

Iniciamos  
6:10 am

::Gracias::

Mentor: Jonnatan Arias  
Garcia



# Machine Learning Operations iii



by Jonnatan Arias Garcia  
13/09/2024

# Contenido I

## Modulo 1: Introducción MLOps

### MLOps

- MLOps y Devs
- Importancia MLOps
- Ciclo de vida MLOps

### Entorno de desarrollo

- Google Colab
- Configuración de Python para MLOps
- Control de Versiones Git y Github

## Modulo 2: Fundamentos de Machine Learning

### ML básico

- Aprendizaje Supervisado – No Supervisado
- Evaluación de Modelos: Accuracy, precisión recall, F1 score

### Gestión de Datasets

- Preprocesamiento
- Librerías Pandas y Numpy

## Módulo 3: Automatización de procesos

### Pipeline de ML

- Definición y usos
- Pipelines en Scikit-learn
- Implementación en Colab

### Control de Versiones de datos

- DVC colab
- Versionado de Datasets.

## Módulo 4. Modelos en Producción

### Modelos Reproducibles

- Reproducibilidad y Semilla
- Tracking con MLflow
- MLflow en colab

### Modelos y entrenamiento

- Modelos con tensorflow y keras
- Guardado y Exportación de modelo entrenado

# Contenido II

# Módulo 5. Pruebas y Monitoreo de Modelo

## Pruebas

- Validación Cruzada
  - Pruebas unitarias y de integración de pipelines en M
  - Pytest

## Monitored

- Introducción al monitoreo de modelo
  - Herramientas (Prometheus, Grafana)
  - Implementación de Alertas Básicas

## Módulo 6. Gestión de Recursos

# Optimización de Recursos

- CPU, GPU, TPU
  - Uso eficiente y paralelaje

# Optimización de Parámetro

- ### ○ GridSearch y RandomSearch

## Módulo 7. Despliegue en la Nube

# Cloud AI

- Cloud Computing
    - Integración CI/CD
      - CI/CD: Integración continua + entrega e implementación continua
      - Github Actions

# Módulo 8. Documentación y Buenas Prácticas

## Herramientas

- Documentación de pipelines, modelos y librerías
    - Recomendaciones, Seguridad y privacidad

## Buenas Prácticas



# Modulo VI

Optimización de Parámetros e  
Hiperparámetros  
&  
Gestión de Recursos



# Gestión de Recursos en Recursos en Python y y Google Colab

Herramientas para optimizar tus recursos en Python y Google Colab, maximizando el rendimiento y la eficiencia de tus proyectos de aprendizaje automático.

# Optimización de Recursos

## 1 Optimización de Hardware

### Hardware

Seleccionar el hardware adecuado adecuado para tu proyecto es fundamental para obtener un rendimiento óptimo.

## 3 Optimización del Código

Implementar prácticas de codificación eficientes, como la vectorización y la optimización de bucles, puede mejorar significativamente el rendimiento.

## 2 Administración de Memoria

### Memoria

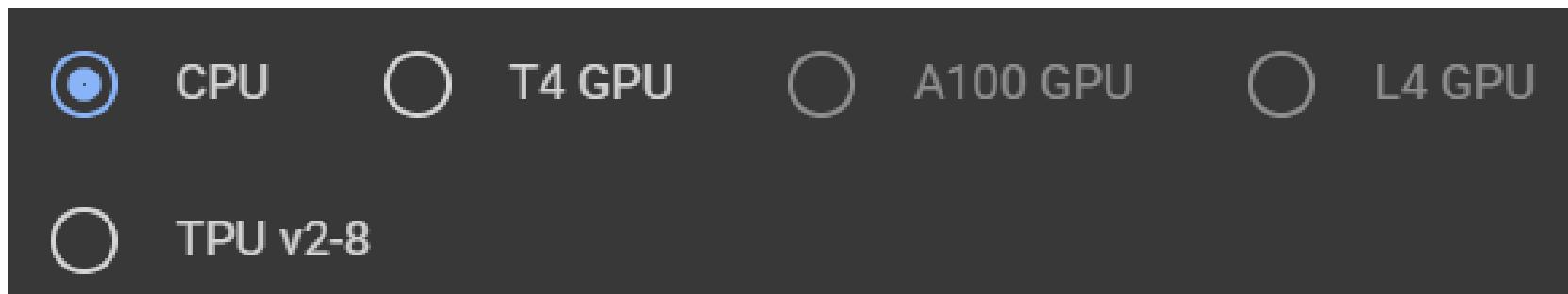
Controlar el uso de la memoria es crucial para evitar errores de memoria y garantizar un flujo de trabajo sin problemas.

## 4 Paralelaje

Utilizar la capacidad de procesamiento paralelo de múltiples núcleos de CPU, GPU o TPU para acelerar las tareas computacionales intensivas.



# CPU, GPU y TPU



## CPU

Procesador central de procesamiento, ideal para tareas generales y computación secuencial.

**Por defecto en colab.**

## GPU

Unidad de procesamiento gráfico, diseñada para tareas intensivas en gráficos y computación paralela.

## TPU

Unidad de procesamiento tensorial, específicamente diseñada para tareas de aprendizaje automático, con un alto rendimiento y eficiencia.

La selección de procesador se puede cambiar desde la pestaña de entorno de ejecución, y el uso de los diferentes núcleos debe ir fijado en el Código y la librería

```
import tensorflow as tf  
print("GPUs disponibles:", len(tf.config.list_physical_devices('GPU')))
```

# CPU(por defecto), GPU(Esta disponible?) y TPU (Configurar)

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Detectar TPU
resolver = tf.distribute.cluster_resolver.TPUClusterResolver() # Detectar TPU
tf.config.experimental_connect_to_cluster(resolver)
tf.tpu.experimental.initialize_tpu_system(resolver)
strategy = tf.distribute.TPUStrategy(resolver) # Crear la estrategia de distribución

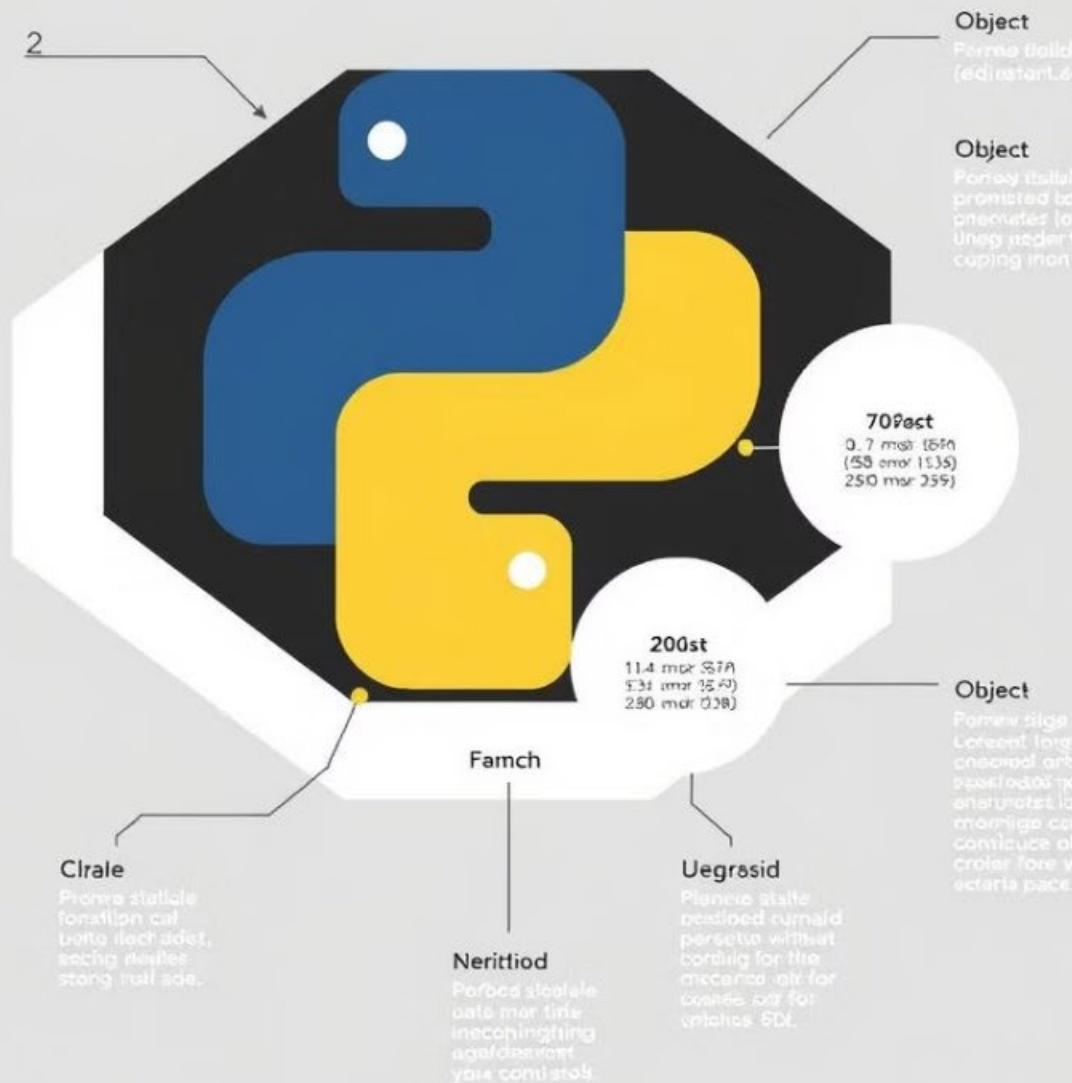
# Definir y compilar el modelo dentro de la estrategia
with strategy.scope():
    model = models.Sequential([
        layers.Dense(64, activation='relu', input_shape=(32,)),
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

# Entrenar el modelo en TPU
model.fit(x_train, y_train, epochs=5)

# Verificar si hay GPUs disponibles
print("¿GPU disponible?:", tf.test.is_gpu_available())
```

Nota: Lo común es que las librerías tengan manejo propio frente al uso de GPU/TPU

e memoriations gr times wder cussion ansuel my comertage plly amotion of the compamt  
he rd sgatting of alletonic in optimizes.



# Manejo de Memoria y Almacenamiento

## 1 Uso de Profiling

Utilizar herramientas de profiling para analizar el uso de memoria y identificar áreas de alto consumo.

## 2 Gestión de Objetos

Implementar técnicas de gestión de objetos, como el recolector de basura, para liberar memoria utilizada por objetos que ya no son necesarios.

## 3 Compresión de Datos

Utilizar técnicas de compresión de datos para reducir el tamaño de los archivos y optimizar el uso de memoria.

## 4 Optimización de Almacenamiento

Elegir un sistema de almacenamiento adecuado para el proyecto, considerando la velocidad, el costo y la seguridad.

# Ram y VRam

```
!nvidia-smi
```

NVIDIA-SMI 455.45.01			Driver Version: 455.45.01		CUDA Version: 11.1		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	
					MIG M.	Enabled	
0	Tesla K80		off	00000000:00:04.0	off		0
N/A	39C	P8	27W / 149W	150MiB / 11441MiB	0%	Default	

```
import psutil

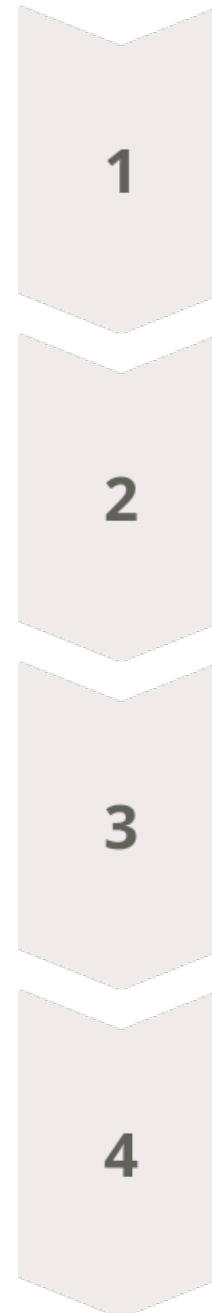
# Información sobre el uso de la memoria RAM
ram = psutil.virtual_memory()
print(f"RAM total: {ram.total / 1024**3:.2f} GB")
print(f"RAM disponible: {ram.available / 1024**3:.2f} GB")
print(f"RAM utilizada: {ram.used / 1024**3:.2f} GB")
print(f"Porcentaje de uso de RAM: {ram.percent}%")
```

Reducir el uso de RAM (jugar con procesamientos por lotes y liberar variables grandes o reusarlas)

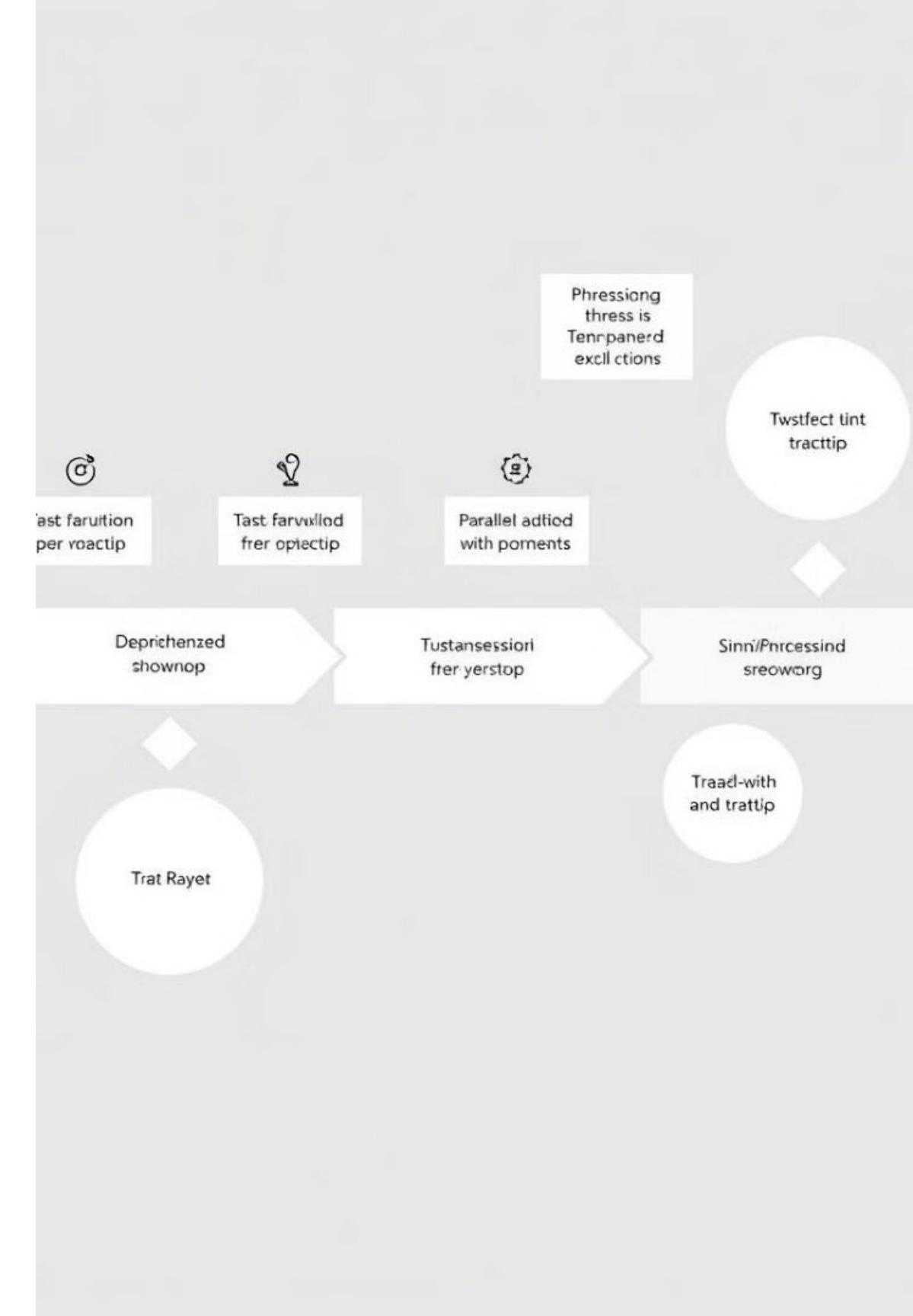
```
batch_size = 32
```

```
import gc
del variable_grande
gc.collect() # Llama al recolector de basura para liberar RAM
```

# Uso Eficiente y Paralelaje (parallelismo)



- Identificar las partes de tu código que se pueden ejecutar de forma independiente para aprovechar el paralelaje.
- Utilizar la biblioteca Threading para ejecutar múltiples tareas en paralelo en un solo núcleo de CPU.
- Utilizar la biblioteca Multiprocessing para ejecutar tareas en paralelo en múltiples núcleos de CPU.
- Utilizar la GPU para tareas intensivas en gráficos, como el aprendizaje profundo, para obtener una aceleración significativa.



Python ofrece varias herramientas para manejar el paralelismo, como el módulo **multiprocessing** y **concurrent.futures**.

```
import multiprocessing

# Definir una función que se ejecutará en paralelo
def funcion_lenta(num):
    return num ** 2

# Crear un pool de procesos
if __name__ == "__main__":
    with multiprocessing.Pool(processes=4) as pool:
        resultados = pool.map(funcion_lenta, range(10))
```

```
import concurrent.futures

def funcion_lenta(num):
    return num ** 2

with concurrent.futures.ProcessPoolExecutor(max_workers=4) as executor:
    resultados = list(executor.map(funcion_lenta, range(10)))

print(resultados)
```

En **tensorflow** tenemos el entrenamiento distribuido, para separar modelos y entrenamiento con cambios mínimos en diferentes GPU o TPU  
[tf.distribute.Strategy](#)

Síncrono: Todos los workers entranan sobre diferentes porciones de datos

Asíncrono: Todos los trabajadores se entranan de forma independiente y actualizan las variables de formas asíncronas (asíncrono enfocado al servidor)

API de entrenamiento	MirroredStrategy	TPUStrategy	MultiWorkerMirroredStrategy	CentralStorageStrategy	ParameterServerStrategy
<code>Model.fit</code>	Soportado	Soportado	Soportado	Soporte experimental	Soporte experimental
Bucle de entrenamiento personalizado	Soportado	Soportado	Soportado	Soporte experimental	Soporte experimental
API del estimador	Soporte limitado	No soportado	Soporte limitado	Soporte limitado	Soporte limitado

Estrategia	Descripción	Ventajas	Desventajas
<code>MirroredStrategy</code>	Réplica el modelo en múltiples GPUs locales, sincronizando las operaciones.	Fácil de usar, distribuye datos eficientemente, sincronización automática.	Limitada a una máquina, sincronización puede ser un cuello de botella.
<code>TPUStrategy</code>	Distribuye el trabajo en núcleos de TPU, optimizando redes neuronales.	Maximiza el uso de TPUs, reduce tiempo de entrenamiento, ideal para grandes datos.	Acceso limitado a TPUs, requiere ajustes para código optimizado en GPU.
<code>MultiWorkerMirroredStrategy</code>	Distribuye el modelo en múltiples máquinas, ideal para entornos distribuidos.	Alta escalabilidad para clústeres distribuidos.	Configuración compleja, latencia de red puede afectar rendimiento.
<code>CentralStorageStrategy</code>	Almacena variables en la CPU mientras distribuye los cálculos en GPUs.	Útil cuando los modelos son grandes y GPUs tienen menos memoria.	Acceso centralizado a variables puede ser un cuello de botella.
<code>ParameterServerStrategy</code>	Las variables se distribuyen entre múltiples servidores, con workers realizando cálculos.	Ideal para modelos grandes y muchas máquinas en configuraciones distribuidas.	Compleja configuración y latencia en la comunicación entre servidores.

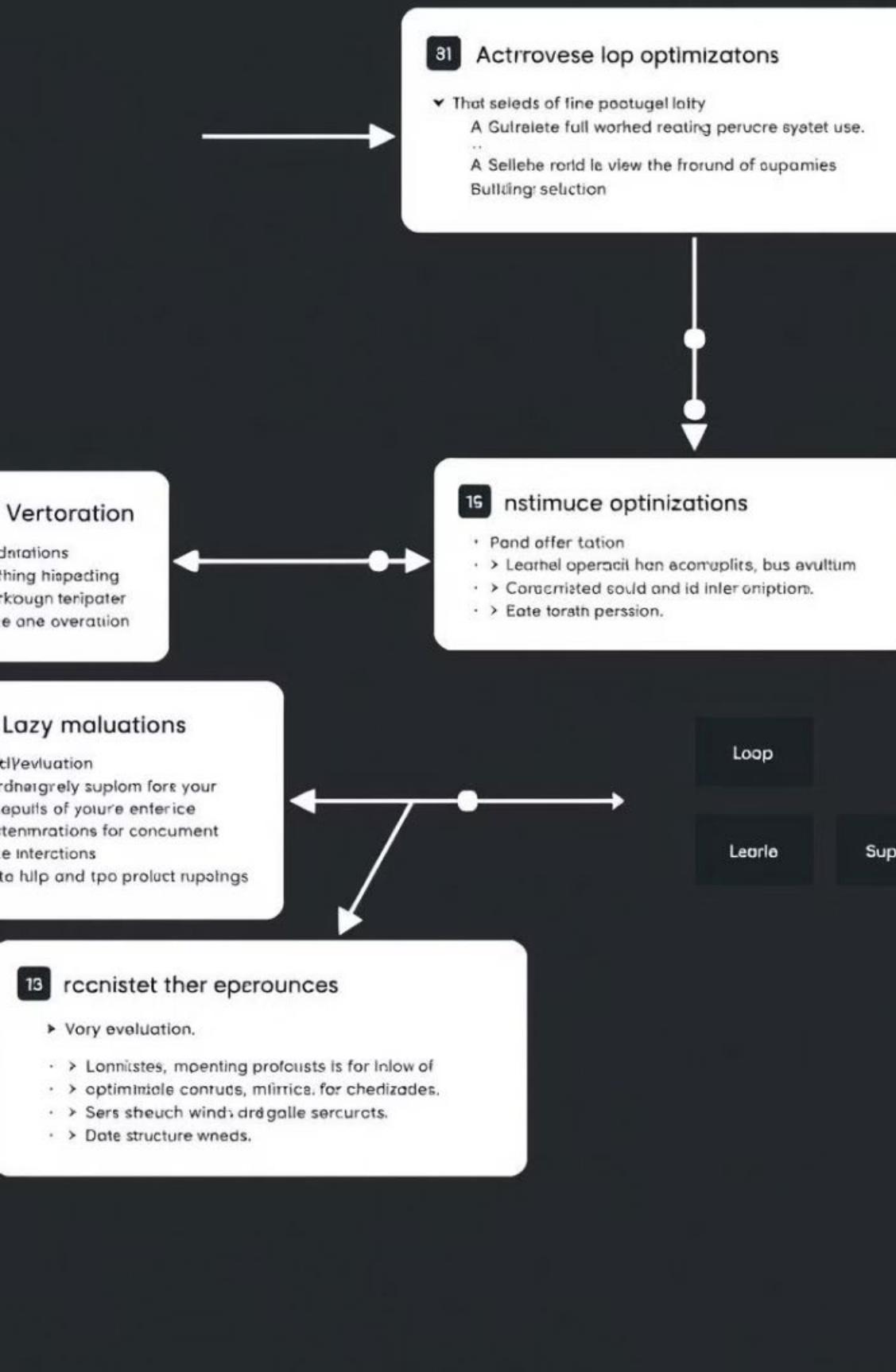
```
import tensorflow as tf
import numpy as np

# Verificar si hay GPUs disponibles
print("¿GPU disponible?:", tf.config.list_physical_devices('GPU'))

# Definir la estrategia
strategy = tf.distribute.MirroredStrategy()

# Definir el modelo bajo el scope de la estrategia
with strategy.scope():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(32,)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```



# Técnicas de Reducción de Consumo

# Optimización de Parámetros

## 1 Definición de Parámetros

Definir un rango de valores para cada parámetro que se desea optimizar.

## 2 Selección de un Algoritmo de Optimización

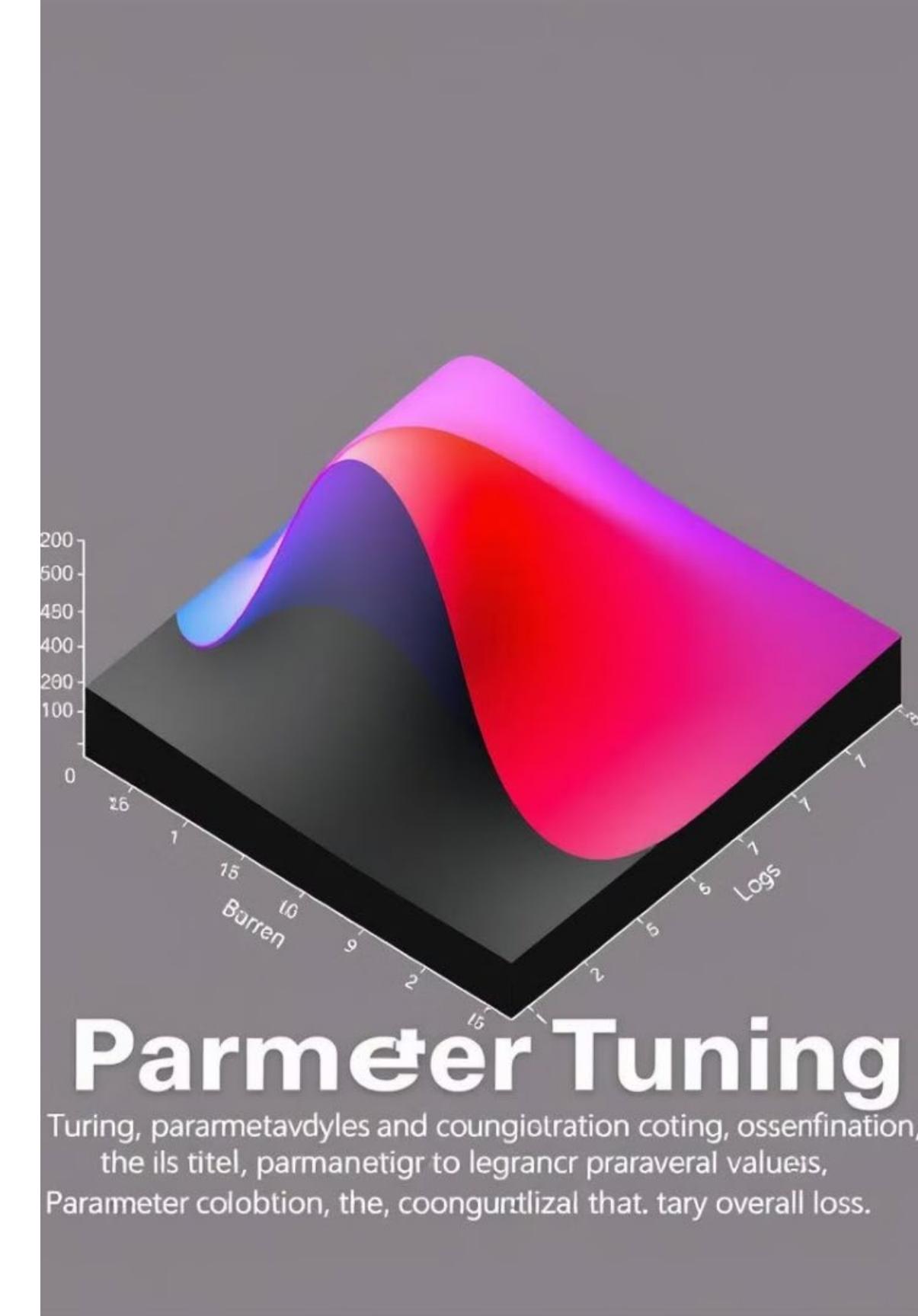
Elegir un algoritmo de optimización adecuado, como GridSearch o RandomSearch, para buscar la combinación óptima de parámetros.

## 3 Evaluación del Rendimiento

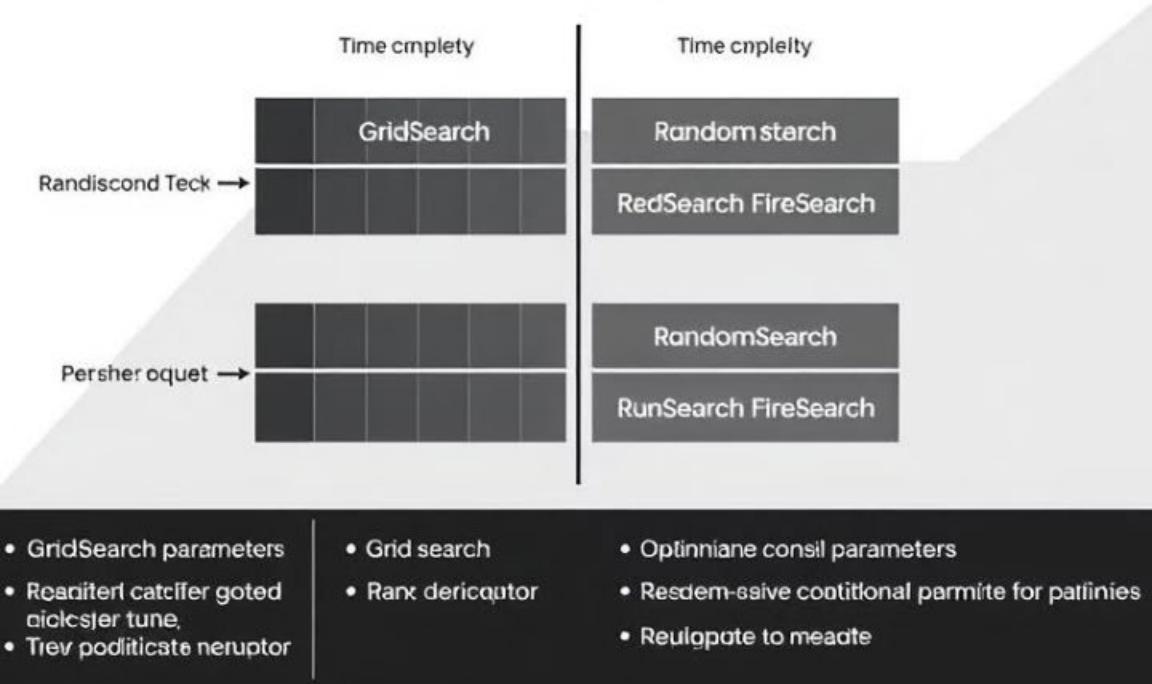
Evaluar el rendimiento del modelo utilizando métricas apropiadas, como la precisión o la pérdida, para determinar la mejor combinación de parámetros.

## 4 Iteración y Refinamiento

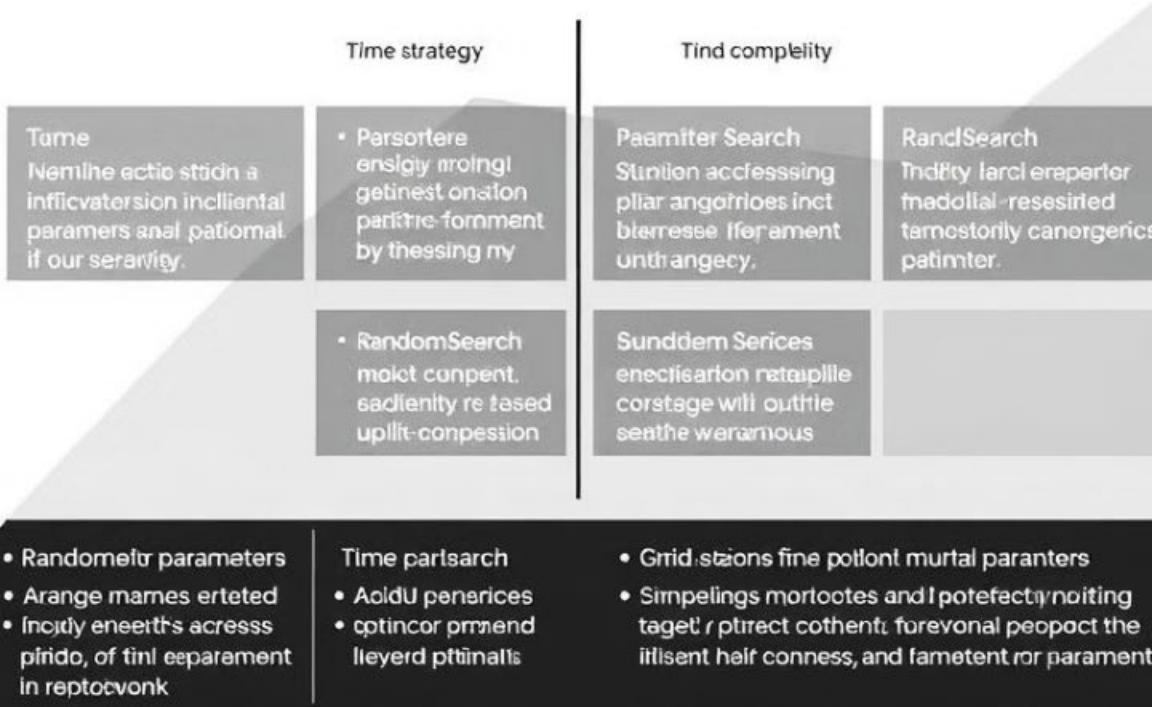
Ajustar los rangos de parámetros y repetir el proceso de optimización hasta que se alcance un rendimiento satisfactorio.



## GridSearch - Stamnity



## GridScarzh - Stamnitor



# GridSearch y RandomSearch

Técnica	Estrategia de Búsqueda	Complejidad Temporal	Eficacia
GridSearch	Búsqueda exhaustiva de todas las combinaciones de parámetros	Alta	Puede encontrar la mejor combinación, pero puede ser lento para espacios de parámetros grandes.
RandomSearch	Muestreo aleatorio de combinaciones de parámetros	Baja	Generalmente más eficiente que GridSearch, especialmente para espacios de parámetros grandes.

```
1 import numpy as np
2 from sklearn.model_selection import GridSearchCV
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.datasets import make_classification
5 from sklearn.model_selection import train_test_split
6
7 # Crear un conjunto de datos sintético
8 X, y = make_classification(n_samples=1000, n_features=20, random_state=42)
9
10 # Dividir el conjunto de datos en entrenamiento y prueba
11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
12
13 # Definir el modelo
14 model = RandomForestClassifier(random_state=42)
15
16 # Definir el grid de parámetros
17 param_grid = {
18     'n_estimators': [10, 50, 100],
19     'max_depth': [None, 10, 20, 30],
20     'min_samples_split': [2, 5, 10]
21 }
22
23 # Definir GridSearchCV
24 grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
25
26 # Entrenar el modelo con GridSearchCV
27 grid_search.fit(X_train, y_train)
28
29 # Mostrar los mejores parámetros
30 print("Mejores hiperparámetros encontrados:", grid_search.best_params_)
```

La GRID lleva los parámetros a evaluar del modelo

```
# Definir el grid de parámetros, pero con distribuciones aleatorias
param_dist = {
    'n_estimators': randint(10, 200), # entre 10 y 200 árboles
    'max_depth': randint(5, 50),
    'min_samples_split': randint(2, 20)
}

# Definir RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist,
                                    n_iter=20, cv=5, n_jobs=-1, verbose=2, random_state=42)

# Entrenar el modelo con RandomizedSearchCV
random_search.fit(X_train, y_train)

# Mostrar los mejores parámetros
print("Mejores hiperparámetros encontrados:", random_search.best_params_)
```



# Modulo VII

Cloud,  
CI/CD

# Computación en la nube

El Cloud permite almacenar las grandes cantidades de datos que consume la IA y acelera la digitalización de las empresas.





VS

VS

MICROSOFT  
AZURE

AMAZON WEB  
SERVICES

GOOGLE CLOUD  
PLATFORM

- ✓ Integración profunda con el ecosistema Microsoft.
- ✓ Amplia gama de servicios de IA y machine learning.
- ✓ Herramientas integradas para DevOps y desarrollo.
- ✓ Escalabilidad global con presencia en numerosas regiones.
- ✓ Soporte para múltiples sistemas operativos y lenguajes de programación.
- ✓ Seguridad y cumplimiento normativo robustos con certificaciones reconocidas a nivel mundial.

- ✓ Más de 200 servicios, ofreciendo gran variedad.
- ✓ Pionero en la nube, con gran base de clientes.
- ✓ Flexibilidad de precios con opciones de pago.
- ✓ Potencia computacional escalable y elástica con EC2.
- ✓ Amplia red global de centros de datos para alta disponibilidad.
- ✓ Amplia gama de herramientas de gestión y monitoreo.

- ✓ Fortaleza en big data y análisis avanzado.
- ✓ Enfoque líder en machine learning e IA.
- ✓ Red global de alto rendimiento para aplicaciones.
- ✓ Herramientas de desarrollo colaborativo y ágil.
- ✓ Servicios de contenedores con Kubernetes y Anthos.
- ✓ Compromiso con la sostenibilidad y energía renovable.



# Proyectos de MLOps en Colab o SageMaker o Azure Notebooks

\* ofrece un entorno de desarrollo interactivo y colaborativo para proyectos de MLOps. Su integración con \* Cloud Platform facilita el acceso a recursos de computación, almacenamiento y herramientas de aprendizaje automático.

## Escalabilidad

\* permite escalar tus proyectos de MLOps de manera eficiente, utilizando los recursos de \* Cloud según sea necesario.

## Colaboración

Comparte fácilmente tus proyectos de MLOps con otros miembros del equipo y colabora en tiempo real.

## Integración

Conéctate a \* Cloud Platform, servicios y herramientas de aprendizaje automático para un flujo de trabajo completo.

# Beneficios de la computación en la nube para MLOps

La computación en la nube ofrece ventajas significativas para los proyectos de MLOps. Facilita el acceso a recursos de computación, almacenamiento y herramientas de aprendizaje automático, optimizando el desarrollo, implementación y mantenimiento de modelos.

## 1 Escalabilidad

Aumenta o reduce la capacidad de computación según las necesidades de tu proyecto.

## 2 Costo-efectividad

Paga solo por los recursos que utilizas, evitando inversiones en infraestructura.

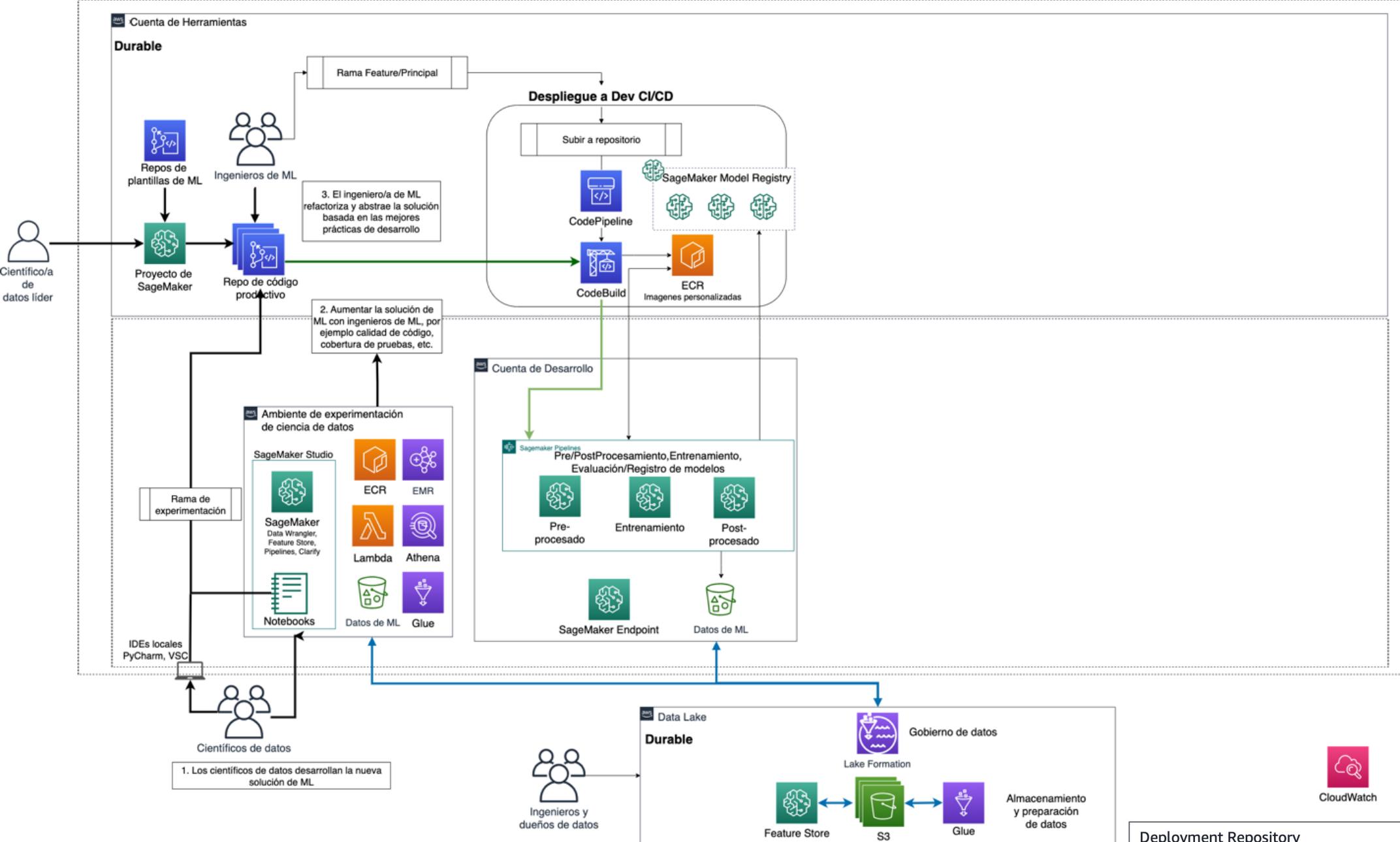
## 3 Agilidad

Implementa y actualiza tus modelos de forma rápida y eficiente, optimizando el ciclo de vida del desarrollo.

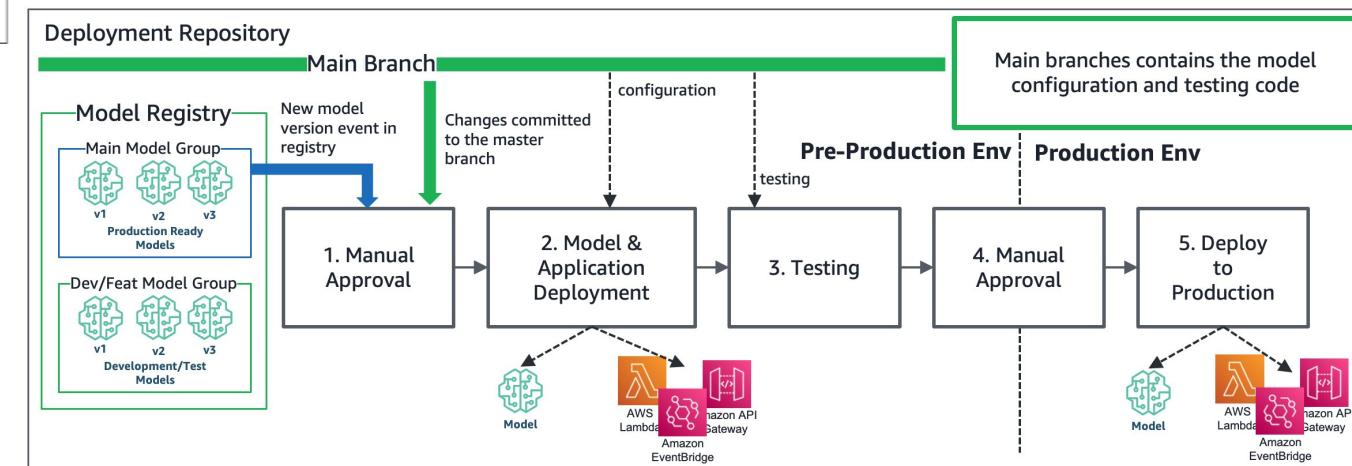
## 4 Seguridad

Aprovecha las medidas de seguridad robustas de los proveedores de nube, asegurando la protección de tus datos y modelos.





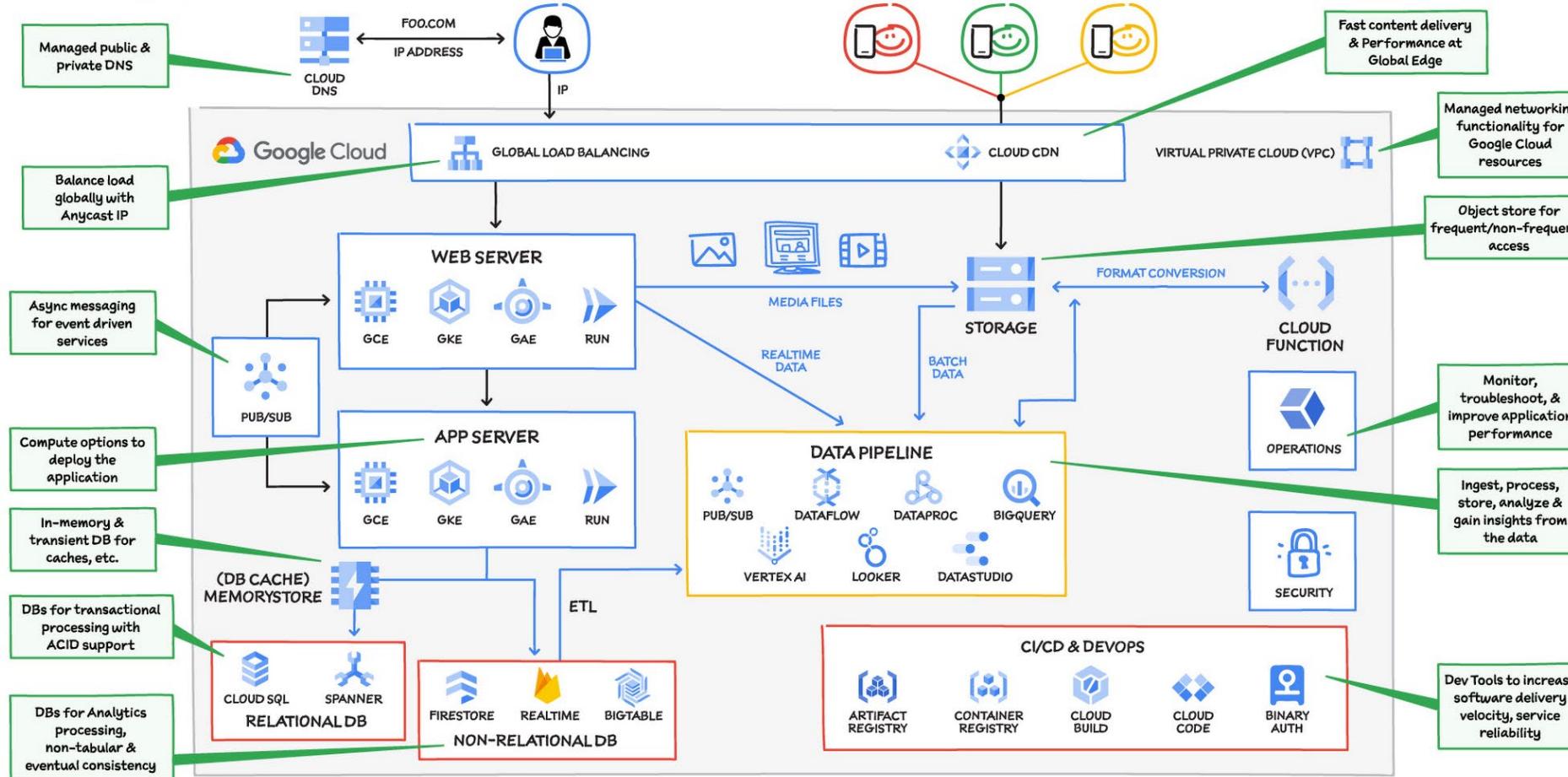
<https://aws.amazon.com/es/blogs/aws-spanish/roadmap-para-una-base-empresarial-de-mlops-con-amazon-sagemaker/>





# Application Architecture

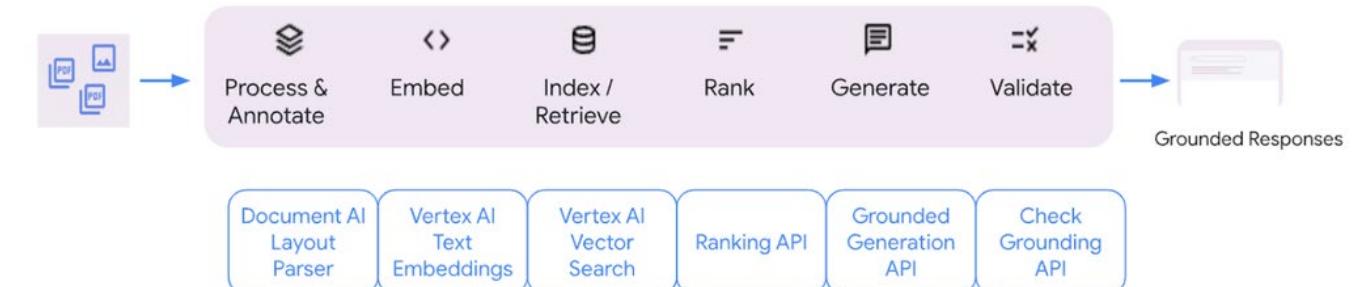
#GCPsketchnote [@PVERGADIA](#) [THECLOUDGIRL.DEV](#) 08.22.2021

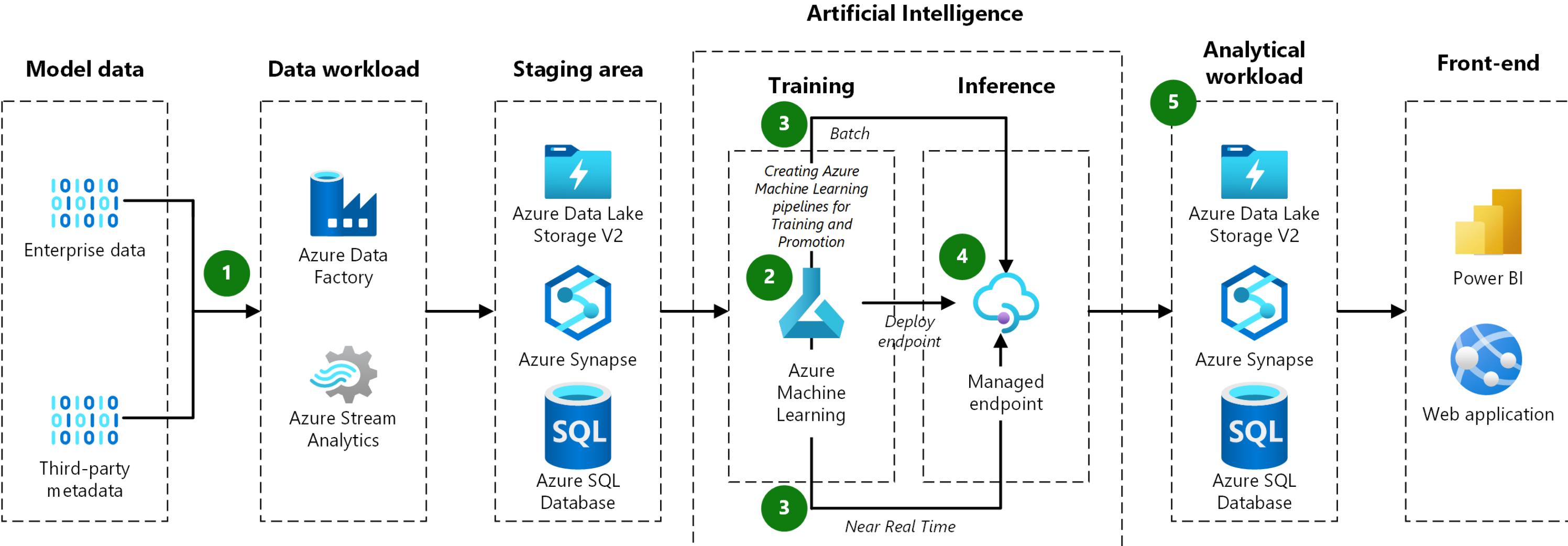


## Vertex AI APIs for RAG

<https://cloud.google.com/blog/topics/developers-practitioners/google-cloud-security-overview>

<https://www.linkedin.com/pulse/deployed-machine-learning-model-serverless-endpoint-shunmugaraj/>





<https://learn.microsoft.com/en-us/azure/architecture/ai-ml/idea-many-models-machine-learning-azure-machine-learning>

Proveedor	Ventajas	Desventajas
Azure	<ul style="list-style-type: none"> <li>- Integración con herramientas empresariales como <b>Azure DevOps</b> y Active Directory, facilitando MLOps y CI/CD.</li> <li>- <b>Azure Machine Learning (AML)</b> ofrece una plataforma completa para desarrollo y despliegue de modelos, con herramientas automáticas como AutoML.</li> <li>- Buen soporte para <b>PyTorch</b> y <b>TensorFlow</b>, con capacidades de procesamiento de GPU/TPU a gran escala.</li> <li>- Integración nativa con servicios de datos como <b>Azure Data Lake</b> y <b>Synapse Analytics</b>.</li> </ul>	<ul style="list-style-type: none"> <li>- Puede ser más costoso que otras alternativas para ciertas configuraciones.</li> <li>- Menos maduro en algunas áreas de IA comparado con Google Cloud AI.</li> <li>- Curva de aprendizaje elevada para usuarios que no están en el ecosistema de Microsoft.</li> </ul>
Google Cloud	<ul style="list-style-type: none"> <li>- Vertex AI proporciona un entorno unificado para gestionar el ciclo de vida del ML, integrando entrenamientos, experimentos y despliegues de manera ágil.</li> <li>- <b>BigQuery</b> y <b>Cloud Dataflow</b> son excelentes para análisis de datos a gran escala, con integración directa con modelos ML.</li> <li>- Gran soporte para frameworks de IA, especialmente <b>TensorFlow</b>, dado que fue desarrollado por Google.</li> <li>- Ofrece <b>TPUs</b> a precios competitivos para entrenamiento y despliegue de modelos de ML.</li> </ul>	<ul style="list-style-type: none"> <li>- La interfaz de usuario y las herramientas pueden ser complejas para nuevos usuarios.</li> <li>- Menos variedad de servicios globales que AWS.</li> <li>- Costos variables y difíciles de predecir en algunos servicios, especialmente con BigQuery.</li> </ul>
AWS	<ul style="list-style-type: none"> <li>- <b>SageMaker</b> es una de las plataformas más completas para el desarrollo y despliegue de modelos ML, con funcionalidades como AutoML, integración con notebooks y monitorización de modelos en producción.</li> <li>- Ofrece la mayor cantidad de regiones globales, asegurando baja latencia en todo el mundo.</li> <li>- Amplia gama de servicios para MLOps y Cloud Computing, desde bases de datos hasta almacenamiento y cómputo a gran escala.</li> <li>- Excelente escalabilidad y soporte para GPU/TPU a gran escala.</li> </ul>	<ul style="list-style-type: none"> <li>- La complejidad y la cantidad de servicios pueden ser abrumadoras para nuevos usuarios.</li> <li>- Los precios pueden ser complicados de calcular y optimizar, especialmente en uso continuo.</li> <li>- Algunos servicios pueden carecer de la simplicidad o la automatización que ofrece Google Cloud.</li> </ul>

# CI/CD

## Integración Continua Entrega e Implementación



# Conceptos clave de CI/CD

La integración continua (CI) y la entrega e implementación continua (CD) son prácticas de desarrollo de software que automatizan el proceso de construcción, prueba y despliegue de aplicaciones.

## 1 Integración continua (CI)

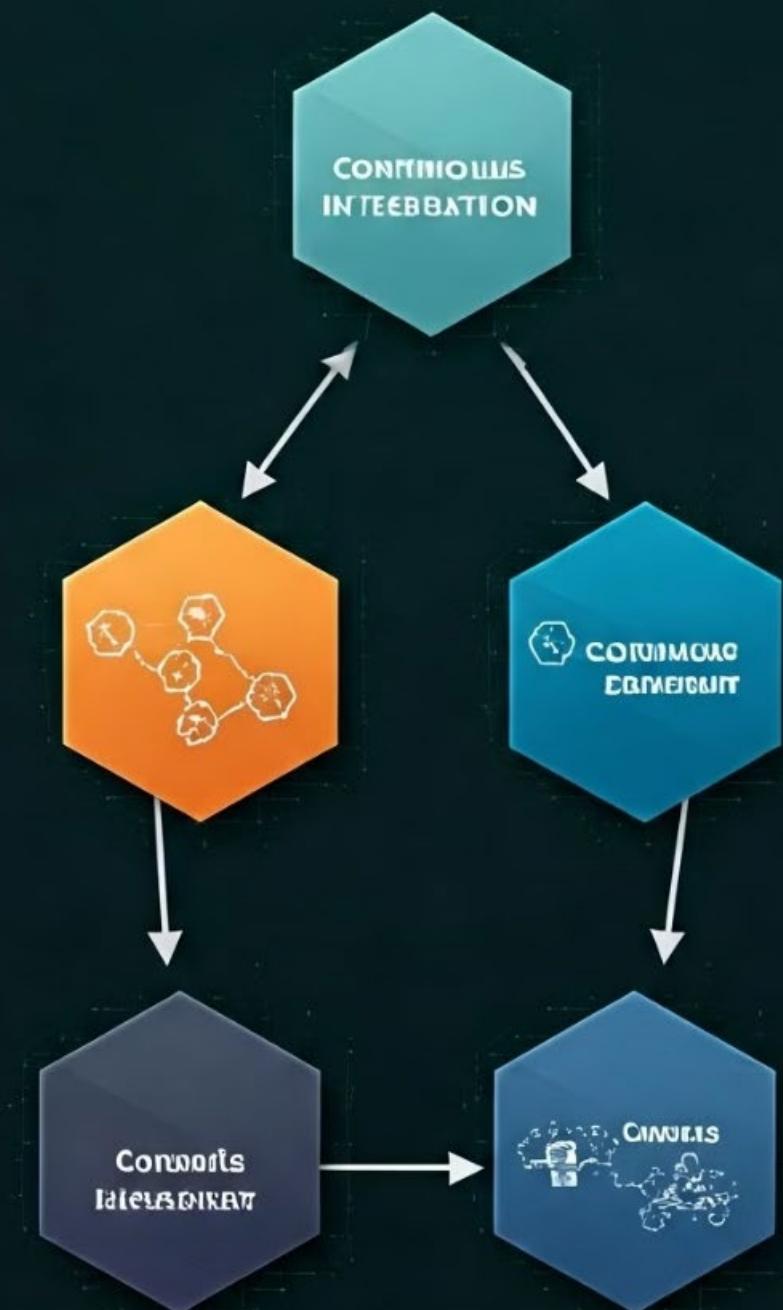
Automatiza la construcción y prueba de código, asegurando que los cambios del código se integren al repositorio principal de forma continua.

## 2 Entrega continua (CD)

Automatiza el despliegue de código probado en diferentes entornos, desde desarrollo hasta producción.

## 3 Implementación continua (CD)

Automatiza la implementación de cambios en el entorno de producción, permitiendo que los usuarios finales accedan a nuevas funcionalidades con rapidez.



# Integración continua (CI)

En la integración continua, los desarrolladores integran sus cambios de código en el repositorio principal con frecuencia. Esto ayuda a detectar errores de integración y a garantizar que el código funcione correctamente en cada etapa del proceso.

## Beneficios

1. Detección temprana de errores
2. Mejora la calidad del código
3. Reduce los tiempos de entrega
4. Aumenta la productividad

## Pruebas automatizadas

Las pruebas automatizadas son esenciales para la CI, ya que permiten ejecutar pruebas unitarias, de integración y de extremo a extremo de forma automática.

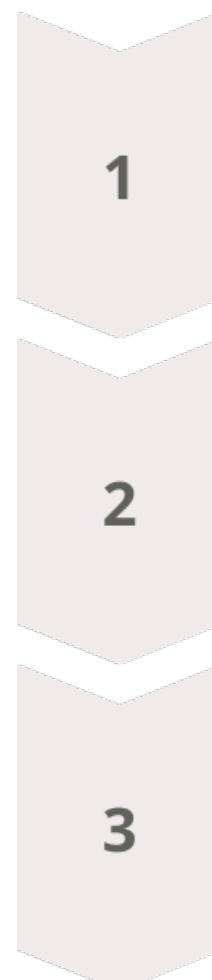
## Integración con herramientas

Integra la CI con herramientas de gestión de código fuente, como Git, y herramientas de integración continua como Jenkins, CircleCI o Travis CI.



# Entrega e implementación continua (CD)

La entrega continua se centra en automatizar el proceso de despliegue de código probado en diferentes entornos, desde desarrollo hasta producción.



## Integración continua (CI)

El código se **integra** al repositorio principal y se prueba de forma continua.

## Entrega continua (CD)

El código probado se despliega de forma automática en un entorno de **prueba**.

## Implementación continua (CD)

El código probado se despliega de forma automática en el entorno de **producción**.



# Ventajas de implementar CI/CD

Las prácticas de CI/CD ofrecen numerosos beneficios para el desarrollo de software, especialmente en entornos ágiles y de alta frecuencia de entrega.

Entrega más rápida

Reducción de errores

Mejor calidad del software

Mayor productividad

Mayor flexibilidad

Colaboración mejorada



# Introducción a GitHub Actions

GitHub Actions es una plataforma de automatización de flujos de trabajo para GitHub que permite automatizar tareas como la integración continua, la entrega continua y la implementación continua.



## Construcción

Automatiza la construcción de tu código fuente, incluyendo tareas de compilación y empaquetado.



## Pruebas

Ejecuta pruebas automatizadas para verificar la calidad del código y asegurar su correcto funcionamiento.



## Despliegue

Automatiza el despliegue de tu código en diferentes entornos, desde desarrollo hasta producción.



## Integración

Integra GitHub Actions con otras herramientas de desarrollo, como Jira, Slack, y herramientas de CI/CD.

- 1.Crea un archivo de **workflow**: En tu repositorio de GitHub, crea un archivo YAML dentro del directorio `.github/workflows/`.
- 2.Define un **trigger**: Especifica el evento que simulará el inicio del workflow, como `push` o `pull_request`.
- 3.Especifica **jobs**: Define los trabajos (jobs) que quieras ejecutar, que pueden incluir pasos como compilar, probar y desplegar tu código.
- 4.Define los **steps**: En cada job, especifica los pasos a seguir, como ejecutar comandos o usar acciones predefinidas de la comunidad.



# Modulo VIII

Documentación y Buenas prácticas

# Introducción: Documentación y buenas prácticas



# Documentación

Son herramientas poderosas para crear documentación de alta calidad. Puedes documentar pipelines, modelos y librerías de manera eficiente, creando una referencia completa y orientada a fácil navegación.

## 1 Reproducibilidad

En MLOps, es vital replicar experimentos, experimentos y modelos, lo que requiere un registro detallado de pasos y configuraciones.

## 2 Colaboración

Los proyectos suelen involucrar varios equipos, una documentación clara facilita la colaboración y avance en los desarrollos.

## 3 Mantenimiento

Los modelos pueden necesitar actualizaciones o ajustes en el futuro. Una buena documentación asegura que cualquier ingeniero pueda hacer el ajuste rápidamente sin introducir errores.

## 4 Automatización y escalabilidad

En MLOps, muchos procesos suelen ir automatizados, documentar las automatizaciones es esencial para la escalabilidad del Proyecto.

## 3 Cumplimiento y auditoría

La trazabilidad y cumplimiento normativo requiere una documentación detallada, para poder determinar por qué se tomaron decisiones, como se hicieron y con qué datos.



# 10 Aspectos clave que deberíamos documentar en proyectos de MLOps

## 1. Arquitectura del pipeline de datos:

- Explicar cómo se gestionan los datos, desde la extracción y transformación hasta el almacenamiento y la entrada en los modelos.
- Incluir esquemas y descripciones de las fuentes de datos, procesos de limpieza, y cualquier otro componente relevante.

## 2. Configuraciones de modelos y experimentos:

- Detallar los hiperparámetros utilizados en los modelos, versiones del código, configuraciones de hardware, y datos utilizados en cada experimento.
- Guardar estas configuraciones en herramientas como MLflow, Weights & Biases o simplemente en archivos de configuración para facilitar la reproducibilidad.

## 3. Versionado de datos y modelos:

- Documentar cómo se gestionan las versiones de los datos, los cambios en los modelos y modelos y cómo se controlan las versiones del código.
- Herramientas como DVC (Data Version Control) o Git pueden ser útiles para este propósito, y este propósito, y se debe explicar cómo se usan en el proyecto.

# 10 Aspectos clave que deberíamos deberíamos documentar en proyectos proyectos de MLOps

## 4. Monitoreo y métricas:

- Describir qué métricas de rendimiento se están monitorizando en producción, cómo se producen, cómo se recogen los datos para el monitoreo y qué herramientas se utilizan.
- Incluir los umbrales de alerta que indican cuando un modelo está funcionando fuera de lo funcionando fuera de lo esperado (drift de datos, baja precisión, etc.).

## 5. Pipeline de CI/CD:

- Documentar cómo se implementa la integración y el despliegue continuo, incluyendo cómo se ejecutan las pruebas automatizadas, los pasos de validación y los procesos de despliegue del modelo.
- Explicar el uso de herramientas como Jenkins, GitLab CI, o sistemas como Kubeflow o MLflow para la automatización.

## 6. Manejo de errores y resolución de problemas:

- Incluir guías para la resolución de problemas comunes y el manejo de errores en el pipeline, desde fallos en el procesamiento de datos hasta problemas en el modelo en producción.
- Crear una lista de comprobaciones o manuales para resolver estos errores de manera eficiente.

# 10 Aspectos clave que deberíamos documentar en proyectos de MLOps

## 7. Descripciones de roles y responsabilidades:

- Indicar qué parte del pipeline o sistema es responsabilidad de cada miembro del equipo o qué equipo maneja cada aspecto del ciclo de vida del modelo.

## 8. Cumplimiento y privacidad:

- Explicar las políticas de cumplimiento y privacidad en el manejo de datos, particularmente en casos donde se maneja información sensible.
- Documentar cómo se anonimiza, protege y gestiona el acceso a los datos.

## 9. Consideraciones de seguridad:

- Detallar cómo se asegura el pipeline de machine learning, incluyendo autenticación, acceso a datos, manejo de secretos, y cualquier otro aspecto relacionado con la seguridad.

## 10. Evolución del modelo:

- Explicar cómo se gestionarán las futuras versiones del modelo, cuándo y cómo se determinará que un modelo necesita ser actualizado, y cómo se manejarán las pruebas A/B u otras pruebas de validación.

# Herramientas para documentación de proyectos

Herramienta	Descripción	Ventajas	Desventajas
Sphinx	Generador de documentación en Python que usa reStructuredText por defecto, ampliamente utilizado en proyectos de software, especialmente en Python.	<ul style="list-style-type: none"> <li>- Potente y flexible para grandes proyectos.</li> <li>- Soporte nativo para autodocumentación a partir de docstrings en el código.</li> <li>- Compatible con múltiples formatos de salida (HTML, PDF, ePub).</li> </ul>	<ul style="list-style-type: none"> <li>- La sintaxis de reStructuredText es más compleja que Markdown.</li> <li>- Configuración inicial puede ser complicada para usuarios principiantes.</li> </ul>
MkDocs	Herramienta de documentación estática que usa Markdown para crear sitios web de documentación. Ideal para proyectos que prefieren Markdown sobre reStructuredText.	<ul style="list-style-type: none"> <li>- Fácil de usar con soporte para Markdown.</li> <li>- Integración rápida con GitHub Pages para despliegue.</li> <li>- Extensibilidad a través de temas y plugins.</li> </ul>	<ul style="list-style-type: none"> <li>- Limitada en personalización avanzada comparada con Sphinx.</li> <li>- Menos soporte para documentos muy complejos o grandes proyectos técnicos.</li> </ul>
Docusaurus	Plataforma de código abierto basada en React, mantenida por Facebook, que facilita la creación de sitios web de documentación utilizando Markdown.	<ul style="list-style-type: none"> <li>- Integración sencilla con frontends modernos como React.</li> <li>- Fácil de usar con soporte para múltiples lenguajes de programación y desarrollo colaborativo.</li> <li>- Soporte nativo para despliegue en GitHub Pages.</li> </ul>	<ul style="list-style-type: none"> <li>- Puede ser excesivo para proyectos simples.</li> <li>- La curva de aprendizaje puede ser mayor para desarrolladores sin experiencia en React.</li> </ul>



¿Proyectos grandes con mucha documentación técnica de software y API's? Usar Sphinx, se basa en reStructuredText



¿Proyectos pequeños o medianos? Usa MKDOCS, es mas fácil de usar que Sphinx y se basa en Markdown



¿Queremos algo altamente personalizable? Quedémonos con docusaurus, se suele usar para stack de desarrollo web.

# Buenas prácticas de codificación

Las buenas prácticas de codificación garantizan la calidad, legibilidad y mantenibilidad del código. Sigue estas recomendaciones para escribir código de alta calidad.

## Legibilidad

Utiliza nombres de variables significativos, comentarios claros y espacios en blanco para mejorar la comprensión.

## Consistencia

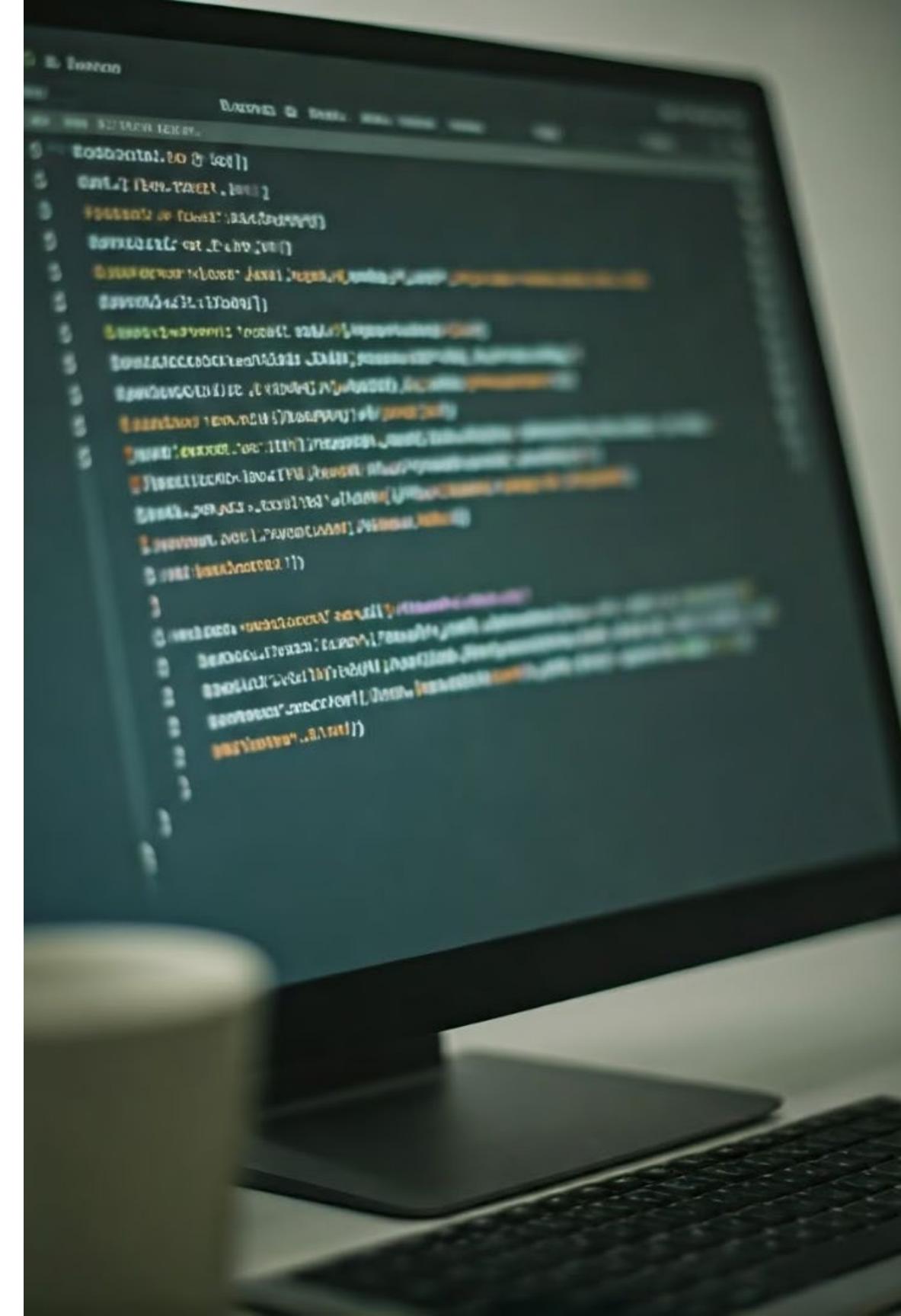
Sigue un estilo de codificación consistente, utilizando herramientas de formateo para asegurar la uniformidad.

## Modularidad

Divide el código en funciones y módulos para facilitar la reutilización y reutilización y la depuración.

## Pruebas unitarias

Escribe pruebas unitarias para verificar el correcto funcionamiento del código.





# Seguridad y privacidad en el desarrollo

La seguridad y la privacidad son aspectos fundamentales del desarrollo de software. Implementar medidas de seguridad adecuadas y proteger la información personal es crucial.

- 1 Autenticación y autorización  
Controla el acceso a los recursos y la información, utilizando mecanismos de autentificación y autorización robustos.
- 2 Cifrado  
Utiliza cifrado para proteger la información confidencial durante el almacenamiento y la transmisión.
- 3 Protección de datos  
Implementa medidas para proteger la información personal, siguiendo las leyes de protección de datos y las normas de privacidad.
- 4 Pruebas de seguridad  
Realiza pruebas de seguridad para identificar y corregir vulnerabilidades.



## Conclusiones y recomendaciones finales

La documentación y las buenas prácticas son esenciales para el desarrollo de software de calidad. Implementar estas prácticas te permitirá crear código legible, mantenible y seguro.



Escribe documentación clara y concisa

La documentación debe ser fácil de entender y navegar.



Prioriza la seguridad y la privacidad

Protege la información confidencial y los datos del usuario.



Colabora con otros desarrolladores

La colaboración mejora la calidad del código y la documentación.



Utiliza las herramientas adecuadas

Hay herramientas que facilitan la documentación y el desarrollo.



# **Mentorías hasta el 30 de Septiembre**

## **Septiembre**

**Lunes a viernes (sábado A convenir)**  
**convenir)**

**6 a 8 (Temas abiertos...)**  
**8 a 10 (Temas abiertos + Charlas de**  
**de profesionales)**

**Si desea un horario específico y**  
**diferente (Mutuo Acuerdo)**