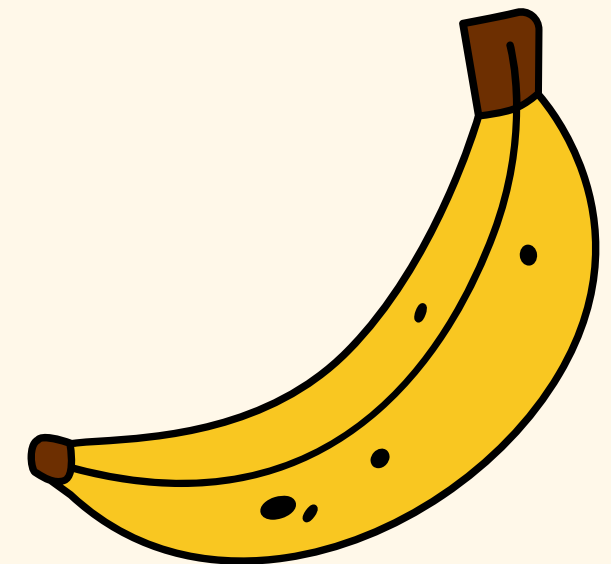


# PROYECTO KAMBUR

LOS ÁNGELES DE CHARLIE



# AGENDA:

Introducción

¿Qué es Kubeflow?

Estructura de los componentes

Código

Ejecución de Kubeflow

Demo

Conclusiones

# INTRODUCCIÓN

Esta práctica tiene como objetivo principal la investigación de la herramienta de MLOps Kubeflow para orquestar y automatizar el ciclo de vida completo del aprendizaje automático. Con Kubeflow, buscamos optimizar la gestión de pipelines, el escalado de modelos y la colaboración entre equipos.



# AGENDA:

Introducción

¿Qué es Kubeflow?

Estructura de los componentes

Código

Ejecución de Kubeflow

Demo

Conclusiones

# ¿QUÉ ES KUBEFLOW?

Kubeflow es un proyecto de código abierto que contiene un set de herramientas y frameworks compatibles entre ellos específicos para el Machine Learning.

Uno de los objetivos clave de Kubeflow es hacer más fácil el manejo, desarrollo e implementación del aprendizaje automático (machine learning).

Es una plataforma basada en Kubernetes, para la orquestación de contenedores basado en tres principios fundamentales



Componibilidad



Portabilidad



Escalabilidad

# PRINCIPIOS DE KUBEFLOW

## *Componibilidad:*

*Poder escoger lo que es mejor para tu proyecto, por ejemplo diferentes versiones de distintas herramientas.*

## *Portabilidad:*

*Poder ejecutar tu código en cualquier lugar en el que ejecutes Kubeflow, ya sea tu portátil o la nube.*

## *Escalabilidad:*

*El proyecto puede acceder o dejar de usar ciertos recursos cuando lo necesite, como la CPU.*

# AGENDA:

Introducción

¿Qué es Kubeflow?

Estructura de los componentes

Código

Ejecución de Kubeflow

Demo

Conclusiones

# ESTRUCTURA DE LOS COMPONENTES

Tenemos 3 componentes principales:



preprocesamiento1



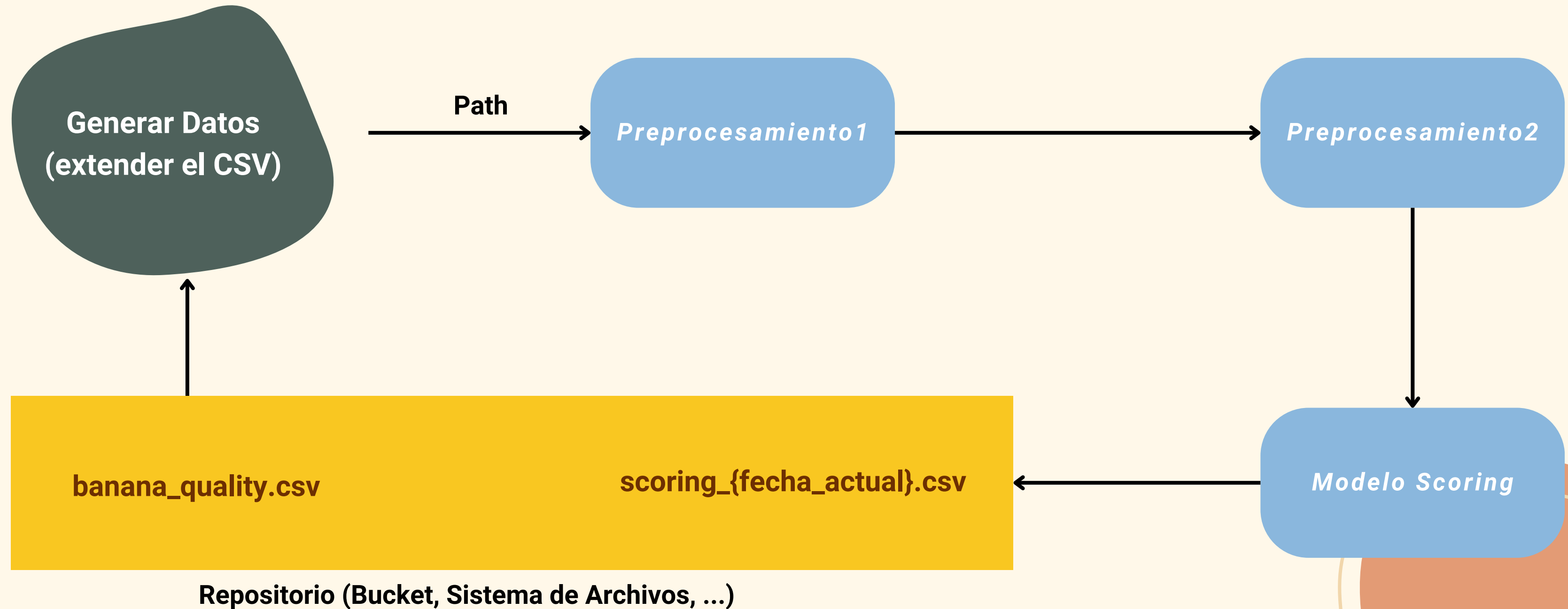
preprocesamiento2



modeloScoring



# ESTRUCTURA DE LOS COMPONENTES:



# AGENDA:

**Introducción**

**¿Qué es Kubeflow?**

**Estructura de los componentes**

**Código**

**Ejecución de Kubeflow**

**Demo**

**Conclusiones**

# PREPROCESAMIENTO1:

*Realiza la primera etapa de preprocesamiento de los datos.*

## **Inputs:**

- *Ruta del archivo .csv de entrada*

## **Outputs:**

- *Ruta del archivo .csv preprocesado*

*El componente preprocesamiento1 toma el archivo .csv de entrada, realiza el preprocesamiento de los datos y genera un .csv preprocesado en la ubicación especificada*

# PREPROCESAMIENTO1:

*preprocesamiento1.yaml*

```
preprocesamiento1:
  description: Preprocesamiento 1
  inputs:
  - {name: input_path, type: InputPath, description: 'Path del archivo CSV de entrada'}
  outputs:
  - {name: output_path, type: OutputPath, description: 'Path del archivo CSV preprocesado'}
  implementation:
    container:
      command:
      - python3
      - /app/preprocesamiento1.py
      - --input_path
      - {inputPath: input_path}
      - --output_path
      - {outputPath: output_path}
```

*dockerfile*

```
# Usa una imagen de Python como base
FROM python:3.10-slim

# Establece el directorio de trabajo en /app
WORKDIR /app

# Copia el código actual al contenedor en /app
COPY preprocesamiento1.py .

# Instala las dependencias de Python
RUN pip install --no-cache-dir pandas scikit-learn kfp

# Define el comando por defecto para ejecutar tu aplicación
CMD ["python", "preprocesamiento1.py"]
```

# PREPROCESAMIENTO1:

```
import pandas as pd
import argparse
from sklearn.preprocessing import StandardScaler, LabelEncoder

def preprocesamiento1(input_path: str, output_path: str):
    df = pd.read_csv(input_path)

    # Escalar características
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df.drop('Quality', axis=1))
    df_transformado = pd.DataFrame(scaled_features, columns=df.columns[:-1])

    # Codificar variable Quality a (1 y 0)
    label_encoder = LabelEncoder()
    df_transformado['Quality_encoded'] = label_encoder.fit_transform(df['Quality'])

    # Eliminar valores NA
    df_transformado = df_transformado.dropna()
```

*preprocesamiento1.py*

```
# Eliminar valores atípicos
q1 = df_transformado.quantile(0.25)
q3 = df_transformado.quantile(0.75)
iqr = q3 - q1
lower_limit = q1 - 1.5 * iqr
upper_limit = q3 + 1.5 * iqr
df_transformado = df_transformado[~((df_transformado < lower_limit) | (df_transformado > upper_limit)).any(axis=1)]

df_transformado.to_csv(output_path, index=False)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Preprocesamiento de datos')
    parser.add_argument('--input_path', type=str, required=True, help='Ruta del archivo CSV de entrada')
    parser.add_argument('--output_path', type=str, required=True, help='Ruta del archivo CSV de salida')
    args = parser.parse_args()
    preprocesamiento1(args.input_path, args.output_path)
```

# PREPROCESAMIENTO2:

*Realiza la segunda etapa de preprocesamiento de datos*

## **Inputs:**

- *Ruta del archivo .csv de entrada*

## **Outputs:**

- *Ruta del archivo .csv preprocesado*

*El componente preprocesamiento2 toma el archivo .csv de entrada, realiza un preprocesamiento adicional de los datos y genera un .csv preprocesado en la ubicación especificada*

# PREPROCESAMIENTO2:

*preprocesamiento2.yaml*



```
preprocesamiento2:
  description: Preprocesamiento 2
  inputs:
  - {name: input_path, type: InputPath, description: 'Path del archivo CSV de entrada'}
  outputs:
  - {name: output_path, type: OutputPath, description: 'Path del archivo CSV preprocesado'}
  implementation:
    container:
      command:
      - python3
      - /app/preprocesamiento2.py
      - --input_path
      - {inputPath: input_path}
      - --output_path
      - {outputPath: output_path}
```

```
# Usa una imagen base que contenga Python
FROM python:3.11
```

```
# Instala las dependencias necesarias
RUN pip install numpy
```

```
# Establece el directorio de trabajo en /app
WORKDIR /app
```

```
# Copia el script de preprocesamiento2.py al contenedor
COPY preprocesamiento2.py .
```

```
# Indica que el contenedor ejecutará este script cuando se inicie
CMD ["python", "preprocesamiento2.py"]
```

*dockerfile*





# PREPROCESAMIENTO2:

```
import numpy as np
import pandas as pd
import argparse

def preprocesamiento2(input_path: str, output_path: str):
    # Leer el DataFrame de entrada
    df = pd.read_csv(input_path)

    # Añadir columnas NotaConsumidor, Temporada, Estacion
    df['NotaConsumidor'] = np.random.randint(0, 11, size=len(df))
    df['Temporada'] = np.random.randint(0, 2, size=len(df))
    df['Estacion'] = np.random.choice(['Primavera', 'Verano', 'Otoño', 'Invierno'], size=len(df))

    # Guardar el DataFrame procesado
    df.to_csv(output_path, index=False)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Preprocesamiento 2 de datos')
    parser.add_argument('--input_path', type=str, required=True, help='Ruta del archivo CSV de entrada')
    parser.add_argument('--output_path', type=str, required=True, help='Ruta del archivo CSV de salida')
    args = parser.parse_args()
    preprocesamiento2(args.input_path, args.output_path)
```

*preprocesamiento2.py*



# MODELOSCORING:

*dockerfile*



*modeloScoring.yaml*

```
modeloscoring:
  description: Modelo Scoring
  inputs:
    - {name: input_csv_path, type: InputPath, description: 'Ruta del archivo CSV de entrada'}
    - {name: mes, type: Integer, description: 'Mes de los datos'}
    - {name: ruta_bucket_csv, type: String, description: 'Ruta del bucket para guardar el archivo CSV de salida'}
  implementation:
    container:
      command:
        - python3
        - modeloscoring.py
        - --input_csv_path
        - {inputPath: input_csv_path}
        - --mes
        - {inputValue: mes}
        - --ruta_bucket_csv
        - {inputValue: ruta_bucket_csv}
```

```
# Usa una imagen de Python como base
FROM python:3.9-slim
```

```
# Establece el directorio de trabajo en /app
WORKDIR /app
```

```
# Copia el código actual al contenedor en /app
COPY modeloScoring.py .
```

```
# Instala las dependencias de Python
RUN pip install --no-cache-dir pandas h2o kfp
```

```
# Define el comando por defecto para ejecutar tu aplicación
CMD ["python", "modeloScoring.py"]
```

# MODELOSCORING:

*modeloScoring.py*

```
import h2o
from h2o.estimators import H2OGBBoostEstimator
import pandas as pd
from kfp import dsl, components

@component(
    base_image="imagen_modelo_scoring:latest", # package to install
)

def modeloScoring(input_csv_path: str, mes: int, ruta_bucket_csv: str):
    df_mes = df[df['Mes'] == mes].copy()

    h2o.init()
    data = h2o.H2OFrame(df_mes)

    # Creacion de subconjuntos
    predictors = ["Precio", "NotaConsumidor", "Size", "Weight", "Sweetness", "Softness",
"HarvestTime", "Ripeness", "Acidity"]
    response = "Quality_encoded"

    data[response] = data[response].asfactor()

    # Dividir los datos en train + dev + test
    train, test, dev = data.split_frame(ratios=[0.75, 0.15], destination_frames=['train_df',
'test_df', 'val_df'], seed=566)
```

```
# Entrenar el modelo
xgb = H2OGBBoostEstimator()
xgb.train(x=predictors, y=response, training_frame=train, validation_frame=dev)
```

```
# Hacer predicciones y evaluar el modelo
pred = xgb.predict(test)
evaluacion = xgb.model_performance(test)
```

```
# Obtener las predicciones como un DataFrame de Pandas
pred_df = pred.as_data_frame()
```

```
# Guardarlo en el bucket
fecha_actual = pd.Timestamp.now().strftime('%Y-%m-%d')
nombre_archivo_csv = f"scoring_{fecha_actual}.csv"
ruta_completa = f"{ruta_bucket_csv}/{nombre_archivo_csv}"
pred_df.to_csv(ruta_completa, index=False)
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Modelo de Scoring')
    parser.add_argument('--input_csv_path', type=str, required=True, help='Ruta del archivo CSV de
entrada')
    parser.add_argument('--mes', type=int, required=True, help='Mes de los datos')
    parser.add_argument('--ruta_bucket_csv', type=str, required=True, help='Ruta del bucket para
guardar el archivo CSV de salida')
    args = parser.parse_args()
    modeloScoring(args.input_csv_path, args.mes, args.ruta_bucket_csv)
```

# AGENDA:

Introducción

¿Qué es Kubeflow?

Estructura de los componentes

Código

Ejecución de Kubeflow

Demo

Conclusiones

# Ejecución de KUBEFLOW

## 1) Ejecutarlo en GCP



### Kubeflow Pipelines

Versión: Kubeflow Pipelines 2.0.0 ▼

[Google Cloud AI Platform](#)

Reusable end-to-end ML workflow platform

CONFIGURAR

Se produjo un error cuando se creaba un clúster. Vuelve a intentarlo o selecciona un clúster existente. Motivo de la falla: Se produjo un error durante la creación del clúster nuevo - Insufficient regional quota to satisfy request: resource "SSD\_TOTAL\_GB": request requires '300.0' and is short '50.0'. project has a quota of '250.0' with '250.0' available. View and manage quotas at <https://console.cloud.google.com/iam-admin/quotas?usage=USED&project=peak-age-420618>.

# EJECUCIÓN DE KUBEFLOW

## 2) Ejecutarlo en GCP con MiniKube



### MiniKF

[Arrikto](#)

The fastest and easiest way to get started with Kubeflow

INICIAR

VER IMPLEMENTACIONES

Tarifa por uso de MiniKF

EUR 85.20/mes

La tarifa por uso de imagen se cobra con un mínimo de 1 minutos, y Google la factura:

Precio basado en CPU virtuales:

- EUR 0.116706 por hora (EUR 85.20 por mes) para todas las instancias.

[^ MOSTRAR MENOS](#)

Tarifa de infraestructura

Instancia de VM: 8 CPU virtuales + 30 GB de memoria (n1-standard-8)

EUR 258.99/mes

Disco de estado sólido: 200 GB

EUR 12.14/mes

Disco de estado sólido: 500 GB

EUR 30.34/mes

Descuento por uso continuo

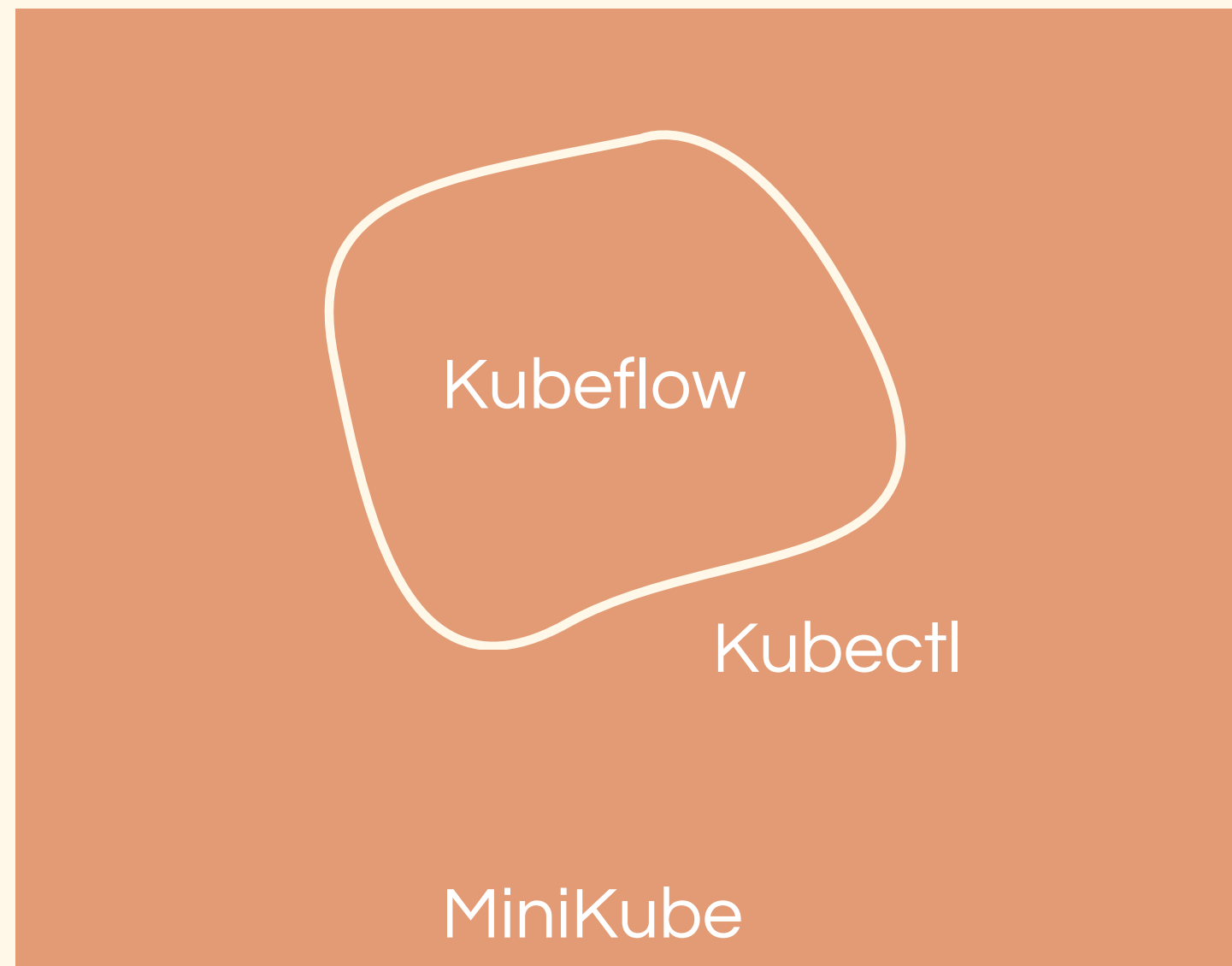
- EUR 77.70/mes

**Total mensual estimado**

**EUR 308.97/mes**

# EJECUCIÓN DE KUBEFLOW

## 3) Descargarse Kubeflow en una máquina virtual



Minikube: Herramienta que permite ejecutar un clúster de Kubernetes local en una máquina virtual para facilitar el desarrollo y las pruebas.

Kubectl: Herramienta de línea de comandos para gestionar y administrar clústeres de Kubernetes.



# EJECUCIÓN DE KUBEFLOW

## 3) Descargarse Kubeflow en una máquina virtual

```
INFO[0002] Processing application: pipeline-visualization-service filename="kustomize/kustomize.go:408"  
INFO[0002] Processing application: profiles filename="kustomize/kustomize.go:408"  
INFO[0002] Processing application: seldon-core-operator filename="kustomize/kustomize.go:408"  
INFO[0002] /home/ubuntu/Desktop/MLOps/mi-kubeflow-sie/.cache/manifests exists; not resyncing filename="kfconfig/types.go:468"  
INFO[0002] namespace: kubeflow filename="utils/k8utils.go:427"  
INFO[0002] Creating namespace: kubeflow filename="utils/k8utils.go:432"  
Error: failed to apply: (kubeflow.error): Code 500 with message: kfApp Apply failed for kustomize: (kubeflow.error): Code 400 with message: could  
n't create namespace kubeflow Error: Post "http://localhost:8080/api/v1/namespaces": dial tcp 127.0.0.1:8080: connect: connection refused  
Usage:
```

```
minikube start --driver=<<driver>>
```

MiniKube

Los drivers de Minikube determinan la tecnología de virtualización o contenedores que se usa para crear el clúster local de Kubernetes.

- Docker
- VirtualBox
- Hyper-V
- KVM
- VMware
- ...

# EJECUCIÓN DE KUBEFLOW

## 3) Descargarse Kubeflow en una máquina virtual

### Install a Hypervisor

If you do not already have a hypervisor or a virtualizer installed, install a new one. Once the hypervisor is installed, you don't need to start or use it directly. Minikube will automatically invoke the hypervisor to start the VM.

Mac OS X

Install [Virtual Box](#) or [VMware Fusion](#).

Ubuntu

Install [Virtual Box](#) or [KVM](#).

The KVM2 driver is intended to replace KVM driver. The KVM2 driver is maintained by the minikube team, and is built, tested and released with minikube. For installing KVM:

Minikube

En la documentación de kubeflow nos indican que se debe utilizar un driver de Virtual Box o de KVM2

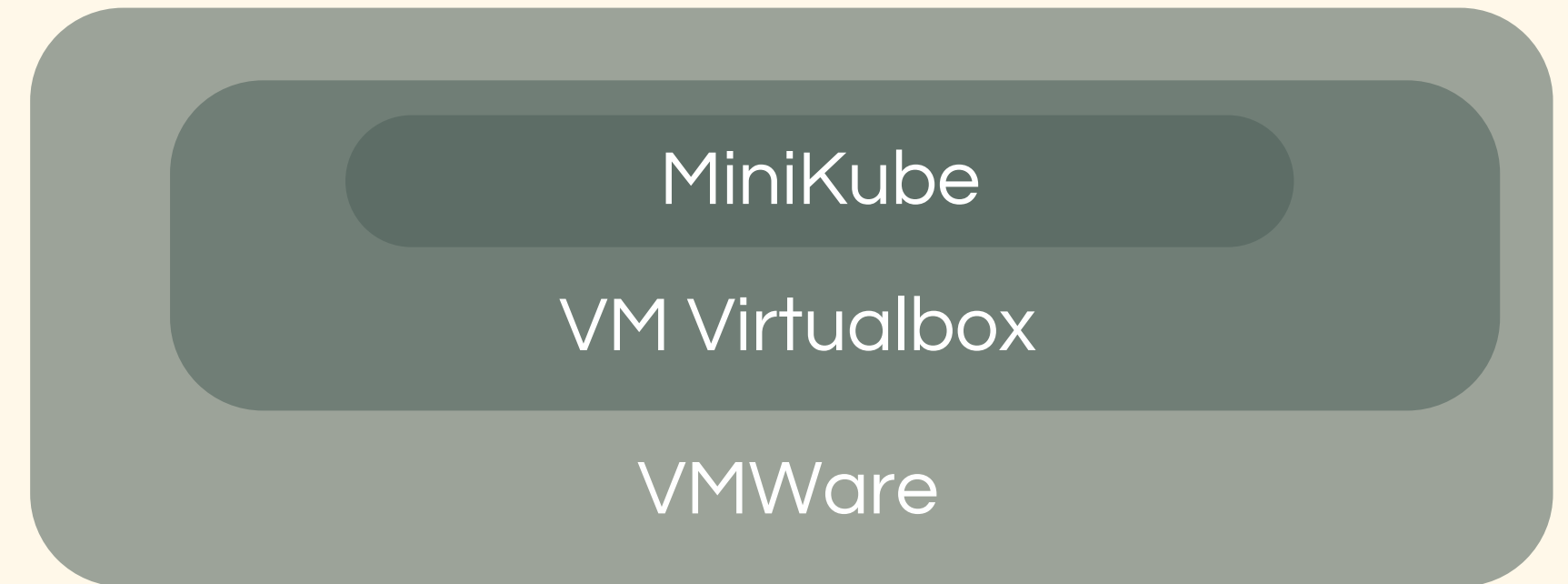
Pero VirtualBox no soporta la virtualización anidada de forma predeterminada, lo que significa que no se puede ejecutar una VM dentro de otra VM .



# EJECUCIÓN DE KUBEFLOW

## 3) Descargarse Kubeflow en una máquina virtual

La otra opción sería KVM2 pero nuestra máquina virtual tendría que ser capaz de virtualizar otra máquina virtual dentro, es decir, el hipervisor principal debería soportar y habilitar la virtualización anidada.



```
ubuntu@ubuntu-virtual-machine:~/Desktop/MLOps$ egrep -c '(vmx|svm)' /proc/cpuinfo
sudo kvm-ok
0
[sudo] password for ubuntu:
INFO: Your CPU does not support KVM extensions
KVM acceleration can NOT be used
ubuntu@ubuntu-virtual-machine:~/Desktop/MLOps$ S
```

# EJECUCIÓN DE KUBEFLOW

## 3) Descargarse Kubeflow en una máquina virtual

```
Error from server (InternalError): error when creating "STDIN": Internal error occurred: failed calling webhook "clusterservingruntime.kserve-webhook-server.validator": failed to call webhook: Post "https://kserve-webhook-server-service.kubeflow.svc:443/validate-serving-kserve-io-v1alpha1-clusterservingruntime?timeout=10s": dial tcp 10.103.238.101:443: connect: connection refused
Error from server (InternalError): error when creating "STDIN": Internal error occurred: failed calling webhook "clusterservingruntime.kserve-webhook-server.validator": failed to call webhook: Post "https://kserve-webhook-server-service.kubeflow.svc:443/validate-serving-kserve-io-v1alpha1-clusterservingruntime?timeout=10s": dial tcp 10.103.238.101:443: connect: connection refused
Error from server (InternalError): error when creating "STDIN": Internal error occurred: failed calling webhook "clusterservingruntime.kserve-webhook-server.validator": failed to call webhook: Post "https://kserve-webhook-server-service.kubeflow.svc:443/validate-serving-kserve-io-v1alpha1-clusterservingruntime?timeout=10s": dial tcp 10.103.238.101:443: connect: connection refused
```

Si en vez de una CentOS utilizamos una Ubuntu da el siguiente error

# EJECUCIÓN DE KUBEFLOW

## 3) Descargarse Kubeflow en una máquina virtual

### Prerequisites

Before we jump into deploying Kubeflow, ensure you have the following prerequisites in place:

**CentOS 7 VM:** Set up a CentOS 7 virtual machine using VMware or your preferred virtualization platform.

**Minikube:** Install Minikube, which will help you create a local Kubernetes cluster on your CentOS 7 VM.

**kubectl:** Ensure you have `kubectl` installed. This command-line tool is essential for interacting with your Kubernetes cluster.

**Kustomize 5.0.3:** Kubeflow requires Kustomize version 5.0.3 or later. Make sure it's installed on your system.



### Upcoming EOL Dates

CentOS Stream 8 end of builds is **May 31, 2024**. CentOS Linux 7 end of life is **June 30, 2024**. Read the **information on upgrade and migration options**.

Otra forma de hacerlo es con una Máquina virtual de CentOS, pero la versión que sugieren en todas las documentaciones está desactualizada.

# EJECUCIÓN DE KUBEFLOW

## 4) Descargarse MiniKF en una máquina virtual

MiniKF es una forma rápida y sencilla de desplegar Kubeflow en tu el ordenador. La última versión ofrece un despliegue local completo y listo para producción en minutos. Es ideal para experimentar y ejecutar flujos de trabajo completos de Kubeflow. Además, permite escalar el entrenamiento fácilmente a un despliegue en la nube de Kubeflow, sin necesidad de reescribir nada.

Entonces... ¿Cómo procedimos?

Para empezar hay que instalar tanto Vagrant como virtual box

```
root@server:/# apt install vagrant
```

### Vagrant

Vagrant es una herramienta para la creación y configuración de entornos de desarrollo virtualizados

# EJECUCIÓN DE KUBEFLOW

## 4) Descargarse MiniKF en una máquina virtual

Para arrancar minikf era necesario 'iniciarlo'... pero esto `root@server:/# vagrant init minikf` demandó pasos previos, como:

```
root@server:/# sudo apt install libvirt-daemon libvirt-clients libvirt-daemon-system qemu-kvm
```

La instalación de los paquetes para utilizar Libvirt y QEMU-KVM en el sistema (lo que permite la creación, gestión y ejecución de máquinas virtuales en un entorno de virtualización basado en Linux).

```
root@server:/# vagrant box add generic/ubuntu2004 --provider=libvirt
```

El siguiente paso consistió en descargar y agregar una 'nueva caja' de Ubuntu al entorno de Vagrant, utilizando el proveedor de virtualización libvirt para gestionar las máquinas virtuales. Una vez que la caja se ha agregado, se puede utilizar como base para crear y ejecutar nuevas máquinas virtuales con Vagrant en un entorno de virtualización basado en Linux. Entonces iniciamos el servicio en nuestro entorno `root@server:/# vagrant init generic/ubuntu2004`

# EJECUCIÓN DE KUBEFLOW

## 4) Descargarse MiniKF en una máquina virtual

```
root@server:/# sudo vim /etc/libvirt/qemu.conf

# Master configuration file for the QEMU driver.
# All settings described here are optional - if omitted, sensible
# defaults are used.
user = "root"
group = "root"
```

```
root@server:/# sudo usermod -aG libvirt,libvirt-qemu root
```

[imagen 2]

Una vez que tenemos el servicio de virtualización de 'libvirt' era necesario habilitar el usuario y el grupo (previamente creados con \*[ver imagen 2]\*) con el que trabajamos para continuar configurando el entorno

Procedemos entonces a arranca una máquina virtual utilizando Vagrant, que a su vez utiliza Libvirt como el proveedor de virtualización.

```
root@server:/# vagrant up --provider=libvirt
```



# EJECUCIÓN DE KUBEFLOW

## 4) Descargarse MiniKF en una máquina virtual

Finalmente, después de corregir diversas inconsistencias relacionadas con permisos, configuraciones del procesador y clientes para el servidor de 'libvirt'... Llegamos a siguiente error:

```
Error while creating domain: Error saving the server: Call to virDomainDefine failed: invalid argument: could not get preferred machine for /usr/bin/qemu-system-x86_64 type=kvm
```


Este error está relacionado con la creación de un dominio en un entorno de virtualización utilizando QEMU. La parte específica del error "invalid argument: could not get preferred machine for /usr/bin/qemu-system-x86\_64 type=kvm" indica que hay un problema al intentar obtener la máquina preferida para ejecutar el sistema QEMU.

Una posible explicación de por qué no se puede solucionar este error es que puede haber un problema con la configuración del entorno de virtualización o con la instalación de los componentes necesarios para ejecutar QEMU correctamente en la máquina virtual. Esto podría ser debido a una variedad de razones, como problemas de configuración del sistema, permisos incorrectos, o la falta de componentes o dependencias necesarios para QEMU.

# EJECUCIÓN DE KUBEFLOW

## 5) Micro8ks

### Add-on: Kubeflow

 **Note:** The add-on to install Kubeflow is no longer the recommended way to get up and running. Instead there is a complete end-to-end tutorial on deploying Kubeflow on MicroK8s now published in the [Charmed Kubeflow documentation](#).

- Conocido por su simplicidad y facilidad de uso. Se puede tener un clúster Kubernetes en funcionamiento con un solo comando, lo cual simplifica mucho el proceso de configuración inicial.
- Proporciona un add-on específico para Kubeflow, lo que significa que puedes instalar Kubeflow de manera directa y sin muchas complicaciones adicionales.



# EJECUCIÓN DE KUBEFLOW

## 5) Micro8ks

### Requirements:

This tutorial assumes you will be deploying Kubeflow on a public cloud VM with the following specs:

- Runs Ubuntu 20.04 (focal) or later.
- Has at least 4 cores, 32GB RAM and 50GB of disk space available.
- Is connected to the internet for downloading the required snaps and charms.
- Has `python3` installed.

### Problema:

- No tenemos el Hardware necesario (32GB RAM)
- Los ordenadores del laboratorio llevan ocupados 2 semanas por los simulacros

# AGENDA:

**Introducción**

**¿Qué es Kubeflow?**

**Estructura de los componentes**

**Código**

**Ejecución de Kubeflow**

**Demo**

**Conclusiones**

The background is a light cream color. It features abstract decorative elements: blue wavy lines in the top-left and bottom-right corners, and yellow circular shapes in the top-right and bottom-left corners.

**DEMO**



# CONCLUSIONES