
UNIVERSIDAD DE GRANADA

# PRÁCTICA 2

## Cloud Computing: Servicios y Aplicaciones

---

Máster Universitario en Ingeniería Informática

**Autor**

Pablo Valenzuela Álvarez (pvalenzuela@correo.ugr.es)

# ÍNDICE

**1. Introducción..... 2**

**2. Minikube.....3**
2.1. Instalación..... 3

**3. OpenFaaS..... 5**
3.1. Instalación..... 5
3.1.1. Instalación de Arkade.....5
3.1.2. Instalación de OpenFaaS.....5
3.1.3. Instalación de faas-cli.....6
3.2. Nuestra primera función en OpenFaaS.....8

**4. Función de reconocimiento facial..... 10**
4.1. Funciones de la store..... 10

**5. Conclusión.....12**

**6. Bibliografía..... 13**

# 1. Introducción

El objetivo de esta práctica es implementar un servicio de reconocimiento facial usando una plataforma FaaS (Función como servicio). Para su realización, deberemos de instalar una herramienta de orquestación de contenedores como Minikube [1] y desplegar la funcionalidad que queremos desarrollar mediante OpenFaaS [2].

Durante el desarrollo de esta memoria, se irán siguiendo los pasos necesarios para cumplir los objetivos y desplegar una función capaz de hacer un reconocimiento facial. Como nota extra, esta práctica propone que el alumno desarrolle su propia función de reconocimiento facial.

# 2. Minikube

Minikube es una herramienta de código abierto que nos permite crear un entorno de clusters locales de Kubernetes en sistemas macOS, Windows y Linux.

## 2.1. Instalación

Para su instalación necesitaremos un archivo ejecutable disponible en la página web de la herramienta y cumplir con los siguientes requerimientos:

- Una máquina con más de dos nucleos de CPU.
- Espacio de memoria RAM de 2 GB.
- 20 GB de espacio libre en disco.
- Conexión a internet.
- Una herramienta de gestión de contenedores (Docker, …) o una máquina virtual.

Una vez instalada, podemos iniciar nuestro primer cluster indicando en una consola la siguiente orden: *minikube start* (ver figura 1).

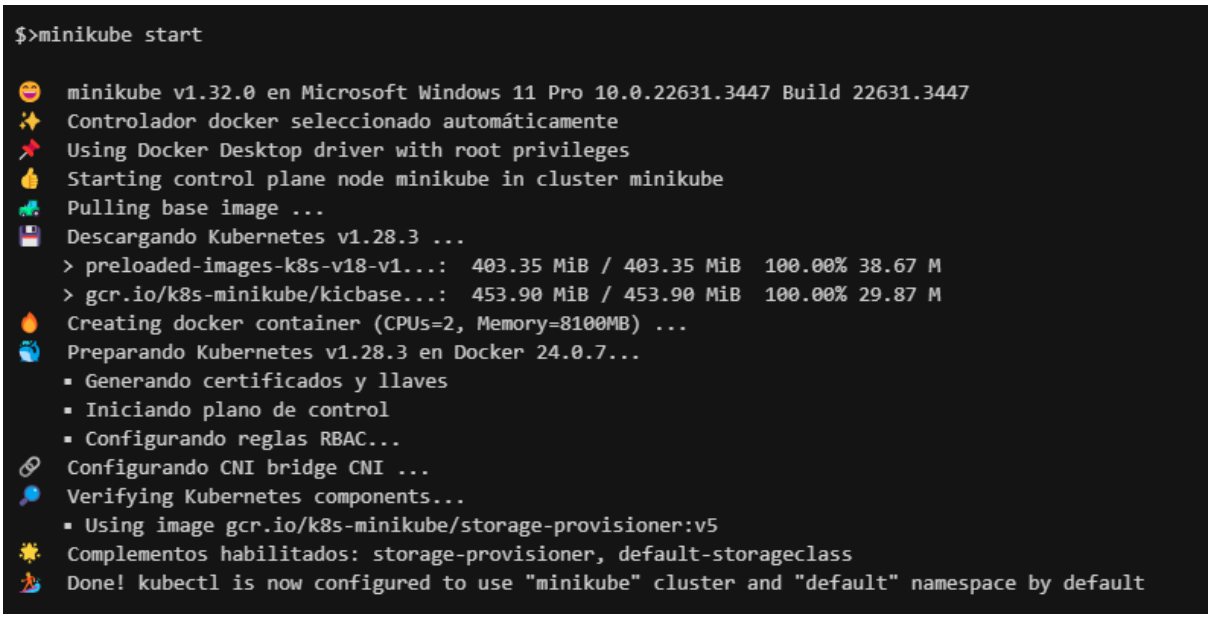


**Figura 1. Creando nuestro primer cluster.**

Si queremos ver nuestro cluster desde una interfaz gráfica necesitaremos del addon *dashboard*. Para instalarlo debemos indicar en una consola la siguiente orden: *minikube dashboard* (ver figura 2).

```
$>minikube dashboard

🔌  Habilitando dashboard
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server


🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:58837/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your
default browser...
```

**Figura 2. Instalación del addon *dashboard*.**

Cuando acabe, nos redirigirá automáticamente a la página de nuestro navegador y mostrará el estado actual del cluster. La siguiente figura (ver figura 3), muestra un despliegue que dispone de tres pods que ejecutan contenedores nginx.

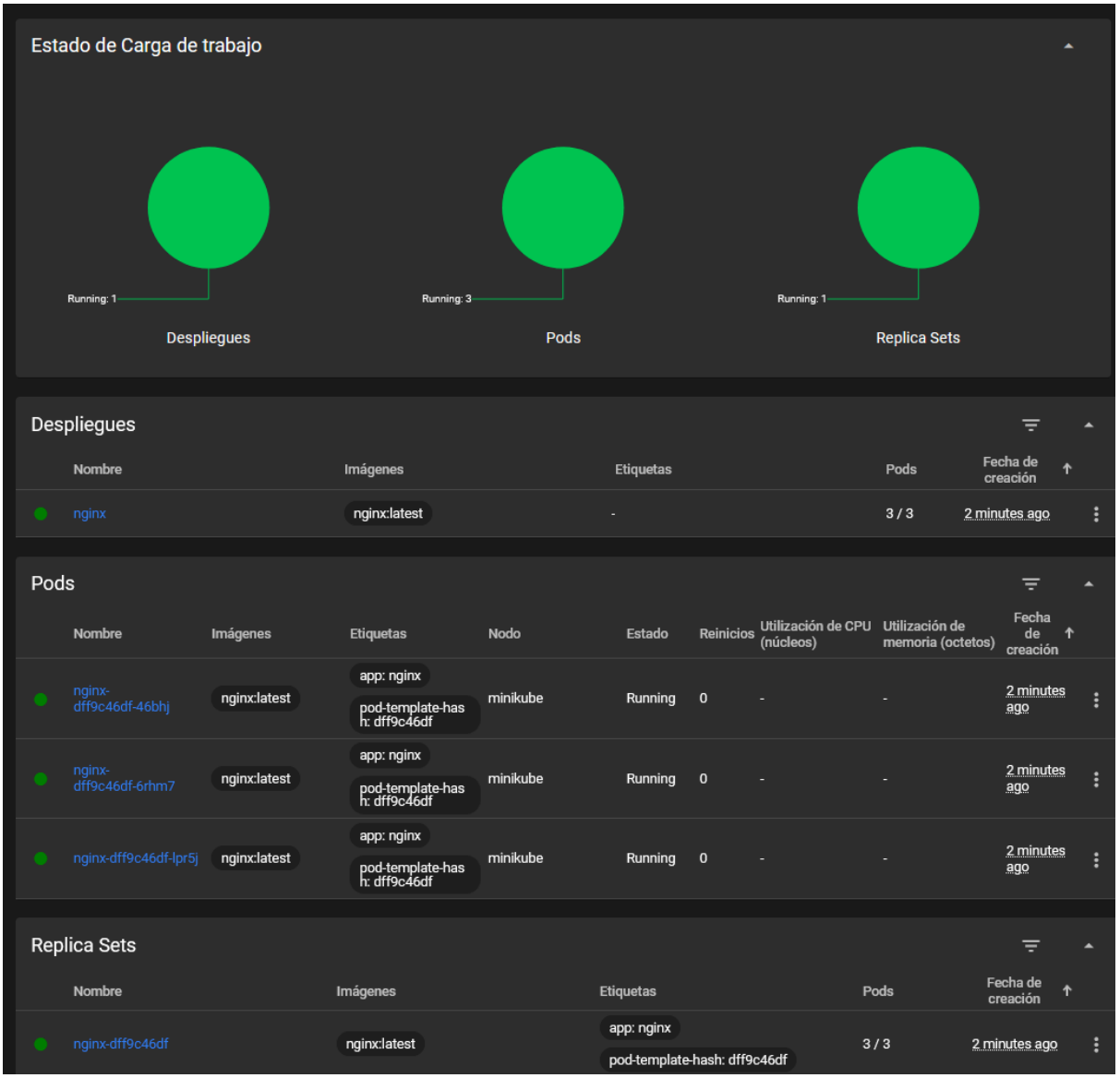


**Figura 3. Ejemplo de estado del cluster .**

# 3. OpenFaaS

OpenFaaS es un proyecto de código abierto que proporciona una plataforma para ejecutar funciones en contenedores, lo que permite a los programadores crear, desplegar y ejecutar funciones sin preocuparse por la infraestructura en la que se ejecute.

Podemos integrar OpenFaaS en el ecosistema de Kubernetes, lo que nos permite aprovechar las características proporcionadas por este último: escalabilidad, balanceo de carga, gestión de recursos, seguridad, portabilidad, etc.

## 3.1. Instalación

La instalación de esta herramienta es un poco más complicada. Antes de nada, necesitaremos de un cluster activo: *minikube start*.

Una vez lo tengamos activo, tenemos que seguir los siguientes pasos.

### 3.1.1. Instalación de Arkade

En mi caso, al usar un sistema Windows, necesito de una consola git para ejecutar el siguiente comando (ver figura 4): *curl -sLS https://get.arkade.dev | sh*

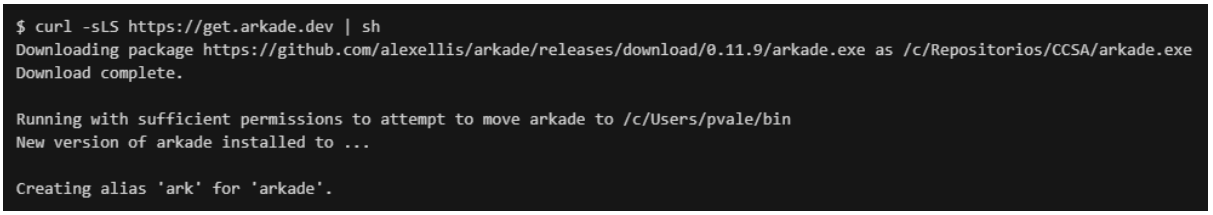


**Figura 4. Instalación de Arkade.**

### 3.1.2. Instalación de OpenFaaS

La instalación de OpenFaaS es similar a el paso anterior, en la consola git tenemos que ejecutar el siguiente comando (ver figura 5): *$ ark install openfaaS.*

```
$ ark install openfaas
Using Kubeconfig: C:\Users\pvale/.kube/config

[...]

To retrieve the admin password, run:

  echo $(kubectl -n openfaas get secret basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode)
=======================================================================
= OpenFaaS has been installed.                                        =
=======================================================================

# Get the faas-cli
curl -SLsf https://cli.openfaas.com | sudo sh

# Forward the gateway to your machine
kubectl rollout status -n openfaas deploy/gateway
kubectl port-forward -n openfaas svc/gateway 8080:8080 &

# If basic auth is enabled, you can now log into your gateway:
PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)
echo -n $PASSWORD | faas-cli login --username admin --password-stdin

faas-cli store deploy figlet
faas-cli list

# For Raspberry Pi
faas-cli store list \
 --platform armhf

faas-cli store deploy figlet \
 --platform armhf

# Find out more at:
# https://github.com/openfaas/faas

🚀 Speed up GitHub Actions/GitLab CI + reduce costs: https://actuated.dev
```

**Figura 5. Instalación de OpenFaaS.**

## 3.1.3. Instalación de faas-cli

Si nos fijamos en la figura anterior ( ver figura 5), nos pide que ejecutemos una serie de comandos. El primero que debemos ejecutar es *curl -SLsf https://cli.openfaas.com* | *sh* para instalar la interfaz de línea de comandos que usaremos para interactuar con la plataforma ( ver figura 6).

```
$ curl -SLsf https://cli.openfaas.com | sh
Finding latest version from GitHub
0.16.26
Downloading package https://github.com/openfaas/faas-cli/releases/download/0.16.26/faas-cli.exe as /c/Repositorios/CCSA/faas-cli.exe
Download complete.

Running with sufficient permissions to attempt to move faas-cli to /c/Users/pvale/bin
New version of faas-cli installed to /c/Users/pvale/bin
Creating alias 'faas' for 'faas-cli'.
main: line 180: /usr/local/bin/faas-cli: No such file or directory
```

**Figura 6. Instalación de faas-cli.**

Seguidamente debemos activar los contenedores y abrir un puerto para comunicarnos con OpenFaaS, para ello debemos ejecutar los siguientes comandos en una consola normal (ver figura 7):

1. *kubectl rollout status -n openfaas deploy/gateway*
2. *kubectl port-forward -n openfaas svc/gateway 8080:8080*

```
PS $> kubectl rollout status -n openfaas deploy/gateway
deployment "gateway" successfully rolled out

PS $> kubectl port-forward -n openfaas svc/gateway 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
...
```

**Figura 7. Activación de OpenFaaS**

El último paso es generar la contraseña de usuario administrador (ver figura 8), con lo que tenemos que ejecutar estos tres comandos;

1. *PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)*
2. *echo -n $PASSWORD | faas-cli login --username admin --password-stdin*
3. *echo $PASSWORD*

```
$ PASSWORD=$(kubectl get secret -n openfaas basic-auth -o jsonpath="{.data.basic-auth-password}" | base64 --decode; echo)
...

$ echo -n $PASSWORD | faas-cli login --username admin --password-stdin
Calling the OpenFaaS server to validate the credentials...
credentials saved for admin http://127.0.0.1:8080
...

$ echo $PASSWORD
[redacted]
```

**Figura 8. Generando la contraseña del administrador.**

Ahora ya podemos loguearnos en el servicio OpenFaaS, solo tenemos que escribir la dirección *127.0.0.1:8080* con el usuario admin y la contraseña que acabamos de generar (ver figura 9).


**Figura 9. Inicio en la interfaz de OpenFaaS.**

## 3.2. Nuestra primera función en OpenFaaS

Para probar el despliegue, podemos definir una función simple que muestre un calendario. Si ajustamos los parámetros de la función a los que se ven en la siguiente figura ( ver figura 10), obtendremos los resultados una vez invocada la función que se pueden ver en la figura 11.

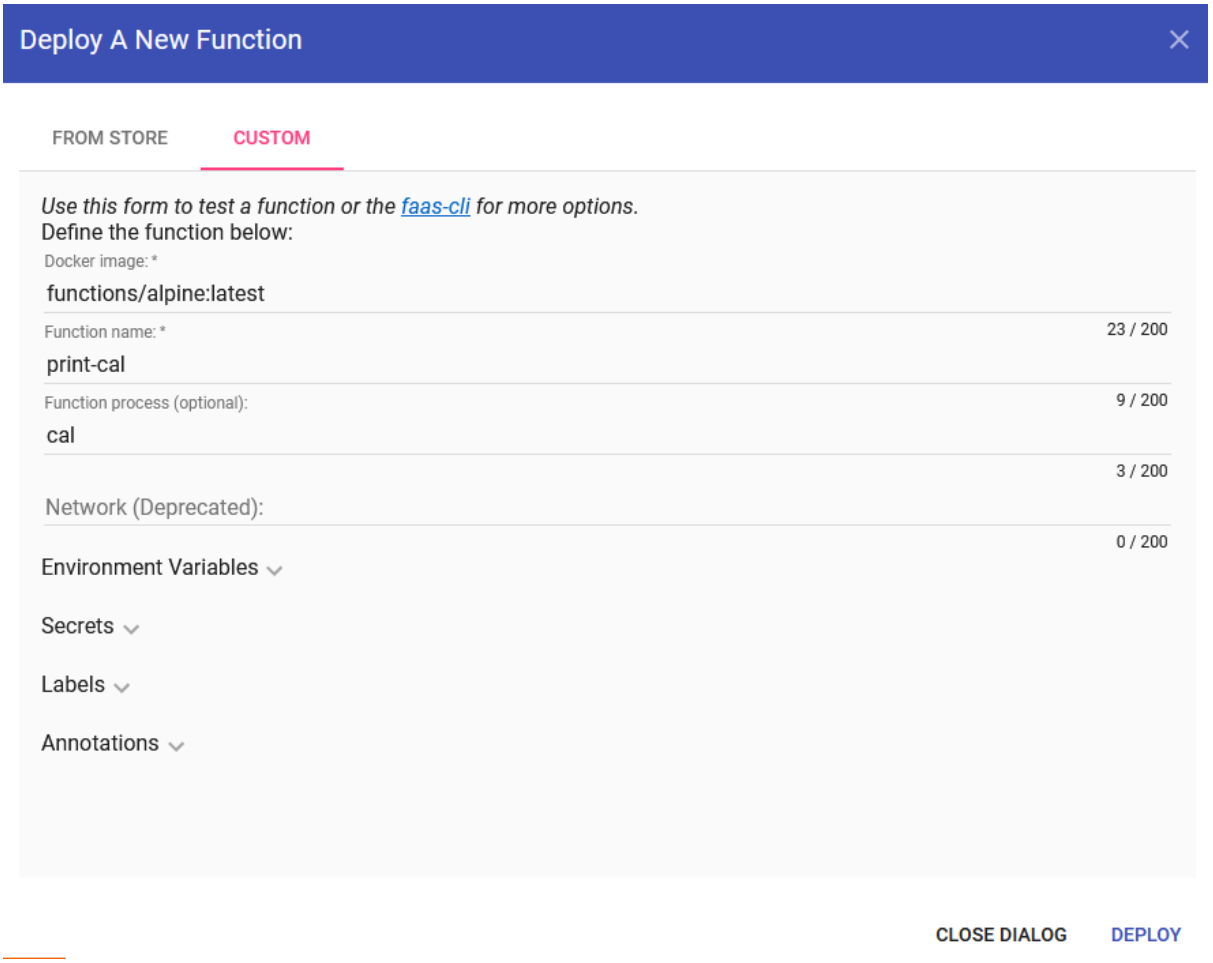


Figura 10. Despliegue de la función print-cal.

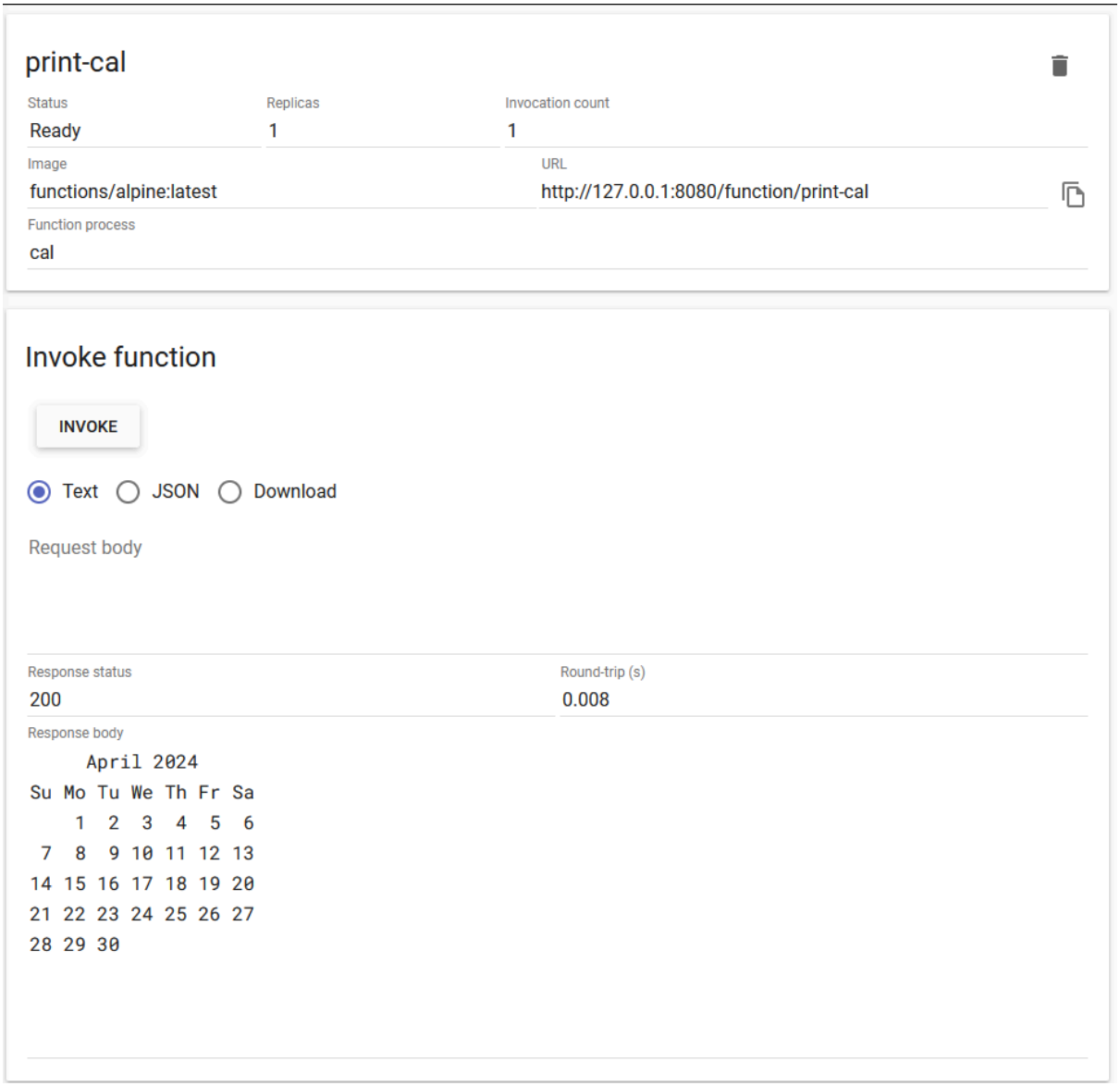


Figura 11. Prueba de la función desplegada.

# 4. Función de reconocimiento facial

## 4.1. Funciones de la store

La store de OpenFaaS tiene disponible dos funciones que implementan reconocimiento facial (ver figura 12) que se pide en la práctica.

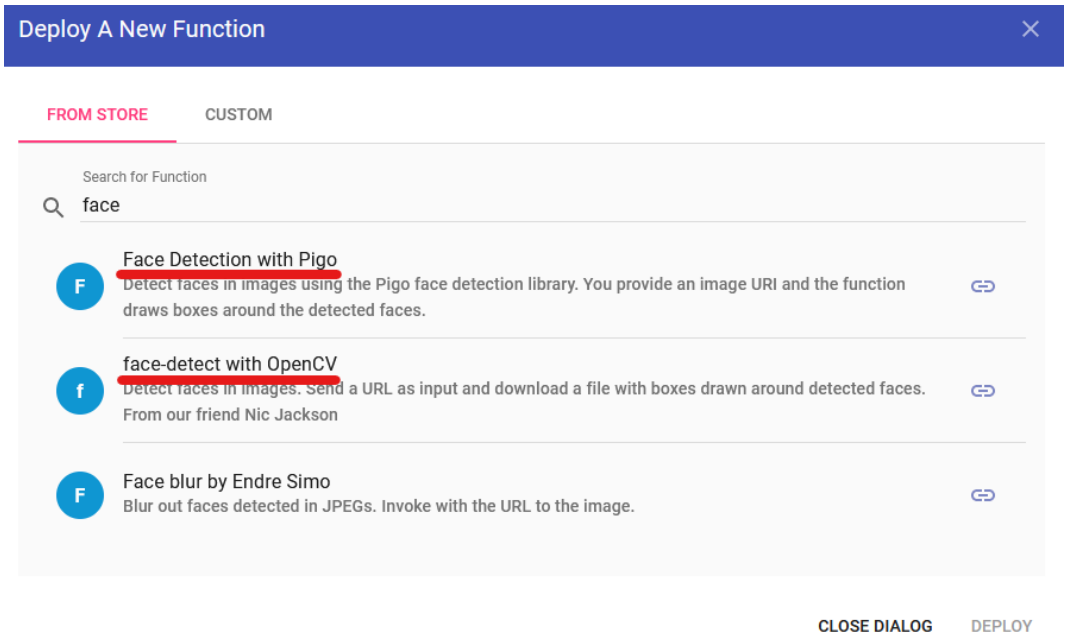


**Figura 12. Funciones de detección de caras de la store.**

El funcionamiento de ambas funciones es similar, ambas requieren un URL de una foto (ver figura 13) y como resultado, devuelven la imagen con las caras identificadas en un marco (ver figura 14).

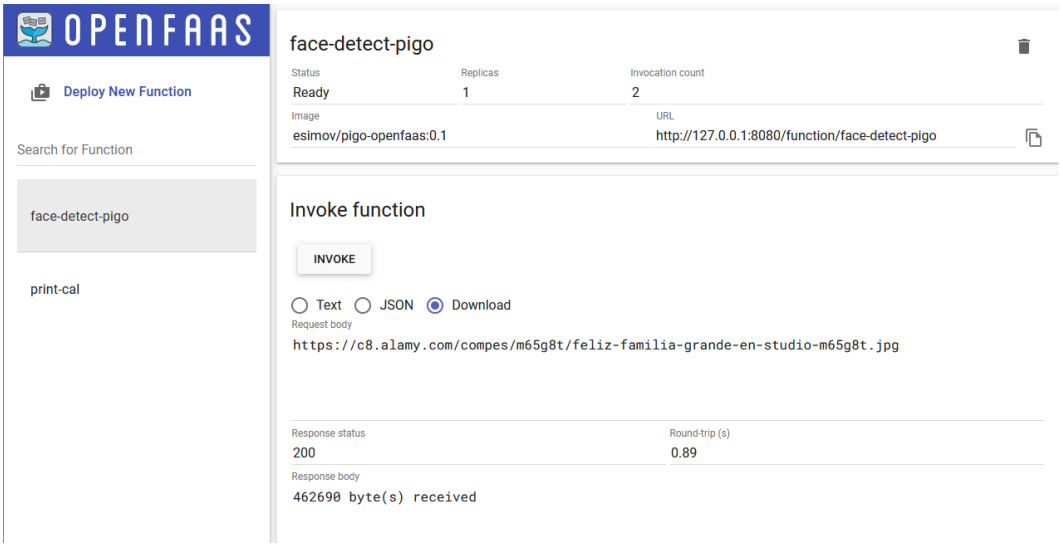


**Figura 13. Funcionamiento de la función de Pigo.**


**Figura 14. Resultado de la identificación de caras. A la izquierda Pigo y a la derecha OpenCV.**

En este caso, la función OpenCV identifica correctamente las dos caras, mientras que Pigo sólo identifica una y presenta un error en la identificación.

# 5. Conclusión

OpenFaaS representa una solución de código abierto que simplifica la ejecución de funciones en contenedores, eliminando la necesidad de gestionar la infraestructura. La integración con Kubernetes potencia su funcionalidad al aprovechar características como escalabilidad, balanceo de carga, gestión de recursos y seguridad.

En resumen, hemos aprovechado las capacidades de la tienda de Openfaas para implementar las funciones de reconocimiento facial Pigo y OpenCV, y comparando los resultados, ambas funciones pueden ser útiles para su propósito.

# 6. Bibliografía

[1]. *Welcome!* | *minikube*, https://minikube.sigs.k8s.io/docs/.

[2]. *Home* | *OpenFaaS - Serverless Functions Made Simple*, https://www.openfaas.com/.

[3]. “First Python Function.” *OpenFaaS*,

https://docs.openfaas.com/tutorials/first-python-function/.