ModelArts

Implementación de inferencia

Edición 01

Fecha 2024-06-25

HUAWEI TECHNOLOGIES CO., LTD.

Marcas y permisos


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

Las funciones, los productos y los servicios adquiridos están estipulados en el contrato celebrado entre Huawei y el cliente. Es posible que la totalidad o parte de los productos, las funciones y los servicios descritos en el presente documento no se encuentren dentro del alcance de compra o de uso. A menos que el contrato especifique lo contrario, ninguna de las afirmaciones, informaciones ni recomendaciones contenidas en este documento constituye garantía alguna, ni expresa ni implícita.

La información contenida en este documento se encuentra sujeta a cambios sin previo aviso. En la preparación de este documento se realizaron todos los esfuerzos para garantizar la precisión de sus contenidos. Sin embargo, ninguna declaración, información ni recomendación contenida en el presente constituye garantía alguna, ni expresa ni implícita.

Huawei Technologies Co., Ltd.

Dirección: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Sitio web: https://www.huawei.com

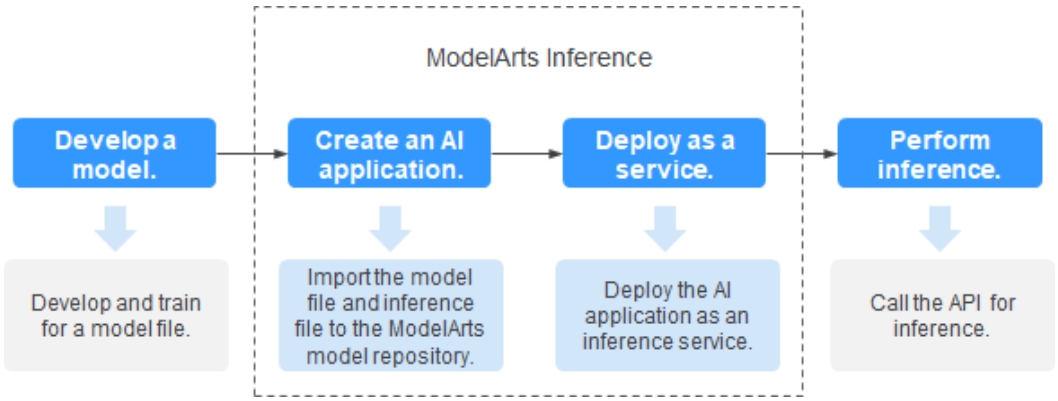
Email: support@huawei.com

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.
- Implementar como servicio: implemente la aplicación de IA como instancia de contenedor en el fondo de recursos y registre las API de inferencia a las que se puede acceder externamente.
- Realizar inferencia: Agregue la función de llamar a las API de inferencia a su aplicación para integrar la inferencia de IA en el proceso de servicio.

Implementación de una aplicación de IA como servicio

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

- **Servicio en tiempo real**

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

- **Servicio por lotes**

  Implemente 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. Los cambios 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 generados durante el entrenamiento y gestionar de forma centralizada todas las aplicaciones de IA iteradas y depuradas.You can also trace back your models with traceback diagrams of datasets, training, and models.

Restricciones de uso

- 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 la implementación y la implementación.
- 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

- **Importar desde modelos entrenados**: puede crear un trabajo de entrenamiento en el ModelArts y completar el entrenamiento de modelos. Después de obtener un modelo satisfactorio, cree una aplicación de IA para su implementación.
- **Importación de una plantilla**: Debido a que las configuraciones de los plantillas con las mismas funciones son similares, el ModelArts integra las configuraciones de dichos plantillas en una plantilla común. Al utilizar esta plantilla, puede importar fácilmente y rápidamente plantillas para crear aplicaciones de IA sin compilar el archivo de configuración **config.json**.
- **Importación desde una imagen de contenedor**: para los motores de IA que no son compatibles con ModelArts puede importar los modelos que compila a ModelArts mediante imágenes personalizadas y crear aplicaciones de IA para su implementación.

- **Importación desde OBS**: Si utiliza un marco común para desarrollar y entrenar un modelo localmente, puede cargar el modelo en un bucket OBS basado en las especificaciones del paquete del modelo, importar el modelo de OBS a ModelArts y crear una aplicación de IA para la implementación del servicio.

Funciones de la gestión de aplicaciones de IA

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

Función soportada	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. ● **Establecer un trabajo de entrenamiento como fuente del metamodelo** ● **Importación de un metamodelo desde OBS a través de una plantilla** ● **Creación e importación de una imagen de modelo** ● **Importación de un metamodelo desde OBS a través de configuraciones manuales**
Gestión 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 basadas en versiones.

Motores de IA compatibles para la inferencia de ModelArts

Si importa un plantilla 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.
- 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 tf1.13-python2.7-gpu tf1.13-python2.7-cpu tf1.13-python3.6-gpu tf1.13-python3.6-cpu tf1.13-python3.7-cpu tf1.13-python3.7-gpu tf2.1-python3.7 tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 (recommended)	● 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.
MXNet	python3.7 python3.6 python2.7	● MXNet 1.2.1 se utiliza en python2.7, python3.6 y python3.7. ● python2.7, python3.6 y python3.7 indican que el modelo puede ejecutarse tanto en CPU como en GPU. ● El tiempo de ejecución predeterminado es python2.7.
Caffe	python2.7 python3.6 python3.7 python2.7-gpu python3.6-gpu python3.7-gpu python2.7-cpu python3.6-cpu python3.7-cpu	● Caffe 1.0.0 se utiliza en python2.7, python3.6, python3.7, python2.7-gpu, python3.6-gpu, python3.7-gpu, python2.7-cpu, python3.6-cpu y python3.7-cpu. ● python 2.7, python3.7 y python3.6 solo se pueden usar para ejecutar modelos en CPU. Para otros valores de tiempo de ejecución, si el sufijo contiene cpu o gpu, el modelo solo puede ejecutarse en CPU o GPU. Utilice el tiempo de ejecución de python2.7-gpu, python3.6-gpu, python3.7-gpu, python2.7-cpu, python3.6-cpu, y python3.7-cpu. ● El tiempo de ejecución predeterminado es python2.7.

Motor	Tiempo de ejecución	Nota
Spark_MLlib	python2.7 python3.6	● 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 python3.6	● 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 python3.6	● 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.
PyTorch	python2.7 python3.6 python3.7 pytorch1.4-python3.7 pytorch1.5-python3.7 pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 (recommended)	● 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 mindspore_1.7.0-cpu-py_3.7-ubuntu_18.04-x86_64 (recommended)	AArch64 solo puede ejecutarse en chips D310.

2.2 Creación de una aplicación de IA

2.2.1 Establecer un trabajo de entrenamiento como fuente del metamodelo

Puede crear un trabajo de entrenamiento en el ModelArts y realizar entrenamiento para obtener un modelo satisfactorio. A continuación, importe el modelo a Model Management

para una gestión centralizada. Además, puede implementar rápidamente el modelo como un servicio.

Fondo

- Si se utiliza un modelo generado por el trabajo de entrenamiento ModelArts asegúrese de que el trabajo de entrenamiento se ha ejecutado correctamente y que el modelo se ha almacenado en el directorio OBS correspondiente. (El parámetro de salida de datos es train_url.)
- Un modelo generado a partir de un trabajo de entrenamiento que utiliza algoritmos suscritos se puede importar directamente a ModelArts sin necesidad de utilizar el código de inferencia o el archivo de configuración.
- Si se genera un modelo a partir de un trabajo de formación 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 OBS que utiliza 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 más información sobre los parámetros, consulte **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.
Label	Etiqueta de aplicación de IA. Se admite un máximo de cinco etiquetas.
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 más información sobre los parámetros, consulte **Tabla 2-4**.

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

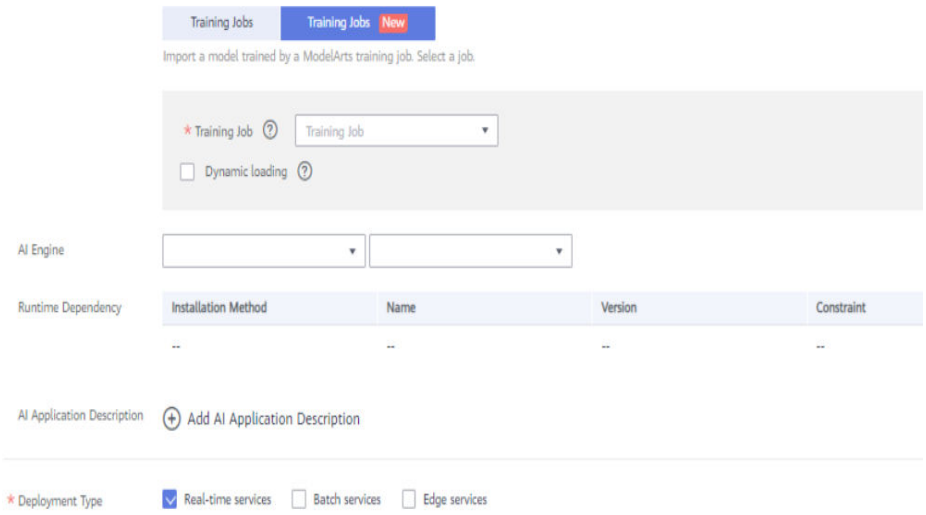


Tabla 2-4 Parámetros de la fuente del metamodelo

Parámetro	Descripción
Meta Model Source	Elija **Training Job** > **Training Jobs** or **Training Job** > **Training Jobs (New)**. ● Seleccione un trabajo de entrenamiento que haya completado la 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**: si esta función está habilitada, los archivos de modelo se descargan de OBS y luego se asocian a contenedores solo cuando se implementa el modelo. Esto acelera la implementación y las actualizaciones del modelo. **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, que coincide automáticamente con el trabajo de entrenamiento seleccionado
Inference Code	El código de inferencia personaliza la lógica de procesamiento de inferencia de una aplicación de IA. Mostrar 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 Description** y establezca **Document name** y **URL**. Puede agregar hasta tres descripciones de aplicaciones de IA.

Parámetro	Descripción
Deployment Type	Seleccione los tipos de servicio que puede implementar la aplicación. Al implementar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services** aquí, solo podrá implementar la aplicación de IA como 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, la implementación rápida de aplicaciones de IA y la publicación de aplicaciones de IA.

Acciones de seguimiento

Implementación de aplicaciones de IA como servicios: En la lista de aplicaciones de IA, haga clic en la flecha hacia abajo a la izquierda del nombre de una aplicación de IA para comprobar todas las versiones de la aplicación de IA. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** y seleccione un tipo de implementación en el cuadro de lista desplegable. La aplicación de IA se puede implementar en un tipo de implementación seleccionado durante la creación de la aplicación de IA.

2.2.2 Importación de un metamodelo desde OBS a través de una plantilla

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**.

Fondo

- 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 fácil y rápidamente aplicaciones de IA. Para obtener más información sobre la plantilla, consulte **Introducción a las plantillas de modelo**.
- Para obtener más información sobre las plantillas compatibles, consulte **Plantillas compatibles**. Para obtener más información sobre los modos de entrada y salida de cada plantilla, consulte **Modos de entrada y salida compatibles**.
- Asegúrese de haber subido el plantilla a OBS de acuerdo con las especificaciones del paquete de plantilla de la plantilla correspondiente. El tamaño total del paquete del modelo no puede superar los 11 GB.
- El directorio OBS que utiliza y ModelArts están en la misma región.
- La creación y gestión de aplicaciones de IA es gratuita.

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 más información sobre los parámetros, consulte **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 (_).
Version	Versión de la aplicación de IA a crear. Para la primera importación, el valor predeterminado es 0.0.1.
Label	Etiqueta de aplicación de IA. Se admite un máximo de cinco etiquetas.
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 más información sobre los parámetros, consulte **Tabla 2-6**.

Figura 2-2 Establecer una plantilla como origen del metamodelo

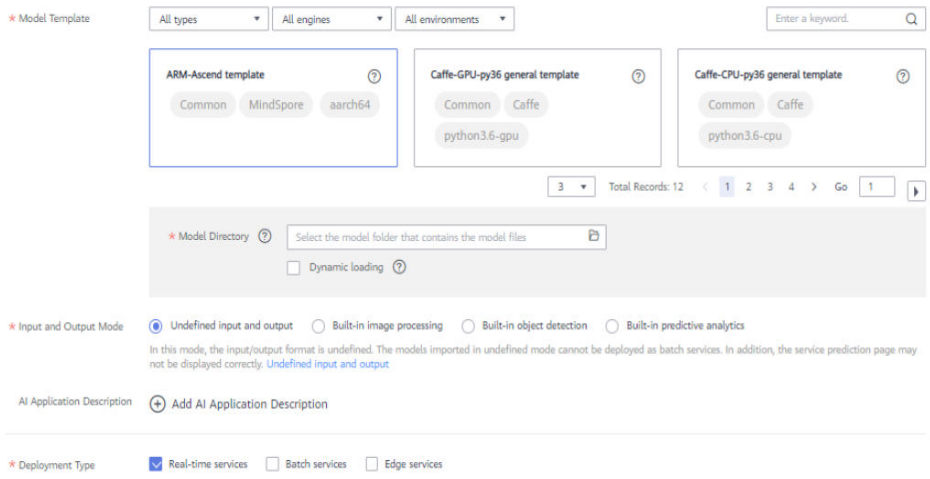


Tabla 2-6 Parámetros de la fuente del metamodelo

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 más información sobre las plantillas compatibles, consulte **Plantillas compatibles**.
Model Directory	Ruta OBS donde se guarda un modelo. Seleccione una ruta OBS para almacenar el plantilla en función de los requisitos de entrada de la plantilla de plantilla seleccionada. La ruta de acceso 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	Si esta función está habilitada, los archivos de modelo se descargan de OBS y, a continuación, se asocian con contenedores solo cuando se implementa el modelo. Esto acelera la implementación y las actualizaciones del modelo.
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 más información sobre los modos de entrada y salida admitidos, consulte **Modos de entrada y salida compatibles**.

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.
Deployment Type	Seleccione los tipos de servicio que puede implementar la aplicación. Al implementar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services** aquí, solo podrá implementar la aplicación de IA como 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, la implementación rápida de aplicaciones de IA y la publicación de aplicaciones de IA.

Acciones de seguimiento

Implementación de aplicaciones de IA como servicios: En la lista de aplicaciones de IA, haga clic en la flecha hacia abajo a la izquierda del nombre de una aplicación de IA para comprobar todas las versiones de la aplicación de IA. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** y seleccione un tipo de implementación en el cuadro de lista desplegable. La aplicación de IA se puede implementar en un tipo de implementación seleccionado durante la creación de la aplicación de IA.

2.2.3 Importación de un metamodelo desde OBS a través de configuraciones manuales

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

Prerrequisitos

- El modelo ha sido desarrollado y entrenado, y el tipo y la versión del motor de IA que utiliza es compatible con ModelArts. Para más detalles, consulte **Motores de IA compatibles para la inferencia de ModelArts**.
- 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, consulte **Introducción a las especificaciones del paquete modelo**, **Especificaciones para compilar el archivo de configuración del modelo** y **Especificaciones para la codificación de inferencia de modelo**.
- El paquete de modelo entrenado, el código de inferencia y el archivo de configuración se han subido a OBS.
- El directorio OBS que utiliza 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 más información sobre los parámetros, consulte **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.
Label	Etiqueta de aplicación de IA. Se admite un máximo de cinco etiquetas.
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 más información sobre los parámetros, consulte **Tabla 2-8**.

   Para el metamodelo importado de OBS, debe compilar el código de inferencia y el archivo de configuración haciendo referencia a **Introducción a las especificaciones del paquete modelo** y colocando 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.

Figura 2-3 Configuración del origen del metamodelo en OBS

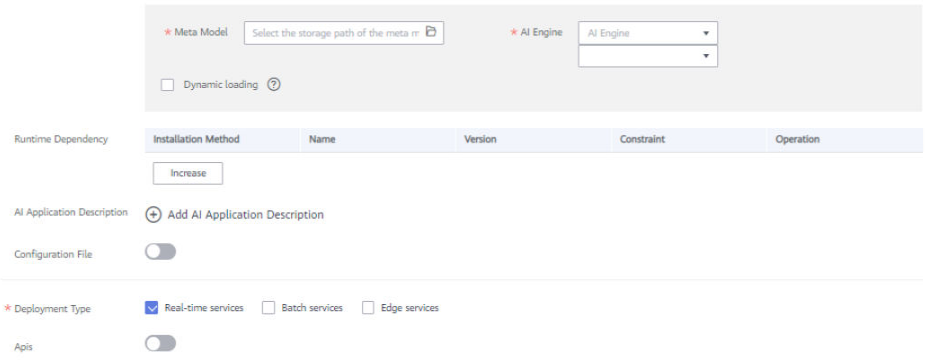


Tabla 2-8 Parámetros de la fuente del metamodelo

Parámetro	Descripción
Meta Model	Ruta OBS para almacenar el metamodelo. La ruta de acceso 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.
Dynamic loading	Si esta función está habilitada, los archivos de modelo se descargan de OBS y, a continuación, se asocian con contenedores solo cuando se implementa el modelo. Esto acelera la implementación y las actualizaciones del modelo.
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 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. Después de eso, puede modificar la configuración del modelo estableciendo AI Engine, Runtime Dependency y Apis.
Deployment Type	Seleccione los tipos de servicio que puede implementar la aplicación. Al implementar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services** aquí, solo podrá implementar la aplicación de IA como servicio en tiempo real después de crearla.
Apis	Cuando habilita esta función, puede editar las API RESTful para definir los formatos de entrada y salida de aplicaciones de IA.

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, la implementación rápida de aplicaciones de IA y la publicación de aplicaciones de IA.

Acciones de seguimiento

Implementación de aplicaciones de IA como servicios: En la lista de aplicaciones de IA, haga clic en la flecha hacia abajo a la izquierda del nombre de una aplicación de IA para

comprobar todas las versiones de la aplicación de IA. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** y seleccione un tipo de implementación en el cuadro de lista desplegable. La aplicación de IA se puede implementar en un tipo de implementación seleccionado durante la creación de la aplicación de IA.

2.2.4 Creación e importación de una imagen de modelo

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

Prerrequisitos

- Para obtener más información sobre las especificaciones y la descripción de las imágenes personalizadas.
- El directorio OBS que utiliza 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 más información sobre los parámetros, consulte **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.
Label	Etiqueta de aplicación de IA. Se admite un máximo de cinco etiquetas.
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 más información sobre los parámetros, consulte **Tabla 2-10**.

Figura 2-4 Establecer una imagen contenedora como fuente del metamodelo

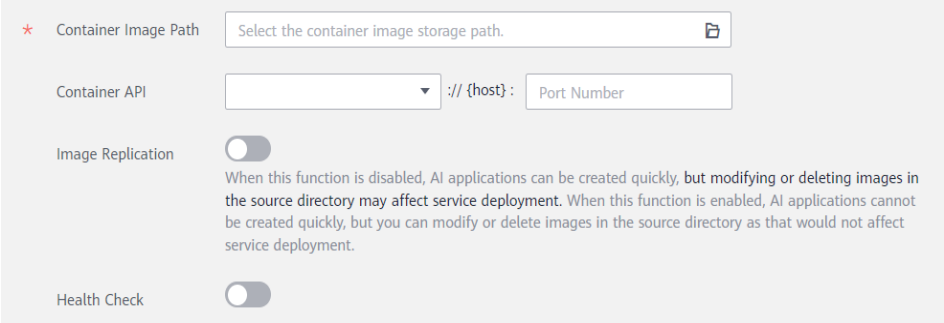


Tabla 2-10 Parámetros de la fuente del metamodelo

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 más información sobre cómo crear una imagen personalizada. **NOTA** La imagen del modelo que seleccione se compartirá con el administrador, así que asegúrese de tener el permiso para compartir la imagen (las imágenes compartidas con otras cuentas no son compatibles). Cuando implementa un servicio, el ModelArts implementa la imagen como un servicio de inferencia. Asegúrese de que la imagen se puede iniciar correctamente y proporcione una interfaz de inferencia.
Container API	Protocolo y número de puerto para iniciar un modelo. Este parámetro es opcional.
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 a la implementación 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 a la implementación del servicio.

Parámetro	Descripción
Health Check	Control de salud en un modelo. Este parámetro es configurable solo cuando la API de comprobación de estado está configurada en la imagen personalizada. De lo contrario, la implementación de la aplicación de IA fallará. ● Health Check URL: Ingrese la URL de comprobación de estado. ● Health Check Period: Introduzca un número entero mayor que 0. El objeto es segundo. ● Maximum Failures: introduzca un número entero mayor que 0. 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.
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 puede implementar la aplicación. Al implementar un servicio, solo están disponibles los tipos de servicio seleccionados aquí. Por ejemplo, si solo selecciona **Real-time services** aquí, solo podrá implementar la aplicación de IA como servicio en tiempo real después de crearla.
Apis	Cuando habilita esta función, puede editar las API RESTful para definir los formatos de entrada y salida de aplicaciones de IA.

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, la implementación rápida de aplicaciones de IA y la publicación de aplicaciones de IA.

Acciones de seguimiento

Implementación de aplicaciones de IA como servicios: En la lista de aplicaciones de IA, haga clic en la flecha hacia abajo a la izquierda del nombre de una aplicación de IA para

comprobar todas las versiones de la aplicación de IA. Busque la fila que contiene la versión de destino, haga clic en **Deploy** en la columna **Operation** y seleccione un tipo de implementación en el cuadro de lista desplegable. La aplicación de IA se puede implementar en un tipo de implementación seleccionado durante la creación de la aplicación de IA.

2.3 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 del 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 2-11 Detalles sobre una aplicación de IA

Categoría	Parámetro	Descripción
Información básica	Name	Nombre de una aplicación de IA
	Status	Estado actual de una aplicación de IA
	Version	Versión actual de una aplicación de IA
	ID	ID de una aplicación de IA
	Size	Tamaño de una aplicación de IA
	Runtime Environment	Entorno de tiempo de ejecución del que depende el metamodelo
	Meta Model Source	Ruta al metamodelo
	AI Engine	Motor de IA utilizado por una aplicación de IA
	Deployment Type	Tipos de servicios que puede implementar una aplicación de IA
	Model Source	Fuente de un modelo, que puede ser ExeML, algoritmo integrado o algoritmo personalizado
	Inference Code	Ruta al código de inferencia
	Dynamic loading	Si una aplicación de IA admite la carga dinámica
	Description	Haga clic en el botón de edición para agregar la descripción de una aplicación de IA.

Categoría	Parámetro	Descripción
	AI Application Description	Descripción documento agregado durante la creación de una aplicación de IA
	Associated Training Job	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.
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.
Constraint	-	Restricciones en la implementación, como el modo de solicitud de API, la arquitectura del sistema implementada y los tipos de tarjetas de aceleración compatibles
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.

2.4 Gestión 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 modelos basados en versiones.

Prerrequisitos

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**. Se muestra la página **Create Version**. Defina los parámetros

relacionados siguiendo las instrucciones en **Creación de una aplicación de IA** y haga clic en **Next**.

Eliminación de una versión

En la página **AI Application Management > AI Applications**, haga clic en la flecha hacia abajo a la izquierda del nombre de la aplicación de IA para expandir una 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

No se puede recuperar una versión eliminada. Tenga cuidado cuando realice esta acción.

3 Implementación de aplicaciones de IA como servicios en tiempo real

3.1 Implementación como servicio en tiempo real

Después de preparar una aplicación de IA, puede implementar la aplicación de IA como un servicio en tiempo real y predecir y llamar al servicio.

NOTA

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

Prerrequisitos

- Se han preparado los datos. Específicamente, ha creado una aplicación de IA en el estado **Normal** de ModelArts.
- Asegúrese de que la cuenta no está en mora. Los recursos se consumen cuando los servicios se están ejecutando.

Procedimiento

1. Inicie sesión en la consola de gestión del 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. En la lista de servicios en tiempo real, haga clic en **Deploy** en la esquina superior izquierda. Se muestra la página **Deploy**.
3. Establezca parámetros para un servicio en tiempo real.
   a. Establezca información básica acerca de la implementación del modelo. Para obtener más información sobre los parámetros, consulte **Tabla 3-1**.

Tabla 3-1 Parámetros básicos del despliegue del modelo

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 habilitada 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.

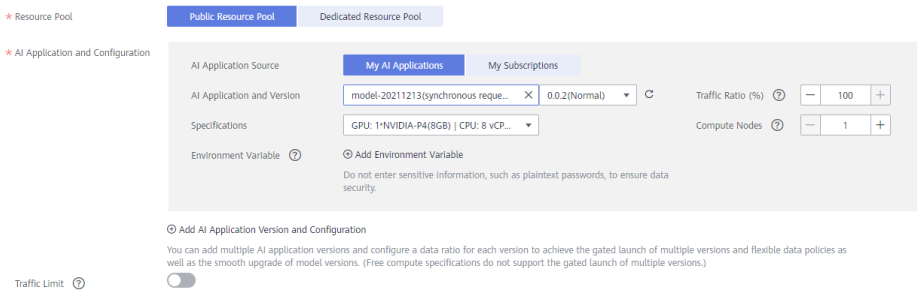
b. Ingrese la información clave, incluidas las configuraciones del grupo de recursos y de la aplicación de IA. Para más detalles, consulte **Tabla 3-2**.

Tabla 3-2 Parámetros

Parámetro	Subparámetro	Descripción
Resource Pool	Public resource pools	Las instancias del grupo de recursos públicos pueden ser del tipo CPU o GPU. Los estándares de fijación de precios para los grupos de recursos con diferentes tipos de instancia son diferentes. Para obtener más información, consulte **Detalles de precios del producto**. El fondo de recursos públicos solo admite el modo de facturación de pago por uso.
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**.

Parámetro	Subparámetro	Descripción
	Traffic Ratio (%)	Establezca la proporción de tráfico del nodo de instancia actual. Las solicitudes de llamadas de servicio se asignan a la versión actual en función de esta proporción. Si implementa solo una versión de una aplicación de IA, establezca este parámetro en 100%. Si selecciona varias versiones para el lanzamiento cerrado, asegúrese de que la suma de los ratios de tráfico de varias versiones sea del 100%.
	Specifications	Seleccione las especificaciones disponibles según la lista mostrada en la consola. Las especificaciones en gris no se pueden utilizar en el entorno actual.
	Compute Nodes	Establezca el número de instancias para la versión actual de la aplicación de IA. Si establece **Instances** en 1, se utiliza el modo de cómputo independiente. Si establece **Instances** en un valor mayor que 1, se utiliza el modo de cómputo distribuida. 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.
	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 utilizar el inicio gris para actualizar sin problemas la versión de la aplicación de IA. **NOTA** Las especificaciones de cómputo libre no admiten el lanzamiento gris de varias versiones.
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.
Application Authentication	Application	Deshabilitada por defecto. Para habilitar esta función, consulte **Acceso autenticado mediante una aplicación** para obtener más detalles y establezca los parámetros según sea necesario.

Figura 3-1 Establecer la información de la aplicación de IA.



4. Después de confirmar la información introducida, complete la implementación del servicio como se le solicite. En general, los trabajos de implementación de servicios 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

   Después de implementar un servicio en tiempo real, se inicia inmediatamente. Durante la carrera, se le cobrará en función de los recursos seleccionados.During the running, you will be charged based on your selected resources.

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

3.2 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, consulte **Tabla 3-3**.

Tabla 3-3 Parámetros de servicio en tiempo real

Parámetro	Descripción
Name	Nombre del servicio en tiempo real.
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

Parámetro	Descripción
Failed Calls/ Total Calls	Número de llamadas de servicio, que se cuenta desde el momento en que se creó el servicio. Si se cambia el número de aplicaciones de IA o se invoca un servicio cuando una aplicación de AI no está lista, no se cuenta el número de llamadas.
Description	Descripción del servicio, que se puede editar después de hacer clic en el botón de edición en el lado derecho.
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, consulte **Modifying Customized Settings**.
Traffic Limit	Número máximo de veces que se puede acceder a un servicio en un segundo.

3. Puede cambiar entre las pestañas de la página de detalles de un servicio en tiempo real para ver más detalles. Para más detalles, consulte **Tabla 3-4**.

Tabla 3-4 Detalles del servicio

Parámetro	Descripción
Usage Guides	Muestra la dirección de API, la información de la aplicación de IA, los parámetros de entrada y los parámetros de salida. Puede hacer clic en para copiar la dirección API para llamar al servicio. Si se admite la autenticación de la aplicación, puede ver los detalles de gestión de la dirección de la API y de la autorización, incluidos el nombre de la aplicación, el AppKey y el AppSecret en **Usage Guides**. También puede agregar o cancelar la autorización para una aplicación.
Prediction	Realiza una prueba de predicción en el servicio en tiempo real. Para más detalles, consulte **Prueba del servicio implementado**.
Configuration Updates	Muestra **Existing Configuration** y **Historical Updates**. ● **Existing Configuration**: incluye el nombre de la aplicación de IA, la versión, el estado, la variante de nodo de cómputo, el número de nodos de cómputo y la relación de tráfico. ● **Historical Updates**: muestra información histórica de la aplicación de IA.

Parámetro	Descripción
Monitoring	Muestra **Resource Usage** y **AI Application Calls**. ● **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**.
Logs	Muestra la información de log de cada aplicación de IA en el 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.

Modifying Customized Settings

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 3-5 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.

Parámetro	Obligatorio	Descripción
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 implementan varias versiones de un servicio en tiempo real para el inicio cerrado, se pueden usar configuraciones personalizadas para distribuir el tráfico por usuario.

  Tabla 3-6 Variables incorporadas

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

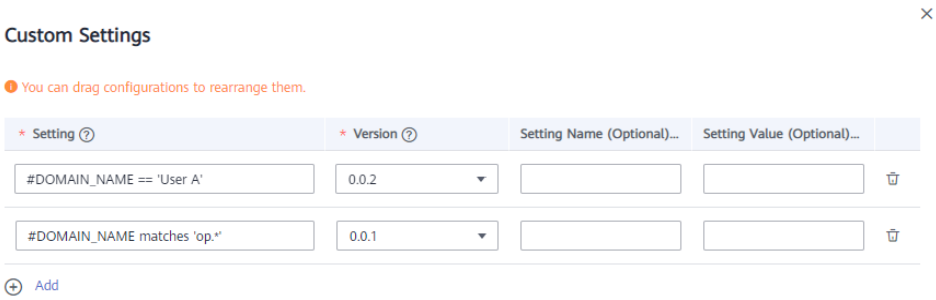
    Si el nombre de cuenta en la solicitud de inferencia comienza con **op**, la versión especificada coincide.

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


Tabla 3-7 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}	Coincidencia para el número especificado por *n*, un entero no negativo. 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]	Coincide con cualquier carácter único contenido en un juego de caracteres. Por ejemplo, [abc] puede coincidir con a en **plain**.

Figura 3-2 Distribución del tráfico por usuario



- Si se implementan 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_**, indicando que la cabecera es referenciada como una 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.

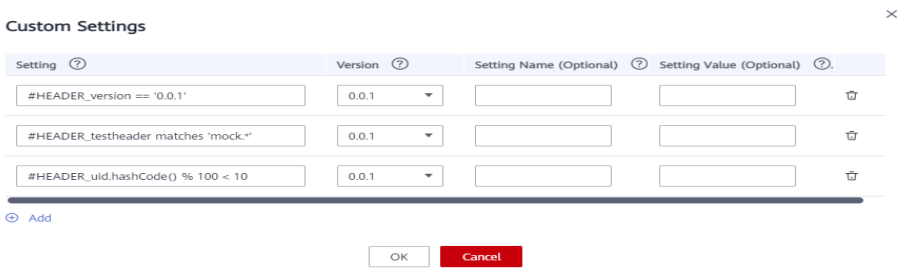
```
#HEADER_testheader matches 'mock.*'
```

– 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

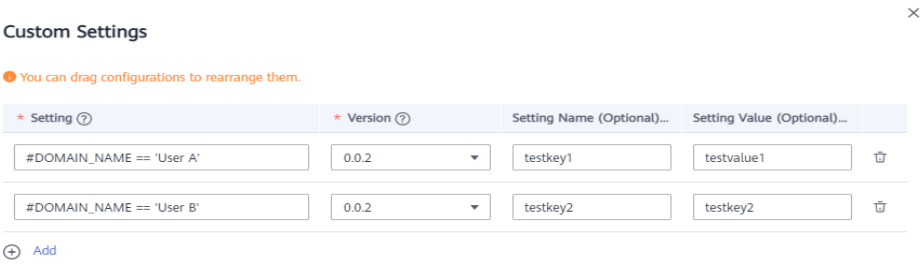


- Si una versión de servicio en tiempo real admite diferentes configuraciones de ejecución, puede usar **Setting Name** y **Setting Value** para especificar parámetros de ejecución personalizados para que diferentes usuarios puedan usar diferentes configuraciones de ejecución.

  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



3.3 Prueba del servicio implementado

Después de implementar una aplicación de IA como servicio en tiempo real, puede depurar código o agregar archivos para realizar pruebas en la página de ficha **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:

1. **Predicción de texto JSON**: Si el tipo de entrada de la aplicación de IA del servicio implementado 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.
2. **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.

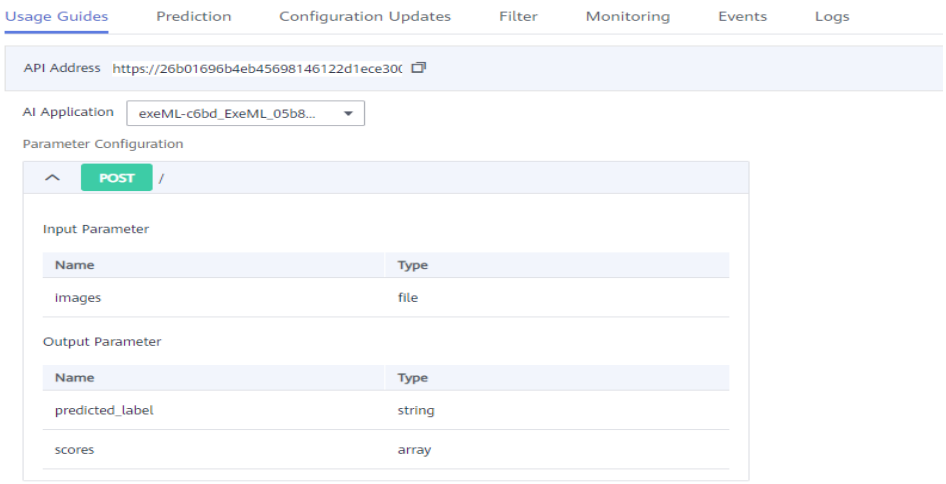
NOTA

- Si el tipo de entrada es imagen, el tamaño de una sola imagen debe ser inferior a 8 MB.
- Se admiten los siguientes tipos de imágenes: png, psd, jpg, jpeg, bmp, gif, webp, psd, svg y tiff.
- Si se utilizan variantes Ascend durante la implementación del servicio, las imágenes PNG con transparencia no se pueden predecir porque Ascend solo admite imágenes RGB-3.
- Esta función se utiliza para la puesta en marcha. En la producción real, se le aconseja llamar a las API. Puede seleccionar **Acceso autenticado mediante un token**, **Acceso autenticado mediante un AK/SK** o **Acceso autenticado mediante una aplicación** en función del modo de autenticación.

Parámetros de entrada

Para el servicio que ha implementado, puede obtener información sobre sus parámetros de entrada del servicio, es decir, el tipo de solicitud de entrada mencionado anteriormente, en la página de ficha **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 página de ficha **Usage Guides** dependen del origen 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, es decir, el código de inferencia y el archivo de configuración son compilados por usted mismo (**Especificaciones para Compilar el archivo de configuración del modelo)**, la página de ficha **Usage Guides** solo visualiza los archivos de configuración. 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 con una plantilla de plantilla, los parámetros de entrada y salida varían con la plantilla. Para obtener más información, consulte la descripción en **Introducción a las plantillas de modelo**.

Predicción de texto JSON

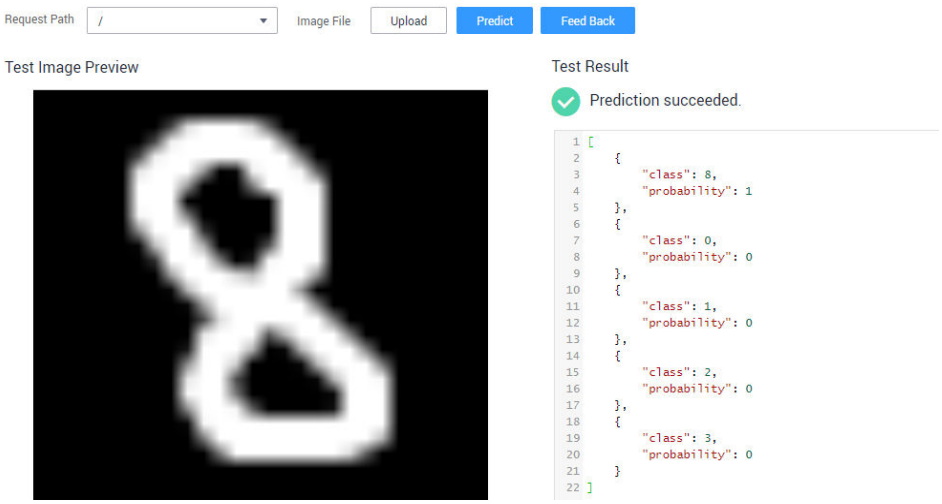
1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment** > **Real-Time Services**.
2. En la página **Real-Time Services**, haga clic en el nombre del servicio de destino. Se muestra la página de detalles del servicio. En la página de ficha **Prediction**, introduzca el código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

   This example is a metamodel trained using ExeML. The input type is officially defined by ModelArts and cannot be changed.

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. En la página de la ficha **Prediction**, haga clic en **Upload** y seleccione un archivo de prueba. Después de cargar el archivo correctamente, haga clic en **Predict** para realizar una prueba de predicción. En **Figura 3-7**, se muestran la etiqueta, las coordenadas de posición y la puntuación de confianza.

Figura 3-7 Predicción de imágenes



3.4 Acceso a los servicios en tiempo real

3.4.1 Acceso autenticado mediante un token

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 RESTful estándar para que los usuarios llamen. 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)**. (Postman se recomienda para Windows.)
- **Método 2: Ejecutar el comando cURL para enviar una solicitud de inferencia** (Los comandos curl se recomiendan para Linux.)
- **Método 3: Usar Python para enviar una solicitud de inferencia**.
- **Método 4: Usar Java para enviar una solicitud de inferencia**.

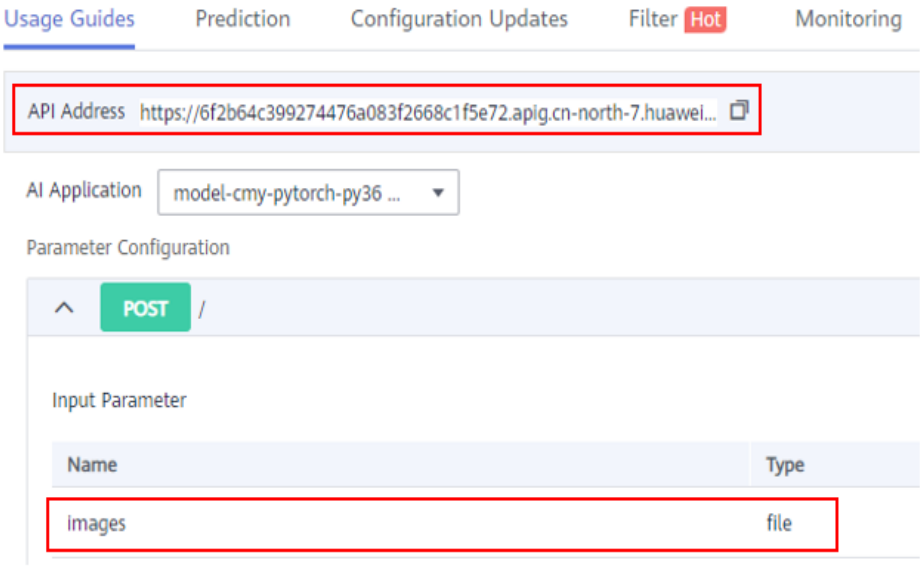
Prerrequisitos

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 dirección API es la URL del servicio en tiempo real.

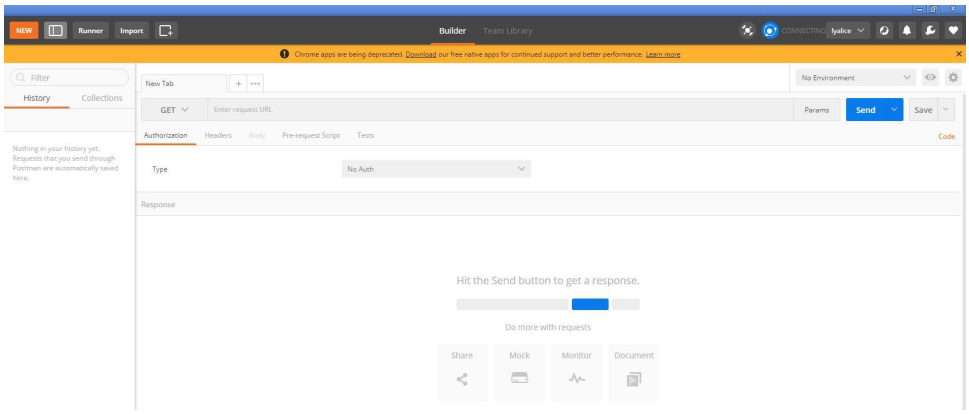
Figura 3-8 Obtención de información sobre un servicio en tiempo real



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. Abra Postman. **Figura 3-9** muestra la interfaz de Postman.

Figura 3-9 Interfaz de Postman

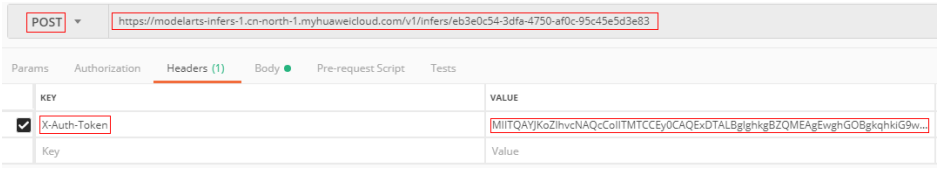


3. Establezca los parámetros en Postman. A continuación se utiliza la clasificación de imágenes como ejemplo.
   - Seleccione una tarea POST y copie URL de la API en el cuadro de texto POST. En la página de pestaña **Headers**, establezca **Key** en **X-Auth-Token** y **Value** en el token de usuario.

     NOTA

     También puede utilizar la AK y la SK para cifrar las solicitudes de llamadas a la API. Para obtener más información, consulte **SDK Reference > Session Authentication > AK/SK-based Authentication**.

Figura 3-10 Ajustes de parámetros

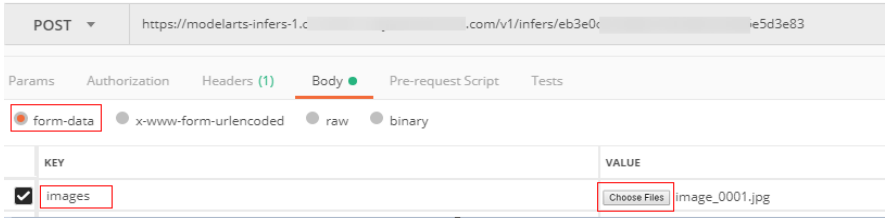


– En la página de la ficha **Body**, la entrada de archivo y la entrada de texto están disponibles.

  ■ **File input**

  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 3-11**.

  Figura 3-11 Configuración de parámetros en la página de ficha **Body**



  ■ **Text input**

  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 llama a una API, el sistema proporciona un UUID. Cuando se devuelve el resultado de la inferencia, 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 son determinados por la aplicación de IA, tales como **sepal_length** y **sepal_width** en este ejemplo.

4. Después de establecer los parámetros, haga clic en **send** para enviar la solicitud. El resultado se mostrará en **Response**.

   – Resultado de inferencia usando entrada de archivo: **Figura 3-12** muestra un ejemplo. Los valores de campo en el resultado devuelto varían con la aplicación de IA.

– Resultado de inferencia usando entrada de texto: **Figura 3-13** muestra un ejemplo. El cuerpo de la solicitud contiene **meta** y **data**. Si la solicitud contiene uuid, uuid será devuelto en la respuesta. De lo contrario, 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 3-12 Resultado de inferencia de archivo

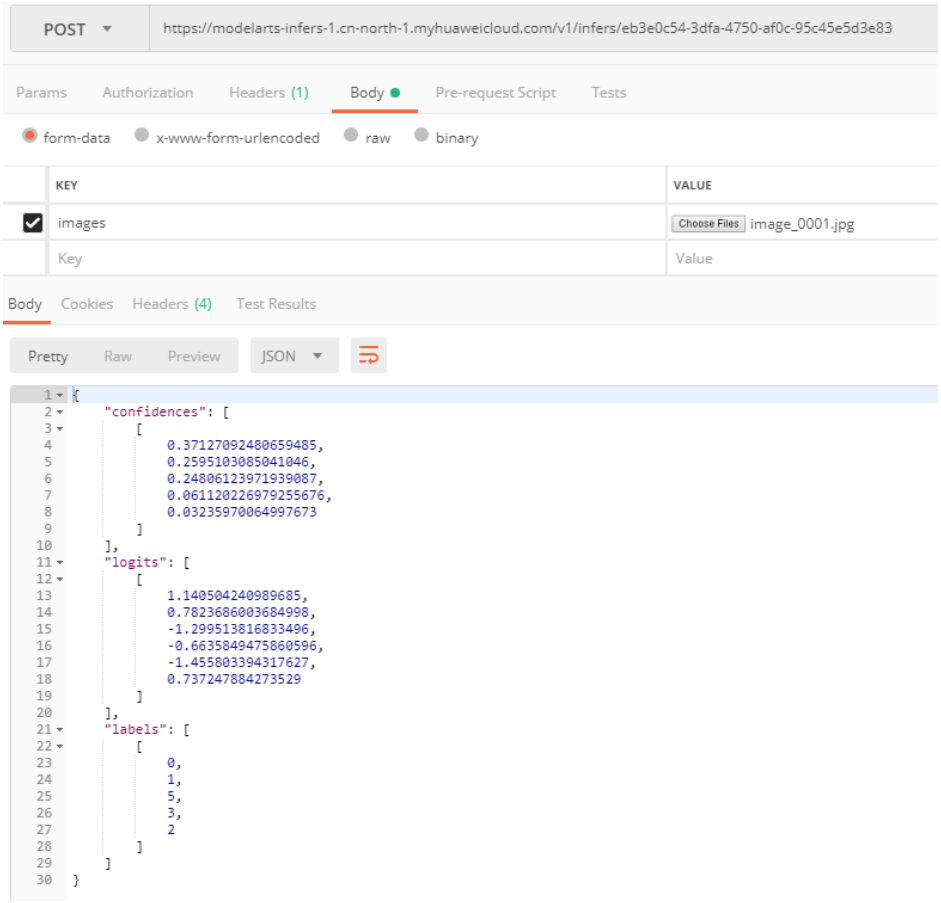
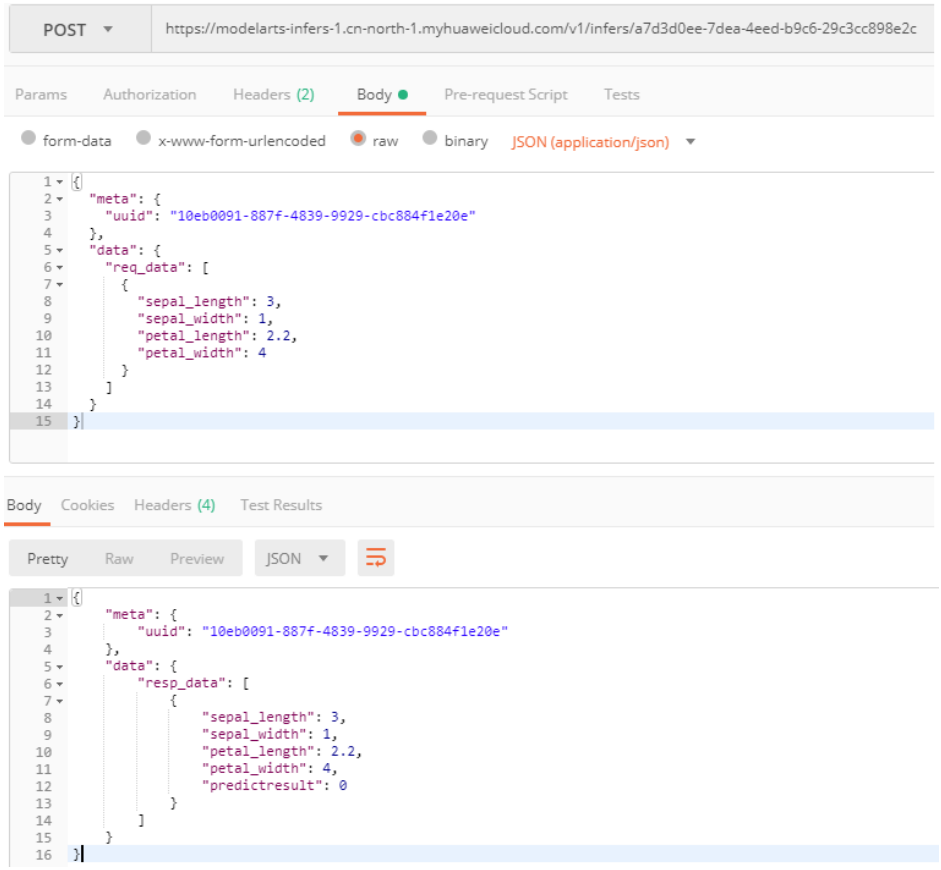


Figura 3-13 Resultado de inferencia de texto



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 sitios web SSL sin utilizar un certificado de seguridad.

  – -F indica la entrada del 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 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.
   - **File input**

     ```
     # 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. Se recomienda que el nombre del parámetro sea el mismo que el del parámetro de entrada del tipo de archivo. El archivo **images** obtenido en **Prerrequisitos** se usan 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:

     ```
     # 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)
```

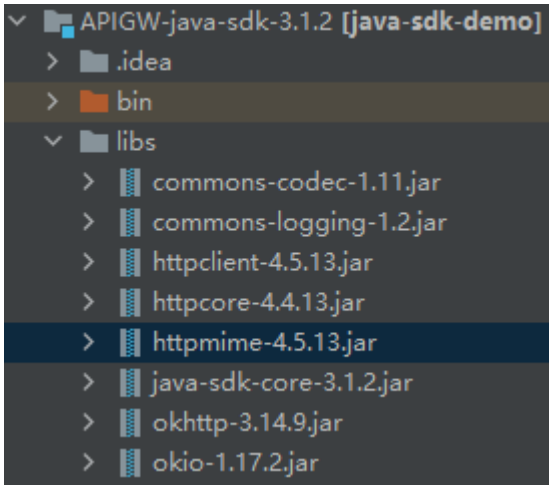
El nombre de **body** viene determinado por el parámetro de entrada del servicio en tiempo real. Se recomienda que el nombre del parámetro sea el mismo que el del parámetro de entrada del tipo de string.

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 3-14** muestra una biblioteca de dependencias Java completa.

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

      Figura 3-14 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. Se recomienda que el nombre del parámetro sea el mismo que el del parámetro de entrada del tipo de archivo. El archivo **images** obtenido en **Prerrequisitos** se usan 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.

3.4.2 Acceso autenticado mediante un 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 RESTful estándar para que los usuarios llamen. Los usuarios pueden llamar 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, consulte **Autenticación basada en AK/SK**. 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 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 offset 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 AK/SK

Si un par 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 3-15 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 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.

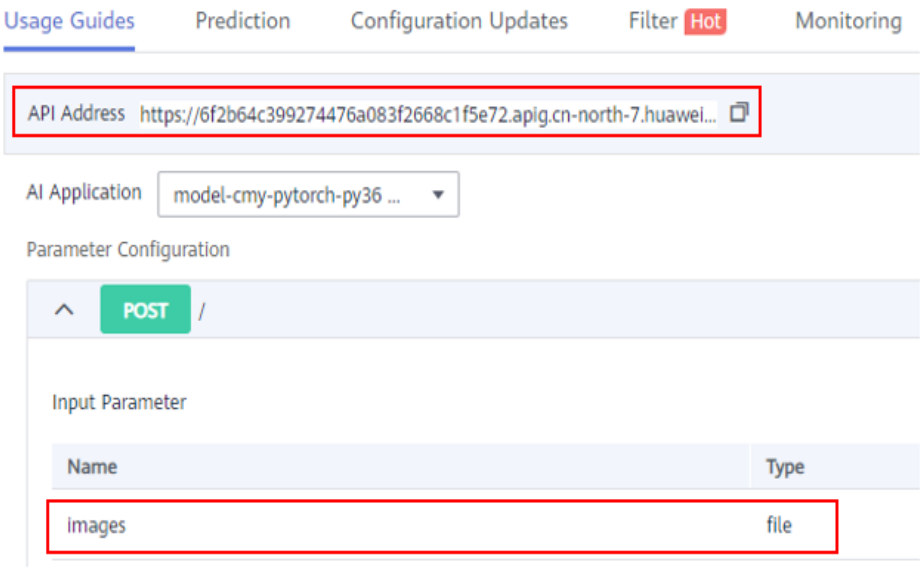
Obtención de información sobre un servicio en tiempo real

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

1. Inicie sesión en la consola de gestión del 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 dirección API es la URL del servicio en tiempo real.

Figura 3-16 Obtención de información sobre 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
from apig_sdk import signer

if __name__ == '__main__':
    # Config url, ak, sk and file path.
    url = "URL of the real-time service"
    ak = "AK"
    sk = "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 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 más información sobre los parámetros de **files**, consulte **Tabla 3-8**.

Tabla 3-8 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**: text/plain ● **jpg/jpeg**: image/jpeg ● **png**: image/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"
    ak = "AK"
    sk = "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. Se recomienda que el nombre del parámetro sea el mismo que el del parámetro de entrada del tipo de string.

Método 2: 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. Cree un cuerpo de solicitud Java para inferencia.

   En el SDK de Java 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 ak = "AK";
        String Sk = "SK";
        String body = "{}";

        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.
            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.

3.4.3 Acceso autenticado mediante una aplicación

Puede habilitar la autenticación de aplicaciones al implementar 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 llamar 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**: Habilite la autenticación de aplicaciones y cree una aplicación.
2. **Gestión de la autorización de servicios en tiempo real**: gestione la aplicación creada, incluyendo la visualización, restablecimiento o eliminación de la aplicación, vincule o desvincule los servicios en tiempo real para la aplicación, y la obtención de AppKey/AppSecret o AppCode.
3. **Autenticación de aplicaciones**: Se requiere autenticación para llamar a una API que admita 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**

Prerrequisitos

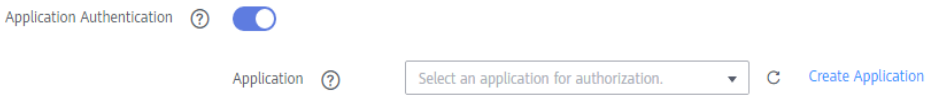
- Se han preparado datos. Específicamente, ha creado una aplicación de IA en el estado **Normal** de ModelArts.
- La cuenta no está en mora para garantizar los recursos disponibles para la ejecución del servicio.
- Se ha obtenido la ruta local al archivo de inferencia. La ruta puede ser una ruta absoluta (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 implementar un modelo como servicio en tiempo real, puede habilitar la autenticación de aplicaciones. También puede modificar un servicio en tiempo real implementado para admitir la autenticación de aplicaciones.

1. Inicie sesión en la consola de gestión ModelArts y elija **Service Deployment > Real-Time Services**.
2. Habilite la autenticación de aplicaciones.
   - Al implementar 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 implementado, 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 3-17 Habilitar la autenticación de aplicaciones

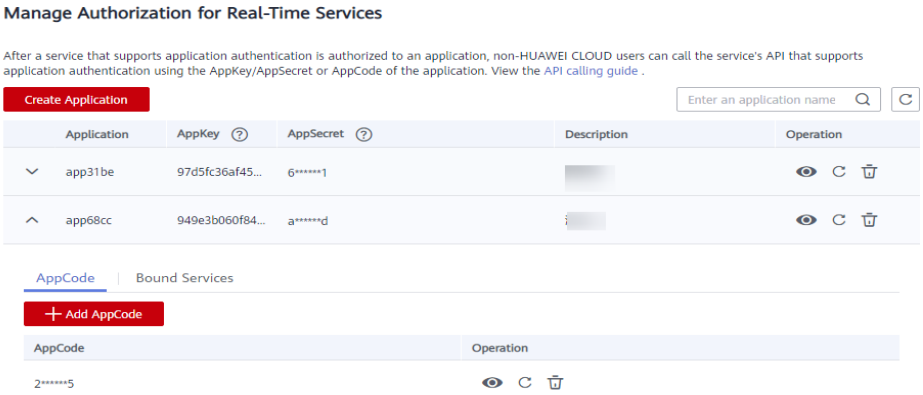
     

3. Seleccione una solicitud de autorización en la lista desplegable. Si no hay ninguna aplicación disponible, cree una de la siguiente manera:
   - Haga clic en **Create Application** a la derecha, introduzca el nombre y la descripción de la aplicación y haga clic en **OK**. De forma predeterminada, 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, consulte **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 llamar 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 implementar un servicio en tiempo real. En la página Service Deployment > Real-Time Services, haga clic en **Authorize**. Se muestra la página **Manage Authorization of Real-Time Services**. En esta página, 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 3-18 Gestión de la autorización para servicios en tiempo real



- Creación de una aplicación

  Haga clic en **Create Application**, 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.

- Ver, restablecer o eliminar una aplicación

  Para ver, restablecer o eliminar una aplicación, haga clic en el icono correspondiente de la columna **Operation** de la aplicación. Después de crear una aplicación, el AppKey y el 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 vinculados a la aplicación. Haga clic en **Unbind** en la columna **Operation** para cancelar el enlace. Entonces, esta API no se puede llamar.

- Obtención de AppKey/AppSecret o AppCode

  La autenticación de la aplicación es necesaria para las llamadas a la API. Los AppKey y AppSecret se generan automáticamente durante la creación de la aplicación. El AppCode se obtiene mediante la adición de un AppCode.

Figura 3-19 Agregar el AppCode

Autenticación de aplicaciones

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

Cuando se utiliza la autenticación de aplicaciones y el modo de autenticación simple está habilitado, puede utilizar AppKey/AppSecret para la firma y la verificación, o AppCode para

la autenticación simple de las solicitudes de API. ModelArts utiliza la autenticación simple de forma predeterminada. Se recomienda la autenticación basada en AppKey/AppSecret porque es más segura que la autenticación basada en AppCode.

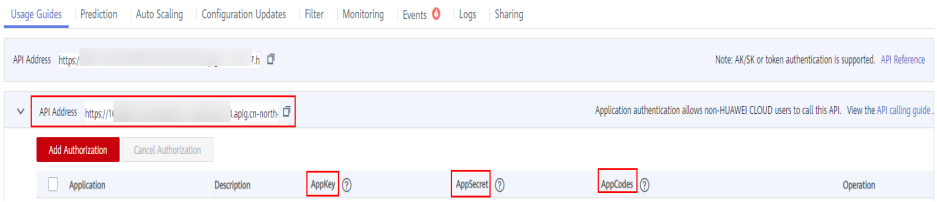
- Autenticación basada en AppKey/AppSecret: AppKey y AppSecret se utilizan para cifrar una solicitud, identificar al remitente y evitar que se modifique la solicitud. Cuando utilice la autenticación basada en AppKey/AppSecret, utilice un SDK de firma dedicado para firmar solicitudes.
  - AppKey: ID de 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 simple. Cuando se llama a una API, el parámetro apikey (valor:AppKey) se agrega al encabezado de solicitud HTTP para acelerar la autenticación.
- Autenticación basada en AppCode: las solicitudes se autentican mediante AppCodes.

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

Puede obtener la API, AppKey/AppSecret y AppCode desde la pestaña **Usage Guides** en la página de detalles del servicio (consulte **Figura 3-20**). o desde la página de gestión de autorización de servicio en tiempo real (consulte **Figura 3-18**). Asegúrese de que se utiliza la dirección API que se muestra en el cuadro rojo de la siguiente figura.

Figura 3-20 Obtención de la dirección 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 más información sobre los parámetros de **files**, consulte **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**: text/plain ● **jpg/jpeg**: image/jpeg ● **png**: image/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. Se recomienda que el nombre del parámetro sea el mismo que el del parámetro de entrada del tipo de string.

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 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. Se recomienda que el nombre del parámetro sea 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)
```

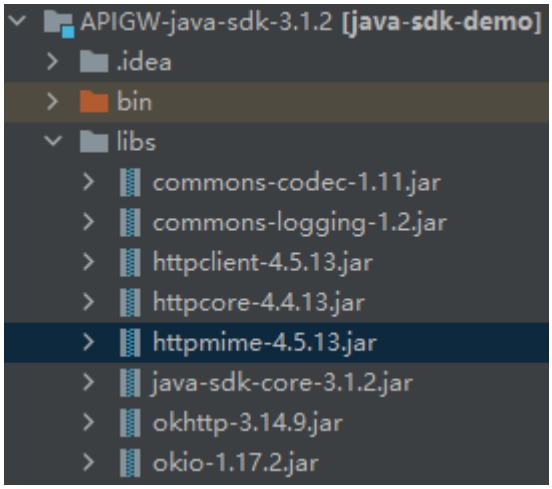
El nombre de **body** viene determinado por el parámetro de entrada del servicio en tiempo real. Se recomienda que el nombre del parámetro sea el mismo que el del parámetro de entrada del tipo de string.

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 3-21** muestra una biblioteca de dependencias Java completa.

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

Figura 3-21 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. Se recomienda que el nombre del parámetro sea 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.

3.5 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.

Prerrequisitos

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 llamar a las API de servicio en tiempo real. Para obtener detalles sobre cómo llamar 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 llamar a las API. Puede buscar y obtener las guías para llamar a las API RESTful estándar en Internet.

4 Implementación de aplicaciones de IA como servicios por lotes

4.1 Implementación como servicio por lotes

Después de preparar una aplicación de IA, puede implementarla como un servicio por lotes. La página **Service Deployment > Batch Services** muestra todos los servicios por lotes. Puede introducir un nombre de servicio en el cuadro de búsqueda en la esquina superior derecha y hacer clic en para consultar el servicio.

Prerrequisitos

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

Fondo

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

Procedimiento

1. Inicie sesión en la consola de gestión del 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. Establezca la información básica, incluidos **Name** y **Description**. 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, consulte **Tabla 4-1**.

Tabla 4-1 Parámetros

Parámetro	Descripción
AI Application Source	Seleccione **My AI Applications** o **My Subscriptions** según sus requisitos.
AI Application and Version	Seleccione la aplicación y la versión de IA que están en el estado Normal.
Input Path	Seleccione el directorio OBS donde se van a cargar los datos. Seleccione una carpeta o un archivo **.manifest**. Para obtener más información sobre las especificaciones del archivo **.manifest**, consulte **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. Para usar chino, establezca el formato de codificación de archivo en UTF-8. Puede convertir el formato de codificación de archivo con código o abrir el archivo CSV con el Bloc de notas y establecer el formato de codificación en la ventana que se muestra después de hacer clic en **Save As**.
Request Path	URI utilizado para llamar a la interfaz de aplicación de IA en un servicio por lotes, y también a 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.

Parámetro	Descripción
Relación de mapeo	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/boolean, es necesario establecer el parámetro index. Para obtener más información sobre la regla de asignación, consulte **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 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.
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 **Instances** en 1, se utiliza el modo de cómputo independiente. Si establece **Instances** en un valor mayor que 1, se utiliza el modo de cómputo distribuida. 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.

4. Después de establecer los parámetros, implemente el modelo como un servicio por lotes según se le solicite. En general, los trabajos de implementación de servicios 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**
>
> Después de implementar un servicio por lotes, se inicia inmediatamente. Durante la carrera, 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 la plataforma de inferencia admiten el archivo de manifiesto. El archivo de manifiesto describe la entrada y salida de datos.

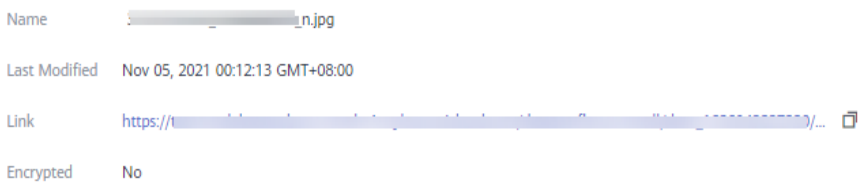
Ejemplo de archivo de manifiesto de entrada

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

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

- Requisitos de 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. *<OBS path>*/{{*Bucket name*}}/{{*Object name*}} se puede utilizar para acceder a sus propios datos de OBS. Puede acceder a la ruta para obtener un objeto en OBS.

     Figura 4-1 Ruta OBS para almacenar datos

     ii. Enlace compartido generado por OBS, incluyendo información de firma. Se aplica al acceso a los datos de OBS de otros usuarios.

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-{{index}}.manifest
```



```
|   |   ├── infer-result
|   |   |   ├── 1.jpg_result.txt
|   |   |   ├── 2.jpg_result.txt
```

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

```
{"source": "/obs-data-bucket/test/data/1.jpg","inference-loc": "<obs path>/
test-bucket/test/infer-result/1.jpg_result.txt"}
{"source ": "/xgboosterdata/2.jpg?
AccessKeyId=2Q0V0TQ461N26DDL18RB&Expires=1550611914&Signature=wZBttZj5QZrReDhz
1uDzwve8GpY%3D&x-obs-security-
token=gQpzb3V0aGNoaW5hixvY8V9a1SnsxmGoHYmB1SArYMyqnQT-
ZaMSxHvl68kKLAy5feYvLDMNZWxzhBZ6Q-3HcoZMh9gISwQOVBwm4ZytB_m8sg1fL6isU7T3CnoL9j
mvDGgT9VBC7dC1EyfSJrUcqfB...",  "inference-loc": "obs://test-bucket/test/
infer-result/2.jpg_result.txt"}
```

- Formato de archivo:
  a. El nombre del archivo es **infer-result-{{index}}.manifest**, donde index es el ID de instancia. Cada instancia en ejecución de un servicio por lotes genera un archivo de manifiesto.
  b. El directorio **infer-result** se crea en el directorio de manifiesto para almacenar el resultado del procesamiento del archivo.
  c. El contenido del archivo está en formato JSON. Cada fila describe el resultado de salida de una pieza de datos de entrada.
  d. El contenido contiene varios campos:
     i. **source**: descripción de datos de entrada, que es la misma que la del archivo de manifiesto de entrada
     ii. **inference-loc**: ruta de resultado de salida en el formato de **<obs path>/{{Bucket name}}/{{Object name}}**

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.

Suponga que el parámetro apis en el archivo de configuración usado por su modelo es el siguiente:

```
[
    {
        "protocol": "http",
        "method": "post",
        "url": "/",
        "request": {
            "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"
                                        }
                                    }
                                }
                            ]
                        }
                    }
                }
            }
        }
    }
]
```

En este punto, la relación de mapeo correspondiente se muestra a continuación. La consola de gestión ModelArts resuelve automáticamente la relación de asignación desde el archivo de configuración. Al llamar a una API de ModelArts escriba usted mismo la relación de asignación de acuerdo con 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
                                }
                            }
                        }
                    ]
                }
            }
        }
    }
}
```

Los datos para inferencia, es decir, los datos CSV, están en el siguiente formato. Los datos deben estar separados por comas (,).

```
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 asignación definida, la solicitud de inferencia se muestra a continuación. El formato es similar al formato utilizado por el servicio en tiempo real.


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

4.2 Consulta del resultado de la predicción del servicio por lotes

Al implementar 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 el estado **Running 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 el estado **Running 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 OBS detallada junto a **Output Path**, cambiar a la ruta y obtener los resultados de predicción de servicio por lotes, incluyendo 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 resultado para cada archivo de audio en el 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.

5 Actualización de un servicio

Para un servicio implementado, puede modificar su información básica para que coincida con los cambios del servicio y 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

Prerrequisitos

El servicio se ha 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 5-1 Escenarios para la actualización sin golpes

Fuente de metamodelo para crear una aplicación de IA	Uso de un fondo de recursos públicos	Uso de un grupo de recursos dedicado
Trabajo de entrenamiento	No soportado	No soportado
Plantilla	No soportado	No soportado
Imagen del contenedor	No soportado	Se admite. .
OBS	No soportado	No soportado

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 más información sobre los parámetros de servicio en tiempo real, consulte **Implementación como servicio en tiempo real**.

   – Para obtener más información sobre los parámetros del servicio por lotes, consulte **Implementación 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 más información sobre los parámetros de servicio en tiempo real, consulte **Implementación como servicio en tiempo real**.

   – Para obtener más información sobre los parámetros del servicio por lotes, consulte **Implementación como servicio por lotes**.

6 Iniciar o detener un servicio

Iniciar un servicio

Puede iniciar los servicios en los estados **Successful**, **Abnormal**, o **Stopped**. Los servicios en el estado **Deploying** no se pueden iniciar. Un servicio se factura cuando se inicia y en el 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 un servicio. (Para un servicio en tiempo real, elija **More** > **Start** 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 **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.

7 Eliminación de un servicio

Si un servicio ya no está en uso, puede eliminarlo 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.

- Para un servicio en tiempo real, elija **More** > **Delete** en la columna **Operation** para eliminar los servicios
- Para un servicio por lotes,

NOTA

No se puede recuperar un servicio eliminado. Tenga cuidado cuando realice esta acción.

8 Monitoreo

8.1 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 utilizar Cloud Eye para monitorear automáticamente los servicios en tiempo real y modelos de ModelArts en tiempo real y gestionar alarmas y notificaciones, para que pueda realizar un seguimiento de las métricas de rendimiento de ModelArts y modelos.

Espacio de nombres

SYS.ModelArts

Métricas de monitoreo

Tabla 8-1 Métricas de ModelArts

ID de métrica	Nombre de la métrica	Descripción	Rango de valores	Entidad supervisada	Intervalo de monitoreo
cpu_usage	CPU Usage	Uso de CPU de ModelArts Unidad: %	≥ 0%	Modelos de ModelArts	1 minuto
mem_usage	Memory Usage	Uso de memoria de ModelArts Unidad: %	≥ 0%	Modelos de ModelArts	1 minuto
gpu_util	GPU Usage	Uso de GPU de ModelArts Unidad: %	≥ 0%	Modelos de ModelArts	1 minuto

ID de métrica	Nombre de la métrica	Descripción	Rango de valores	Entidad supervisada	Intervalo de monitoreo
successfully_called_times	Number of Successful Calls	Veces que ModelArts ha sido llamado con éxito Unidad: Tiempos/min	≥Vez/min	Modelos de ModelArts ModelArts servicios en tiempo real	1 minuto
failed_called_times	Number of Failed Calls	Veces que ModelArts no fue llamado Unidad: Tiempos/min	≥Vez/min	Modelos de ModelArts ModelArts servicios en tiempo real	1 minuto
total_called_times	API Calls	Veces que se llama ModelArts Unidad: Tiempos/min	≥Vez/min	Modelos de ModelArts ModelArts servicios en tiempo real	1 minuto

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 8-2 Descripción de la dimensión

Clave	Valor
service_id	Real-time service ID
model_id	Model ID

8.2 Configuración de reglas de alarmas

Caso

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.

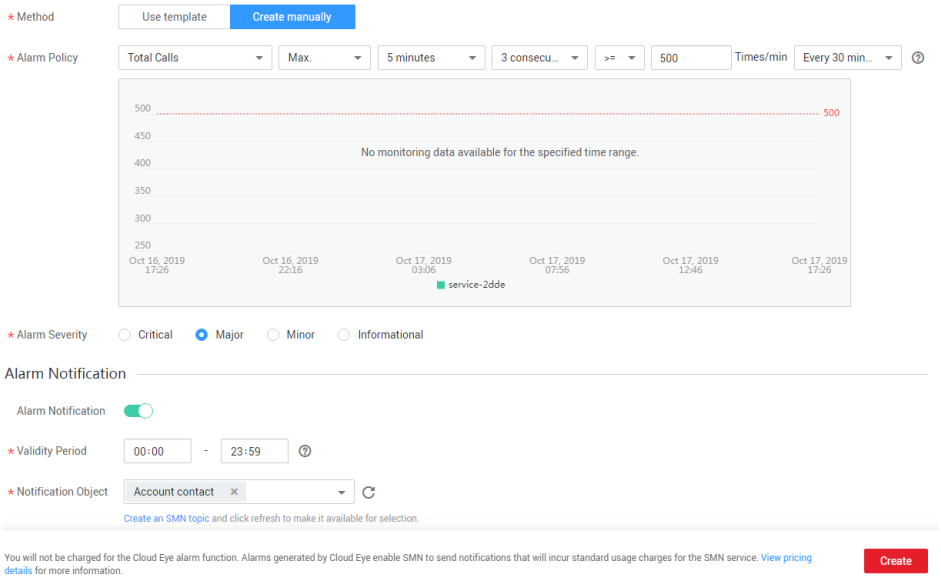
Prerrequisitos

Ha creado un servicio ModelArts en tiempo real.

Procedimiento

1. Inicie sesión en la consola de gestión.
2. Haga clic en **Service List**. En **Management & Deployment**, haga clic en **Cloud Eye**.
3. En la página Cloud Eye, haga clic en **Custom Monitoring**. A continuación, habilite la supervisión de ModelArts según se le indique.
4. En el panel de navegación izquierdo, seleccione **Cloud Service Monitoring > ModelArts**.
5. 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**.
6. 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.

Figura 8-1 Creación de una regla de alarma



7. Una vez completada la configuración, haga clic en **Create**. Cuando se genera una alarma que cumple con la regla, el sistema envía automáticamente una notificación.

8.3 Consulta de métricas de monitoreo

Caso

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.

Prerrequisitos

- El servicio en tiempo real ModelArts se está ejecutando correctamente.
- Las reglas de alarma se han configurado en la página Cloud Eye. Para más detalles, consulte **Configuración de reglas de alarmas**.
- El servicio en tiempo real ha estado funcionando correctamente durante al menos 10 minutos.
- Los datos y gráficos de monitorización están disponibles para un nuevo servicio en tiempo real después de que el servicio funcione 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, consulte **Configuración de reglas de alarmas**.

Procedimiento

1. Inicie sesión en la consola de gestión.
2. Haga clic en **Service List**. En **Management & Deployment**, haga clic en **Cloud Eye**.
3. En el panel de navegación izquierdo, seleccione **Cloud Service Monitoring > ModelArts**.
4. Ver gráficos de monitorización.
   - Consulta de gráficos de supervisión del servicio en tiempo real: Haga clic en **View Graph** en la columna **Operation**.
   - Consulta de gráficos de supervisión de las cargas de modelo: Haga clic en junto al servicio en tiempo real de destino y seleccione **View Graph** en la lista desplegable para cargas de modelo en la columna **Operation**.
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.

9 Especificaciones de Inferencia

9.1 Especificaciones del paquete de modelo

9.1.1 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.

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, consulte **Ejemplo de paquete de modelo**.
- **The model configuration file**: debe existir y su nombre es **config.json**. Solo existe un archivo de configuración de modelo. Para obtener más información sobre cómo compilar el archivo de configuración del modelo, consulte **Especificaciones para compilar el archivo de configuración del modelo**.
- **Model inference code file**: Es opcional. Si este archivo es necesario, el nombre del archivo se fija en **customize_service.py**. Debe haber uno y solo uno de esos archivos. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación 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 información, consulte **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 haciendo referencia a **Muestras de**

ModelArts. ModelArts también proporciona ejemplos de scripts personalizados de motores de IA comunes. Para obtener más información, consulte **Ejemplos de scripts personalizados**.

Ejemplo de paquete de modelo

- Structure of the TensorFlow-based model package

  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 (Optional) 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 MXNet

  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
|   |   ├── 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
|   |   ├──config.json (Mandatory) Model configuration file. The file name is
fixed to config.json. Only one model configuration file is supported.
|   |   ├──customize_service.py (Optional) 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 imágenes

  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
|   |   ├──config.json (Mandatory) Model configuration file (the address of
the SWR image must be configured). The file name is fixed to config.json.
Only one model configuration file is supported.
```

- Estructura del paquete de modelo basado en PySpark

  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
|   │  ├── spark_model (Mandatory) Model directory, which contains the model
content saved by PySpark
|   │  ├──config.json (Mandatory) Model configuration file. The file name is
fixed to config.json. Only one model configuration file is supported.
|   │  ├──customize_service.py (Optional) 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.
```

- Estructura del paquete modelo basado en Caffe

  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
|   |   |── 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
|   |   |── config.json (Mandatory) Model configuration file. The file name
is fixed to config.json. Only one model configuration file is supported.
|   |   |── customize_service.py  (Optional) 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 XGBoost

  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
|   |   |── *.m (Mandatory): Model file whose extension name is .m
|   |   |── config.json (Mandatory) Model configuration file. The file name
is fixed to config.json. Only one model configuration file is supported.
|   |   |── customize_service.py  (Optional) 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 Scikit_Learn

  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
|   |   |── *.m (Mandatory): Model file whose extension name is .m
|   |   |── config.json (Mandatory) Model configuration file. The file name
is fixed to config.json. Only one model configuration file is supported.
|   |   |── customize_service.py  (Optional) 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.
```

9.1.2 Especificaciones para compilar el archivo de configuración del modelo

Un desarrollador de modelos necesita compilar un archivo de configuración al publicar un modelo. El archivo de configuración del modelo describe el uso del modelo, el marco informático, 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 9-1** describe los parámetros.

Tabla 9-1 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 para imágenes personalizadas.

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 framework de uso frecuente, seleccione el motor y el entorno de tiempo de ejecución. Para obtener más información sobre los entornos de ejecución admitidos, consulte **Motores de IA compatibles para la inferencia de ModelArts**. Si su modelo necesita ejecutarse en una CPU o GPU especificada, seleccione el tiempo de ejecución basado en la información del sufijo. Si el tiempo de ejecución no contiene la información de CPU o GPU, lea la descripción de cada tiempo de ejecución en *Motores de IA compatibles para inferencia de ModelArts*.
swr_location	No	String	Dirección de imagen de SWR. ● Si importa un metamodelo de imagen personalizado desde una imagen contenedora, no es necesario establecer swr_location. ● Si importa un metamodelo de imagen personalizado de OBS (no recomendado) y establece model_type en Image, debe establecer swr_location. swr_location indica la dirección de la imagen Docker en SWR, indicando que la imagen Docker en SWR se usa para publicar el modelo.
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 9-2**. El resultado se muestra en el área de precisión del modelo de la página de detalles de la aplicación de IA.

Parámetro	Obligatorio	Tipo de datos	Descripción
apis	No	api array	Formato de las solicitudes recibidas y devueltas por un modelo. El valor es dato de estructura. Es la matriz API RESTful proporcionada por un modelo. Para obtener más información sobre la estructura de datos de la API, consulte **Tabla 9-3**. ● Cuando model_type se establece en Image, es decir, en el escenario de modelo de una imagen personalizada, las API con diferentes rutas se pueden declarar en **apis** basándose en la ruta de solicitud expuesta por la imagen. ● 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 /.
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 admite el modo de instalación pip. **Tabla 9-6** 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.Packages on the top are installed first. The wheel package on premises can be used for installation. (The wheel package must be stored in the same directory as the model file). For details, see **How Do I Edit the Installation Package Dependency Parameters in a Model Configuration File When Importing a Model?**
health	No	**health** data structure	Configuración de una interfaz de estado de imagen. Este parámetro solo es obligatorio cuando **model_type** se establece en **Image**. Si los servicios no se pueden interrumpir durante la actualización continua, se debe proporcionar un puerto de comprobación de estado para que ModelArts llame. Para obtener más información sobre la estructura de datos de salud, consulte **Tabla 9-8**.

Tabla 9-2 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 retirada. El valor se redondea a 17 decimales.
precision	No	Number	Precisión. El valor se redondea a 17 decimales.
accuracy	No	Number	Exactitud. El valor se redondea a 17 decimales.

Tabla 9-3 Matriz **api**

Parámetro	Obligatorio	Tipo de datos	Descripción
protocol	No	String	Solicite el protocolo. Establezca el valor del parámetro en http o https en función de su imagen personalizada. Si utiliza un metamodelo importado de OBS, el protocolo predeterminado es https. Para obtener más información sobre otros parámetros, consulte **Ejemplo del archivo de configuración del modelo de detección de objetos**.
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 obtener más información sobre la estructura **request**, consulte **Tabla 9-4**.
response	No	Object	Cuerpo de respuesta. Para obtener más información sobre la estructura **response**, consulte **Tabla 9-5**.

Tabla 9-4 Descripción de **request**

Parámetro	Obligatorio	Tipo de datos	Descripción
Content-type	Sí para los servicios en tiempo real No para los 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**: envía datos JSON. ● **multipart/form-data**: carga un archivo. **NOTA** Para los modelos de aprendizaje automático, solo se admite **application/json**.
data	Sí para los servicios en tiempo real No para los 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 9-5 Descripción de **response**

Parámetro	Obligatorio	Tipo de datos	Descripción
Content-type	Sí para los servicios en tiempo real No para los 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	Sí para los servicios en tiempo real No para los 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 9-6 Matriz de **dependency**

Parámetro	Obligatorio	Tipo de datos	Descripción
installer	Sí	String	Método de instalación. Solo se admite pip.

Parámetro	Obligatorio	Tipo de datos	Descripción
packages	Sí	package array	Colección de paquetes de dependencia. Para obtener más información sobre la matriz de estructura de paquetes, consulte **Tabla 9-7**.

Tabla 9-7 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 del número de versión, 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 preferentemente EXACT. 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 9-8 Descripción de la estructura de datos **health**

Parámetro	Obligatorio	Tipo	Descripción
url	Sí	String	Solicitud de URL de la interfaz de comprobación de estado

Parámetro	Obligatorio	Tipo	Descripción
protocol	No	String	Solicitud de protocolo de la interfaz de comprobación de estado. Solo se soporta el protocolo HTTP.
initial_delay_seconds	No	String	Después de iniciar una instancia, se inicia una comprobación de estado después de segundos configurados en **initial_delay_seconds**.
timeout_seconds	No	String	Tiempo de espera de la comprobación de estado

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

  Clave: images

  Valor: 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": [{
        "protocol": "http",
        "url": "/",
        "method": "post",
```



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

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

```
{
    "model_type": "TensorFlow",
    "model_algorithm": "predict_analysis",
    "metrics": {
```



```
        "f1": 0.345294,
        "accuracy": 0.462963,
        "precision": 0.338977,
        "recall": 0.351852
    },
    "apis": [
        {
            "protocol": "http",
            "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": "multipart/form-data",
                "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 los 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": [{
        "protocol": "http",
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
```



```
                    "images": {
                        "type": "file"
                    }
                }
            }
        },
        "response": {
            "Content-type": "multipart/form-data",
            "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

```
{
    "data": {
        "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

```
{
    "data": {
```



```
        "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": [
    {
      "protocol": "http",
      "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": "applicaton/json",
        "data": {
          "type": "object",
          "properties": {
            "resp_data": {
              "type": "array",
              "items": [
                {
                  "type": "object",
                  "properties": {
                    "predict_result": {
                      "type": "number"
                    }
                  }
                }
              ]
            }
          }
        }
      }
    }
  ]
}
```


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": [{
             "procotol": "http",
             "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": 0.0023493892851938493,
         "accuracy": 0.00746268656716417
     },
    "dependencies": [{
        "installer": "pip",
        "packages": [{
                "restraint": "EXACT",
                "package_version": "1.16.4",
                "package_name": "numpy"
            }
        ]
    }]
}
```

9.1.3 Especificaciones para la codificación de inferencia de modelo

En esta sección se describe cómo compilar código de inferencia de modelo de ModelArts. Para obtener detalles sobre los ejemplos de código de script personalizados (incluidos ejemplos de código de inferencia) de motores de IA comunes, consulte **Ejemplos de scripts personalizados**. A continuación también se proporciona un ejemplo de código de inferencia

para el motor TensorFlow y un ejemplo de personalización de la lógica de inferencia en un script de inferencia.

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 9-9**.

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

Tipo de modelo	Clase de padres	Instrucción de importación
TensorFlow	TfServingBaseService	from model_service.tfserving_model_service import TfServingBaseService
MXNet	MXNetBaseService	from mms.model_service.mxnet_model_service import MXNetBaseService
PyTorch	PTServingBaseService	from model_service.pytorch_model_service import PTServingBaseService
Caffe	CaffeBaseService	from model_service.caffe_model_service import CaffeBaseService

2. Se pueden reescribir los siguientes métodos:

Tabla 9-10 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

Método	Descripción
_inference(self, data)	Método de solicitud de inferencia. No se recomienda reescribir el método porque una vez que se reescribe el método, 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 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 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, Scikit_Learn o PySpark, self.model_path indica la ruta del archivo 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 TensorFlow consulte **TensorFlow** y **TensorFlow 2.1**.Para obtener más información sobre el código de inferencia de otros motores, consulte **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 32x32 se lee de la biblioteca Pillow y se redimensiona a 1x784 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 **Scikit Learn**.

```
# 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

Primero, defina un paquete de dependencias en el archivo de configuración. Para más detalles, consulte **Ejemplo de un archivo de configuración de modelo que utiliza un paquete de dependencia personalizado**. A continuación, utilice el siguiente ejemplo de código para implementar la carga y la inferencia del modelo en formato **saved_model**.

```
# -*- 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

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()
```

9.2 Plantillas de modelo

9.2.1 Introducción a las plantillas de modelo

Debido a que las configuraciones de modelos con las mismas funciones son similares, el ModelArts integra las configuraciones de tales modelos en una plantilla común. Mediante el uso de esta plantilla, puede importar modelos y crear aplicaciones de IA de manera fácil y rápida sin compilar 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 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 los modos de entrada y salida fijos. Por ejemplo, la **TensorFlow-based image classification template** utiliza el modo de procesamiento de imágenes integrado.
- Las plantillas generales integran un motor de IA específico y un entorno de ejecución y utilizan el modo de entrada y salida indefinidos. Es necesario seleccionar un modo de

entrada y salida basado en la función de modelo o escenario de aplicación para sobrescribir el modo indefinido. Por ejemplo, un modelo de clasificación de imágenes requiere el modo de procesamiento de imágenes incorporado, y un modelo de detección de objetos requiere el modo de detección de objetos incorporado.

NOTA

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

Uso de una plantilla

Sube el paquete de modelo a OBS antes de usar la plantilla. Almacene los archivos de modelo en el directorio de **model**. Al crear una aplicación de IA con esta plantilla, debe seleccionar el directorio del **model**. Para más detalles, consulte **Importación de un metamodelo desde OBS a través de una plantilla**.

Plantillas compatibles

- **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-CPU-py36**
- **Plantilla Arm-Ascend**

Modos de entrada y salida compatibles

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

9.2.2 Plantillas

9.2.2.1 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 de **SavedModel**. Esta plantilla utiliza el modo de procesamiento de imágenes integrado de ModelArts. Para obtener más información sobre el modo de procesamiento de imágenes, consulte **Modo de procesamiento de imágenes incorporado**. Asegúrese de que el modelo

puede procesar imágenes cuya **key** es **images**, ya que necesita introducir una imagen cuya **key** es **images** en el modelo para inferencia. Cuando utilice la plantilla para importar un modelo, seleccione el directorio de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en TensorFlow almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **Ejemplo de paquete de modelo**.

Modo de entrada y salida

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

Especificaciones de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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   (Optional) 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.
```


9.2.2.2 Plantilla general de TensorFlow-py27

Introducción

Motor AI: 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en TensorFlow almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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   (Optional) 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.
```

9.2.2.3 Plantilla general de TensorFlow-py36

Introducción

Motor AI: 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en TensorFlow almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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   (Optional) 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.
```

9.2.2.4 Plantilla general MXNet-py27

Introducción

Motor AI: 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en MXNet almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **Ejemplo de paquete de modelo**.

Modo de entrada y salida

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

Especificaciones de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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 the 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      (Optional) 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.
```

9.2.2.5 Plantilla general MXNet-py36

Introducción

Motor AI: MXNet 1.2.1; Entorno: Python 3.6; 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en MXNet almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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      (Optional) 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.
```

9.2.2.6 Plantilla general PyTorch-py27

Introducción

Motor de IA: PyTorch 1.0; Entorno: Python 2.7; 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en PyTorch almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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    (Optional) 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.
```

9.2.2.7 Plantilla general PyTorch-py36

Introducción

Motor de IA: PyTorch 1.0; Entorno: Python 3.6; 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en PyTorch almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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    (Optional) 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.
```

9.2.2.8 Plantilla general Caffe-CPU-py27

Introducción

Motor AI: CPU-based Caffe 1.0; 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en Caffe almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **Ejemplo de paquete de modelo**.

Modo de entrada y salida

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

Especificaciones de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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 the 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    (Optional) 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.
```

9.2.2.9 Plantilla general Caffe-GPU-py27

Introducción

Motor AI: GPU-based Caffe 1.0; Entorno: Python 2.7; 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en Caffe almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.

- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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    (Optional) 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.
```

9.2.2.10 Plantilla general Caffe-CPU-py36

Introducción

Motor AI: CPU-based Caffe 1.0; Entorno: Python 3.6; 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en Caffe almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de

inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.

- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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    (Optional) 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.
```

9.2.2.11 Plantilla general Caffe-CPU-py36

Introducción

Motor AI: Caffe 1.0 basado en GPU; Entorno: Python 3.6; 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 de **model** que contiene los archivos de modelo.

Entrada de plantilla

La entrada de plantilla es el paquete de modelo basado en Caffe almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.

- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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 del **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    (Optional) 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.
```

9.2.2.12 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 plantilla, seleccione el directorio **model** que contiene los archivos de plantilla.

Entrada de plantilla

La entrada de plantilla es el paquete de plantilla basado en OM almacenado en OBS. Asegúrese de que el directorio OBS que utiliza y el ModelArts están en la misma región. Para obtener más información acerca de los requisitos del paquete modelo, consulte **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 de paquete de modelo

- El paquete de modelo debe almacenarse en la carpeta OBS denominada **model**. Los archivos de modelo y el archivo de código de inferencia de modelo se almacenan en la carpeta de **model**.
- El archivo de código de inferencia de modelo es opcional. Si el archivo existe, el nombre del archivo debe ser **customize_service.py**. Solo puede existir un archivo de código de inferencia en la carpeta **model**. Para obtener más información sobre cómo compilar código de inferencia de modelo, consulte **Especificaciones para la codificación de inferencia de modelo**.
- La estructura del paquete de modelo importado con 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  //(Optional) 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   (Optional) 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.
```

9.2.3 Modos de entrada y salida

9.2.3.1 Modo de detección de objetos incorporado

Entrada

Este es un modo de entrada y salida incorporado 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 de /, 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 su modelo puede procesar los datos de entrada cuya **key** son **images**.

Salida

El resultado de la inferencia se devuelve en formato JSON. Para obtener más información sobre los campos, consulte **Tabla 9-11**.

Tabla 9-11 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 detección

El **JSON Schema** del resultado de la 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"
        }
    }
}
```

Muestra de solicitud

En este modo, introduzca una imagen a procesar en la solicitud de inferencia. El resultado de la inferencia se devuelve en formato JSON. A continuación citamos varios ejemplos:

- Realizar la predicción en la consola

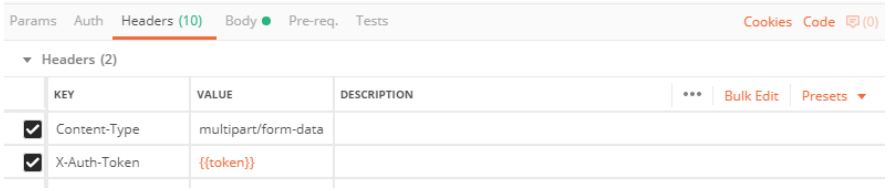
  En la página de pestaña **Prediction** de la página de detalles del servicio, cargue una imagen y haga clic en **Predict** para obtener el resultado de la predicción.

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

  Después de implementar 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.

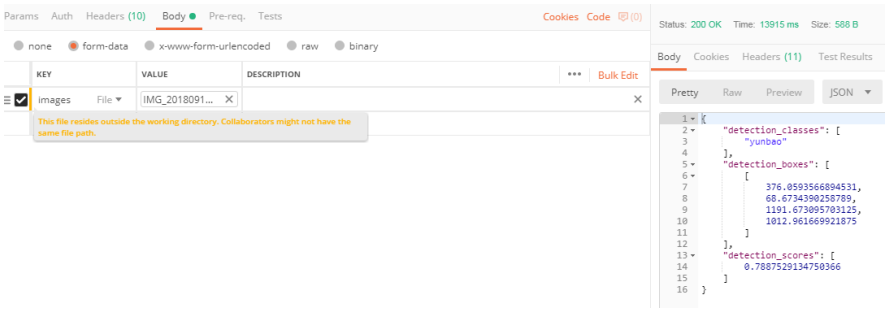
  – En la página de pestaña **Headers**, establezca **Content-Type** en **multipart/form-data** y **X-Auth-Token** en el token real obtenido.

Figura 9-1 Configuración del encabezado de solicitud



– En la página de la pestaña **Body**, establezca el cuerpo de la solicitud. Establezca **key** en **images**, seleccione **File**, seleccione la imagen que se va a procesar y haga clic en **send** para enviar su solicitud de predicción.

Figura 9-2 Configuración del cuerpo de la solicitud



9.2.3.2 Modo de procesamiento de imágenes incorporado

Entrada

El modo de entrada y salida de procesamiento de imágenes incorporado se puede aplicar a modelos tales como clasificación de imágenes, detección de objetos y segmentación semántica de imágenes. La ruta de solicitud de predicción es de /, 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 su modelo puede procesar los datos de entrada cuya **key** son **images**.

Salida

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

Muestra de solicitud

En este modo, introduzca una imagen a procesar en la solicitud de inferencia. La respuesta en formato JSON varía según el modelo. A continuación citamos varios ejemplos:

- Realizar la predicción en la consola
- Uso de Postman para llamar a una API RESTful para la predicción

  Después de implementar 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 de la pestaña **Body**, establezca el cuerpo de la solicitud. Establezca **key** en **images**, seleccione **File**, seleccione la imagen que se va a procesar y haga clic en **send** para enviar su solicitud de predicción.

Figura 9-3 Llamar a una API RESTful

9.2.3.3 Modo de análisis predictivo incorporado

Entrada

Este es un modo de entrada y salida incorporado para el análisis predictivo. Los modelos que utilizan 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 JSON. Para obtener más información sobre los campos JSON, consulte **Tabla 9-12**. Antes de seleccionar este modo, asegúrese de que su modelo puede procesar los datos de entrada en formato de **JSON Schema**. Para obtener más información sobre el formato de **JSON Schema**, consulte **la guía oficial**.

Tabla 9-12 Descripción del campo JSON

Campo	Tipo	Descripción
data	Data structure	Datos de inferencia. Para más detalles, consulte **Tabla 9-13**.

Tabla 9-13 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 de modelo personalizado debe ser capaz de procesar correctamente los datos introducidos 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 JSON. Para obtener más información sobre los campos JSON, consulte **Tabla 9-14**.

Tabla 9-14 Descripción del campo JSON

Campo	Tipo	Descripción
data	Data structure	Datos de inferencia. Para más detalles, consulte **Tabla 9-15**.

Tabla 9-15 Descripción de **Data**

Campo	Tipo	Descripción
resp_data	RespData array	Lista de resultados de predicción

Al igual que el **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 de modelo personalizado debe ser capaz de generar datos correctamente 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": {}
                    }]
                }
            }
        }
    }
}
```

Muestra de solicitud

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

- Realizar la predicción en la consola

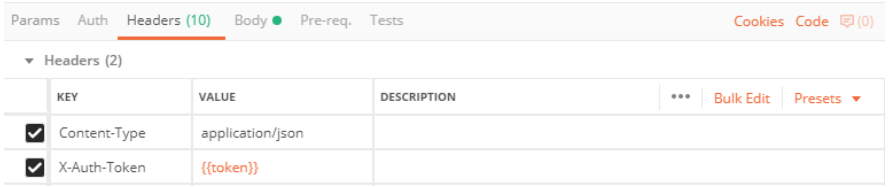
En la página de pestaña **Prediction** de la página de detalles del servicio, introduzca el código de inferencia y haga clic en **Predict** para obtener el resultado de la predicción.

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

  Después de implementar 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 de pestaña **Headers**, establezca **Content-Type** en **application/json** y **X-Auth-Token** en el token real obtenido.

  Figura 9-4 Establecer el encabezado de solicitud para la predicción

  – En la página de la pestaña On the **Body** tab page, edite los datos que se van a predecir y haga clic en **send** para enviar su solicitud de predicción.

9.2.3.4 Modo indefinido

Descripción

El modo indefinido no define el modo de entrada y salida. El modo de entrada y salida viene determinado por el modelo. Seleccione este modo sólo 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 implementar como servicios por lotes. Además, es posible que la página de predicción de servicio no se muestre correctamente.

Entrada

No hay límite.

Salida

No hay límite.

Muestra de solicitud

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

9.3 Ejemplos de scripts personalizados

9.3.1 TensorFlow

TensorFlow tiene dos tipos de API: Keras y tf. Keras y tf usan código diferente para entrenar y guardar modelos, pero el mismo código para inferencia.

Entrenamiento de un modelo (Keras API)

```
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 (Keras API)

```
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 (tf API)

```
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!')
```

Guardar un modelo (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 (Keras y tf APIs)

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

```
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
```

9.3.2 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 9-9**.

```
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())
        }
```

9.3.3 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 secundaria. Esta clase de modelo secundaria hereda las propiedades de su clase de modelo principal. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 9-9**.

```
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
```

9.3.4 Caffe

Entrenamiento y guardado de un modelo

archivo **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 **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
```

Entrena el modelo.

```
./build/tools/caffe train --solver=examples/mnist/lenet_solver.prototxt
```

El archivo **caffemodel** se genera después del entrenamiento del modelo. Vuelva a escribir el archivo **lenet_train_test.prototxt** en el archivo **lenet_deploy.prototxt** usado para la implementación 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 secundaria. Esta clase de modelo secundaria hereda las propiedades de su clase de modelo principal. Para obtener más información sobre las instrucciones de importación de diferentes tipos de clases de modelo padre, consulte **Tabla 9-9**.

```
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
```

9.3.5 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**, descomprimelo 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 compilar el archivo de configuración del modelo**. Para obtener más información sobre el código de inferencia, consulte **Código de inferencia**.

Código de inferencia

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

```
# 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
```

9.3.6 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 compilar el archivo de configuración del modelo**. Para obtener más información sobre el código de inferencia, consulte **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 9-9**.

```
# 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
```

9.3.7 Scikit Learn

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**, descomprimelo 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 contenidos durante la publicación. Para obtener más información sobre el método de definición, consulte **Especificaciones del paquete de 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 9-9**.

```
# 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
```

