**IES FRANCISCO DE QUEVEDO**



Proyecto final

Neuropod - PaaS con Kubernetes

18 de mayo de 2025

Zidan Marvin David Avendaño Vilchez

TutoR: CésAr López

CICLO: 2024/25.

**INDICE**

[1. Introducción 3](#_Toc199173847)

[2. Objetivos 3](#_Toc199173848)

[Objetivo principal 3](#_Toc199173849)

[Objetivos específicos 3](#_Toc199173850)

[3. Estado del Arte 3](#_Toc199173851)

[4. Planteamiento / Arquitectura del Proyecto 4](#_Toc199173852)

[Esquema del proyecto 4](#_Toc199173853)

[Esquema del clúster de minikube 4](#_Toc199173854)

[Tecnologías Utilizadas 4](#_Toc199173855)

[React + Vite (Frontend) 4](#_Toc199173856)

[Node.js + Express (Backend) 4](#_Toc199173857)

[Google OAuth2 + JWT (Autenticación) 5](#_Toc199173858)

[MongoDB (Base de datos) 5](#_Toc199173859)

[Docker (Contenedores) 5](#_Toc199173860)

[Kubernetes (Minikube) 5](#_Toc199173861)

[NGINX Ingress Controller 6](#_Toc199173862)

[Cloudflare Tunnel + DNS Wildcard 6](#_Toc199173863)

[Persistent Volume Claim (PVC) 6](#_Toc199173864)

[Documentación generada 6](#_Toc199173865)

[Arquitectura de la base de datos 6](#_Toc199173866)

[5. Desarrollo del Proyecto 7](#_Toc199173867)

[Instalación, configuraciones previas y gestión 7](#_Toc199173868)

[MongoDB 7](#_Toc199173869)

[Minikube y Docker 7](#_Toc199173870)

[Vscode y Git 8](#_Toc199173871)

[fnm y Node.js 9](#_Toc199173872)

[Google Cloud Console 9](#_Toc199173873)

[Transferir la Gestión DNS a Cloudflare 10](#_Toc199173874)

[Cloudflare Tunnel 12](#_Toc199173875)

[Base de datos (aun no terminada) 13](#_Toc199173876)

[Métodos de Autenticación 14](#_Toc199173877)

[Componentes del Sistema 14](#_Toc199173878)

[Variables de Entorno 14](#_Toc199173879)

[Flujo de Autenticación 15](#_Toc199173880)

[Cierre de Sesión (Logout) 15](#_Toc199173881)

[Control de Acceso 17](#_Toc199173882)

[Seguridad 17](#_Toc199173883)

[Desarrollo vs Producción 17](#_Toc199173884)

[Minikube 18](#_Toc199173885)

[Iniciar un entorno limpio 18](#_Toc199173886)

[Verificación inicial de Minikube 18](#_Toc199173887)

[Habilitación de complementos en Minikube 18](#_Toc199173888)

[Creación del directorio workspace 19](#_Toc199173889)

[Aplicación del manifiesto Neuropod 19](#_Toc199173890)

[Contenido del manifiesto Neuropod 19](#_Toc199173891)

[Verificación de recursos desplegados 25](#_Toc199173892)

[Esquema de la arquitectura Neuropod en Minikube 26](#_Toc199173893)

[Gestión de pods 27](#_Toc199173894)

[6. Conclusiones 27](#_Toc199173895)

[7. Base de datos 27](#_Toc199173896)

# 1. Introducción

Este proyecto consiste en diseñar e implementar una plataforma web escalable que permita ejecutar contenedores de forma remota, desde cualquier lugar, a través de un navegador. Inspirado en servicios como RunPod, busca ofrecer una solución accesible y eficiente para trabajar con modelos de generación de imágenes basados en inteligencia artificial, sin depender exclusivamente de la ejecución local. El sistema está pensado como una Plataforma como Servicio (PaaS), gestionada desde un único equipo físico, pero con una arquitectura modular basada en contenedores, autenticación segura, y acceso web mediante interfaz moderna.

# 2. Objetivos

## Objetivo principal

Desarrollar una aplicación web modular y escalable que permita a los usuarios gestionar contenedores personalizados (como ComfyUI y Ubuntu) o propios a través de una interfaz gráfica remota.

## Objetivos específicos

* Ejecutar contenedores Docker con acceso remoto mediante subdominios personalizados.
* Implementar un sistema de autenticación OAuth2 con Google.
* Gestionar sesiones seguras con JWT.
* Gestionar los pods dinámicamente desde Node.js usando la API de Kubernetes.
* Proporcionar persistencia de datos vía PVC montado en /workspace.
* Controlar el uso mediante sistema de saldo por usuario.
* Exponer los servicios mediante Cloudflare Tunnel y configurar DNS wildcard.

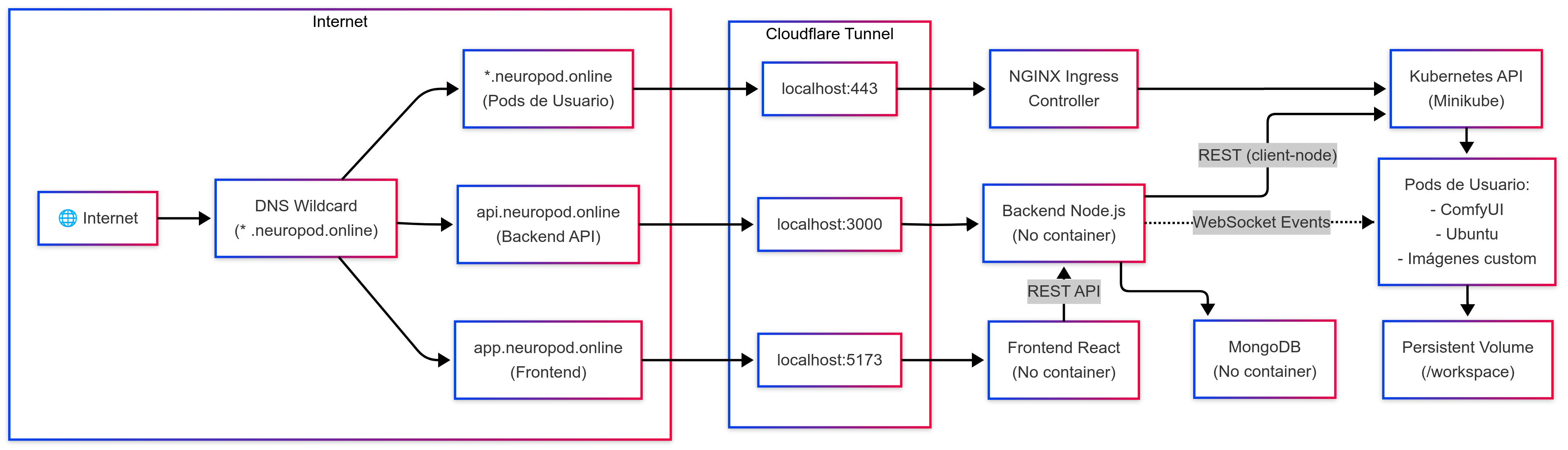
# 3. Estado del Arte

Actualmente, existen plataformas como RunPod, ComfyUI Online, y Hugging Face Spaces, que permiten ejecutar interfaces de IA desde la nube. Estas herramientas aprovechan tecnologías modernas de contenedores (Docker), orquestación (Kubernetes), e interfaces gráficas basadas en la web. Las metodologías DevOps y la computación serverless también están en auge, facilitando despliegues rápidos, escalado automático y reducción de costes de infraestructura. Sin embargo, muchas de estas soluciones requieren recursos económicos considerables o conocimientos avanzados. Este proyecto se plantea como una alternativa educativa y personal a estas soluciones comerciales, pero aplicando las mismas tecnologías utilizadas en la industria, adaptadas a un único servidor físico.

# 4. Planteamiento / Arquitectura del Proyecto

La arquitectura del proyecto sigue un enfoque estructurado en cinco fases, iniciando con la preparación del entorno mediante la instalación de MongoDB y Kubernetes. En la segunda fase, se desarrolla el backend con Node.js y Express, integrando autenticación con Google OAuth2 y JWT, además de WebSockets opcionalmente. La tercera fase aborda el frontend con React, estableciendo una interfaz que se comunica con el backend a través de Axios y WebSockets. Luego, en la fase de contenerización, se implementan Docker y Kubernetes para el despliegue, asegurando configuración de seguridad con TLS y NGINX. Finalmente, la última fase refuerza la seguridad y documentación del sistema, gestionando backups y errores, además de proporcionar instrucciones detalladas para su uso y despliegue.

## Esquema del proyecto



## Esquema del clúster de Minikube

## Tecnologías Utilizadas

### React + Vite (Frontend)

* Función: Construcción de la interfaz gráfica de usuario.
* Descripción: React es una biblioteca de JavaScript para construir interfaces dinámicas y modulares. Vite, por su parte, es una herramienta de desarrollo rápido que acelera la carga y compilación de proyectos React.
* Motivo de uso: Se eligió React por su popularidad, comunidad y eficiencia para construir interfaces reactivas. Vite fue seleccionado por su velocidad en entornos locales y su integración fluida con React, lo que permite un desarrollo ágil y moderno.

### Node.js + Express (Backend)

* Función: Servidor de la API que gestiona usuarios, sesiones, contenedores y comunicación con Kubernetes.
* Descripción: Node.js permite ejecutar JavaScript en el servidor, mientras que Express es un framework minimalista para construir servidores web.
* Motivo de uso: Node.js ofrece un entorno ligero y no bloqueante ideal para aplicaciones web en tiempo real. Express simplifica la creación de APIs REST y se integra fácilmente con librerías como google-auth-library, mongoose y jsonwebtoken.

### Google OAuth2 + JWT (Autenticación)

* Función: Sistema de autenticación de usuarios y gestión de sesiones seguras.
* Descripción: OAuth2 permite a los usuarios iniciar sesión usando su cuenta de Google. JWT (JSON Web Token) se utiliza para firmar y verificar sesiones sin almacenar datos en el servidor.
* Motivo de uso: Google OAuth2 facilita el acceso seguro sin gestionar contraseñas. JWT permite un sistema de sesiones eficiente y escalable, muy útil en aplicaciones distribuidas.

### MongoDB (Base de datos)

* Función: Almacenamiento de datos persistentes del sistema (usuarios, pods, logs, sesiones, plantillas).
* Descripción: MongoDB es una base de datos NoSQL orientada a documentos que guarda los datos en formato JSON.
* Motivo de uso: Su flexibilidad de esquemas es ideal para prototipos y sistemas que evolucionan rápidamente, como NeuroPod. Además, se integra fácilmente con Node.js mediante Mongoose.

### Docker (Contenedores)

* Función: Estandarización del entorno de ejecución de aplicaciones como ComfyUI y Ubuntu.
* Descripción: Docker permite crear contenedores ligeros y portables con todo lo necesario para ejecutar una aplicación.
* Motivo de uso: Facilita el despliegue y gestión de entornos aislados para cada usuario, permitiendo ejecutar distintas imágenes sin conflictos entre ellas.

### Kubernetes (Minikube)

* Función: Orquestación de contenedores: despliegue, monitorización, escalado y eliminación automática.
* Descripción: Kubernetes es un sistema de orquestación para contenedores. Minikube permite ejecutar un clúster Kubernetes local en un único equipo.
* Motivo de uso: Proporciona control total sobre el ciclo de vida de los contenedores. Usar Minikube simplifica el desarrollo y pruebas sin requerir infraestructura en la nube.

### NGINX Ingress Controller

* Función: Gestión del tráfico entrante hacia los pods de usuario mediante rutas dinámicas y subdominios.
* Descripción: Un Ingress Controller actúa como proxy inverso que enruta las peticiones externas al servicio Kubernetes correspondiente, usando reglas configuradas.
* Motivo de uso: Permite exponer múltiples contenedores a través de subdominios sin necesidad de asignar puertos diferentes. Es esencial para un sistema multiusuario con acceso web.

### Cloudflare Tunnel + DNS Wildcard

* Función: Exposición segura de los servicios internos al exterior usando HTTPS y subdominios automáticos.
* Descripción: Cloudflare Tunnel conecta de forma segura una máquina local a internet sin abrir puertos. El DNS wildcard permite que cualquier subdominio (\*.neuropod.online) apunte al túnel.
* Motivo de uso: Simplifica el acceso remoto, evita configuración de routers/firewalls, y garantiza seguridad TLS sin necesidad de certificados manuales. Ideal para proyectos locales expuestos temporalmente.

### Persistent Volume Claim (PVC)

* Función: Proveer almacenamiento persistente para cada pod en el directorio /workspace.
* Descripción: En Kubernetes, los PVC permiten que los datos de un contenedor persistan incluso después de que el pod sea destruido.
* Motivo de uso: Es fundamental para que los usuarios conserven sus archivos entre sesiones, permitiendo continuidad de trabajo en entornos temporales.

## Documentación generada

Gracias a la inteligencia artificial y la plataforma deepwiki se puede generar documentación sobre los repositorios públicos en Github: <https://deepwiki.com/Zidane0MA/NeuroPod>.

## Arquitectura de la base de datos

La arquitectura de la base de datos del proyecto Neuropod está centrada en el uso de MongoDB, y se ha diseñado para soportar múltiples usuarios con sesiones autenticadas, Registrar contenedores/pods desplegados por cada usuario, Permitir la persistencia de logs, transacciones económicas (uso del saldo) y plantillas reutilizables, Escalar fácilmente sin requerir esquemas rígidos (gracias a la naturaleza NoSQL).

* users — Usuarios del sistema.
* pods — Contenedores desplegados.
* sessions — Sesiones activas
* logs — Registro de acciones
* templates — Plantillas de contenedores
* transactions — Movimiento de saldo

# 5. Desarrollo del Proyecto

## Instalación, configuraciones previas y gestión

### MongoDB

Para este proyecto se usó la versión de comunidad de [MongoDB](https://www.mongodb.com/try/download/community), se puede instalar como un servicio o no, en mi caso no lo hice, por consecuencia debo arrancar manualmente el servicio, también se instaló Mongosh para crear la base de datos por comandos.

**# Crear la ruta**

**mkdir C:\**data\**db**

**# Comando para arrancar el servicio manualmente**

**"C:\Program Files\MongoDB\Server\8.0\bin\mongod.exe" --dbpath="C:\data\db"**

**# Comprobar URL en el navegador**

**mongodb://localhost:27017**

### Minikube y Docker

Para instalar Minikube en Windows utilizando el archivo ejecutable (.exe), descarga el instalador desde el sitio oficial de [Minikube](https://minikube.sigs.k8s.io/docs/start/) y ejecuta el archivo minikube-installer.exe. Sigue las instrucciones en pantalla para completar la instalación. Asegúrate de tener kubectl configurado en tu PATH y de instalar [Docker Desktop](https://www.docker.com/products/docker-desktop/) para Windows, ya que Minikube utilizará Docker como controlador.

Antes de iniciar con la configuración de Minikube, debemos verificar que Docker tiene acceso a nuestra GPU, como voy a poner a disposición mi gráfica nvidia, el comando es:

docker run --rm --gpus all nvidia/cuda:12.0.1-base-ubuntu22.04 nvidia-smi

docker info | Select-String "Runtimes" # Verificar instalación

Para asegurarte de que tu cliente de kubernetes (kubectl) está correctamente instalado y es compatible con tu clúster de Kubernetes, puedes verificar su versión.

**kubectl version –cli**en**t** **# Verificar kubectl**

Minikube permite ejecutar un clúster de Kubernetes localmente. Usar Docker como controlador es eficiente para gestionar contenedores. Para iniciar Minikube con Docker, usa el comando:

**minikube start --driver=Docker** **# Arrancar temporalmente**

Una vez que tu clúster de Kubernetes está en funcionamiento, debemos configurar el entorno para trabajar.

# Acceder a Minikube y crear la ruta para workspace

minikube ssh

mkdir -p /mnt/data/workspace

exit

# Habilitar addons

minikube addons enable ingress

minikube addons enable storage-provisioner

minikube addons enable default-storageclass

# Acceso a GPU en Kubernetes

minikube addons enable nvidia-gpu-device-plugin

# Instalar el device plugin de NVIDIA para Minikube

kubectl apply -f [https://raw.githubusercontent.com/NVIDIA/k8s-device-](https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.14.1/nvidia-device-plugin.yml) plugin/v0.14.1/nvidia-device-plugin.yml

# Verificar que el complemento se desplego

kubectl get pods -n kube-system | Select-String "nvidia"

# Iniciar Minukibe con los addons

minikube start --gpus all --driver=docker --addons=ingress

Necesitarás gestionar y supervisar sus recursos. Aquí hay algunos comandos clave:

**kubectl get nodes** **# Verificar nodo activo**

**kubectl get pods -A** **# Ver pods**

**kubectl get deployments** **# Ver deployments**

**kubectl describe pod NOMBRE** **# Describir un pod**

**kubectl apply -f archivo.yaml** **# Aplicar Manifiesto**

**minikube stop** **# Parar minikube**

**Stop-Service -Name com.docker.service** **# Parar Docker**

### Vscode y Git

Para instalar [Visual Studio Code](https://code.visualstudio.com/Download) en Windows, descarga el instalador desde su sitio oficial, ejecútalo y sigue las instrucciones. Acepta los términos, selecciona tus preferencias y haz clic en "Instalar". Luego, abre Visual Studio Code desde el menú Inicio y añade extensiones desde el Marketplace para mejorar tu entorno:

**Docker** **// Gestionar Docker**

**ESlint** **// Detecta errores de sintaxis**

**Prettier** **// Formatear código**

**npm IntelliSense** **// Autocompletado**

**Thunder Client** **// Probar API Rest**

Además, instala [Git](https://git-scm.com/downloads) descargando el instalador desde su sitio oficial, ejecútalo y sigue las instrucciones, asegurándote de agregar Git al PATH de Windows.

### fnm y Node.js

Fast Node Manager es un gestor de versiones de Node.js rápido y sencillo, construido en Rust. Permite instalar, desinstalar y cambiar entre diferentes versiones de Node.js de manera eficiente.

Antes de instalar fnm, para permitir la ejecución de scripts en PowerShell y asegurarte de que tu entorno está correctamente configurado, sigue estos pasos:

**Set-ExecutionPolicy RemoteSigned** **# Cambiar Política PowerShell**

**$PROFILE** **# Ver la ruta de $PROFILE**

**Microsoft.PowerShell\_profile.ps1** **# Crear $PROFILE e ingresar:**

**fnm env --use-on-cd --shell powershell | Out-String | Invoke-Expression**

Para instalar fnm y Node.js en PowerShell, sigue estos pasos:

**winget install Schniz.fnm** **# Instalar fnm (Gestor de versiones)**

**fnm install 22** **# Instalar node "v22.15.0"**

**fnm use 22** **# Usar la versión "v22.15.0"**

**node -v**  **# Versión de node**

**npm -v**  **# Versión del gestor de paquetes**

### Google Cloud Console

Es la plataforma para gestionar la configuración de OAuth, asegura que solo las aplicaciones autorizadas puedan acceder a los datos del usuario. Utilizando OAuth, puedes autenticar usuarios y obtener tokens de acceso para interactuar con las APIs de Google. Para este paso no es necesario comprar un dominio todavía.

* En [console.cloud.google.com](https://console.cloud.google.com/), en la sección *APIs y servicios* > *Credenciales*.
* Crear una credencial OAuth: Seleccionar *Crear credencial* y elige *ID de cliente OAuth*.
* Configurar la pantalla de consentimiento:
  + Define el tipo de usuario como *Externo*.
* Crear la credencial OAuth:
  + Seleccionar *Aplicación Web* como tipo de aplicación.
  + Asignar el nombre *NeuroPod Web Client*.
  + Especificar los *Orígenes JavaScript autorizados*:
    - http://localhost:5173
    - https://app.neuropod.online
  + Definir las *URLs de redirección autorizadas*:
    - http://localhost:5173
    - http://localhost:5173/login
    - https://app.neuropod.online
    - https://app.neuropod.online/login
* Agregar usuarios de prueba: Ve a Público y añade los usuarios de prueba.
* Obtener credenciales:
  + *ID CLIENTE*
  + *SECRETO DE CLIENTE*

### Transferir la Gestión DNS a Cloudflare

#### 1. Crear una Cuenta en Cloudflare

Para comenzar a gestionar tu dominio desde Cloudflare, primero debes crear una cuenta y añadir tu sitio.

* Accede a <https://cloudflare.com> y regístrate o inicia sesión.
* Haz clic en “Agregar un sitio” (o “Add a site”) en el panel de control.
* Introduce tu dominio (por ejemplo: neuropod.online) y haz clic en “Agregar sitio”.
* Selecciona el plan gratuito, que es suficiente para la mayoría de proyectos.

#### 2. Configurar los Servidores DNS en Hostinger

Después de añadir tu dominio en Cloudflare, la plataforma te mostrará los servidores DNS que debes configurar en tu proveedor actual (en este caso, Hostinger). Por ejemplo:

* brad.ns.cloudflare.com
* ella.ns.cloudflare.com

Pasos para actualizar los DNS en Hostinger u otro servicio:

* Inicia sesión en tu cuenta de Hostinger.
* Ve a la sección "Dominios" y selecciona tu dominio.
* Busca la opción "Servidores de nombres" o “Nameservers”.
* Sustituye los servidores de nombres actuales por los proporcionados por Cloudflare.
* Guarda los cambios.

Nota: La propagación de los DNS puede tardar entre 24 a 48 horas. Durante este período, Cloudflare mostrará el estado como “Pendiente”.

#### 3. Completar la Configuración en Cloudflare

Una vez cambiados los servidores DNS en Hostinger, regresa a Cloudflare para verificar y finalizar la configuración:

* Haz clic en “Verificar servidores de nombres” (Check nameservers).
* Cloudflare escaneará tus registros DNS actuales e importará los más relevantes.
* Revisa los registros importados y haz clic en “Continuar”.

Configuración rápida recomendada:

* SSL/TLS: Full (o Full (Strict) si usas certificados propios).
* Siempre usar HTTPS: Activado.
* Auto Minify: Activado (opcional, pero recomendado).
* Brotli: Activado.
* Rocket Loader: Desactivado (puede causar problemas con algunas apps).
* Haz clic en “Guardar” (Save) para finalizar.

#### 4. Crear Registros DNS Básicos

Una vez tu dominio esté activo en Cloudflare, es momento de añadir los registros DNS necesarios para que tus subdominios apunten al túnel que crearás más adelante.

Ve a la sección “DNS” dentro del panel de control de Cloudflare y añade los siguientes registros:

* Registro para API (Backend):
  + Tipo: CNAME
  + Nombre: api
  + Destino: (se completará más adelante con el UUID del túnel)
  + Proxy status: Activado (Proxied)
  + TTL: Auto
* Registro para App (Frontend):
  + Tipo: CNAME
  + Nombre: app
  + Destino: (se completará más adelante con el UUID del túnel)
  + Proxy status: Activado (Proxied)
  + TTL: Auto
* Registro Wildcard para Pods:
  + Tipo: CNAME
  + Nombre: \*
  + Destino: (se completará más adelante con el UUID del túnel)
  + Proxy status: Activado (Proxied)
  + TTL: Auto

Nota: Mientras terminas de configurar el túnel, puedes dejar los destinos apuntando a example.com o a un dominio temporal. Luego los actualizaremos con el ID real del túnel generado por Cloudflare.

### Cloudflare Tunnel

Para utilizar Cloudflare Tunnel en Windows, primero necesitas instalar el binario cloudflared. Esto se puede hacer fácilmente con el gestor de paquetes winget.

winget install --id Cloudflare.cloudflared

cloudflared.exe –version # Verificar instalación

Antes de crear el túnel, debes autenticarte con tu cuenta de Cloudflare. Esto abrirá una pestaña en tu navegador para autorizar el acceso a tu cuenta. Sigue las instrucciones en pantalla y selecciona el dominio que deseas usar (por ejemplo: neuropod.online).

cloudflared.exe tunnel login

Una vez autenticado, crea un túnel con un nombre identificativo. Esto generará un archivo de credenciales (JSON) en tu carpeta .cloudflared.

cloudflared.exe tunnel create neuropod-tunnel

Este comando mostrará el **UUID** del túnel, que será necesario más adelante, también mostrara la ruta donde se creó el JSON (C:\Users\<USER>\.cloudflared\<UUID>.json)

El archivo de configuración se utiliza para definir cómo se enrutan los subdominios a los servicios locales. Ábrelo con el Bloc de notas desde PowerShell:

notepad $env:USERPROFILE\.cloudflared\config.yml

Ingresa el siguiente contenido adaptando tu nombre (<USER>) y el **UUID** del túnel:

tunnel: neuropod-tunnel

credentials-file: C:\Users\<USER>\.cloudflared\<UUID>.json

ingress:

# Frontend React

- hostname: app.neuropod.online

service: **http://localhost:5173**

# Backend API

- hostname: api.neuropod.online

service: **http://localhost:3000**

# DNS Wildcard para los pods de usuario

- hostname: "\*.neuropod.online"

service: **http://localhost:443**

originRequest:

noTLSVerify: true

# Configuración para WebSockets (Jupyter Lab)

connectTimeout: 30s

tlsTimeout: 30s

tcpKeepAlive: 30s

disableChunkedEncoding: true

# Configuración para tokens de acceso y Jupyter Lab

http2Origin: false

# Fallback

- service: http\_status:404

Nota: Los puertos especificados (5173, 3000, 443) deben coincidir con los puertos donde se ejecutan tus servicios localmente.

Finalmente, ejecuta el túnel utilizando el nombre definido previamente. Si todo está correctamente configurado, el túnel establecerá conexión con Cloudflare y podrás acceder a tus servicios a través de los subdominios definidos (app.neuropod.online, api.neuropod.online, etc.). Los servicios locales deben estar ejecutándose.

cloudflared.exe tunnel run neuropod-tunnel

## Base de datos (aun no terminada)

La creación de la base de datos se maneja desde mongosh

Crear la base de datos (Mongosh):

**use plataforma**

**db.createCollection("users")**

**# Crear índices para usuarios**

**db.users.createIndex({ "email": 1 }, { unique: true })**

**db.users.createIndex({ "googleId": 1 })**

**db.users.createIndex({ "role": 1 })**

**db.createCollection("pods")**

**# Crear índices para pods**

**db.pods.createIndex({ "userId": 1 })**

**db.pods.createIndex({ "containerId": 1 }, { unique: true, sparse: true })**

**db.pods.createIndex({ "status": 1 })**

**db.pods.createIndex({ "createdAt": 1 })**

**db.createCollection("sessions")**

**# Crear índices para sesiones y TTL (expiración automática)**

**db.sessions.createIndex({ "token": 1 }, { unique: true })**

**db.sessions.createIndex({ "userId": 1 })**

**db.sessions.createIndex({ "expiresAt": 1 }, { expireAfterSeconds: 0 }**

**db.createCollection("logs")**

**# Crear índices para logs**

**db.logs.createIndex({ "userId": 1 })**

**db.logs.createIndex({ "action": 1 })**

**db.logs.createIndex({ "timestamp": 1 })**

**db.logs.createIndex({ "timestamp": 1, "userId": 1 })**

**db.createCollection("templates")**

**# Crear índices para templates**

**db.logs.createIndex({ "userId": 1 })**

## Métodos de Autenticación

NeuroPod soporta dos métodos de autenticación:

#### Google OAuth2 (Recomendado para producción)

* Utiliza el estándar OAuth2 de Google para autenticar usuarios.
* Requiere que los usuarios tengan una cuenta de Google.
* Proporciona un alto nivel de seguridad y confiabilidad.

#### Autenticación Simulada (Solo para desarrollo)

* Permite iniciar sesión con cualquier correo electrónico sin verificación real.
* Solo disponible cuando NODE\_ENV=development.
* Útil para pruebas rápidas en entorno de desarrollo.

### Componentes del Sistema

#### Frontend

* Utiliza el paquete @react-oauth/google para la integración con Google OAuth2.
* Implementa un flujo de autenticación implícito, sin redirecciones.
* Almacena los tokens JWT en localStorage para mantener la sesión activa.

#### Backend

* Verifica los tokens de Google mediante la API oficial.
* Genera tokens JWT propios para mantener las sesiones.
* Implementa lógica de verificación de roles y permisos.
* Gestiona la creación y actualización de usuarios en la base de datos.

### Variables de Entorno

#### Frontend (.env):

**# URL del Backend API**  
**VITE\_API\_URL=http://localhost:3000**  
**# Google OAuth Client ID (desde Google Cloud Console)**  
**VITE\_GOOGLE\_CLIENT\_ID=tu\_google\_client\_id\_aqui**

#### Backend (.env):

**# Entorno**  
**NODE\_ENV=production** **// development** **o production**  
**# JWT**  
**JWT\_SECRET=tu\_clave\_secreta\_aqui**  
**JWT\_EXPIRE=24h**  
**# Google OAuth**  
**GOOGLE\_CLIENT\_ID=tu\_google\_client\_id\_aqui**  
**GOOGLE\_CLIENT\_SECRET=tu\_google\_client\_secret\_aqui**  
**# Control de acceso**  
**TRUST\_GOOGLE\_AUTH=true**

ALLOWED\_EMAILS=correoadmin@gmail.com,correouser@gmail.com

ADMIN\_EMAILS=correoadmin@gmail.com

### Flujo de Autenticación

#### Login / Signup con Google:

1. El usuario hace clic en el botón de Google en la página de acceso.
2. Google muestra el selector de cuentas.
3. Al seleccionar una cuenta, se obtiene un token de ID.
4. El frontend envía este token al backend mediante **/api/auth/google**.
5. El backend verifica el token con Google.
6. Si el usuario no existe, se crea; si ya existe, se actualiza su información.
7. El backend genera un JWT propio y lo devuelve al frontend.
8. El frontend almacena el JWT y redirecciona al *dashboard*.

#### Verificación de Sesión:

1. Al cargar la aplicación, se comprueba si hay un JWT en localStorage.
2. Si existe, se envía al backend a través de /api/auth/verify.
3. Si el token es válido, se carga la información del usuario.
4. Si no es válido, se redirige automáticamente al login.

### Cierre de Sesión (Logout)

El sistema implementa un cierre de sesión seguro que:

1. **Elimina el token JWT en el servidor:**
   1. Borra la sesión en la base de datos.
   2. Registra el evento en el log de acciones.
2. **Limpia los datos en el cliente:**
   1. Elimina el token y la información del usuario desde el localStorage.
   2. Restablece el estado de la aplicación.
3. **Proporciona retroalimentación visual:**
   1. Muestra un mensaje de confirmación al usuario.
   2. Redirige al login.
4. **Gestiona errores de forma segura:**
   1. Incluso si hay problemas con el servidor, limpia los datos locales.
   2. Evita dejar al usuario en un estado inconsistente.

#### Frontend (AuthContext.tsx):

**const logout = async () => {**

**try {**

**// Intentar logout en el servidor si hay conexión**

**if (!isOfflineMode) {**

**await authService.logout();**

**}**

**// Limpiar datos locales**

**setUser(null);**

**localStorage.removeItem("token");**

**localStorage.removeItem("user");**

**// Feedback visual y redirección**

**toast({ title: "Sesión cerrada", ... });**

**navigate("/login");**

**} catch (error) {**

**// Manejar errores pero seguir con el cierre local**

**console.error('Error durante logout:', error);**

**// Limpiar datos de todas formas**

**setUser(null);**

**localStorage.removeItem(...)**

**}**

**};**

#### Backend (auth.controller.js):

**exports.logout = async (req, res) => {**

**try {**

**// Obtener token de diferentes fuentes**

**let token = req.body.token || req.token || (req.headers.authorization**   **&& req.headers.authorization.split(' ')[1]);**

**// Eliminar sesión de la base de datos**

**await Session.deleteOne({ token });**

**// Registrar evento**

**if (req.user) {**

**await logAction(req.user.\_id, 'LOGOUT');**

**}**

**res.status(200).json({ success: true, message: 'Sesión cerrada**   **correctamente' });**

**} catch (error) {**

**// Manejar errores**

**}**

**};**

### Control de Acceso

#### Verificación de Usuario

En producción, el sistema puede comportarse de dos maneras:

1. **Confiar exclusivamente en Google OAuth** (TRUST\_GOOGLE\_AUTH=true)
   1. Se permite acceso a cualquier cuenta autenticada con Google.
   2. Los roles se asignan según la lista ADMIN\_EMAILS.
2. **Verificar una lista interna de permitidos** (TRUST\_GOOGLE\_AUTH=false)
   1. Aparte de autenticarse con Google, el correo debe estar en ALLOWED\_EMAILS.
   2. Aumenta la seguridad añadiendo un segundo filtro.

#### Asignación de Roles

* Los correos incluidos en ADMIN\_EMAILS reciben rol de administrador.
* El resto de los usuarios obtienen rol de cliente.
* Los roles se actualizan automáticamente al iniciar sesión.

### Seguridad

* Los tokens JWT tienen caducidad, según lo definido en JWT\_EXPIRE.
* Todas las rutas protegidas validan el token y permisos del usuario.
* No se almacenan contraseñas ni datos sensibles en el frontend.

### Desarrollo vs Producción

* En **desarrollo** (NODE\_ENV=development), el sistema es permisivo y acepta autenticación simulada.
* En **producción** (NODE\_ENV=production), se aplican todas las restricciones de seguridad.

## Minikube

### Iniciar un entorno limpio

minikube delete

Si el estado es Running, el clúster está listo para usarse. Este paso garantiza que las siguientes instrucciones se apliquen sobre un entorno Kubernetes operativo

[Compatibilidad con GPU en Kubernetes de Docker para escritorio en Windows 11 con WSL 2 - Stack Overflow](https://stackoverflow.com/questions/77875365/gpu-support-on-kubernetes-of-docker-for-desktop-on-windows-11-with-wsl-2?rq=2)

### Verificación inicial de Minikube

Antes de desplegar cualquier recurso, se comprueba que el clúster de Minikube está activo y funcionando. Esto se hace con el comando:

minikube status

Si el estado es *Running*, el clúster está listo para usarse. Este paso garantiza que las siguientes instrucciones se apliquen sobre un entorno Kubernetes operativo.

### Habilitación de complementos en Minikube

Minikube incluye varios *addons* útiles que se deben activar para soportar Neuropod. En primer lugar, se activa el controlador Ingress (NGINX) y el provisionador de almacenamiento:

minikube addons enable ingress  
minikube addons enable storage-provisioner  
minikube addons enable default-storageclass

Estos comandos habilitan, respectivamente, el controlador de entrada (para enrutar tráfico HTTP/HTTPS) y los complementos de almacenamiento que permiten crear PersistentVolumes usando el host. Si el nodo Minikube tiene una GPU disponible y se quiere utilizar aceleración por hardware, también conviene activar el plugin de Nvidia:

minikube addons enable nvidia-device-plugin

Este complemento instala los controladores necesarios para exponer las GPUs al clúster.

### Creación del directorio workspace

Neuropod utiliza un volumen persistente en el nodo para almacenar los workspaces de los usuarios. Se crea la carpeta en el nodo de Minikube y se ajustan permisos:

minikube ssh "sudo mkdir -p /mnt/data/workspace && sudo chmod 777 /mnt/data/workspace"

Con este comando se ingresa por SSH al nodo de Minikube y se crea el directorio /mnt/data/workspace (si no existía), asignándole permisos abiertos. Este directorio servirá como ruta *hostPath* para el PersistentVolume neuropod-pv-global definido en el manifiesto.

### Aplicación del manifiesto Neuropod

Una vez preparados Minikube y sus complementos, se aplica el manifiesto de Kubernetes que contiene todos los recursos de Neuropod. Se asume que el contenido YAML de configuración se ha guardado en un archivo llamado neuropod-k8s.yaml. Para desplegarlo, se ejecuta:

kubectl apply -f neuropod-k8s.yaml

Este comando crea en Kubernetes los objetos descritos en el archivo. Es importante aplicarlos todos juntos (ya que el manifiesto contiene múltiples recursos separados por ---) y en el orden indicado por dependencias (por ejemplo, el StorageClass antes del PersistentVolume).

### Contenido del manifiesto Neuropod

El manifiesto neuropod-k8s.yaml declara varios recursos fundamentales. A continuación, se describen los principales:

* **ConfigMap neuropod-config:** almacena la configuración global del proyecto. Entre sus datos clave están:
  + domain: "neuropod.online" – dominio base usado para generar subdominios de acceso a los pods de usuario.
  + defaultStorageClass: "standard" – la clase de almacenamiento por defecto (referencia al StorageClass definido abajo).
  + maxPodsPerUser: "5" – límite de pods concurrentes por usuario.
  + workspacePath: "/workspace" – ruta interna dentro del contenedor donde se montará el volumen compartido.
  + defaultNamespace: "default" – namespace donde se crearán los pods de usuario.

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: neuropod-config  
data:  
 domain: "neuropod.online"  
 defaultStorageClass: "standard"  
 maxPodsPerUser: "5"  
 workspacePath: "/workspace"  
 defaultNamespace: "default"

Estos valores son leídos por el código de Neuropod en tiempo de ejecución para configurar nombres de dominio, rutas de volumen y límites operativos.

* **IngressClass neuropod-nginx:** define una clase de Ingress específica para Neuropod y la marca como predeterminada. Indica que el controlador a usar es ingress-nginx (NGINX Ingress Controller). Con la anotación is-default-class: "true", cualquier Ingress sin clase explícita usará esta:

apiVersion: networking.k8s.io/v1  
kind: IngressClass  
metadata:  
 name: neuropod-nginx  
 annotations:  
 ingressclass.kubernetes.io/is-default-class: "true"  
spec:  
 controller: k8s.io/ingress-nginx

Esto permite enrutar tráfico HTTP(S) entrante hacia los pods de Neuropod usando subdominios bajo el dominio base.

* **ConfigMap nginx-configuration:** se crea en el namespace ingress-nginx para personalizar el comportamiento del Ingress Controller. Contiene ajustes como server-name-hash-bucket-size para nombres largos de servidor, timeouts para WebSockets, habilitar el uso de cabeceras Forwarded (útil con túneles como Cloudflare), y configuración SSL segura (protocolos TLS y cifrados). Ejemplo simplificado de sus datos:

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: nginx-configuration  
 namespace: ingress-nginx  
data:  
 server-name-hash-bucket-size: "256"  
 proxy-read-timeout: "3600"  
 use-forwarded-headers: "true"  
 ssl-protocols: "TLSv1.2 TLSv1.3"  
 ssl-ciphers: "ECDHE-ECDSA-AES128-GCM-SHA256:..."

Estas configuraciones optimizan el servidor NGINX para largos nombres de host y sesiones de Jupyter Lab (que pueden usar WebSockets), y aplican restricciones modernas de TLS para seguridad.

* **StorageClass standard:** un *provisioner* basado en hostPath para proveer volúmenes persistentes. Define k8s.io/minikube-hostpath como provisionador, una política de retención *Retain*, y permite expansión de volúmenes. Ejemplo simplificado:

apiVersion: storage.k8s.io/v1  
kind: StorageClass  
metadata:  
 name: standard  
provisioner: k8s.io/minikube-hostpath  
reclaimPolicy: Retain  
allowVolumeExpansion: true  
parameters:  
 type: "hostPath"

Esta clase se usa por defecto según el ConfigMap anterior, y sirve para crear volúmenes montados en el host de Minikube.

* **PersistentVolume neuropod-pv-global:** volumen físico que provee almacenamiento de 500Gi en el nodo de Minikube. Está vinculado al StorageClass standard y monta el path /mnt/data/workspace del host (que creamos antes). Usa accessModes: ReadWriteMany, lo que permite que múltiples pods lean y escriban simultáneamente. Un fragmento clave:

apiVersion: v1  
kind: PersistentVolume  
metadata:  
 name: neuropod-pv-global  
spec:  
 capacity:  
 storage: 500Gi  
 accessModes:  
 - ReadWriteMany  
 storageClassName: standard  
 hostPath:  
 path: /mnt/data/workspace  
 type: DirectoryOrCreate

Este PV ofrece un área de trabajo común en el nodo para los contenedores de usuarios, alineada con workspacePath: "/workspace".

* **ServiceAccount neuropod-gpu-sa y ClusterRoleBinding neuropod-gpu-crb:** permiten que los pods que necesitan acceso a GPU obtengan permisos adecuados. Se crea una cuenta de servicio en el namespace default:

apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: neuropod-gpu-sa

Luego se liga a un rol de edición preexistente (edit) con ámbito de clúster:

apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: neuropod-gpu-crb  
roleRef:  
 kind: ClusterRole  
 name: edit  
subjects:  
- kind: ServiceAccount  
 name: neuropod-gpu-sa  
 namespace: default

Así, cualquier pod que use neuropod-gpu-sa heredará permisos amplios sobre recursos del clúster, necesarios para la GPU.

* **Namespace neuropod-system:** un namespace adicional (opcional) reservado para recursos de sistema internos de Neuropod. No se definen objetos específicos en este manifiesto más allá de crear el namespace:

apiVersion: v1  
kind: Namespace  
metadata:  
 name: neuropod-system

Esto puede usarse para alojar futuros servicios internos o controladores del proyecto.

* **NetworkPolicy neuropod-isolation:** restringe la comunicación entre pods de diferentes usuarios. Selecciona todos los pods con etiqueta app: neuropod-user-pod y aplica reglas de ingreso (Ingress) y egreso (Egress). En el manifiesto se especifica:
  + Ingreso permitido solo desde el namespace del Ingress-NGINX (controlador de entrada) y desde pods con la misma etiqueta neuropod-user: "same-user", es decir, del mismo usuario.
  + Egreso permitido en cualquier destino (- {} indica todo el tráfico saliente).  
     Fragmento relevante:

apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
 name: neuropod-isolation  
spec:  
 podSelector:  
 matchLabels:  
 app: neuropod-user-pod  
 policyTypes:  
 - Ingress  
 - Egress  
 ingress:  
 - from:  
 - namespaceSelector:  
 matchLabels:  
 name: ingress-nginx  
 - from:  
 - podSelector:  
 matchLabels:  
 neuropod-user: "same-user"  
 egress:  
 - {}

Con esto, se asegura que los pods de un usuario sólo sean accesibles a través del Ingress (p. ej. Jupyter Lab accedido via HTTPS) y no puedan comunicarse libremente con pods de otros usuarios.

* **Secret neuropod-tls:** contiene un certificado TLS para comunicaciones internas seguras bajo el dominio \*.neuropod.local. En el YAML original los campos tls.crt y tls.key aparecen como cadenas base64 largas (omitidas aquí). Si se prefiere generar el certificado aparte, el paso 8 del setup muestra cómo crearlo manualmente. En resumen, este Secret (tipo kubernetes.io/tls) permite que servicios internos de Neuropod usen HTTPS.

apiVersion: v1

kind: Secret

metadata:

name: neuropod-tls

namespace: default

labels:

app: neuropod

component: security

type: kubernetes.io/tls

data:

tls.crt: LS0tLS1CRUdJTi... # Base64 encoded certificate

tls.key: LS0tLS1CRUdJTi... # Base64 encoded private key

* **ConfigMap neuropod-scripts:** incluye scripts auxiliares usados por Neuropod dentro de los contenedores de usuario. Contiene, por ejemplo:
  + install-jupyter.sh: para instalar JupyterLab si no está presente.
  + setup-workspace.sh: crea carpetas /workspace/notebooks, /workspace/data y /workspace/models en el volumen compartido.
  + health-check.sh: verifica la disponibilidad de Jupyter usando un curl local.  
    Extracto:

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: neuropod-scripts  
data:  
 install-jupyter.sh: |  
 #!/bin/bash  
 pip install jupyterlab  
 setup-workspace.sh: |  
 #!/bin/bash  
 mkdir -p /workspace/notebooks /workspace/data /workspace/models  
 health-check.sh: |  
 #!/bin/bash  
 curl -f http://localhost:$1/health || exit 1

Estos scripts se montan en los pods de usuario y simplifican tareas comunes de inicialización.

### Verificación de recursos desplegados

Después de aplicar el manifiesto, es recomendable comprobar que Kubernetes ha creado cada recurso correctamente. Se pueden usar comandos como:

kubectl get configmaps  
kubectl get storageclass  
kubectl get pv  
kubectl get serviceaccount neuropod-gpu-sa  
kubectl get clusterrolebinding neuropod-gpu-crb

Estos comandos listan los ConfigMaps, StorageClasses y PersistentVolumes disponibles, así como la cuenta de servicio y el enlace de rol para GPU, para confirmar que existen. También conviene revisar el estado del controlador Ingress:

kubectl get pods -n ingress-nginx  
kubectl get configmap nginx-configuration -n ingress-nginx

Aquí se verifica que el pod del Ingress-NGINX esté en ejecución y que el ConfigMap con parámetros (nginx-configuration) esté presente en su espacio de nombres.

Si el manifiesto no incluye el certificado TLS, puede generarse manualmente con *openssl* y luego crearse el Secret:

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \  
 -keyout tls.key -out tls.crt -subj "/CN=\*.neuropod.local"  
kubectl create secret tls neuropod-tls --key tls.key --cert tls.crt

Esto produce un certificado autofirmado para \*.neuropod.local y lo almacena en Kubernetes. Finalmente, para verificar el despliegue completo:

kubectl get all --selector=app=neuropod  
kubectl describe ingressclass neuropod-nginx

Se debe ver que los pods y servicios con etiqueta app=neuropod estén desplegados, y que la IngressClass neuropod-nginx esté definida correctamente. Opcionalmente, se pueden revisar los logs del Ingress-NGINX para diagnóstico:

kubectl logs -n ingress-nginx deployment/ingress-nginx-controller

### Esquema de la arquitectura Neuropod en Minikube

A continuación se presenta un diagrama textual que ilustra la arquitectura general de los componentes desplegados. En un único nodo de Minikube, el Ingress-NGINX dirige el tráfico externo basado en subdominios hacia los pods de usuario, que montan el volumen compartido del PV global. El NetworkPolicy limita la comunicación entre estos pods. El esquema es el siguiente:

+---------------------------------------------------------+  
| Minikube VM |  
| |  
| +----------------------+ +---------------------+ |  
| | Ingress Controller |<-----| User Pod (jupyter) | |  
| | (NGINX, clase | | App=neuropod- | |  
| | neuropod-nginx) |----->| user-pod | |  
| | (\*.neuropod.online) | +---------------------+ |  
| +----------------------+ | | |  
| | ^ | | |  
| | | HTTP/S | | |  
| v | | | |  
| +----------------------+ +---------------------+ |  
| | User Pod (GPU app) | | PersistentVolume | |  
| | App=neuropod-user-pod|<-----| neuropod-pv-global | |  
| +----------------------+ | (hostPath=/mnt/ | |  
| | data/workspace) | |  
| +---------------------+ |  
| |  
| NetworkPolicy: permite tráfico sólo desde Ingress |  
| y entre pods del mismo usuario; todo egreso está |  
| permitido. |  
+---------------------------------------------------------+

Este esquema muestra cómo el controlador Ingress recibe peticiones para \*.neuropod.online y las envía a los pods correspondientes. Cada pod de usuario monta el PersistentVolume neuropod-pv-global para acceder a /workspace. El NetworkPolicy neuropod-isolation garantiza que sólo Ingress (y pods del mismo usuario) puedan llegar a estos pods. Otros objetos (ConfigMaps, Secrets, ServiceAccounts) se distribuyen internamente conforme a lo explicado arriba.

## Gestión de pods

* Función: Exposición segura de los servicios internos al exterior usando HTTPS y subdominios au

# 6. Conclusiones

* La arquitectura del proyecto sigue un enfoque estructurado en cinco fases, iniciando con la preparación del e

# 7. Base de datos

* La arquitectura del proyecto sigue un enfoque estructurado en cinco fases, iniciando con la preparación del e

# 8. Bibliografia

* La arquitectura del proyecto sigue un enfoque estructurado en cinco fases, iniciando con la preparación del e

[Configuración de configuración avanzada en WSL | Microsoft Learn](https://learn.microsoft.com/en-us/windows/wsl/wsl-config#example-wslconfig-file)