

MASTERTech

Deployment

Mikel Cañizo

Abril 2022

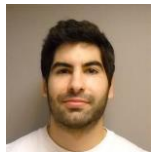
LA TECNOLOGÍA,
NUESTRA
ACTITUD

...

ikerlan
MEMBER OF BASQUE RESEARCH
& TECHNOLOGY ALLIANCE



@IKERLANofficial



Mikel Cañizo
Zubizarreta

Investigador del equipo de **Data Analytics e Inteligencia Artificial** en

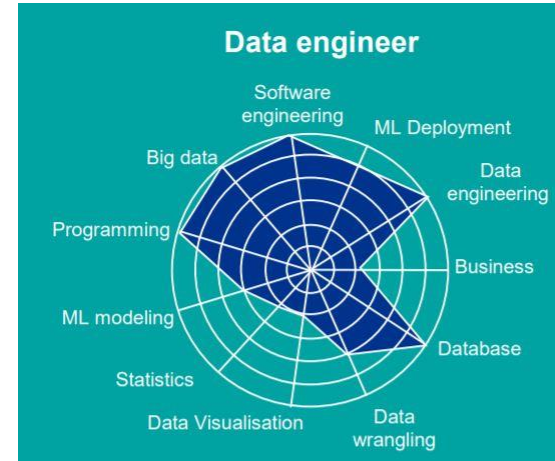
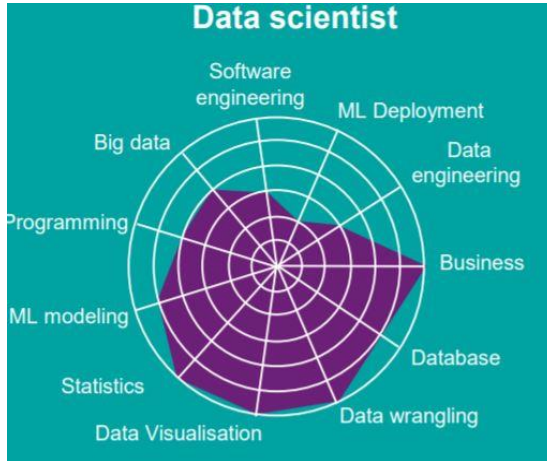
*Mantenimiento
Predictivo*

*Analítica
Avanzada*

*Monitorización
en Tiempo Real*

Deep Learning

*Plataformas
Digitales*



Híbrido entre científico de datos e ingeniero de datos

Plataformas digitales:

- *Herramientas y servicios para el envío, procesamiento y almacenamiento de los datos*
- *Big Data*

Preprocesado:

- Procesado de los datos crudos
- Preparación de los datos para generar modelos IA

Visualización:

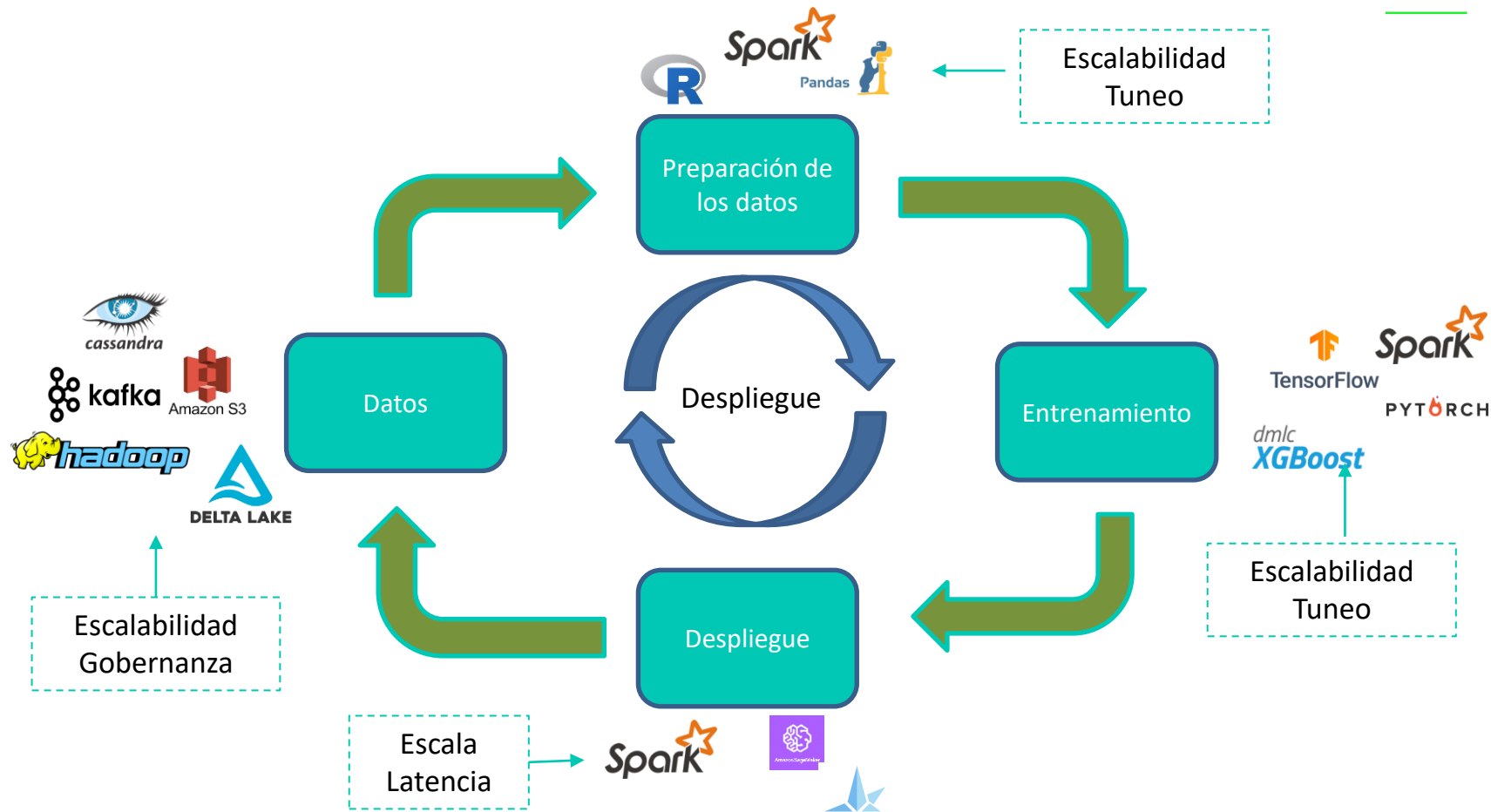
- Visualización de los datos
- ¿Qué forma tienen los datos?

Análisis y validación:

- Tipos de modelos de IA
- Generación de modelos (detección de anomalías)

¿¿Despliegue??

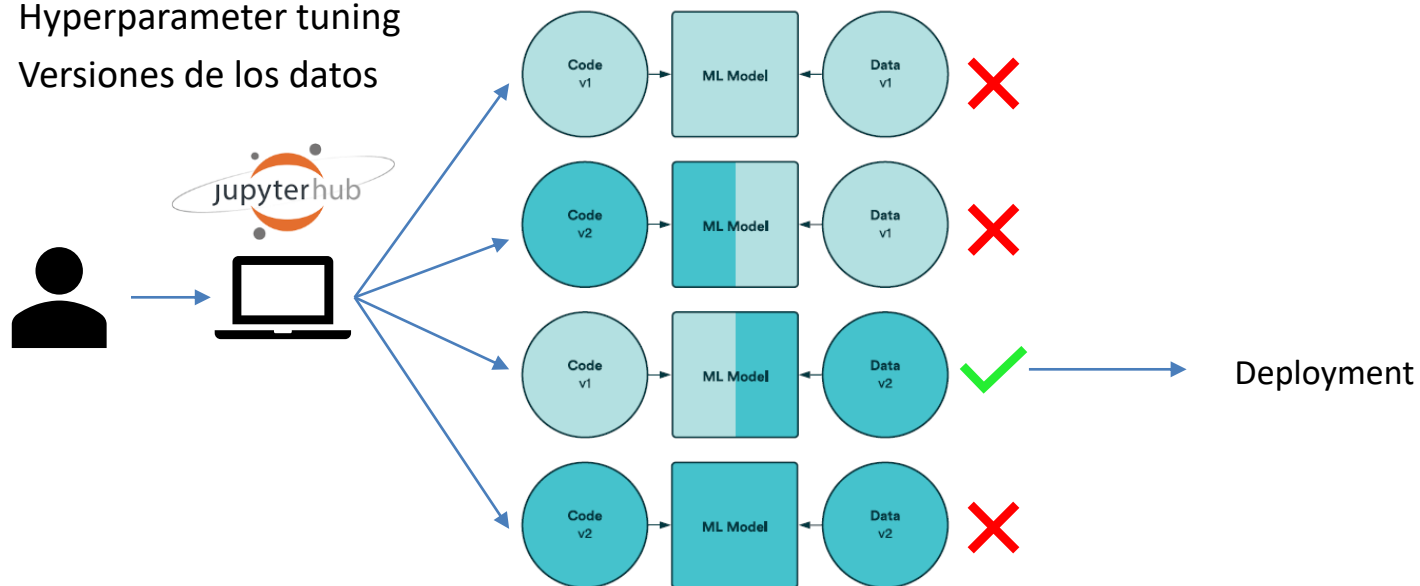
Qué es el despliegue?



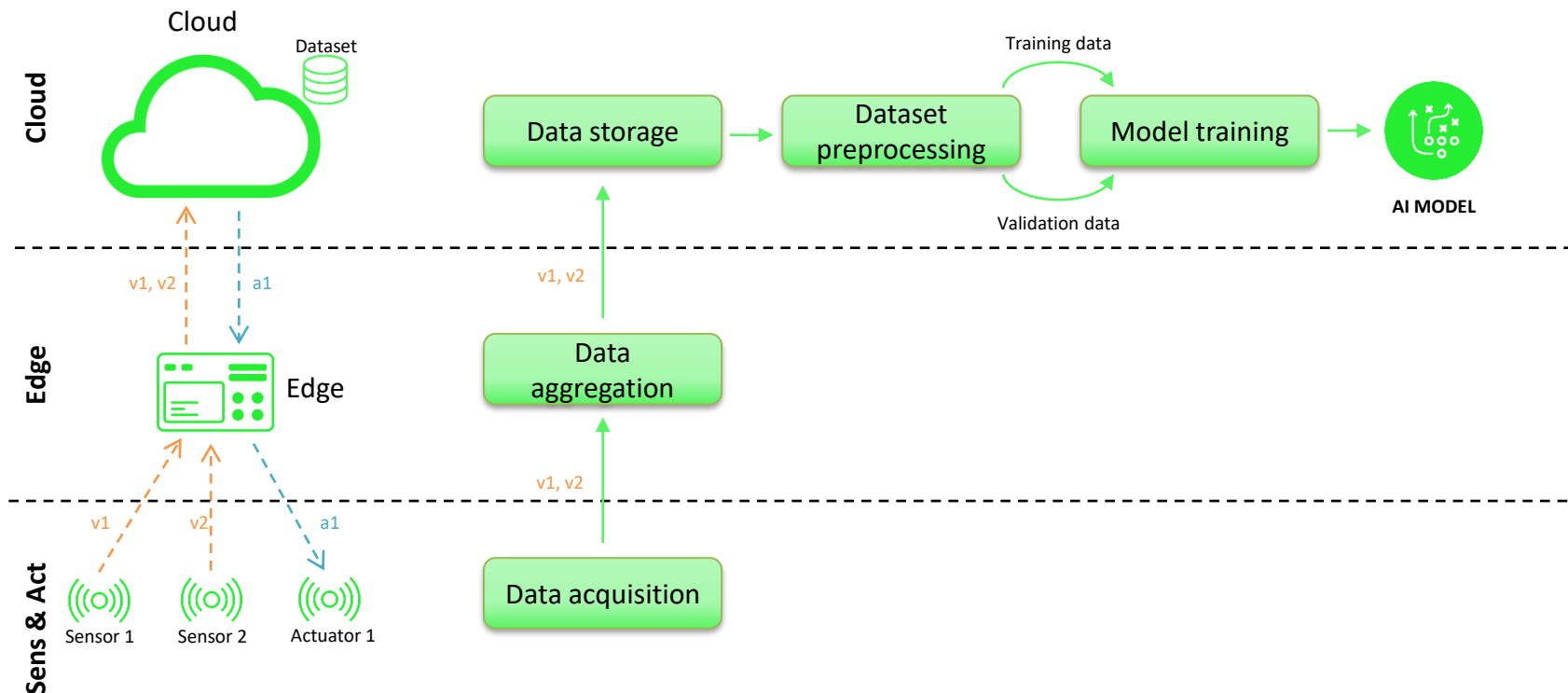
- ¿Es la **realidad** así de simple?

¿Qué versión del modelo desplegamos?

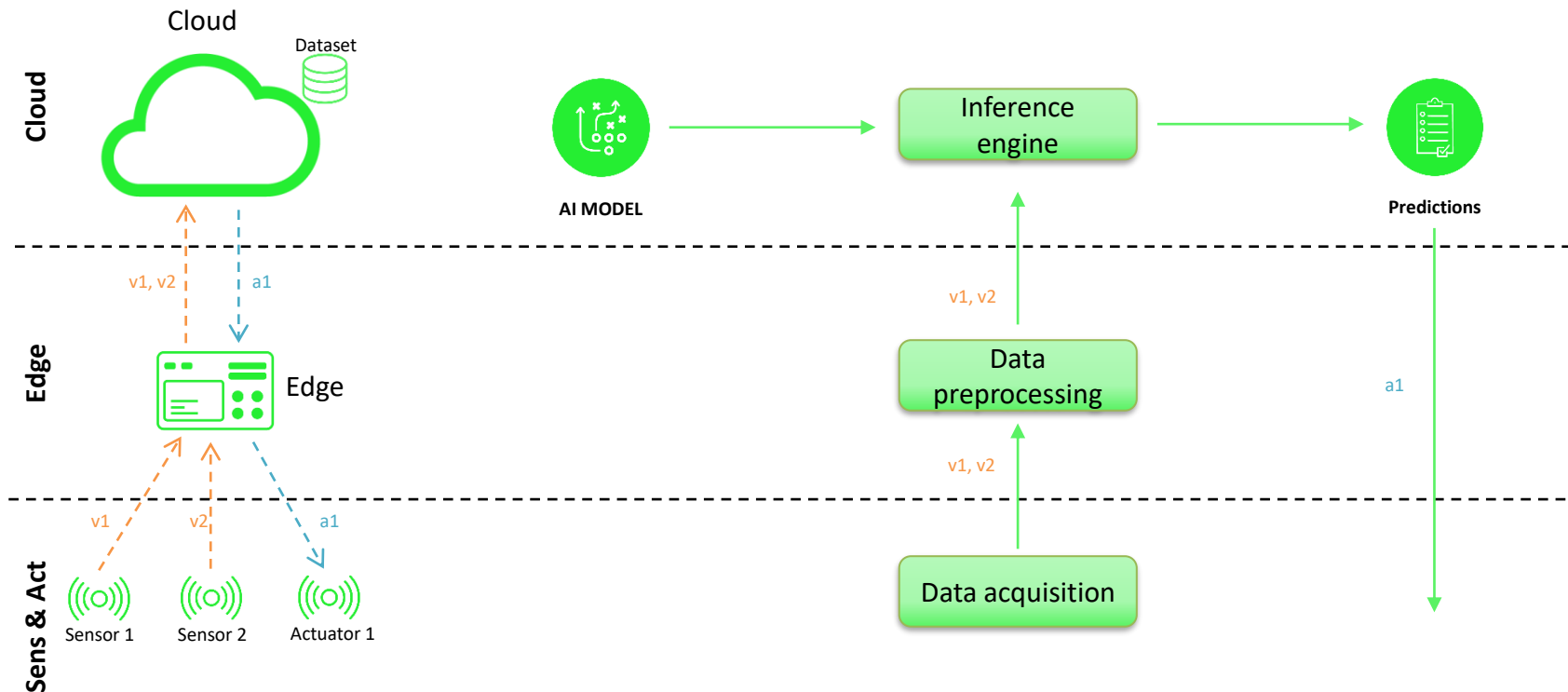
- Preprocesamientos diferentes
- Feature engineering
- Hyperparameter tuning
- Versiones de los datos



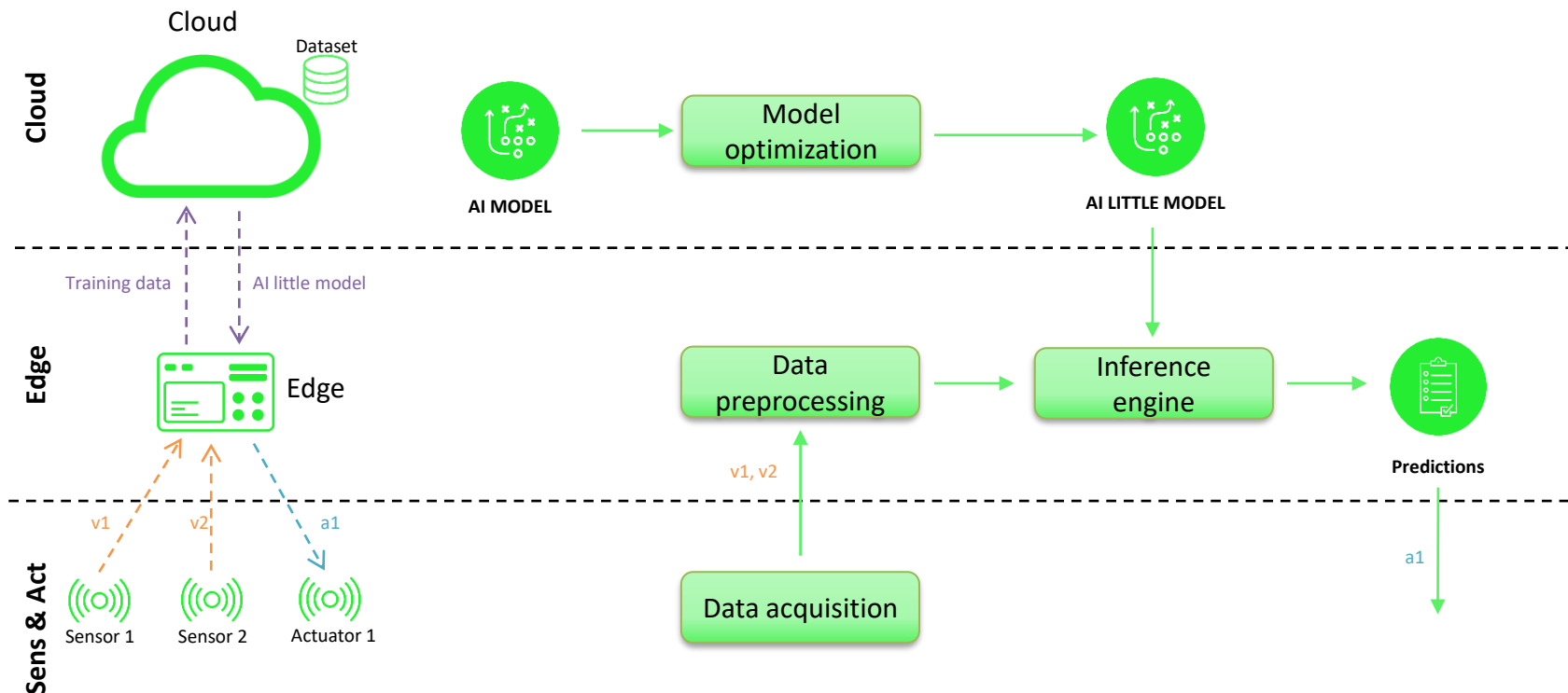
- ¿ **Dónde** despliegue el modelo?



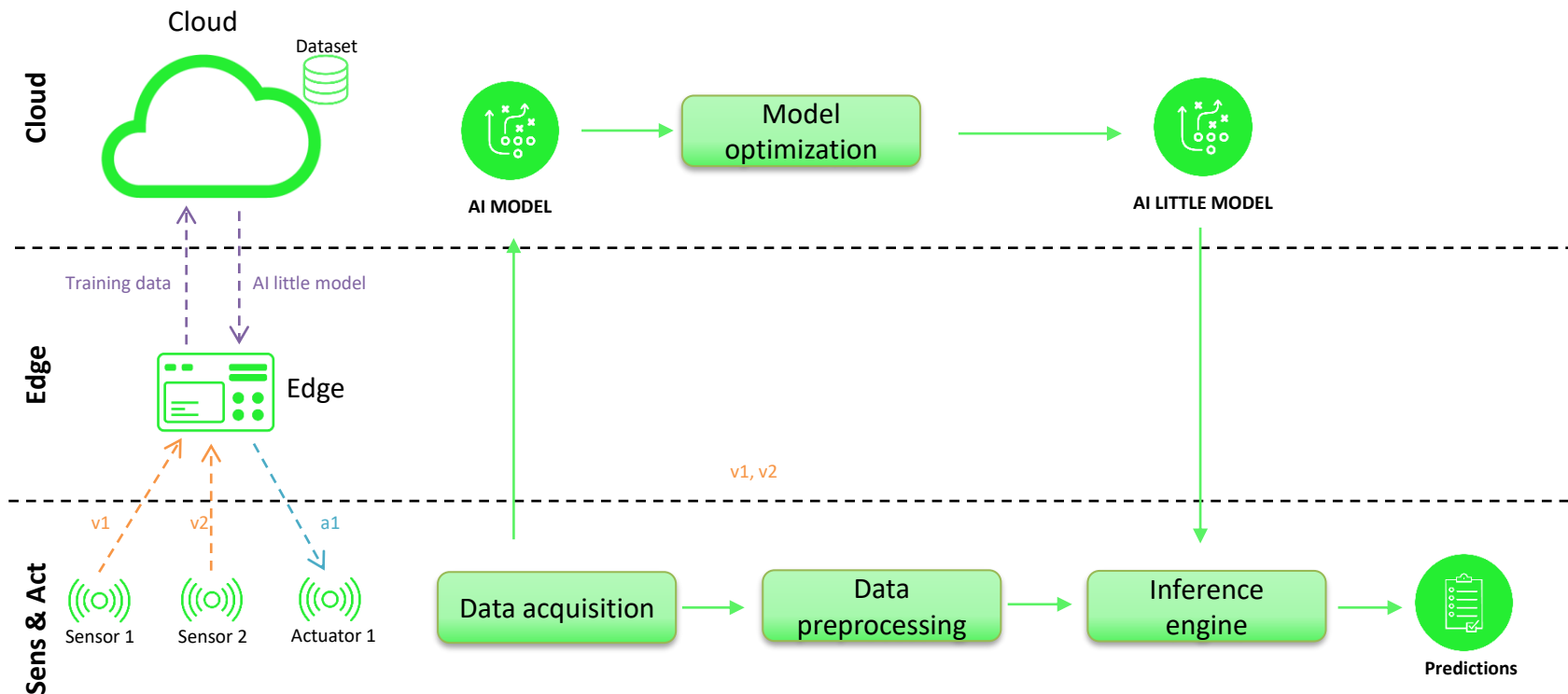
- ¿ **Dónde** despliego el modelo?



- ¿ **Dónde** despliego el modelo?

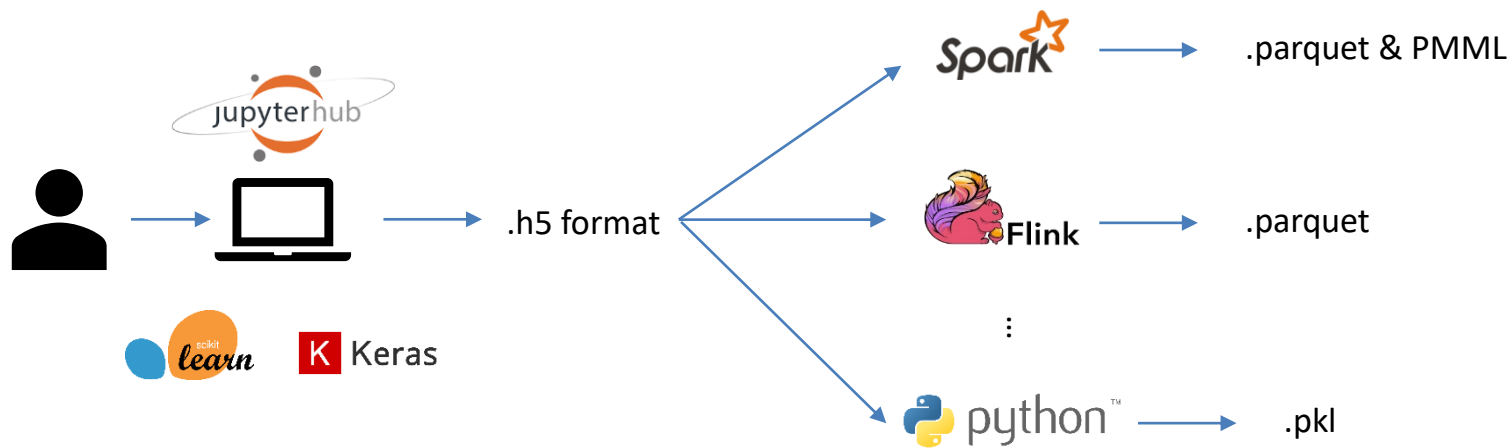


- ¿ **Dónde** despliego el modelo?



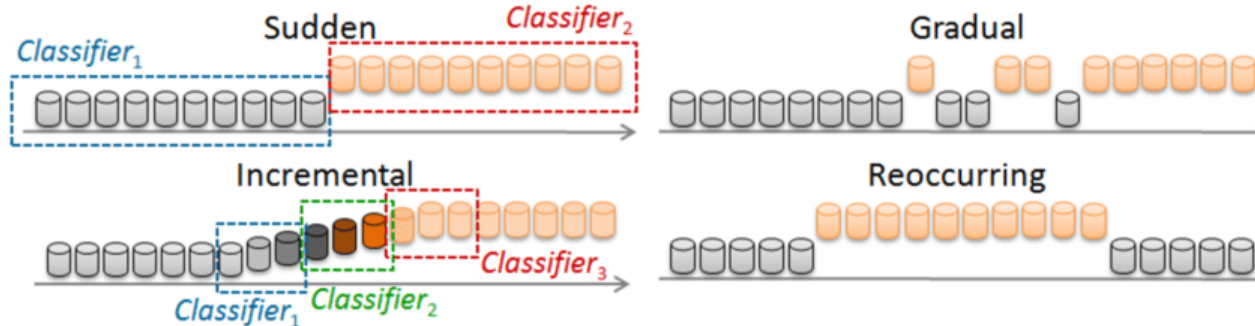
¿En qué formato desplegamos el modelo?

- No todas las herramientas usan los mismos formatos
- Diferentes herramientas de entrenamiento y producción
- Limitaciones de memoria (embebidos, móviles...)



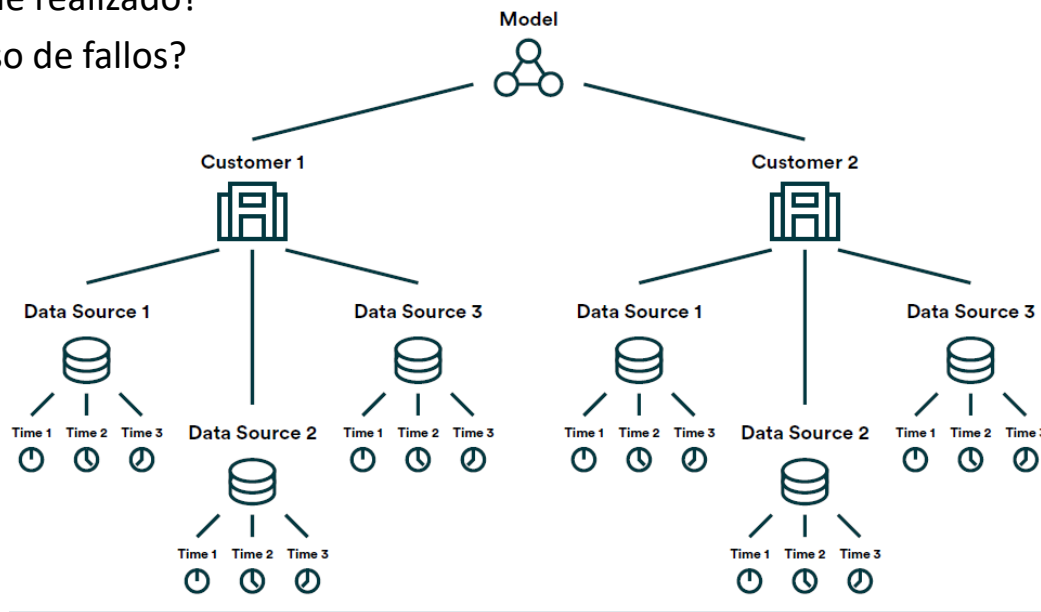
¿Cuándo despliego el nuevo modelo?

- **Evolución de los sistemas** monitorizados
- **Monitorización** de los modelos
- **Concept drift**
- ¿Cuándo **re-entrenar** el modelo?

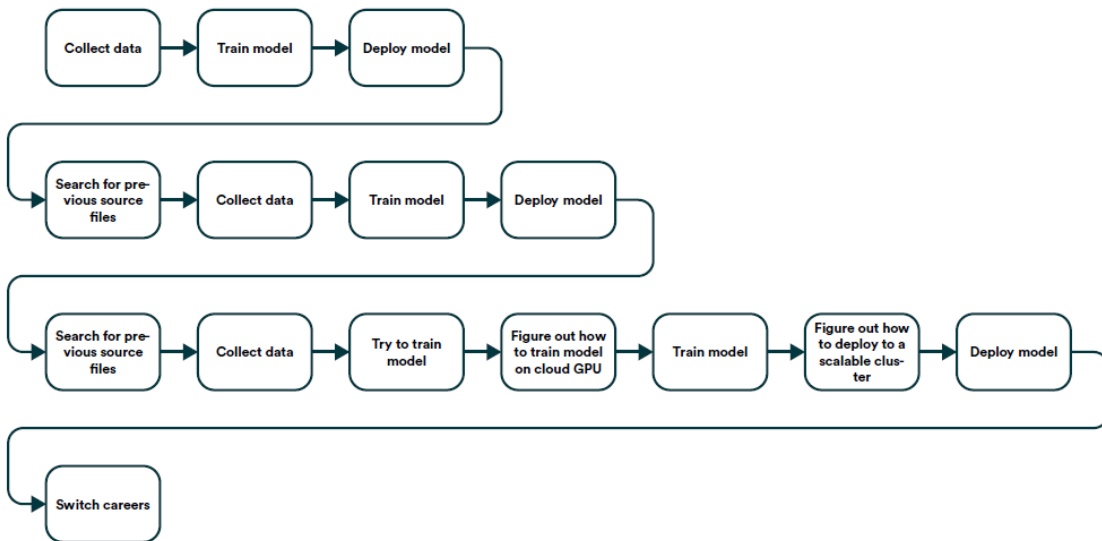


¿Cómo llevo la **trazabilidad** del modelo?

- **Múltiples versiones** de los modelos (hiperparámetros, datos...)
- ¿**Cómo saber qué pruebas** he realizado?
- ¿**Cómo volver a atrás** en caso de fallos?
- **Versiones** de los datos

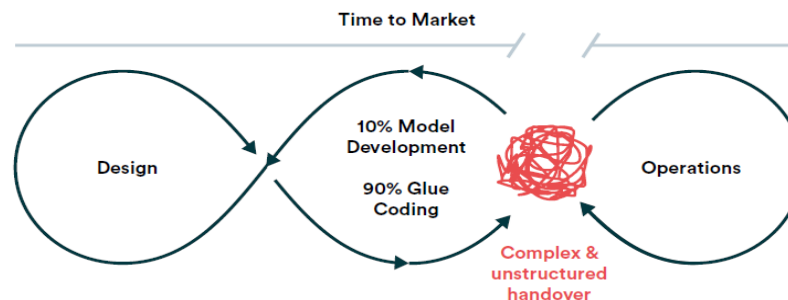


18 Trained Models

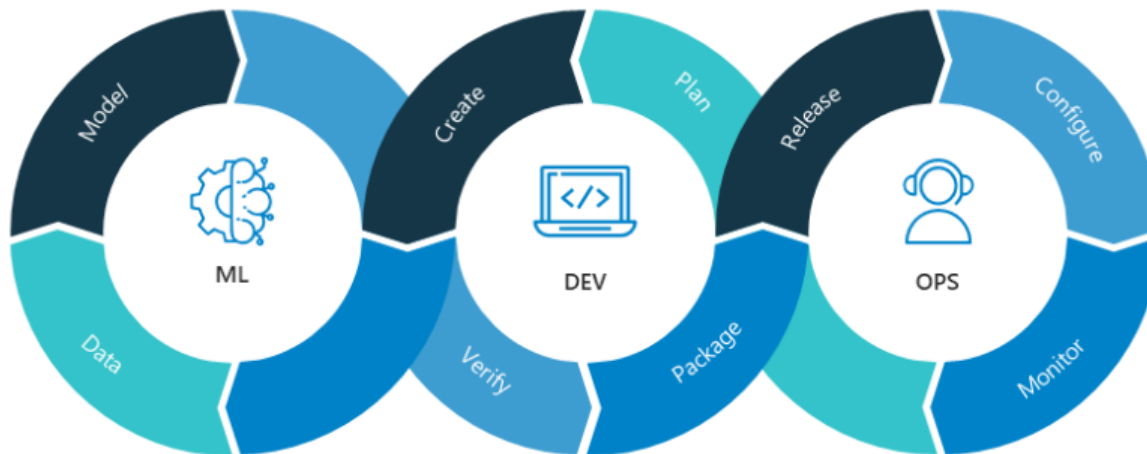


¿Cómo llevar a producción todo esto?

- ¿Cómo puedo registrar un histórico de modelos, configuraciones y pruebas?
- ¿Cómo pongo en producción los modelos?



MLOps = ML + DEV + OPS



Experiment:

*Data acquisition
Business understanding
Initial modeling*

Develop:

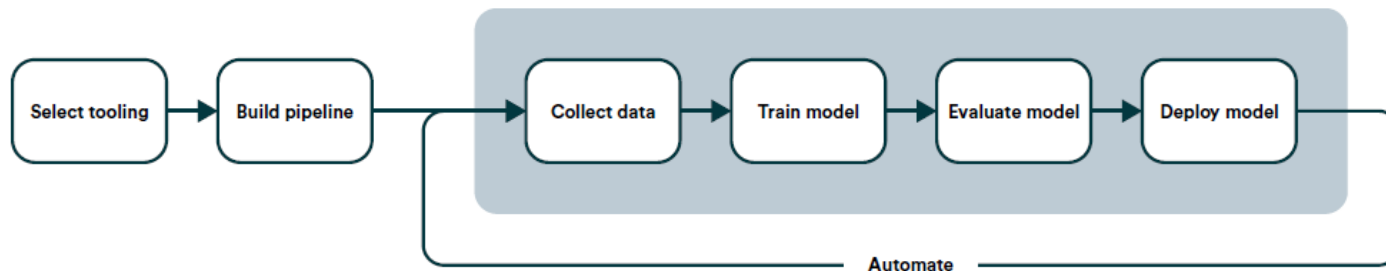
*Modeling + Testing
Continuous integration
Continuous deployment*

Operate:

Continuous delivery
*Data Feedback Loop
System + Model monitoring*

Automatización del ciclo de vida de los modelos

- Seleccionar las herramientas para todo el ciclo de vida
- Generar pipelines
- Automatización de toda la fase de desarrollo + validación + despliegue



Trazabilidad de los modelos

Reproducibilidad

Control del versiones

Conexión entre Data Scientist y Data Engineer

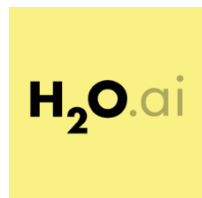
Ciclo de vida **automatizado** (menos errores humanos)

- ¿Cómo aplico el **MLOps** al mundo real?

En la actualidad existen varias plataformas



AWS MLOps

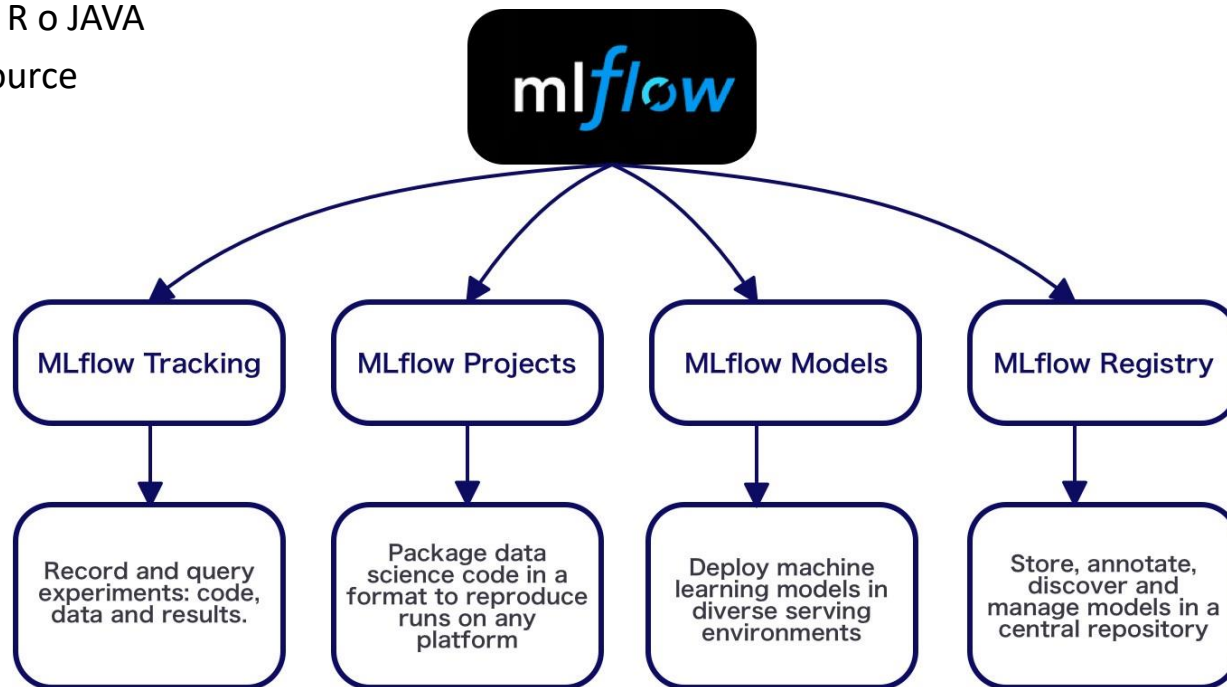


Microsoft Machine Learning for Apache Spark



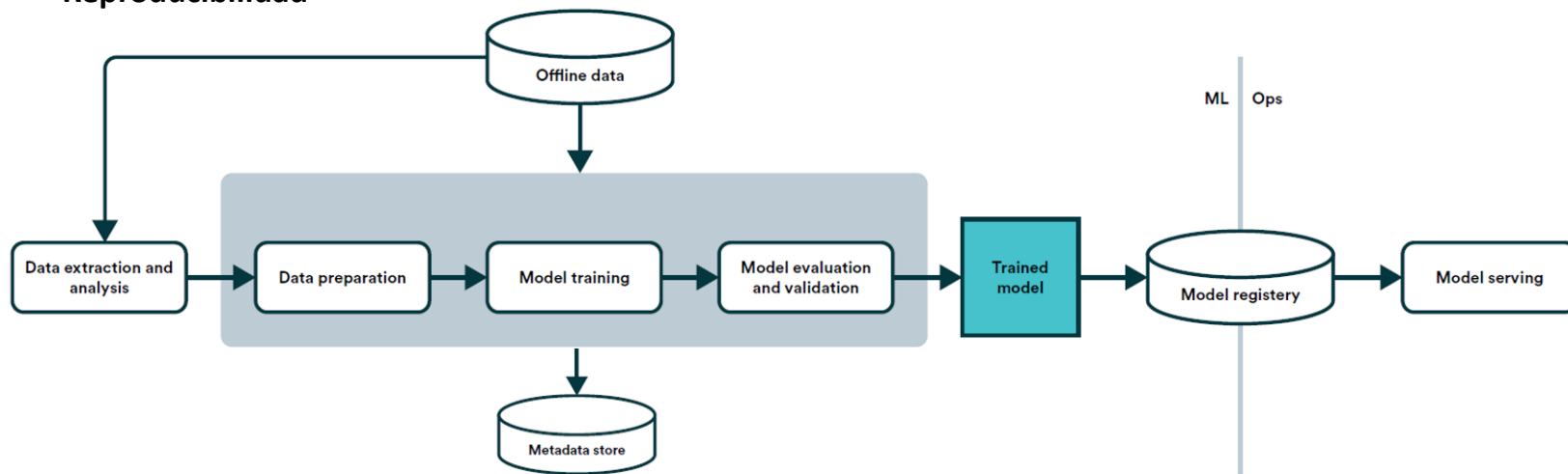
Es una **plataforma end-to-end** para la **gestión del ciclo de vida** de los modelos IA

- Integración con cualquier librería (scikit-learn, TF, keras, Pytorch, Spark, Flink...)
- Python, R o JAVA
- Open source



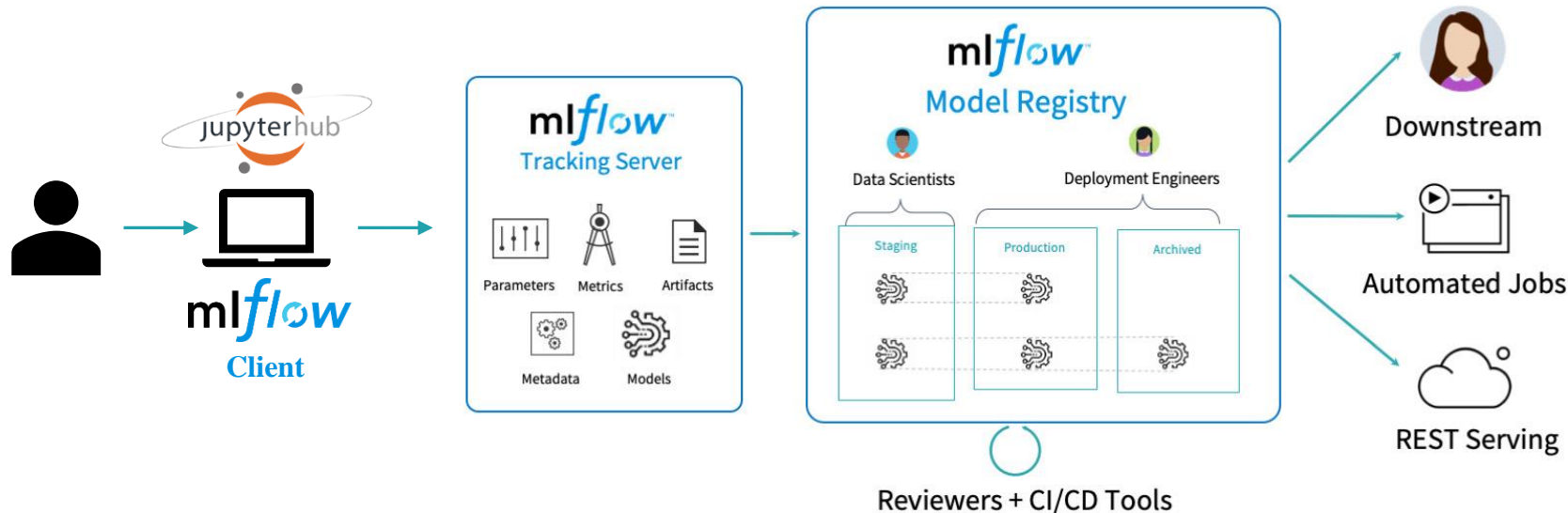
Esta herramienta facilita la gestión del ciclo de vida de los modelos

- Se **almacena cada paso** durante la **fase de desarrollo** (metadata store)
- Se **registran** todos los **modelos** (model registry)
- **Mapeo** prueba -> modelo
- Facilita el **despliegue**
- **Reproducibilidad**

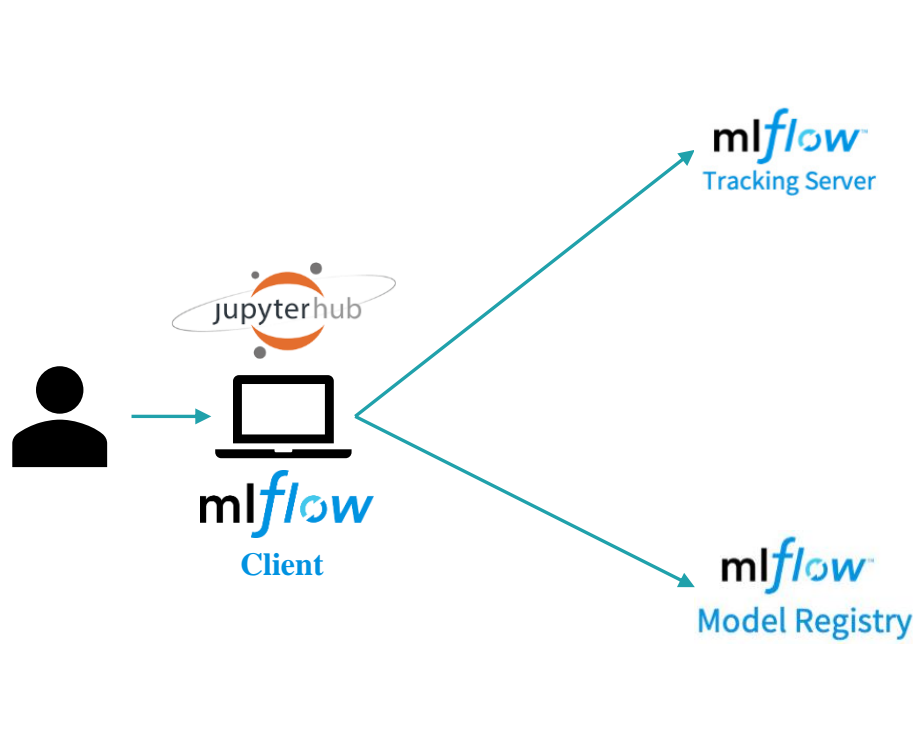


Mlflow dispone de un servidor para hacer el tracking

- Los usuarios se pueden **conectar** de forma **remota al servidor**
- Es **necesario** tener instalado **MLflow** en el lado **cliente**
- Los **metadatos** y **modelos** se **guardan** en el **servidor**



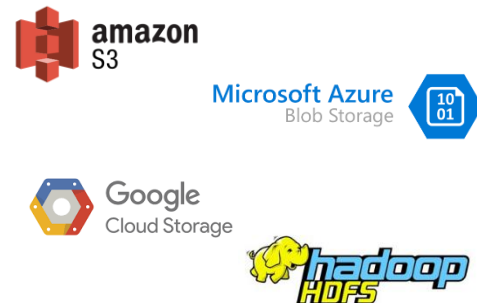
MLflow tiene **integración con múltiples BBDD y repositorios**



- Fichero local
- SQLite
- MySQL
- MSSQL
- PostgreSQL



- Amazon S3
- Azure Blob Storage
- Google Cloud Storage
- Servidor FTP o SFTP
- NFS
- HDFS



Estructura:

- **Experimento**: se refiere a un **proyecto común** que engloba las pruebas realizadas para generar un modelo para un caso de uso específico

```
mlflow.create_experiment(name, artifact_location=None) [source]
```

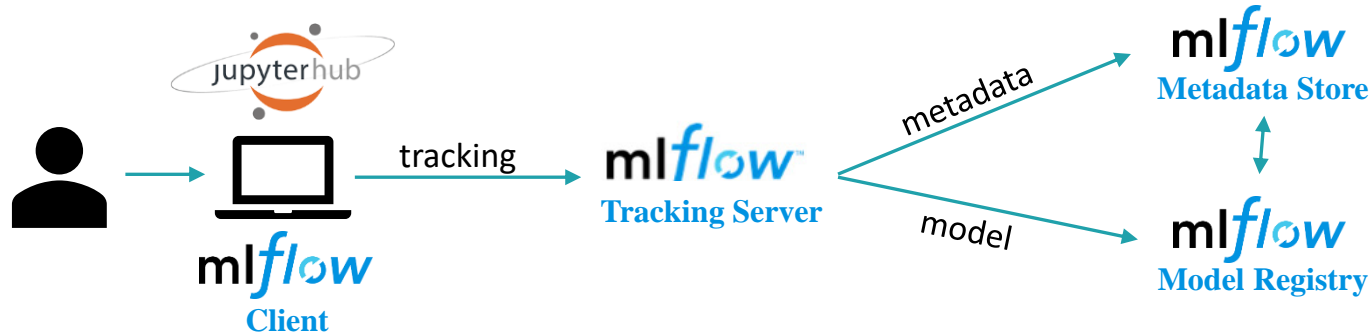
- **RUN**: se refiere a una prueba o ejecución de entrenamiento que se realizan dentro de un experimento.
- **Parámetros**: hiperparámetros del modelo. Cada RUN tiene uno o varios parámetros asociados, pero para una ejecución en concreto cada parámetro solo puede tener un valor.
- **Métricas**: métricas de evaluación del modelo. Cada RUN tiene una o varias métricas asociadas, pero para tener un valor.

```
with mlflow.start_run():  
    mlflow.log_param("x", 1)  
    mlflow.log_metric("y", 2)  
    ...
```

Registra todos los metadatos de la prueba

Cada prueba se organiza bajo el concepto ***RUN***:

- model metadata
- **Code version**: versión del código (si está en un repositorio Git se registra el hash del commit).
 - **Start & end time**: cuándo se ha iniciado y finalizado el run.
 - **Source**: nombre del fichero donde se encuentra el código fuente o el nombre del proyecto.
 - **Parameters**: registra en formato clave/valor los parámetros utilizados para entrenar el modelo.
 - **Metrics**: registra en formato clave/valor las métricas definidas para evaluar el rendimiento.
 - **Artifacts**: registra el modelo entrenado en el formato especificado (artifact = modelo)



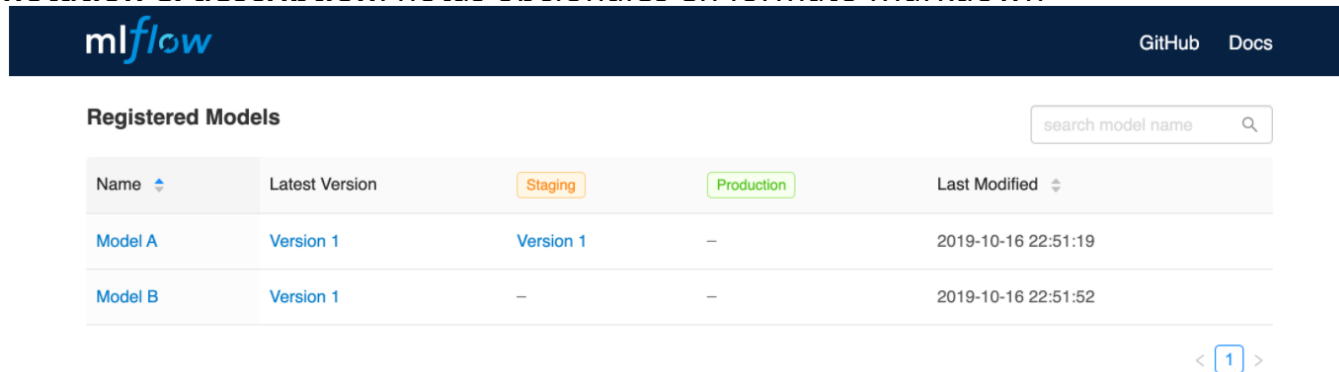
Interfaz gráfica (<http://<ip-servidor>:5000>)

<input type="checkbox"/>	Date ▼	User	Source	Version	Parameters		Metrics		
					alpha	lambda	mae	r2	rmse
<input type="checkbox"/>	2018-08-30 15:42:55	mlflow	R:train.R	da3f0a	1	1	0.638	0.03	0.857
<input type="checkbox"/>	2018-08-30 15:42:50	mlflow	R:train.R	da3f0a	1	0.5	0.639	0.039	0.853
<input type="checkbox"/>	2018-08-30 15:42:45	mlflow	R:train.R	da3f0a	1	0.2	0.617	0.153	0.804
<input type="checkbox"/>	2018-08-30 15:42:40	mlflow	R:train.R	da3f0a	1	0	0.597	0.224	0.77
<input type="checkbox"/>	2018-08-30 15:42:35	mlflow	R:train.R	da3f0a	0.5	1	0.639	0.039	0.853
<input type="checkbox"/>	2018-08-30 15:42:30	mlflow	R:train.R	da3f0a	0.5	0.5	0.621	0.125	0.818
<input type="checkbox"/>	2018-08-30 15:42:26	mlflow	R:train.R	da3f0a	0.5	0.2	0.616	0.169	0.794
<input type="checkbox"/>	2018-08-30 15:42:21	mlflow	R:train.R	da3f0a	0.5	0	0.597	0.224	0.77
<input type="checkbox"/>	2018-08-30 15:42:15	mlflow	R:train.R	da3f0a	0	1	0.617	0.158	0.801
<input type="checkbox"/>	2018-08-30 15:42:09	mlflow	R:train.R	da3f0a	0	0.5	0.617	0.171	0.793
<input type="checkbox"/>	2018-08-30 15:42:04	mlflow	R:train.R	da3f0a	0	0.2	0.618	0.178	0.788
<input type="checkbox"/>	2018-08-30 15:41:50	mlflow	R:train.R	da3f0a	0	0	0.597	0.224	0.77

Registra los modelos de cada prueba

Cada prueba se organiza bajo el concepto ***RUN***:

- **Model**: modelo generado a partir de un experimento o run
- **Registered modelo**: modelo registrado con un identificador único que tiene asociado la versión, el estado, la trazabilidad y otros metadatos
- **Model version**: versión del modelo. Versión incremental automática
- **Model Stage**: el estado del modelo (staging, production, archived, etc.)
- **Annotation & description**: notas opcionales en formato Markdown



The screenshot shows the mlflow Model Registry interface. At the top is the mlflow logo and navigation links for GitHub and Docs. Below is a section titled 'Registered Models' with a search bar. A table lists the registered models with columns for Name, Latest Version, Staging, Production, and Last Modified. The table contains two entries: Model A and Model B, both at Version 1. Model A is in the Staging stage, while Model B is in the Production stage. The last modified times are 2019-10-16 22:51:19 for Model A and 2019-10-16 22:51:52 for Model B. A pagination control at the bottom right shows page 1 of 1.

Name	Latest Version	Staging	Production	Last Modified
Model A	Version 1	Version 1	—	2019-10-16 22:51:19
Model B	Version 1	—	—	2019-10-16 22:51:52

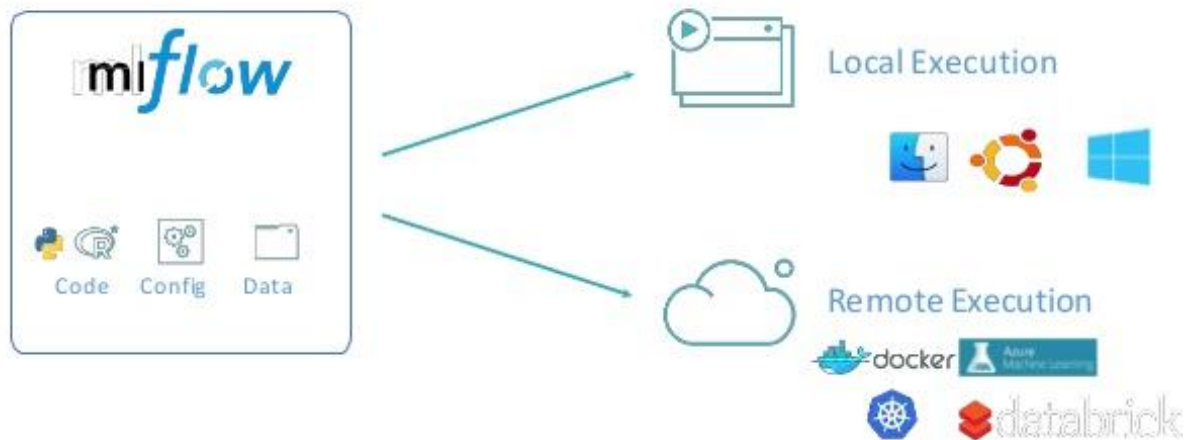
Empaqueta el proyecto en un formato reproducible en cualquier plataforma

- Incluye dependencias de librerías
- Incluye el código
- Define los parámetros de entrada (opcional)

*Un **proyecto MLflow** contiene lo siguiente (MLproject file):*

- **Name:** nombre del proyecto
- **Environment:** entorno de ejecución
 - **Conda:** entorno de ejecución generado con Conda (*conda.yaml*)
 - **Docker:** entorno de ejecución generado mediante Docker (*Dockerfile*)
- **Entry point:** punto de entrada del proyecto o comandos para ejecutar el proyecto
 - Puede tener varios puntos de entrada
 - Puede ser un **.py** o un **.sh**

MLFlow Projects



Ejemplo

```
name: My Project

conda_env: my_env.yaml
# Can have a docker_env instead of a conda_env, e.g.
# docker_env:
#   image: mlflow-docker-example

entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization} {data_file}"
  validate:
    parameters:
      data_file: path
    command: "python validate.py {data_file}"
```

Guarda el modelo en un formato estándar y es capaz de servirlo

Cada MLflow Model es un directorio que contiene ficheros arbitrarios junto con un fichero llamado ***MLmodel***

```
# Directory written by mlflow.sklearn.save_model(model, "my_model")
my_model/
├── MLmodel
└── model.pkl
```

El fichero MLmodel contiene los flavors (formatos) en los que se puede ejecutar el modelo (soporte)

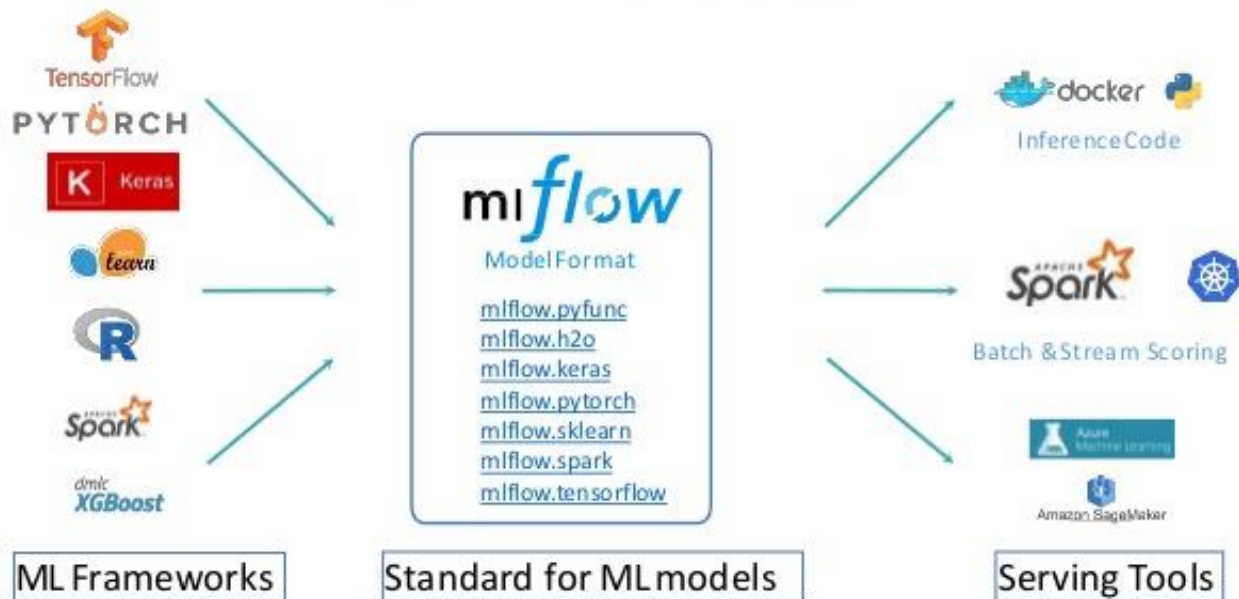
```
time_created: 2018-05-25T17:28:53.35

flavors:
  sklearn:
    sklearn_version: 0.19.1
    pickled_model: model.pkl
  python_function:
    loader_module: mlflow.sklearn
```


¿Qué es un **flavor**?

- Es una **convención** para que las herramientas de despliegue **entiendan cómo pueden usar el modelo**

MLflow Models

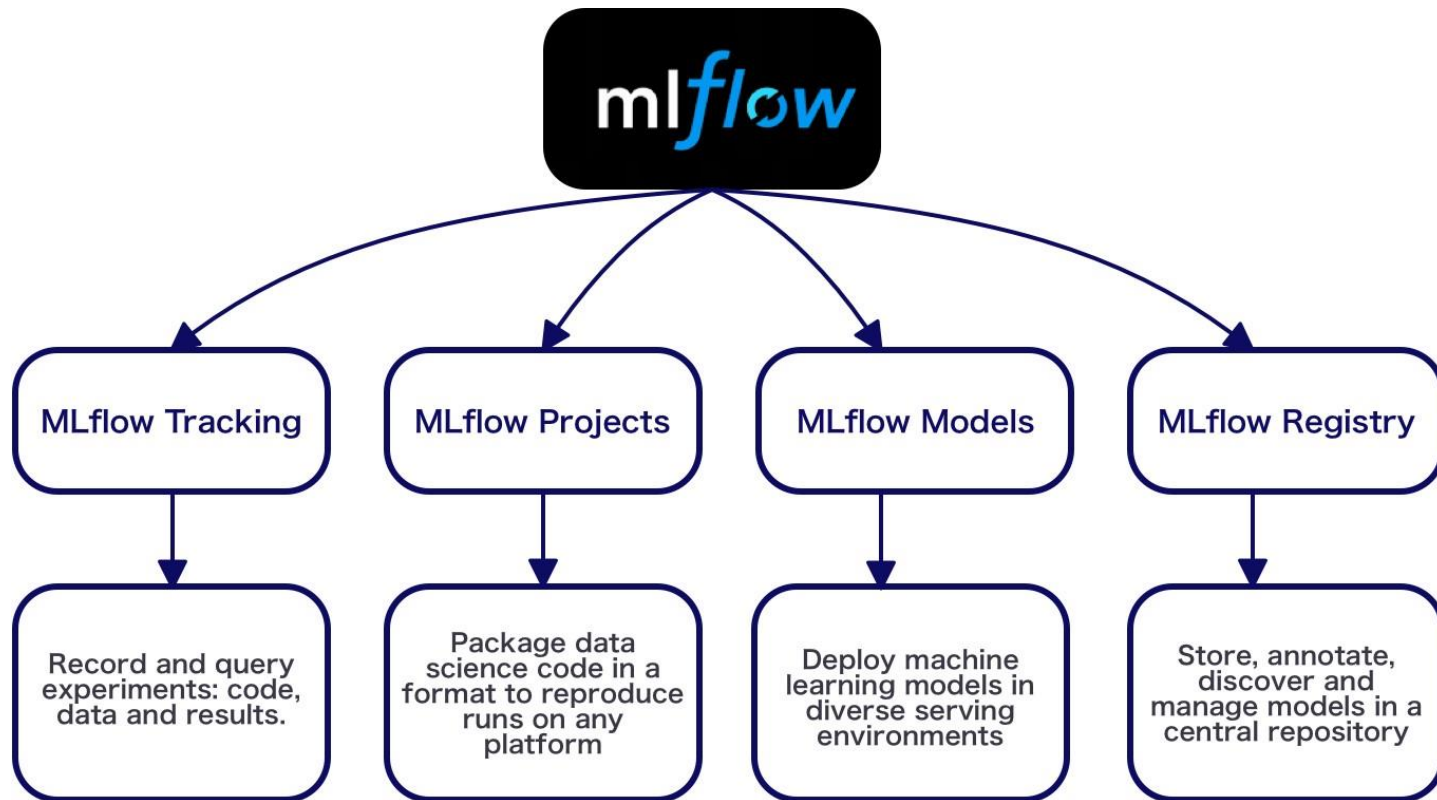


Model Signature

- Define el **esquema** de los **parámetros** de **entrada** y de **salida** del modelo
 - Nombre
 - Tipos de datos
- **Fuerza** que el modelo se ejecute con el **esquema** definido
 - **Genera excepciones** si las **variables** de entrada **no** son las **correctas**
 - O el **orden** de las variables **no es el correcto**

```
signature:  
  inputs: '[{"name": "sepal length (cm)", "type": "double"}, {"name": "sepal width  
          (cm)", "type": "double"}, {"name": "petal length (cm)", "type": "double"}, {"name":  
          "petal width (cm)", "type": "double"}]'  
  outputs: '[{"type": "integer"}]'
```

Resumen



¿Cómo automatizar esto?

