Herramientas para la Computación en la Nube Dirigida a Inteligencia Artificial

Tema 9. Implantación de MLOps a través de servicios en la nube

Índice

Esquema

IMPLANTACIÓN DE MLOPS A TRAVÉS DE SERVICIOS EN LA NUBE				
PARADIGMA MLOPS	MLFLOW	AWS SAGEMAKER	AZURE MACHINE LEARNING	GOOGLE VERTEX AI
▸ **Conceptos clave** · Automatización · Registro de experimentos	▸ **Características** · Creación de experimentos · Registro de ejecuciones · Comparación de métricas de rendimiento · Registro de modelos · Despliegue de modelos	▸ **Arquitectura** · S3 · RDS · Fargate · ELB · ECR · CDK	▸ **Estrategias** · Azure DevOps · GitHub Actions	▸ **Arquitectura** · Cloud Bild · Vertex Pipelines · Big Query · Artifact Registry · Google Cloud Storage
▸ **Tipos de *pipelines*** · Provisión de infraestructura · Entrenamiento de modelos · Despliegue de modelos · Creación de pipelines	▸ **Ejemplos** · `mlflow.set_experiment` · `mlflow.start_run` · `mlflow.sklearn.log_model` · `mlflow.end_run` · `mlflow.log_metric` · `mlflow.log_artifact` · `mlflow ui` · `mlflow.sklearn.load_model` · `mlflow models serve`	▸ **Implantación** · Prerequisitos de CDK · Instalación de CDK · Creación del stack · Acceso a la URL de MLflow · `mlflow.set_tracking_uri`	▸ **Implantación** · Requisitos previos · Service principal · Service connection · Importar repositorio · Seguridad de pipelines · Despliegue de infraestructura · Entrenamiento de modelos · Despliegue de modelos	▸ **Implantación** · Implantación de arquitectura con Terraform · Definición del pipeline de entrenamiento · Configuración de CI/CD

9.1. Introducción y objetivos

En este tema se describe el paradigma de **MLOps** y su aplicación a los entornos de computación en la nube. En primer lugar, se establecen los paralelismos entre este paradigma y la filosofía DevOps utilizada en el desarrollo de *software,* poniendo especial énfasis en la componente de automatización y en el registro de experimentos. Se describen, asimismo, los principales elementos que participan en el proceso: infraestructura, contenedores, *pipelines* y *endpoints.*

A continuación, se describen las características y funcionamiento de **MLflow,** la principal plataforma *open-source* de MLOps, que además tiene compatibilidad nativa con proveedores de nube como AWS y su plataforma de IA SageMaker.

También se describen las aproximaciones MLOps de los tres principales proveedores de computación en la nube**: Azure, AWS** y **Google Cloud.** Se establecen los paralelismos y diferencias entre los diferentes enfoques, señalando que todos ellos satisfacen los requisitos de automatización y registro de experimentos.

Por último, este tema se plantea los siguientes objetivos para el alumno:

- Ser capaces de explicar en qué consiste el paradigma MLOps a un cliente para determinar si tiene aplicación en el contexto de una organización que desee optimizar sus procesos de *machine learning.*
- Disponer de los conocimientos necesarios para utilizar MLflow con solvencia, materializando de esta forma los beneficios del paradigma MLOps.
- Ser capaces de diseñar una implementación MLOps con cualquiera de los principales proveedores de nube.

9.2. Características del proceso de MLOps

El **paradigma MLOps** se fundamenta en la necesidad de crear y desplegar modelos de una forma repetible y controlada. Para ello, se automatizan las tareas repetitivas para evitar errores y agilizar el proceso, y se registran los resultados obtenidos en cada prueba para garantizar que siempre se elige el modelo óptimo de entre todos los que se han construido durante la fase de experimentación.

Además, MLOps incluye **automatizaciones** para desplegar el modelo, monitorizar su rendimiento en producción y proporcionar *feedback* al equipo. De esa forma, el proceso de MLOps no es *one-shot,* es decir, no termina una vez desplegado el modelo, sino que implica generar nuevas versiones de los modelos para introducir mejoras o corregir errores de forma iterativa.

Esta forma de trabajar es muy similar a la utilizada en el **paradigma DevOps.** De hecho, MLOps es una adaptación de DevOps, considerando las particularidades del *machine learning.* La principal de estas particularidades es que los resultados obtenidos en cada iteración del proceso dependen no solo del código fuente empleado, sino de los datos facilitados en el entrenamiento del modelo.

En la Figura 1 se muestra un esquema del paradigma MLOps, donde puede comprobarse la similitud con las etapas del proceso DevOps:

Figura 1. Esquema de actividades de MLOps. Fuente: Alla y Adari, 2020.

Ideas clave

Para la **implementación** de las automatizaciones se utilizan diferentes *pipelines,* que son diagramas de tareas ejecutadas de forma secuencial o paralela para completar una etapa del proceso. Por ejemplo, en MLOps se utilizan *pipelines* para:

- **Provisionar la infraestructura** que se utilizará para construir modelos y servir modelos en producción. Generalmente, esta infraestructura se explota a través de contenedores Docker, cada uno de los cuales está especializado en una función del proceso MLOps.

- **Ejecutar las actividades de preparación de datos, entrenamiento de modelos, validación y selección de modelos y ajuste de hiperparámetros.** Si estas tareas se completan satisfactoriamente, el modelo se guarda en un registro de modelos, desde donde puede ser desplegado.

- **Desplegar modelos registrados en un *endpoint,*** desde donde puede ser utilizado para realizar inferencias en modo *batch* o en modo *online.*

- **Crear los propios *pipelines*** anteriores, con un *pipeline* maestro.

Por último, para el registro de trazas de la actividad de experimentación se utiliza un *framework* como MLflow o las plataformas equivalentes que proporcionan los proveedores de computación en la nube.

9.3. Uso de MLflow como plataforma de productización en la nube

Introducción

MLflow es una plataforma *open-source* para implementar el paradigma MLOps. Puede utilizarse en modalidad *on premise,* es decir, en el centro de datos de una organización, o bien desplegarse y utilizarse con un proveedor *cloud.* Los proveedores *cloud* proporcionan facilidades para hospedar y operar MLflow (AWS) o brindan plataformas con funcionalidades similares (Google, Azure).

Para utilizar el MLflow, solo es necesario incorporar llamadas a las funciones de su API en el código fuente. De forma nativa, MLflow soportar los *frameworks* más populares: scikit-learn, TensorFlow/Keras, Pytorch, ONNX, LangChain, H2O, Gluon, FastAI, Pyspark y LightGBM, entre otros.

De forma general, MLflow permite:

- **Crear experimentos,** es decir, un paquete que incluye modelos y sus métricas de rendimiento asociadas.
- **Registrar ejecuciones** *(model runs),* es decir, los resultados específicos obtenidos para un modelo dentro del experimento.
- **Comparar métricas de rendimiento** entre ejecuciones/experimentos.
- **Registro de modelos,** que permite seguir la evolución del modelo desde su creación inicial, a lo largo de un historial de versiones con las mejoras incorporadas gradualmente.
- **Despliegue de modelos,** que permite poner en marcha un servidor de inferencias para realizar predicciones sobre nuevos datos, es decir, operacionalizar el modelo.

En las siguientes secciones se describen con más detalle las características anteriores.

Creación de experimentos

MLflow permite crear experimentos y asociarle **ejecuciones** para probar con diferentes combinaciones de hiperparámetros y registrar los resultados obtenidos en cada caso.

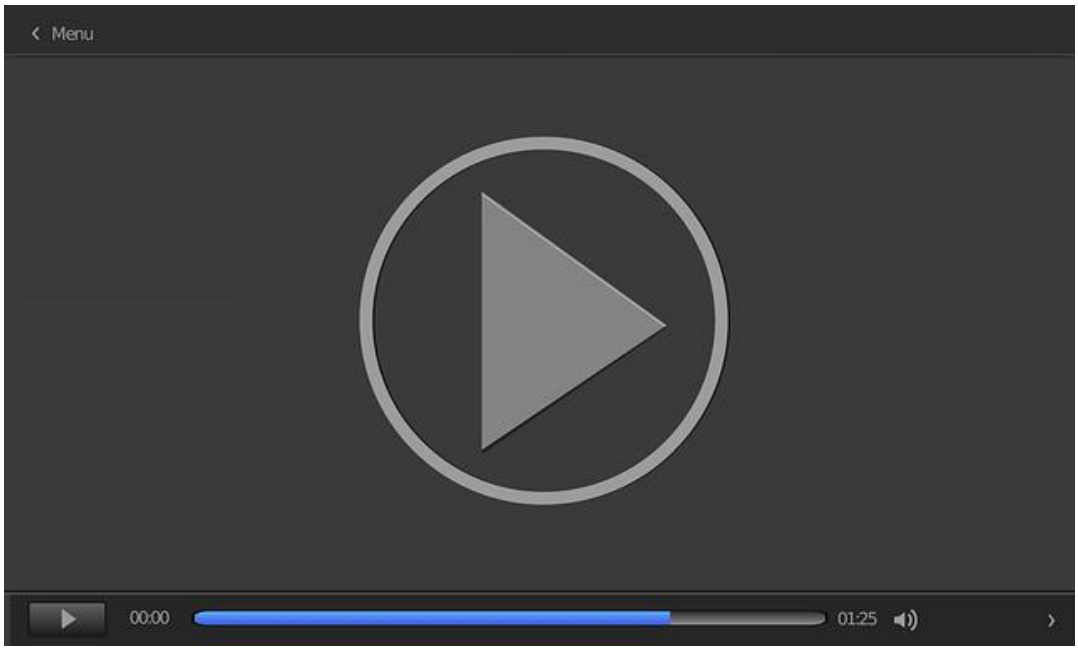
En el siguiente ejemplo se muestra cómo: 1) seleccionar un experimento, 2) lanzar una ejecución (`run`) basada en un modelo de regresión logística, 3) asociar esta ejecución al experimento y 4) registrar el modelo resultante. Si el experimento aún no existe, set_experiment() se encarga de crearlo. Por otro lado, es importante llamar a la función end_run() para confirmar la finalización de la ejecución actual, de manera que no se mezclen las trazas de diferentes ejecuciones.

```
sk_model = LogisticRegression(random_state=None, max_iter=400,
solver='newton-cg')

mlflow.set_experiment("scikit_learn_experiment")

with mlflow.start_run():

    train(sk_model, x_train, y_train)

    evaluate(sk_model, x_test, y_test)

    mlflow.sklearn.log_model(sk_model, "log_reg_model")

mlflow.end_run()
```

En el ejemplo anterior solo se ha lanzado una ejecución. Sin embargo, lo habitual es embeber el bloque `start_run()` en un bucle con cada una de las combinaciones de hiperparámetros que se desean probar. De esa forma, MLflow almacena un `run` (y sus métricas de rendimiento) por cada combinación, facilitando la comparación de los resultados en el proceso de ajuste.

La función `log_model()` tiene diferentes implementaciones en función del *framework* empleado para construir el modelo. En este caso, se utiliza la versión para sci-kit-learn.

En el siguiente vídeo, *Técnicas avanzadas de entrenamiento de modelos y ajuste de hiperparámetros con MLflow en Azure Machine Learning Workspaces,* se muestra un ejemplo real de ajuste de hiperparámetros mediante MLOps con MLflow en Azure Machine Learning.



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=2987c10a-059d-413f-aa59-b14900cad973

Registro de métricas y artefactos

Durante el proceso de entrenamiento, MLflow permite registrar valores de métricas de rendimiento con la función `log_metric()`, como se muestra a continuación:

```
def train(sk_model, x_train, y_train):

    sk_model = sk_model.fit(x_train, y_train)

    train_acc = sk_model.score(x_train, y_train)

    mlflow.log_metric("train_acc", train_acc)
```

También es posible utilizar `log_metric()` en la fase de validación. Por otro lado, es posible registrar los gráficos generados en esta etapa, que se incorporan como artefactos de MLflow mediante la llamada a `log_artifact()`:

```
def evaluate(sk_model, x_test, y_test):

    eval_acc = sk_model.score(x_test, y_test)

    auc_score = roc_auc_score(y_test, preds)

    mlflow.log_metric("eval_acc", eval_acc)

    mlflow.log_metric("auc_score", auc_score)

    roc_plot = plot_roc_curve(sk_model, x_test, y_test, name=' ROC Curve')

    plt.savefig("sklearn_roc_plot.png")

    mlflow.log_artifact("sklearn_roc_plot.png")
```

Acceso a la interfaz web

El servidor web de MLflow puede lanzarse con el siguiente comando:

```
mlflow ui
```

En la web aparecen los experimentos y sus respectivos *runs,* y es posible consultar las métricas de rendimiento obtenidas en cada caso:

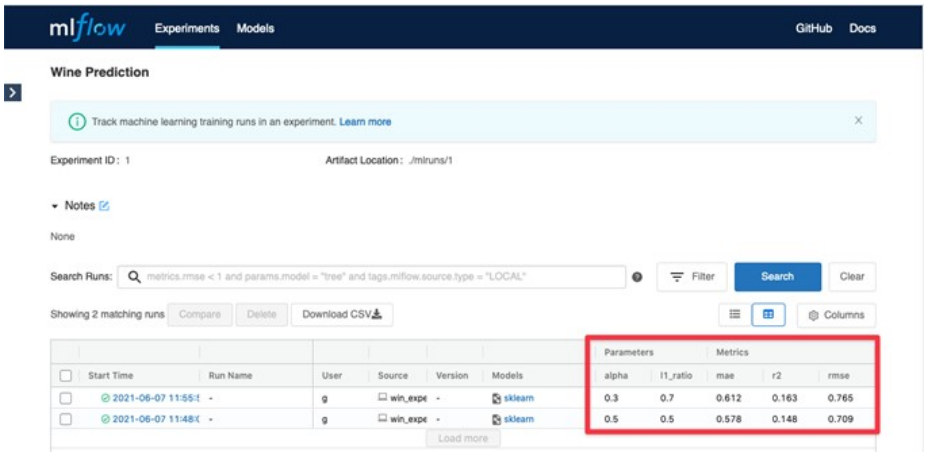


Figura 2. Interfaz web de MLflow con experimentos y ejecuciones. Fuente: Schmitt, s. f.

Carga y despliegue de modelos

Los modelos registrados para cada ejecución pueden cargarse posteriormente para realizar **predicciones.** Para ello se utiliza load_model() :

```
loaded_model = mlflow.sklearn.load_model("runs:/[ID
EJECUCIÓN]/log_reg_model"
```

También es posible desplegar el modelo en un *endpoint* a través de un servidor. Para ello, solo es necesario especificar el identificador de la ejecución (run) y el nombre del modelo:

```
mlflow models serve --model-uri runs:/[ID EJECUCIÓN]/[NOMBRE DEL MODELO] -p
4444
```

En el siguiente ejemplo, se muestra cómo lanzar **peticiones** al *endpoint*:

```
curl -X POST -H "Content-Type:application/json; format=pandas-split" –data
"[JSON]" "http://127.0.0.1:4444/invocations"
```

La referencia a [JSON] debe sustituirse por una lista de diccionarios en formato JSON, cada uno de los cuales representa un conjunto de valores de entrada (x) para los que se desea obtener una predicción (y).

9.4. Productización de modelos con SageMaker MLOps

La aproximación de MLOps de AWS SageMaker está basada en MLflow. En esta sección se muestra cómo desplegar un servidor MLflow en SageMaker a través de un conjunto de servicios de AWS: almacenamiento (S3), bases de datos (RDS), infraestructura (Fargate), balanceo de carga (ELB) y gestión de imágenes Docker (ECR). RDS implementa el *backend store* de MLflow y S3, el *artifact store.*

La arquitectura de servicios utilizada se muestra en la Figura 3:

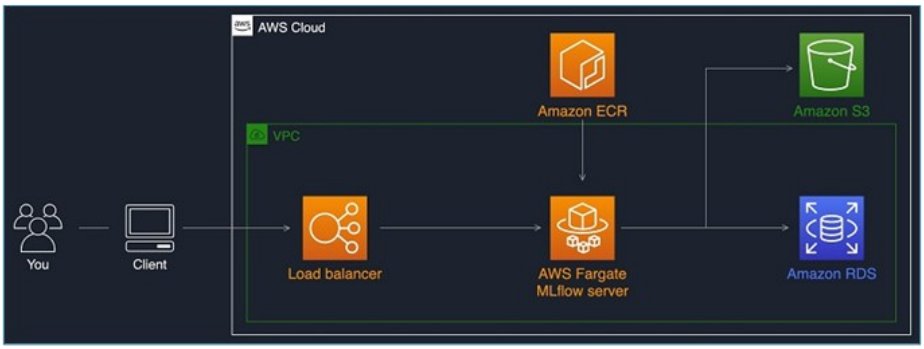


Figura 3. Arquitectura de servicios para despliegue de MLflow en SageMaker. Fuente: Hamiti, S. y Subramanian, 2021.

La **puesta en marcha de los servicios** se realiza con **CDK** (Cloud Development Kit). Este servicio recibe como entrada la definición de un *stack* de servicios y automatiza su despliegue.

AWS proporciona una definición del *stack* correspondiente a MLflow en la siguiente URL: https://github.com/aws-samples/amazon-sagemaker-mlflow-fargate/blob/main/app.py

---

Ideas clave

Para poder **desplegar** este *stack* con CDK es necesario:

- **Una cuenta de AWS.**
- **Una instalación local de Docker en el PC.** Se utiliza para construir la imagen del contenedor que implementa el servidor de MLflow.
- Un **clon** del siguiente repositorio Git: https://github.com/aws-samples/amazon-sagemaker-mlflow-fargate.git

La **instalación** de CDK se puede realizar con los siguientes comandos:

```
npm install -g aws-cdk@2.51.1

python3 -m venv .venv

source .venv/bin/activate

pip3 install -r requirements.txt
```

La creación del *stack* de servicios puede completarse con los comandos que se incluyen abajo. Los dos primeros se utilizan para obtener el identificador de la cuenta de AWS y la región en la que se desplegarán los servicios. Los **comandos** cdk construyen la imagen Docker con el servidor MLflow, la suben a ECR (el Docker Registry de AWS) y despliegan el *stack.*

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account | tr -d '"')

AWS_REGION=$(aws configure get region)

cdk bootstrap aws://${ACCOUNT_ID}/${AWS_REGION}

cdk deploy --parameters ProjectName=mlflow --require-approval never
```

Ideas clave

Para acceder a la interfaz web de MLflow, se utiliza la URL del balanceador de carga, que se muestra en la columna «Value» de la lista de recursos creados.

A partir de este momento, se puede utilizar la API de MLflow Tracking para registrar las **trazas** de toda la actividad de creación de modelos: hiperparámetros, modelos, métricas de rendimiento, imágenes de gráficas y otros artefactos. Para ello, se tiene que incluir el siguiente fragmento de código en los *scripts* de entrenamiento de modelos ejecutados en SageMaker:

```
import mlflow

mlflow.set_tracking_uri('<YOUR LOAD BALANCER URI>')
```

En la interfaz de MLflow, quedarán disponibles las trazas de cada experimento, facilitando la comparación de los resultados obtenidos por diferentes modelos y la elección del modelo óptimo que debe ser desplegado.

9.5. Productización de modelos con Azure Machine Learning

Introducción

L a **aproximación de MLOps de Azure** consta de dos componentes: la automatización del proceso de creación y despliegue de modelos a través de DevOps (en Azure DevOps o GitHub Actions), y el registro de trazas de los experimentos realizados (en Azure Machine Learning).

Azure proporciona dos plataformas diferentes para la **componente de automatización: Azure DevOps** y **GitHub Actions.** Estas plataformas automatizan las actividades de despliegue de infraestructura, preparación de datos, entrenamiento de modelos, evaluación de rendimiento, despliegue de modelos y monitorización de su funcionamiento en producción.

A continuación, se describe el **procedimiento** para implementar **tres *pipelines*** en Azure DevOps: uno para la provisión de infraestructura, otro para la creación de modelos y otra para el despliegue del modelo en un *endpoint,* de manera que pueda ser consumido de forma *online* para realizar inferencias.

MLOps con Azure DevOps

El procedimiento que se describe en esta sección está basado en el siguiente repositorio de Github: https://github.com/Azure/mlops-v2-ado-demo. Este repositorio es el *quickstart* proporcionado por Azure para acelerar el despliegue de MLOps.

Para la **implementación** de los *pipelines,* deben cumplirse los siguientes requisitos:

- Disponer de una suscripción de Azure.
- Haber creado una organización en Azure DevOps. El concepto de organización es equivalente al de *workspace* en Azure Machine Learning: proporciona un entorno dedicado del servicio.

Configuración inicial

El **primer paso** consiste en la creación de una cuenta de usuario para autenticarse en Azure desde Azure DevOps. Este usuario se denomina *service principal.* Por seguridad, se crea un *service principal* por cada entorno (desarrollo, producción).

```
projectName="[PROYECTO]"

roleName="Contributor"

subscriptionId="[SUSCRIPCIÓN]"

environment="<Dev|Prod>"

servicePrincipalName="Azure-ARM-${environment}-${projectName}"

az ad sp create-for-rbac --name $servicePrincipalName --role $roleName --
scopes /subscriptions/$subscriptionId
```

El resultado del comando `az` incluye identificador y credenciales del usuario, así como el identificador del *tenant* en el que ha sido creado. Estos datos deben ser anotados, puesto que se necesita a continuación.

El siguiente paso consiste en crear un **proyecto** en Azure DevOps y configurar una ***service connection*** en «Project Settings». Las *service connections* sirven para conectar Azure DevOps a servicios de Azure a través de un *service principal* —que hemos creado en el paso anterior—. Esta conexión se utiliza para **automatizar** la provisión de la infraestructura que hospedará el modelo en Azure. El servicio que realiza la provisión es Azure Resource Manager (ARM).

Para crear la *service connection,* se debe hacer clic en «Create Service Connection», seleccionar «Azure Resource Manager» y rellenar los siguientes valores: nombre e identificador de la suscripción y *tenant*, identificador y contraseña del *service principal.*

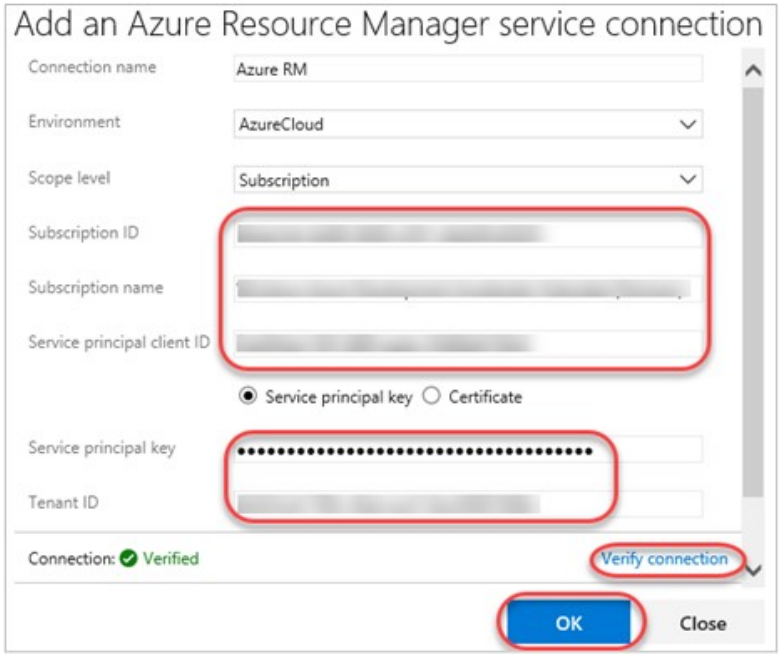


Figura 4. Configuración de service connection en Azure DevOps. Fuente: Azure DevOps Labs, 2019.

A continuación, se debe importar el repositorio Git que almacena los ficheros del proyecto —código fuente, datos, configuraciones y otros artefactos—. Para ello, se debe seleccionar la sección «Repos», y a continuación «Import Repository».

Figura 5. Cuadro de diálogo para importar el repositorio Git a Azure DevOps. Fuente: Blackmist, s-polly, AbeOmor, hyoshioka0128, 19BMG00, shohei1029 y samelhousseini, 2023.

En la misma sección de «Repos», se debe seleccionar «Repositories», elegir el repositorio recién creado y hacer clic en la pestaña «Security». El usuario con el sufijo «Build Service» es el encargado de gestionar el repositorio Git. Debe tener los permisos «Contribute» y «Create branch» con el valor «Allow».

El último paso es la configuración de seguridad asociada a los *pipelines.* Para ello, se debe hacer clic en «Pipelines», y después en el botón con tres puntos verticales (ver Figura 4). En «Manage security» se debe otorgar al usuario con el sufijo Build service, el permiso Edit build pipeline. Este usuario se utilizará para modificar los *pipelines* cuando sea necesario.

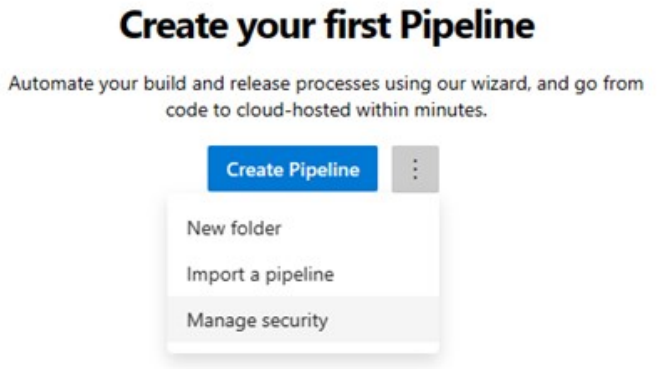


Figura 6. Configuración de permisos del pipeline en Azure DevOps. Fuente: Blackmist, s-polly, AbeOmor, hyoshioka0128, 19BMG00, shohei1029 y samelhousseini, 2023.

Despliegue de infraestructura

El siguiente paso es la creación en Azure DevOps del *pipeline* encargado de la provisión de infraestructura. Esta infraestructura se compone de un grupo de recursos, un *workspace* de Azure Machine Learning (AML) y un *cluster* de computación, dentro del *workspace.*

Para crear el *pipeline,* se hace clic en «Create Pipeline», se selecciona «Azure Repos Git» y se elige el repositorio creado en la sección anterior. A continuación, se selecciona el fichero YAML que incluye la definición del *pipeline* — mlops/devops-pipelines/cli-ado-deploy-infra.yml — en este ejemplo.

En la Figura 7 se incluye un fragmento de este fichero:

```
# Create the Azure ML workspace
# Note: The step uses Azure CLI to create the Azure ML workspace.
#
#       See the documentation for more information.
#       https://docs.microsoft.com/en-us/cli/azure/extension?view=azure-cli-latest#az_extension_add
- template: /infra/create-workspace.yml
  parameters:
    rg_name: $(resource_group)
    workspace_name: $(aml_workspace)
    location: $(location)
```

Figura 7. Fichero cli-ado-deploy-infra.yml con la definición de infraestructura. Fuente: Microsoft Azure, Cindyweng, setuc, iamramengirl, samelhousseini, dependabot[bot]m dhangerkapil y microsoft-github-operations[bot], 2024.

Desde cli-ado-deploy-infra.yml se referencia a otro fichero, config-infra-prod.yml , que contiene la definición del *cluster* AML:

```
# Azure DevOps
ado_service_connection_rg: Azure-ARM-Prod
ado_service_connection_aml_ws: Azure-ARM-Prod

# Compute target for pipeline
training_target: cpu-cluster
training_target_sku: STANDARD_D2_V2
training_target_min_nodes: 0
training_target_max_nodes: 4
training_target_tier: dedicated
```

Figura 8. Fichero config-infra-prod.yml con la definición de infraestructura. Fuente: Blackmist, s-polly, AbeOmor, hyoshioka0128, 19BMG00, shohei1029 y samelhousseini, 2023.

Una vez creado, solo es necesario ejecutar el *pipeline* para que toda la infraestructura quede automáticamente provisionada. Es importante resaltar que esta aproximación —basada en el paradigma MLOps— facilita la **reproducibilidad y control de cambios** del proceso de provisión.

Entrenamiento de modelos

El siguiente paso es crear el *pipeline* encargado del entrenamiento del modelo. Para ello, se repiten los pasos de la sección anterior, pero en esta ocasión se selecciona el fichero /mlops/devops-pipelines/deploy-model-training-pipeline.yml .

En la Figura 9 se incluye un fragmento de su contenido:

```yaml
  # Run the ML Pipeline
  # Note: This step uses Azure ML CLI to register the data in AML.
- template: /aml-cli-v2/register-data.yml
  parameters:
    data_type: uri_file
    data_name: taxi-data
    data_file: mlops/azureml/train/data.yml


  # Run the ML Pipeline
  # Note: This step uses Azure ML CLI to run the pipeline.
  #       The step also waits for the pipeline to complete after submitting the pipeline.
  #       This may not be the best practice for production pipelines.
  #       In production, you may want to run the pipeline and then continue with other steps.
  #       The pipeline can be monitored and the pipeline can be stopped if needed from the ML Workspace
  #
  #       See the documentation for more information.
  #       https://learn.microsoft.com/en-us/azure/machine-learning/how-to-train-model?tabs=azurecli#3-run-the-pipeline
- template: /aml-cli-v2/run-pipeline.yml
  parameters:
    pipeline_file: mlops/azureml/train/pipeline.yml
    experiment_name: $(environment)_taxi_fare_train_$(Build.SourceBranchName)
    display_name: $(environment)_taxi_fare_run_$(Build.BuildID)
```

Figura 9. Fichero deploy-model-training-pipeline.yml que describe el pipeline de entrenamiento. Fuente: Microsoft Azure, Cindyweng, setuc, iamramengirl, samelhousseini, dependabot[bot]m dhangerkapil y microsoft-github-operations[bot], 2024.

En el fragmento mostrado se incluye el proceso de registro de los datos utilizados en el entrenamiento en Azure Machine Learning (AML) y la generación de un **nuevo experimento** (asociado al entrenamiento). Una vez completado el entrenamiento, el modelo resultante se registra en AML como un artefacto más.

Este *pipeline* se encarga, por tanto, de las mismas actividades que tendríamos que realizar en la consola web de AML de forma manual, con la ventaja de que en este caso: 1) son **reproducibles** y 2) están **automatizadas.** Ambas ventajas son, nuevamente, principios clave del paradigma MLOps.

Despliegue de modelos

El último paso es crear el *pipeline* encargado del despliegue del modelo en un *endpoint,* desde el que puede ser utilizado para realizar inferencias. Para ello, se repiten los pasos de la sección anterior, pero en esta ocasión se selecciona el fichero /mlops/devops-pipelines/deploy-online-endpoint-pipeline.yml .

En la Figura 10 se incluye un fragmento de su contenido:

```
# Create the online endpoint
# Note: The step uses Azure ML CLI to create the online endpoint.
#
#       See the documentation for more information.
#       https://learn.microsoft.com/en-us/azure/machine-learning/how-to-deploy-and-where?tabs=python#create-an-endpoint
- template: /aml-cli-v2/create-endpoint.yml
  parameters:
    endpoint_file: mlops/azureml/deploy/online/online-endpoint.yml

  # Create the online deployment
  # Note: The step uses Azure ML CLI to create the online deployment.
  #
  #       See the documentation for more information.
  #       https://learn.microsoft.com/en-us/azure/machine-learning/how-to-deploy-and-where?tabs=python#create-a-deployment
- template: /aml-cli-v2/create-deployment.yml
  parameters:
    deployment_name: taxi-online-dp
    deployment_file: mlops/azureml/deploy/online/online-deployment.yml
```

Figura 10. Fragmento del fichero deploy-online-endpoint-pipeline con el pipeline de despliegue. Fuente: Microsoft Azure, Cindyweng, setuc, iamramengirl, samelhousseini, dependabot[bot]m dhangerkapil y microsoft-github-operations[bot], 2024.

En este fragmento se define la creación del *endpoint* y el *deployment* asociado —cada *deployment* se refiere a una versión del modelo—.

Con este *pipeline* se completa el proceso MLOps: disponemos de un **modelo en producción,** que ha sido construido y desplegado de forma completamente automatizada a través de *pipelines.* De esta forma, el esfuerzo asociado al ciclo de puesta en producción se reduce sustancialmente. Esto permite al equipo disminuir el esfuerzo en tareas repetitivas y concentrarse en la construcción de modelos óptimos, así como en la identificación de mejoras para preservar o mejorar su rendimiento a lo largo del tiempo.

9.6. Productización de modelos con Vertex AI MLOps

Introducción

La aproximación de MLOps de **Google** es similar a la de Azure: se basa en la automatización del proceso mediante *pipelines* y el registro de las trazas de cada experimento en Vertex AI.

La arquitectura de referencia de Google Cloud para la implementación de MLOps es la que se refleja en la Figura 11:

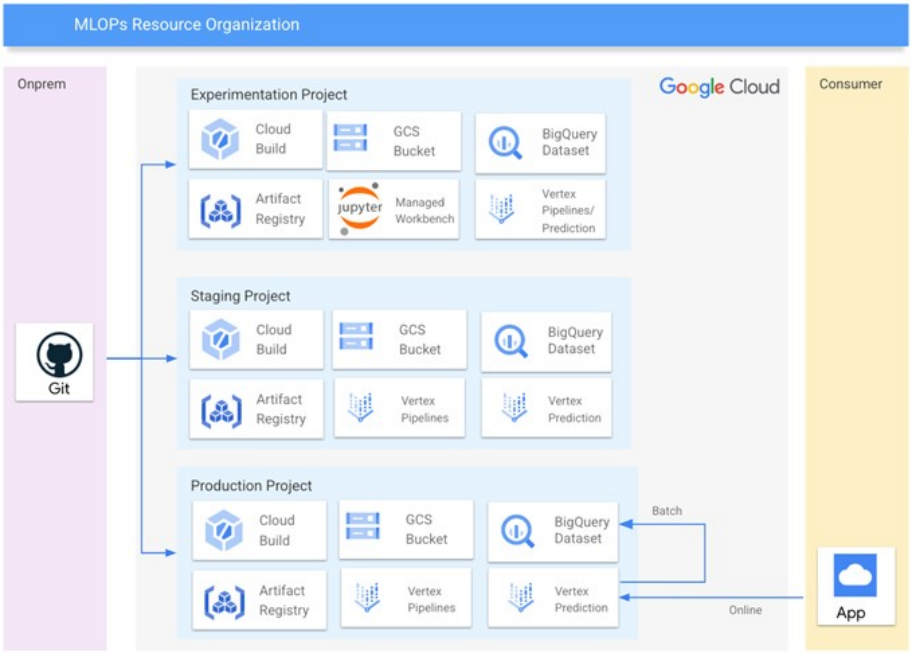


Figura 11. Arquitectura de referencia de Google Cloud para MLOps. Fuente: Puga, 2024a.

Esta arquitectura contempla tres entornos: **experimentación, preproducción** *(staging)* y **producción.** En cada uno de estos se utilizan servicios similares. Son los siguientes:

- **Cloud Build,** plataforma de CI/CD, utilizada para generar las imágenes Docker necesarias a partir del código fuente alojado en un repositorio.
- **Vertex Pipelines,** que permite definir e implementar *pipelines* con las fases del proceso de creación de modelos.
- **Big Query,** como entorno de almacenamiento de los *datasets.*
- **Artifact Registry,** como plataforma de registro de las imágenes Docker.
- **Google Cloud Storage (GCS),** para almacenar los artefactos generados por Cloud Build y Vertex AI.

También es necesario disponer de Git en cada uno de los PCs que acceden al repositorio para crear y evolucionar el código fuente asociado al modelo.

Creación del entorno MLOps

En esta sección se resumen los pasos a seguir para la puesta en marcha de una **arquitectura de MLOps** similar a la descrita en la sección previa. Dada la complejidad del proceso completo, no se incluyen los detalles técnicos de implementación, sino un *overview* de la estrategia a seguir.

Implementación de arquitectura

La provisión de la arquitectura de servicios mostrada en la sección anterior se realiza con **Terraform,** que es una plataforma de automatización de la gestión de infraestructura, compatible con Google Cloud.

Cada **entorno** (desarrollo, *staging,* producción) se provisiona mediante un *script* Terraform, que incluye las variables propias del entorno.

Antes de lanzar los *scripts* Terraform, se debe clonar el repositorio GitHub que proporciona Google Cloud, como quickstart de MLOps, alojado en la siguiente URL: https://github.com/GoogleCloudPlatform/professional-services

La **ruta a estos ficheros** en el repositorio Git es:

```
GoogleCloudPlatform/professional-
services/blob/main/examples/vertex_mlops_enterprise/terraform/01-
dev/main.tf
```

```
GoogleCloudPlatform/professional-
services/blob/main/examples/vertex_mlops_enterprise/terraform/02-
staging/main.tf
```

```
GoogleCloudPlatform/professional-
services/blob/main/examples/vertex_mlops_enterprise/terraform/03-
prod/main.tf
```

Los **recursos clave** provisionados en este proceso son:

- Un **proyecto en Google Cloud Platform** (GCP) para albergar todos los demás recursos.
- Una **red privada** para la comunicación entre Vertex —la plataforma de AI— y Dataflow —fuente de datos—.
- ***Buckets*** (contenedores de almacenamiento) en Google Cloud Storage para almacenar los artefactos generados por Vertex y Dataflow.
- **Un d*ataset* de BigQuery**, en el que se almacenarán los datos de entrenamiento.
- **Cuentas de servicio** utilizadas por Vertex y Dataflow para autenticarse en los servicios de la arquitectura, y otra cuenta de servicio para GitHub.

En la siguiente captura se muestra un fragmento del *script* Terraform para el **entorno de desarrollo:**

```
  github = {
    organization = var.github.organization
    repo         = var.github.repo
    branch       = var.environment
  }

  identity_pool_claims = try("attribute.repository/${var.github.organization}/${var.github.repo}", null)

  project_config = {
    billing_account_id = var.project_config.billing_account_id
    parent             = var.project_config.parent
    project_id         = "${var.project_config.project_id}"
  }

}
module "mlops" {
  source                = "github.com/GoogleCloudPlatform/cloud-foundation-fabric//blueprints/data-solutions/vertex-mlops"
  project_config        = local.project_config
  prefix                = var.prefix
  bucket_name           = local.bucket_name
  dataset_name          = var.dataset_name
  groups                = var.groups
  identity_pool_claims  = local.identity_pool_claims
  labels                = local.labels
  notebooks             = var.notebooks
  service_encryption_keys = var.service_encryption_keys
}
```

Figura 12. Fragmento del *script* Terraform que provisiona el entorno de desarrollo. Fuente: elaboración propia.

Definición de *pipeline* de entrenamiento

La componente de automatización del enfoque MLOps de Google Cloud se materializa de la misma forma que hemos visto para Azure: con *pipelines.* Estos se definen e implementan con el servicio Vertex AI Pipelines. A su vez, este servicio soporta de forma nativa **dos tipos de *pipelines:***

- Los basados en **Kubeflow Pipelines (kfp).** Esta es una plataforma de *machine learning* (ML) para construir y desplegar modelos en **Kubernetes,** que es una plataforma de orquestación de aplicaciones basada en contenedores —generalmente Docker—. Por otro lado, Kubeflow Pipelines es un componente de Kubeflow para definir y ejecutar *pipelines* de *machine learning.* Estos *pipelines* permiten ejecutar de forma secuencial o paralela todas las fases del proceso de creación y despliegue de modelos.

- Los basados en **TensorFlow Extended (tfx).** Esta es una plataforma de creación de modelos basados en redes neuronales. TensorFlow Extended es un componente de TensorFlow con la misma función que KubeFlow Pipelines, pero específicamente diseñado para TensorFlow.

En el *quickstart* proporcionado por Google en GitHub se incluyen ejemplos de ambos tipos para completar la fase de entrenamiento.

```
GoogleCloudPlatform/professional-
services/tree/main/examples/vertex_pipeline/pipelines
```

En la Tabla 1 se incluyen los *steps* de cada *pipeline.*

Tipo de *pipeline*	*Steps* del *pipeline*
KubeFlow	1. Cargar los datos desde BigQuery y hacer el *split* (entrenamiento, validación, pruebas). 2. Entrenamiento del modelo: generar el modelo y las métricas de rendimiento. 3. Subir el modelo a Vertex Model Registry. 4. Generar las predicciones para los datos de validación, calcular las métricas de rendimiento e incorporarlas al modelo en Vertex Model Registry. 5. Generación de un *model card*, es decir, un resumen de los resultados, que incluye: a. Visualizaciones. b. Estadísticas de rendimiento.
TensorFlow	1. Cargar los datos desde BigQuery y transformarlos al formato óptimo de TensorFlow. 2. Generar estadísticas de los datos: media, mediana, máximo, mínimo, datos ausentes. 3. Importar un *schema* de los datos, que incluye rangos de variables numéricas y valores posibles de variables categóricas. 4. Validar que los datos proporcionados respetan el *schema*. 5. Entrenamiento del modelo, con los valores de hiperparámetros configurados. 6. Validación del modelo, para lo cual se puede comparar el rendimiento con otro modelo, por ejemplo, el que está actualmente en producción. 7. Publicar el modelo si los resultados son mejores que los del modelo con el que se realiza la validación

Tabla 1. Pasos incluidos en cada tipo de pipeline de Vertex AI Pipelines. Fuente: elaboración propia.

En la Figura 13 se incluye como ejemplo un fragmento del *pipeline* basado en KubeFlow, en el que aparecen reflejados los pasos *(steps)* anteriores:

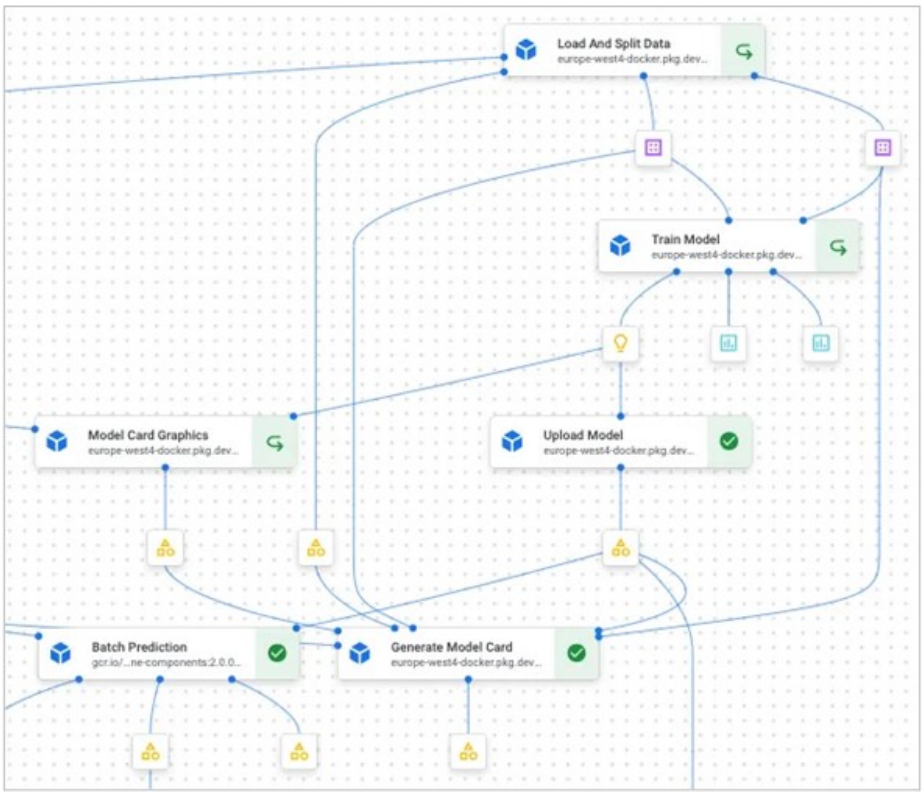


Figura 13. Ejemplo de *pipeline* basado en KubeFlow Pipelines para el entrenamiento del modelo. Fuente: Balm, 2024.

En los pasos incluidos en ambos tipos de *pipeline* quedan patentes los principios clave del paradigma MLOps:

- La **automatización de las tareas repetitivas** que comprende el ciclo de vida de creación y despliegue de modelos.
- El **registro de trazas** con los resultados de cada experimento.

En la Figura 14 se muestra un ejemplo de métricas de rendimiento asociadas a la ejecución de un experimento en la interfaz gráfica de Vertex AI:

Figura 14. Métricas de rendimiento obtenidas en la ejecución del pipeline. Fuente: Nardini y Dugan, 2022.

Configuración de CI/CD

Todos los recursos utilizados en el proyecto MLOps deben estar definidos en un **repositorio.** Por ejemplo, en la Figura 15, el repositorio es el *quickstart* de GitHub referenciado en secciones previas, proporcionado por Google.

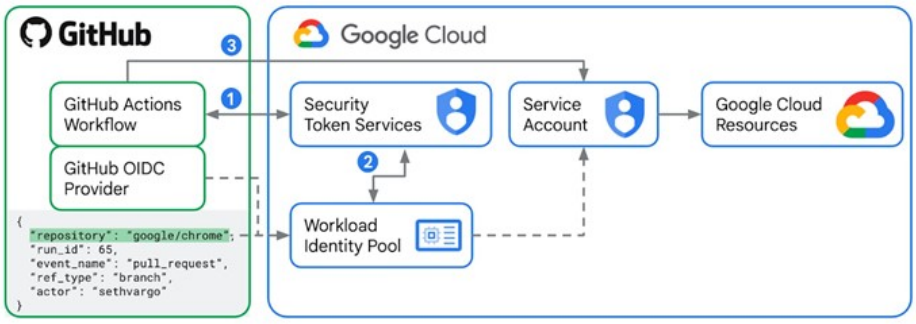


Figura 15. Interacción entre Google Cloud y el repositorio GitHub utilizado como *quickstart* de MLOps. Fuente: Puga, 2024b.

Ideas clave

Este repositorio incluye el **código fuente** asociado a cada fase del proceso de creación y despliegue de modelos, la definición de **imágenes Docker** utilizadas para cada una de ellas y la propia definición de los ***pipelines*** vistos en la sección anterior. Puede utilizarse como base para implementación de nuestro propio proceso de MLOps, modificando los parámetros e implementaciones para ajustarlos a un caso de uso diferente.

A través de este repositorio, también es posible iniciar la **ejecución** de los *pipelines,* bien de forma manual o condicionado a la ocurrencia de un evento. El evento más habitual es un *commit* en la rama *master,* que significa que está disponible una nueva versión de la implementación del modelo. En ese caso, disparamos el *pipeline* para construir el nuevo modelo y comprobar si mejora las predicciones del actualmente desplegado; en caso afirmativo, el nuevo modelo reemplaza al anterior.

El repositorio también proporciona ***workflows*** para automatiza otras actividades, más allá de la ejecución del *pipeline.* Por ejemplo:

- **Creación de contenedores Docker.** Se refiere a la preparación de los contenedores necesarias para ejecutar el resto del proceso. En este ejemplo, se utilizan los siguientes contenedores:
  - **Contenedor de CI/CD.** Este *workflow* se utiliza para las tareas relacionadas con la ejecución del *pipeline.*
  - **Contenedor de Vertex.** Se utiliza para ejecutar el proceso de entrenamiento, que es uno de los *steps* del *pipeline.*
  - **Contenedor de Dataflow.** Es utilizado por TensorFlow Extended para acceder a los datos utilizados para el entrenamiento.
  - **Contenedor de Model Card.** Se utiliza para crear el *model card* o resumen de resultados.

- **Construir el *pipeline.*** Este *workflow* se utiliza para compilar la definición del *pipeline* que existe en el repositorio y almacenar esta versión compilada en Google Cloud Storage como un artefacto más del proceso.

- **Desplegar el modelo.** Este *workflow* se utiliza para desplegar un modelo almacenado en Vertex Model Registry. Para ello, se crea un *endpoint,* se procede al despliegue del modelo y se verifica que el *endpoint* funciona correctamente.

Estas automatizaciones están disponibles a través de GitHub Actions, un *framework* de creación de *workflows* que proporciona GitHub.

Conclusiones

Google Vertex AI proporciona un **repositorio GitHub** con todos los elementos necesarios para poner en marcha un proyecto de MLOps, incluyendo: la provisión de servicios (con Terraform), la creación de contenedores Docker especializados, la implementación de *pipelines* para la automatización del proceso de entrenamiento (Vertex AI Pipelines), el registro de trazas de experimentos en Vertex AI y el despliegue de modelos en un *endpoint.*

La puesta en marcha de este proyecto implica **la personalización de los parámetros de cada fichero configuración** del repositorio para adaptarlo a nuestro caso de uso. Por último, este proyecto evolucionará en el tiempo para reflejar los cambios derivados del proceso de mejora continua que implica el paradigma MLOps. El control de cambios necesario para gestionar correctamente estas evoluciones está garantizado por el uso de un entorno de CI/CD como GitHub.

9.7. Cuaderno de ejercicios

1. Utilizando la REST API de MLflow, definir el JSON que permite registrar de forma conjunta *(log-batch)* los siguientes valores de un `run`:

- Métricas de rendimiento: Mean Absolute Error (MAE): 3.12, Root Mean Squared Error (RMSE): 2.86. Ambos con un timestamp de 1552440801.
- Tipo de modelo: regresión logística.

```
{
   "run_id": "[ID EJECUCIÓN]",
   "metrics": [
     {"key": "mae", "value": 3.12, "timestamp": 1552550804},
     {"key": "rmse", "value": 2.86, "timestamp": 1552550804},
   ],
   "params": [
     {"key": "model_class", "value": "LogisticRegression"},
   ]
}
```

2. Utilizando el SDK de MLflow para Python, escribir el código fuente que realiza las siguientes tareas:

- Importar el Iris *dataset* de scikit-learn, el módulo de árboles de decisión y el módulo de MLflow para scikit-learn.
- Iniciar un nuevo run en el que 1) se carga el *dataset* y 2) se entrena un árbol de decisión estándar.
- Se registran los parámetros *criterion* (criterio para decidir los *splits,* por defecto el coeficiente Gini) y *splitter* del modelo.
- Extraer la firma del modelo a partir de los datos.
- Registrar el modelo, incluyendo su firma.

```
import mlflow

import mlflow.sklearn

from mlflow.models import infer_signature

from sklearn.datasets import load_iris

from sklearn import tree


with mlflow.start_run():

    iris = load_iris()

    sk_model = tree.DecisionTreeClassifier()

    sk_model = sk_model.fit(iris.data, iris.target)
```

```
    mlflow.log_param("criterion", sk_model.criterion)

    mlflow.log_param("splitter", sk_model.splitter)

    signature = infer_signature(iris.data, sk_model.predict(iris.data))


    mlflow.sklearn.log_model(sk_model, "sk_models", signature=signature)
```

3. Utilizando la línea de comandos de MLflow, realizar las siguientes tareas:

- Crear un fichero `input.json` con tres variables (`feature1` , `feature2` , `feature3`) y sus respectivos valores: `1` , `2` , `3` .
- Utilizando el rol de AWS SageMaker definido por la siguiente URI, obtener el valor de la predicción para las entradas anteriores a través de un *deployment* previamente creado en SageMaker, cuyo nombre es `deployment1`.

```
sagemaker:/us-west-1/arn:aws:1234567890:role/assumed_role
```

```
cat > ./input.json <<- input
{"feature1": {"0": 1}, "feature2": {"0": 2}, "feature3": {"0": 3}}
input
```

```
mlflow deployments predict \
    --target sagemaker:/us-west-1/arn:aws:1234567890:role/assumed_role \
    --name deployment1 \
```

```
    --input-path ./input.json
```

4. Utilizando el SDK de MLflow para Python, escribir el código fuente que realiza las siguientes tareas:

- Carga el CSV almacenado en la siguiente URL en un *dataframe* de Pandas: http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv
- Cargar ese mismo CSV en un PandasDataset de MLflow
- Registrar el *dataset* en MLflow con log_input() , indicando que se trata de datos de entrenamiento.
- Recuperar el PandasDataset a partir del identificador del run y extraer la siguiente información: nombre, *digest,* perfil y *schema.*

```
import mlflow.data

import pandas as pd

from mlflow.data.pandas_dataset import PandasDataset



dataset_source_url = "http://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-red.csv"

df = pd.read_csv(dataset_source_url)

dataset: PandasDataset = mlflow.data.from_pandas(df,
source=dataset_source_url)



with mlflow.start_run():
```

```
    mlflow.log_input(dataset, context="training")


run = mlflow.get_run(mlflow.last_active_run().info.run_id)

dataset_info = run.inputs.dataset_inputs[0].dataset

print(f"Name: {dataset_info.name}")

print(f"Digest: {dataset_info.digest}")

print(f"Profile: {dataset_info.profile}")

print(f"Schema: {dataset_info.schema}")
```

5. Utilizando el SDK de MLflow para Python, escribir el código fuente que realiza imprime los *runs* que cumplen las siguientes condiciones:

- El nombre del experimento al que pertenecen es experiment1 .
- Su identificador es 1234 o 5678.
- El F1 score tiene un valor inferior a 0.5.
- El nombre del modelo incluye la palabra clave GPT.

```
import mlflow


run_ids = ["1234", "5678"]

run_id_condition = "'" + "','".join(run_ids) + "'"


complex_filter = f"""
```

```
attributes.run_id IN ({run_id_condition})
  AND metrics."f1 score" < 0.5
  AND params.model LIKE "GPT%"
"""

filter = mlflow.search_runs(
    experiment_names=["experiment1"],
    filter_string=complex_filter,
)
print(filter)
```

9.8. Referencias bibliográficas

Alla, S. y Adari, S. K. (2020). *Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure.* Apress.

Azure DevOps Labs (2019, marzo 25). Creating an Azure Service Principal for use with an Azure Resource Manager service connection. https://azuredevopslabs.com/labs/devopsserver/azureserviceprincipal/

Balm, P. (2024, febrero 7). Enterprise MLOps part 2: ML Pipelines with Vertex AI. *Medium.* https://medium.com/google-cloud/machine-learning-pipeline-development-on-google-cloud-5cba36819058

Blackmist, s-polly, AbeOmor, hyoshioka0128, 19BMG00, shohei1029 y samelhousseini (2023, noviembre 3). Set up MLOps with Azure DevOps. Microsoft Learn.

GoogleCloudPlatform (s. f.). professional-services. GitHub. https://github.com/GoogleCloudPlatform/professional-services/tree/main/examples/vertex_mlops_enterprise

Hamiti, S. y Subramanian, S. (2021, enero 28). Managing your machine learning lifecycle with MLflow and Amazon SageMaker. *AWS.* https://aws.amazon.com/es/blogs/machine-learning/managing-your-machine-learning-lifecycle-with-mlflow-and-amazon-sagemaker/

https://learn.microsoft.com/en-us/azure/machine-learning/how-to-setup-mlops-azureml?view=azureml-api-2&tabs=azure-shell

Microsoft Azure, Cindyweng, setuc, iamramengirl, samelhousseini, dependabot[bot]m dhangerkapil y microsoft-github-operations[bot] (2024, julio 15). Azure - mlops-v2-ado-demo. GitHub. https://github.com/Azure/mlops-v2-ado-demo

Nardini, I. y Dugan, S. (2022, octubre 7). Vertex AI Model Registry. Google Cloud Blog. https://cloud.google.com/blog/products/ai-machine-learning/vertex-ai-model-registry

Puga, J. G. (2024a, febrero 13). Enterprise MLOps with Google Cloud Vertex AI (part 1). *Medium.* https://medium.com/google-cloud/enterprise-mlops-with-google-cloud-vertex-ai-part-1-9f36211ea66c

Puga, J. G. (2024b, febrero 13). Enterprise MLOps with Google Cloud Vertex AI part 3: CI/CD. Medium. https://medium.com/google-cloud/enterprise-mlops-with-google-cloud-vertex-ai-part-3-ci-cd-33d5e6e774a7

Schmitt, M. (s. f.). How We Track Machine Learning Experiments with MLFlow. DataRevenue. https://www.datarevenue.com/en-blog/how-we-track-machine-learning-experiments-with-mlflow

Sofianhamiti, Harishpandu43, dependabot[bot], cmosh y amazon-auto (2021). *amazon-sagemaker-mlflow-fargate.* GitHub. https://github.com/aws-samples/amazon-sagemaker-mlflow-fargate

Coursera: MLOPs platforms: Amazon SageMaker and Azure ML

Duke University, Gift, N. y Deza, A. (2023, mayo 1). Plataformas MLOps: Amazon SageMaker y Azure ML. Coursera. https://www.coursera.org/learn/mlops-aws-azure-duke. https://www.coursera.org/learn/mlops-aws-azure-duke

En este curso gratuito, elaborado por la Duke University, el alumno puede profundizar en los aspectos de operacionalización de modelos con la plataforma de IA de Amazon, AWS SageMaker.

Microsoft Learn: end-to-end machine learning operations (MLOps) with Azure Machine Learning

Microsoft Learn (s. f.). *End-to-end machine learning operations (MLOps) with Azure Machine Learning - Training.* https://learn.microsoft.com/en-us/training/paths/build-first-machine-operations-workflow/

En este curso *online* gratuito de Microsoft, el alumno puede repasar los conceptos clave del paradigma MLOps, al tiempo que profundiza en las cuestiones clave de implementación: control de versiones, automatización *(pipelines)* y CI/CD.

Test

1. ¿Cuáles son los dos aspectos clave del paradigma MLOps?

   A. Uso de MLflow y registro de modelos.

   B. Automatización de tareas repetitivas mediante *pipelines* y registro de experimentos.

   C. Registro de experimentos y ejecuciones.

   D. Operacionalización de modelos y uso de servicios *cloud.*

2. MLflow es una plataforma MLOps *open-source* que puede desplegarse:

   A. Solo en un *data center,* pero no en un proveedor de nube. Para implantar MLOps en nube, es necesario utilizar las plataformas proporcionadas por el proveedor.

   B. En un *data center* o en cualquier proveedor de nube.

   C. Solo en un entorno de computación en la nube.

   D. En un *data center* o en AWS.

3. Las principales funciones de MLflow son:

   A. Crear experimentos, registrar ejecuciones, entrenar modelos y desplegar modelos en *endpoints* para inferencia

   B. Crear experimentos, registrar ejecuciones, comparar métricas de rendimiento, registrar modelos y desplegar modelos.

   C. Crear experimentos, registrar ejecuciones, entrenar modelos y validar modelos, comparando las métricas de rendimiento entre ellos.

   D. Crear experimentos y desplegar modelos.

4. La función **mlflow.start_run()** de MLflow se utiliza para:

   A. Declarar un nuevo experimento.

   B. Señalizar el inicio de una ejecución, de manera que todas las trazas registradas queden asociadas a este run .

   C. Crear un nuevo experimento, si aún no existe.

   D. Señalizar el inicio de un proceso de registro de modelos.

5. La función **log_artifact()** de MLFlow puede utilizarse para:

   A. Registrar métricas de rendimiento con valores numéricos.

   B. Registrar imágenes generadas en el proceso de validación.

   C. Registrar modelos en formato binario.

   D. No es una función de MLflow.

6. Para desplegar un servidor de inferencias en MLflow, es necesario especificar:

   A. El identificador del experimento.

   B. El identificador del run y el nombre del modelo.

   C. El nombre del modelo.

   D. El identificador del run .

7. Para simplificar el despliegue de una arquitectura MLOps en SageMaker, se utiliza:

   A. Fargate.

   B. Cloud Development Kits.

   C. S3.

   D. RDS.

8. La función que permite registrar trazas con MLflow en SageMaker desde el código fuente es:

A. mlflow.startrun()

B. mlflow.settrackinguri()

C. mlflow.loadmodel()

D. mlflow.infersignature()

9. La plataforma de Azure DevOps puede utilizarse para desplegar la infraestructura necesaria a través de:

A. Una *service connection* y el servicio Azure Resource Manager (ARM).

B. Un *service principal,* una *service connection* y el servicio Azure Resource Manager (ARM).

C. Un *service principal* y el servicio Azure Resource Manager (ARM).

D. Un *service principal,* una *service connection* y el servicio RDS.

10. En la arquitectura MLOps de Google, Vertex Pipelines soporta de forma nativa:

A. KubeFlow Pipelines únicamente.

B. KubeFlow Pipelines y TensorFlow Extended.

C. TensorFlow Extended únicamente.

D. KubeFlow Pipelines y Keras Pipelines.