



códigofacilito

Módulo 5. Implementación y consumo de modelos con Azure Machine Learning

Implementación de un modelo en un punto de conexión por lotes

CINTHYA CABANZO





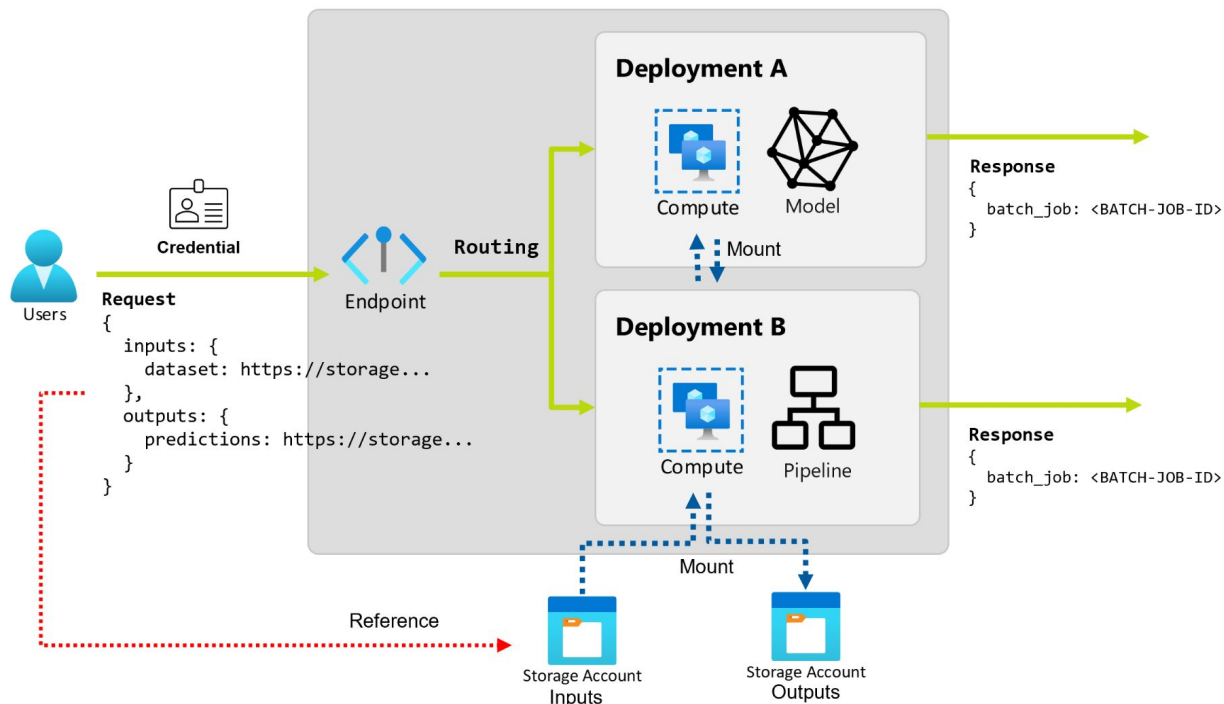
>_ Agenda

- Introducción
- Descripción y creación de puntos de conexión por lotes
- Implementación del modelo de MLflow en un punto de conexión por lotes
- Implementación de un modelo personalizado en un punto de conexión por lotes
- Invocación y solución de problemas de puntos de conexión por lotes
- Ejercicio: Implementación de un modelo de MLflow en un punto de conexión por lotes





Punto de Conexión por Lotes (Batch endpoints)



En Azure Machine Learning, puede implementar soluciones de **inferencia por lotes** mediante la **implementación** de un **modelo** en un **punto de conexión por lotes**.



¿Cuándo se usa?

Azure Machine Learning te permite implementar puntos conexión por lotes y despliegues para realizar **inferencias asincrónicas** y de **larga duración** con modelos y pipelines de aprendizaje automático.

En lugar de manejar cada solicitud de inferencia individualmente y en tiempo real, un punto de conexión por lotes recibe un **conjunto de datos**, procesa este conjunto en **una operación**, y luego devuelve los resultados.

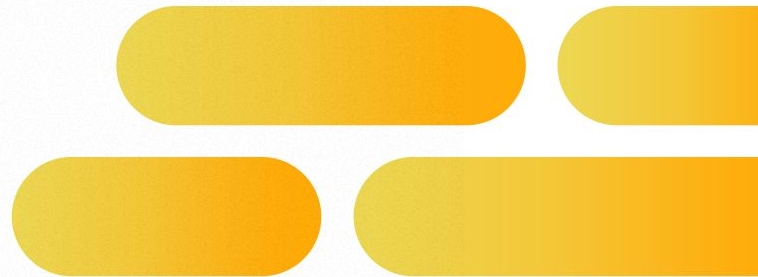
Cuando entrenas un modelo o pipeline de aprendizaje automático, necesitas **desplegarlo** para que otros puedan usarlo con nuevos datos de entrada para **generar predicciones**. Este proceso de generar predicciones con el modelo o pipeline se llama **inferencia**.





¿Cuándo se usa?

- Tienes modelos o pipelines **costosos** que requieren **más tiempo** para ejecutarse.
- Quieres operacionalizar pipelines de aprendizaje automático y **reutilizar componentes**.
- Necesitas realizar inferencia sobre **grandes cantidades de datos, distribuidos** en múltiples **archivos**.
- **No** tienes requisitos de **baja latencia**.
- Las entradas de tu modelo están almacenadas en una Cuenta de Almacenamiento o en un activo de datos de Azure Machine Learning.
- Quieres aprovechar la **paralelización**.





Ventajas




- **Optimización de Costos:** Al ejecutar múltiples tareas juntas, se pueden aprovechar mejor los recursos computacionales, reduciendo costos.
- **Mejora de Rendimiento:** La ejecución por lotes permite una mejor gestión de la carga de trabajo, distribuyendo tareas en múltiples nodos para acelerar el procesamiento.
- **Facilidad de Manejo:** Es más sencillo manejar y monitorear un conjunto de tareas agrupadas en un lote que individualmente, lo que simplifica la gestión y supervisión.
- **Resiliencia:** La capacidad de volver a ejecutar lotes individuales en caso de fallo mejora la robustez y fiabilidad del sistema.

Desventajas

- **Latencia:** Las tareas por lotes pueden introducir una latencia adicional, ya que los trabajos individuales deben esperar a que se complete el lote entero antes de que los resultados estén disponibles.
- **Complejidad de Configuración:** Configurar y manejar tareas por lotes puede ser más complejo, requiriendo una planificación cuidadosa y una gestión adecuada de los recursos.
- **Limitación en Tareas en Tiempo Real:** Las tareas por lotes no son adecuadas para aplicaciones que requieren procesamiento en tiempo real, ya que la latencia puede ser inaceptable.
- **Dependencias:** Las tareas dentro de un lote pueden tener dependencias entre sí, lo que puede complicar la gestión y la resolución de fallos.

 Aspecto	Ejecución por Lotes	Ejecución en Línea (Tiempo Real)
Latencia 	Alta latencia. Los resultados están disponibles después de que se completa el procesamiento del lote.	Baja latencia. Los resultados están disponibles casi instantáneamente.
Eficiencia de Recursos	Alta eficiencia. Optimiza el uso de recursos agrupando tareas.	Menor eficiencia. Requiere recursos disponibles en todo momento para manejar las solicitudes en tiempo real.
Escalabilidad	Muy escalable. Puede manejar grandes volúmenes de datos distribuyendo la carga de trabajo.	Escalable, pero puede ser más costoso y complejo debido a la necesidad de recursos siempre disponibles.
Robustez	Alta robustez. Permite implementar estrategias de recuperación ante fallos para lotes completos.	Menor robustez. La recuperación ante fallos puede ser más complicada y crítica debido a la naturaleza en tiempo real.



 Aspecto	Ejecución por Lotes	Ejecución en Línea (Tiempo Real)
Manejo de Fallos 	Fácil recuperación. Los lotes pueden ser reejecutados en caso de fallos.	Recuperación más compleja. Los fallos en tiempo real deben ser manejados inmediatamente.
Configuración y Gestión	Complejo. Requiere planificación y gestión cuidadosa de los recursos y los trabajos por lotes.	Complejo. Requiere infraestructura y monitoreo constante para asegurar la disponibilidad y el rendimiento.
Aplicaciones Adecuadas	Adecuado para tareas que no necesitan resultados inmediatos, como procesamiento de datos históricos, generación de informes, y análisis de grandes volúmenes de datos.	Adecuado para aplicaciones que requieren respuestas inmediatas, como servicios web, aplicaciones interactivas, y sistemas de recomendación en tiempo real.
Costo	Generalmente más económico. Los recursos se utilizan de manera óptima en intervalos específicos.	Generalmente más costoso. Requiere recursos dedicados en todo momento para asegurar la capacidad de respuesta.
Dependencias 	Puede tener dependencias complejas entre tareas dentro del mismo lote.	Las tareas son generalmente independientes, procesando una solicitud a la vez.



Descripción y creación de puntos de conexión por lotes

Para obtener **predicciones por lotes**, puede implementar un modelo en un **punto de conexión (endpoint)**.

Un punto de conexión es un **punto de conexión HTTPS** al que puede llamar para desencadenar un trabajo de puntuación por lotes.

La ventaja de este punto de conexión es que puede desencadenar el trabajo de puntuación por lotes desde otro servicio, como **Azure Synapse Analytics** o **Azure Databricks**. Un punto de conexión por lotes permite integrar la puntuación por lotes con una canalización de ingesta y transformación de datos existente.

Cada vez que se invoca el punto de conexión, se envía un trabajo de puntuación por lotes al área de trabajo de Azure Machine Learning. Normalmente, el trabajo usa un **clúster de proceso** para **puntuar varias entradas**.

Los resultados se pueden almacenar en un **almacén de datos**, conectado al área de trabajo de Azure Machine Learning.





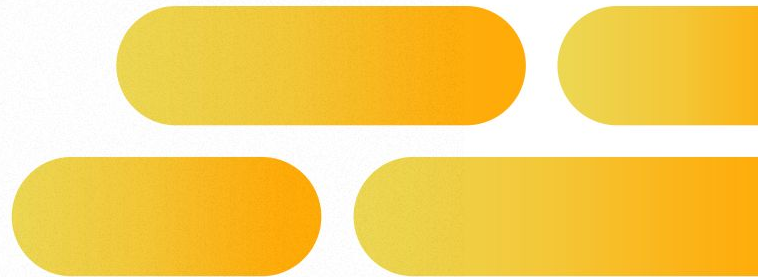
Creación de un punto de conexión por lotes

BatchEndpoint

Python

```
# create a batch endpoint
endpoint = BatchEndpoint(
    name="endpoint-example",
    description="A batch endpoint",
)

ml_client.batch_endpoints.begin_create_or_update(endpoint)
```



```
# Importar la clase BatchEndpoint desde el paquete de Azure Machine Learning.
# Esto es necesario para definir un nuevo punto de conexión por lotes.
from azure.ai.ml import BatchEndpoint

# Crear una instancia de BatchEndpoint.
# Aquí estamos definiendo un nuevo punto de conexión por lotes.
# Un punto de conexión por lotes es una interfaz a través de la cual se pueden enviar
# grandes cantidades de datos para su procesamiento asincrónico.

# name: Especifica el nombre del punto de conexión. Este nombre debe ser único dentro de tu
# espacio de trabajo de Azure Machine Learning. En este caso, lo llamamos "endpoint-example".
# description: Proporciona una breve descripción del punto de conexión. Aquí, simplemente
# indicamos que es un punto de conexión por lotes.
endpoint = BatchEndpoint(
    name="endpoint-example",
    description="A batch endpoint",
)

# Ahora que hemos definido nuestro punto de conexión por lotes, necesitamos crearlo o actualizarlo
# en Azure Machine Learning. Para esto, utilizamos un cliente específico para gestionar los recursos
# de aprendizaje automático en Azure (ml_client).

# La función begin_create_or_update es asíncrona, lo que significa que iniciará el proceso de
# creación o actualización del punto de conexión y luego devolverá el control al programa
# inmediatamente. El procesamiento real se llevará a cabo en segundo plano.
```


`ml_client.batch_endpoints` es el componente que gestiona los puntos de conexión por lotes.
`begin_create_or_update(endpoint)` es la llamada que crea o actualiza el punto de conexión definido anteriormente.
En resumen, este comando asegura que el punto de conexión "endpoint-example" esté disponible en tu espacio de trabajo de Azure Machine Learning, listo para recibir trabajos de procesamiento por lotes.
`ml_client.batch_endpoints.begin_create_or_update(endpoint)`

A continuación, explicamos más en detalle cada parte del proceso:

1. ****Definición del Punto de Conexión por Lotes****:

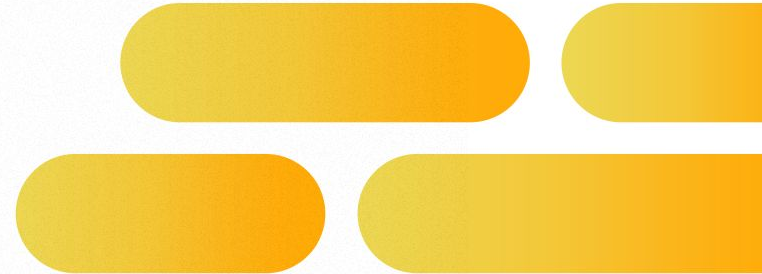
- ``BatchEndpoint(name="endpoint-example", description="A batch endpoint")``: Aquí creamos una definición del punto de conexión por lotes. Le damos un nombre y una descripción.
Este punto de conexión actuará como un receptor para las solicitudes de procesamiento por lotes.

2. ****Creación/Actualización del Punto de Conexión****:

- ``ml_client.batch_endpoints.begin_create_or_update(endpoint)``: Con esta línea, le decimos a Azure Machine Learning que queremos crear o actualizar el punto de conexión definido.
Este comando es asíncrono, lo que significa que el programa no esperará a que la operación termine para continuar ejecutándose. En lugar de eso, comenzará la operación en segundo plano.




```
# **Conceptos Clave**:  
# - **Batch Endpoint (Punto de Conexión por Lotes)**: Es una interfaz para enviar grandes cantidades  
# de datos a un modelo de aprendizaje automático para su procesamiento. A diferencia de los puntos  
# de conexión en tiempo real, los puntos de conexión por lotes procesan los datos de forma  
asíncrona,  
# lo que es útil para tareas que requieren mucho tiempo de procesamiento.  
# - **Asíncrono**: Significa que la operación se inicia y el control se devuelve inmediatamente al  
programa,  
# permitiendo que otras tareas se ejecuten mientras el procesamiento se realiza en segundo plano.  
# - **ml_client**: Es un cliente de Azure Machine Learning que maneja la comunicación con el  
servicio de Azure.  
# - **begin_create_or_update**: Es una función que inicia la creación o actualización de un recurso  
en Azure Machine Learning.  
  
# Este código es parte de un flujo de trabajo más grande para implementar y gestionar modelos de  
aprendizaje  
# automático en Azure, permitiendo el procesamiento eficiente y escalable de grandes volúmenes de  
datos.
```





Implementación de un modelo en un punto de conexión por lotes

Puede **implementar varios modelos en un punto de conexión por lotes**.

Siempre que llame al punto de conexión por lotes, que desencadena un trabajo de puntuación por lotes, se usará la **implementación predeterminada**, a menos que se especifique lo contrario.

The screenshot shows a web application interface for managing batch connections. On the left is a sidebar with navigation icons. The main content area is titled 'lote' and has two tabs: 'Detalles' (selected) and 'Trabajos'. Below the tabs is a toolbar with the following actions: 'Agregar implementación' (with a plus icon), 'Actualizar' (with a refresh icon), 'Actualizar implementación predeterminada' (with a pencil icon), 'Crear trabajo' (with a plus icon), and 'Eliminar' (with a trash icon). The 'Detalles' tab displays two sections: 'Atributos' and 'Resumen de la implementación'. The 'Atributos' section shows 'Identificador del servicio' as 'batch-11071032873754' and 'Descripción' as 'Un punto de conexión por lotes para clasificar la diabetes en pacientes'. The 'Resumen de la implementación' section shows 'Implementaciones' as 'classifier-diabetes-mflow' and 'Valor predeterminado' as 'Valor predeterminado'. The 'Resumen de la implementación' section is highlighted with a red border.

lote

Detalles Trabajos

+ Agregar implementación Actualizar Actualizar implementación predeterminada + Crear trabajo Eliminar

Atributos

Identificador del servicio
batch-11071032873754

Descripción
Un punto de conexión por lotes para clasificar la diabetes en pacientes

Resumen de la implementación

Implementaciones
classifier-diabetes-mflow Valor predeterminado



Uso de clústeres de proceso para implementaciones por lotes

AMLCompute

Si desea que el trabajo de puntuación por lotes procese los nuevos datos en lotes paralelos, debe aprovisionar un clúster de proceso con **más de una instancia máxima**.

Python

```
from azure.ai.ml.entities import AmlCompute

cpu_cluster = AmlCompute(
    name="aml-cluster",
    type="amlcompute",
    size="STANDARD_DS11_V2",
    min_instances=0,
    max_instances=4,
    idle_time_before_scale_down=120,
    tier="Dedicated",
)

cpu_cluster = ml_client.compute.begin_create_or_update(cpu_cluster)
```



```
# Importar la clase AmlCompute desde el paquete de Azure Machine Learning.  
# AmlCompute se utiliza para definir clústeres de cómputo en Azure Machine Learning.  
from azure.ai.ml.entities import AmlCompute
```

```
# Definir un clúster de cómputo llamado "aml-cluster".  
# AmlCompute es una clase que representa un clúster de máquinas virtuales en Azure  
# que puede ser utilizado para ejecutar experimentos y trabajos de aprendizaje automático.  
cpu_cluster = AmlCompute(  
    name="aml-cluster", # El nombre del clúster de cómputo.  
    type="amlcompute", # Especifica que el tipo de clúster es "amlcompute".  
    size="STANDARD_DS11_V2", # El tamaño de las máquinas virtuales en el clúster. "STANDARD_DS11_V2" es  
    un tipo de máquina virtual con recursos específicos de CPU y memoria.  
    min_instances=0, # El número mínimo de instancias en el clúster. Aquí se especifica 0, lo que  
    significa que el clúster puede escalar hasta no tener ninguna instancia en uso.  
    max_instances=4, # El número máximo de instancias en el clúster. Aquí se especifica 4, lo que  
    significa que el clúster puede escalar hasta 4 instancias.  
    idle_time_before_scale_down=120, # El tiempo en segundos que una instancia debe estar inactiva antes  
    de que se reduzca el escalado. Aquí se especifican 120 segundos.  
    tier="Dedicated", # Especifica el tipo de clúster. "Dedicated" significa que las máquinas virtuales  
    están dedicadas exclusivamente a tu cuenta y no se comparten con otros clientes.  
)
```

```
# Crear o actualizar el clúster de cómputo en Azure Machine Learning.  
# ml_client es una instancia del cliente de Azure Machine Learning que se utiliza para interactuar con los  
    recursos de Azure ML.  
# begin_create_or_update es una función asíncrona que inicia el proceso de creación o actualización del  
    clúster de cómputo.  
cpu_cluster = ml_client.compute.begin_create_or_update(cpu_cluster)
```

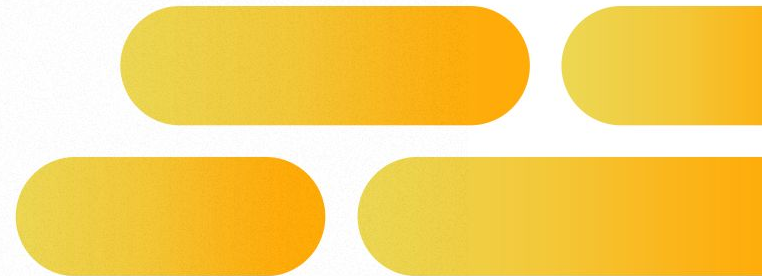



Implementación del modelo de MLflow en un punto de conexión por lotes

Una manera **sencilla** de implementar un modelo en un punto de conexión por **lotes** es usar un **modelo de MLflow**.

Azure Machine Learning generará **automáticamente** el **script de puntuación** y el **entorno** para los modelos de **MLflow**.

Para implementar un modelo de MLflow, debe haber **creado** un **punto de conexión**.





Registro de un modelo de MLflow

create_or_update

Para **evitar** la necesidad de un **entorno** y un **script de puntuación**, es necesario **registrar** un modelo de **MLflow** en el área de trabajo de Azure Machine Learning para poder implementarlo en un punto de conexión por lotes

Python

```
from azure.ai.ml.entities import Model
from azure.ai.ml.constants import AssetTypes

model_name = 'mlflow-model'
model = ml_client.models.create_or_update(
    Model(name=model_name, path='./model', type=AssetTypes.MLFLOW_MODEL)
)
```



```
# Importamos la clase Model del módulo azure.ai.ml.entities.
from azure.ai.ml.entities import Model

# Importamos AssetTypes del módulo azure.ai.ml.constants.
from azure.ai.ml.constants import AssetTypes

# Definimos el nombre del modelo que queremos registrar en Azure ML.
model_name = 'mlflow-model'

# Creamos una instancia del modelo utilizando la clase Model.
# Para ello, proporcionamos:
# - name: el nombre del modelo.
# - path: la ruta donde se encuentra el modelo que queremos registrar.
# - type: el tipo de modelo que estamos registrando, en este caso es un modelo MLflow.
model = Model(
    name=model_name,
    path='./model',
    type=AssetTypes.MLFLOW_MODEL
)

# Registramos o actualizamos el modelo en Azure ML utilizando el cliente de modelos (ml_client.models).
# La función create_or_update registrará el modelo si no existe o lo actualizará si ya está registrado.
# - ml_client: una instancia del cliente de Azure ML que tiene acceso a la suscripción y el workspace donde
# se registrará el modelo.
# - models: accede a la parte del cliente responsable de manejar los modelos.
# - create_or_update: método para registrar o actualizar el modelo.
model = ml_client.models.create_or_update(model)
```




Implementación de un modelo de MLflow en un punto de conexión

BatchDeployment

Python

```
from azure.ai.ml.entities import BatchDeployment, BatchRetrySettings
from azure.ai.ml.constants import BatchDeploymentOutputAction

deployment = BatchDeployment(
    name="forecast-mlflow",
    description="A sales forecaster",
    endpoint_name=endpoint.name,
    model=model,
    compute="aml-cluster",
    instance_count=2,
    max_concurrency_per_instance=2,
    mini_batch_size=2,
    output_action=BatchDeploymentOutputAction.APPEND_ROW,
    output_file_name="predictions.csv",
    retry_settings=BatchRetrySettings(max_retries=3, timeout=300),
    logging_level="info",
)
ml_client.batch_deployments.begin_create_or_update(deployment)
```

Al implementar un modelo, deberá especificar cómo desea que se comporte el trabajo de puntuación por lotes.

La ventaja de usar un clúster de proceso para ejecutar el script de puntuación (generado automáticamente por Azure Machine Learning), es que puede ejecutar el **script de puntuación en instancias independientes en paralelo**.


```

# Importamos las clases BatchDeployment y BatchRetrySettings del módulo azure.ai.ml.entities.
from azure.ai.ml.entities import BatchDeployment, BatchRetrySettings

# Importamos BatchDeploymentOutputAction del módulo azure.ai.ml.constants.
from azure.ai.ml.constants import BatchDeploymentOutputAction

# Creamos una instancia de BatchDeployment para definir la configuración de un despliegue batch.
# Para ello, proporcionamos varios parámetros:
deployment = BatchDeployment(
    name="forecast-mlflow", # Nombre del despliegue batch.
    description="A sales forecaster", # Descripción del despliegue.
    endpoint_name=endpoint.name, # Nombre del endpoint donde se desplegará el modelo.
    model=model, # El modelo que queremos desplegar, definido previamente.
    compute="aml-cluster", # Nombre del cluster de computo de Azure ML que se utilizará.
    instance_count=2, # Número de instancias de computo a utilizar.
    max_concurrency_per_instance=2, # Máxima concurrencia por instancia.
    mini_batch_size=2, # Tamaño del mini batch para procesamiento batch.
    output_action=BatchDeploymentOutputAction.APPEND_ROW, # Acción a tomar con la salida del batch (en
este caso, agregar filas).
    output_file_name="predictions.csv", # Nombre del archivo donde se guardarán las predicciones.
    retry_settings=BatchRetrySettings(max_retries=3, timeout=300), # Configuración de reintentos: máximo
de 3 reintentos y 300 segundos de timeout.
    logging_level="info", # Nivel de logging para el despliegue.
)

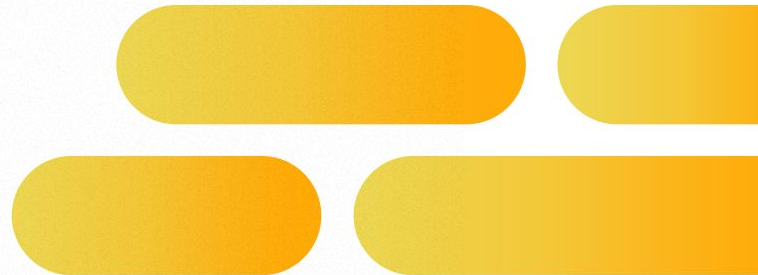
# Utilizamos el cliente de Azure ML para iniciar el proceso de creación o actualización del despliegue
batch.
# La función begin_create_or_update iniciará el proceso de creación o actualización del despliegue batch.
ml_client.batch_deployments.begin_create_or_update(deployment)

```

● Implementación de un modelo personalizado en un punto de conexión por lotes (Sin MLFLOW)

Para implementar un modelo, deberá crear el **script de puntuación**, definir el **entorno** necesario durante la inferencia y **archivos de modelo** almacenados en la ruta de acceso local o en el **modelo registrado**.

Para implementar un modelo, debe haber **creado** un **punto de conexión**. A continuación, puede implementar el modelo en el punto de conexión.



Creación del script de puntuación

`init(), run()`

Python

```
import os
import mlflow
import pandas as pd

def init():
    global model

    # get the path to the registered model file and load it
    model_path = os.path.join(os.environ["AZUREML_MODEL_DIR"], "model")
    model = mlflow.pyfunc.load(model_path)

def run(mini_batch):
    print(f"run method start: {__file__}, run({len(mini_batch)} files)")
    resultList = []

    for file_path in mini_batch:
        data = pd.read_csv(file_path)
        pred = model.predict(data)

        df = pd.DataFrame(pred, columns=["predictions"])
        df["file"] = os.path.basename(file_path)
        resultList.extend(df.values)

    return resultList
```

- **AZUREML_MODEL_DIR** es una variable de entorno que puede usar para buscar los archivos asociados al modelo.
- Use la variable **global** para que los recursos estén disponibles para puntuar los nuevos datos, como el modelo cargado.
- El tamaño de **mini_batch** se define en la configuración de implementación. Si los archivos del minilote son demasiado grandes para procesarse, debe dividir los archivos en archivos más pequeños.
- De forma predeterminada, las **predicciones** se escribirán en un **único archivo**.




```
# Importamos las librerías necesarias.
```

```
import os
import mlflow
import pandas as pd
```

```
# La función init se ejecuta una vez al inicio del despliegue para inicializar el modelo.
```

```
def init():
    global model # Declaramos una variable global para almacenar el modelo.
```

```
    # Obtenemos la ruta del archivo del modelo registrado desde la variable de entorno
    AZUREML_MODEL_DIR.
```

```
    model_path = os.path.join(os.environ["AZUREML_MODEL_DIR"], "model")
```

```
    # Cargamos el modelo utilizando mlflow.pyfunc.load.
```

```
    model = mlflow.pyfunc.load_model(model_path)
```

```
# La función run se ejecuta cada vez que se procesan datos en batch.
```

```
# mini_batch es una lista de rutas de archivos que contienen los datos a procesar.
```

```
def run(mini_batch):
    print(f"run method start: {__file__}, run({len(mini_batch)} files)") # Imprimimos
    información sobre la ejecución.
```

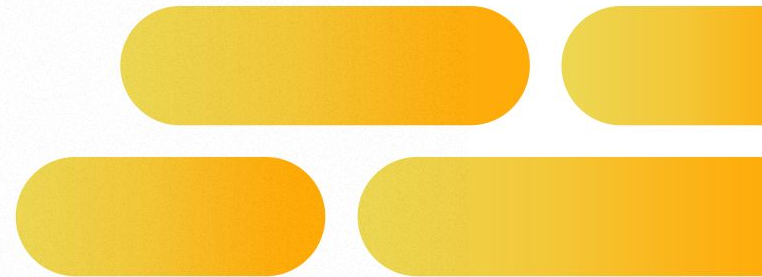
```
    resultList = [] # Inicializamos una lista para almacenar los resultados.
```




```
# Iteramos sobre cada archivo en el mini_batch.
for file_path in mini_batch:
    # Leemos los datos del archivo CSV.
    data = pd.read_csv(file_path)
    # Realizamos la predicción utilizando el modelo cargado.
    pred = model.predict(data)

    # Convertimos las predicciones a un DataFrame.
    df = pd.DataFrame(pred, columns=["predictions"])
    # Añadimos una columna con el nombre del archivo para referencia.
    df["file"] = os.path.basename(file_path)
    # Extendemos la lista de resultados con los valores del DataFrame.
    resultList.extend(df.values)

# Devolvemos la lista de resultados.
return resultList
```



Creación de un entorno

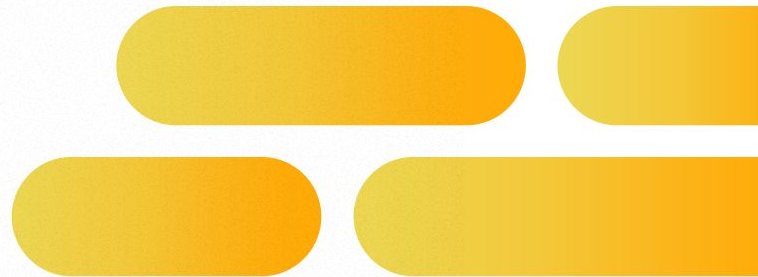
conda.yml

yml

```
name: basic-env-cpu
channels:
  - conda-forge
dependencies:
  - python=3.8
  - pandas
  - pip
  - pip:
    - azureml-core
    - mlflow
```

Puede crear un entorno con una imagen de Docker con dependencias de Conda o con Dockerfile.

También tendrá que agregar la biblioteca **azureml-core**, ya que es necesaria para el funcionamiento de las implementaciones por lotes.

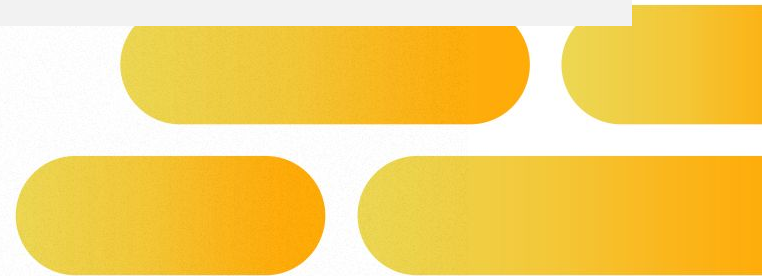


Creación de un entorno

Python

```
from azure.ai.ml.entities import Environment

env = Environment(
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    conda_file="./src/conda-env.yml",
    name="deployment-environment",
    description="Environment created from a Docker image plus Conda environment.",
)
ml_client.environments.create_or_update(env)
```





Importamos la clase Environment del módulo azure.ai.ml.entities.

```
from azure.ai.ml.entities import Environment
```

Creamos una instancia de Environment para definir la configuración del entorno de ejecución.

```
env = Environment(
```

```
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04", #
```

Especificamos la imagen de Docker que se usará como base.

```
    conda_file="./src/conda-env.yml", # Especificamos la ruta al archivo
```

Conda que define las dependencias del entorno.

```
    name="deployment-environment", # Asignamos un nombre al entorno.
```

```
    description="Environment created from a Docker image plus Conda
```

environment.", # Añadimos una descripción del entorno.

```
)
```

Utilizamos el cliente de Azure ML para crear o actualizar el entorno.

La función create_or_update se encargará de crear el entorno si no existe o

actualizarlo si ya existe.

```
ml_client.environments.create_or_update(env)
```



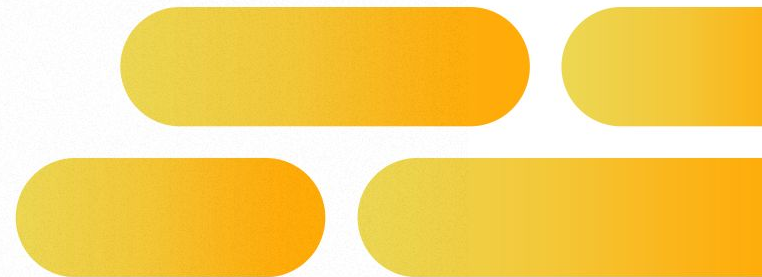
Creación de la implementación

BatchDeployment

Python

```
from azure.ai.ml.entities import BatchDeployment, BatchRetrySettings
from azure.ai.ml.constants import BatchDeploymentOutputAction

deployment = BatchDeployment(
    name="forecast-mlflow",
    description="A sales forecaster",
    endpoint_name=endpoint.name,
    model=model,
    compute="aml-cluster",
    code_path="./code",
    scoring_script="score.py",
    environment=env,
    instance_count=2,
    max_concurrency_per_instance=2,
    mini_batch_size=2,
    output_action=BatchDeploymentOutputAction.APPEND_ROW,
    output_file_name="predictions.csv",
    retry_settings=BatchRetrySettings(max_retries=3, timeout=300),
    logging_level="info",
)
ml_client.batch_deployments.begin_create_or_update(deployment)
```





Importamos las clases BatchDeployment y BatchRetrySettings del módulo azure.ai.ml.entities.

```
from azure.ai.ml.entities import BatchDeployment, BatchRetrySettings
```

Importamos BatchDeploymentOutputAction del módulo azure.ai.ml.constants.

```
from azure.ai.ml.constants import BatchDeploymentOutputAction
```

Creamos una instancia de BatchDeployment para definir la configuración de un despliegue batch.

```
deployment = BatchDeployment(  
    name="forecast-mlflow", # Nombre del despliegue batch.  
    description="A sales forecaster", # Descripción del despliegue.  
    endpoint_name=endpoint.name, # Nombre del endpoint donde se desplegará  
    el modelo.  
    model=model, # El modelo que queremos desplegar, definido previamente.  
    compute="aml-cluster", # Nombre del cluster de computo de Azure ML que  
    se utilizará.
```

```
.....  
.....  
.....  
.....
```



```
code_path="./code", # Ruta al directorio que contiene el código
necesario para la inferencia.
scoring_script="score.py", # Nombre del script de scoring que se
ejecutará para hacer predicciones.
environment=env, # Entorno de ejecución definido previamente.
instance_count=2, # Número de instancias de computo a utilizar.
max_concurrency_per_instance=2, # Máxima concurrencia por instancia.
mini_batch_size=2, # Tamaño del mini batch para procesamiento batch.
output_action=BatchDeploymentOutputAction.APPEND_ROW, # Acción a tomar
con la salida del batch (en este caso, agregar filas).
output_file_name="predictions.csv", # Nombre del archivo donde se
guardarán las predicciones.
retry_settings=BatchRetrySettings(max_retries=3, timeout=300), #
Configuración de reintentos: máximo de 3 reintentos y 300 segundos de
timeout.
logging_level="info", # Nivel de logging para el despliegue.
)
```

Utilizamos el cliente de Azure ML para iniciar el proceso de creación o actualización del despliegue batch.

~~# La función begin_create_or_update iniciará el proceso de creación o actualización del despliegue batch.~~

```
ml_client.batch_deployments.begin_create_or_update(deployment)
```

Desencadenamiento del trabajo de puntuación por lotes

`batch_endpoints.invoke`

Python

```
from azure.ai.ml import Input
from azure.ai.ml.constants import AssetTypes

input = Input(type=AssetTypes.URI_FOLDER, path="azureml:new-data:1")

job = ml_client.batch_endpoints.invoke(
    endpoint_name=endpoint.name,
    input=input)
```



```
# Importamos la clase Input del módulo azure.ai.ml.
from azure.ai.ml import Input

# Importamos AssetTypes del módulo azure.ai.ml.constants.
from azure.ai.ml.constants import AssetTypes



# Creamos una instancia de Input para definir la entrada del job batch.
input = Input(
    type=AssetTypes.URI_FOLDER, # Especificamos que el tipo de entrada es una
    carpeta URI.
    path="azureml:new-data:1" # Ruta a los datos en Azure ML. Esto hace referencia
    a un dataset registrado en Azure ML.
)









# Ejecutamos el job de inferencia batch utilizando el cliente de Azure ML.
# La función invoke enviará los datos al endpoint especificado y ejecutará la
inferencia.
job = ml_client.batch_endpoints.invoke(
    endpoint_name=endpoint.name, # Nombre del endpoint donde se desplegará el
    modelo.
    input=input # Entrada definida anteriormente que contiene la ruta a los datos.
)
```



Desencadenamiento del trabajo de puntuación por lotes

lote -						
Detalles Trabajos						
+ Crear trabajo Actualizar Editar columnas Restablecer vista						
Búsqueda Estado Creado por Etiquetas Todos los filtros Borrar todo						
Mostrando de 1 a 5 de 5 trabajos Tamaño de página: 25						
Nombre para mostrar	Estado	Creado el	Hora de inicio ↓	Duración	Creado por	Etiquetas
teal_snake_15h0585c	✓ Completado	7 de noviembre de 2022, 17:31	7 de noviembre de 2022, 17:31	6 min 9 s		
magenta_sail_hz2lz2dl	✗ Con errores ⓘ	7 de noviembre de 2022, 17:10	7 de noviembre de 2022, 17:10	8 min 9 s		
keen_wire_8xvn0lbz	✗ Con errores ⓘ	7 de noviembre de 2022, 16:38	7 de noviembre de 2022, 16:38	8 min 8 s		
clever_yak_r79s4xzz	✗ Con errores ⓘ	7 de noviembre de 2022, 16:01	7 de noviembre de 2022, 16:01	8 min 9 s		
clever_kitchen_vxtqj99s	✓ Completado	7 de noviembre de 2022, 11:32	7 de noviembre de 2022, 11:32	2 min 6 s		

● Solución de problemas de un trabajo de puntuación por lotes


teal_snake_15h0585c   Completado





 Datos


Salida de datos




dataset_param_config


 batchscoring
batchscoring

puntuación

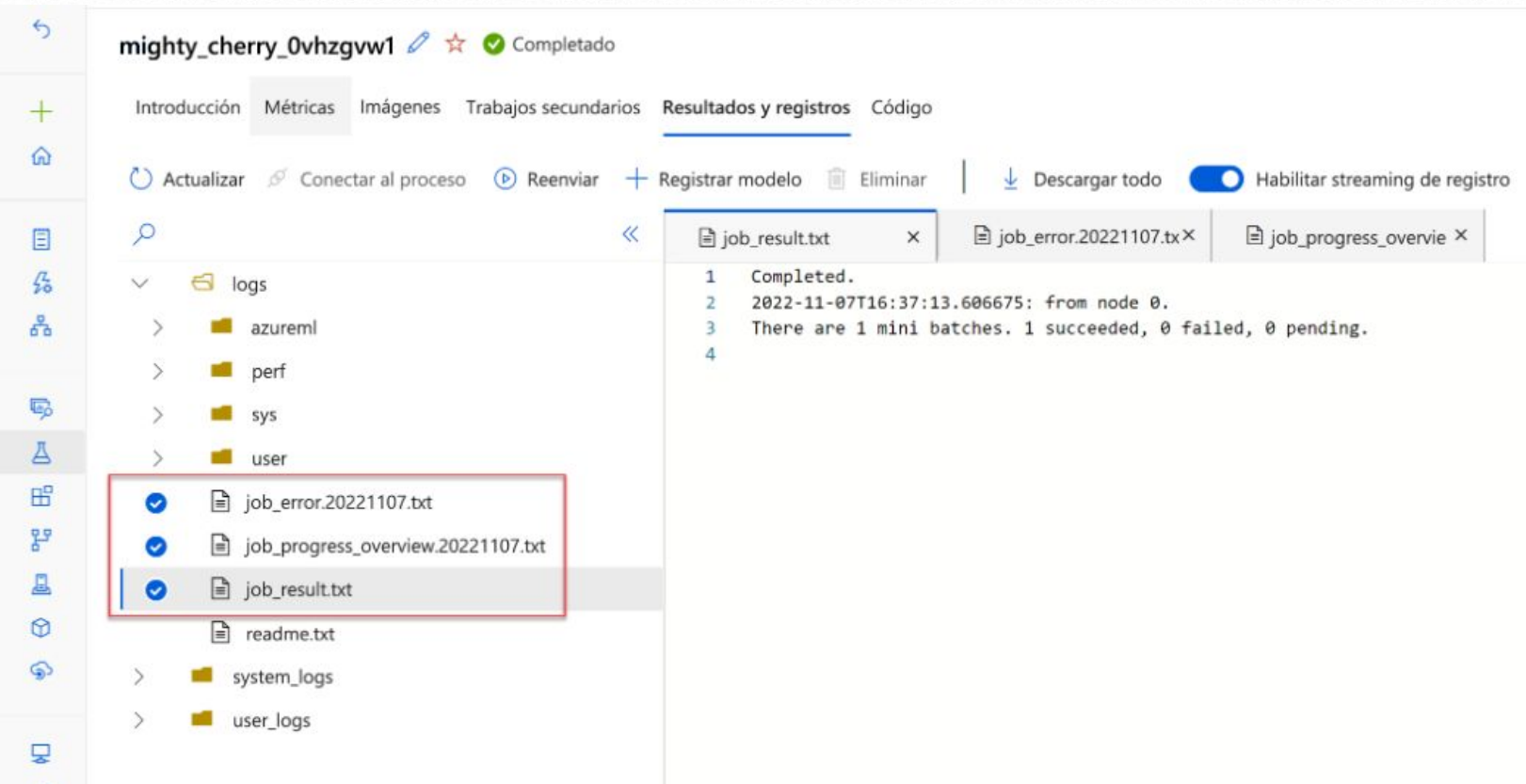
Información general del trabajo de canalización  

Introducción Parámetros de canalización Resultados y registros Métricas **Trabajos secundarios** 

 Actualizar  Registrar modelo  Conectar al proceso

Nombre para mostrar	Estado	Fecha de creación
mighty_cherry_0vhzgw1	 Completado	7 de noviembre de 2022, 17:31

Solución de problemas de un trabajo de puntuación por lotes



The screenshot displays the Azure ML portal interface for a job named **mighty_cherry_0vzhgww1**, which is marked as **Completado** (Completed). The **Resultados y registros** (Results and logs) tab is active, showing a file explorer on the left and a log viewer on the right.

File Explorer (Left):

- logs
 - azureml
 - perf
 - sys
 - user
 - ☒ job_error.20221107.txt
 - ☒ job_progress_overview.20221107.txt
 - ☒ job_result.txt
 - readme.txt
 - system_logs
 - user_logs

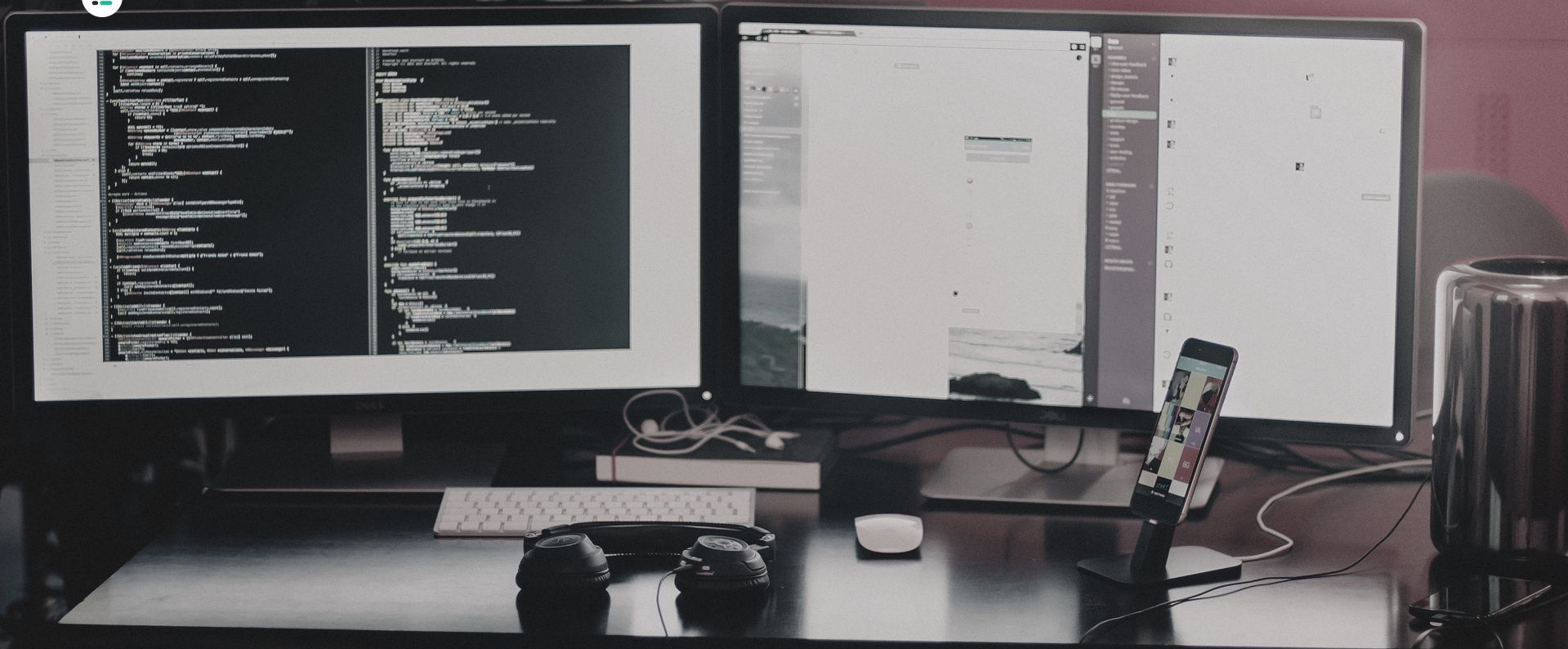
Log Viewer (Right):

The log viewer shows the contents of **job_result.txt**:

```
1 Completed.  
2 2022-11-07T16:37:13.606675: from node 0.  
3 There are 1 mini batches. 1 succeeded, 0 failed, 0 pending.  
4
```

Interface Elements:

- Navigation:** Introducción, Métricas, Imágenes, Trabajos secundarios, Resultados y registros (selected), Código.
- Actions:** Actualizar, Conectar al proceso, Reenviar, Registrar modelo, Eliminar, Descargar todo, Habilitar streaming de registro (toggle).
- File Tabs:** job_result.txt, job_error.20221107.tx, job_progress_overvie.



Vamos a la practica!!!