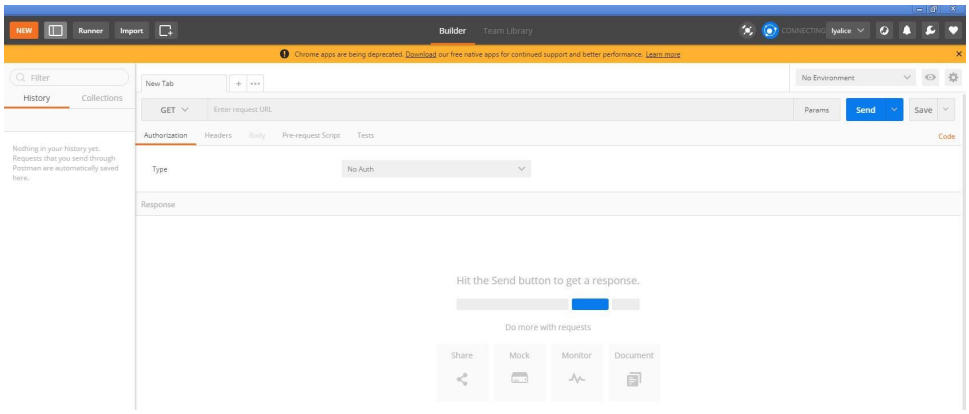
Figura 3-9 Obtención de la URL de la API y los parámetros de entrada de predicción de texto de un servicio en tiempo real



Método 1: Usar software basado en GUI para inferencia (Postman)

1. Descargue Postman e instalarlo, o instalar la extensión Postman Chrome. Alternativamente, utilice otro software que pueda enviar solicitudes POST. Se recomienda Postman 7.24.0.
2. Active Postman. **Figura 3-10** muestra la interfaz de Postman.

Figura 3-10 Interfaz de Postman

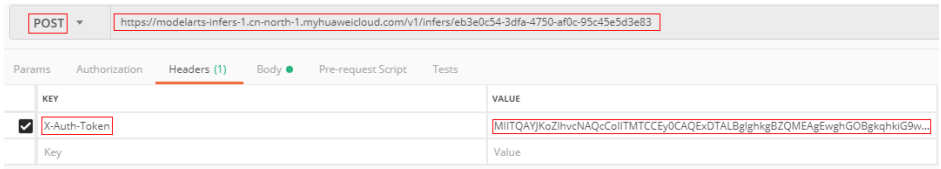


3. Establezca los parámetros en Postman. A continuación se utiliza la clasificación de imágenes como ejemplo.
   - Seleccione una tarea de POST y copie URL de la API en el cuadro de texto POST. En la página de pestaña **Headers**, establezca **Key** en **X-Auth-Token** y **Value** en el token de usuario.

     NOTA

     También puede utilizar las AK y SK para cifrar las solicitudes de invocaciones a la API. Para obtener más información, véase **Descripción de autenticación de sesión**.

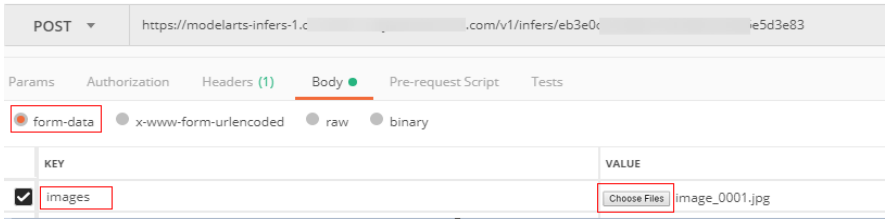
     Figura 3-11 Configuración de parámetros

   - En la página de la ficha **Body**, la entrada de archivo y la entrada de texto están disponibles.
     - **Ingreso de archivo**

       Seleccione **form-data**. Establezca **KEY** en el parámetro de entrada de la aplicación de IA, que debe ser el mismo que el parámetro de entrada del servicio en tiempo real. En este ejemplo, **KEY** es **images**. Establezca **VALUE** en una imagen que se va a inferir (solo se puede inferir una imagen). Consulte **Figura 3-12**.

       Figura 3-12 Configuración de parámetros en la página de ficha **Body**

     - **Ingreso de texto**

       Seleccione **raw** y luego **JSON(application/json)**. Introduzca el cuerpo de la solicitud en el cuadro de texto siguiente. Un cuerpo de solicitud de ejemplo es el siguiente:


```
{
  "meta": {
    "uuid": "10eb0091-887f-4839-9929-cbc884f1e20e"
  },
  "data": {
    "req_data": [
      {
        "sepal_length": 3,
        "sepal_width": 1,
        "petal_length": 2.2,
        "petal_width": 4
      }
    ]
  }
}
```

meta puede llevar un identificador único universal (UUID). Cuando se devuelve el resultado de la inferencia después de invocar a la API, se devuelve el UUID para rastrear la solicitud. Si no necesita esta función, deje **meta** en blanco. **data** contiene una matriz **req_data** para una o varias piezas de datos de entrada. Los parámetros de cada pieza de datos, como **sepal_length** y **sepal_width** en este ejemplo son determinados por la aplicación de IA.

4. Después de establecer los parámetros, haga clic en **send** para enviar la solicitud. El resultado se mostrará en **Response**.
   - Resultado de inferencia usando la entrada de archivo: **Figura 3-13** muestra un ejemplo. Los valores de campo en el resultado devuelto varían según la aplicación de IA.
   - Resultado de inferencia usando la entrada de texto: **Figura 3-14** muestra un ejemplo. El cuerpo de la solicitud contiene **meta** y **data**. Si la solicitud contiene **uuid**, **uuid** se devolverá en la respuesta. De lo contrario, el **uuid** se deja en blanco. **data** contiene una matriz **resp_data** para los resultados de inferencia de una o varias piezas de datos de entrada. Los parámetros de cada resultado se determinan mediante la aplicación de IA, por ejemplo, **sepal_length** y **predictresult** en este ejemplo.

Figura 3-13 Resultado de inferencia de archivo

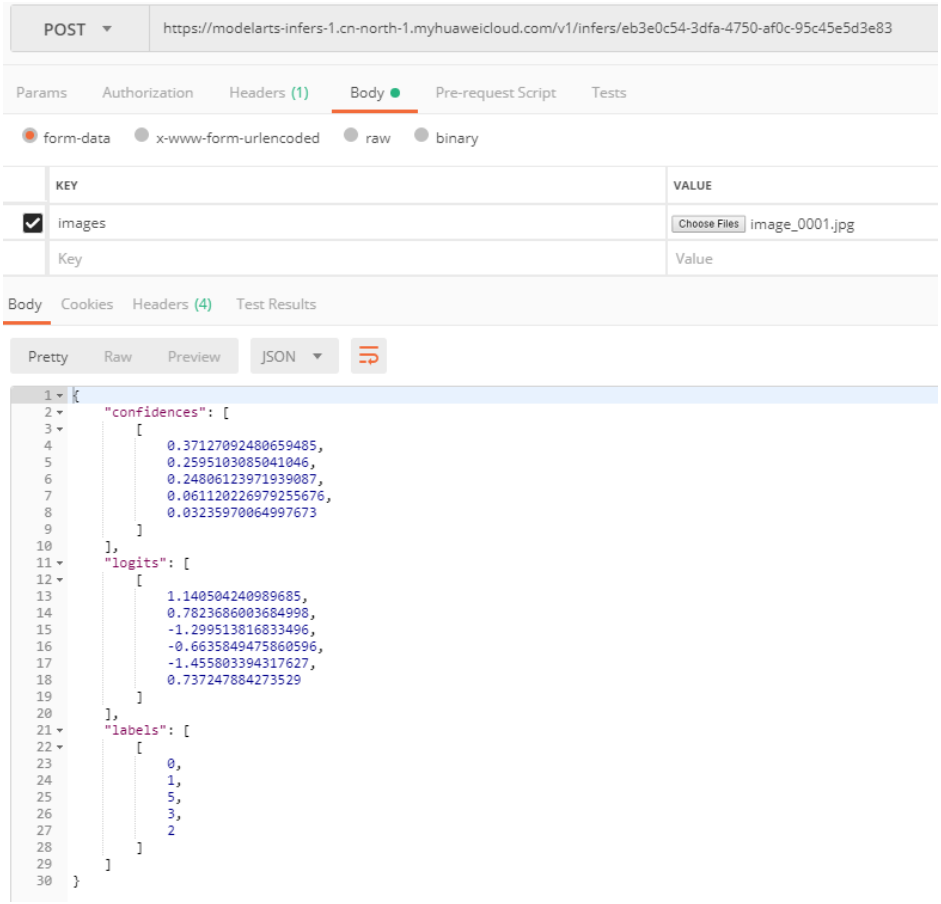
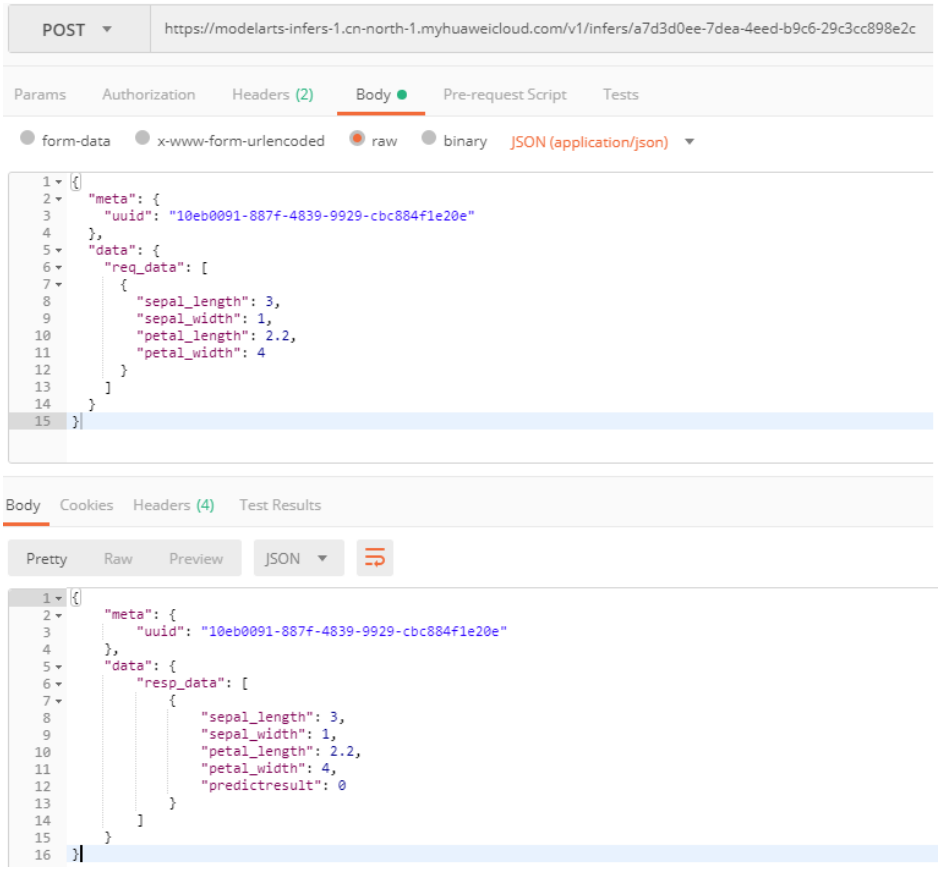


Figura 3-14 Resultado de inferencia de texto



Método 2: Ejecutar el comando cURL para enviar una solicitud de inferencia

El comando para enviar solicitudes de inferencia se puede introducir como un archivo o texto.

- Ingreso de archivo

  ```
  curl -kv -F 'images=@Image path' -H 'X-Auth-Token:Token value' -X POST Real-time service URL
  ```

  – **-k** indica que se puede acceder a los sitios web SSL sin utilizar un certificado de seguridad.

  – **-F** indica la entrada de archivo. En este ejemplo, el nombre del parámetro es **images**, que se pueden cambiar según sea necesario. La ruta de almacenamiento de imágenes sigue a **@**.

  – **-H** indica el encabezado de un comando de POST. **X-Auth-Token** es la clave de encabezado, que es fija. *Token value* indica el token de usuario.

  – **POST** es seguido por la URL de la API del servicio en tiempo real.

  El siguiente es un ejemplo del comando cURL para inferencia con entrada de archivo:

  ```
  curl -kv -F 'images=@/home/data/test.png' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -X POST https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
  ```

- Ingreso de texto

  ```
  curl -kv -d '{"data":{"req_data":[{"sepal_length":3,"sepal_width":1,"petal_length":2.2,"petal_width":4}]}}' -H 'X-Auth-Token:MIISkAY***80T9wHQ==' -H 'Content-type: application/json' -X POST https://modelarts-infers-1.xxx/v1/infers/eb3e0c54-3dfa-4750-af0c-95c45e5d3e83
  ```


-d indica la entrada de texto del cuerpo de la solicitud.

Método 3: Usar Python para enviar una solicitud de inferencia

1. Descargue el SDK de Python y configúrelo en la herramienta de desarrollo. Para obtener más información, consulte **Integración del SDK de Python para la firma de solicitudes de API**.
2. Cree un cuerpo de solicitud para inferencia.
   - **Ingreso de archivo**

     ```
     # coding=utf-8

     import requests

     if __name__ == '__main__':
         # Config url, token and file path.
         url = "URL of the real-time service"
         token = "User token"
         file_path = "Local path to the inference file"

         # Send request.
         headers = {
             'X-Auth-Token': token
         }
         files = {
             'images': open(file_path, 'rb')
         }
         resp = requests.post(url, headers=headers, files=files)

         # Print result.
         print(resp.status_code)
         print(resp.text)
     ```

     El nombre de **files** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de archivo. El parámetro de entrada **images** obtenido en **Requisitos previos** es un ejemplo.

   - **Ingreso de texto (JSON)**

     A continuación se muestra un ejemplo del cuerpo de la solicitud para leer el archivo de inferencia local y realizar la codificación Base64:

     ```
     # coding=utf-8

     import base64
     import requests

     if __name__ == '__main__':
         # Config url, token and file path
         url = "URL of the real-time service"
         token = "User token"
         file_path = "Local path to the inference file"
         with open(file_path, "rb") as file:
             base64_data = base64.b64encode(file.read()).decode("utf-8")

         # Set body,then send request
         headers = {
             'Content-Type': 'application/json',
             'X-Auth-Token': token
         }
         body = {
             'image': base64_data
         }
         resp = requests.post(url, headers=headers, json=body)

         # Print result
     ```



```
print(resp.status_code)
print(resp.text)
```

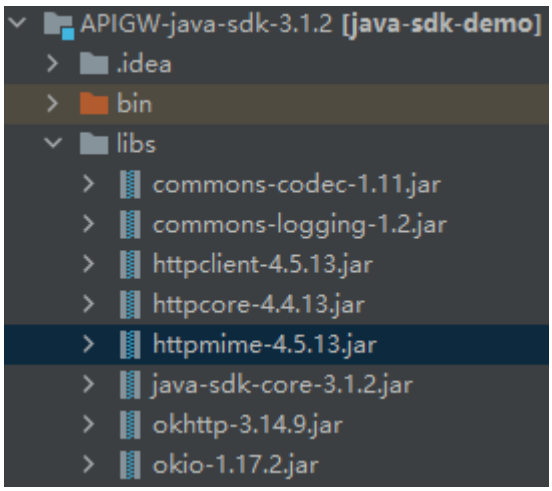
El nombre de **body** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de cadena. El parámetro de entrada **images** obtenido en **Requisitos previos** es un ejemplo. El valor de **base64_data** de **body** es del tipo de cadena.

Método 4: Usar Java para enviar una solicitud de inferencia

1. Descargue el SDK de Java y configúrelo en la herramienta de desarrollo. Para obtener más información, consulte **Integración del SDK de Java para la firma de solicitudes de API**.
2. (Opcional) Si la entrada de la solicitud de inferencia está en el formato de archivo, el proyecto Java depende del módulo httpmime.
   a. Agregue **httpmime-x.x.x.jar** a la carpeta **libs**. **Figura 3-15** muestra una biblioteca de dependencias Java completa.

      Se recomienda utilizar httpmime-x.x.x.jar 4.5 o una versión posterior. Descargue httpmime-x.x.x.jar desde **https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime**.

      Figura 3-15 Biblioteca de dependencias Java

   b. Después de agregar **httpmime-x.x.x.jar**, agregue la información httpmime al archivo **.classpath** del proyecto Java de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="con"
path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="src" path="src"/>
<classpathentry kind="lib" path="libs/commons-codec-1.11.jar"/>
<classpathentry kind="lib" path="libs/commons-logging-1.2.jar"/>
<classpathentry kind="lib" path="libs/httpclient-4.5.13.jar"/>
<classpathentry kind="lib" path="libs/httpcore-4.4.13.jar"/>
<classpathentry kind="lib" path="libs/httpmime-x.x.x.jar"/>
<classpathentry kind="lib" path="libs/java-sdk-core-3.1.2.jar"/>
<classpathentry kind="lib" path="libs/okhttp-3.14.9.jar"/>
<classpathentry kind="lib" path="libs/okio-1.17.2.jar"/>
<classpathentry kind="output" path="bin"/>
</classpath>
```

3. Cree un cuerpo de solicitud Java para inferencia.

– **Ingreso de archivo**

Un cuerpo de solicitud Java de ejemplo es el siguiente:

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import java.io.File;

public class MyTokenFile {

    public static void main(String[] args) {
        // Config url, token and filePath
        String url = "URL of the real-time service";
        String token = "User token";
        String filePath = "Local path to the inference file";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader("X-Auth-Token", token);

            // Add a body if you have specified the PUT or POST method.
Special characters, such as the double quotation mark ("), contained in
the body must be escaped.
            File file = new File(filePath);
            HttpEntity entity =
MultipartEntityBuilder.create().addBinaryBody("images",
file).setContentType(ContentType.MULTIPART_FORM_DATA).setCharset(Consts.U
TF_8).build();
            httpPost.setEntity(entity);

            // Send post
            CloseableHttpResponse response =
HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());

System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

El nombre **addBinaryBody** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de archivo. El archivo **images** obtenido en **Requisitos previos** se usa como ejemplo.

– **Ingreso de texto (JSON)**

A continuación se muestra un ejemplo del cuerpo de la solicitud para leer el archivo de inferencia local y realizar la codificación Base64:

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.HttpHeaders;
```



```
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyTokenTest {

    public static void main(String[] args) {
        // Config url, token and body
        String url = "URL of the real-time service";
        String token = "User token";
        String body = "{}";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/
json");
            httpPost.setHeader("X-Auth-Token", token);

            // Special characters, such as the double quotation mark
("), contained in the body must be escaped.
            httpPost.setEntity(new StringEntity(body));

            // Send post.
            CloseableHttpResponse response =
HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());

System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

body está determinado por el formato de texto. JSON se utiliza como ejemplo.

3.1.4.2.2 Acceso autenticado con una AK/SK

Si un servicio en tiempo real está en el estado **Running**, el servicio en tiempo real se ha implementado correctamente. Este servicio proporciona una API de RESTful estándar para que los usuarios invoquen. Los usuarios pueden invocar a la API mediante autenticación basada en AK/SK.

Cuando se utiliza la autenticación basada en AK/SK, puede utilizar el SDK de APIG o el SDK de ModelArts para acceder a las API. Para obtener más información, véase **Descripción de autenticación de sesión**. Esta sección describe cómo utilizar el SDK de APIG para acceder a un servicio en tiempo real. El proceso es el siguiente:

1. **Obtención de un par de AK/SK**
2. **Obtención de información sobre un servicio en tiempo real**
3. Envío de una solicitud de inferencia
   - **Método 1: Usar Python para enviar una solicitud de inferencia**
   - **Método 2: Usar Java para enviar una solicitud de inferencia**

NOTA

1. La autenticación basada en AK/SK admite solicitudes de API con un cuerpo de no más de 12 MB. Para las solicitudes de API con un cuerpo más grande, se recomienda la autenticación basada en tokens.
2. La hora local en el cliente debe sincronizarse con el servidor de reloj para evitar un desplazamiento grande en el valor del encabezado de solicitud **X-Sdk-Date**. API Gateway comprueba el formato de hora y compara la hora con la hora en que API Gateway recibe la solicitud. Si la diferencia de tiempo supera los 15 minutos, API Gateway rechazará la solicitud.

Obtención de un par de AK/SK

Si un par de AK/SK ya está disponible, omita este paso. Encuentre el archivo AK/SK descargado, que normalmente se llama **credentials.csv**.

Como se muestra en la siguiente figura, el archivo contiene el nombre de usuario, AK y SK.

Figura 3-16 Contenido del archivo credential.csv

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

Realice las siguientes operaciones para generar un par de AK/SK:

1. Regístrese e inicie sesión en la consola de gestión.
2. Haga clic en el nombre de usuario y elija **My Credentials** en la lista desplegable.
3. En la página **My Credentials**, elija **Access Keys** en el panel de navegación.
4. Haga clic en **Create Access Key**. Aparece el cuadro de diálogo **Identity Verification**.
5. Complete la autenticación de identidad como se le indique, descargue la clave de acceso y manténgala segura.

Obtención de información sobre un servicio en tiempo real

Al invocar a una API, es necesario obtener la dirección de API y los parámetros de entrada del servicio en tiempo real. El procedimiento es el siguiente:

1. Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación izquierdo, elija **Service Deployment** > **Real-Time Services**. De forma predeterminada, el sistema cambia a la página **Real-Time Services**.
2. Haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio.
3. En la página de detalles de un servicio en tiempo real, obtenga la dirección API y los parámetros de entrada del servicio.

   La URL de API es la URL de servicio del servicio en tiempo real. Si se define una ruta para **apis** en el archivo de configuración del modelo, la URL debe ir seguida de la ruta definida por el usuario, por ejemplo, {*URL of the real-time service*}**/predictions/poetry**.

Figura 3-17 Obtención de la URL de la API y los parámetros de entrada de predicción de archivos de un servicio en tiempo real

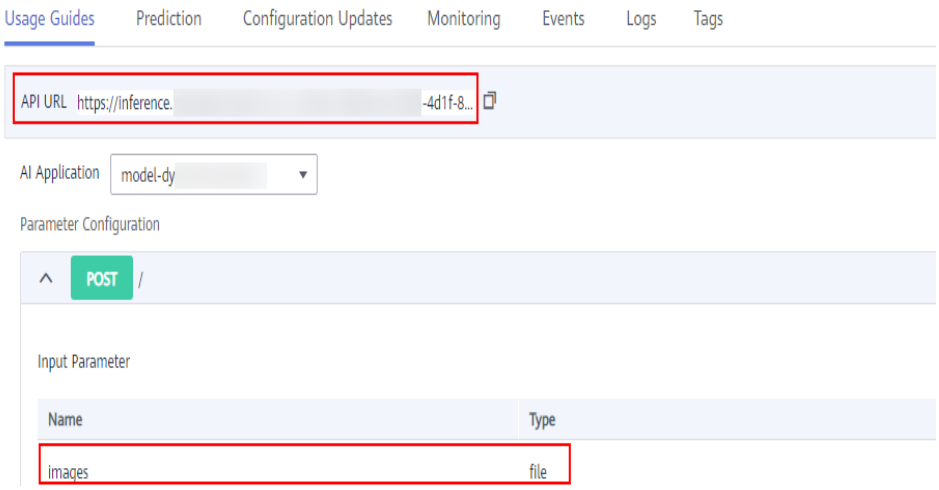
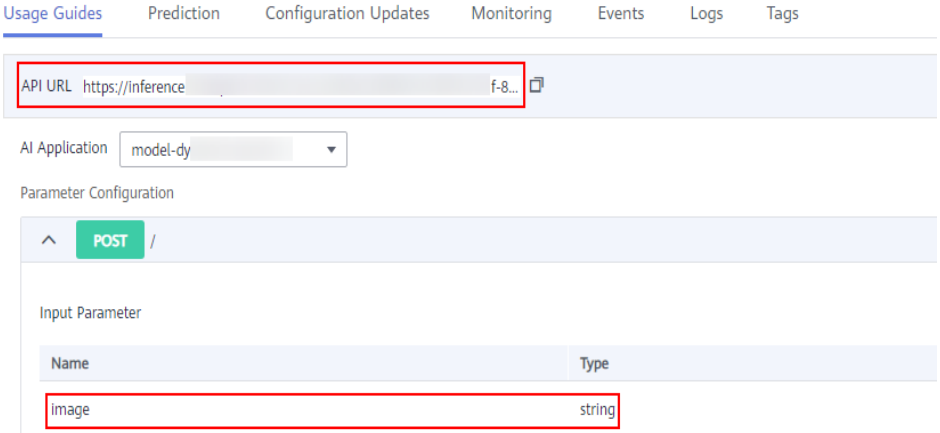


Figura 3-18 Obtención de la URL de la API y los parámetros de entrada de predicción de texto de un servicio en tiempo real



Método 1: Usar Python para enviar una solicitud de inferencia

1. Descargue el SDK de Python y configúrelo en la herramienta de desarrollo. Para obtener más información, consulte **Integración del SDK de Python para la firma de solicitudes de API**.
2. Cree un cuerpo de solicitud para inferencia.
   - **Ingreso de archivo**

```
# coding=utf-8

import requests
import os
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "URL of the real-time service"
    # Hardcoded or plaintext AK/SK is risky. For security, encrypt your
AK/SK and store them in the configuration file or environment variables.
    # In this example, the AK/SK are stored in environment variables for
```



```
identity authentication. Before running this example, set environment
variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    file_path = "Local path to the inference file"

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {"x-sdk-content-sha256": "UNSIGNED-PAYLOAD"}
    request = signer.HttpRequest(method, url, headers)

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = ak
    sig.Secret = sk
    sig.Sign(request)

    # Send request
    files = {'images': open(file_path, 'rb')}
    resp = requests.request(request.method, request.scheme + "://" +
request.host + request.uri, headers=request.headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

file_path es la ruta de acceso local al archivo de inferencia. La ruta puede ser una ruta absoluta (por ejemplo, **D:/test.png** para Windows y **/opt/data/test.png** para Linux) o una ruta relativa (por ejemplo, **./test.png**).

Formato del cuerpo de la solicitud de **files**: files = {"Request parameter": ("Load path", File content, "File type")}. Para obtener detalles sobre los parámetros de **files**, véase **Tabla 3-9**.

Tabla 3-9 Parámetros de **files**

Parámetro	Obligatorio	Descripción
Request parameter	Sí	Introduzca el nombre del parámetro del servicio en tiempo real.
Load path	No	Ruta de acceso en la que se almacena el archivo.
File content	Sí	Contenido del archivo que se va a cargar.
File type	No	Tipo del archivo que se va a cargar, que puede ser una de las siguientes opciones: ● **txt**: texto/plano ● **jpg/jpeg**: imagen/jpeg ● **png**: imagen/png

– **Ingreso de texto (JSON)**

A continuación se muestra un ejemplo del cuerpo de la solicitud para leer el archivo de inferencia local y realizar la codificación Base64:

```
# coding=utf-8

import base64
import json
```



```
import os
import requests
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "URL of the real-time service"
    # Hardcoded or plaintext AK/SK is risky. For security, encrypt your
AK/SK and store them in the configuration file or environment variables.
    # In this example, the AK/SK are stored in environment variables for
identity authentication. Before running this example, set environment
variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    file_path = "Local path to the inference file"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {
        'Content-Type': 'application/json'
    }
    body = {
        'image': base64_data
    }
    request = signer.HttpRequest(method, url, headers, json.dumps(body))

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = ak
    sig.Secret = sk
    sig.Sign(request)

    # Send request
    resp = requests.request(request.method, request.scheme + "://" +
request.host + request.uri, headers=request.headers, data=request.body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

El nombre de **body** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de cadena. **image** se utiliza como ejemplo. El valor de **base64_data** de **body** es del tipo de cadena.

Método 2: Usar Java para enviar una solicitud de inferencia

1. Descargue el SDK de Java y configúrelo en la herramienta de desarrollo.
2. Cree un cuerpo de solicitud Java para inferencia.

   En el SDK de Java de APIG, **request.setBody()** solo puede ser una string. Por lo tanto, solo se admiten solicitudes de inferencia de texto. Si hay un archivo de entrada, conviértalo en texto usando Base64.

   – **Ingreso de archivo**

   El siguiente es un ejemplo del cuerpo de la solicitud (JSON) para leer el archivo de inferencia local y realizar la codificación de Base64.

```
package com.apig.sdk.demo;
import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
```



```
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
public class MyAkSkTest2 {
    public static void main(String[] args) {
        String url = "URL of the real-time service";
       // Hard-coded or plaintext AK/SK is risky. For security, encrypt
your AK/SK and store them in the configuration file or environment
variables.
       // In this example, the AK/SK are stored in environment variables
for identity authentication. Before running this example, set
environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");
        String filePath = "Local path to the inference file";
        try {
            // Create request
            Request request = new Request();
            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);
            // Specify a request method, such as GET, PUT, POST, DELETE,
HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);
            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/
json");
            // Set a request URL in the format of https://{Endpoint}/
{URI}.
            request.setUrl(url);
            // build your json body
            String body = "{\"image\":\"" + getBase64FromFile(filePath)
+ "\"}";
            // Special characters, such as the double quotation mark
("), contained in the body must be escaped.
            request.setBody(body);
            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);
            // Send request.
            CloseableHttpResponse response =
HttpClients.createDefault().execute(signedRequest);
            // Print result
            System.out.println(response.getStatusLine().getStatusCode());

System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * Convert the file into a byte array and Base64 encode it
     * @return
     */
    private static String getBase64FromFile(String filePath) {
        // Convert the file into a byte array
        InputStream in = null;
        byte[] data = null;
        try {
            in = new FileInputStream(filePath);
            data = new byte[in.available()];
            in.read(data);
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Base64 encode
```



```
        return new String(Base64.encodeBase64(data));
    }
}
```

⚠ ATENCIÓN

Si se utiliza la codificación de Base64, debe agregar el código para decodificar el cuerpo de la solicitud al código de inferencia del modelo.

– **Ingreso de texto (JSON)**

```
// Package name of the demo.
package com.apig.sdk.demo;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAkSkTest {

    public static void main(String[] args) {
        String url = "URL of the real-time service";
       // Hard-coded or plaintext AK/SK is risky. For security, encrypt
your AK/SK and store them in the configuration file or environment
variables.
       // In this example, the AK/SK are stored in environment variables
for identity authentication. Before running this example, set
environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK.
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
       String sk = System.getenv("HUAWEICLOUD_SDK_SK");

        try {
            // Create request
            Request request = new Request();

            // Set the AK/SK to sign and authenticate the request.
            request.setKey(ak);
            request.setSecret(sk);

            // Specify a request method, such as GET, PUT, POST, DELETE,
HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);

            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/
json");

            // Set a request URL in the format of https://{Endpoint}/
{URI}.
            request.setUrl(url);

            // Special characters, such as the double quotation mark
("), contained in the body must be escaped.
            String body = "{}";
            request.setBody(body);

            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);

            // Send request.
            CloseableHttpResponse response =
HttpClients.createDefault().execute(signedRequest);
```



```
            // Print result
            System.out.println(response.getStatusLine().getStatusCode());

System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

body está determinado por el formato de texto. JSON se utiliza como ejemplo.

3.1.4.2.3 Acceso autenticado mediante una aplicación

Puede habilitar la autenticación de aplicaciones al desplegar un servicio en tiempo real. ModelArts registra una API que admite la autenticación de aplicaciones para el servicio. Después de que esta API esté autorizada a una aplicación, puede invocar a esta API usando AppKey/AppSecret o AppCode de la aplicación.

El proceso de autenticación de aplicaciones para un servicio en tiempo real es el siguiente:

1. **Habilitación de la autenticación de aplicaciones**: Activar autenticación de aplicaciones. Puede seleccionar una aplicación existente o crear una aplicación.
2. **Gestión de la autorización de servicios en tiempo real**: Gestionar la aplicación creada, lo que incluye consultar, restablecer o eliminar la aplicación, vincular o desvincular servicios en tiempo real para la aplicación y obtener la AppKey/AppSecret o el AppCode.
3. **Autenticación de aplicaciones**: Se requiere la autenticación para invocar a una API que admita la autenticación de aplicaciones. Se proporcionan dos modos de autenticación (AppKey+AppSecret o AppCode). Puede seleccionar cualquiera de ellos.
4. Envío de una solicitud de inferencia
   - **Método 1: Utilice Python para enviar una solicitud de inferencia a través de la autenticación basada en AppKey/AppSecret**
   - **Método 2: Utilice Java para enviar una solicitud de inferencia a través de la autenticación basada en AppKey/AppSecret**
   - **Método 3: Utilice Python para enviar una solicitud de inferencia a través de la autenticación basada en AppCode**
   - **Método 4: Utilice Java para enviar una solicitud de inferencia a través de la autenticación basada en AppCode**

Requisitos previos

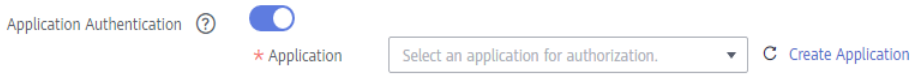
- Se han preparado los datos. Específicamente, ha creado una aplicación de IA en el estado **Normal** de ModelArts.
- La cuenta no está en mora para garantizar los recursos disponibles para la ejecución del servicio.
- Se ha obtenido la ruta local al archivo de inferencia. La ruta puede ser una ruta absoluta (por ejemplo, **D:/test.png** para Windows y **/opt/data/test.png** para Linux) o una ruta relativa (por ejemplo, **./test.png**).

Habilitación de la autenticación de aplicaciones

Al desplegar una aplicación de IA como servicio en tiempo real, puede habilitar la autenticación de aplicaciones. También puede modificar un servicio en tiempo real desplegado para admitir la autenticación de aplicaciones.

1. Inicie sesión en la consola de gestión de ModelArts y elija **Service Deployment > Real-Time Services**.
2. Habilite la autenticación de aplicaciones.
   - Al desplegar un modelo como servicio en tiempo real, configure los parámetros necesarios y habilite la autenticación de aplicaciones en la página **Deploy**.
   - Para un servicio en tiempo real desplegado, vaya a la página **Real-Time Services** y haga clic en **Modify** en la columna **Operation** del servicio. En la página de modificación de servicio que se muestra, habilite la autenticación de aplicaciones.

     Figura 3-19 Habilitación de la autenticación de aplicaciones

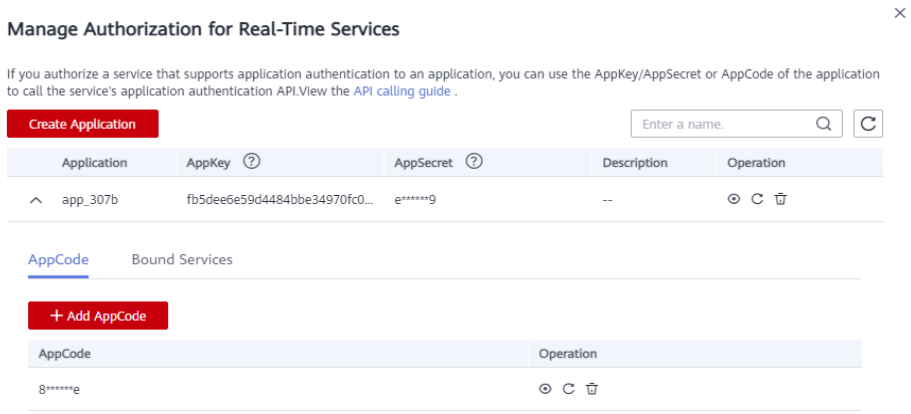
     

3. Seleccione una solicitud de autorización en la lista desplegable. Si no hay ninguna aplicación disponible, cree una de la siguiente manera:
   - Haga clic en **Create Application** a la derecha, introduzca el nombre y la descripción de la aplicación y haga clic en **OK**. De forma predeterminada, el nombre de la aplicación comienza con **app_**. Puede cambiar el nombre de la aplicación.
   - En la página **Service Deployment** > **Real-Time Services**, haga clic en **Authorize**. En la página **Manage Authorization of Real-Time Services**, haga clic en **Create Application**. Para más detalles, véase **Gestión de la autorización de servicios en tiempo real**.
4. Después de habilitar la autenticación de aplicaciones, autorice un servicio que admita la autenticación de aplicaciones en la aplicación. A continuación, puede usar AppKey/AppSecret o AppCode generados para invocar a la API del servicio que admite la autenticación de aplicaciones.

Gestión de la autorización de servicios en tiempo real

Si desea utilizar la autenticación de aplicaciones, es una buena práctica crear una aplicación en la página de gestión de autorizaciones antes de desplegar un servicio en tiempo real. En el panel de navegación, seleccione **Service Deployment** > **Real-Time Services**. En la página **Real-Time Services**, haga clic en **Authorize**. Aparecerá el cuadro de diálogo **Manage Authorization of Real-Time Services**. Desde allí, puede crear y gestionar aplicaciones, incluida la visualización, el restablecimiento y la eliminación de aplicaciones, la desvinculación de servicios en tiempo real de las aplicaciones y la obtención de AppKey/AppSecret o AppCode.

Figura 3-20 Gestión de la autorización para servicios en tiempo real



- **Crear una aplicación**

  Haga clic en **Create Application**, introduzca el nombre y la descripción de la aplicación y haga clic en **OK**. Por defecto, el nombre de la aplicación comienza con **app_**. Puede cambiar el nombre de la aplicación.

- **Consultar, restablecer o eliminar una aplicación**

  Consultar, restablecer o eliminar una aplicación haciendo clic en el icono correspondiente en la columna **Operation** de la aplicación. Después de crear una aplicación, la AppKey y la AppSecret se generan automáticamente para la autenticación de la aplicación.

- **Desvincular un servicio**

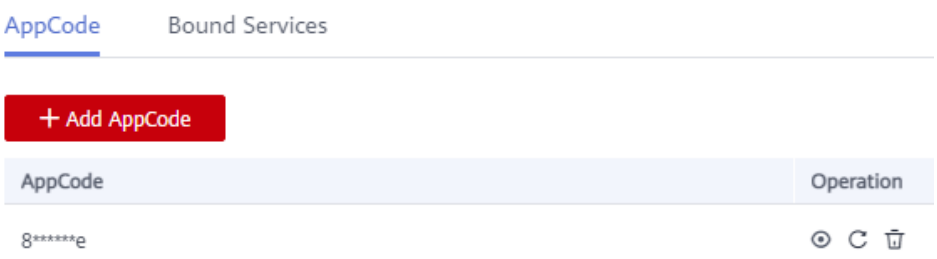
  Delante del nombre de la aplicación de destino, haga clic en para ver los servicios en tiempo real asociados a la aplicación. Haga clic en **Unbind** en la columna **Operation** para cancelar la asociación. Entonces, esta API no se puede invocar.

- **Obtenerlas AppKey/AppSecret o AppCode**

  La autenticación de la aplicación es necesaria para las invocaciones a la API. Las AppKey y AppSecret se generan automáticamente durante la creación de la aplicación.

  Haga clic en en la columna **Operation** de la aplicación en el cuadro de diálogo **Manage Authorization of Real-Time Services** para ver la AppSecret completa. Haga clic en delante del nombre de la aplicación para expandir la lista desplegable. Haga clic en **+Add AppCode** para generar un AppCode automáticamente. Luego, haga clic en en la columna **Operation** para ver el AppCode completo.

Figura 3-21 Agregar el AppCode



Autenticación de aplicaciones

Cuando un servicio en tiempo real que admite la autenticación de aplicaciones está en estado **Running**, se puede invocar a la API del servicio. Antes de invocar a la API, realice la autenticación de la aplicación.

Cuando se utiliza la autenticación de aplicaciones y se habilita el modo de autenticación simplificada, puede usar las AppKey/AppSecret para la firma y la verificación, o AppCode para la autenticación simplificada de solicitudes de API. (ModelArts utiliza la autenticación simplificada de forma predeterminada.) Se recomienda la autenticación basada en AppKey/AppSecret porque es más segura que la autenticación basada en AppCode.

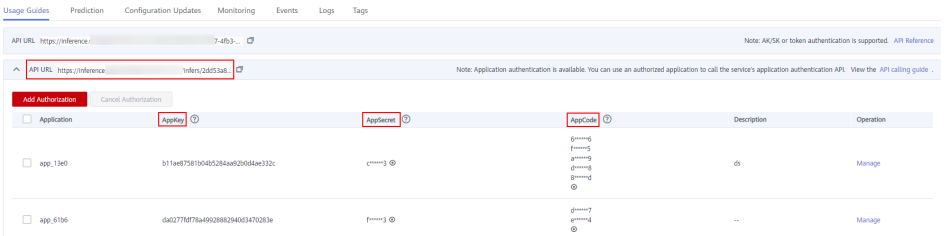
- **AppKey/AppSecret-based authentication**: Las AppKey y AppSecret se utilizan para cifrar una solicitud, identificar al remitente y evitar que se modifique la solicitud. Cuando utilice la autenticación basada en AppKey/AppSecret, utilice un SDK de firma dedicado para firmar solicitudes.
  - AppKey: ID de clave de acceso de la aplicación, que es un identificador único utilizado junto con una clave de acceso secreta para firmar solicitudes criptográficamente.
  - AppSecret: clave de acceso secreta de la aplicación, utilizada junto con el ID de clave de acceso para cifrar la solicitud, identificar al remitente y evitar que la solicitud sea templada.

  AppKeys se puede utilizar para la autenticación simplificada. Cuando se invoca a una API, se agrega el parámetro **apikey** (valor:**AppKey**) al encabezado de solicitud de HTTP para acelerar la autenticación.
- **AppCode-based authentication**: Requests are authenticated using AppCodes.

  En la autenticación basada en AppCode, se agrega el parámetro **X-Apig-AppCode** (valor: **AppCode**)al encabezado de solicitud de HTTP cuando se llama a una API. No es necesario firmar el contenido de la solicitud. El gateway de API solo verifica el AppCode y logra una respuesta rápida.

Puede obtener la API, AppKey/AppSecret y AppCode desde la pestaña **Guías de uso** en la página de detalles del servicio (véase **Figura 3-22**) o desde la página de gestión de autorizaciones de servicio en tiempo real (véase **Figura 3-20**). Utilice la URL de la API en el cuadro rojo de la siguiente figura. Si se define una ruta para **apis** en el archivo de configuración del modelo, la URL debe ir seguida de la ruta definida por el usuario, por ejemplo, {*URL of the real-time service*}**/predictions/poetry**.

Figura 3-22 Obtención de la dirección de API



Método 1: Utilice Python para enviar una solicitud de inferencia a través de la autenticación basada en AppKey/AppSecret

1. Descargue el SDK de Python y configúrelo en la herramienta de desarrollo. Para obtener más información, consulte **Integración del SDK de Python para la firma de solicitudes de API**.
2. Cree un cuerpo de solicitud para inferencia.
   - **Ingreso de archivo**

```
# coding=utf-8

import requests
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "URL of the real-time service"
    app_key = "AppKey"
    app_secret = "AppSecret"
    file_path = "Local path to the inference file"

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {"x-sdk-content-sha256": "UNSIGNED-PAYLOAD"}
    request = signer.HttpRequest(method, url, headers)

    # Create sign, set the AK/SK to sign and authenticate the request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
    sig.Sign(request)

    # Send request
    files = {'images': open(file_path, 'rb')}
    resp = requests.request(request.method, request.scheme + "://" +
request.host + request.uri, headers=request.headers, files=files)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

Formato del cuerpo de la solicitud de **files**: files = {"Request parameter": ("Load path", File content, "File type")}. Para obtener detalles sobre los parámetros de **files**, véase **Tabla 3-10**.

Tabla 3-10 Parámetros de **files**

Parámetro	Obligatorio	Descripción
Request parameter	Sí	Nombre del parámetro del servicio en tiempo real.
Load path	No	Ruta de acceso en la que se almacena el archivo.
File content	Sí	Contenido del archivo que se va a cargar.

Parámetro	Obligatorio	Descripción
File type	No	Tipo del archivo que se va a cargar, que puede ser una de las siguientes opciones: ● **txt**: texto/plano ● **jpg/jpeg**: imagen/jpeg ● **png**: imagen/png

– **Ingreso de texto (JSON)**

A continuación se muestra un ejemplo del cuerpo de la solicitud para leer el archivo de inferencia local y realizar la codificación Base64:

```
# coding=utf-8

import base64
import json
import requests
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "URL of the real-time service"
    app_key = "AppKey"
    app_secret = "AppSecret"
    file_path = "Local path to the inference file"
    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # Create request, set method, url, headers and body.
    method = 'POST'
    headers = {
        'Content-Type': 'application/json'
    }
    body = {
        'image': base64_data
    }
    request = signer.HttpRequest(method, url, headers, json.dumps(body))

    # Create sign, set the AppKey&AppSecret to sign and authenticate the
request.
    sig = signer.Signer()
    sig.Key = app_key
    sig.Secret = app_secret
    sig.Sign(request)

    # Send request
    resp = requests.request(request.method, request.scheme + "://" +
request.host + request.uri, headers=request.headers, data=request.body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

El nombre de **body** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de cadena. **image** se utiliza como ejemplo. El valor de **base64_data** de **body** es del tipo de cadena.

Método 2: Utilice Java para enviar una solicitud de inferencia a través de la autenticación basada en AppKey/AppSecret

1. Descargue el SDK de Java y configúrelo en la herramienta de desarrollo. Para obtener más información, consulte **Integración del SDK de Java para la firma de solicitudes de API**.
2. Cree un cuerpo de solicitud Java para inferencia.

   En el SDK de Java de APIG, **request.setBody()** solo puede ser una string. Por lo tanto, solo se admiten solicitudes de inferencia de texto.

   A continuación se muestra un ejemplo del cuerpo de la solicitud (JSON) para leer el archivo de inferencia local y realizar la codificación Base64:

```
// Package name of the demo.
package com.apig.sdk.demo;

import com.cloud.apigateway.sdk.utils.Client;
import com.cloud.apigateway.sdk.utils.Request;
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAkSkTest {

    public static void main(String[] args) {
        String url = "URL of the real-time service";
        String appKey = "AppKey";
        String appSecret = "AppSecret";
        String body = "{}";

        try {
            // Create request
            Request request = new Request();

            // Set the AK/AppSecret to sign and authenticate the request.
            request.setKey(appKey);
            request.setSecret(appSecret);

            // Specify a request method, such as GET, PUT, POST, DELETE,
HEAD, and PATCH.
            request.setMethod(HttpPost.METHOD_NAME);

            // Add header parameters
            request.addHeader(HttpHeaders.CONTENT_TYPE, "application/json");

            // Set a request URL in the format of https://{Endpoint}/{URI}.
            request.setUrl(url);

            // Special characters, such as the double quotation mark ("),
contained in the body must be escaped.
            request.setBody(body);

            // Sign the request.
            HttpRequestBase signedRequest = Client.sign(request);

            // Send request.
            CloseableHttpResponse response =
HttpClients.createDefault().execute(signedRequest);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());
            System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
```



```
        }
    }
}
```

body está determinado por el formato de texto. JSON se utiliza como ejemplo.

Método 3: Utilice Python para enviar una solicitud de inferencia a través de la autenticación basada en AppCode

1. Descargue el SDK de Python y configúrelo en la herramienta de desarrollo. Para obtener más información, consulte **Integración del SDK de Python para la firma de solicitudes de API**.
2. Cree un cuerpo de solicitud para inferencia.
   - **Ingreso de archivo**

     ```
     # coding=utf-8

     import requests

     if __name__ == '__main__':
         # Config url, app code and file path.
         url = "URL of the real-time service"
         app_code = "AppCode"
         file_path = "Local path to the inference file"

         # Send request.
         headers = {
             'X-Apig-AppCode': app_code
         }
         files = {
             'images': open(file_path, 'rb')
         }
         resp = requests.post(url, headers=headers, files=files)

         # Print result
         print(resp.status_code)
         print(resp.text)
     ```

     El nombre de **files** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de archivo. En este ejemplo, se utilizan **images**.
   - **Ingreso de texto (JSON)**

     A continuación se muestra un ejemplo del cuerpo de la solicitud para leer el archivo de inferencia local y realizar la codificación Base64:

     ```
     # coding=utf-8

     import base64
     import requests

     if __name__ == '__main__':
         # Config url, app code and request body.
         url = "URL of the real-time service"
         app_code = "AppCode"
         file_path = "Local path to the inference file"
         with open(file_path, "rb") as file:
             base64_data = base64.b64encode(file.read()).decode("utf-8")

         # Send request
         headers = {
             'Content-Type': 'application/json',
             'X-Apig-AppCode': app_code
         }
         body = {
             'image': base64_data
         }
     ```



```
    resp = requests.post(url, headers=headers, json=body)

    # Print result
    print(resp.status_code)
    print(resp.text)
```

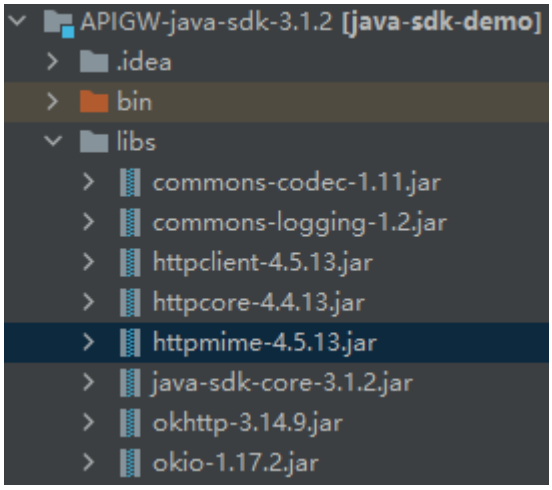
El nombre de **body** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de cadena. **image** se utiliza como ejemplo. El valor de **base64_data** de **body** es del tipo de cadena.

Método 4: Utilice Java para enviar una solicitud de inferencia a través de la autenticación basada en AppCode

1. Descargue el SDK de Java y configúrelo en la herramienta de desarrollo. Para obtener más información, consulte **Integración del SDK de Java para la firma de solicitudes de API**.
2. (Opcional) Si la entrada de la solicitud de inferencia está en el formato de archivo, el proyecto Java depende del módulo httpmime.
   a. Agregue **httpmime-x.x.x.jar** a la carpeta **libs**. **Figura 3-23** muestra una biblioteca de dependencias Java completa.

      Se recomienda utilizar httpmime-x.x.x.jar 4.5 o una versión posterior. Descargue httpmime-x.x.x.jar desde **https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime**.

      Figura 3-23 Biblioteca de dependencias Java

   b. Después de agregar **httpmime-x.x.x.jar**, agregue la información httpmime al archivo **.classpath** del proyecto Java de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="con"
path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="src" path="src"/>
<classpathentry kind="lib" path="libs/commons-codec-1.11.jar"/>
<classpathentry kind="lib" path="libs/commons-logging-1.2.jar"/>
<classpathentry kind="lib" path="libs/httpclient-4.5.13.jar"/>
<classpathentry kind="lib" path="libs/httpcore-4.4.13.jar"/>
<classpathentry kind="lib" path="libs/httpmime-x.x.x.jar"/>
<classpathentry kind="lib" path="libs/java-sdk-core-3.1.2.jar"/>
<classpathentry kind="lib" path="libs/okhttp-3.14.9.jar"/>
<classpathentry kind="lib" path="libs/okio-1.17.2.jar"/>
```



```
<classpathentry kind="output" path="bin"/>
</classpath>
```

3. Cree un cuerpo de solicitud Java para inferencia.

   – **Ingreso de archivo**

   Un cuerpo de solicitud Java de ejemplo es el siguiente:

```
// Package name of the demo.
package com.apig.sdk.demo;

import org.apache.http.Consts;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ContentType;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

import java.io.File;

public class MyAppCodeFile {

    public static void main(String[] args) {
        String url = "URL of the real-time service";
        String appCode = "AppCode";
        String filePath = "Local path to the inference file";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader("X-Apig-AppCode", appCode);

            // Special characters, such as the double quotation mark
("), contained in the body must be escaped.
            File file = new File(filePath);
            HttpEntity entity =
MultipartEntityBuilder.create().addBinaryBody("images",
file).setContentType(ContentType.MULTIPART_FORM_DATA).setCharset(Consts.U
TF_8).build();
            httpPost.setEntity(entity);

            // Send post
            CloseableHttpResponse response =
HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());

System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

   El nombre **addBinaryBody** viene determinado por el parámetro de entrada del servicio en tiempo real. El nombre del parámetro debe ser el mismo que el del parámetro de entrada del tipo de archivo. En este ejemplo, se utilizan **images**.

   – **Ingreso de texto (JSON)**

   A continuación se muestra un ejemplo del cuerpo de la solicitud para leer el archivo de inferencia local y realizar la codificación Base64:

```
// Package name of the demo.
package com.apig.sdk.demo;
```



```
import org.apache.http.HttpHeaders;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class MyAppCodeTest {

    public static void main(String[] args) {
        String url = "URL of the real-time service";
        String appCode = "AppCode";
        String body = "{}";

        try {
            // Create post
            HttpPost httpPost = new HttpPost(url);

            // Add header parameters
            httpPost.setHeader(HttpHeaders.CONTENT_TYPE, "application/
json");
            httpPost.setHeader("X-Apig-AppCode", appCode);

            // Special characters, such as the double quotation mark
("), contained in the body must be escaped.
            httpPost.setEntity(new StringEntity(body));

            // Send post
            CloseableHttpResponse response =
HttpClients.createDefault().execute(httpPost);

            // Print result
            System.out.println(response.getStatusLine().getStatusCode());

System.out.println(EntityUtils.toString(response.getEntity()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

body está determinado por el formato de texto. JSON se utiliza como ejemplo.

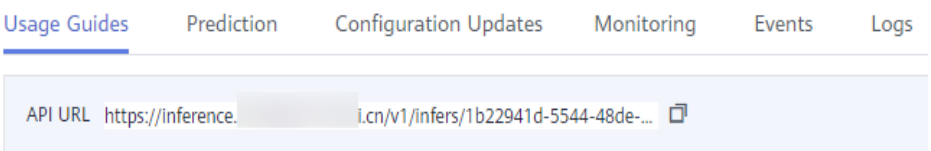
3.1.4.3 Modo de acceso

3.1.4.3.1 Acceso a un servicio en tiempo real (canal de red pública)

Contexto

Por defecto, la inferencia de ModelArts utiliza la red pública para acceder a los servicios en tiempo real. Después de desplegar un servicio en tiempo real, se proporciona una API de RESTful estándar para que invoque. Puede ver la URL de la API en la pestaña **Usage Guides** de la página de detalles del servicio.

Figura 3-24 URL de API



Acceso a un servicio en tiempo real

Los siguientes modos de autenticación están disponibles para acceder a servicios en tiempo real desde una red pública:

- **Acceso autenticado mediante un token**
- **Acceso autenticado con una AK/SK**
- **Acceso autenticado mediante una aplicación**

3.1.4.3.2 Acceso a un servicio en tiempo real (canal de VPC de alta velocidad)

Contexto

Al acceder a un servicio en tiempo real, es posible que necesite:

- Alto throughput y baja latencia
- Solicitudes de TCP o de RPC

Para cumplir con estos requisitos, ModelArts permite el acceso de alta velocidad por la interconexión de las VPC.

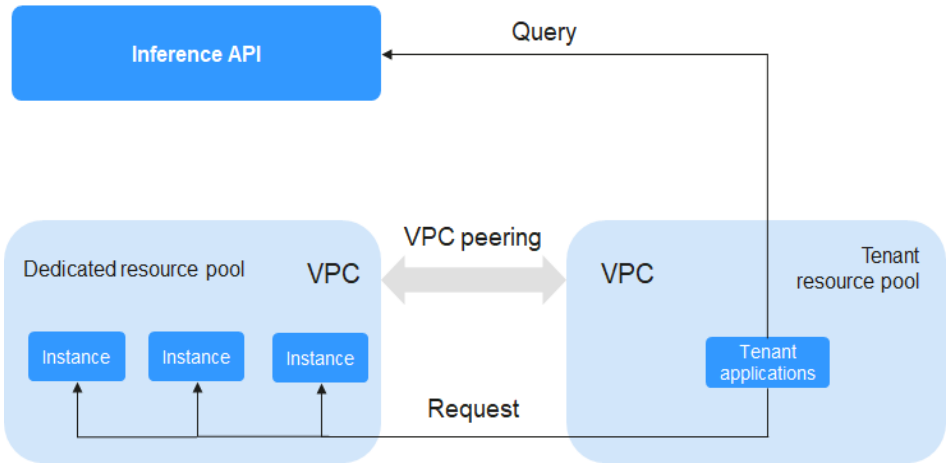
En el acceso de alta velocidad por la interconexión de las VPC, sus solicitudes de servicio se envían directamente a las instancias por la interconexión de las VPC, pero no con la plataforma de inferencia. Esto acelera el acceso al servicio.

NOTA

Las siguientes funciones disponibles con la plataforma de inferencia no estarán disponibles si utiliza el acceso de alta velocidad:

- Autenticación
- Distribución del tráfico por configuración
- Balanceo de carga
- Alarma, monitoreo y estadísticas

Figura 3-25 Acceso de alta velocidad con interconexión de las VPC



Preparación

Despliegue un servicio en tiempo real en un grupo de recursos dedicado y asegúrese de que el servicio se está ejecutando.

AVISO

- Para obtener más detalles sobre cómo desplegar servicios en grupos de recursos dedicados de nueva versión, consulte **Actualizaciones amplias de funciones de gestión de grupos de recursos de ModelArts**.
- Solo los servicios desplegados en un grupo de recursos dedicado soportan acceso de alta velocidad por la interconexión de las VPC.
- El acceso de alta velocidad por interconexión de las VPC solo está disponible para los servicios en tiempo real.
- Debido al control del tráfico, existe un límite en la frecuencia con la que se puede obtener la dirección IP y el número de puerto de un servicio en tiempo real. La cantidad de llamadas de cada cuenta de tenant no puede superar las 2000 por minuto y la de cada cuenta de usuario de IAM no puede superar las 20 por minuto.
- El acceso de alta velocidad por interconexión de VPC solo está disponible para los servicios desplegados con las aplicaciones de IA importadas desde las imágenes personalizadas.

Procedimiento

Para habilitar el acceso de alta velocidad a un servicio en tiempo real por interconexión de las VPC, realice las siguientes operaciones:

1. **Interconecte el grupo de recursos dedicados a la VPC.**
2. **Cree un ECS en la VPC.**
3. **Obtenga la dirección IP y el número de puerto del servicio en tiempo real.**
4. **Acceda al servicio con la dirección IP y el número de puerto.**

Paso 1 Interconecte el grupo de recursos dedicados a la VPC.

Inicie sesión en la consola de gestión de ModelArts, seleccione **Dedicated Resource Pools** > **Elastic Cluster**, localice el grupo de recursos dedicado utilizado para el despliegue de servicio y haga clic en su nombre/ID para ir a la página de detalles del grupo de recursos. Obtenga la configuración de red. Vuelva a la lista del grupo de recursos dedicados, haga clic en la pestaña **Network**, localice la red asociada al grupo de recursos dedicados e interconéctela con la VPC. Después de acceder a la VPC, esta VPC aparecerá en las páginas de detalles de lista de redes y del grupo de recursos. Haga clic en la VPC para ir a la página de detalles.

Figura 3-26 Localización del grupo de recursos dedicados de destino

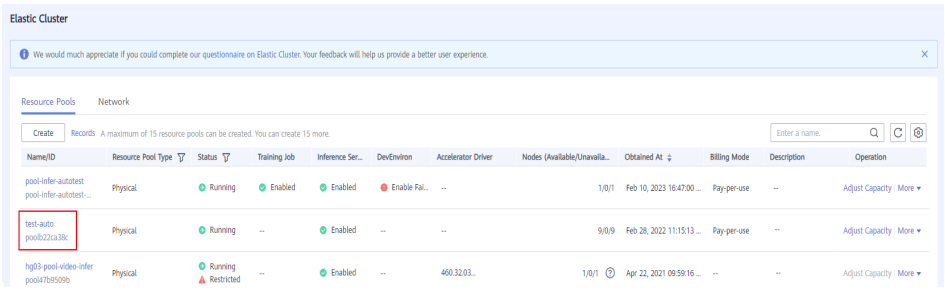


Figura 3-27 Obtención de la configuración de red

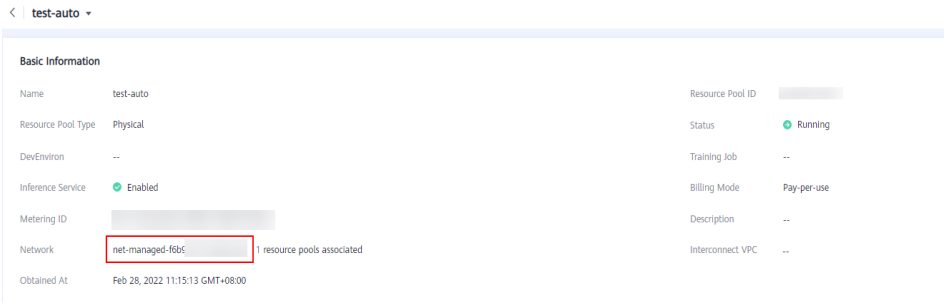
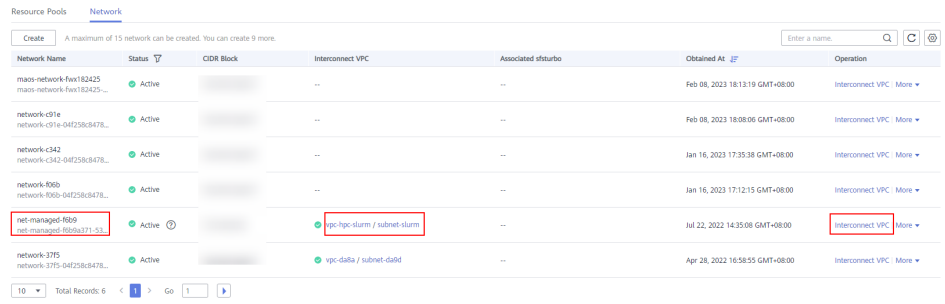


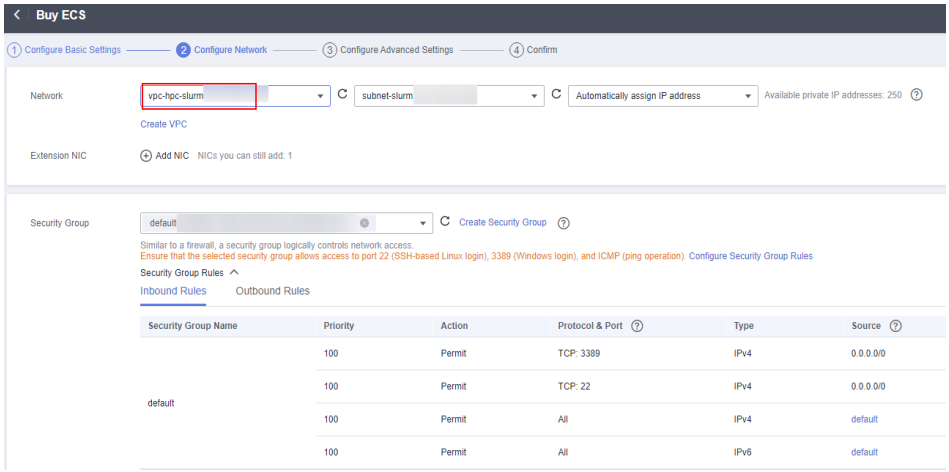
Figura 3-28 Interconexión de las VPC

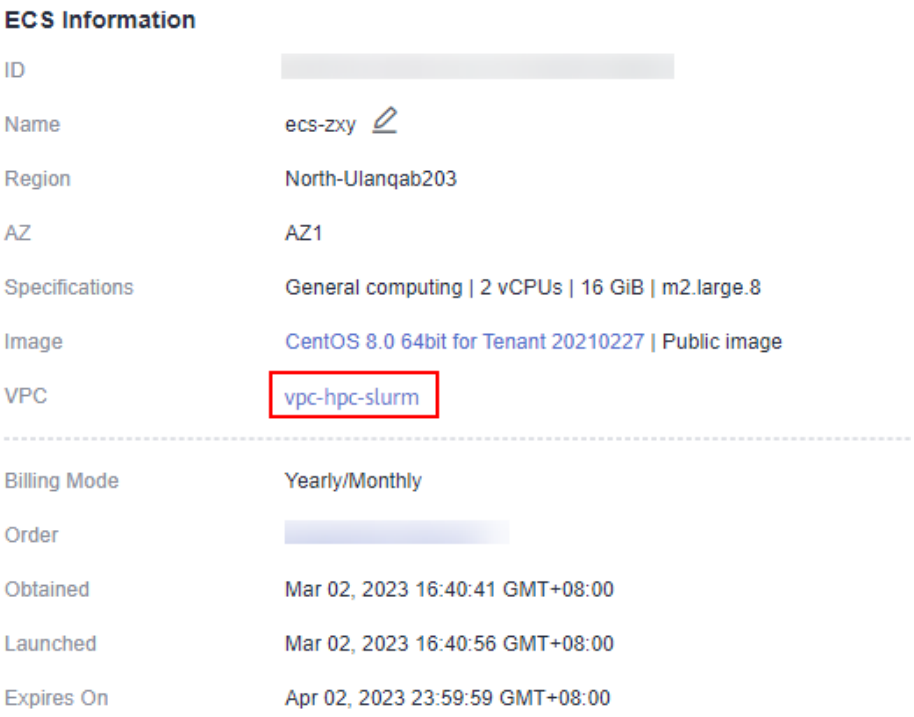


Paso 2 Creación de un ECS en la VPC.

Inicie sesión en la consola de gestión de ECS y haga clic en **Buy ECS** en la esquina superior derecha. En la página **Buy ECS**, configure los ajustes básicos y haga clic en **Next: Configure Network**. En la página **Configure Network**, seleccione la VPC conectada en **Paso 1**, configure otros parámetros, confirme las configuraciones y haga clic en **Submit**. Cuando el estado del ECS cambia a **Running**, se ha creado el ECS. Haga clic en su nombre o ID para ir a la página de detalles del servidor y ver la configuración de VPC.

Figura 3-29 Selección de una VPC al comprar un ECS





Paso 3 Obtenga la dirección IP y el número de puerto del servicio en tiempo real.

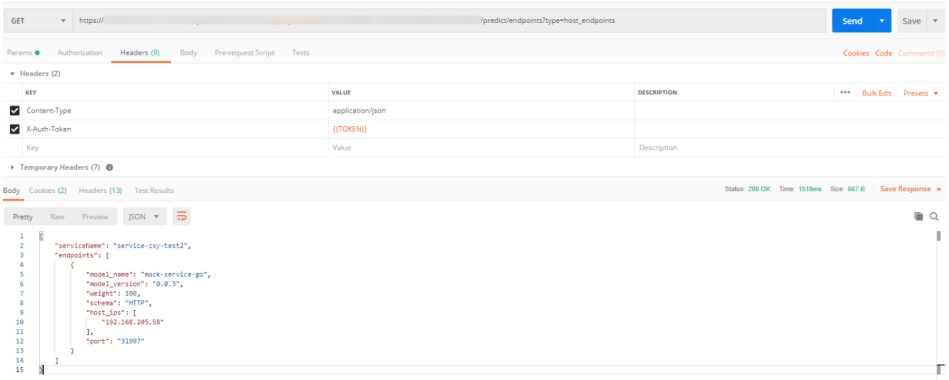
Se puede utilizar el software de GUI, por ejemplo, con Postman puede obtener la dirección IP y el número de puerto. De forma alternativa, inicie sesión en el ECS, cree un entorno de Python y ejecute el código para obtener la dirección IP del servicio y el número de puerto.

API:

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- Método 1: Obtenga la dirección IP y el número de puerto utilizando el software de GUI.

Figura 3-30 Ejemplo de la respuesta



- Método 2: Obtenga la dirección IP y el número de puerto usando Python.

  Es necesario modificar los siguientes parámetros en el código Python que aparece a continuación:

  – **project_id**: Su ID de proyecto. Para obtenerlo, consulte **Obtención de un ID y nombre de proyecto**.

  – **service_id**: ID del servicio, que puede consultarse en la página de detalles del servicio.

  – **REGION_ENDPOINT**: servicio de punto de conexión. Para obtenerlo, véase **Puntos de conexión**.

```
def get_app_info(project_id, service_id):
    list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?
type=host_endpoints"
    url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id,
service_id)
    headers = {'X-Auth-Token': X_Auth_Token}
    response = requests.get(url, headers=headers)
    print(response.content)
```

Paso 4 Acceda al servicio con la dirección IP y el número de puerto.

Inicie sesión en el ECS y acceda al servicio en tiempo real ejecutando comandos de Linux o creando un entorno de Python y ejecutando código de Python. Obtenga los valores de **schema**, **ip** y **port** de **Paso 3**.

- Ejecute el siguiente comando para acceder al servicio en tiempo real:

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type:  application/json' \
--data-raw '{"a":"a"}'
```

Figura 3-31 Acceso a un servicio en tiempo real

- Cree un entorno de Python y ejecute el código de Python para acceder al servicio en tiempo real.

```
def vpc_infer(schema, ip, port, body):
    infer_url = "{}://{}:{}"
    url = infer_url.format(schema, ip, port)
    response = requests.post(url, data=body)
    print(response.content)
```


NOTA

El acceso de alta velocidad no admite el balanceo de carga. Debe personalizar las políticas de equilibrio de carga cuando desplegar varias instancias.

----Fin

3.1.4.4 Acceso a un servicio en tiempo real con WebSocket

Contexto

WebSocket es un protocolo de transmisión de red que admite comunicación dúplex completa a través de una sola conexión de TCP. Se encuentra en la capa de aplicación del modelo OSI. El protocolo de comunicación de WebSocket fue establecido por el IETF como estándar RFC 6455 en 2011 y complementado por RFC 7936. La API de WebSocket en Web IDL está estandarizada por W3C.

WebSocket simplifica el intercambio de datos entre el cliente y el servidor y permite que el servidor envíe datos de forma proactiva al cliente. En la API de WebSocket, si el handshake inicial entre el cliente y el servidor es exitoso, se puede establecer una conexión persistente entre ellos y se puede realizar la transmisión de datos bidireccional.

Requisitos previos

- Se ha desplegado un servicio en tiempo real con **WebSocket** habilitado.
- La imagen para importar la aplicación de IA es compatible con WebSocket.

Restricciones

- WebSocket solo soporta el despliegue de servicios sincrónicos en tiempo real.
- WebSocket solo admite servicios en tiempo real desplegados mediante aplicaciones de IA importadas a partir de imágenes personalizadas.

Invocación a un servicio en tiempo real de WebSocket

WebSocket no requiere la autenticación adicional. ModelArts WebSocket es compatible con WebSocket Secure, independientemente de si WebSocket o WebSocket Secure está activado en la imagen personalizada. WebSocket Secure solo admite la autenticación unidireccional, desde el cliente hasta el servidor.

Puede utilizar uno de los siguientes métodos de autenticación proporcionados por ModelArts:

- **Acceso autenticado con un token**
- **Acceso autenticado con una AK/SK**
- **Acceso autenticado con una aplicación**

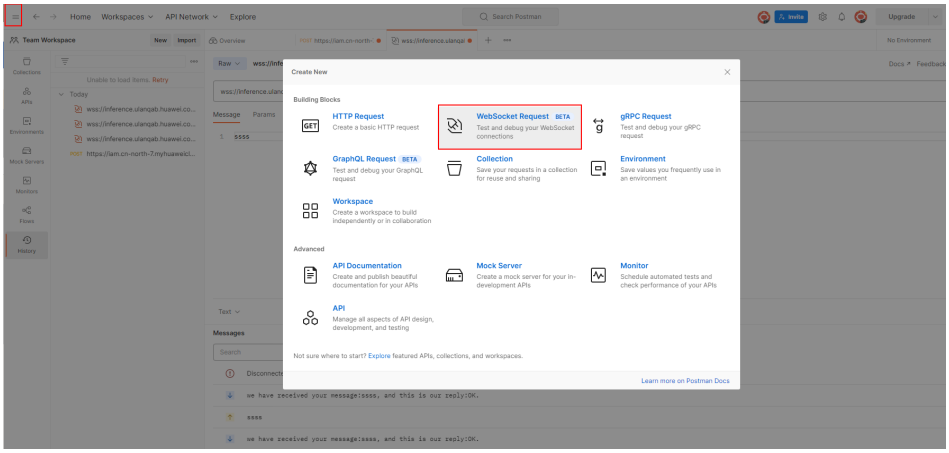
La siguiente sección utiliza el software de GUI Postman para predicción y la autenticación de token como ejemplo para describir cómo invocar a WebSocket.

1. **Establezca una conexión de WebSocket.**
2. **Intercambie datos entre el cliente de WebSocket y el servidor.**

Paso 1 Establecer una conexión de WebSocket.

1. Abra Postman de una versión posterior a 8.5, por ejemplo, 10.12.0. Haga clic en en la esquina superior izquierda y elija **File** > **New**. En el cuadro de diálogo mostrado, seleccione **WebSocket Request** (versión de beta actual).

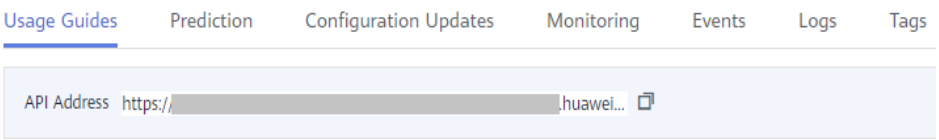
Figura 3-32 Solicitud de WebSocket



2. Configure los parámetros para la conexión de WebSocket.

   Seleccione **Raw** en la esquina superior izquierda. No seleccione **Socket.IO** (un tipo de implementación de WebSocket que requiere que tanto el cliente como el servidor funcionen con **Socket.IO**). En el cuadro de direcciones, introduzca la **API Address** obtenida en la ficha **Usage Guides** de la página de detalles del servicio. Si hay una dirección URL detallada en la imagen personalizada, agréguela al final de la dirección. Si **queryString** está disponible, agregue este parámetro a la columna **params**. Agregue información de autenticación al encabezado. El encabezado varía según el modo de autenticación, que es el mismo que el del servicio de inferencia compatible con HTTPS. Haga clic en **Connect** en la esquina superior derecha para establecer una conexión de WebSocket.

Figura 3-33 Obtención de la dirección API

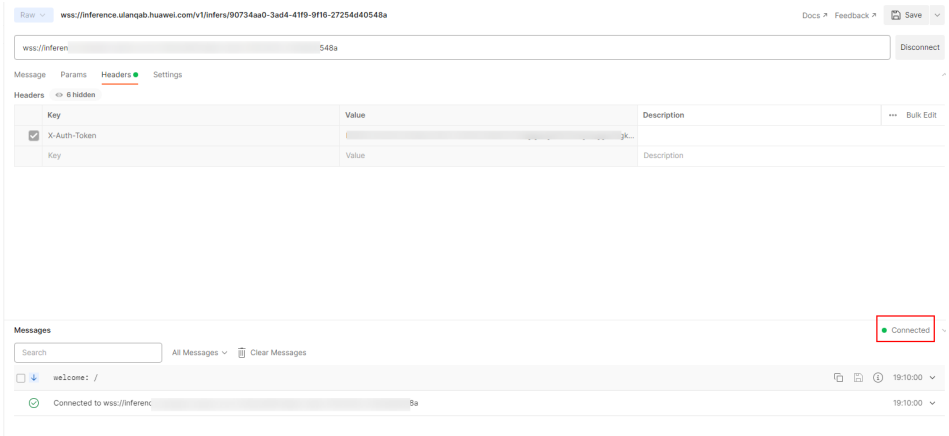


NOTA

- Si la información es correcta, aparecerá **CONNECTED** en la esquina inferior derecha.
- Si no se puede establecer la conexión y el código de estado es 401, verifique la autenticación.
- Si aparece una palabra clave como **WRONG_VERSION_NUMBER**, verifique si el puerto configurado en la imagen personalizada es el mismo que el configurado en WebSocket o WebSocket Secure.

A continuación se muestra una conexión de WebSocket establecida.

Figura 3-34 Conexión establecida



AVISO

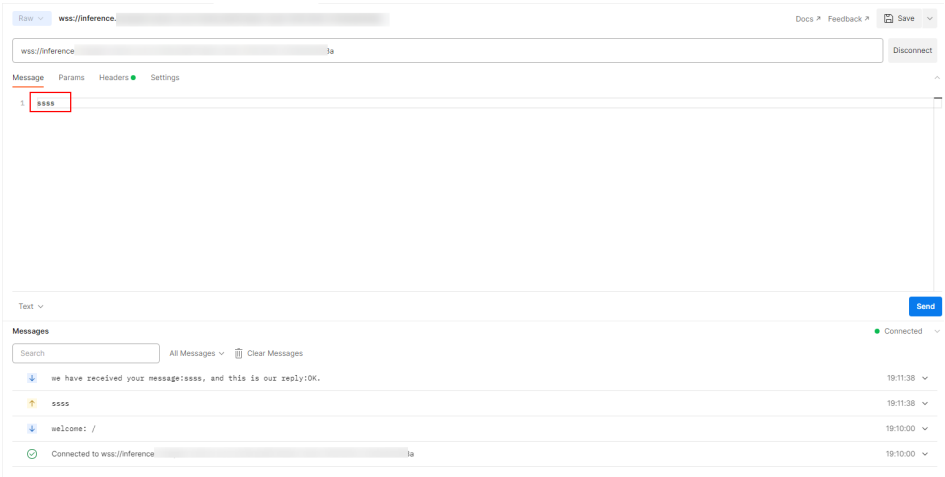
Verifique preferentemente el servicio de WebSocket proporcionado por la imagen personalizada. El tipo de implementación de WebSocket varía según la herramienta utilizada. Los problemas posibles son los siguientes: se puede establecer una conexión de WebSocket pero no se puede mantener, o la conexión se interrumpe tras una solicitud y es necesario volver a conectarse. ModelArts solo garantiza que no afectará al estado de WebSocket en una imagen personalizada (la dirección de API y el modo de autenticación pueden cambiarse en ModelArts).

Paso 2 Intercambie datos entre el cliente de WebSocket y el servidor.

Una vez establecida la conexión, WebSocket utiliza TCP para la comunicación dúplex completa. El cliente de WebSocket envía datos al servidor. Los tipos de implementación varían dependiendo del cliente, y el paquete de lib también puede ser diferente para el mismo idioma. Aquí no se tienen en cuenta los distintos tipos de implementación.

El formato de los datos enviados por el cliente no está limitado por el protocolo. Postman admite datos de texto, JSON, XML, HTML y Binary. Tome el texto como ejemplo. Introduzca los datos de texto en el cuadro de texto y haga clic en **Send** a la derecha para enviar la solicitud al servidor. Si el texto es demasiado grande, Postman puede ser suspendido.

Figura 3-35 Envío de datos



----Fin

3.1.4.5 Server-Sent Events

Contexto

Server-Sent Events (SSE) es una tecnología de inserción de servidor que permite a un servidor enviar eventos a un cliente a través de una conexión de HTTP. Esta tecnología se utiliza generalmente para permitir que un servidor envíe datos en tiempo real a un cliente, por ejemplo, una aplicación de chat o una actualización de noticias en tiempo real.

SSE facilita principalmente la comunicación unidireccional en tiempo real desde el servidor al cliente, como el streaming de respuestas de ChatGPT. A diferencia de los WebSockets que proporcionan comunicación bidireccional en tiempo real, SSE está diseñado para ser más liviano y más simple de implementar.

Requisitos previos

La imagen para importar la aplicación de IA es compatible con SSE.

Restricciones

- SSE solo soporta el despliegue de servicios en tiempo real.
- Solo admite servicios en tiempo real desplegados mediante aplicaciones de IA importadas de imágenes personalizadas.

Llamada a un servicio en tiempo real de SSE

El protocolo de SSE en sí no introduce nuevos mecanismos de autenticación; se basa en los mismos métodos que las solicitudes de HTTP.

Puede utilizar uno de los siguientes métodos de autenticación proporcionados por ModelArts:

- **Acceso autenticado con un token**
- **Acceso autenticado con una AK/SK**
- **Acceso autenticado con una aplicación**

La siguiente sección utiliza el software de GUI de Postman para la predicción y la autenticación de tokens como ejemplo para describir cómo invocar a un servicio de SSE.

Figura 3-36 Invocar a un servicio de SSE

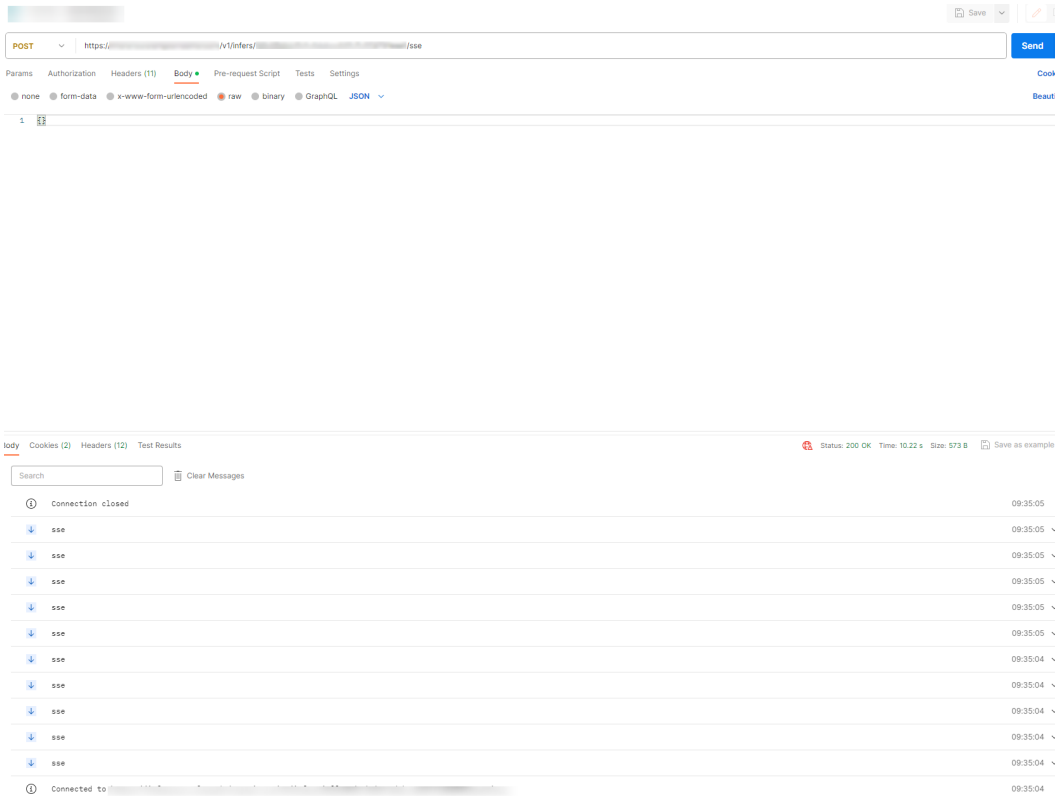


Figura 3-37 Encabezado de respuesta **Content-Type**

NOTA

En casos normales, el valor de **Content-Type** en el encabezado de respuesta es **text/event-stream;charset=UTF-8**.

3.1.5 Integración de un servicio en tiempo real

Para una API de servicio en tiempo real que se ha encargado, puede integrarla en el entorno de producción.

Requisitos previos

El servicio en tiempo real se está ejecutando. De lo contrario, las aplicaciones en el entorno de producción no estarán disponibles.

Modo de integración

Los servicios en tiempo real de ModelArts proporcionan API RESTful estándar, a las que se puede acceder mediante HTTPS. ModelArts proporciona SDK para invocar a las API de servicio en tiempo real. Para obtener detalles sobre cómo invocar a los SDK, consulte "Escenario 1: Realizar una prueba de inferencia usando el predictor" en ***Referencia del SDK***.

Además, puede usar herramientas y lenguajes de desarrollo comunes para invocar a las API. Puede buscar y obtener las guías para invocar a las API de RESTful estándar en Internet.

3.1.6 Cloud Shell

Escenarios

Puede usar Cloud Shell proporcionado por la consola de ModelArts para iniciar sesión en un contenedor de instancias de servicio en tiempo real en ejecución.

Restricciones

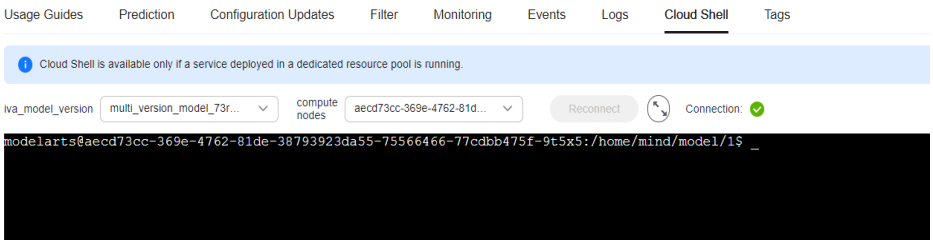
- Cloud Shell solo puede acceder a un contenedor cuando el servicio en tiempo real asociado se despliega dentro de un grupo de recurso dedicado
- Cloud Shell solo puede acceder a un contenedor cuando se está ejecutando el servicio en tiempo real asociado.

Uso de Cloud Shell

1. Inicie sesión en la consola de ModelArts. En el panel de navegación, seleccione **Service Deployment** > **Real-Time Services**.
2. En la página de lista de servicios en tiempo real, haga clic en el nombre o ID del servicio de destino. Aparecerá la página de detalles del servicio en tiempo real.
3. Haga clic en la ficha **Cloud Shell** y seleccione la versión de la aplicación de IA y el nodo de cómputo. Cuando el estado de la conexión cambia a , se ha iniciado sesión en el contenedor de instancias.

   Si el servidor se desconecta debido a un error o permanece inactivo durante 10 minutos, puede seleccionar **Reconnect** para recuperar el acceso a la instancia de contenedor.

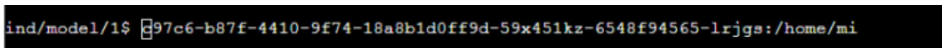
Figura 3-38 Cloud Shell



NOTA

Es posible que se produzca una excepción de visualización de rutas al iniciar sesión en la página de Cloud Shell. En este caso, presione **Enter** para detectar la falla.

Figura 3-39 Ruta anormal



3.2 Despliegue de aplicaciones de IA como servicios por lotes

3.2.1 Despliegue como servicio por lotes

Después de preparar una aplicación de IA, puede desplegarla como un servicio por lotes. La página **Service Deployment > Batch Services** muestra todos los servicios por lotes.

Requisitos previos

- Hay disponible una aplicación de ModelArts en estado **Normal**.
- Los datos que se van a procesar por lotes están listos y se han subido a un directorio de OBS.
- Se ha creado al menos una carpeta vacía en OBS para almacenar la salida.

Contexto

- Se puede crear un máximo de 1,000 de servicios por lotes.
- En función de la solicitud de entrada (JSON o archivo) definida por la aplicación de IA, se ingresan diferentes parámetros. Si la entrada de la aplicación de IA es un archivo de JSON, se requiere un archivo de configuración para generar un archivo de asignación. Si la entrada de la aplicación AI es un archivo, no se requiere ningún archivo de mapeo.
- Los servicios por lotes solo se pueden desplegar en un grupo de recursos público, pero no en un grupo de recursos dedicado.

Procedimiento

1. Inicie sesión en la consola de gestión de ModelArts. En el panel de navegación izquierdo, elija **Service Deployment** > **Batch Services**. De forma predeterminada, se muestra la página **Batch Services**.
2. En la lista de servicios por lotes, haga clic en **Deploy** en la esquina superior izquierda. Se muestra la página **Deploy**.
3. Establezca parámetros para un servicio por lotes.
   a. Configure la información básica, como **Name** y **Descripción**. El nombre se genera de forma predeterminada, por ejemplo, **service-bc0d**. Puede especificar **Name** y **Description** de acuerdo con los requisitos reales.
   b. Establezca otros parámetros, incluidas las configuraciones del grupo de recursos y de la aplicación de IA. Para más detalles, véase **Tabla 3-11**.

Tabla 3-11 Parámetros

Parámetro	Descripción
AI Application Source	Seleccione **My AI Applications** o **My Subscriptions** según sus requisitos.
AI Application and Version	Seleccione una aplicación de IA y su versión que se ejecuten correctamente.
Input Path	Seleccione el directorio de OBS donde se almacenan los datos cargados. Seleccione una carpeta o un archivo .manifest. Para obtener detalles sobre las especificaciones del archivo .manifest, véase **Especificaciones del archivo de manifiesto**. **NOTA** ● Si los datos de entrada son una imagen, asegúrese de que el tamaño de una sola imagen sea inferior a 10 MB. ● Si los datos de entrada están en formato CSV, asegúrese de que no se incluya ningún carácter chino. ● Si los datos de entrada están en formato de CSV, asegúrese de que el tamaño del archivo no exceda los 12 MB.
Request Path	URL utilizada para invocar a la API de aplicación de IA en un servicio por lotes, y también la ruta de solicitud del servicio de aplicación de IA. Su valor se obtiene del campo **url** de **apis** en el archivo de configuración de la aplicación de IA.
Mapping Relationship	Si la entrada de la aplicación de IA está en formato JSON, el sistema genera automáticamente la asignación basada en el archivo de configuración correspondiente a la aplicación de IA. Si la entrada de la aplicación de IA es otro archivo, la asignación no es necesaria. Archivo de asignación generado automáticamente. Introduzca el índice de campo correspondiente a cada parámetro en el archivo CSV. El índice comienza desde 0. Regla de asignación: La regla de asignación proviene del parámetro de entrada (**request**) en el archivo de configuración del modelo **config.json**. Cuando **type** se establece en **string, number, integer, or boolean**, se requiere configurar el parámetro de index. Para obtener detalles sobre la regla de mapeo, véase **Ejemplo de mapeo**. El índice debe ser un entero positivo a partir de 0. Si el valor de index no cumple con la regla, este parámetro se omite en la solicitud. Después de configurar la regla de asignación, los datos de CSV correspondientes deben estar separados por comas (,).
Output Path	Seleccione la ruta para guardar el resultado de la predicción por lotes. Puede seleccionar la carpeta vacía que cree.

Parámetro	Descripción
Specifications	El sistema proporciona recursos informáticos disponibles que coinciden con su aplicación de IA. Seleccione un recurso disponible en la lista desplegable. Por ejemplo, si el modelo proviene de un proyecto ExeML, los recursos de cálculo se asocian automáticamente a las especificaciones de ExeML para su uso.
Compute Nodes	Establezca el número de instancias para la versión actual de la aplicación de IA. Si establece el número de nodos en **1**, se utilizará el modo de cómputo independiente. Si establece el número de nodos en un valor mayor que 1, se utilizará el modo de cómputo distribuido. Seleccione un modo de cómputo basado en los requisitos reales.
Environment Variable	Establezca las variables de entorno e inyéctelas en el pod. Para garantizar la seguridad de los datos, no introduzca información confidencial, como contraseñas de texto sin formato, en las variables de entorno.
Timeout	Tiempo de espera de un único modelo, incluido el tiempo de despliegue y de inicio. El valor predeterminado es 20 minutos. El valor debe estar dentro del rango de 3 a 120.
Runtime Log Output	Esta función está deshabilitada por defecto. Los logs de ejecución de los servicios por lotes solo se almacenan en el sistema de log de ModelArts. Puede consultar los logs de ejecución en la página de fichas **Logs** de la página de detalles del servicio. Si esta función está habilitada, los logs de ejecución de los servicios por lotes se exportan y almacenan en Log Tank Service (LTS). LTS crea automáticamente grupos de logs y flujos de logs y cachés ejecuta logs generados en un plazo de siete días de forma predeterminada. Para obtener más detalles sobre la función de gestión de logs LTS, véase **Log Tank Service**. **NOTA** ● Esto no se puede deshabilitar una vez que está habilitado. ● Se le facturarán las funciones de consulta de logs y almacenamiento de logs proporcionadas por LTS. Para obtener más detalles, véase la sección **Detalles de precios de LTS**.

4. Después de establecer los parámetros, despliegue el modelo como un servicio por lotes según se le solicite. El despliegue de un servicio generalmente requiere un período de tiempo, que puede ser de varios minutos o decenas de minutos, dependiendo de la cantidad de datos y recursos.

   NOTA

   Después de desplegar un servicio por lotes, se inicia inmediatamente. Durante el funcionamiento, se le cobrará en función de los recursos seleccionados.

Puede ir a la lista de servicios por lotes para ver la información básica sobre el servicio por lotes. En la lista de servicios por lotes, después de que el estado del servicio recién implementado cambie de **Deploying** a **Running**, el servicio se implementa correctamente.

Especificaciones del archivo de manifiesto

Los servicios por lotes de ModelArts soportan archivos de manifiesto, que describen la entrada y salida de datos.

Ejemplo de archivo del manifiesto de entrada

- Nombre del archivo: **test.manifest**
- Contenido del archivo:

```
{"source": "obs://test/data/1.jpg"}
{"source": "s3://test/data/2.jpg"}
{"source": "https://infers-data.obs.cn-north-1.myhuaweicloud.com:443/
xgboosterdata/data.csv?
AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz
1uDzwve8GpY%3D&x-obs-security-
token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-
ZaMSxHvl68kKLAy5feYvLDM..."}
```

- Requerimientos del archivo:
  a. La extensión del nombre de archivo debe ser **.manifest**.
  b. El contenido del archivo está en formato JSON. Cada fila describe un fragmento de datos de entrada, que debe ser preciso para un archivo en lugar de una carpeta.
  c. Se debe definir un campo **source** para el contenido JSON. El valor del campo es la dirección URL OBS del archivo en cualquiera de los siguientes formatos:
     i. Ruta de acceso de bucket *<obs path>{{Bucket name}}/{{Object name}}/File name* que se utiliza para acceder a los datos de OBS. Puede acceder a la ruta para obtener un objeto en OBS. *<obs path>* puede ser **obs://** o **s3://**.
     ii. Enlace compartido generado por OBS, incluida la información de firma. Se aplica al acceso a los datos de OBS de otros usuarios. El enlace tiene un período de validez. Realizar operaciones dentro del período.

Ejemplo de archivo de manifiesto de salida

Se generará un archivo de manifiesto en el directorio de salida de los servicios por lotes.

- Supongamos que la ruta de salida es **//test-bucket/test/**. El resultado se almacena en la siguiente ruta:

```
OBS bucket/directory name
├── test-bucket
│   ├── test
│   │   ├── infer-result-{{task_id}}.manifest
│   │   ├── infer-result
│   │   │ ├── 1.jpg_result.txt
│   │   │ ├── 2.jpg_result.txt
```

- Contenido del archivo **infer-result-0.manifest**:

```
{"source": "obs://obs-data-bucket/test/data/
1.jpg","result":"SUCCESSFUL","inference-loc": "obs://test-bucket/test/infer-
result/1.jpg_result.txt"}
{"source": "s3://obs-data-bucket/test/data/
2.jpg","result":"FAILED","error_message": "Download file failed."}
{"source ": "https://infers-data.obs.xxx.com:443/xgboosterdata/2.jpg?
AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz
1uDzwve8GpY%3D&x-obs-security-
token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-
ZaMSxHvl68kKLAy5feYvLDMNZWxzhBZ6Q-3HcoZMh9gISwQOVBwm4ZytB_m8sg1fL6isU7T3CnoL9j
```



```
mvDGgT9VBC7dC1EyfSJrUcqfB_N0ykCsfrA1Tt_IQYZFDu_HyqVk-
GunUcTVdDfWlCV3TrYcpmznZjliAnYUO89kAwCYGeRZsCsC0ePu4PHMsBvYV9gWmN9AUZIDn1sfRL4
voBpwQnp6tnAgHW49y5a6hP2hCAoQ-95SpUriJ434QlymoeKfTHVMKOeZxZea-
JxOvevOCGI5CcGehEJaz48sgH81UiHzl21zocNB_hpPfus2jY6KPglEJxMv6Kwmro-
ZBXWuSJUDOnSYXI-3ciYjg9-
h10b8W3sW1mOTFCWNGoWsd74it7l_5-7UUhoIeyPByO_REwkur2FOJsuMpGlRaPyglZxXm_jfdLFXo
bYtzZhbul4yWXga6oxTOkfcwykTOYH0NPoPRt5MYGYweOXXxFs3d5w2rd0y7p0QYhyTzIkk5CIz7Fl
WNapFISL7zdhsl8RfchTqESq94KgkeqatSF_iIvnYMW2r8P8x2k_eb6NJ7U_q5ztMbO9oWEcfr0D2f
7n7Bl_nb2HIB_H9tjzKvqwngaimYhBbMRPfibvttW86GiwVP8vrC27FOn39Be9z2hSfJ_8pHej0yMl
yNqZ481FQ5vWT_vFV3JHM-7I1ZB0_hIdaHfItm-J69cTfHSEOzt7DGaMIES1o7U3w%3D
%3D","result":"SUCCESSFUL","inference-loc": "obs://test-bucket/test/infer-
result/2.jpg_result.txt"}
```


- Formato de archivo:
  a. El nombre del archivo es **infer-result-{{task_id}}.manifest**, que **task_id** es el ID de la tarea por lotes, que es único para un servicio por lotes.
  b. Si es necesario procesar un gran número de archivos, se pueden generar varios archivos de manifiesto con el mismo sufijo **.manifest** y se distinguen por sufijo, por ejemplo, **infer-result-{{task_id}}_1.manifest**.
  c. El directorio **infer-result-***{{task_id}}* se crea en el directorio del manifiesto para almacenar el resultado del procesamiento del archivo.
  d. El contenido del archivo está en formato de JSON. Cada fila describe el resultado de salida de una pieza de datos de entrada.
  e. El archivo contiene varios campos:
     i. **source**: descripción de datos de entrada, que es la misma que la del archivo de manifiesto de entrada
     ii. **result**: resultado de procesamiento de archivos, que puede ser **SUCCESSFUL** **FAILED**
     iii. inference-loc: ruta de resultados de salida. Este campo está disponible cuando el resultado es **SUCCESSFUL**. El formato es **obs://{{***Bucket name***}}/{***Object name***}**.
     iv. error_message: información de error. Este campo está disponible cuando el resultado es FAILED.

Ejemplo de mapeo

En el ejemplo siguiente se muestra la relación entre el archivo de configuración, la regla de asignación, los datos CSV y la solicitud de inferencia.

A continuación se utiliza un archivo para la predicción como ejemplo:

```
[
    {
        "method": "post",
        "url": "/",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "data": {
                        "type": "object",
                        "properties": {
                            "req_data": {
                                "type": "array",
                                "items": [
                                    {
                                        "type": "object",
```

```
                                        "properties": {
                                            "input_1": {
                                                "type": "number"
                                            },
                                            "input_2": {
                                                "type": "number"
                                            },
                                            "input_3": {
                                                "type": "number"
                                            },
                                            "input_4": {
                                                "type": "number"
                                            }
                                        }
                                    }
                                ]
                            }
                        }
                    }
                }
            }
        }
    }
]
```

La consola de gestión de ModelArts resuelve automáticamente la relación de asignación desde el archivo de configuración como se muestra a continuación. Al invocar a una API de ModelArts, configure la asignación siguiendo la regla.

```
{
    "type": "object",
    "properties": {
        "data": {
            "type": "object",
            "properties": {
                "req_data": {
                    "type": "array",
                    "items": [
                        {
                            "type": "object",
                            "properties": {
                                "input_1": {
                                    "type": "number",
                                    "index": 0
                                },
                                "input_2": {
                                    "type": "number",
                                    "index": 1
                                },
                                "input_3": {
                                    "type": "number",
                                    "index": 2
                                },
                                "input_4": {
                                    "type": "number",
                                    "index": 3
                                }
                            }
                        }
                    ]
                }
            }
        }
    }
}
```

Múltiples partes de datos de CSV para inferencia se separan con comas (,). A continuación se muestra un ejemplo:


```
5.1,3.5,1.4,0.2
4.9,3.0,1.4,0.2
4.7,3.2,1.3,0.2
```

Dependiendo de la relación de mapeo definida, a continuación se muestra la solicitud de inferencia, cuyo formato es similar al de los servicios en tiempo real.

```
{
    "data": {
        "req_data": [{
            "input_1": 5.1,
            "input_2": 3.5,
            "input_3": 1.4,
            "input_4": 0.2
        }]
    }
}
```

3.2.2 Consulta de detalles de un servicio por lotes

Después de desplegar una aplicación de IA como servicio por lotes, puede acceder a la página **Batch Services** para ver sus detalles.

1. Inicie sesión en la consola de ModelArts y seleccione **Service Deployment** > **Batch Services**.
2. Haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio.

   Consulte la información del servicio. Para más detalles, véase **Tabla 3-12**.

Tabla 3-12 Parámetros del servicio por lotes

Parámetro	Descripción
Name	Nombre del servicio por lotes.
Service ID	ID del servicio por lotes.
Status	Estado del servicio por lotes.
Job ID	ID de trabajo del servicio por lotes.
Instance Flavor	Variante de nodo del servicio por lotes.
Compute Nodes	Número de nodos del servicio por lotes.
Start Time	Hora de inicio del trabajo de servicio por lotes.
Environment Variable	Variables de entorno agregadas durante la creación del servicio por lotes.
End Time	Hora de fin del trabajo de servicio por lotes.
Description	Descripción del servicio, que puede hacer clic en el botón de edición para modificar.
Input Path	Ruta de OBS a los datos de entrada en el servicio por lotes.
Output Path	Ruta de OBS a los datos de salida en el servicio por lotes.

Parámetro	Descripción
AI Application Name & Version	Nombre y versión de la aplicación de IA utilizada por el servicio por lotes.
Advanced Log Management	Esta función está deshabilitada por defecto. Los logs de tiempo de ejecución de los servicios por lotes se almacenan solo en el sistema de logs de ModelArts. Si esta función está habilitada, los logs de tiempo de ejecución de los servicios por lotes se exportarán y almacenarán en Log Tank Service (LTS). LTS crea automáticamente grupos de logs y flujos de logs y cachés ejecuta logs generados en un plazo de siete días de forma predeterminada. Para obtener más detalles sobre la función de gestión de logs LTS, véase **Log Tank Service**. **NOTA** ● Esto no se puede deshabilitar una vez que está habilitado. ● Se le facturarán las funciones de consulta de logs y almacenamiento de logs proporcionadas por LTS. Para obtener más detalles, véase la sección **Detalles de precios de LTS**.

3. Cambie entre las pestañas de la página de detalles de un servicio por lotes para ver más detalles. Para más detalles, véase **Tabla 3-13**.

Tabla 3-13 Fichas de servicio por lotes

Parámetro	Descripción
Events	Esta página muestra operaciones clave durante el uso del servicio, como el progreso de despliegue del servicio, causas detalladas de excepciones de despliegue y puntos en el tiempo cuando se inicia, para o modifica un servicio. Los eventos se guardan durante un mes y luego se borran automáticamente. Para obtener detalles sobre cómo ver los eventos de un servicio, véase **Consulta de eventos de servicio**.

Parámetro	Descripción
Logs	Esta página muestra la información de log de cada aplicación de IA del servicio. Puede ver los registros generados en los últimos 5 minutos, los últimos 30 minutos, las últimas 1 hora y el segmento de tiempo definido por el usuario. Puede seleccionar la hora de inicio y la hora de finalización al definir un segmento de tiempo. Si esta función está habilitada, se mostrarán los logs almacenados en LTS. Puede hacer clic en **View Complete Logs on LTS** para ver todos los logs. Reglas de búsqueda de logs: ● No ingrese una cadena que contenga alguna de las siguientes separadores: ,'";=()[]{}@&<>/:\n\t\r. ● Puede usar la búsqueda exacta por palabra clave. Una palabra clave hace referencia a la palabra entre dos delimitadores adyacentes. ● Puede usar la búsqueda difusa por palabra clave. Por ejemplo, puede escribir **error**, **er?or**, **rro*** o **er*r**. ● Puede introducir una frase para la búsqueda exacta. Por ejemplo, **Start to refresh**. ● Antes de habilitar esta función, puede combinar palabras clave con **&&** o ||. Por ejemplo, **query logs&&erro*** o **query logs||erro***. Una vez habilitada esta función, se pueden combinar palabras clave con **AND** u **OR**. Por ejemplo, **query logs AND erro*** o **query logs OR erro***.

3.2.3 Consulta del resultado de la predicción del servicio por lotes

Al desplegar un servicio por lotes, puede seleccionar la ubicación del directorio de datos de salida. Puede ver el resultado de ejecución del servicio por lotes que se encuentra en estado **Completed**.

Procedimiento

1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** > **Batch Services**.
2. Haga clic en el nombre del servicio de destino en estado **Completed**. Se muestra la página de detalles del servicio.
   - Puede ver el nombre del servicio, el estado, el ID, la ruta de entrada, la ruta de salida y la descripción.
   - Puede hacer clic en  en el área **Description** para editar la descripción.
3. Obtener la ruta de OBS detallada junto a **Output Path**, cambiar a la ruta y obtener los resultados de predicción de servicio por lotes, incluido el archivo de resultado de predicción y el resultado de predicción de aplicación de IA.

Si la predicción tiene éxito, el directorio contiene el archivo de resultado de predicción y el resultado de predicción de aplicación de IA. De lo contrario, el directorio contiene solo el archivo de resultado de la predicción.

- Archivo de resultados de predicción: El archivo está en formato *xxx***.manifest**, que contiene la ruta de acceso del archivo y el resultado de predicción, y más.
- Resultado de la predicción de la aplicación de IA:
  - Si se introducen imágenes, se genera un archivo de resultados para cada imagen en el formato *Image name*__result.txt, por ejemplo, **IMG_20180919_115016.jpg_result.txt**.
  - Si se introducen archivos de audio, se genera un archivo de resultados para cada archivo de audio en formato *Audio file name*__**result.txt**, por ejemplo, **1-36929-A-47.wav_result.txt**.
  - Si se introducen datos de tabla, el archivo de resultados se genera en el formato *Table name*__**result.txt**, por ejemplo, **train.csv_result.txt**.

3.3 Actualización de un servicio

Para un servicio desplegado, puede modificar su información básica para que coincida con los cambios del servicio y cambiar la versión de la aplicación de IA para actualizarla.

Puede modificar la información básica sobre un servicio de cualquiera de las siguientes maneras:

Método 1: Modificar información de servicio en la página Administración de servicios

Método 2: Modificar la información del servicio en la página Detalles del servicio

Requisitos previos

El servicio ha sido desplegado. El servicio en el estado **Deploying** no se puede actualizar modificando la información del servicio.

Restricciones

- Las operaciones de actualización inadecuadas interrumpirán la ejecución del servicio durante la actualización. Por lo tanto, realice esta operación con precaución.
- ModelArts admite la actualización continua sin problemas de servicios en tiempo real en algunos escenarios. Antes de actualizar, prepárese y confirme los requisitos previos.

Tabla 3-14 Escenarios para la actualización sin golpes

Fuente de metamodelo para crear una aplicación de IA	Uso de un grupo de recursos públicos	Uso de un grupo de recursos dedicado
Trabajo de entrenamiento	No se admiten	No se admiten
Plantilla	No se admiten	No se admiten

Fuente de metamodelo para crear una aplicación de IA	Uso de un grupo de recursos públicos	Uso de un grupo de recursos dedicado
Imagen de contenedores	No se admiten	Se admite. La imagen personalizada para crear una aplicación de IA debe cumplir con las **Especificaciones de imagen personalizada para crear aplicaciones de IA**.
OBS	No se admiten	No se admiten

Método 1: Modificar información de servicio en la página Administración de servicios

1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** en el panel de navegación izquierdo. Vaya a la página de gestión de servicios del servicio de destino.
2. En la lista de servicios, haga clic en **Modify** en la columna **Operation** del servicio de destino, modifique la información básica del servicio y envíe la tarea de modificación según se le solicite.

   Cuando se modifican algunos parámetros, el sistema reinicia automáticamente el servicio para que la modificación surta efecto. Cuando envíe una tarea de modificación de servicio, si es necesario reiniciar, se mostrará un cuadro de diálogo.

   – Para obtener detalles sobre los parámetros de servicio en tiempo real, véase **Despliegue como servicio en tiempo real**. Para modificar un servicio en tiempo real, también se debe configurar **Max. Invalid Instances** para configurar el número máximo de nodos que se pueden actualizar simultáneamente, durante el cual estos nodos no son válidos.

   – Para obtener detalles sobre los parámetros del servicio por lotes, véase **Despliegue como servicio por lotes**.

Método 2: Modificar la información del servicio en la página Detalles del servicio

1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** en el panel de navegación izquierdo. Vaya a la página de gestión de servicios del servicio de destino.
2. Haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio.
3. Haga clic en **Modify** en la esquina superior derecha de la página, modifique los detalles del servicio y envíe la tarea de modificación según se le solicite.

   Cuando se modifican algunos parámetros, el sistema reinicia automáticamente el servicio para que la modificación surta efecto. Cuando envíe una tarea de modificación de servicio, si es necesario reiniciar, se mostrará un cuadro de diálogo.

   – Para obtener detalles sobre los parámetros de servicio en tiempo real, véase **Despliegue como servicio en tiempo real**. Para modificar un servicio en tiempo

real, también se debe configurar **Max. Invalid Instances** para configurar el número máximo de nodos que se pueden actualizar simultáneamente, durante el cual estos nodos no son válidos.

- Para obtener detalles sobre los parámetros del servicio por lotes, véase **Despliegue como servicio por lotes**.

3.4 Inicio, parada, supresión o reinicio de un servicio

Iniciar un servicio

Puede iniciar los servicios en los estados **Successful**, **Abnormal**, o **Stopped**. No se pueden iniciar los servicios en estado **Deploying**. Un servicio se factura cuando se inicia y se encuentra en estado **Running**. Puede iniciar un servicio de las siguientes maneras:

- Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** en el panel de navegación izquierdo. Vaya a la página de gestión de servicios del servicio de destino. Haga clic en **Start** en la columna **Operation** para iniciar el servicio de destino.
- Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** en el panel de navegación izquierdo. Vaya a la página de gestión de servicios del servicio de destino. Haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio. Haga clic en **Start** en la esquina superior derecha de la página para iniciar el servicio.

Detener un servicio

Un servicio detenido ya no se facturará. Detener un servicio de cualquiera de las siguientes maneras:

- Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** en el panel de navegación izquierdo. Vaya a la página de gestión de servicios del servicio de destino. Haga clic en **Stop** en la columna **Operation** para detener un servicio. (Para un servicio en tiempo real, elija **More** > **Stop** en la columna **Operation**.)
- Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** en el panel de navegación izquierdo. Vaya a la página de gestión de servicios del servicio de destino. Haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio. Haga clic en **Stop** en la esquina superior derecha de la página para detener el servicio.

Eliminación de un servicio

Si un servicio ya no está en uso, elimínelo para liberar recursos.

Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** en el panel de navegación izquierdo. Vaya a la página de gestión de servicios del servicio de destino.

- Servicios en tiempo real
  - En la lista de servicios en tiempo real, seleccione **More** > **Delete** en la columna **Operation** del servicio de destino para eliminarlo.
  - Seleccione los servicios en la lista de servicios en tiempo real y haga clic en **Delete** sobre la lista para eliminar los servicios por lotes.

  - Haga clic en el nombre del servicio de destino. En la página de detalles del servicio que aparece en pantalla, haga clic en **Delete** en el extremo superior derecho para eliminar el servicio.

- Servicios por lotes
  - En la lista de servicios por lotes, haga clic en **Delete** en la columna **Operation** del servicio de destino para eliminarlo.
  - Seleccione servicios en la lista de servicios por lotes y haga clic en **Delete** arriba de la lista para eliminar servicios por lotes.
  - Haga clic en el nombre del servicio de destino. En la página de detalles del servicio que aparece en pantalla, haga clic en **Delete** en el extremo superior derecho para eliminar el servicio.

NOTA

- No se puede recuperar un servicio eliminado.
- No se puede eliminar un servicio sin autorización de la delegación.
- Si **Advanced Log Management** está habilitado para un servicio en tiempo real, se recomienda eliminar los logs y flujos de LTS al eliminar el servicio. Esto evita cargos adicionales incurridos por los logs y los flujos.

Reinicio de un servicio

Se puede reiniciar un servicio en tiempo real solo cuando el servicio se encuentra en estado **Running** o **Alarm**. Los servicios por lotes y los servicios de borde no se pueden reiniciar. Puede reiniciar un servicio en tiempo real de cualquiera de las siguientes maneras:

- Inicie sesión en la consola de gestión de ModelArts y seleccione **Service Deployment** en el panel de navegación. Vaya a la página de lista de servicios en tiempo real. Haga clic en **More** > **Restart** en la columna **Operation** para reiniciar el servicio deseado.
- Inicie sesión en la consola de gestión de ModelArts y seleccione **Service Deployment** en el panel de navegación. Vaya a la página de lista de servicios en tiempo real. Haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio. Haga clic en **Restart** en el extremo superior derecho de la página para reiniciar el servicio.

3.5 Consulta de eventos de servicio

Durante todo el ciclo de vida de un servicio, cada evento clave se registra automáticamente. Puede ver los eventos en la página de detalles del servicio en cualquier momento.

Esto permite comprender mejor el proceso de despliegue de un servicio y localizar fallas con mayor precisión cuando se produce una excepción de tarea. En la siguiente tabla se enumeran los eventos disponibles.

Tabla 3-15 Eventos

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Normal	El servicio comienza a desplegarse.	-

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Abnormal	Recursos insuficientes. Espere hasta que los recursos inactivos sean suficientes.	Espere hasta que se lance los recursos y vuelva a intentar.
Abnormal	*xxx* insuficiente. Error en la programación. Información complementaria *xxx*	Conozca los detalles de la insuficiencia de recursos según la información complementaria. Para obtener más información, véase **Preguntas frecuentes**.
Normal	La imagen comienza a crearse.	-
Abnormal	Error al crear la imagen de modelo **xxx**. Para obtener más detalles, véase logs *:\nxxx*.	Localice y rectifique la falla según los logs de compilación.
Abnormal	Error al crear la imagen.	Contacte con el servicio de asistencia técnica.
Normal	La imagen creada.	-
Abnormal	Error de servicio *xxx*. Error: *xxx*	Localice y rectifique la falla según la información del error.
Abnormal	Error al actualizar el servicio. Realice una reversión.	Contacte con el servicio de asistencia técnica.
Normal	El servicio se está actualizando.	-
Normal	El servicio se está iniciando.	-
Normal	Se está deteniendo el servicio.	-
Normal	Se detuvo el servicio.	-
Normal	Se ha deshabilitado la parada automática.	-
Normal	Se ha habilitado la parada automática. El servicio se detendrá después de los *x*.	-
Normal	El servicio se detiene cuando expira el tiempo de parada automática.	-

Tipo	Evento (se debe reemplazar *xxx* por el valor real.)	Solución
Abnormal	El servicio se detuvo porque la cuota supera el límite superior.	Contacte con el servicio de asistencia técnica.
Abnormal	Error al detener automáticamente el servicio. Error: *xxx*	Localice y rectifique la falla según la información del error.
Normal	Instancias de servicio eliminadas del grupo de recursos *xxx*.	-
Normal	Instancias de servicio detenidas en el grupo de recursos *xxx*.	-
Abnormal	Error en el servicio por lotes. Intente de nuevo más tarde.Error: *xxx*	Localice y rectifique la falla según la información del error.
Normal	El servicio se ha ejecutado.	-
Abnormal	Error al detener el servicio.Error: *xxx*	Localice y rectifique la falla según la información del error.
Normal	La licencia de suscripción *xxx* está por vencer.	-
Normal	Servicio *xxx* iniciado.	-
Abnormal	Error al iniciar el servicio *xxx*.	Para obtener detalles sobre cómo localizar y rectificar la falla, véase **Preguntas frecuentes**.
Abnormal	Tiempo fuera de despliegue de servicio.Error: *xxx*	Localice y rectifique la falla según la información del error.
Normal	Error al actualizar el servicio. La actualización se ha revertido.	-
Abnormal	Error al actualizar el servicio. Error al revertir.	Contacte con el servicio de asistencia técnica.

Durante el despliegue y la ejecución del servicio, los eventos clave se pueden actualizar manualmente y automáticamente.

Consulta de eventos

1. En el panel de navegación izquierdo de la consola de gestión de ModelArts, seleccione **Service Deployment** > **Real-Time Services** o **Batch Services** o **Edge Services**. En la lista de servicios, haga clic en el nombre o ID del servicio de destino para ir a su página de detalles.
2. Vea los eventos en la pestaña **Events**.

4 Especificaciones de inferencia

4.1 Especificaciones del paquete de modelo

4.1.1 Introducción a las especificaciones del paquete modelo

Al crear una aplicación de IA en la página de gestión de aplicaciones de IA, asegúrese de que cualquier metamodelo importado de OBS cumpla con ciertas especificaciones.

NOTA

- Las especificaciones del paquete modelo se utilizan cuando se importa un modelo. Si importa varios modelos, por ejemplo, hay varios archivos de modelo, utilice imágenes personalizadas.
- Si desea utilizar un motor de IA que no sea compatible con ModelArts utilice una imagen personalizada.
- Para obtener más información sobre cómo crear una imagen personalizada, véase **Especificaciones de imagen personalizadas para crear aplicaciones de IA** y **Creación de una imagen personalizada y usarla para crear una aplicación de IA** .
- Para obtener más ejemplos de scripts personalizados, véase **Ejemplos de scripts personalizados**.

El paquete de modelo debe contener el directorio **model**. El directorio **model** almacena el archivo de modelo, el archivo de configuración de modelo y el archivo de código de inferencia de modelo.

- **Model files:** Los requisitos para los archivos de modelo varían según la estructura del paquete del modelo. Para más detalles, véase **Ejemplo de paquete de modelo**.
- **Model configuration file**: El archivo de configuración del modelo debe estar disponible y su nombre debe ser **config.json**. Solo hay que existirse un archivo de configuración del modelo. Para obtener más información sobre cómo editar un archivo de configuración de modelo, véase **Especificaciones para editar un archivo de configuración de modelo**.
- **Model inference code file**: Obligatorio. El nombre del archivo debe ser **customize_service.py** de forma consistente. Solo hay que existirse un archivo de código de inferencia de modelo. Para obtener más información sobre cómo editar el código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.

- El archivo **.py** del que **customize_service.py** depende puede almacenarse directamente en el directorio **model**. Utilice un modo de importación relativa para importar el paquete personalizado.
- Los otros archivos de los que depende **customize_service.py** se pueden almacenar en el directorio **model**. Debe utilizar rutas de acceso absolutas para tener acceso a estos archivos. Para obtener más detalles, véase **Obtención de una ruta absoluta**.

ModelArts proporciona ejemplos y código de ejemplo para varios motores. Puede compilar los archivos de configuración y el código de inferencia consultando **Ejemplos de ModelArts**. ModelArts también proporciona ejemplos de script personalizados de motores de IA comunes. Para obtener más información, véase **Ejemplos de scripts personalizados**.

Si encuentra algún problema al importar un metamodelo, **póngase en contacto con el soporte técnico de Huawei Cloud**.

Ejemplo de paquete de modelo

- Estructura del paquete de modelo basado en TensorFlow

  Al publicar el modelo, solo necesita especificar el directorio **ocr**.

```
OBS bucket/directory name
|── ocr
|   ├── model (Mandatory) Name of a fixed subdirectory, which is used to
store model-related files
|   |   ├── <<Custom Python package>> (Optional) User's Python package, which
can be directly referenced in model inference code
|   |   ├── saved_model.pb (Mandatory) Protocol buffer file, which contains
the diagram description of the model
|   |   ├── variables Name of a fixed sub-directory, which contains the
weight and deviation rate of the model. It is mandatory for the main file of
the *.pb model.
|   |   |   ├── variables.index Mandatory
|   |   |   ├── variables.data-00000-of-00001 Mandatory
|   |   ├──config.json (Mandatory) Model configuration file. The file name is
fixed to config.json. Only one model configuration file is supported.
|   |   ├──customize_service.py (Mandatory) Model inference code. The file
name is fixed to customize_service.py. Only one model inference code file
exists.
The files on which customize_service.py depends can be directly stored in the
model directory.
```

- Estructura del paquete de modelo basado en PyTorch

  Al publicar el modelo, solo necesita especificar el directorio **resnet**.

```
OBS bucket/directory name
|── resnet
|   ├── model (Mandatory) Name of a fixed subdirectory, which is used to
store model-related files
|   |   ├── <<Custom Python package>> (Optional) User's Python package, which
can be directly referenced in model inference code
|   |   ├── resnet50.pth (Mandatory) PyTorch model file, which contains
variable and weight information and is saved as state_dict
|   |   ├──config.json (Mandatory) Model configuration file. The file name is
fixed to config.json. Only one model configuration file is supported.
|   |   ├──customize_service.py (Mandatory) Model inference code. The file
name is fixed to customize_service.py. Only one model inference code file
exists. The files on which customize_service.py depends can be directly
stored in the model directory.
```

- La estructura de un paquete de modelos personalizados depende del motor de IA de la imagen personalizada. Por ejemplo, si TensorFlow es el motor de IA de la imagen personalizada, el paquete de modelo utiliza la estructura de TensorFlow.

4.1.2 Especificaciones para editar un archivo de configuración de modelo

Un desarrollador de modelos necesita editar un archivo de configuración **config.json** al publicar un modelo. El archivo de configuración del modelo describe el uso del modelo, el marco de cómputo, la precisión, el paquete de dependencia del código de inferencia y la API del modelo.

Formato de archivo de configuración

El archivo de configuración está en formato JSON. **Tabla 4-1** describe los parámetros.

Tabla 4-1 Parámetros

Parámetro	Obligatorio	Tipo de datos	Descripción
model_algorithm	Sí	String	Algoritmo del modelo, que es establecido por el desarrollador del modelo para ayudar a los usuarios del modelo a entender el uso del modelo. El valor debe comenzar con una letra y no contener más de 36 caracteres. Los caracteres chinos y los caracteres especiales (&!'\"<>=) no están permitidos. Los algoritmos de modelos comunes incluyen **image_classification** (clasificación de imágenes), **object_detection** (detección de objetos) y **predict_analysis** (análisis de predicción).
model_type	Sí	String	Modelo de motor de IA, que indica el marco de cálculo utilizado por un modelo. Los motores comunes de IA e **Image** son compatibles. ● Para obtener más información sobre los motores de IA compatibles, consulte **Motores de IA compatibles para la inferencia de ModelArts**. ● Si **model_type** se establece en **Image**, la aplicación de IA se crea mediante una imagen personalizada. En este caso, el parámetro **swr_location** es obligatorio. Para obtener más información sobre las especificaciones de imágenes personalizadas, véase **Especificaciones de imágenes personalizadas para crear aplicaciones de IA**.

Parámetro	Obligatorio	Tipo de datos	Descripción
runtime	No	String	Entorno de tiempo de ejecución del modelo. Python2.7 se usa por defecto. El valor de **runtime** depende del valor de **model_type**. Si **model_type** se establece en **Image**, no es necesario establecer **runtime**. Si **model_type** se establece en otro marco estándar, seleccione el motor y el entorno de tiempo de ejecución. Para obtener detalles sobre los entornos de ejecución soportados, véase **Motores de IA compatibles para la inferencia de ModelArts**. Si el modelo debe ejecutarse en CPU o GPU especificadas, seleccione las CPU o GPU en función del sufijo de tiempo de ejecución. Si el tiempo de ejecución no contiene la información de la CPU o la GPU, compruebe la description del tiempo de ejecución en *Motores de IA compatibles para Inferencia de ModelArts*.
metrics	No	Object	Información de precisión del modelo, incluido el valor promedio, la tasa de recuperación, la precisión y la precisión. Para obtener más información sobre la estructura de objeto **metrics**, consulte **Tabla 4-2**. El resultado se muestra en el área de precisión del modelo de la página de detalles de la aplicación de IA.
apis	No	api array	Formato de las solicitudes recibidas y devueltas por un modelo. El valor es dato de estructura. Es la matriz de API de RESTful proporcionada por un modelo. Para obtener más información sobre la estructura de datos de la API, consulte **Tabla 4-3**. Para obtener más información sobre el ejemplo de código, véase **Ejemplo de código de parámetros apis**. ● Si **model_type** se establece en Imagen, la aplicación de IA se crea con una imagen personalizada. ● Cuando **model_type** no es **Image**, solo una API cuya ruta de solicitud es / puede declararse en **apis** porque el motor de IA preconfigurado expone solo una API de inferencia cuya ruta de solicitud es /.

Parámetro	Obligatorio	Tipo de datos	Descripción
dependencies	No	dependency array	Paquete del que depende el código de inferencia del modelo, que son datos de estructura. Los desarrolladores de modelos deben proporcionar el nombre del paquete, el modo de instalación y las restricciones de versión. Solo se soporta el modo de instalación de pip. **Tabla 4-6** describe la matriz de dependencias. Si el paquete modelo no contiene el archivo **customize_service.py**, no es necesario establecer este parámetro. No se pueden instalar paquetes de dependencias para modelos de imágenes personalizados. **NOTA** El parámetro **dependencies** admite múltiples matrices de estructura de dependencias en formato de lista y se aplica a escenarios en los que los paquetes de instalación tienen relaciones de dependencia. Los paquetes de la parte superior se instalan primero. El paquete de ruedas en las instalaciones se puede utilizar para la instalación. (El paquete de rueda debe almacenarse en el mismo directorio que el archivo de modelo). Para obtener más información, véase **¿Cómo edito los parámetros de dependencia del paquete de instalación en un archivo de configuración de modelo al importar un modelo?**
health	No	**health** data structure	Configuración de una interfaz de mantenimiento de imagen. Este parámetro solo es obligatorio cuando **model_type** se establece en **Image**. Si los servicios no se pueden interrumpir durante una actualización sucesiva, se debe proporcionar una API de comprobación de estado para que ModelArts invoque. Para obtener más información sobre la estructura de datos de mantenimiento, véase **Tabla 4-8**.

Tabla 4-2 Descripción del objeto **metrics**

Parámetro	Obligatorio	Tipo de datos	Descripción
f1	No	Number	Puntuación de F1. El valor se redondea a 17 decimales.
recall	No	Number	Tasa de rellamada. El valor se redondea a 17 decimales.
precision	No	Number	Precisión. El valor se redondea a 17 decimales.

Parámetro	Obligatorio	Tipo de datos	Descripción
accuracy	No	Number	Exactitud. El valor se redondea a 17 decimales.

Tabla 4-3 Matriz de **api**

Parámetro	Obligatorio	Tipo de datos	Descripción
url	No	String	Ruta de solicitud. El valor predeterminado es una barra diagonal (/). Para un modelo de imagen personalizado (**model_type** es **Image**), establezca este parámetro en la ruta de solicitud real expuesta en la imagen. Para un modelo de imagen no personalizado (**model_type** no es **Image**), la URL solo puede ser /.
method	No	String	Método de solicitud. El valor predeterminado es **POST**.
request	No	Object	Cuerpo de la solicitud. Para más detalles, véase **Tabla 4-4**.
response	No	Object	Cuerpo de respuesta. Para más detalles, véase **Tabla 4-5**.

Tabla 4-4 Descripción de **request**

Parámetro	Obligatorio	Tipo de datos	Descripción
Content-type	No para servicios en tiempo real Sí para servicios por lotes	String	Los datos se envían en un formato de contenido especificado. El valor predeterminado es **application/json**. Las opciones son las siguientes: ● **application/json**: Se cargan los datos de JSON. ● **multipart/form-data**: Se ha cargado un archivo. **NOTA** Para los modelos de aprendizaje automático, solo se admite **application/json**.

Parámetro	Obligatorio	Tipo de datos	Descripción
data	No para servicios en tiempo real Sí para servicios por lotes	String	El cuerpo de la solicitud se describe en el esquema JSON. Para obtener más información sobre la descripción de los parámetros, consulte la **guía oficial**.

Tabla 4-5 Descripción de **response**

Parámetro	Obligatorio	Tipo de datos	Descripción
Content-type	No para servicios en tiempo real Sí para servicios por lotes	String	Los datos se envían en un formato de contenido especificado. El valor predeterminado es **application/json**. **NOTA** Para los modelos de aprendizaje automático, solo se admite **application/json**.
data	No para servicios en tiempo real Sí para servicios por lotes	String	El cuerpo de la respuesta se describe en el esquema JSON. Para obtener más información sobre la descripción de los parámetros, consulte la **guía oficial**.

Tabla 4-6 Matriz de **dependency**

Parámetro	Obligatorio	Tipo de datos	Descripción
installer	Sí	String	Método de instalación. Solo se admite **pip**.
packages	Sí	package array	Recolección de paquetes de dependencia. Para obtener más información sobre la matriz de estructura de paquetes, consulte **Tabla 4-7**.

Tabla 4-7 Matriz de package

Parámetro	Obligatorio	Tipo	Descripción
package_name	Sí	String	Nombre del paquete de dependencia. Los caracteres chinos y los caracteres especiales (&!'"<>=) no están permitidos.
package_version	No	String	Versión del paquete de dependencia. Si el paquete de dependencias no depende de las versiones del paquete, deje este campo en blanco. Los caracteres chinos y los caracteres especiales (&!'"<>=) no están permitidos.
restraint	No	String	Restricción de versión. Este parámetro es obligatorio solo cuando se configura **package_version**. Los valores posibles son **EXACT**, **ATLEAST** y **ATMOST**. ● **EXACT** indica que se ha instalado una versión especificada. ● **ATLEAST** indica que la versión del paquete de instalación no es anterior a la versión especificada. ● **ATMOST** indica que la versión del paquete de instalación no es posterior a la versión especificada. **NOTA** ● Si hay requisitos específicos en la versión, utilice **EXACT** preferentemente. Si **EXACT** entra en conflicto con los paquetes de instalación del sistema, puede seleccionar **ATLEAST**. ● Si no hay ningún requisito específico en la versión, conserve solo el parámetro **package_name** y deje **restraint** y **package_version** en blanco.

Tabla 4-8 Descripción de la estructura de datos **health**

Parámetro	Obligatorio	Tipo	Descripción
check_method	Sí	String	Método de comprobación de estado. El valor puede ser **HTTP** o **EXEC**. ● **HTTP**: Utilice una solicitud de HTTP. ● **EXEC**: Ejecute un comando.
command	No	String	Comando de control de estado. Este parámetro es obligatorio cuando **check_method** se establece en **EXEC**.
url	No	String	URL de solicitud de una API de comprobación de estado. Este parámetro es obligatorio cuando **check_method** se establece en **HTTP**.
protocol	No	String	Protocolo de solicitud de una API de comprobación de estado. El valor predeterminado es **http**. Este parámetro es obligatorio cuando **check_method** se establece en **HTTP**.
initial_delay_seconds	No	String	Retraso en la inicialización de la comprobación de estado.
timeout_seconds	No	String	Tiempo de espera de comprobación de estado.
period_seconds	Sí	String	Período de comprobación de estado, en segundos. Ingrese un número entero mayor que 0 y no mayor que 2147483647.
failure_threshold	Sí	String	Número máximo de errores de comprobación de estado. Ingrese un número entero mayor que 0 y no mayor que 2147483647.

Ejemplo de código de parámetros apis

```
[{
    "url": "/",
    "method": "post",
    "request": {
        "Content-type": "multipart/form-data",
        "data": {
            "type": "object",
            "properties": {
                "images": {
                    "type": "file"
```



```
                }
            }
        }
    },
    "response": {
        "Content-type": "applicaton/json",
        "data": {
            "type": "object",
            "properties": {
                "mnist_result": {
                    "type": "array",
                    "item": [
                        {
                            "type": "string"
                        }
                    ]
                }
            }
        }
    }
}]
```

Ejemplo del archivo de configuración del modelo de detección de objetos

El siguiente código utiliza el motor TensorFlow como ejemplo. Puede modificar el parámetro **model_type** en función del tipo de motor real.

- Entrada de modelo

  Key: **images**

  Value: **image files**

- Salida del modelo

```
{
    "detection_classes": [
        "face",
        "arm"
    ],
    "detection_boxes": [
        [
            33.6,
            42.6,
            104.5,
            203.4
        ],
        [
            103.1,
            92.8,
            765.6,
            945.7
        ]
    ],
    "detection_scores": [0.99, 0.73]
}
```

- Archivo de configuración

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "object_detection",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "url": "/",
        "method": "post",
```



```
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "application/json",
            "data": {
                "type": "object",
                "properties": {
                    "detection_classes": {
                        "type": "array",
                        "items": [{
                            "type": "string"
                        }]
                    },
                    "detection_boxes": {
                        "type": "array",
                        "items": [{
                            "type": "array",
                            "minItems": 4,
                            "maxItems": 4,
                            "items": [{
                                "type": "number"
                            }]
                        }]
                    },
                    "detection_scores": {
                        "type": "array",
                        "items": [{
                            "type": "number"
                        }]
                    }
                }
            }
        }
    }],
    "dependencies": [{
        "installer": "pip",
        "packages": [{
                "restraint": "EXACT",
                "package_version": "1.15.0",
                "package_name": "numpy"
            },
            {
                "restraint": "EXACT",
                "package_version": "5.2.0",
                "package_name": "Pillow"
            }
        ]
    }]
}
```

Ejemplo del archivo de configuración del modelo de clasificación de imágenes

El siguiente código utiliza el motor TensorFlow como ejemplo. Puede modificar el parámetro **model_type** en función del tipo de motor real.

- Entrada de modelo

  Clave: images

  Valor: image files

- Salida del modelo

```
{
    "predicted_label": "flower",
    "scores": [
       ["rose", 0.99],
       ["begonia", 0.01]
    ]
}
```

- Archivo de configuración

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "image_classification",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "application/json",
            "data": {
                "type": "object",
                "properties": {
                    "predicted_label": {
                        "type": "string"
                    },
                    "scores": {
                        "type": "array",
                        "items": [{
                            "type": "array",
                            "minItems": 2,
                            "maxItems": 2,
                            "items": [
                                {
                                    "type": "string"
                                },
                                {
                                    "type": "number"
                                }
                            ]
                        }]
                    }
                }
            }
        }
    }],
    "dependencies": [{
        "installer": "pip",
        "packages": [{
                "restraint": "ATLEAST",
                "package_version": "1.15.0",
                "package_name": "numpy"
            },
            {
                "restraint": "",
```



```
                "package_version": "",
                "package_name": "Pillow"
            }
        ]
    }]
}
```

El código siguiente utiliza el motor de MindSpore como ejemplo. Puede modificar el parámetro **model_type** según el tipo de motor que utilice.

- Entrada de modelo

  Key: **images**

  Value: **image files**

- Salida del modelo

```
"[[-2.404526   -3.0476532  -1.9888215   0.45013925 -1.7018927   0.40332815\n
-7.1861157  11.290332   -1.5861531   5.7887416 ]]"
```

- Archivo de configuración

```
{
     "model_algorithm": "image_classification",
     "model_type": "MindSpore",
     "metrics": {
         "f1": 0.124555,
         "recall": 0.171875,
         "precision": 0.0023493892851938493,
         "accuracy": 0.00746268656716417
     },
     "apis": [{
             "url": "/",
             "method": "post",
             "request": {
                 "Content-type": "multipart/form-data",
                 "data": {
                     "type": "object",
                     "properties": {
                         "images": {
                             "type": "file"
                         }
                     }
                 }
             },
             "response": {
                 "Content-type": "applicaton/json",
                 "data": {
                     "type": "object",
                     "properties": {
                         "mnist_result": {
                             "type": "array",
                             "item": [{
                                 "type": "string"
                             }]
                         }
                     }
                 }
             }
         }
     ],
    "dependencies": []
 }
```

Ejemplo del archivo de configuración del modelo de análisis predictivo

El siguiente código utiliza el motor TensorFlow como ejemplo. Puede modificar el parámetro **model_type** en función del tipo de motor real.

- Entrada de modelo

```
{
    "data": {
        "req_data": [
            {
                "buying_price": "high",
                "maint_price": "high",
                "doors": "2",
                "persons": "2",
                "lug_boot": "small",
                "safety": "low",
                "acceptability": "acc"
            },
            {
                "buying_price": "high",
                "maint_price": "high",
                "doors": "2",
                "persons": "2",
                "lug_boot": "small",
                "safety": "low",
                "acceptability": "acc"
            }
        ]
    }
}
```

- Salida del modelo

```
{
    "data": {
        "resp_data": [
            {
                "predict_result": "unacc"
            },
            {
                "predict_result": "unacc"
            }
        ]
    }
}
```

- Archivo de configuración

  NOTA

  En el código, el parámetro **data** de las estructuras de solicitud y de respuesta se describe en el JSON Schema. El contenido de **data** y **properties** corresponde a la entrada y salida del modelo.

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "predict_analysis",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [
        {
            "url": "/",
            "method": "post",
            "request": {
                "Content-type": "application/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "data": {
                            "type": "object",
                            "properties": {
                                "req_data": {
                                    "items": [
```



```
                                        {
                                            "type": "object",
                                            "properties": {}
                                        }
                                    ],
                                    "type": "array"
                                }
                            }
                        }
                    }
                },
                "response": {
                    "Content-type": "application/json",
                    "data": {
                        "type": "object",
                        "properties": {
                            "data": {
                                "type": "object",
                                "properties": {
                                    "resp_data": {
                                        "type": "array",
                                        "items": [
                                            {
                                                "type": "object",
                                                "properties": {}
                                            }
                                        ]
                                    }
                                }
                            }
                        }
                    }
                }
            }
        ],
        "dependencies": [
            {
                "installer": "pip",
                "packages": [
                    {
                        "restraint": "EXACT",
                        "package_version": "1.15.0",
                        "package_name": "numpy"
                    },
                    {
                        "restraint": "EXACT",
                        "package_version": "5.2.0",
                        "package_name": "Pillow"
                    }
                ]
            }
        ]
}
```

Ejemplo del archivo de configuración del modelo de imagen personalizado

Las entradas y salidas del modelo son similares a las de **Ejemplo del archivo de configuración del modelo de detección de objetos**.

- Si la entrada es una imagen, el ejemplo de solicitud es el siguiente.

  En el ejemplo, se recibe una solicitud de predicción de modelo que contiene el parámetro **images** con el tipo de parámetro de **file**. En este ejemplo, el botón de carga de archivos se muestra en la página de inferencia y la inferencia se realiza en formato de archivo.

  ```
  {
      "Content-type": "multipart/form-data",
  ```



```
    "data": {
        "type": "object",
        "properties": {
            "images": {
                "type": "file"
            }
        }
    }
}
```

- Si la entrada es datos JSON, el ejemplo de solicitud es el siguiente.

  En este ejemplo, se recibe el cuerpo de solicitud JSON de predicción de modelo. En la solicitud, solo hay una solicitud de predicción que contiene el parámetro **input** con el tipo de parámetro de string. En la página de inferencia, se muestra un cuadro de texto para que introduzca la solicitud de predicción.

  ```
  {
      "Content-type": "application/json",
      "data": {
          "type": "object",
          "properties": {
              "input": {
                  "type": "string"
              }
          }
      }
  }
  ```

Un ejemplo de solicitud completo es el siguiente:

```
{
    "model_algorithm": "image_classification",
    "model_type": "Image",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [{
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "application/json",
            "data": {
                "type": "object",
                "required": [
                    "predicted_label",
                    "scores"
                ],
                "properties": {
                    "predicted_label": {
                        "type": "string"
                    },
                    "scores": {
                        "type": "array",
                        "items": [{
                            "type": "array",
```



```
                                "minItems": 2,
                                "maxItems": 2,
                                "items": [{
                                        "type": "string"
                                    },
                                    {
                                        "type": "number"
                                    }
                                ]
                            }]
                        }
                    }
                }
            }
        }]
}
```

Ejemplo del archivo de configuración del modelo de aprendizaje automático

El siguiente ejemplo utiliza XGBoost:

- Entrada de modelo

```
{
    "req_data": [
        {
            "sepal_length": 5,
            "sepal_width": 3.3,
            "petal_length": 1.4,
            "petal_width": 0.2
        },
        {
            "sepal_length": 5,
            "sepal_width": 2,
            "petal_length": 3.5,
            "petal_width": 1
        },
        {
            "sepal_length": 6,
            "sepal_width": 2.2,
            "petal_length": 5,
            "petal_width": 1.5
        }
    ]
}
```

- Salida del modelo

```
{
    "resp_data": [
        {
            "predict_result": "Iris-setosa"
        },
        {
            "predict_result": "Iris-versicolor"
        }
    ]
}
```

- Archivo de configuración

```
{
    "model_type": "XGBoost",
    "model_algorithm": "xgboost_iris_test",
    "runtime": "python2.7",
    "metrics": {
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
```



```
    "apis": [
        {
            "url": "/",
            "method": "post",
            "request": {
                "Content-type": "application/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "req_data": {
                            "items": [
                                {
                                    "type": "object",
                                    "properties": {}
                                }
                            ],
                            "type": "array"
                        }
                    }
                }
            },
            "response": {
                "Content-type": "applicaton/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "resp_data": {
                            "type": "array",
                            "items": [
                                {
                                    "type": "object",
                                    "properties": {
                                        "predict_result": {}
                                    }
                                }
                            ]
                        }
                    }
                }
            }
        }
    ]
}
```

Ejemplo de un archivo de configuración de modelo que utiliza un paquete de dependencia personalizado

En el siguiente ejemplo se define el entorno de dependencia NumPy 1.16.4.

```
{
    "model_algorithm": "image_classification",
    "model_type": "TensorFlow",
    "runtime": "python3.6",
    "apis": [
        {
            "url": "/",
            "method": "post",
            "request": {
                "Content-type": "multipart/form-data",
                "data": {
                    "type": "object",
                    "properties": {
                        "images": {
                            "type": "file"
                        }
                    }
                }
```



```
            },
            "response": {
                "Content-type": "applicaton/json",
                "data": {
                    "type": "object",
                    "properties": {
                        "mnist_result": {
                            "type": "array",
                            "item": [
                                {
                                    "type": "string"
                                }
                            ]
                        }
                    }
                }
            }
        }
    ],
    "metrics": {
        "f1": 0.124555,
        "recall": 0.171875,
        "precision": 0.00234938928519385,
        "accuracy": 0.00746268656716417
    },
    "dependencies": [
        {
            "installer": "pip",
            "packages": [
                {
                    "restraint": "EXACT",
                    "package_version": "1.16.4",
                    "package_name": "numpy"
                }
            ]
        }
    ]
}
```


4.1.3 Especificaciones para escribir el código de inferencia de modelo

Esta sección describe el método general de edición de código de inferencia de modelo de ModelArts. Para obtener detalles sobre los ejemplos de script personalizados (incluidos ejemplos de código de inferencia) de los principales motores de IA, véase **Ejemplos de scripts personalizados**. Esta sección también proporciona un ejemplo de código de inferencia para el motor de TensorFlow y un ejemplo de personalización de la lógica de inferencia en el script de inferencia.

Debido a la limitación de API Gateway, la duración de una sola predicción en ModelArts no puede superar los 40 segundos. El código de inferencia del modelo debe ser lógicamente claro y conciso para un rendimiento de inferencia satisfactorio.

Especificaciones para compilar código de inferencia

1. En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo. Esta clase de modelo hijo hereda las propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.

Tabla 4-9 Importar instrucciones de diferentes tipos de clases de modelo padre

Tipo de modelo	Clase primaria	Instrucción de importación
TensorFlow	TfServingBaseService	from model_service.tfserving_model_service import TfServingBaseService
PyTorch	PTServingBaseService	from model_service.pytorch_model_service import PTServingBaseService
MindSpore	SingleNodeService	from model_service.model_service import SingleNodeService

2. Se pueden reescribir los siguientes métodos:

Tabla 4-10 Métodos a reescribir

Método	Descripción
__init__(self, model_name, model_path)	Método de inicialización, que es adecuado para modelos creados basados en marcos de aprendizaje profundo. Los modelos y las etiquetas se cargan con este método. Este método debe ser reescrito para modelos basados en PyTorch y Caffe para implementar la lógica de carga del modelo.
__init__(self, model_path)	Método de inicialización, que es adecuado para modelos creados basados en marcos de aprendizaje automático. La ruta del modelo (**self.model_path**) se inicializa usando este método. En Spark_MLlib, este método también inicializa SparkSession (**self.spark**).
_preprocess(self, data)	Método de preproceso, que se llama antes de una solicitud de inferencia y se usa para convertir los datos de solicitud originales de una API en los datos de entrada esperados de un modelo
_inference(self, data)	Método de solicitud de inferencia. No se recomienda volver a escribir el método porque una vez que se haya reescrito, el proceso de inferencia integrado de ModelArts se sobrescribirá y se ejecutará la lógica de inferencia personalizada.
_postprocess(self, data)	Método de posprocesamiento, que se llama después de completar una solicitud de inferencia y se utiliza para convertir la salida del modelo en la salida de la API

NOTA

- Puede elegir reescribir los métodos de preproceso y postproceso para implementar el preprocesamiento de la entrada de API y el postprocesamiento de la salida de inferencia.
- La reescritura del método init de la clase de modelo padre puede hacer que una aplicación de IA se ejecute de forma anormal.

3. El atributo que se puede utilizar es la ruta local donde reside el modelo. El nombre del atributo es **self.model_path**. Además, los modelos basados en PySpark pueden usar **self.spark** para obtener el objeto de SparkSession en **customize_service.py**.

   NOTA

   Se requiere una ruta absoluta para leer archivos en el código de inferencia. Puede obtener la ruta local del modelo desde el atributo **self.model_path**.

   - Cuando se utiliza TensorFlow, Caffe o MXNet, **self.model_path** indica la ruta del archivo de modelo. Vea el siguiente ejemplo:

     ```
     # Store the label.json file in the model directory. The following information is read:
     with open(os.path.join(self.model_path, 'label.json')) as f:
         self.label = json.load(f)
     ```

   - Cuando se utiliza PyTorch o PySpark, **self.model_path** indica la ruta del archivo de modelo. Vea el siguiente ejemplo:

     ```
     # Store the label.json file in the model directory. The following information is read:
     dir_path = os.path.dirname(os.path.realpath(self.model_path))
     with open(os.path.join(dir_path, 'label.json')) as f:
         self.label = json.load(f)
     ```

4. **data** importados a través de la API para el procesamiento previo, la solicitud de inferencia real y el procesamiento posterior pueden ser **multipart/form-data** o **application/json**.

   – Solicitud de **multipart/form-data**

     ```
     curl -X POST \
       <modelarts-inference-endpoint> \
       -F image1=@cat.jpg \
       -F images2=@horse.jpg
     ```

     Los datos de entrada correspondientes son los siguientes:

     ```
     [
        {
           "image1":{
              "cat.jpg":"<cat.jpg file io>"
           }
        },
        {
           "image2":{
              "horse.jpg":"<horse.jpg file io>"
           }
        }
     ]
     ```

   – Solicitud de **application/json**

     ```
     curl -X POST \
       <modelarts-inference-endpoint> \
       -d '{
        "images":"base64 encode image"
        }'
     ```

     El dato de entrada correspondiente es **python dict**.

     ```
     {
        "images":"base64 encode image"
     }
     ```

Ejemplo de script de inferencia de TensorFlow

A continuación se muestra un ejemplo de TensorFlow MnistService. Para obtener más ejemplos de código de inferencia de TensorFlow, véase **TensorFlow** y **TensorFlow 2.1**.Para obtener más detalles sobre el código de inferencia de otros motores, véase **PyTorch** y **Caffe**.

- Código de inferencia


```
from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):

    def _preprocess(self, data):
        preprocessed_data = {}

        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1, 784))
                preprocessed_data[k] = image1

        return preprocessed_data

    def _postprocess(self, data):

        infer_output = {}

        for output_name, result in data.items():

            infer_output["mnist_result"] = result[0].index(max(result[0]))

        return infer_output
```

- Solicitud

```
curl -X POST \ Real-time service address \ -F images=@test.jpg
```

- Respuesta

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

El ejemplo de código anterior cambia el tamaño de las imágenes importadas al formulario del usuario para adaptarse a la forma de entrada del modelo. La imagen **32×32** se lee de la biblioteca de Pillow y se cambia de tamaño a **1×784** para que coincida con la entrada del modelo. En el procesamiento posterior, convierta la salida del modelo en una lista para mostrar la API RESTful.

Ejemplo de Script de Inferencia XGBoost

Para obtener más información sobre el código de inferencia de otros motores de aprendizaje automático, consulte **PySpark** y **Aprendizaje de Scikit**.

```
# coding:utf-8
import collections
import json
import xgboost as xgb
from model_service.python_model_service import XgSklServingBaseService


class UserService(XgSklServingBaseService):

    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:
            array = []
            for each in element:
                array.append(element[each])
                list_data.append(array)
        return list_data

    #   predict
    def _inference(self, data):
        xg_model = xgb.Booster(model_file=self.model_path)
```



```
        pre_data = xgb.DMatrix(data)
        pre_result = xg_model.predict(pre_data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self, data):
        resp_data = []
        for element in data:
            resp_data.append({"predict_result": element})
        return resp_data
```

Ejemplo de script de inferencia de la lógica de inferencia personalizada

Consulte **Ejemplo de un archivo de configuración de modelo que utiliza un paquete de dependencia personalizado** para personalizar un paquete de dependencias en el archivo de configuración. A continuación, utilice el siguiente ejemplo de código para cargar el modelo en formato **saved_model** para la inferencia.

NOTA

El módulo de log de Python utilizado por la imagen de inferencia base utiliza el nivel de log predeterminado Warning. Por defecto, solo se pueden consultar los logs de warning. Para consultar logs de INFO, configure el nivel de log como INFO en el código.

```
# -*- coding: utf-8 -*-
import json
import os
import threading
import numpy as np
import tensorflow as tf
from PIL import Image
from model_service.tfserving_model_service import TfServingBaseService
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %
(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model_inputs = {}
        self.model_outputs = {}

       # The label file can be loaded here and used in the post-processing
function.
        # Directories for storing the label.txt file on OBS and in the model
package

        # with open(os.path.join(self.model_path, 'label.txt')) as f:
        #     self.label = json.load(f)

        # Load the model in saved_model format in non-blocking mode to prevent
blocking timeout.
        thread = threading.Thread(target=self.get_tf_sess)
        thread.start()

    def get_tf_sess(self):
        # Load the model in saved_model format.
       # The session will be reused. Do not use the with statement.
        sess = tf.Session(graph=tf.Graph())
        meta_graph_def = tf.saved_model.loader.load(sess,
[tf.saved_model.tag_constants.SERVING], self.model_path)
        signature_defs = meta_graph_def.signature_def
        self.sess = sess
        signature = []
```



```
        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)
        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logger.warning("signatures more than one, use serving_default
signature")
            model_signature =
tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY

        logger.info("model signature: %s", model_signature)

        for signature_name in
meta_graph_def.signature_def[model_signature].inputs:
            tensorinfo =
meta_graph_def.signature_def[model_signature].inputs[signature_name]
            name = tensorinfo.name
            op = self.sess.graph.get_tensor_by_name(name)
            self.model_inputs[signature_name] = op

        logger.info("model inputs: %s", self.model_inputs)

        for signature_name in
meta_graph_def.signature_def[model_signature].outputs:
            tensorinfo =
meta_graph_def.signature_def[model_signature].outputs[signature_name]
            name = tensorinfo.name
            op = self.sess.graph.get_tensor_by_name(name)
            self.model_outputs[signature_name] = op

        logger.info("model outputs: %s", self.model_outputs)

    def _preprocess(self, data):
        # Two request modes using HTTPS
        # 1. The request in form-data file format is as follows: data = {"Request
key value":{"File name":<File io>}}
       # 2. Request in JSON format is as follows: data = json.loads("JSON body
transferred by the API")
        preprocessed_data = {}

        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1, 28, 28))
                preprocessed_data[k] = image1

        return preprocessed_data

    def _inference(self, data):
        feed_dict = {}
        for k, v in data.items():
            if k not in self.model_inputs.keys():
                logger.error("input key %s is not in model inputs %s", k,
list(self.model_inputs.keys()))
                raise Exception("input key %s is not in model inputs %s" % (k,
list(self.model_inputs.keys())))
            feed_dict[self.model_inputs[k]] = v

        result = self.sess.run(self.model_outputs, feed_dict=feed_dict)
        logger.info('predict result : ' + str(result))
        return result

    def _postprocess(self, data):
        infer_output = {"mnist_result": []}
        for output_name, results in data.items():
```



```
            for result in results:
                infer_output["mnist_result"].append(np.argmax(result))

        return infer_output

    def __del__(self):
        self.sess.close()
```

NOTA

Para cargar modelos que no son compatibles con ModelArts ni con varios modelos, especifique la ruta de carga mediante el método **__init__**. Código de ejemplo:

```
# -*- coding: utf-8 -*-
import os
from model_service.tfserving_model_service import TfServingBaseService

class MnistService(TfServingBaseService):
    def __init__(self, model_name, model_path):
        # Obtain the path to the model folder.
        root = os.path.dirname(os.path.abspath(__file__))
        # test.onnx is the name of the model file to be loaded and must be
stored in the model folder.
        self.model_path = os.path.join(root, test.onnx)
        # Loading multiple models, for example, test2.onnx
        # self.model_path2 = os.path.join(root, test2.onnx)
```

4.2 Plantillas de modelo

4.2.1 Introducción a las plantillas de modelo

NOTA

La importación de un modelo desde una plantilla dejará de estar disponible en breve. Después de desconectarse, puede usar las plantillas para el motor de IA y las configuraciones de modelo eligiendo **OBS**, configurando **AI Engine** en **Custom** e importando su motor de IA personalizado.

Debido a que las configuraciones de plantillas con las mismas funciones son similares, el ModelArts integra las configuraciones de tales plantillas en una plantilla común. Con esta plantilla, puede importar modelos y crear aplicaciones de IA de forma fácil y rápida sin escribir el archivo de configuración **config.json**. En términos simples, una plantilla integra el motor de IA y las configuraciones del modelo. Cada plantilla corresponde a un motor de IA específico y a un modo de inferencia. Con las plantillas, puede importar rápidamente modelos a ModelArts y crear aplicaciones de IA.

Antecedentes

Las plantillas incluyen plantillas generales y no generales.

- Las plantillas no generales se personalizan para escenarios específicos con el modo de entrada y salida fijo. Por ejemplo, la **TensorFlow-based image classification template** utiliza el modo de procesamiento de imágenes integrado.
- Las plantillas generales integran un motor de IA y un entorno de ejecución específicos y utilizan el modo de entrada y salida no definido. Seleccione un modo de entrada y salida basado en la función de modelo o el escenario de aplicación para sobrescribir el modo no definido. Por ejemplo, un modelo de clasificación de imágenes requiere el modo de procesamiento de imágenes integrado, y un modelo de detección de objetos requiere el modo de detección de objetos integrado.

NOTA

Los modelos importados en modo indefinido no se pueden desplegar como servicios por lotes.

Mediante una plantilla

Cargue el paquete de plantilla en OBS antes de usar la plantilla. Almacene los archivos de modelo en el directorio **model**. Al crear una aplicación de IA con esta plantilla, seleccione el directorio **model**. Para más detalles, véase **Importación de un metamodelo desde una plantilla**.

Plantillas admitidas

- **Plantilla de clasificación de imágenes basada en TensorFlow**
- **Plantilla general de TensorFlow-py27**
- **Plantilla general de TensorFlow-py36**
- **Plantilla general MXNet-py27**
- **Plantilla general MXNet-py36**
- **Plantilla general PyTorch-py27**
- **Plantilla general PyTorch-py36**
- **Plantilla general Caffe-CPU-py27**
- **Plantilla general Caffe-GPU-py27**
- **Plantilla general Caffe-CPU-py36**
- **Plantilla general Caffe-GPU-py36**
- **Plantilla Arm-Ascend**

Modos de entrada y salida admitidos

- **Modo de detección de objetos incorporado**
- **Modo de procesamiento de imágenes incorporado**
- **Modo de análisis predictivo incorporado**
- **Modo indefinido**

4.2.2 Plantillas

4.2.2.1 Plantilla de clasificación de imágenes basada en TensorFlow

Introducción

Motor de IA: TensorFlow 1.8; Entorno: Python 2.7. Esta plantilla se utiliza para importar un modelo de clasificación de imágenes basado en TensorFlow guardado en formato **SavedModel**. Esta plantilla utiliza el modo de procesamiento de imágenes de ModelArts incorporado. Para obtener detalles sobre el modo de procesamiento de imágenes, véase **Modo de procesamiento de imágenes incorporado**. Asegúrese de que el modelo pueda procesar imágenes cuyo **key** sea **images**, porque debe introducir una imagen cuyo **key** sea **images** al modelo para la inferencia. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de plantilla basado en TensorFlow almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

El modo de procesamiento de imágenes integrado no se puede sobrescribir. No se puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en TensorFlow

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    ├── <<Custom Python package>>    (Optional) User's Python package, which can
be directly referenced in model inference code
    ├── saved_model.pb        (Mandatory) Protocol buffer file, which contains
the diagram description of the model
    ├── variables             Mandatory for the main file of the *.pb model. The
folder must be named variables and contains the weight deviation of the model.
        ├── variables.index                  Mandatory
        ├── variables.data-00000-of-00001    Mandatory
    ├──customize_service.py   (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.2 Plantilla general de TensorFlow-py27

Introducción

Motor de IA: TensorFlow 1.8; Entorno: python2.7; Modo de entrada y salida: modo indefinido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo

o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de plantilla basado en TensorFlow almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Se puede sobrescribir **Modo indefinido**. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en TensorFlow

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    ├── <<Custom Python package>>    (Optional) User's Python package, which can
be directly referenced in model inference code
    ├── saved_model.pb        (Mandatory) Protocol buffer file, which contains
the diagram description of the model
    ├── variables             Mandatory for the main file of the *.pb model. The
folder must be named variables and contains the weight deviation of the model.
        ├── variables.index                  Mandatory
        ├── variables.data-00000-of-00001    Mandatory
    ├──customize_service.py   (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```


4.2.2.3 Plantilla general de TensorFlow-py36

Introducción

Motor de IA: TensorFlow 1.8; Entorno: Python 3.6; Modo de entrada y salida: modo indefinido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de plantilla basado en TensorFlow almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Modo indefinido se puede sobrescribir. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en TensorFlow

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    ├── <<Custom Python package>>    (Optional) User's Python package, which can
be directly referenced in model inference code
    ├── saved_model.pb        (Mandatory) Protocol buffer file, which contains
the diagram description of the model
    ├── variables             Mandatory for the main file of the *.pb model. The
folder must be named variables and contains the weight deviation of the model.
        ├── variables.index                  Mandatory
        ├── variables.data-00000-of-00001    Mandatory
```



```
    ├──customize_service.py   (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.4 Plantilla general MXNet-py27

Introducción

Motor de IA: MXNet 1.2.1; Entorno: Python 2.7; Modo de entrada y salida: modo indefinido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de modelos basado en MXNet almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Se puede sobrescribir **Modo indefinido**. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en MXNet

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    ├── <<Custom Python package>>       (Optional) User's Python package, which
can be directly referenced in model inference code
```



```
    ├── resnet-50-symbol.json    (Mandatory) Model definition file, which
contains the neural network description of the model
    ├── resnet-50-0000.params    (Mandatory) Model variable parameter file, which
contains parameter and weight information
    ├──customize_service.py      (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.5 Plantilla general MXNet-py36

Introducción

Motor de IA: MXNet 1.2.1; Entorno: Python 3.6; Modo de entrada y salida: No definido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de modelos basado en MXNet almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Modo indefinido se puede sobrescribir. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en MXNet

Al publicar el modelo, solo necesita especificar el directorio **model**.


```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    ├── <<Custom Python package>>       (Optional) User's Python package, which
can be directly referenced in model inference code
    ├── resnet-50-symbol.json    (Mandatory) Model definition file, which
contains the neural network description of the model
    ├── resnet-50-0000.params    (Mandatory) Model variable parameter file, which
contains parameter and weight information
    ├──customize_service.py      (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.6 Plantilla general PyTorch-py27

Introducción

Motor de IA: PyTorch 1.0; Entorno: Python 2.7; Modo de entrada y salida: No definido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de modelos basado en PyTorch almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Modo indefinido se puede sobrescribir. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en PyTorch

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
   ├── <<Custom Python package>>     (Optional) User's Python package, which can
be directly referenced in model inference code
   ├── resnet50.pth           (Mandatory) PyTorch model file, which contains
variable and weight information
   ├──customize_service.py   (Mandatory) Model inference code file. The file must
be named customize_service.py. Only one inference code file exists. The .py file
on which customize_service.py depends can be directly put in the model directory.
```

4.2.2.7 Plantilla general PyTorch-py36

Introducción

Motor de IA: PyTorch 1.0; Entorno: Python 3.6; Modo de entrada y salida: No definido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de modelos basado en PyTorch almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Se puede sobrescribir **Modo indefinido**. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en PyTorch

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    ├── <<Custom Python package>>     (Optional) User's Python package, which can
be directly referenced in model inference code
   ├── resnet50.pth           (Mandatory) PyTorch model file, which contains
variable and weight information
   ├──customize_service.py   (Mandatory) Model inference code file. The file must
be named customize_service.py. Only one inference code file exists. The .py file
on which customize_service.py depends can be directly put in the model directory.
```

4.2.2.8 Plantilla general Caffe-CPU-py27

Introducción

Motor de IA: Caffe 1.0 basado en CPU; Entorno: Python 2.7; Modo de entrada y salida: modo indefinido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de plantilla basado en Caffe almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Modo indefinido se puede sobrescribir. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete modelo basado en Caffe

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    |── <<Custom Python package>>      (Optional) User's Python package, which
can be directly referenced in model inference code
    |── deploy.prototxt         (Mandatory) Caffe model file, which contains
information such as the model network structure
    |── resnet.caffemodel       (Mandatory) Caffe model file, which contains
variable and weight information
   |── customize_service.py    (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.9 Plantilla general Caffe-GPU-py27

Introducción

Motor de IA: Caffe 1.0 basado en GPU; Entorno: Python 2.7; Modo de entrada y salida: No definido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de plantilla basado en Caffe almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Se puede sobrescribir **Modo indefinido**. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```


Ejemplo de paquete de modelo

Estructura del paquete modelo basado en Caffe

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    |── <<Custom Python package>>      (Optional) User's Python package, which
can be directly referenced in model inference code
    |── deploy.prototxt         (Mandatory) Caffe model file, which contains
information such as the model network structure
    |── resnet.caffemodel       (Mandatory) Caffe model file, which contains
variable and weight information
   |── customize_service.py    (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.10 Plantilla general Caffe-CPU-py36

Introducción

Motor de IA: Caffe 1.0 basado en CPU; Entorno: Python 3.6; Modo de entrada y salida: No definido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de plantilla basado en Caffe almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Se puede sobrescribir **Modo indefinido**. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package          //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
```



```
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete modelo basado en Caffe

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    |── <<Custom Python package>>      (Optional) User's Python package, which
can be directly referenced in model inference code
    |── deploy.prototxt         (Mandatory) Caffe model file, which contains
information such as the model network structure
    |── resnet.caffemodel       (Mandatory) Caffe model file, which contains
variable and weight information
   |── customize_service.py    (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.11 Plantilla general Caffe-GPU-py36

Introducción

Motor de IA: Caffe 1.0 basado en GPU; Entorno: Python 3.6; Modo de entrada y salida: No definido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de la plantilla es el paquete de plantilla basado en Caffe almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Modo indefinido se puede sobrescribir. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
```



```
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete modelo basado en Caffe

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    |── <<Custom Python package>>      (Optional) User's Python package, which
can be directly referenced in model inference code
    |── deploy.prototxt         (Mandatory) Caffe model file, which contains
information such as the model network structure
    |── resnet.caffemodel       (Mandatory) Caffe model file, which contains
variable and weight information
   |── customize_service.py    (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.2.12 Plantilla Arm-Ascend

Introducción

Motor de IA: MindSpore; Entorno: Python 3.5; Modo de entrada y salida: Indefinido. Seleccione un modo de entrada y salida adecuado basado en la función del modelo o escenario de aplicación. Cuando utilice la plantilla para importar un modelo, seleccione el directorio **model** que contiene los archivos del modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de plantilla basado en OM almacenado en OBS. Asegúrese de que el directorio de OBS que utiliza y ModelArts están en la misma región. Para obtener detalles sobre los requisitos de paquetes de modelos, véase **Ejemplo de paquete de modelo**.

Modo de entrada y salida

Modo indefinido se puede sobrescribir. Puede seleccionar otro modo de entrada y salida durante la creación del modelo.

Especificaciones del paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta **model** de OBS. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta **model**.
- El archivo de código de inferencia del modelo es obligatorio. El nombre del archivo debe ser **customize_service.py**. Solo puede haber un archivo de código de inferencia en la carpeta **model**. Para obtener detalles sobre cómo escribir código de inferencia de modelo, véase **Especificaciones para escribir el código de inferencia de modelo**.
- La estructura del paquete de plantilla importado utilizando la plantilla es la siguiente:

```
model/
|
```



```
├── Model file                 //(Mandatory) The model file format varies
according to the engine. For details, see the model package example.
├── Custom Python package           //(Optional) User's Python package, which
can be directly referenced in model inference code
├── customize_service.py  //(Mandatory) Model inference code file. The file
name must be customize_service.py. Otherwise, the code is not considered as
inference code.
```

Ejemplo de paquete de modelo

Estructura del paquete de modelo basado en OM

Al publicar el modelo, solo necesita especificar el directorio **model**.

```
OBS bucket/directory name
|── model    (Mandatory) The folder must be named model and is used to store
model-related files.
    ├── <<Custom Python package>>    (Optional) User's Python package, which can
be directly referenced in model inference code
    ├── model.om              (Mandatory) Protocol buffer file, which contains
the diagram description of the model
    ├──customize_service.py   (Mandatory) Model inference code file. The file
must be named customize_service.py. Only one inference code file exists. The .py
file on which customize_service.py depends can be directly put in the model
directory.
```

4.2.3 Modos de entrada y salida

4.2.3.1 Modo de detección de objetos incorporado

Entrada

Este es un modo de entrada y salida integrado para la detección de objetos. Los modelos que utilizan este modo se identifican como modelos de detección de objetos. La ruta de solicitud de predicción es la /, el protocolo de solicitud es **HTTP**, el método de solicitud es **POST**, **Content-Type** es **multipart/form-data**, **key** es **images** y **type** es **file**. Antes de seleccionar este modo, asegúrese de que el modelo pueda procesar los datos de entrada cuyo **key** esté **images**.

Salida

El resultado de la inferencia se devuelve en formato de JSON. Para obtener detalles sobre los campos, véase **Tabla 4-11**.

Tabla 4-11 Parámetros

Campo	Tipo	Descripción
detection_classes	String array	Lista de objetos detectados, por ejemplo, **["yunbao","cat"]**
detection_boxes	Float array	Coordenadas del cuadro delimitador, en el formato de $[y_{min}, x_{min}, y_{max}, x_{max}]$
detection_scores	Float array	Puntuaciones de confianza de los objetos detectados, que se utilizan para medir la precisión de la detección

El **JSON Schema** del resultado de inferencia es el siguiente:

```
{
    "type": "object",
    "properties": {
        "detection_classes": {
            "items": {
                "type": "string"
            },
            "type": "array"
        },
        "detection_boxes": {
            "items": {
                "minItems": 4,
                "items": {
                    "type": "number"
                },
                "type": "array",
                "maxItems": 4
            },
            "type": "array"
        },
        "detection_scores": {
            "items": {
                "type": "string"
            },
            "type": "array"
        }
    }
}
```

Ejemplo de solicitud

En este modo, ingrese una imagen para ser procesada en la solicitud de inferencia. El resultado de la inferencia se devuelve en formato de JSON. A continuación citamos varios ejemplos:

- Realización de la predicción en la consola

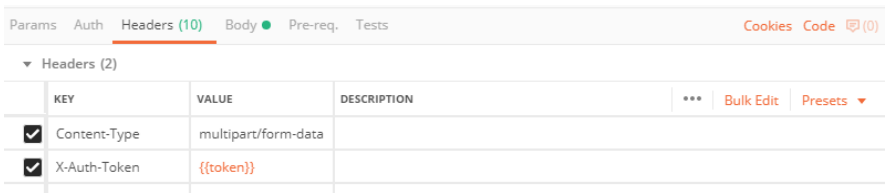
  En la pestaña **Prediction** de la página de detalles del servicio, cargue una imagen y haga clic en **Predict** para obtener el resultado de la predicción.

- Uso de Postman para invocar a una API de RESTful para la predicción

  Después de desplegar un modelo como servicio, puede obtener la URL de la API en la página de pestaña **Usage Guides** de la página de detalles del servicio.

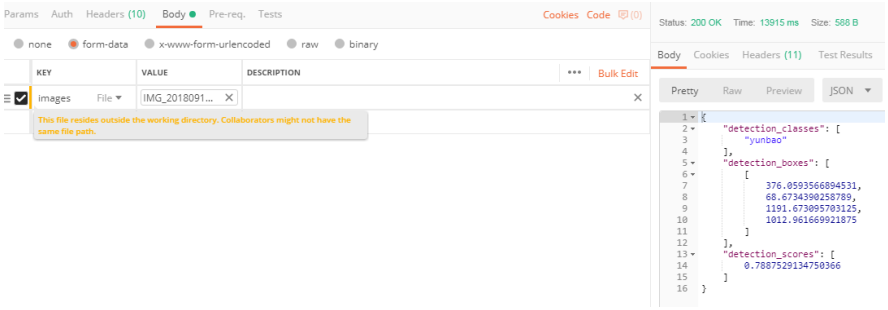
  – En la página **Headers**, configure **Content-Type** en **multipart/form-data** y **X-Auth-Token** en el token real obtenido.

  Figura 4-1 Configuración del encabezado de solicitud

  – En la pestaña **Body**, configure el cuerpo de la solicitud. Establezca **key** en **images**, seleccione **File**, seleccione la imagen que desea procesar y haga clic en **send** para enviar la solicitud de predicción.

Figura 4-2 Configuración del cuerpo de la solicitud



4.2.3.2 Modo de procesamiento de imágenes incorporado

Entrada

El modo de entrada y salida de procesamiento de imágenes integrado se puede aplicar a modelos como la clasificación de imágenes, la detección de objetos y la segmentación semántica de imágenes. La ruta de solicitud de predicción es la /, el protocolo de solicitud es **HTTPS**, el método de solicitud es **POST**, **Content-Type** es **multipart/form-data**, **key** es **images** y **type** es **file**. Antes de seleccionar este modo, asegúrese de que el modelo pueda procesar los datos de entrada cuyo **key** esté **images**.

Salida

El resultado de la inferencia se devuelve en formato de JSON. Los campos específicos están determinados por el modelo.

Ejemplo de solicitud

En este modo, ingrese una imagen para ser procesada en la solicitud de inferencia. La respuesta en formato de JSON varía según el modelo. A continuación citamos varios ejemplos:

- Realización de la predicción en la consola
- Uso de Postman para invocar a una API de RESTful para la predicción

  Después de desplegar un modelo como servicio, puede obtener la URL de la API en la página de pestaña **Usage Guides** de la página de detalles del servicio. En la pestaña **Body**, configure el cuerpo de la solicitud. Establezca **key** en **images**, seleccione **File**, seleccione la imagen que desea procesar y haga clic en **send** para enviar la solicitud de predicción.

Figura 4-3 Invocar a una API de RESTful

4.2.3.3 Modo de análisis predictivo incorporado

Entrada

Este es un modo de entrada y salida integrado para el análisis predictivo. Los modelos que usan este modo se identifican como modelos de análisis predictivo. La ruta de solicitud de predicción es /, el protocolo de solicitud es **HTTP**, el método de solicitud es **POST** y **Content-Type** es **application/json**. El cuerpo de la solicitud está en formato de JSON. Para obtener detalles sobre los campos de JSON, véase **Tabla 4-12**. Antes de seleccionar este modo, asegúrese de que el modelo puede procesar los datos de entrada en formato **JSON Schema**. Para obtener detalles sobre el formato de **JSON Schema**, véase la **guía oficial**.

Tabla 4-12 Descripción del campo JSON

Campo	Tipo	Descripción
data	Data structure	Datos de inferencia. Para más detalles, véase **Tabla 4-13**.

Tabla 4-13 Descripción de **Data**

Campo	Tipo	Descripción
req_data	ReqData array	Lista de datos de inferencia

ReqData es del tipo **Object** e indica los datos de inferencia. La estructura de datos está determinada por el escenario de aplicación. Para los modelos que utilizan este modo, la lógica de preprocesamiento en el código de inferencia del modelo personalizado debe ser capaz de procesar correctamente los datos ingresados en el formato definido por el modo.

El **JSON Schema** de una solicitud de predicción es el siguiente:

```
{
    "type": "object",
    "properties": {
        "data": {
            "type": "object",
            "properties": {
                "req_data": {
                    "items": [{
                        "type": "object",
                        "properties": {}
                    }],
                    "type": "array"
                }
            }
        }
    }
}
```

Salida

El resultado de la inferencia se devuelve en formato de JSON. Para obtener detalles sobre los campos de JSON, véase **Tabla 4-14**.

Tabla 4-14 Descripción del campo JSON

Campo	Tipo	Descripción
data	Data structure	Datos de inferencia. Para más detalles, véase **Tabla 4-15**.

Tabla 4-15 Descripción de **Data**

Campo	Tipo	Descripción
resp_data	RespData array	Lista de resultados de predicción

Similar a **ReqData**, **RespData** también es del tipo **Object** e indica el resultado de la predicción. Su estructura está determinada por el escenario de aplicación. Para los modelos que utilizan este modo, la lógica de postprocesamiento en el código de inferencia del modelo personalizado debe ser capaz de generar correctamente los datos en el formato definido por el modo.

El **JSON Schema** de un resultado de predicción es el siguiente:

```
{
    "type": "object",
    "properties": {
        "data": {
            "type": "object",
            "properties": {
                "resp_data": {
                    "type": "array",
                    "items": [{
                        "type": "object",
                        "properties": {}
                    }]
                }
            }
        }
    }
}
```

Ejemplo de solicitud

En este modo, ingrese los datos que se desea predecir en formato de JSON. El resultado de la predicción se devuelve en formato de JSON. A continuación citamos varios ejemplos:

- Realización de la predicción en la consola

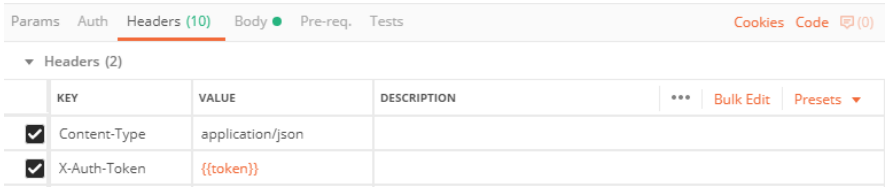
  En la pestaña **Prediction** de la página de detalles del servicio, ingrese el código de inferencia y haga clic en **Predict** para obtener el resultado de la predicción.

- Uso de Postman para invocar a una API de RESTful para la predicción

  Después de desplegar un modelo como servicio, puede obtener la URL de la API en la página de pestaña **Usage Guides** de la página de detalles del servicio.

  – En la pestaña **Headers**, configure **Content-Type** en **application/json** y **X-Auth-Token** en el token real obtenido.

Figura 4-4 Configuración del encabezado de solicitud para la predicción



– En la pestaña **Body**, edite los datos que desea predecir y haga clic en **send** para enviar la solicitud de predicción.

4.2.3.4 Modo indefinido

Descripción

El modo no definido no define el modo de entrada y salida. El modo de entrada y salida está determinado por el modelo. Seleccione este modo solo cuando el modo de entrada y salida existente no sea aplicable al escenario de aplicación del modelo. Los modelos importados en modo indefinido no se pueden desplegar como servicios por lotes. Además, es posible que la página de predicción del servicio no se muestre correctamente.

Entrada

Sin límite.

Salida

Sin límite.

Ejemplo de solicitud

El modo no definido no tiene ninguna solicitud de muestra específica porque la entrada y la salida de la solicitud están totalmente determinadas por el modelo.

4.3 Ejemplos de scripts personalizados

4.3.1 TensorFlow

Hay dos tipos de API de TensorFlow, Keras y tf. Utilizan diferentes códigos para entrenar y guardar modelos, pero el mismo código para la inferencia.

Entrenamiento de un modelo (API de Keras)

```
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense
import tensorflow as tf

# Import a training dataset.
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

print(x_train.shape)
```



```
from keras.layers import Dense
from keras.models import Sequential
import keras
from keras.layers import Dense, Activation, Flatten, Dropout

# Define a model network.
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(units=5120,activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(units=10, activation='softmax'))

# Define an optimizer and loss functions.
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
# Train the model.
model.fit(x_train, y_train, epochs=2)
# Evaluate the model.
model.evaluate(x_test, y_test)
```

Guardar un modelo (API de Keras)

```
from keras import backend as K

# K.get_session().run(tf.global_variables_initializer())

# Define the inputs and outputs of the prediction API.
# The key values of the inputs and outputs dictionaries are used as the index
keys for the input and output tensors of the model.
 # The input and output definitions of the model must match the custom inference
script.
predict_signature = tf.saved_model.signature_def_utils.predict_signature_def(
    inputs={"images" : model.input},
    outputs={"scores" : model.output}
)

# Define a save path.
builder = tf.saved_model.builder.SavedModelBuilder('./mnist_keras/')

builder.add_meta_graph_and_variables(

    sess = K.get_session(),
    # The tf.saved_model.tag_constants.SERVING tag needs to be defined for
inference and deployment.
    tags=[tf.saved_model.tag_constants.SERVING],
    """
    signature_def_map: Only single items can exist, or the corresponding key
needs to be defined as follows:
    tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY
    """
    signature_def_map={
        tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
            predict_signature
    }

)
builder.save()
```

Entrenamiento de un modelo (API de tf)

```
from __future__ import print_function

import gzip
import os
```



```
import urllib

import numpy
import tensorflow as tf
from six.moves import urllib

# Training data is obtained from the Yann LeCun official website http://
yann.lecun.com/exdb/mnist/.
SOURCE_URL = 'http://yann.lecun.com/exdb/mnist/'
TRAIN_IMAGES = 'train-images-idx3-ubyte.gz'
TRAIN_LABELS = 'train-labels-idx1-ubyte.gz'
TEST_IMAGES = 't10k-images-idx3-ubyte.gz'
TEST_LABELS = 't10k-labels-idx1-ubyte.gz'
VALIDATION_SIZE = 5000


def maybe_download(filename, work_directory):
    """Download the data from Yann's website, unless it's already here."""
    if not os.path.exists(work_directory):
        os.mkdir(work_directory)
    filepath = os.path.join(work_directory, filename)
    if not os.path.exists(filepath):
        filepath, _ = urllib.request.urlretrieve(SOURCE_URL + filename, filepath)
        statinfo = os.stat(filepath)
        print('Successfully downloaded %s %d bytes.' % (filename,
statinfo.st_size))
    return filepath


def _read32(bytestream):
    dt = numpy.dtype(numpy.uint32).newbyteorder('>')
    return numpy.frombuffer(bytestream.read(4), dtype=dt)[0]


def extract_images(filename):
    """Extract the images into a 4D uint8 numpy array [index, y, x, depth]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2051:
            raise ValueError(
                'Invalid magic number %d in MNIST image file: %s' %
                (magic, filename))
        num_images = _read32(bytestream)
        rows = _read32(bytestream)
        cols = _read32(bytestream)
        buf = bytestream.read(rows * cols * num_images)
        data = numpy.frombuffer(buf, dtype=numpy.uint8)
        data = data.reshape(num_images, rows, cols, 1)
        return data


def dense_to_one_hot(labels_dense, num_classes=10):
    """Convert class labels from scalars to one-hot vectors."""
    num_labels = labels_dense.shape[0]
    index_offset = numpy.arange(num_labels) * num_classes
    labels_one_hot = numpy.zeros((num_labels, num_classes))
    labels_one_hot.flat[index_offset + labels_dense.ravel()] = 1
    return labels_one_hot


def extract_labels(filename, one_hot=False):
    """Extract the labels into a 1D uint8 numpy array [index]."""
    print('Extracting %s' % filename)
    with gzip.open(filename) as bytestream:
        magic = _read32(bytestream)
        if magic != 2049:
            raise ValueError(
                'Invalid magic number %d in MNIST label file: %s' %
```



```
                (magic, filename))
        num_items = _read32(bytestream)
        buf = bytestream.read(num_items)
        labels = numpy.frombuffer(buf, dtype=numpy.uint8)
        if one_hot:
            return dense_to_one_hot(labels)
        return labels


class DataSet(object):
    """Class encompassing test, validation and training MNIST data set."""

    def __init__(self, images, labels, fake_data=False, one_hot=False):
        """Construct a DataSet. one_hot arg is used only if fake_data is true."""

        if fake_data:
            self._num_examples = 10000
            self.one_hot = one_hot
        else:
            assert images.shape[0] == labels.shape[0], (
                    'images.shape: %s labels.shape: %s' % (images.shape,
                                                           labels.shape))
            self._num_examples = images.shape[0]

            # Convert shape from [num examples, rows, columns, depth]
            # to [num examples, rows*columns] (assuming depth == 1)
            assert images.shape[3] == 1
            images = images.reshape(images.shape[0],
                                    images.shape[1] * images.shape[2])
            # Convert from [0, 255] -> [0.0, 1.0].
            images = images.astype(numpy.float32)
            images = numpy.multiply(images, 1.0 / 255.0)
        self._images = images
        self._labels = labels
        self._epochs_completed = 0
        self._index_in_epoch = 0

    @property
    def images(self):
        return self._images

    @property
    def labels(self):
        return self._labels

    @property
    def num_examples(self):
        return self._num_examples

    @property
    def epochs_completed(self):
        return self._epochs_completed

    def next_batch(self, batch_size, fake_data=False):
        """Return the next `batch_size` examples from this data set."""
        if fake_data:
            fake_image = [1] * 784
            if self.one_hot:
                fake_label = [1] + [0] * 9
            else:
                fake_label = 0
            return [fake_image for _ in range(batch_size)], [
                fake_label for _ in range(batch_size)
            ]
        start = self._index_in_epoch
        self._index_in_epoch += batch_size
        if self._index_in_epoch > self._num_examples:
            # Finished epoch
            self._epochs_completed += 1
```



```
            # Shuffle the data
            perm = numpy.arange(self._num_examples)
            numpy.random.shuffle(perm)
            self._images = self._images[perm]
            self._labels = self._labels[perm]
            # Start next epoch
            start = 0
            self._index_in_epoch = batch_size
            assert batch_size <= self._num_examples
        end = self._index_in_epoch
        return self._images[start:end], self._labels[start:end]


def read_data_sets(train_dir, fake_data=False, one_hot=False):
    """Return training, validation and testing data sets."""

    class DataSets(object):
        pass

    data_sets = DataSets()

    if fake_data:
        data_sets.train = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.validation = DataSet([], [], fake_data=True, one_hot=one_hot)
        data_sets.test = DataSet([], [], fake_data=True, one_hot=one_hot)
        return data_sets

    local_file = maybe_download(TRAIN_IMAGES, train_dir)
    train_images = extract_images(local_file)

    local_file = maybe_download(TRAIN_LABELS, train_dir)
    train_labels = extract_labels(local_file, one_hot=one_hot)

    local_file = maybe_download(TEST_IMAGES, train_dir)
    test_images = extract_images(local_file)

    local_file = maybe_download(TEST_LABELS, train_dir)
    test_labels = extract_labels(local_file, one_hot=one_hot)

    validation_images = train_images[:VALIDATION_SIZE]
    validation_labels = train_labels[:VALIDATION_SIZE]
    train_images = train_images[VALIDATION_SIZE:]
    train_labels = train_labels[VALIDATION_SIZE:]

    data_sets.train = DataSet(train_images, train_labels)
    data_sets.validation = DataSet(validation_images, validation_labels)
    data_sets.test = DataSet(test_images, test_labels)
    return data_sets

training_iteration = 1000

modelarts_example_path =  './modelarts-mnist-train-save-deploy-example'

export_path = modelarts_example_path + '/model/'
data_path = './'

print('Training model...')
mnist = read_data_sets(data_path, one_hot=True)
sess = tf.InteractiveSession()
serialized_tf_example = tf.placeholder(tf.string, name='tf_example')
feature_configs = {'x': tf.FixedLenFeature(shape=[784], dtype=tf.float32), }
tf_example = tf.parse_example(serialized_tf_example, feature_configs)
x = tf.identity(tf_example['x'], name='x')  # use tf.identity() to assign name
y_ = tf.placeholder('float', shape=[None, 10])
w = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
sess.run(tf.global_variables_initializer())
y = tf.nn.softmax(tf.matmul(x, w) + b, name='y')
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
```



```
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
values, indices = tf.nn.top_k(y, 10)
table = tf.contrib.lookup.index_to_string_table_from_tensor(
    tf.constant([str(i) for i in range(10)]))
prediction_classes = table.lookup(tf.to_int64(indices))
for _ in range(training_iteration):
    batch = mnist.train.next_batch(50)
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
print('training accuracy %g' % sess.run(
    accuracy, feed_dict={
        x: mnist.test.images,
        y_: mnist.test.labels
    }))
print('Done training!')
```

Saving a Model (tf API)

```
# Export the model.
# The model needs to be saved using the saved_model API.
print('Exporting trained model to', export_path)
builder = tf.saved_model.builder.SavedModelBuilder(export_path)

tensor_info_x = tf.saved_model.utils.build_tensor_info(x)
tensor_info_y = tf.saved_model.utils.build_tensor_info(y)

# Define the inputs and outputs of the prediction API.
# The key values of the inputs and outputs dictionaries are used as the index
keys for the input and output tensors of the model.
 # The input and output definitions of the model must match the custom inference
script.
prediction_signature = (
    tf.saved_model.signature_def_utils.build_signature_def(
        inputs={'images': tensor_info_x},
        outputs={'scores': tensor_info_y},
        method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME))

legacy_init_op = tf.group(tf.tables_initializer(), name='legacy_init_op')
builder.add_meta_graph_and_variables(
    # Set tag to serve/tf.saved_model.tag_constants.SERVING.
    sess, [tf.saved_model.tag_constants.SERVING],
    signature_def_map={
        'predict_images':
            prediction_signature,
    },
    legacy_init_op=legacy_init_op)

builder.save()

print('Done exporting!')
```

Código de inferencia (API de Keras y tf)

En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo que herede propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.

```
from PIL import Image
import numpy as np
from model_service.tfserving_model_service import TfServingBaseService


class MnistService(TfServingBaseService):

    # Match the model input with the user's HTTPS API input during preprocessing.
```



```
    # The model input corresponding to the preceding training part is
{"images":<array>}.
    def _preprocess(self, data):

        preprocessed_data = {}
        images = []
        # Iterate the input data.
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((1,784))
                images.append(image1)
        # Return the numpy array.
        images = np.array(images,dtype=np.float32)
        # Perform batch processing on multiple input samples and ensure that the
shape is the same as that inputted during training.
        images.resize((len(data), 784))
        preprocessed_data['images'] = images
        return preprocessed_data

    # Processing logic of the inference for invoking the parent class.

    # The output corresponding to model saving in the preceding training part is
{"scores":<array>}.
    # Postprocess the HTTPS output.
    def _postprocess(self, data):
        infer_output = {"mnist_result": []}
        # Iterate the model output.
        for output_name, results in data.items():
            for result in results:
                infer_output["mnist_result"].append(result.index(max(result)))
        return infer_output
```

4.3.2 TensorFlow 2.1

Entrenamiento y guardado de un modelo

```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
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.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    # Name the output layer output, which is used to obtain the result during
model inference.
    tf.keras.layers.Dense(10, activation='softmax', name="output")
])

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

model.fit(x_train, y_train, epochs=10)

tf.keras.models.save_model(model, "./mnist")
```


Código de inferencia

En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo. Esta clase de modelo hijo hereda las propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.

```
import logging
import threading

import numpy as np
import tensorflow as tf
from PIL import Image

from model_service.tfserving_model_service import TfServingBaseService

logger = logging.getLogger()
logger.setLevel(logging.INFO)


class MnistService(TfServingBaseService):

    def __init__(self, model_name, model_path):
        self.model_name = model_name
        self.model_path = model_path
        self.model = None
        self.predict = None

       # The label file can be loaded here and used in the post-processing
function.
        # Directories for storing the label.txt file on OBS and in the model
package

        # with open(os.path.join(self.model_path, 'label.txt')) as f:
        #     self.label = json.load(f)
        # Load the model in saved_model format in non-blocking mode to prevent
blocking timeout.
        thread = threading.Thread(target=self.load_model)
        thread.start()

    def load_model(self):
        # Load the model in saved_model format.
        self.model = tf.saved_model.load(self.model_path)

        signature_defs = self.model.signatures.keys()

        signature = []
        # only one signature allowed
        for signature_def in signature_defs:
            signature.append(signature_def)

        if len(signature) == 1:
            model_signature = signature[0]
        else:
            logging.warning("signatures more than one, use serving_default
signature from %s", signature)
            model_signature = tf.saved_model.DEFAULT_SERVING_SIGNATURE_DEF_KEY

        self.predict = self.model.signatures[model_signature]

    def _preprocess(self, data):
        images = []
        for k, v in data.items():
            for file_name, file_content in v.items():
                image1 = Image.open(file_content)
                image1 = np.array(image1, dtype=np.float32)
                image1.resize((28, 28, 1))
                images.append(image1)
```



```
        images = tf.convert_to_tensor(images, dtype=tf.dtypes.float32)
        preprocessed_data = images

        return preprocessed_data

    def _inference(self, data):

        return self.predict(data)

    def _postprocess(self, data):

        return {
            "result": int(data["output"].numpy()[0].argmax())
        }
```

4.3.3 PyTorch

Entrenamiento de un modelo

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms

# Define a network structure.
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
# The second dimension of the input must be 784.
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu((self.hidden1(x)))
        x = F.dropout(x, 0.2)
        x = self.output(x)
        return F.log_softmax(x)

def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                       100. * batch_idx / len(train_loader), loss.item()))

def test( model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  #
sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max
```



```
log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

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

device = torch.device("cpu")

batch_size=64

kwargs={}

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor()
                   ])),
    batch_size=batch_size, shuffle=True, **kwargs)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('.', train=False, transform=transforms.Compose([
        transforms.ToTensor()
    ])),
    batch_size=1000, shuffle=True, **kwargs)

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
optimizer = optim.Adam(model.parameters())

for epoch in range(1, 2 + 1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
```

Guardar un modelo

```
# The model must be saved using state_dict and can be deployed remotely.
torch.save(model.state_dict(), "pytorch_mnist/mnist_mlp.pt")
```

Código de inferencia

En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo. Esta clase de modelo hijo hereda las propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.

```
from PIL import Image
import log
from model_service.pytorch_model_service import PTServingBaseService
import torch.nn.functional as F

import torch.nn as nn
import torch
import json

import numpy as np

logger = log.getLogger(__name__)

import torchvision.transforms as transforms

# Define model preprocessing.
infer_transformation = transforms.Compose([
    transforms.Resize((28,28)),
    # Transform to a PyTorch tensor.
    transforms.ToTensor()
```



```
])


import os


class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # Call the constructor of the parent class.
        super(PTVisionService, self).__init__(model_name, model_path)
        # Call the customized function to load the model.
        self.model = Mnist(model_path)
         # Load tags.
        self.label = [0,1,2,3,4,5,6,7,8,9]
        # Labels can also be loaded by label file.
        # Store the label.json file in the model directory. The following
information is read:
        dir_path = os.path.dirname(os.path.realpath(self.model_path))
        with open(os.path.join(dir_path, 'label.json')) as f:
            self.label = json.load(f)


    def _preprocess(self, data):

        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # Gray processing
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch,
dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    def _inference(self, data):

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

        return result

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden1 = nn.Linear(784, 5120, bias=False)
        self.output = nn.Linear(5120, 10, bias=False)

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = F.relu((self.hidden1(x)))
        x = F.dropout(x, 0.2)
```



```
        x = self.output(x)
        return F.log_softmax(x)



def Mnist(model_path, **kwargs):
    # Generate a network.
    model = Net()
    # Load the model.
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))
    # CPU or GPU mapping
    model.to(device)
    # Declare an inference mode.
    model.eval()

    return model
```

4.3.4 Caffe

Entrenamiento y guardado de un modelo

Archivo de **lenet_train_test.prototxt**

```
name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
```



```
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
```



```
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

Archivo de **lenet_solver.prototxt**

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
```



```
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 1000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: CPU
```

Entrenar el modelo.

```
./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt
```

El archivo de **caffemodel** se genera después del entrenamiento del modelo. Vuelva a escribir el archivo **lenet_train_test.prototxt** en el archivo **lenet_deploy.prototxt** utilizado para el despliegue modificando las capas de entrada y salida.

```
name: "LeNet"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 1 dim: 1  dim: 28 dim: 28 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "conv2"
```



```
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
```



```
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "prob"
  type: "Softmax"
  bottom: "ip2"
  top: "prob"
}
```

Código de inferencia

En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo. Esta clase de modelo hijo hereda las propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.

```
from model_service.caffe_model_service import CaffeBaseService

import numpy as np

import os, json

import caffe

from PIL import Image


class LenetService(CaffeBaseService):

    def __init__(self, model_name, model_path):
        # Call the inference method of the parent class.
        super(LenetService, self).__init__(model_name, model_path)

        # Configure preprocessing information.
        transformer = caffe.io.Transformer({'data':
self.net.blobs['data'].data.shape})
        # Transform to NCHW.
        transformer.set_transpose('data', (2, 0, 1))
        # Perform normalization.
        transformer.set_raw_scale('data', 255.0)

        # If the batch size is set to 1, inference is supported for only one
image.
        self.net.blobs['data'].reshape(1, 1, 28, 28)
        self.transformer = transformer

       # Define the class labels.
        self.label = [0,1,2,3,4,5,6,7,8,9]


    def _preprocess(self, data):

        for k, v in data.items():
            for file_name, file_content in v.items():
                im = caffe.io.load_image(file_content, color=False)
               # Pre-process the images.
                self.net.blobs['data'].data[...] =
self.transformer.preprocess('data', im)
```



```
                return

    def _postprocess(self, data):

        data = data['prob'][0, :]
        predicted = np.argmax(data)
        predicted = {"predicted" : str(predicted) }

        return predicted
```

4.3.5 XGBoost

Entrenamiento y guardado de un modelo

```
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split

# Prepare training data and setting parameters
iris = pd.read_csv('/home/ma-user/work/iris.csv')
X = iris.drop(['variety'],axis=1)
y = iris[['variety']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1234565)
params = {
    'booster': 'gbtree',
    'objective': 'multi:softmax',
    'num_class': 3,
    'gamma': 0.1,
    'max_depth': 6,
    'lambda': 2,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'min_child_weight': 3,
    'silent': 1,
    'eta': 0.1,
    'seed': 1000,
    'nthread': 4,
}
plst = params.items()
dtrain = xgb.DMatrix(X_train, y_train)
num_rounds = 500
model = xgb.train(plst, dtrain, num_rounds)
model.save_model('/tmp/xgboost.m')
```

Antes del entrenamiento, descarga el conjunto de datos **iris.csv**, descomprímelo y súbelo al directorio **/home/ma-user/work/** de la instancia del cuaderno. Descargue el conjunto de datos **iris.csv** desde **https://gist.github.com/netj/8836201**. Para obtener más información sobre cómo cargar un archivo en una instancia de notebook, consulte **Escenarios de carga y entradas**.

Después de guardar el modelo, debe subirse al directorio OBS antes de publicarse. La configuración **config.json** y el código de inferencia **customize_service.py** deben incluirse durante la publicación. Para obtener más información sobre cómo compilar **config.json**, consulte **Especificaciones para editar un archivo de configuración de modelo**. Para obtener más información sobre el código de inferencia, véase **Código de inferencia**.

Código de inferencia

En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo. Esta clase de modelo hijo hereda las propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.


```
# coding:utf-8
import collections
import json
import xgboost as xgb
from model_service.python_model_service import XgSklServingBaseService
class UserService(XgSklServingBaseService):

    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:
            array = []
            for each in element:
                array.append(element[each])
            list_data.append(array)
        return list_data

    #   predict
    def _inference(self, data):
        xg_model = xgb.Booster(model_file=self.model_path)
        pre_data = xgb.DMatrix(data)
        pre_result = xg_model.predict(pre_data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self,data):
        resp_data = []
        for element in data:
            resp_data.append({"predictresult": element})
        return resp_data
```

4.3.6 PySpark

Entrenamiento y guardado de un modelo

```
from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

# Prepare training data using tuples.
# Prepare training data from a list of (label, features) tuples.
training = spark.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])

# Create a training instance. The logistic regression algorithm is used for
training.
# Create a LogisticRegression instance. This instance is an Estimator.
lr = LogisticRegression(maxIter=10, regParam=0.01)

# Train the logistic regression model.
# Learn a LogisticRegression model. This uses the parameters stored in lr.
model = lr.fit(training)

# Save the model to a local directory.
# Save model to local path.
model.save("/tmp/spark_model")
```

Después de guardar el modelo, debe subirse al directorio OBS antes de publicarse. La configuración **config.json** y el código de inferencia **customize_service.py** deben incluirse durante la publicación. Para obtener más información sobre cómo compilar **config.json**, consulte **Especificaciones para editar un archivo de configuración de modelo**. Para obtener más información sobre el código de inferencia, véase **Código de inferencia**.

Código de inferencia

En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo. Esta clase de modelo hijo hereda las propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.

```
# coding:utf-8
import collections
import json
import traceback

import model_service.log as log
from model_service.spark_model_service import SparkServingBaseService
from pyspark.ml.classification import LogisticRegression

logger = log.getLogger(__name__)


class UserService(SparkServingBaseService):
    # Pre-process data.
    def _preprocess(self, data):
        logger.info("Begin to handle data from user data...")
        # Read data.
        req_json = json.loads(data, object_pairs_hook=collections.OrderedDict)
        try:
            # Convert data to the spark dataframe format.
            predict_spdf =
self.spark.createDataFrame(pd.DataFrame(req_json["data"]["req_data"]))
        except Exception as e:
            logger.error("check your request data does meet the requirements ?")
            logger.error(traceback.format_exc())
            raise Exception("check your request data does meet the
requirements ?")
        return predict_spdf

    # Perform model inference.
    def _inference(self, data):
        try:
             # Load a model file.
            predict_model = LogisticRegression.load(self.model_path)
            # Perform data inference.
            prediction_result = predict_model.transform(data)
    except Exception as e:
            logger.error(traceback.format_exc())
            raise Exception("Unable to load model and do dataframe
transformation.")
        return prediction_result

    # Post-process data.
    def _postprocess(self, pre_data):
        logger.info("Get new data to respond...")
        predict_str = pre_data.toPandas().to_json(orient='records')
        predict_result = json.loads(predict_str)
        return predict_result
```

4.3.7 Aprendizaje de Scikit

Entrenamiento y guardado de un modelo

```
import json
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib
iris = pd.read_csv('/home/ma-user/work/iris.csv')
```



```
X = iris.drop(['variety'],axis=1)
y = iris[['variety']]
# Create a LogisticRegression instance and train model
logisticRegression = LogisticRegression(C=1000.0, random_state=0)
logisticRegression.fit(X,y)
# Save model to local path
joblib.dump(logisticRegression, '/tmp/sklearn.m')
```

Antes del entrenamiento, descarga el conjunto de datos **iris.csv**, descomprímelo y súbelo al directorio **/home/ma-user/work/** de la instancia del cuaderno. Descargue el conjunto de datos **iris.csv** desde **https://gist.github.com/netj/8836201**. Para obtener más información sobre cómo cargar un archivo en una instancia de notebook, consulte **Escenarios de carga y entradas**.

Después de guardar el modelo, debe subirse al directorio OBS antes de publicarse. Los archivos **config.json** y **customize_service.py** deben estar incluidos durante la publicación. Para obtener detalles sobre el método de definición, véase **Introducción a las especificaciones del paquete modelo**.

Código de inferencia

En el archivo de código de inferencia de modelo **customize_service.py**, agregue una clase de modelo hijo. Esta clase de modelo hijo hereda las propiedades de su clase de modelo padre. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 4-9**.

```
# coding:utf-8
import collections
import json
from sklearn.externals import joblib
from model_service.python_model_service import XgSklServingBaseService

class UserService(XgSklServingBaseService):

    # request data preprocess
    def _preprocess(self, data):
        list_data = []
        json_data = json.loads(data, object_pairs_hook=collections.OrderedDict)
        for element in json_data["data"]["req_data"]:
            array = []
            for each in element:
                array.append(element[each])
                list_data.append(array)
        return list_data

    # predict
    def _inference(self, data):
        sk_model = joblib.load(self.model_path)
        pre_result = sk_model.predict(data)
        pre_result = pre_result.tolist()
        return pre_result

    # predict result process
    def _postprocess(self,data):
        resp_data = []
        for element in data:
            resp_data.append({"predictresult": element})
        return resp_data
```


5 ModelArts monitoreo en Cloud Eye

5.1 Métricas de ModelArts

Descripción

La plataforma de servicios en la nube proporciona Cloud Eye para ayudarlo a comprender mejor el estado de sus servicios y modelos en tiempo real de ModelArts. Puede usar Cloud Eye para monitorear automáticamente sus servicios en tiempo real y cargas de modelos en tiempo real de ModelArts, y gestionar alarmas y notificaciones para obtener las métricas de rendimiento de ModelArts y modelos.

Espacio de nombres

SYS.ModelArts

Métricas de monitoreo

Tabla 5-1 Métricas de ModelArts

ID de la métrica	Nombre de la métrica	Descripción	Rango de valor	Entidad monitoreada	Intervalo de monitoreo
cpu_usage	CPU Usage	Uso de CPU de ModelArts Unidad: %	≥ 0%	Cargas de modelo de ModelArts	1 minuto
mem_usage	Memory Usage	Uso de memoria de ModelArts Unidad: %	≥ 0%	Cargas de modelo de ModelArts	1 minuto
gpu_util	GPU Usage	Uso de GPU de ModelArts Unidad: %	≥ 0%	Cargas de modelo de ModelArts	1 minuto

ID de la métrica	Nombre de la métrica	Descripción	Rango de valor	Entidad monitoreada	Intervalo de monitoreo
gpu_mem_usage	GPU Memory Usage	Uso de memoria de GPU de ModelArts Unidad: %	≥ 0%	Cargas de modelo de ModelArts	1 minuto
npu_util	NPU Usage	Uso de NPU de ModelArts Unidad: %	≥ 0%	Cargas de modelo de ModelArts	1 minuto
npu_mem_usage	NPU Memory Usage	Uso de memoria de NPU de ModelArts Unidad: %	≥ 0%	Cargas de modelo de ModelArts	1 minuto
successfully_called_times	Number of Successful Calls	Veces que ModelArts ha sido invocado con éxito Unidad: veces/minuto	≥ veces/minuto	Modelos de ModelArts Servicios en tiempo real de ModelArts	1 minuto
failed_called_times	Number of Failed Calls	Veces que no se pudo invocar a ModelArts Unidad: veces/minuto	≥ veces/minuto	Modelos de ModelArts Servicios en tiempo real de ModelArts	1 minuto
total_called_times	Total Calls	Veces que se invoca ModelArts Unidad: veces/minuto	≥ veces/minuto	Cargas de modelo de ModelArts Servicios en tiempo real de ModelArts	1 minuto
disk_read_rate	Disk Read Rate	Velocidad de lectura de disco de ModelArts Unidad: bit/minuto	≥ bit/minuto	Cargas de modelo de ModelArts	1 minuto
disk_write_rate	Disk Write Rate	Velocidad de escritura en disco de ModelArts Unidad: bit/minuto	≥ bit/minuto	Cargas de modelo de ModelArts	1 minuto

ID de la métrica	Nombre de la métrica	Descripción	Rango de valor	Entidad monitoreada	Intervalo de monitoreo
send_bytes_rate	Uplink rate	Tasa de tráfico saliente de ModelArts Unidad: bit/minuto	≥ bit/minuto	Cargas de modelo de ModelArts	1 minuto
recv_bytes_rate	Downlink rate	Tasa de tráfico entrante de ModelArts	≥ bit/minuto	Cargas de modelo de ModelArts	1 minuto
req_count_2xx	2xx Responses	Número de veces que la API devuelve una respuesta 2xx	≥ veces/minuto	Servicios en tiempo real de ModelArts	1 minuto
req_count_4xx	4xx Errors	Número de veces que la API devuelve un error 4xx	≥ veces/minuto	Servicios en tiempo real de ModelArts	1 minuto
req_count_5xx	5xx Errors	Número de veces que la API devuelve un error 5xx	≥ veces/minuto	Servicios en tiempo real de ModelArts	1 minuto
avg_latency	Average Latency	Latencia promedio de la API	≥ ms	Servicios en tiempo real de ModelArts	1 minuto

ID de la métrica	Nombre de la métrica	Descripción	Rango de valor	Entidad monitoreada	Intervalo de monitoreo

Si un objeto de medición tiene varias dimensiones de medición, todas las dimensiones de medición son obligatorias cuando se utiliza una API para consultar métricas de supervisión.
● A continuación se proporciona un ejemplo de uso del **dim** multidimensional para consultar una única métrica de monitorización: dim.0=service_id,530cd6b0-86d7-4818-837f-935f6a27414d&dim.1="model_id,3773b058-5b4f-4366-9035-9bbd9964714a
● A continuación se proporciona un ejemplo de uso de **dim** multidimensional para consultar métricas de supervisión por lotes:
"dimensions": [
{
"name": "service_id",
"value": "530cd6b0-86d7-4818-837f-935f6a27414d"
}
{
"name": "model_id",
"value": "3773b058-5b4f-4366-9035-9bbd9964714a"
}
]					

Dimensiones

Tabla 5-2 Descripción de la dimensión

Clave	Valor
service_id	Real-time service ID
model_id	Model ID

5.2 Configuración de reglas de alarma

Escenario

Establecer reglas de alarma le permite personalizar los objetos monitoreados y las políticas de notificación para que pueda conocer el estado de los servicios y modelos en tiempo real de ModelArts de manera oportuna.

Una regla de alarma incluye el nombre de la regla de alarma, el objeto monitorizado, la métrica, el umbral, el intervalo de monitorización y si se debe enviar una notificación. Esta sección describe cómo establecer reglas de alarma para los servicios y modelos de ModelArts.

NOTA

Solo los servicios en tiempo real en el estado **Running** pueden interconectarse con CES.

Requisitos previos

- Se ha creado un servicio en tiempo real de ModelArts.
- En Cloud Eye se ha habilitado la supervisión de ModelArts. Para ello, inicie sesión en la consola de Cloud Eye. En la página Cloud Eye, haga clic en **Custom Monitoring**. A continuación, habilite la supervisión de ModelArts según se le indique.

Procedimiento

Configure una regla de alarma de cualquiera de las siguientes maneras:

- Configure una regla de alarma para todos los servicios de ModelArts.
- Configure una regla de alarma para un servicio de ModelArts.
- Configure una regla de alarma para una versión de modelo.
- Configure una regla de alarma para una métrica de una versión de servicio o modelo.

Método 1: Establecer una regla de alarma para todos los servicios de ModelArts

1. Inicie sesión en la consola de gestión.
2. En la **Service List**, haga clic en **Cloud Eye** en **Management & Governance**.
3. En el panel de navegación de la izquierda, seleccione **Alarm Management** > **Alarm Rules** y haga clic en **Create Alarm Rule**.
4. En la página **Create Alarm Rule**, configure **Resource Type** en **ModelArts**, **Dimension** en **Service** y **Method** en **Configure manually** y configure las políticas de alarmas. Luego, confirme los ajustes y haga clic en **Create**.

Método 2: Configuración de una regla de alarma para un servicio único

1. Inicie sesión en la consola de gestión.
2. En la **Service List**, haga clic en **Cloud Eye** en **Management & Governance**.
3. En el panel de navegación izquierdo, seleccione **Cloud Service Monitoring > ModelArts**.
4. Seleccione un servicio en tiempo real para el que desea crear una regla de alarma y haga clic en **Create Alarm Rule** en la columna **Operation**.
5. En la página **Create Alarm Rule**, cree una regla de alarma para los servicios y modelos en tiempo real de ModelArts según se le solicite.

Método 3: Configuración de una regla de alarma para una versión de modelo

1. Inicie sesión en la consola de gestión.
2. En la **Service List**, haga clic en **Cloud Eye** en **Management & Governance**.
3. En el panel de navegación izquierdo, seleccione **Cloud Service Monitoring > ModelArts**.
4. Haga clic en la flecha abajo junto al nombre del servicio en tiempo real de destino. Luego, haga clic en **Create Alarm Rule** en la columna **Operation** de la versión de destino.

5. En la página **Create Alarm Rule**, cree una regla de alarma para las cargas de modelo según se indique.

Método 4: Configuración de una regla de alarma para una métrica de una versión de servicio o modelo

1. Inicie sesión en la consola de gestión.
2. En la **Service List**, haga clic en **Cloud Eye** en **Management & Governance**.
3. En el panel de navegación izquierdo, seleccione **Cloud Service Monitoring > ModelArts**.
4. Haga clic en la flecha abajo junto al nombre del servicio en tiempo real de destino. Luego, haga clic en la versión de destino y vea los detalles de las reglas de alarma.
5. En la página de detalles de la regla de alarma, haga clic en el signo más (+) en la esquina superior derecha de una métrica y configure una regla de alarma para la métrica.

5.3 Consulta de métricas de monitoreo

Escenario

Cloud Eye en la plataforma de servicios en la nube monitoriza el estado de los servicios en tiempo real de ModelArts y las cargas de modelos. Puede obtener las métricas de monitorización de cada servicio en tiempo real de ModelArts y cargas de modelo en la consola de gestión. Los datos monitoreados requieren un período de tiempo para su transmisión y visualización. El estado de ModelArts que se muestra en la consola de Cloud Eye suele ser el estado obtenido de 5 a 10 minutos antes. Puede ver los datos monitoreados de un servicio en tiempo real recién creado de 5 a 10 minutos más tarde.

Requisitos previos

- El servicio en tiempo real ModelArts se está ejecutando correctamente.
- Las reglas de alarma se han configurado en la página de Cloud Eye. Para más detalles, véase **Configuración de reglas de alarma**.
- El servicio en tiempo real ha estado funcionando correctamente durante al menos 10 minutos.
- Los datos monitoreados y los gráficos están disponibles para un nuevo servicio en tiempo real después de que el servicio se ejecute durante al menos 10 minutos.
- Cloud Eye no muestra las métricas de un servicio en tiempo real defectuoso o eliminado. Las métricas de monitorización se pueden ver después de que se inicie o se recupere el servicio en tiempo real.

Los datos de supervisión no están disponibles sin reglas de alarma configuradas en Cloud Eye. Para más detalles, véase **Configuración de reglas de alarma**.

Procedimiento

1. Inicie sesión en la consola de gestión.
2. En la **Service List**, haga clic en **Cloud Eye** bajo **Management & Governance**.
3. En el panel de navegación izquierdo, seleccione **Cloud Service Monitoring > ModelArts**.

4. Consulte gráficos de monitorización.
   - Consulta de gráficos de monitoreo de un servicio en tiempo real: Haga clic en **View Metric** en la columna **Operation**.
   - Consulta de gráficos de monitoreo de las cargas del modelo: Haga clic en junto al servicio en tiempo real de destino y haga clic en **View Metric** en la columna **Operation** del modelo de destino.
5. En el área de supervisión, puede seleccionar una duración para ver los datos de supervisión.

   Puede ver los datos de supervisión en las últimas 1 hora, 3 horas o 12 horas. Para ver la curva de supervisión de un rango de tiempo más largo, haga clic en para ampliar el gráfico.

