Centro Profesional
Universidad Europea Madrid
LAUREATE INTERNATIONAL UNIVERSITIES

UNIVERSIDAD EUROPEA

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

CICLO FORMATIVO DE GRADO SUPERIOR ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS EN RED

PROYECTO FIN DE CICLO

Despliegue Automatizado de Instancias de Odoo en Kubernetes Mediante una Interfaz Web en Flask

Daniel Paipa – Edison Palomino

CURSO 23-25

CURSO 2023-2025

TÍTULO: Despliegue Automatizado de Instancias de Odoo en Kubernetes con Minikube y Helm

AUTOR: Daniel Paipa Scarpitti y Edison Palomino

TUTOR DEL PROYECTO: David Álvarez Boyero

FECHA DE LECTURA: 17 de junio de 2025

CALIFICACIÓN:

Fdo: DAVID ÁLVAREZ BOYERO

Tutor/a del Proyecto

RESUMEN:

En este proyecto hemos desarrollado una solución que permite el despliegue automático de instancias personalizadas de Odoo utilizando contenedores sobre Kubernetes. Para facilitar la interacción con el sistema, hemos implementado una aplicación web desarrollada en Flask que permite a los usuarios introducir el nombre de su instancia y seleccionar los módulos que desean activar. El sistema genera dinámicamente los archivos YAML necesarios para desplegar tanto la base de datos PostgreSQL como el servidor de Odoo con la configuración requerida.

Este proyecto permite la creación de múltiples instancias personalizadas de Odoo, cada una con su URL de acceso, lo que facilita su uso en entornos educativos o empresariales donde se necesiten varios entornos aislados. Además, se ha trabajado en la persistencia de datos y en la correcta configuración de volúmenes y servicios para asegurar la continuidad de los despliegues.

El trabajo se ha dividido entre ambos autores para cubrir las diferentes áreas: uno de nosotros se centró en la parte backend (generación de manifiestos YAML, conexión con Kubernetes, configuración de red), y el otro en el desarrollo de la interfaz en Flask, integración con HTML y comunicación con el backend.

ABSTRACT:

This project presents a solution that allows the automatic deployment of customized Odoo instances using containers on Kubernetes. To facilitate user interaction, we developed a web application using Flask that allows users to input the name of their instance and select the modules they wish to activate. The system dynamically generates the necessary YAML files to deploy both the PostgreSQL database and the Odoo server with the required configuration.

This system enables the creation of multiple isolated Odoo instances, each with its own access URL, which is useful in educational or business environments where different testing or production environments are needed. We also ensured data persistence and proper configuration of volumes and services to maintain system integrity.

The tasks were divided between both authors: one focused mainly on backend logic (YAML generation, Kubernetes communication, network setup), while the other developed the Flask-based user interface and its integration with the backend.

AGRADECIMIENTOS

Queremos agradecer en primer lugar a nuestro tutor, David Alvares Boyero, por su guía constante y sus valiosos consejos a lo largo del desarrollo del proyecto. También agradecemos a nuestros compañeros de clase por su colaboración y ayuda técnica en momentos clave. A nuestras familias, por su apoyo incondicional y por comprender las largas horas dedicadas a este trabajo.

INDICE

1. INTRODUCCIÓN

El propósito de este Proyecto Final de Ciclo es diseñar y desarrollar una solución automatizada para el despliegue de instancias personalizadas de Odoo sobre un clúster de Kubernetes, utilizando una interfaz web desarrollada en Flask como canal principal de interacción. A través de este sistema, cualquier usuario podrá crear su propia instancia de Odoo con los módulos deseados, obteniendo al final una URL funcional de acceso.

El proyecto parte de la necesidad de simplificar el despliegue de ERP en entornos educativos o empresariales, donde a menudo se requieren múltiples entornos aislados. Este enfoque permite reducir tiempos, minimizar errores y facilitar el acceso a la tecnología Odoo mediante herramientas libres.

1.1. Objetivos

- Desarrollar una interfaz web con Flask que permita al usuario configurar su instancia de Odoo.
- Automatizar la generación de manifiestos YAML para el despliegue en Kubernetes.
- Desplegar servicios PostgreSQL y Odoo en Kubernetes de forma dinámica y personalizada.
- Validar que cada instancia sea accesible mediante su propia URL.
- Asegurar persistencia de datos y correcta separación entre instancias.

Estos objetivos permiten ofrecer una solución reutilizable, escalable y práctica para centros de formación o pymes interesadas en Odoo.

1.2. Motivación

El interés por este proyecto surge de la necesidad frecuente en el ámbito formativo y empresarial de disponer de instancias de Odoo que puedan ser personalizadas y desplegadas sin complicaciones técnicas. A menudo, configurar y poner en marcha Odoo requiere conocimientos previos en bases de datos, administración de servidores y configuración del entorno. Esto representa una barrera de entrada para muchos usuarios.

Con esta solución, buscamos facilitar el acceso a Odoo con un proceso guiado, visual e intuitivo, capaz de reducir drásticamente el tiempo de despliegue y errores de configuración. También representa una oportunidad para aplicar nuestros conocimientos en contenedores, orquestación con Kubernetes y desarrollo web.

1.3. Antecedentes

Odoo es un ERP modular muy utilizado en empresas por su capacidad de adaptación y su comunidad activa. Aunque existen scripts y herramientas para desplegar Odoo en servidores dedicados o en la nube, la mayoría de las soluciones requieren conocimientos técnicos avanzados y no están orientadas a la personalización masiva de instancias.

Kubernetes ha demostrado ser una tecnología robusta para la orquestación de contenedores, permitiendo una gestión eficiente de recursos, escalabilidad y alta disponibilidad. Combinar Kubernetes con una interfaz web permite acercar su potencial a usuarios no técnicos.

Durante el desarrollo, se investigaron múltiples métodos de despliegue de Odoo, incluyendo el uso de Docker Compose, Helm Charts y manifiestos manuales. Finalmente, se optó por una solución intermedia: generación dinámica de manifiestos YAML, aplicada con comandos `kubectl`, permitiendo mayor control sobre los recursos creados.

Este trabajo se relaciona con prácticas previas del ciclo sobre virtualización, redes, servicios web y bases de datos. Integra conocimientos adquiridos en distintas asignaturas del ciclo formativo, siendo una aplicación transversal y completa.

2. DESARROLLO DE LA PRÁCTICA

Este capítulo describe el proceso completo de desarrollo del proyecto, desde la selección de tecnologías y herramientas hasta la implementación, despliegue y validación del sistema. El objetivo fue construir una plataforma funcional que permitiera desplegar instancias personalizadas de Odoo en un clúster Kubernetes real alojado en la nube, todo gestionado desde una interfaz web sencilla e intuitiva creada con Flask.

2.1. Material

Kubernetes

Se empleó un clúster Kubernetes real desplegado en la nube. Esto permitió simular un entorno de producción y garantizar una alta disponibilidad y escalabilidad.

Docker

Cada instancia de Odoo y su base de datos PostgreSQL se basan en contenedores Docker. Los contenedores permiten el aislamiento completo de cada instancia, asegurando que los datos y configuraciones de cada usuario sean independientes.

Flask

Flask se utilizó para desarrollar la interfaz web que permite al usuario introducir el nombre de su instancia y los módulos deseados. Esta aplicación genera los archivos YAML personalizados y los aplica directamente al clúster Kubernetes.

PostgreSQL

Es el sistema gestor de base de datos utilizado por Odoo. Cada instancia personalizada cuenta con su propia base de datos PostgreSQL, desplegada como un servicio y un volumen persistente dentro del clúster.

HTML y CSS

Se desarrolló una interfaz web simple pero funcional que recibe los datos necesarios del usuario y muestra el enlace final a su instancia de Odoo.

Servidor Cloud (AWS)

Se utilizo para emplear este programa de creación de servicios de odoo en la nube para que estén en la disposición de todo internet

2.2. Planificación

El desarrollo del proyecto se organizó en varias fases, que se llevaron a cabo de forma iterativa:

- **Fase 1 - Investigación y diseño**: Se analizaron las tecnologías disponibles para el despliegue de Odoo y PostgreSQL en Kubernetes. Se eligió una arquitectura modular y dinámica basada en YAML.
- **Fase 2 - Desarrollo backend Flask**: Se creó la lógica necesaria para que el servidor web Flask genere archivos YAML personalizados.
- **Fase 3 - Integración con Kubernetes**: Se implementó la ejecución automática de los comandos kubectl desde Python, apuntando al clúster real en la nube mediante la configuración del archivo kubeconfig.
- **Fase 4 - Diseño de la interfaz web**: Se desarrolló una interfaz gráfica para facilitar la entrada de parámetros por parte del usuario final.
- **Fase 5 - Validación y pruebas**: Se desplegaron varias instancias de Odoo y se probó su correcto funcionamiento, incluyendo conexión a base de datos, instalación de módulos y acceso web.

2.3. Descripción del trabajo realizado

El trabajo se inició con una prueba de concepto en local, para luego migrar el sistema al entorno en la nube. A continuación, se describen los pasos clave:

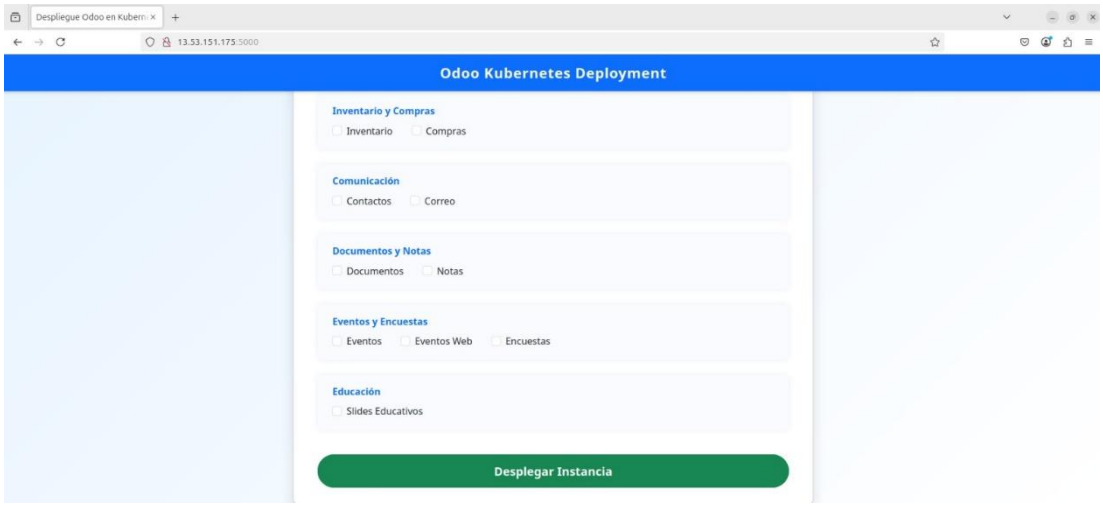
1. **Configuración del entorno en la nube**: Se desplegó un clúster Kubernetes en la nube, configurando los accesos con `kubectl` desde el entorno de desarrollo.
2. **Creación de imágenes Docker**: Se seleccionaron imágenes oficiales de Odoo y PostgreSQL, ajustándolas según las necesidades del proyecto.
3. **Generación dinámica de YAML**: Se diseñó una plantilla base de manifiestos Kubernetes (para Deployment, Service y PVC) que es modificada dinámicamente según los datos ingresados en el formulario web.
4. **Despliegue desde Flask**: Al enviar el formulario, el backend ejecuta `kubectl apply -f archivo.yaml`, desplegando en tiempo real los pods y servicios en el clúster.
5. **Creación de URLs de acceso**: A cada instancia se le asigna una URL única basada en el nombre introducido, permitiendo el acceso individual y simultáneo desde distintos navegadores.
6. **Persistencia y aislamiento**: Se garantiza que cada instancia tenga su base de datos separada y un volumen persistente para evitar pérdida de datos.

Se tomaron capturas de pantalla para documentar el proceso, incluyendo formularios web, terminal de despliegue, panel de pods y prueba de acceso a la URL final.

Imagen del Programa App.py Flask corriendo desde el servidor AWS

```
(venv) ubuntu@ip-172-31-8-102:~/Proyecto$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.8.102:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 798-207-489
92.58.196.79 - - [07/Jun/2025 21:45:23] "POST / HTTP/1.1" 200 -
92.58.196.79 - - [07/Jun/2025 21:45:38] "POST / HTTP/1.1" 200 -
 * Detected change in '/home/ubuntu/Proyecto/app.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 798-207-489
92.58.196.79 - - [07/Jun/2025 21:49:42] "POST / HTTP/1.1" 200 -
92.58.196.79 - - [07/Jun/2025 21:52:08] "GET / HTTP/1.1" 200 -
92.58.196.79 - - [07/Jun/2025 21:54:27] "GET / HTTP/1.1" 200 -
92.58.196.79 - - [07/Jun/2025 21:59:44] "GET / HTTP/1.1" 200 -
92.58.196.79 - - [07/Jun/2025 22:01:01] "GET / HTTP/1.1" 200 -
92.58.196.79 - - [07/Jun/2025 22:03:17] "POST / HTTP/1.1" 200 -
```

Imagen de la página Frontend App.py



2.4. Resultados y validación

Los resultados del proyecto fueron satisfactorios:

- Se desplegaron múltiples instancias de Odoo con éxito desde la interfaz web.
- Cada instancia funcionó correctamente, accediendo a su propia base de datos y módulos.
- Las URLs generadas permitieron acceder al ERP desde cualquier navegador sin configuración adicional.
- Se validó la persistencia de datos tras reiniciar los pods o escalar los servicios.
- El sistema demostró ser escalable y reutilizable, incluso para usuarios sin conocimientos técnicos.

Entre los problemas detectados durante el proceso destacaron:

- Configuración inicial del acceso al clúster desde Python.
- Errores de red relacionados con servicios mal definidos en YAML.
- Necesidad de gestionar correctamente los volúmenes persistentes para evitar conflictos.

Imagen de la creación de un odoo personalizado de prueba

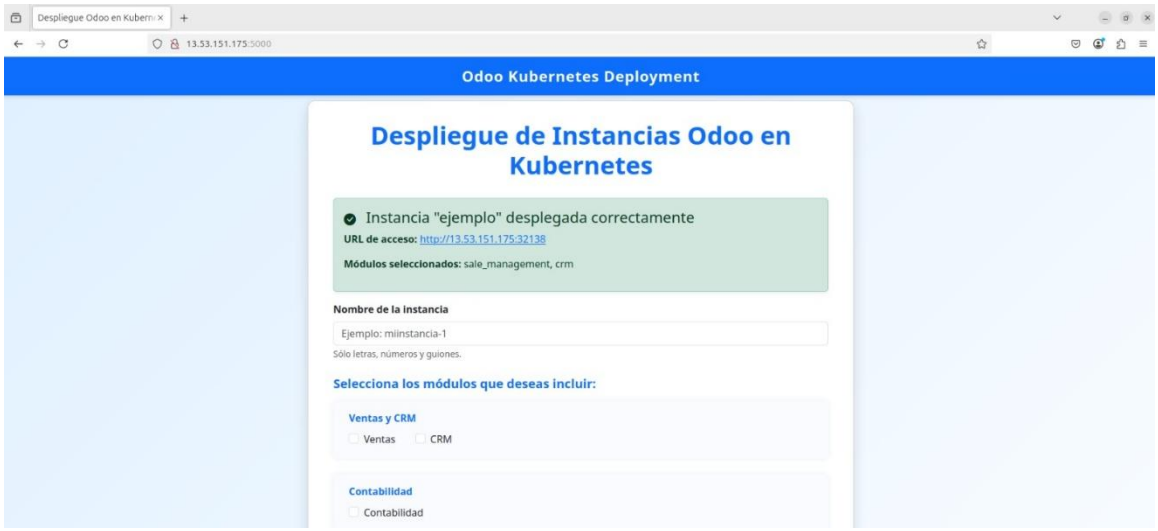


Imagen de los servicios y deployments creados en la prueba de la imagen anterior.

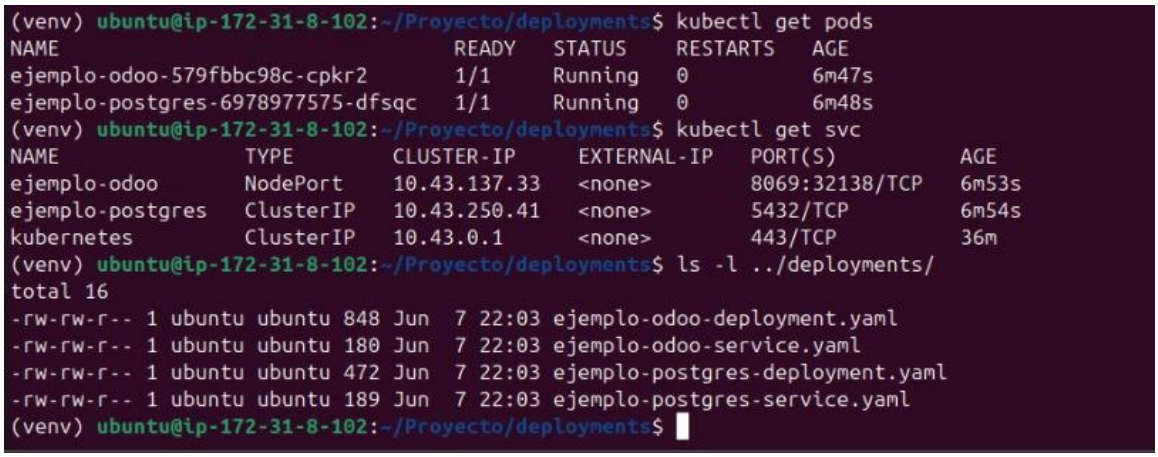


Imagen de la creación del odoo personalizados corriendo a la perfección

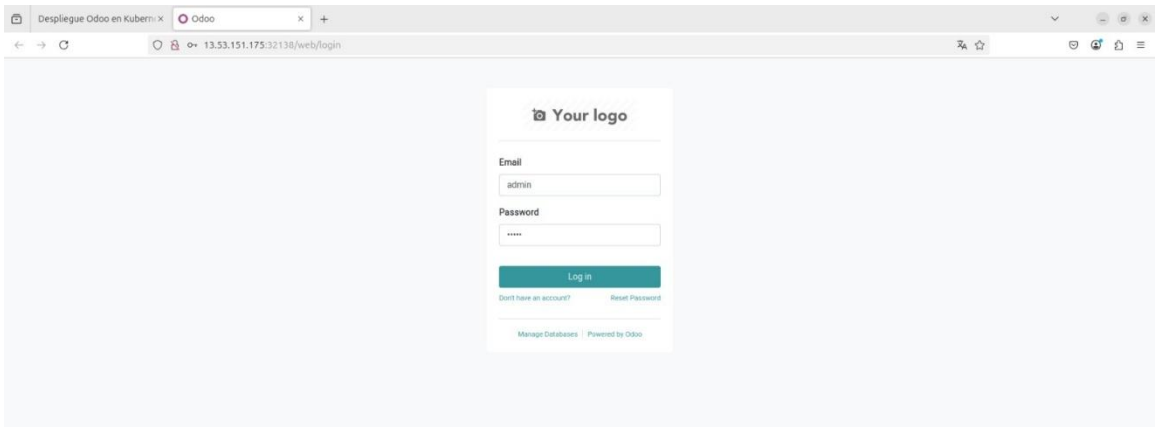
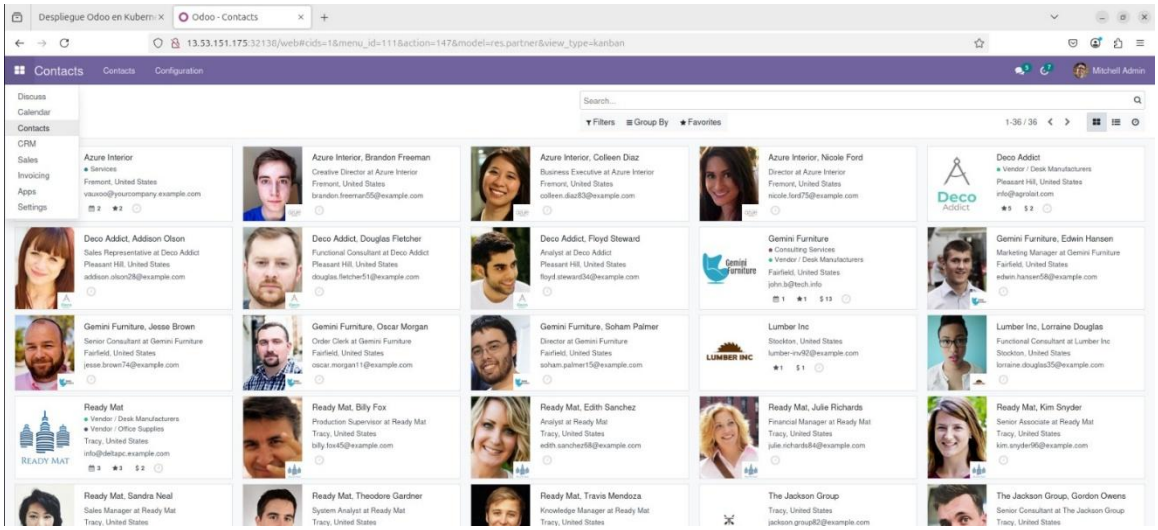


Imagen del odoo personalizado con los módulos que se crearon respectivamente



3. CONCLUSIONES

El presente proyecto ha consistido en el diseño e implementación de una plataforma capaz de desplegar instancias personalizadas de Odoo en un clúster Kubernetes real alojado en la nube, mediante una interfaz web desarrollada en Flask. A lo largo del desarrollo se han cumplido los objetivos iniciales propuestos: creación de una interfaz funcional, automatización del despliegue mediante generación de manifiestos YAML, asignación de URL únicas para cada instancia, y persistencia de datos en cada servicio desplegado.

Uno de los principales hitos alcanzados ha sido la integración de distintas tecnologías (Docker, Kubernetes, Flask, PostgreSQL) en un flujo completamente automatizado accesible desde un navegador web, sin necesidad de conocimientos técnicos por parte del usuario final. Además, se logró migrar el entorno de pruebas local (inicialmente en Minikube) a un clúster real en la nube, lo que añade realismo y viabilidad al proyecto desde una perspectiva empresarial.

3.1. Aportaciones

Este proyecto aporta una solución práctica y novedosa para el despliegue simplificado de Odoo en entornos formativos o empresariales. Su principal valor radica en eliminar la complejidad técnica normalmente asociada con la puesta en marcha de Odoo, permitiendo que cualquier usuario pueda tener su instancia personalizada en minutos, directamente desde una web.

A diferencia de soluciones existentes como Helm Charts o Docker Compose, este sistema permite crear instancias independientes con parámetros personalizados, lo que puede facilitar la implantación de Odoo en academias, pruebas de software, o entornos donde se requiera mantener aislamiento entre usuarios.

3.2. Trabajo futuro

A pesar de haber conseguido un sistema funcional, el proyecto deja abiertas diversas líneas de mejora y expansión:

- **Sistema de autenticación de usuarios**: Permitir que los usuarios creen cuentas y gestionen múltiples instancias desde su panel personal.
- **Gestión de instancias activas**: Crear una interfaz que muestre las instancias activas, permita pausarlas, eliminarlas o reiniciarlas desde la web.
- **Implementación de HTTPS y seguridad de red**: Añadir certificados SSL y aplicar políticas de seguridad para el acceso a las instancias.
- **Logs y métricas**: Incluir un sistema de monitorización (como Prometheus + Grafana) para visualizar el estado del clúster y las instancias.

3.3. Cronograma

Semana	Fechas	Objetivos principales	Tareas realizadas o planificadas
1	5 – 11 de mayo	**Preparación del entorno local y herramientas**	- Instalación de Docker y kubectl- Estructura base del proyecto- Familiarización con Kubernetes
2	12 – 18 de mayo	**Despliegue de PostgreSQL en Kubernetes**	- YAML de ConfigMap con credenciales- PVC + PV- Deployment y Service de PostgreSQL
3	19 – 25 de mayo	**Despliegue de Odoo en Kubernetes**	- YAML de Deployment de Odoo- Variables para conexión a PostgreSQL- Service tipo NodePort
4	26 de mayo – 1 de junio	**Automatización con Flask – Backend**	- Desarrollo de app.py- Generación dinámica de YAMLs- Aplicación de manifiestos vía kubectl
5	2 – 8 de junio	**Automatización con Flask – Frontend y testing**	- Creación de index.html- Conexión con app.py- Pruebas de interfaz y despliegue dinámico
6	9 – 15 de junio	**Despliegue en Azure y presentación final**	- Configuración de VM en Azure- Instalación de dependencias- Acceso remoto vía navegador- Verificación y documentación final

4. BIBLIOGRAFÍA Y WEBGRAFÍA

PANDORA FMS. (s.f.). *Guía Kubernetes: ¿Qué es, para qué sirve y cómo funciona?*. Fecha de consulta: 15:37, junio 6, 2025 de https://pandorafms.com/blog/es/guia-kubernetes/

AJNAIKKAVIGNESH, A. (2023). *Odoo with Minikube made easy*. Medium. Fecha de consulta: 15:30, Mayo 13, 2025 de https://medium.com/@ajnaikkavignesh2707/odoo-with-minikube-made-easy-b6cf6b3c0ac9

LABEX. (s.f.). *Flask: Introducción al framework web Flask*. Fecha de consulta: 15:39, mayo , 2025 de https://labex.io/es/tutorials/flask-getting-started-with-flask-web-framework-136334

NORFIPC. (s.f.). *Cómo insertar scripts y hojas de estilo CSS con JavaScript en documentos HTML*. Fecha de consulta: 15:41, junio 3, 2025 de https://norfipc.com/inf/como-insertar-scripts-css-con-javascript-html.php

DATACAMP. (s.f.). *Aprende AWS: Qué es Amazon Web Services y cómo empezar*. Fecha de consulta: 15:35, junio 15, 2025 de https://www.datacamp.com/es/blog/learn-aws

5. ANEXOS

1. Problema: Fallos de conexión entre Odoo y PostgreSQL
Al principio, las instancias de Odoo no podían conectarse correctamente con su base de datos PostgreSQL. Esto causaba errores al iniciar los contenedores.

Solución:
Se ajustó correctamente la variable de entorno DB_HOST dentro del manifiesto YAML para que apuntara al nombre del servicio de PostgreSQL generado dinámicamente para cada instancia. Además, se validó que ambos contenedores estuvieran en el mismo namespace y se desplegaran en el orden correcto.

2. Problema: El sistema solo funcionaba en Minikube
Inicialmente, el sistema se diseñó y probó sobre Minikube, un entorno local que no representaba un entorno productivo. Esto limitaba la accesibilidad desde fuera y no permitía URLs públicas.

Solución:
Se migró todo el sistema a un clúster Kubernetes en la nube, lo que permitió desplegar servicios accesibles desde el exterior, generar URLs públicas para las instancias y hacer el sistema más realista y escalable.

3. Problema: Error en la generación dinámica de YAML
Al principio, los archivos YAML generados automáticamente por Flask tenían errores de indentación o formato, lo que provocaba fallos al aplicar kubectl apply.

Solución:
Se implementaron funciones en Python que generan correctamente los manifiestos utilizando plantillas bien estructuradas, controlando variables como puertos, nombres únicos y módulos de Odoo.

4. Problema: Las instancias no eran accesibles desde el navegador
Aunque los pods y servicios se desplegaban correctamente, no se podía acceder a las instancias de Odoo desde el navegador, ya que no se exponían de forma pública.

Solución:
Se cambió el tipo de servicio a NodePort o LoadBalancer según el proveedor del clúster, permitiendo que se genere una URL pública por instancia. Posteriormente, Flask se encargó de mostrar esta URL al usuario al finalizar el despliegue.