



INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

Carrera:

Ingeniería en Sistemas Computacionales

Unidad de aprendizaje:

Sistemas Distribuidos

Práctica 10:

Despliegue en la Nube.

Alumno:

Cardoso Osorio Atl Yosafat

Hernández Vázquez Jorge Daniel

Profesor:

Chadwick Carreto Arellano

Fecha:

20/06/2025

INSTITUTO POLITÉCNICO NACIONAL



Índice de contenido

Antecedentes.....	3
Planteamiento del problema	4
Propuesta de solución	4
Materiales y métodos empleados.....	5
AWS	5
Materiales	5
Métodos Empleados	6
Google Cloud.....	7
Microsoft Azure.....	7
Servicios Principales Utilizados	7
Desarrollo	9
Azure	9
Fase 1: Control de Versiones con Git y GitHub.....	10
Fase 2: Contenerización de Microservicios con Docker	12
Fase 3: Aprovisionamiento de la Infraestructura en Azure	14
Fase 4: Adaptación del Código y Pipeline de CI/CD	18
Fase 5: Verificación y Resultados Finales.....	24
AWS	33
Despliegue de PWA en AWS.....	33
Creación del Bucket en S3.....	33
Subida de los Archivos del Frontend.....	34
Configuración de Permisos.....	36
Activación de Sitio Web Estático	36
Acceso a la PWA	37
Google Cloud.....	41
Creación de proyecto	41
Creación de base de datos en Google Cloud SQL.....	42
Despliegue de Servicio Web en Google Cloud Run	46
Despliegue de PWA en Google Cloud Storage	50
Conclusión.....	53

Antecedentes

Antes de la llegada de la computación en la nube, las empresas y organizaciones dependían de infraestructuras físicas locales (on-premise), lo que representaba diversas desventajas. Uno de los principales inconvenientes era los altos costos iniciales, ya que las organizaciones debían realizar inversiones significativas en servidores, licencias de software y en el mantenimiento de sus infraestructuras. Además, la escalabilidad limitada era otro desafío importante, ya que adaptarse a picos de demanda requería la compra de más hardware, lo que no siempre era práctico ni rentable. Además, este modelo de gestión implicaba una complejidad operativa, pues era necesario contar con equipos de TI dedicados exclusivamente a gestionar el hardware y las redes.

El cambio hacia la computación en la nube se aceleró con la globalización y el aumento de los datos, conocidos como big data, durante la década de 2000. Este avance dio paso a un modelo más flexible y eficiente, que ofrecía múltiples beneficios. Entre estos destacan el pago por uso, lo que eliminó la necesidad de realizar inversiones iniciales en hardware costoso; la escalabilidad bajo demanda, que permite ajustar los recursos en tiempo real según las necesidades de la empresa; y el acceso global, que proporciona a las empresas la posibilidad de acceder a servicios desde cualquier lugar del mundo, facilitando la expansión internacional.

Hoy en día, los tres proveedores más relevantes en el mercado de la computación en la nube son Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP). AWS, lanzado en 2006, comenzó con su primer servicio, S3, para escalar su infraestructura y soportar las ventas masivas. Actualmente, es el líder del mercado, con un 33% de participación. Por su parte, Microsoft Azure, iniciado en 2010 como Windows Azure, surgió para modernizar el modelo de negocio de Microsoft frente al auge de los servicios SaaS. Actualmente, ocupa el 22% del mercado. Finalmente, Google Cloud Platform, que comenzó en 2008 y se hizo público en 2011, surgió de la necesidad de Google de gestionar su infraestructura global y ofrecerla como un servicio. GCP cuenta actualmente con un 10% del mercado.

Este cambio hacia la nube ha propiciado una verdadera transformación digital, ya que el 94% de las empresas utilizan al menos un servicio en la nube. Además, han surgido nuevos paradigmas de gestión como el serverless, que permite la ejecución de código sin la necesidad de gestionar servidores, como en los casos de AWS Lambda o Google Cloud Functions. El uso de contenedores también ha ganado protagonismo, y Kubernetes, desarrollado originalmente por Google, se ha convertido en el estándar para la orquestación de contenedores. Por último, la inteligencia artificial accesible ha permitido democratizar el uso

de machine learning, con servicios como AWS SageMaker, Azure AI y Vertex AI, que facilitan la implementación de estas tecnologías en empresas de todos los tamaños.

Planteamiento del problema

Hoy en día, muchas aplicaciones web y servicios digitales enfrentan el reto de mantener un rendimiento óptimo y una alta disponibilidad a medida que crece el número de usuarios y las demandas de nuevas funcionalidades. Las arquitecturas monolíticas tradicionales presentan restricciones en términos de escalabilidad, mantenimiento y flexibilidad, lo que dificulta la rápida implementación de nuevas características y la adaptación a los cambios tecnológicos.

Además, el despliegue y la gestión de aplicaciones en servidores físicos o infraestructuras locales generan costos elevados, requieren mantenimiento constante y carecen de la elasticidad necesaria para ajustarse dinámicamente a las variaciones de demanda. Este escenario provoca que las organizaciones enfrenten dificultades para garantizar la continuidad del servicio y la satisfacción de los usuarios.

Por otro lado, la creciente adopción de tecnologías en la nube ofrece soluciones que superan estas limitaciones, mediante el uso de servicios escalables, flexibles y con alta disponibilidad. No obstante, persiste una brecha en el conocimiento práctico sobre cómo diseñar, desarrollar, desplegar y aprovechar adecuadamente los recursos que ofrece la nube para lograr sistemas eficientes y seguros.

En este contexto, surge la necesidad de realizar una práctica que permita explorar y comprender las arquitecturas en la nube, trabajando con proveedores como AWS, Azure y GCP en diferentes aspectos relacionados con la configuración de redes y seguridad, con el fin de garantizar un acceso adecuado y una protección eficaz de los datos. Esta experiencia facilitará el aprendizaje aplicado de los conceptos clave en arquitecturas modernas, y nos preparará para afrontar los desafíos reales en la creación y mantenimiento de aplicaciones escalables y resilientes.

Propuesta de solución

Durante esta práctica de sistemas distribuidos, trabajaremos en el diseño y desarrollo de una solución que aborda los desafíos comunes en el desarrollo de aplicaciones escalables y resilientes, aprovechando los servicios en la nube de proveedores como AWS, Azure y Google Cloud Platform (GCP). Nuestro objetivo es mejorar la disponibilidad, el rendimiento y la seguridad de las aplicaciones, al mismo tiempo que optimizo los costos y el esfuerzo de mantenimiento.

Seguridad: Gestión de identidades (IAM), políticas de acceso, encriptación y protección de datos.

Facturación: Control de costos, presupuestos, alertas y optimización de recursos.

Monitoreo: Herramientas para supervisar rendimiento, disponibilidad y logs.

Servicios: Configuración de recursos clave como computación, almacenamiento y bases de datos.

Materiales y métodos empleados

Para llevar a cabo la práctica y desarrollarla se implementó la arquitectura de Microservicios para resolver el problema de la venta de boletos y gestión se emplearon las siguientes herramientas y métodos:

AWS

Materiales

Para el desarrollo y despliegue en la nube, se utilizaron los siguientes materiales y herramientas:

Equipos y Software:

- Computadora personal con sistema operativo Windows 10/11.
- Conexión a Internet estable para acceder a los servicios en la nube.

Amazon Web Services (AWS):

- Instancia EC2 (Elastic Compute Cloud) con el sistema operativo Ubuntu Server 22.04.
- Par de claves (.pem) para el acceso remoto y seguro a través de SSH.
- Configuración de grupos de seguridad (Security Groups) para permitir tráfico HTTP (puerto 5000) y SSH (puerto 22).

Herramientas de Desarrollo:

- **Flask:** Framework de Python utilizado para construir el microservicio RESTful.
- **Python 3.x** y **pip:** Para la gestión de paquetes y ejecución del código.
- **Visual Studio Code** o cualquier editor de texto para escribir y desarrollar el código.
- **Git** y **GitHub:** Herramientas para la gestión de versiones y control de código fuente.

- **Terminal Bash / PowerShell:** Utilizados para la ejecución de comandos y la conexión remota con la instancia en la nube.

Herramienta Adicional (opcional):

- **Ngrok:** Utilizado para realizar tunelado HTTP durante los entornos de prueba.

Métodos Empleados

Configuración del entorno de desarrollo local:

- Se creó un entorno virtual en Python utilizando venv para garantizar el aislamiento de las dependencias específicas del proyecto.
- Se procedió a la instalación de los paquetes requeridos, destacando Flask, a través de pip.

Desarrollo del microservicio:

- Se desarrolló un microservicio RESTful empleando Flask, el cual simula una base de datos en memoria para gestionar productos.
- Se implementaron las rutas necesarias para llevar a cabo operaciones CRUD (crear, leer, actualizar y eliminar).
- El archivo principal del microservicio fue denominado app.py.

Configuración del repositorio Git:

- El proyecto fue inicializado como un repositorio Git y se subió a GitHub para su gestión.
- Se utilizaron los comandos git pull y git checkout en la instancia EC2 para mantener el código actualizado de manera remota.

Despliegue en la nube con AWS EC2:

- Se lanzó una instancia EC2 desde la consola de AWS, utilizando una imagen del sistema operativo Ubuntu Server.
- Se accedió de forma remota a la instancia mediante SSH, utilizando una clave privada (.pem).
- Se instalaron las dependencias necesarias (Python, pip, Flask) en la instancia EC2.
- El microservicio fue ejecutado con el comando python3 app.py, configurando el host a 0.0.0.0 para permitir conexiones externas.

Configuración de acceso a la aplicación:

- Se configuró el grupo de seguridad de la instancia EC2 para permitir el tráfico entrante a través del puerto 5000.
- Se verificó la IP pública de la instancia y se accedió al microservicio tanto desde un navegador web como utilizando la herramienta curl.

Pruebas y validación:

- Se llevaron a cabo pruebas de conexión y funcionalidad para cada una de las rutas del microservicio (GET, POST, PUT, DELETE).
- Se observó la respuesta en formato JSON y se validó el comportamiento de la aplicación frente a diferentes tipos de peticiones.

Google Cloud

Para la configuración en la nube, se utilizaron los siguientes servicios de Google Cloud Platform (GCP):

- **Google Cloud Monitoring:** Para la configuración del monitoreo de los servicios.
- **Facturación:** Para gestionar y monitorear los presupuestos del proyecto.
- **IAM (Identity and Access Management):** Para la configuración de la seguridad y acceso.
- **App Engine:** Para la implementación y configuración del servicio.
- **Cloud SQL y Compute Engine:** Para la configuración y gestión de la base de datos.
- **Cloud Storage:** Para la configuración de almacenamiento de los datos.

Además, se emplearon las siguientes herramientas de apoyo:

- **Gemini:** Asistente de inteligencia artificial utilizado para explicar secciones de uso específico.
- **GitHub:** Utilizado dentro del servicio de App Engine para la gestión del código.

Microsoft Azure

Servicios Principales Utilizados

a) **Azure App Service – Aplicación Web**

- Se configuró un servicio de tipo App Service (Web App) con sistema operativo Windows y tecnología Java 21, alineado con el lenguaje de desarrollo utilizado para las prácticas futuras.
- Se seleccionó el plan de hospedaje Free (F1) o Standard (S1), dependiendo de los requerimientos de rendimiento específicos.
- Se habilitó la opción de HTTPS obligatorio para garantizar la seguridad de las conexiones, además de configurar un entorno de ejecución compatible con la versión local utilizada durante el desarrollo.

b) Azure Authentication (Autenticación y Autorización)

- Se integró el módulo de Autenticación y Autorización de Azure App Service para asegurar y proteger el acceso a la aplicación, garantizando que solo los usuarios autorizados pudieran acceder a ella.

c) Azure Monitor – Supervisión

- Se activó Azure Monitor para recopilar métricas clave de rendimiento, incluyendo el uso de CPU, memoria, disponibilidad y tiempos de respuesta de la aplicación.
- Se habilitaron los logs de diagnóstico y la integración con Application Insights, lo que permitió rastrear errores y analizar el tráfico de la aplicación en tiempo real.
- Se configuraron alertas automáticas para notificar sobre fallos en el sistema o el sobreuso de recursos, garantizando una respuesta rápida ante posibles problemas.

d) Cuenta de Almacenamiento de Azure

- Se implementó una cuenta de almacenamiento del tipo StorageV2, adecuada para las necesidades de la aplicación.
- Se utilizó Blob Storage para almacenar archivos estáticos, como imágenes, documentos y otros archivos generados por la aplicación, asegurando una gestión eficiente de los datos.

e) Etiquetado y Organización de Recursos

- Todos los recursos creados fueron organizados en un grupo de recursos lógico dentro de Azure, facilitando su administración.
- Se aplicaron etiquetas clave-valor (tags) para facilitar la clasificación y gestión de los recursos según su proyecto, entorno y responsable, mejorando la eficiencia en la administración de los mismos.

Desarrollo

Azure

En este apartado, se detalla el proceso completo para el despliegue de un proyecto en la nube de Microsoft Azure. La aplicación sigue una arquitectura moderna y distribuida, compuesta por tres elementos principales:

1. **Backend (Microservicios):** El núcleo de la aplicación está construido bajo un patrón de microservicios. Funcionalidades específicas como la gestión de películas, ventas y compras operan como servicios web independientes y autónomos.
2. **API Gateway:** Para unificar y gestionar el acceso a los microservicios, se implementó un API Gateway. Este componente actúa como un único punto de entrada para todas las solicitudes del cliente, simplificando la comunicación y la seguridad.
3. **Frontend (PWA):** La interfaz de usuario es una Progressive Web App (PWA), la cual se despliega como un sitio web estático para garantizar una entrega de contenido rápida, eficiente y una experiencia de usuario óptima en múltiples dispositivos.

Para llevar a cabo el despliegue de manera ordenada y escalable, el proceso se dividió en las siguientes cuatro fases fundamentales:

- **Fase 1: Control de Versiones con Git y GitHub:** Preparación, limpieza y centralización del código fuente en un repositorio remoto.
- **Fase 2: Contenerización con Docker:** Empaquetado de cada microservicio en un contenedor para asegurar su portabilidad y consistencia.
- **Fase 3: Aprovisionamiento de la Infraestructura en Azure:** Creación de todos los recursos necesarios en la nube (servicios de aplicación, base de datos, registro de contenedores, etc.).
- **Fase 4: Automatización del Despliegue (CI/CD) con GitHub Actions:** Implementación de un flujo de trabajo automatizado para construir y desplegar la aplicación continuamente.

A continuación, se describe en detalle el procedimiento realizado en la Fase 1.

Fase 1: Control de Versiones con Git y GitHub

El objetivo de esta fase inicial fue establecer una base sólida para el proyecto, asegurando que todo el código fuente estuviera debidamente versionado, limpio de archivos innecesarios y centralizado en un repositorio en la nube para facilitar la colaboración y la automatización posterior.

Paso 1.1: Creación del Repositorio Local y Central

Se utilizó el sistema de control de versiones **Git**. Primero, se inicializó un repositorio local en la carpeta raíz del proyecto. Simultáneamente, se creó un repositorio vacío en **GitHub**, que serviría como el origen central (remoto) para todo el código.

Paso 1.2: Configuración del Archivo `.gitignore`

Un paso fundamental fue la creación de un archivo `.gitignore` en la raíz del proyecto. Este archivo de configuración instruye a Git para que ignore archivos y directorios que no deben ser incluidos en el control de versiones, tales como:

- Carpetas de dependencias (`node_modules/`).
- Archivos de registro (`.log`).
- Archivos de entorno con credenciales o configuraciones locales (`.env`, `*.local`).
- Archivos generados por el sistema operativo (`.DS_Store`, `Thumbs.db`).

Este procedimiento es crucial para mantener el repositorio limpio, optimizar los tiempos de clonación y subida, y evitar la exposición de información sensible.

Contenido del archivo `.gitignore` implementado:

```
.gitignore
1  # Logs
2  logs
3  *.log
4  npm-debug.log*
5  yarn-debug.log*
6  yarn-error.log*
7
8  # Dependencias
9  node_modules/
10 dist/
11 dist-ssr/
12 *.local
13
14 # Archivos de entorno
15 .env
16 .env.local
17 .env.development.local
18 .env.test.local
19 .env.production.local
20
21 # Archivos de Sistema Operativo
22 .DS_Store
23 Thumbs.db
```

Paso 1.3: Confirmación Inicial y Sincronización con GitHub

Una vez configurado el .gitignore, se procedió a realizar la primera confirmación (commit) y a sincronizar el repositorio local con el de GitHub. Para ello, se ejecutaron los siguientes comandos en la terminal, obviamente dentro de la carpeta del proyecto que subimos a github.

```
PS C:\Users\user\Downloads\Microservicios> git init
Reinitialized existing Git repository in C:/Users/user/Downloads/Microservicios/.git/
PS C:\Users\user\Downloads\Microservicios> git add .
warning: in the working copy of 'apigateway/package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'apigateway/package.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'compras-service/package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'compras-service/package.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'películas-service/package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'películas-service/package.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'ventas-service/package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'ventas-service/package.json', LF will be replaced by CRLF the next time Git touches it
PS C:\Users\user\Downloads\Microservicios> git commit -m "Commit inicial del proyecto de microservicios"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
PS C:\Users\user\Downloads\Microservicios> git remote add origin https://github.com/Dxniel7/prueba.git
error: remote origin already exists.
PS C:\Users\user\Downloads\Microservicios> git branch -M main
PS C:\Users\user\Downloads\Microservicios> git push -u origin main
branch 'main' set up to track 'origin/main'.
Everything up-to-date
PS C:\Users\user\Downloads\Microservicios> |
```

Al finalizar esta fase, el código fuente completo y limpio del proyecto quedó alojado y versionado en GitHub, listo para las siguientes etapas de contenerización y despliegue.

The screenshot shows the GitHub interface for a repository named 'microservicio' by user 'Dxniel7'. The repository is public and has 12 commits. The commit history is listed on the left, showing a series of commits for adding workflows, dependencies, and initial project setup. The right sidebar shows the repository's activity, including stars, forks, and releases. The 'About' section is currently empty, and the 'Packages' section shows no published packages. The 'Languages' section shows the repository is primarily JavaScript (85.9%), followed by CSS (8.8%) and HTML (5.3%).

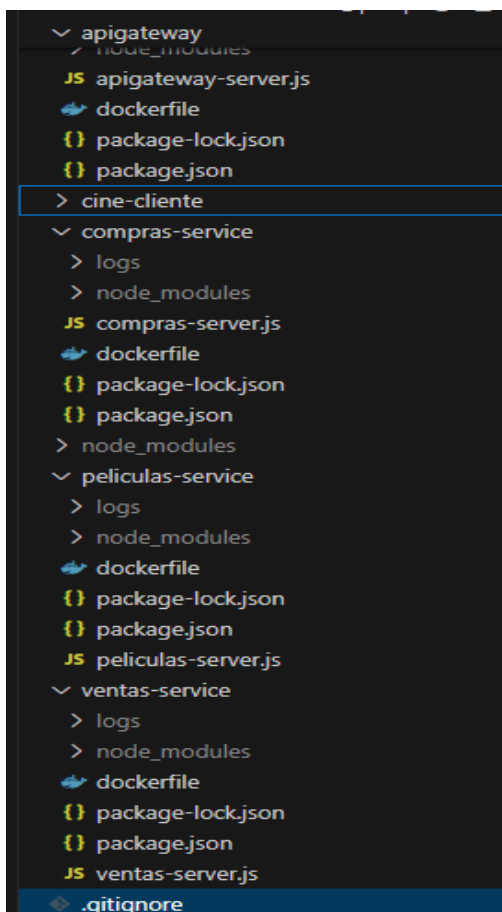
File	Commit Message	Time
.github/workflows	Añadir workflows de CI/CD para el backend	20 hours ago
apigateway	FIX: Añadir sequelize como dependencia a cada servicio	19 hours ago
cine-cliente	CHORE: Mejorar estrategia de caché del Service Worker	19 hours ago
compras-service	FIX: Añadir sequelize como dependencia a cada servicio	19 hours ago
películas-service	FIX: Añadir sequelize como dependencia a cada servicio	19 hours ago
ventas-service	FIX: Añadir sequelize como dependencia a cada servicio	19 hours ago
.gitignore	Commit inicial del proyecto de microservicios	yesterday
cine-cliente.zip	Commit inicial del proyecto de microservicios	yesterday
package-lock.json	Commit inicial del proyecto de microservicios	yesterday
package.json	Commit inicial del proyecto de microservicios	yesterday

Fase 2: Contenerización de Microservicios con Docker

Una vez que el código estuvo versionado en GitHub, el siguiente paso fue prepararlo para su ejecución en la nube. Para lograr portabilidad, aislamiento y consistencia entre el entorno de desarrollo local y el de producción en Azure, se utilizó **Docker**. El objetivo de esta fase fue "empaquetar" cada microservicio del backend en su propia **imagen de contenedor**.

Paso 2.1: Creación y Configuración de los Dockerfiles

Se creó un archivo llamado Dockerfile (sin extensión) dentro de la carpeta de cada uno de los microservicios del backend: apigateway, compras-service, peliculas-service y ventas-service. Este archivo actúa como un plano o receta que define todos los pasos necesarios para construir la imagen del contenedor del servicio.



Nota: El frontend (cine-cliente) no requirió un Dockerfile, ya que su despliegue se planificó usando un servicio especializado para aplicaciones estáticas (Azure Static Web Apps), que simplifica este proceso.

Contenido del Dockerfile Genérico Utilizado para Cada Servicio:

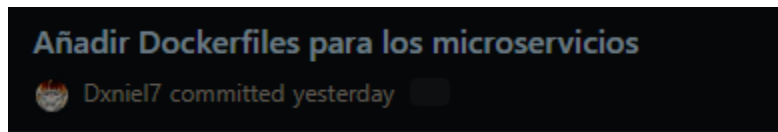
```
apigateway > dockerfile > ...
1  FROM node:18-alpine
2
3  # Directorio de trabajo dentro del contenedor
4  WORKDIR /usr/src/app
5
6  # Copiar los archivos de configuración de npm y package.json
7  COPY package*.json ./
8  RUN npm install
9
10 # Copiar el resto del código de la aplicación
11 COPY . .
12
13 # Exponer el puerto en el que la aplicación va a correr
14 EXPOSE 3000
15
16 # El comando para iniciar la aplicación.
17 CMD ["node", "apigateway-server.js"]
```

Análisis de las directivas clave del Dockerfile:

- FROM node:18-alpine: Especifica que la imagen se basará en la versión 18 de Node.js sobre Alpine Linux, una distribución ligera ideal para contenedores.
- WORKDIR /usr/src/app: Crea y establece el directorio de trabajo dentro del contenedor.
- COPY y RUN npm install: Una optimización clave. Al copiar primero los archivos package.json, Docker solo reinstalará las dependencias si estos archivos han cambiado, acelerando significativamente las reconstrucciones posteriores.
- EXPOSE: Documenta el puerto que la aplicación escuchará. Aunque no publica el puerto automáticamente.
- CMD: Especifica el comando para ejecutar la aplicación. Para cada servicio, se ajustó al nombre de su archivo principal (ej. peliculas-server.js, ventas-server.js, etc.).

Paso 2.2: Actualización del Repositorio

Finalmente, los nuevos Dockerfile se añadieron al control de versiones y se subieron al repositorio de GitHub, dejándolos listos para ser utilizados por el sistema de automatización en la fase de CI/CD.

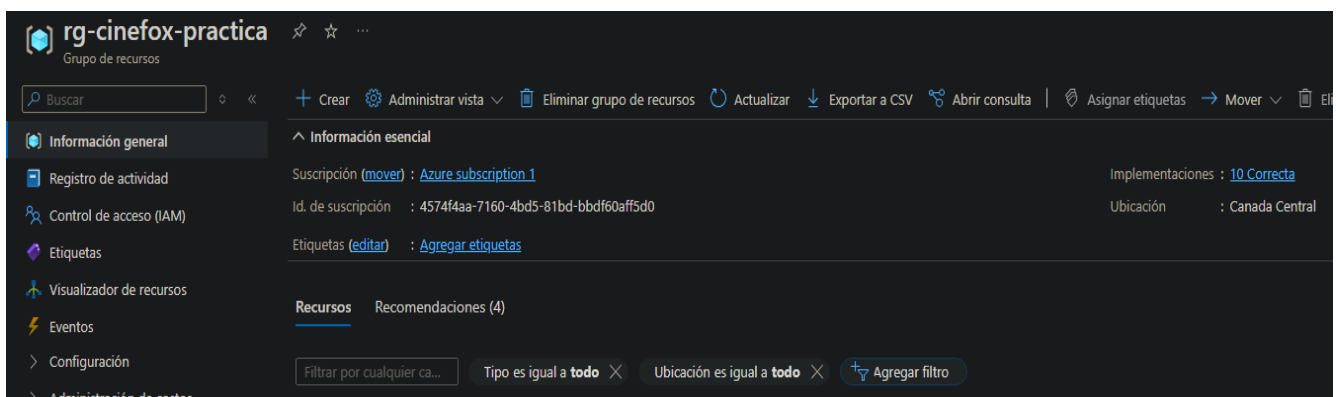


Fase 3: Aprovisionamiento de la Infraestructura en Azure

Con el código fuente preparado y "dockerizado", el siguiente paso fue construir el "hogar" para la aplicación en la nube de Microsoft Azure. Esta fase consistió en crear y configurar todos los servicios de Azure necesarios para alojar el frontend, los microservicios del backend y la base de datos.

Paso 3.1: Creación del Grupo de Recursos

Como primer paso organizativo, se creó un **Grupo de Recursos** en el portal de Azure. Este grupo actúa como un contenedor lógico para agrupar y gestionar todos los recursos del proyecto (servicios de aplicación, bases de datos, etc.) de manera centralizada, facilitando la administración y el control de costos.



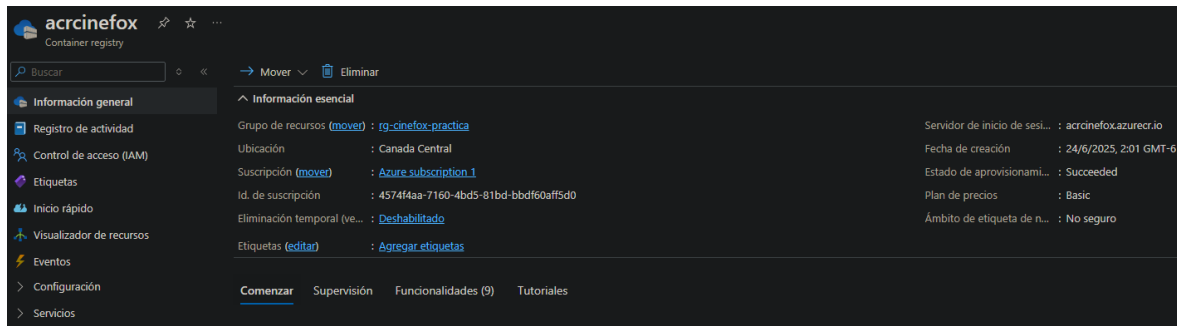
Paso 3.2: Azure Container Registry (ACR)

Para almacenar de forma privada y segura las imágenes Docker creadas en la Fase 2, se aprovisionó un **Azure Container Registry (ACR)**.

- **Nombre:** Se le asignó un nombre único global, en este caso accinefox.

- **SKU:** Se seleccionó el plan *Basic*, adecuado para las necesidades de desarrollo y pruebas de este proyecto.

Este registro fue fundamental para el pipeline de CI/CD, ya que es el punto desde el cual los servicios de aplicación desplegarán los contenedores.

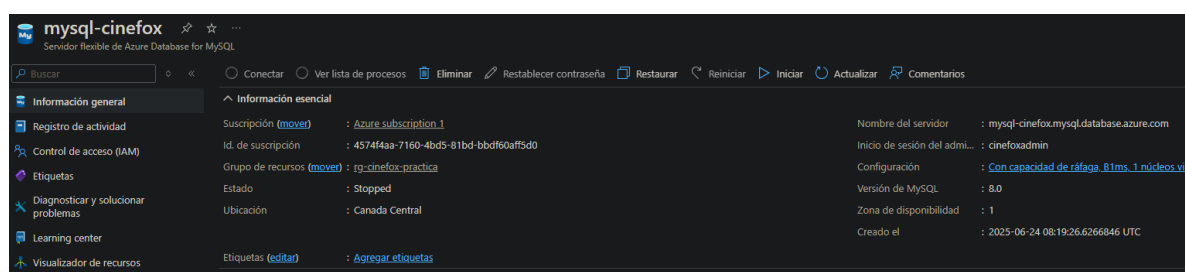


Paso 3.3: Creación de la Base de Datos Centralizada

Un punto crítico de la arquitectura fue la transición de una base de datos localhost a una solución en la nube. Los microservicios, al ejecutarse en contenedores aislados, no pueden compartir un localhost.

- **Servicio:** Se optó por **Azure Database for MySQL - Servidor flexible**, un servicio de base de datos gestionado que elimina la necesidad de mantener un servidor propio.
- **Configuración de Red:** El paso más importante en su configuración fue en la pestaña "Redes", donde se marcó la casilla **"Permitir el acceso público desde cualquier servicio de Azure dentro de Azure a este servidor"**. Esta regla de firewall es la que habilita la comunicación segura entre los App Services (donde corren los microservicios) y el servidor de la base de datos.
- **Credenciales:** Se crearon y guardaron de forma segura el nombre del servidor, el nombre de usuario del administrador y la contraseña. Estos datos fueron cruciales para la configuración de la conexión en la siguiente fase.

En la siguiente Figura podemos observar la información general del **Azure Database for MySQL**.



Posteriormente como se indicó, en el apartado de redes se tuvo que habilitar la casilla **"Permitir el acceso público desde cualquier servicio de Azure dentro de Azure a este servidor"**.

Conexión TLS/SSL aplicada

TLS/SSL se aplica en el servidor de forma predeterminada. Puede descargar el certificado público de SSL desde el menú anterior. Para deshabilitar la SSL, actualice el parámetro del servidor `require_secure_transport` en OFF (desactivado). También puede cambiar la versión de TLS al actualizar el parámetro del servidor `tls_version`. [Más información](#)

Acceso público

☒ Permitir el acceso público a este recurso a través de Internet mediante una dirección IP pública ⓘ

Reglas de firewall

Se permitirán las conexiones entrantes desde las direcciones IP especificadas a continuación en el puerto 3306 de este servidor. [Más información](#) ⓘ

ⓘ Es posible que algunos entornos de red no informen de la dirección IP real de acceso público necesaria para acceder a su servidor. Si, tras agregar la dirección IP, no se permite el acceso al servidor, póngase en contacto con su administrador de red.

☒ Permitir acceso público a este servidor desde cualquier servicio de Azure dentro de Azure ⓘ

+ Agregar dirección IP del cliente actual (187.169.237.146) + Agregar 0.0.0.0 - 255.255.255.255

Nombre de la regla de firewall	Dirección IP inicial	Dirección IP final
ClientIPAddress	187.169.237.146	187.169.237.146

Y finalmente, es importante mencionar que, durante el desarrollo de este servicio, fue necesario configurar un **usuario y contraseña para el acceso a la base de datos**. Esta información es la que se utilizó para establecer la conexión con la BD.

Además, una vez que se creó nuestro servicio de BD, finalmente tuvimos que crear nuestra BD para la aplicación, en base al nombre especificado en el desarrollo del proyecto, en nuestro caso fue “cineboletos”.

+ Agregar Eliminar Actualizar Comentarios

Puede crear, ver y eliminar bases de datos MySQL en este servidor. Tenga en cuenta que no puede eliminar ninguna base de datos del sistema, como mysql, s...

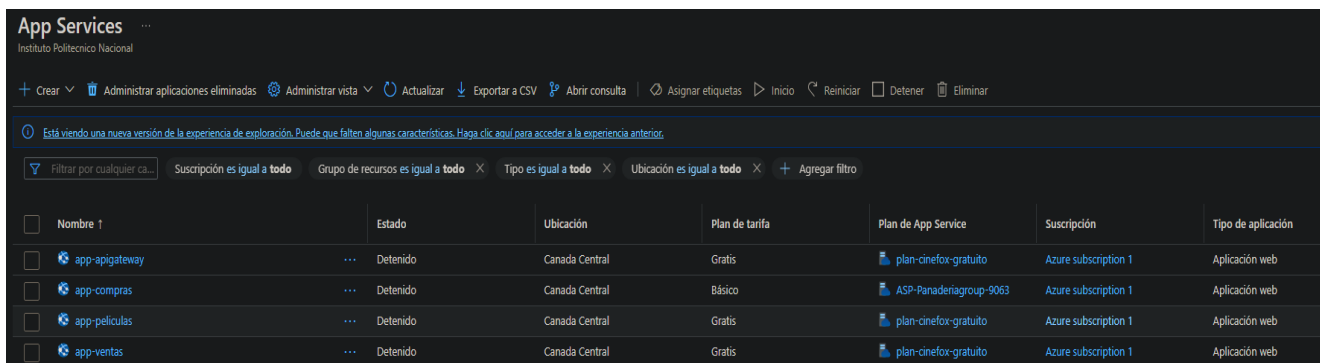
<input type="checkbox"/>	Nombre ↑	Juego de carac...	Colación	Tipo de esque...	
	mysql	utf8mb4	utf8mb4_0900_...	System	
	information_schema	utf8mb3	utf8mb3_gener...	System	
	performance_schema	utf8mb4	utf8mb4_0900_...	System	
	sys	utf8mb4	utf8mb4_0900_...	System	
	cineboletos	utf8mb3	utf8mb3_gener...	User	Abrir en Power BI

Paso 3.4: App Services para los Microservicios

Para cada microservicio del backend (apigateway, peliculas, ventas y compras), se aprovisionó un recurso de **Azure App Service**. Estos servicios son la plataforma donde se ejecutan los contenedores.

- **Publicación:** Se configuraron para publicar desde un **Contenedor de Docker**.
- **Sistema Operativo:** Se seleccionó **Linux**.
- **Plan de App Service:** Todos los servicios se asociaron a un plan de App Service con la SKU F1 (nivel gratuito), suficiente para un entorno de desarrollo y pruebas.

Ahora, cada **App Service** se visualiza de la siguiente manera, ya con la **imagen de Docker configurada**, esa parte de configuración se explico en la practica pasada.



The screenshot shows the Azure App Services interface. At the top, there's a header with 'App Services' and a sub-header 'Instituto Politécnico Nacional'. Below this is a toolbar with various actions like 'Crear', 'Administrar aplicaciones eliminadas', 'Actualizar', etc. A message indicates a new version of the experience is available. Below the message is a filter bar with 'Filtrar por cualquier ca...' and several active filters: 'Suscripción es igual a todo', 'Grupo de recursos es igual a todo', 'Tipo es igual a todo', and 'Ubicación es igual a todo'. The main content is a table with the following columns: 'Nombre', 'Estado', 'Ubicación', 'Plan de tarifa', 'Plan de App Service', 'Suscripción', and 'Tipo de aplicación'. The table lists four services: 'app-apigateway', 'app-compras', 'app-peliculas', and 'app-ventas'. All services are in a 'Detenido' (Stopped) state, located in 'Canada Central', on a 'Gratis' (Free) plan, using the 'plan-cinefox-gratuito' App Service plan, under 'Azure subscription 1', and are of type 'Aplicación web' (Web application).

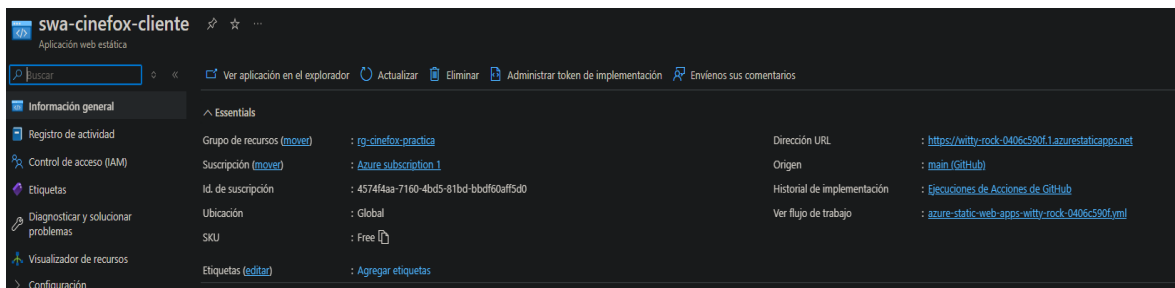
Nombre	Estado	Ubicación	Plan de tarifa	Plan de App Service	Suscripción	Tipo de aplicación
app-apigateway	Detenido	Canada Central	Gratis	plan-cinefox-gratuito	Azure subscription 1	Aplicación web
app-compras	Detenido	Canada Central	Básico	ASP-Panadenagroup-9063	Azure subscription 1	Aplicación web
app-peliculas	Detenido	Canada Central	Gratis	plan-cinefox-gratuito	Azure subscription 1	Aplicación web
app-ventas	Detenido	Canada Central	Gratis	plan-cinefox-gratuito	Azure subscription 1	Aplicación web

Paso 3.5: Azure Static Web App para el Frontend (PWA)

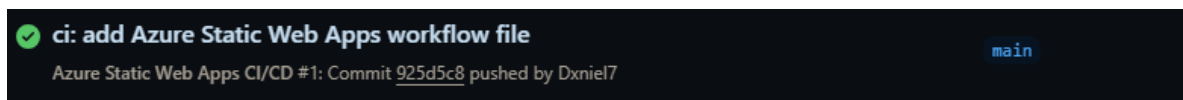
Para el despliegue del frontend, se eligió el servicio **Azure Static Web Apps**, optimizado para alojar aplicaciones web estáticas como las creadas con React, Angular, Vue o, en este caso, HTML, CSS y JavaScript puro.

- **Origen del Despliegue:** Se configuró para conectarse directamente al repositorio de GitHub del proyecto y a la rama main.
- **Configuración de Compilación:**
 - **Ubicación de la aplicación:** Se especificó la ruta a la carpeta del frontend: `./cine-cliente`.
 - **Ubicación de la API y Salida:** Se dejaron en blanco, ya que el proceso de compilación es simple y no genera una carpeta de salida build o dist separada.

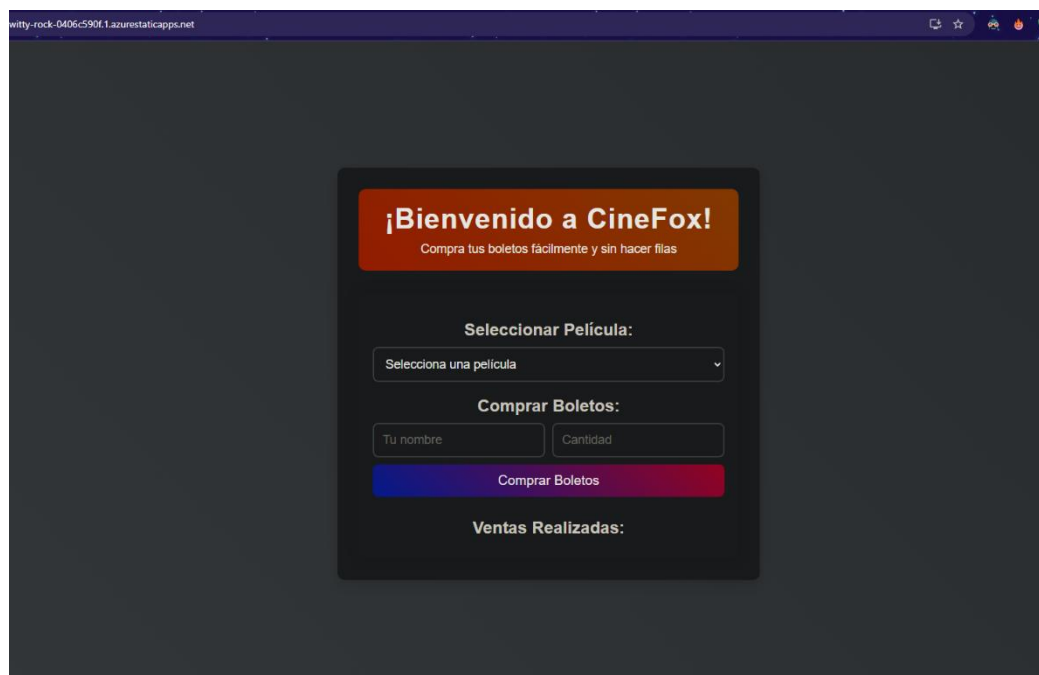
Al completar la creación de este recurso, Azure automáticamente generó y añadió un archivo de workflow de GitHub Actions (`azure-static-web-apps-....yaml`) al repositorio, estableciendo el pipeline de CI/CD para el frontend.



Desde GitHub, se generó el **pipeline de CI/CD (Integración Continua/Entrega Continua)**. Además, este proceso nos proporcionó una **URL** donde el servicio está accesible.



Y al acceder a la URL proporcionada por Azure, por ahora solo podemos ver el sitio de manera estática, puro diseño.



Al concluir esta fase, todos los "cascarones" de la infraestructura estaban creados y listos en Azure, esperando ser configurados y poblados con la aplicación.

Fase 4: Adaptación del Código y Pipeline de CI/CD

Esta fase es el núcleo del proceso de despliegue, donde se conecta el código fuente con la infraestructura de Azure. Se divide en dos partes principales: primero, modificar la aplicación

para que funcione en un entorno de nube distribuido; y segundo, automatizar por completo el proceso de despliegue utilizando GitHub Actions.

Parte A: Adaptación del Código para el Entorno de Nube

El código, que funcionaba correctamente en un entorno local, requería modificaciones para poder operar en la nube, donde cada servicio se ejecuta de forma independiente y las direcciones localhost ya no son válidas.

Paso 4.1: Parametrizar la Conexión a la Base de Datos

Los microservicios de películas, ventas y compras estaban configurados para conectarse a una base de datos en localhost. Se modificó el código de conexión de Sequelize en cada uno de estos servicios para que leyera las credenciales desde **variables de entorno** (process.env), un método seguro y flexible.

Código de Conexión Corregido (Ejemplo de compras-server.js):

```
// --- Configuración de la Conexión a la Base de Datos con Sequelize (CORREGIDO) ---
const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASS,
  {
    host: process.env.DB_HOST,
    dialect: 'mysql',
    logging: msg => logger.info(msg),
    define: { timestamps: false },
    dialectOptions: {
      ssl: {
        require: true,
        rejectUnauthorized: false
      }
    }
  }
);
```

La sección dialectOptions fue especialmente importante, ya que habilita la conexión segura (SSL) que Azure Database for MySQL requiere por defecto. Y ese proceso se realizó con cada servicio que llegara a utilizar la conexión a la BD como películas-service y ventas-service.

Cabe mencionar que estos valores de DB_HOST, DB_NAME, DB_USER y DB_PASS, lo tuvimos que colocar en cada App Service, en el apartado de variables de entorno.

DB_HOST	mysql-cinefox.mysql.database.azure.com
DB_NAME	cineboletos
DB_PASS	Mostrar valor
DB_USER	cinefoxadmin

Paso 4.2: Dinamizar las URLs en el API Gateway

De manera similar, el API Gateway necesitaba saber las direcciones de los otros microservicios en la nube. Se modificó el archivo `apigateway-server.js` para que, en lugar de apuntar a `localhost`, obtuviera las URLs de los servicios de películas, ventas y compras desde variables de entorno, las cuales colocamos en el App service del Api Gateway, en el apartado de variables de entorno.

The screenshot shows the 'Variables de entorno' (Environment variables) page for the 'app-apigateway' web application in the Azure portal. The page has a sidebar on the left with navigation links like 'Introducción', 'Registro de actividad', 'Control de acceso (IAM)', etc. The main area is titled 'Configuración de aplicación' and contains a table of environment variables. At the top of the table, there are buttons for '+ Agregar', 'Actualizar', 'Mostrar valores', 'Edición avanzada', and 'Extraer valores de referencia'. The table has two columns: 'Nombre' (Name) and 'Valor' (Value).

Nombre	Valor
COMPRAS_SERVICE_URL	https://app-compras-creudycqb7acch7.canadacentral-01.azurewebsites.net
DOCKER_REGISTRY_SERVER_PASSWORD	Mostrar valor
DOCKER_REGISTRY_SERVER_URL	Mostrar valor
DOCKER_REGISTRY_SERVER_USERNAME	Mostrar valor
PELICULAS_SERVICE_URL	https://app-peliculas-g6czdzhffgbceea7.canadacentral-01.azurewebsites.net
VENTAS_SERVICE_URL	https://app-ventas-hgdaekeebncravdz.canadacentral-01.azurewebsites.net
WEBSITES_ENABLE_APP_SERVICE_STORAGE	Mostrar valor

Paso 4.3: Actualizar el Frontend para Consumir la API en la Nube

El último ajuste en el código fue en la PWA. En el archivo `cine-cliente/script.js`, todas las llamadas `fetch` que apuntaban a `http://localhost:3000` fueron actualizadas para apuntar a la URL pública del API Gateway desplegado en Azure, únicamente la copiamos y fuimos cambiando cada llamada `fetch`.

```

// Función para obtener las películas
function obtenerPelículas() {
  fetch('https://app-apigateway-gnabbzdfd8h8g4hd.canadacentral-01.azurewebsites.net/api/peliculas')
    .then(response => response.json())
    .then(peliculasData => {
      peliculas.length = 0; // Limpiar el array de películas
      const selectPelicula = document.getElementById("peliculaSeleccionada");
      selectPelicula.innerHTML = '<option value="">Selecciona una película</option>'; // Limpiar antes de añadir
      peliculasData.forEach(pelicula => {
        peliculas.push(pelicula);
        const option = document.createElement("option");
        option.value = pelicula.nombre;
        option.textContent = `${pelicula.nombre} - Stock: ${pelicula.stock}`;
        selectPelicula.appendChild(option);
      });
    })
    .catch(error => console.error('Error al obtener las películas', error));
}

// Función para obtener las ventas realizadas
function obtenerVentas() {
  fetch('https://app-apigateway-gnabbzdfd8h8g4hd.canadacentral-01.azurewebsites.net/api/ventas') // Obtener ventas desde el backend
    .then(response => response.json())
    .then(ventasData => {
      ventas = ventasData; // Guardar las ventas en el array
      actualizarVentas(); // Actualizar la interfaz con las ventas obtenidas
    })
    .catch(error => console.error('Error al obtener las ventas', error));
}

```

Paso 4.4: Configurar las Variables de Entorno en Azure

Para que los cambios en el código funcionaran, se "inyectaron" los valores correspondientes en la configuración de cada App Service en Azure. A través del portal, en la sección Configuración > Configuración de la aplicación de cada servicio, se añadieron las variables (ej. DB_HOST, PELICULAS_SERVICE_URL, etc.) con sus respectivos valores (la URL del servidor de la base de datos, la URL del App Service de películas, etc.). Como se vio anteriormente.

Parte B: Automatización del Despliegue (CI/CD) con GitHub Actions

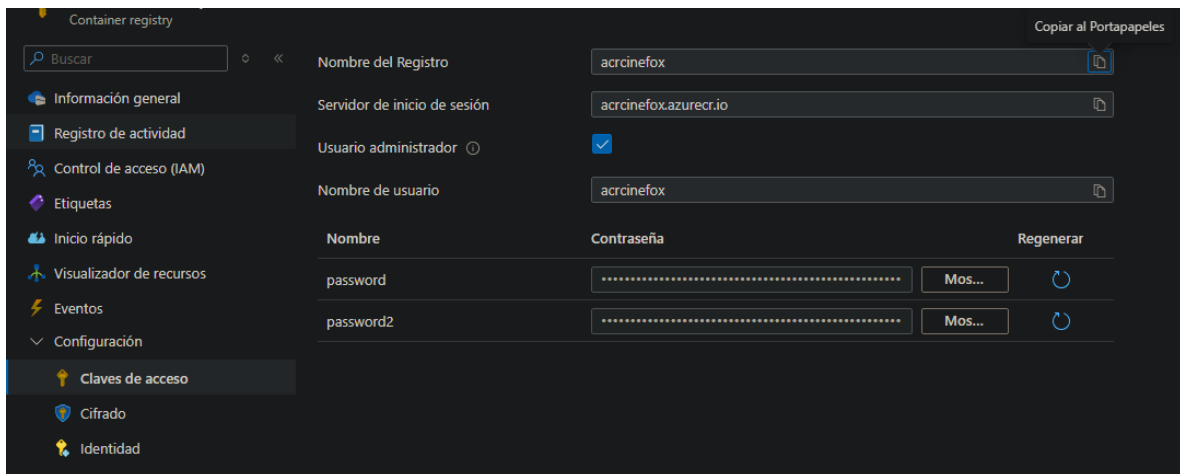
El objetivo final era que cualquier cambio en el código subido a GitHub se desplegara automáticamente en Azure. Para ello, se configuró un pipeline de Integración y Despliegue Continuo (CI/CD) utilizando GitHub Actions.

Paso 4.5: Configuración de la Conexión Segura entre GitHub y Azure (Secrets)

Para que el flujo de trabajo de GitHub Actions pudiera acceder a recursos privados en Azure (específicamente, para subir imágenes al Azure Container Registry - ACR), era necesario establecer un método de autenticación seguro. Almacenar credenciales directamente en los archivos de workflow es una mala práctica de seguridad.

La solución fue utilizar los **secretos encriptados** de GitHub.

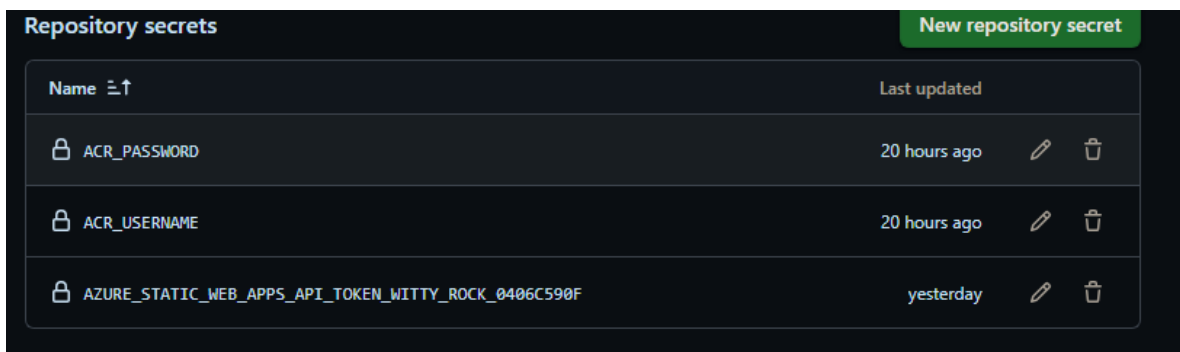
1. **Obtención de Credenciales desde Azure:** En el Portal de Azure, se navegó al recurso ACR (acrcinefox) y se seleccionó la opción "**Claves de acceso**". Allí, se habilitó la opción de "**Usuario administrador**", lo que reveló un nombre de usuario y una contraseña para el registro.



2. **Creación de Secretos en GitHub:** En la configuración del repositorio de GitHub, bajo Settings > Secrets and variables > Actions, se crearon dos secretos de repositorio:

- ACR_USERNAME
- ACR_PASSWORD

En cada uno, se pegó el valor correspondiente obtenido del ACR de Azure. De esta manera, los workflows pueden autenticarse de forma segura utilizando la sintaxis `${{ secrets.ACR_USERNAME }}` sin exponer nunca las credenciales en el código.



Paso 4.6: Creación de los Workflows para los Microservicios

Dentro de la carpeta `.github/workflows` del repositorio, se creó un archivo de workflow (.yaml) para cada microservicio del backend. Estos archivos definen un pipeline de Integración Continua (CI) que se activa automáticamente con cada push a la rama main en la carpeta del servicio correspondiente. Las tareas que realiza son:

1. **Login al ACR:** Inicia sesión de forma segura en el Azure Container Registry.

2. **Construir la Imagen Docker:** Ejecuta el comando docker build utilizando el Dockerfile del servicio.
3. **Subir la Imagen Docker:** Publica (push) la nueva imagen construida al ACR con la etiqueta :latest.

Ejemplo de Workflow (apigateway.yml):

```
.github > workflows > ! apigateway.yml
1  name: Construir y Subir - API Gateway
2
3  on:
4    push:
5      branches: [ "main" ]
6      paths:
7        - 'apigateway/**' # Se activa solo si hay cambios en esta carpeta
8
9  env:
10   # Asegúrate de que este valor coincida EXACTAMENTE con tu servidor de ACR
11   ACR_LOGIN_SERVER: acrcinefox.azurecr.io
12   IMAGE_NAME: apigateway
13
14  jobs:
15    build-and-push:
16      runs-on: ubuntu-latest
17      steps:
18        - name: Checkout del código
19          uses: actions/checkout@v3
20
21        - name: Login a Azure Container Registry
22          uses: azure/docker-login@v1
23          with:
24            login-server: ${ env.ACR_LOGIN_SERVER }
25            username: ${ secrets.ACR_USERNAME }
26            password: ${ secrets.ACR_PASSWORD }
27
28        - name: Construir y subir imagen a ACR
29          run: |
30            docker build ./apigateway -t ${ env.ACR_LOGIN_SERVER }/${ env.IMAGE_NAME }:latest
31            docker push ${ env.ACR_LOGIN_SERVER }/${ env.IMAGE_NAME }:latest
```

Paso 4.7: Habilitación del Despliegue Continuo (CD) basado en Webhooks

Para la etapa final del Despliegue Continuo, se implementó una estrategia de "pull" en lugar de "push". En este modelo, el servicio de destino (App Service) es responsable de "jalar" la nueva imagen, en lugar de que el pipeline de CI tenga que "empujarla" activamente. Este enfoque es más seguro y desacoplado.

El proceso se configuró en el Portal de Azure para cada App Service del backend:

1. Se navegó al "**Centro de implementación**" del App Service.
2. Como origen (Source), se seleccionó "**Container Registry**".

3. Se configuraron los detalles del registro, apuntando al ACR, a la imagen específica del servicio (ej. apigateway) y a la etiqueta latest.
4. El paso crucial fue activar el interruptor de **"Implementación continua" (Continuous deployment)**.

Al activar esta opción, Azure configura automáticamente un **webhook** en el Azure Container Registry. Este webhook "escucha" los cambios en la imagen y etiqueta especificadas. Cuando el pipeline de GitHub Actions sube una nueva versión de la imagen con la etiqueta :latest, el webhook se dispara y notifica al App Service. Inmediatamente, el App Service procede a descargar la nueva imagen y a reiniciarse con el contenedor actualizado, completando el ciclo de despliegue de forma automática y segura.

En este caso vemos como ejemplo, el App Service de Compras-service, donde se configuro dicho apartado.

The screenshot shows the 'Configuración del registro' (Registry Configuration) page in the Azure Portal. The left sidebar contains navigation options like 'Microsoft Defender for Cloud', 'Eventos', 'Servicios recomendados', 'Secuencia de registro', 'Visualizador de recursos', 'Implementación', 'Espacios de implementación', 'Centro de implementación', 'Configuración', 'Variables de entorno', 'Configuración', 'Autenticación', 'Identidad', 'Copias de seguridad', 'Dominios personalizados', 'Certificados', 'Redes', and 'Escalar verticalmente'. The main content area is titled 'Configuración del registro' and includes the following fields:

- Tipo de contenedor:** Contenedor único (dropdown)
- Origen del registro:** Azure Container Registry (dropdown)
- Id. de suscripción ***: Azure subscription 1 (dropdown)
- Autenticación:** Credenciales de administrador (radio button), Identidad administrada (radio button, selected)
- Identidad ***: ua-id-a40c (dropdown)
- Registro ***: acrcinefox (dropdown)
- Imagen ***: compras-service (text input)
- Etiqueta ***: latest (text input)
- Archivo o comando de i...**: (empty text input)
- Implementación continua:** Activado (radio button, selected), Desactivado (radio button)

A warning message is displayed: 'Al usar la identidad administrada, los campos de imagen y etiqueta no se rellenarán automáticamente. Escriba manualmente la imagen y la etiqueta a continuación.'

Fase 5: Verificación y Resultados Finales

Una vez completadas todas las fases de configuración, el último paso consistió en verificar que el sistema completo funcionara correctamente de extremo a extremo y que la automatización se ejecutara según lo esperado.

Paso 5.1: Monitoreo del Pipeline en GitHub Actions

La primera verificación se realizó en el repositorio de GitHub. Al navegar a la pestaña **"Actions"**, se pudo observar el estado de todos los flujos de trabajo (workflows):

- Un workflow para cada microservicio del backend (apigateway, peliculas, etc.), que se activan con cambios en sus respectivas carpetas.
- Un workflow para la PWA (Azure Static Web Apps CI/CD), que se activa con cambios en la carpeta /cine-cliente.

Se confirmó que, tras un push a la rama main, los workflows correspondientes se ejecutaban y finalizaban con una marca de verificación verde, indicando que las imágenes Docker se construyeron y subieron al ACR exitosamente, y que la PWA se desplegó sin errores.

Workflow Name	Branch	Status
CHORE: Mejorar estrategia de caché del Service Worker	main	Success
FIX: Añadir sequelize como dependencia a cada servicio	main	Success
FIX: Añadir sequelize como dependencia a cada servicio	main	Success
FIX: Añadir sequelize como dependencia a cada servicio	main	Success
FIX: Añadir sequelize como dependencia a cada servicio	main	Success
FIX: Añadir sequelize como dependencia a cada servicio	main	Success
FIX: Añadir protocolo https a las URLs del API Gateway	main	Success
FIX: Parametrizar BD en todos los servicios backend	main	Success
FIX: Parametrizar BD en todos los servicios backend	main	Success

Paso 5.2: Revisión de Logs en Azure App Service

La segunda verificación se llevó a cabo en el Portal de Azure. Para cada uno de los App Services de los microservicios, se utilizó la herramienta **"Secuencia de registro" (Log stream)**. Esta consola en tiempo real nos permitió una visibilidad instantánea del comportamiento de los contenedores y las aplicaciones.

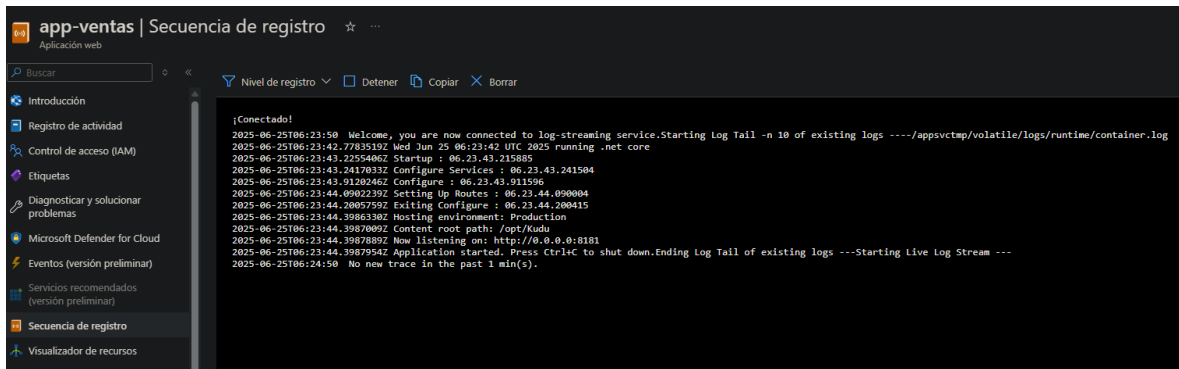
A través de esta revisión exhaustiva, se pudo confirmar lo siguiente para cada microservicio:

- El App Service descargó exitosamente la imagen más reciente desde el ACR.
- El contenedor se inició sin errores.

- La aplicación Node.js dentro del contenedor arrancó correctamente y comenzó a escuchar en el puerto configurado, mostrando los mensajes de log definidos en el código (ej. "API Gateway escuchando en el puerto...").

A continuación, se detallan los logs de cada App Service:

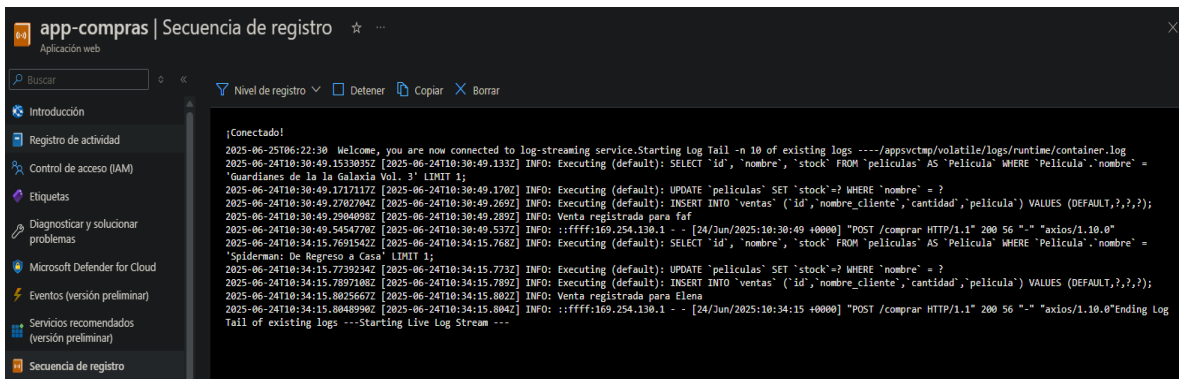
Logs del App Service: app-ventas



The screenshot shows the 'app-ventas' log stream in the Azure portal. The log output includes the following messages:

```
[Conectado!
2025-06-25T06:23:50 Welcome, you are now connected to log-streaming service.Starting Log Tail -n 10 of existing logs ----/appsvc/volatile/logs/runtime/container.log
2025-06-25T06:23:42.7783519Z Wed Jun 25 06:23:42 UTC 2025 running .net core
2025-06-25T06:23:43.2254867Z Startup : 06:23:43.215885
2025-06-25T06:23:43.2417033Z Configure Services : 06:23:43.241504
2025-06-25T06:23:43.9120246Z Configure : 06:23:43.911596
2025-06-25T06:23:44.0902239Z Setting Up Routes : 06:23:44.090004
2025-06-25T06:23:44.2805759Z Exiting Configure : 06:23:44.280415
2025-06-25T06:23:44.3986330Z Hosting environment: Production
2025-06-25T06:23:44.3987009Z Content root path: /opt/kudu
2025-06-25T06:23:44.3987809Z Now listening on: http://0.0.0.0:8181
2025-06-25T06:23:44.3987954Z Application started. Press Ctrl+C to shut down.Ending Log Tail of existing logs ---Starting Live Log Stream ---
2025-06-25T06:24:50 No new trace in the past 1 min(s).]
```

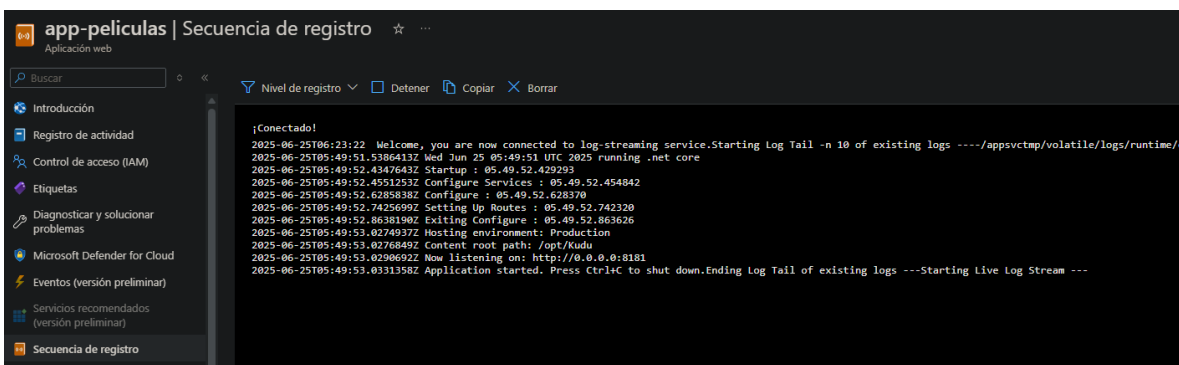
Logs del App Service: app-compras



The screenshot shows the 'app-compras' log stream in the Azure portal. The log output includes the following messages:

```
[Conectado!
2025-06-25T06:22:30 Welcome, you are now connected to log-streaming service.Starting Log Tail -n 10 of existing logs ----/appsvc/volatile/logs/runtime/container.log
2025-06-24T10:30:49.1533035Z [2025-06-24T10:30:49.133Z] INFO: Executing (default): SELECT 'id', 'nombre', 'stock' FROM 'películas' AS 'Película' WHERE 'Película'. 'nombre' = 'Guardianes de la la Galaxia Vol. 3' LIMIT 1;
2025-06-24T10:30:49.1717117Z [2025-06-24T10:30:49.170Z] INFO: Executing (default): UPDATE 'películas' SET 'stock'=? WHERE 'nombre' = ?
2025-06-24T10:30:49.2702704Z [2025-06-24T10:30:49.269Z] INFO: Executing (default): INSERT INTO 'ventas' ('id','nombre_cliente','cantidad','película') VALUES (DEFAULT,?,?,?);
2025-06-24T10:30:49.2904988Z [2025-06-24T10:30:49.289Z] INFO: Venta registrada para fef
2025-06-24T10:30:49.5459780Z [2025-06-24T10:30:49.537Z] INFO: ::ffff:169.254.130.1 - - [24/Jun/2025:10:30:49 +0000] "POST /comprar HTTP/1.1" 200 56 "-" "axios/1.10.0"
2025-06-24T10:34:15.7691542Z [2025-06-24T10:34:15.768Z] INFO: Executing (default): SELECT 'id', 'nombre', 'stock' FROM 'películas' AS 'Película' WHERE 'Película'. 'nombre' = 'Spiderman: De Regreso a Casa' LIMIT 1;
2025-06-24T10:34:15.7739234Z [2025-06-24T10:34:15.773Z] INFO: Executing (default): UPDATE 'películas' SET 'stock'=? WHERE 'nombre' = ?
2025-06-24T10:34:15.7897108Z [2025-06-24T10:34:15.789Z] INFO: Executing (default): INSERT INTO 'ventas' ('id','nombre_cliente','cantidad','película') VALUES (DEFAULT,?,?,?);
2025-06-24T10:34:15.8025667Z [2025-06-24T10:34:15.802Z] INFO: Venta registrada para Elena
2025-06-24T10:34:15.8048900Z [2025-06-24T10:34:15.804Z] INFO: ::ffff:169.254.130.1 - - [24/Jun/2025:10:34:15 +0000] "POST /comprar HTTP/1.1" 200 56 "-" "axios/1.10.0"Ending Log Tail of existing logs --Starting Live Log Stream ---]
```

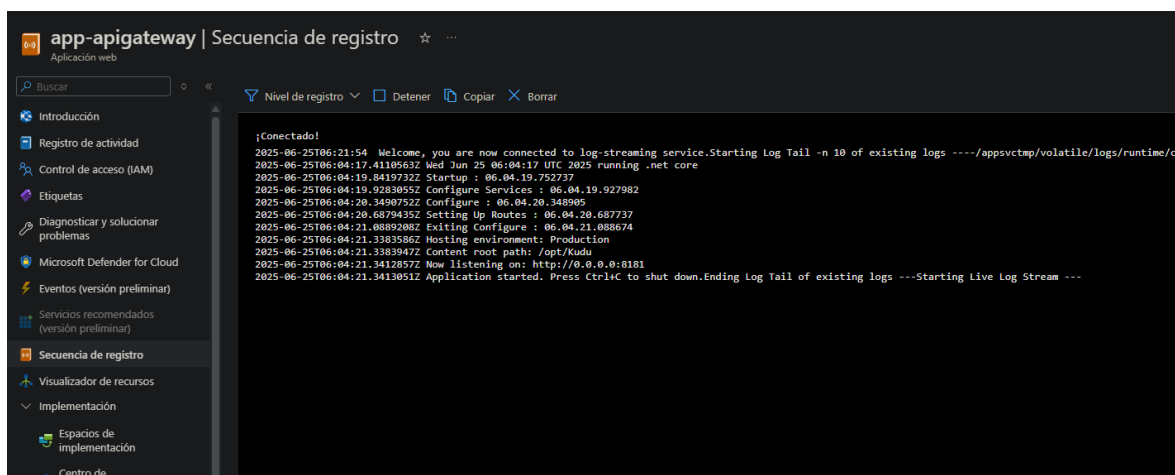
Logs del App Service: app-películas



The screenshot shows the 'app-películas' log stream in the Azure portal. The log output includes the following messages:

```
[Conectado!
2025-06-25T06:23:22 Welcome, you are now connected to log-streaming service.Starting Log Tail -n 10 of existing logs ----/appsvc/volatile/logs/runtime/c
2025-06-25T05:49:51.5386413Z Wed Jun 25 05:49:51 UTC 2025 running .net core
2025-06-25T05:49:52.4347643Z Startup : 05:49:52.429293
2025-06-25T05:49:52.4551253Z Configure Services : 05:49:52.454842
2025-06-25T05:49:52.6285838Z Configure : 05:49:52.628370
2025-06-25T05:49:52.7425609Z Setting Up Routes : 05:49:52.742370
2025-06-25T05:49:52.8638100Z Exiting Configure : 05:49:52.863626
2025-06-25T05:49:53.0274937Z Hosting environment: Production
2025-06-25T05:49:53.0276849Z Content root path: /opt/kudu
2025-06-25T05:49:53.0290692Z Now listening on: http://0.0.0.0:8181
2025-06-25T05:49:53.031358Z Application started. Press Ctrl+C to shut down.Ending Log Tail of existing logs ---Starting Live Log Stream ---]
```

Logs del App Service: app-apigateway



Paso 5.3: Prueba Funcional de la Aplicación

La prueba final y definitiva fue la funcional. Se accedió a la URL pública proporcionada por el recurso Azure Static Web App para la PWA cine-cliente. Se realizaron las siguientes pruebas de usuario:

1. **Carga de Datos:** Se verificó que el listado de películas se cargara correctamente, confirmando la comunicación: **Frontend -> API Gateway -> peliculas-service -> Base de Datos.**

Para verificar directamente el estado de la base de datos en la nube (confirmar la creación de tablas, revisar los datos insertados por los microservicios, etc.), se estableció una conexión desde la herramienta de escritorio MySQL Workbench. Este proceso requirió una configuración específica para conectar de forma segura con el servicio de Azure.

Paso 1: Iniciar una Nueva Conexión

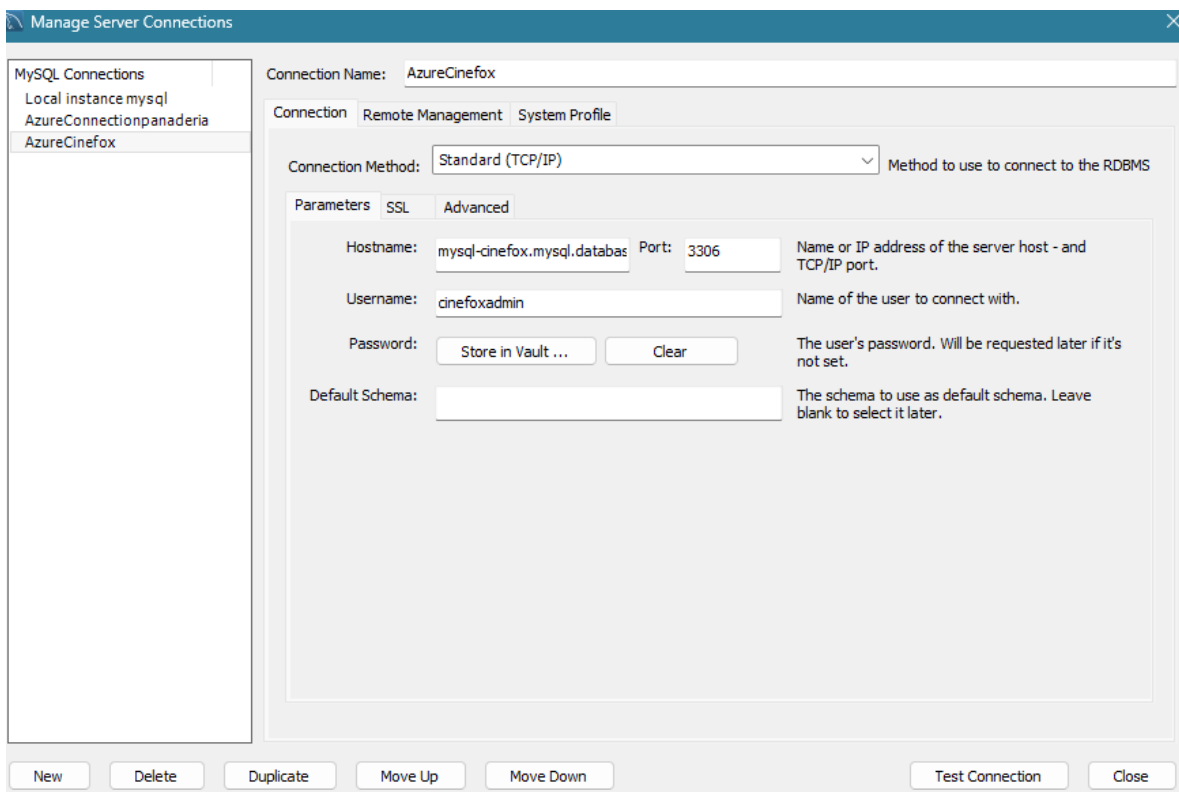
Dentro de MySQL Workbench, se hizo clic en el icono + junto a "MySQL Connections" para abrir el diálogo "Setup New Connection".

Paso 2: Configurar los Parámetros de Conexión

En la pestaña "**Parameters**", se rellenaron los siguientes campos:

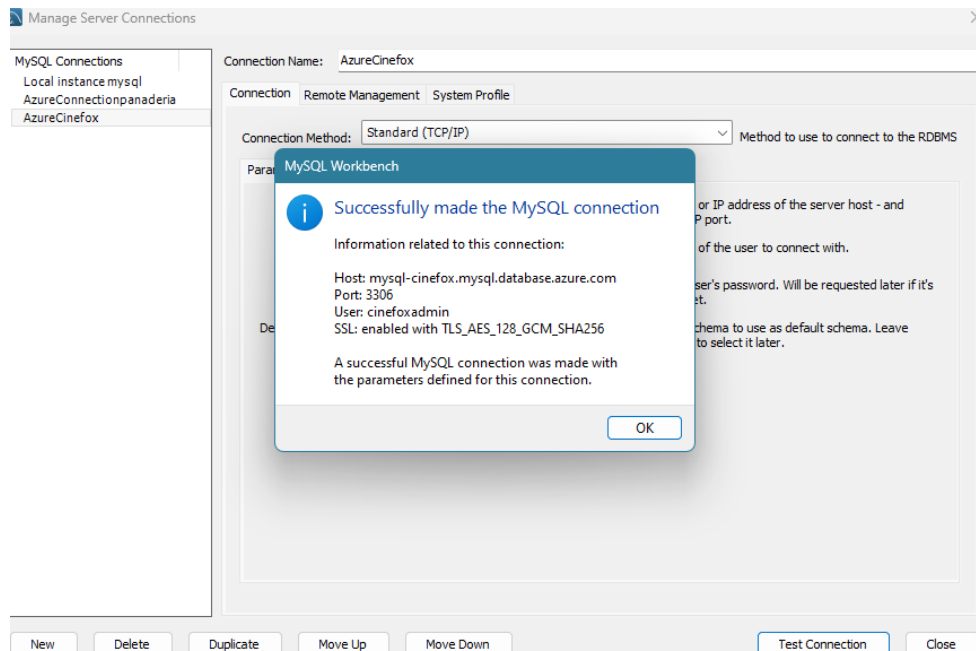
- **Connection Name:** Se asignó un nombre descriptivo para identificar la conexión, por ejemplo: Azure DB - CineFox.
- **Connection Method:** Se mantuvo el valor por defecto: Standard (TCP/IP).

- **Hostname:** En este campo se introdujo el "**Nombre del servidor**" del recurso "Azure Database for MySQL" obtenido desde el Portal de Azure. No se utilizó localhost. El valor fue: mysql-cinefox.mysql.database.azure.com.
- **Port:** Se mantuvo el puerto por defecto para MySQL: 3306.
- **Username:** Se introdujo el nombre de usuario del administrador que se creó durante el aprovisionamiento de la base de datos en Azure (ej. cinefoxadmin), **no** el usuario root local.
- **Password:** Se hizo clic en el botón "**Store in Vault...**" para guardar de forma segura la contraseña del administrador de la base de datos de Azure.



Paso 3: Probar y Guardar la Conexión

Una vez configurados los parámetros y el SSL, se hizo clic en el botón "**Test Connection**". Al ser la configuración correcta, MySQL Workbench mostró un mensaje de éxito. Tras la prueba exitosa, se guardó la conexión haciendo clic en "OK".



Paso 5: Verificación de las Tablas y Datos

Con la conexión ya guardada en la pantalla principal de MySQL Workbench, se hizo doble clic sobre ella para conectarse al servidor. Una vez dentro, en el panel izquierdo "SCHEMAS", fue posible expandir la base de datos cineboletos y verificar que las tablas películas, ventas y compras se habían creado y poblado correctamente como resultado de la ejecución de los microservicios desplegados en Azure.

id	nombre	stock
1	Spiderman: De Regreso a Casa	5
2	Doctor Strange en el Multiverso de la Locura	39
3	Guardianes de la la Galaxia Vol. 3	58
4	Avatar: El Sentido del Agua	34
	NULL	NULL

Y en el sitio web, podemos observar lo siguiente, las películas cargadas desde la BD y podemos desde ese momento seleccionar alguna y realizar nuestra compra.

¡Bienvenido a CineFox!

Compra tus boletos fácilmente y sin hacer filas

Seleccionar Película:

Selecciona una película ▼

- Selecciona una película
- Spiderman: De Regreso a Casa - Stock: 5
- Doctor Strange en el Multiverso de la Locura - Stock: 39
- Guardianes de la la Galaxia Vol. 3 - Stock: 58
- Avatar: El Sentido del Agua - Stock: 34

Ventas Realizadas:

- bbs compró 21 boleto(s) para la película Spiderman: De Regreso a Casa
- bbs compró 21 boleto(s) para la película Spiderman: De Regreso a Casa
- bbs compró 21 boleto(s) para la película Spiderman: De Regreso a Casa
- bbs compró 21 boleto(s) para la película Spiderman: De Regreso a Casa
- bbs compró 21 boleto(s) para la película Spiderman: De Regreso a Casa
- bbs compró 21 boleto(s) para la película Spiderman: De Regreso a Casa

2. **Registro de Transacciones:** Se realizó una compra de boletos, introduciendo un nombre, cantidad y seleccionando una película.

¡Bienvenido a CineFox!

Compra tus boletos fácilmente y sin hacer filas

Seleccionar Película:

Guardianes de la la Galaxia Vol. 3 - Stock: 47 ▼

Comprar Boletos:

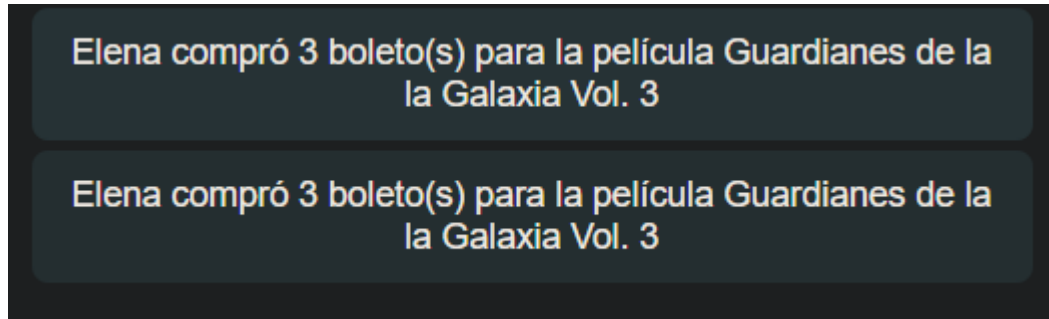
Elena 3

Comprar Boletos

Compra exitosa de 3 boleto(s) para la película Guardianes de la la Galaxia Vol. 3.

Ventas Realizadas:

Y al dar clic en “Comprar Boletos”, se disparó la lógica de negocio para procesar la transacción, interactuando con los microservicios correspondientes para registrar la venta y actualizar el inventario de películas. Además de notificar a cada usuario de la compra realizada.



3. **Actualización de Estado:** Se constató que la compra fue exitosa, el stock de la película se actualizó en la interfaz y la nueva venta apareció en la lista de "Ventas Realizadas". Esto validó el flujo completo: **Frontend -> API Gateway -> compras-service -> Base de Datos.**

Podemos ver que en la interfaz el stock se actualizo correctamente.



Y en la BD, de igual manera verificamos que el stock se actualizo correctamente.

Result Grid			
		Filter Rows:	
Edit:			
	id	nombre	stock
▶	1	Spiderman: De Regreso a Casa	5
	2	Doctor Strange en el Multiverso de la Locura	39
	3	Guardianes de la la Galaxia Vol. 3	44
	4	Avatar: El Sentido del Agua	34
*	NULL	NULL	NULL

Como pudimos ver, el proyecto se completó con éxito, logrando el despliegue íntegro de una aplicación web moderna con arquitectura de microservicios en la plataforma de Microsoft Azure. A lo largo de las fases documentadas, se alcanzaron los siguientes objetivos clave:

- **Implementación de una Arquitectura Distribuida:** Se desplegó una aplicación funcional compuesta por un frontend (PWA), un API Gateway y múltiples microservicios de backend, cada uno con su propia lógica y responsabilidad.
- **Adopción de Contenedores:** Se utilizó Docker para empaquetar los microservicios, garantizando la portabilidad y consistencia entre los entornos de desarrollo y producción.
- **Infraestructura como Código (IaC) y Automatización:** Se definió y automatizó el ciclo de vida completo de la aplicación mediante la creación de workflows de Integración Continua y Despliegue Continuo (CI/CD) con GitHub Actions.
- **Gestión de Configuración en la Nube:** Se aplicaron buenas prácticas de seguridad y flexibilidad al gestionar las configuraciones y credenciales (como las conexiones a la base de datos y las URLs de los servicios) a través de variables de entorno en Azure App Services y secretos en GitHub.

El resultado final es una aplicación robusta, escalable y mantenible, cuyo proceso de despliegue está completamente automatizado, permitiendo entregar nuevas funcionalidades y correcciones de manera rápida y fiable.

Este resultado confirmó el cumplimiento de los objetivos principales del proyecto: la correcta subida de las tres prácticas al entorno de la nube. Si bien la meta inicial era desplegar cada servicio web (que conformaban la arquitectura de microservicios) y la PWA de forma completamente separada e individual, durante el proceso de implementación identificamos que una única instancia o despliegue consolidado era una opción más factible y eficiente para alcanzar la funcionalidad deseada en Azure, simplificando significativamente el despliegue general sin comprometer la arquitectura.

AWS

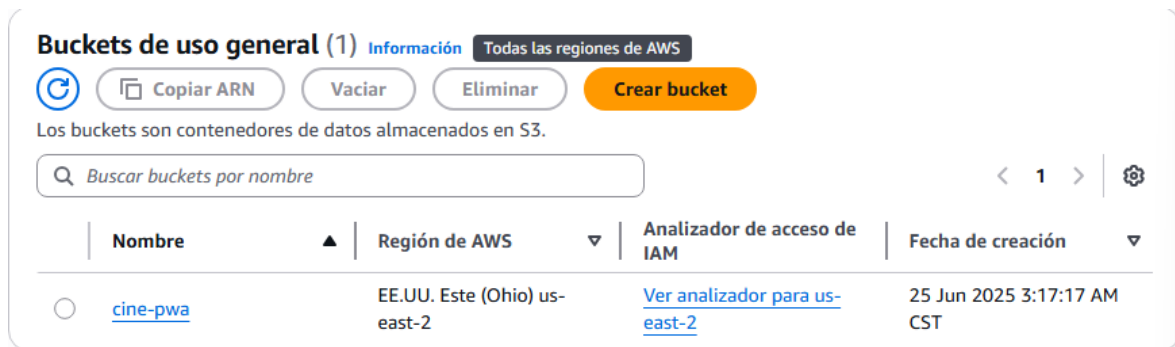
AWS (Amazon Web Services) es una plataforma de servicios en la nube que ofrece una amplia gama de soluciones informáticas a través de internet, proporcionadas por Amazon. AWS es una de las principales plataformas de servicios en la nube utilizadas por empresas y desarrolladores para almacenar, gestionar y analizar datos, así como para ejecutar aplicaciones sin necesidad de tener infraestructura física propia.

Despliegue de PWA en AWS

Una vez que hemos desarrollado nuestra aplicación web progresiva (PWA), el siguiente paso es hacerla accesible a través de internet. En nuestro caso, el despliegue de nuestra aplicación frontend se realiza utilizando Amazon S3 (Simple Storage Service), que nos permite servir archivos estáticos como HTML, CSS, JavaScript e imágenes de manera eficiente, escalable y segura. Este servicio convierte al bucket de S3 en una excelente solución para publicar nuestras PWAs en un entorno distribuido y altamente disponible.


Creación del Bucket en S3

Para comenzar, debemos acceder a la consola de AWS y navegar hasta el servicio S3. Dentro de S3, seleccionamos la opción Crear bucket, donde se nos pedirá que definamos un nombre único para el bucket. Por ejemplo, en nuestro caso pondremos el nombre “cine-pwa”. Este nombre debe ser único a nivel global, ya que S3 no permite duplicados. A continuación, seleccionamos la región deseada. En este caso, elegimos la región us-east-1 (Norte de Virginia) debido a su baja latencia y costes, aunque se recomienda elegir la misma región en la que se desplegará el backend para optimizar la velocidad de respuesta.



Luego, configuramos el bucket con acceso público, lo que permitirá que los archivos dentro del bucket sean accesibles desde cualquier navegador. Al habilitar el acceso público, debemos asegurarnos de que el bucket no tenga restricciones adicionales que bloqueen el acceso.

- ☐ **Bloquear todo el acceso público**
Activar esta configuración equivale a activar las cuatro opciones que aparecen a continuación. Cada uno de los siguientes ajustes son independientes entre sí.
- ☐ **Bloquear el acceso público a buckets y objetos concedido a través de *nuevas* listas de control de acceso (ACL)**
S3 bloqueará los permisos de acceso público aplicados a objetos o buckets agregados recientemente, y evitará la creación de nuevas ACL de acceso público para buckets y objetos existentes. Esta configuración no cambia los permisos existentes que permiten acceso público a los recursos de S3 mediante ACL.
 - ☐ **Bloquear el acceso público a buckets y objetos concedido a través de *cualquier* lista de control de acceso (ACL)**
S3 ignorará todas las ACL que conceden acceso público a buckets y objetos.
 - ☐ **Bloquear el acceso público a buckets y objetos concedido a través de políticas de bucket y puntos de acceso públicas *nuevas***
S3 bloqueará las nuevas políticas de buckets y puntos de acceso que concedan acceso público a buckets y objetos. Esta configuración no afecta a las políticas ya existentes que permiten acceso público a los recursos de S3.
 - ☐ **Bloquear el acceso público y entre cuentas a buckets y objetos concedido a través de *cualquier* política de bucket y puntos de acceso pública**
S3 ignorará el acceso público y entre cuentas en el caso de buckets o puntos de acceso que tengan políticas que concedan acceso público a buckets y objetos.

 **Desactivar el bloqueo de todo acceso público puede provocar que este bucket y los objetos que contiene se vuelvan públicos**
AWS recomienda que active la opción para bloquear todo el acceso público, a menos que se requiera acceso público para casos de uso específicos y verificados, como el alojamiento de sitios web estáticos.

☒ Reconozco que la configuración actual puede provocar que este bucket y los objetos que contiene se vuelvan públicos.

✓ **El bucket "cine-pwa" se creó correctamente** Ver detalles ✕
Para cargar archivos y carpetas, o para configurar ajustes adicionales del bucket, elija **Ver detalles**.

► **Instantánea de la cuenta: actualizada cada 24 horas** Todas las regiones de AWS
Ver panel de Storage Lens
Storage Lens permite visualizar el uso del almacenamiento y las tendencias de la actividad. Las métricas no incluyen los buckets de directorio. [Más información](#)

Subida de los Archivos del Frontend

Una vez creado el bucket, el siguiente paso consiste en subir los archivos estáticos de la PWA, tales como index.html, hojas de estilo CSS, scripts JavaScript, imágenes, íconos, el archivo manifest.json y el service-worker.js. Para hacer esto, nos dirigimos a la consola de AWS S3 y seleccionamos el bucket recién creado. Dentro del bucket, hacemos clic en el botón Subir y seleccionamos los archivos locales desde nuestra computadora. Si los archivos están organizados en una carpeta, podemos usar la opción Subir carpeta para cargar toda la estructura de la PWA en un solo paso.

Objetos (0)

 Copiar URI de S3

 Copiar URL

 Descargar

 Abrir 

Eliminar

Acciones ▼

Crear carpeta

 Cargar

Los objetos son las entidades fundamentales que se almacenan en Amazon S3. Puede utilizar el [inventario de Amazon S3](#) para obtener una lista de todos los objetos de su bucket. Para que otras personas obtengan acceso a sus objetos, tendrá que concederles permisos de forma explícita. [Más información](#)

Q Buscar objetos por prefijo

< 1 > 

☐ Nombre ▲ Tipo ▼ Última modificación ▼ Tamaño ▼ Clase de almacenamiento ▼

No hay objetos
No tiene objetos en este bucket.

 Cargar

Es importante asegurarse de que el archivo index.html se suba directamente en la raíz del bucket, ya que será el punto de entrada de la aplicación cuando los usuarios accedan a la URL pública. Si index.html se sube dentro de una subcarpeta, el despliegue no funcionará correctamente y se generará un error al intentar acceder al sitio.

Se ha realizado la carga correctamente
Para obtener más información, consulte la tabla Archivos y carpetas.

Cargar: estado

Cerrar

Después de salir de esta página, la siguiente información ya no estará disponible.

Resumen

Destino
s3://cine-pwa

Realizado correctamente
5 archivos, 11.6 KB (100.00%)

Con errores
0 archivos, 0 B (0%)

Archivos y carpetas Configuración

Archivos y carpetas (5 total, 11.6 KB)

Q Buscar por nombre

< 1 >

Nombre	Carpeta	Tipo	Tamaño	Estado	Error
index.html	-	text/html	1.6 KB	Realizado correx	-
manifest.json	-	application/json	464.0 B	Realizado correx	-
script.js	-	text/javascript	4.8 KB	Realizado correx	-
style.css	-	text/css	2.3 KB	Realizado correx	-
sw.js	-	text/javascript	2.5 KB	Realizado correx	-

cine-pwa Información

Objetos Metadatos Propiedades Permisos Métricas Administración Puntos de acceso

Objetos (6)

Los objetos son las entidades fundamentales que se almacenan en Amazon S3. Puede utilizar el [inventario de Amazon S3](#) para obtener una lista de todos los objetos de su bucket. Para que otras personas obtengan acceso a sus objetos, tendrá que concederles permisos de forma explícita. [Más información](#)

Q Buscar objetos por prefijo

< 1 > 

<input type="checkbox"/>	Nombre	Tipo	Última modificación	Tamaño	Clase de almacenamiento
<input type="checkbox"/>	icons/	Carpeta	-	-	-
<input type="checkbox"/>	index.html	html	25 Jun 2025 3:26:14 AM CST	1.6 KB	Estándar
<input type="checkbox"/>	manifest.json	json	25 Jun 2025 3:26:14 AM CST	464.0 B	Estándar
<input type="checkbox"/>	script.js	js	25 Jun 2025 3:26:14 AM CST	4.8 KB	Estándar
<input type="checkbox"/>	style.css	css	25 Jun 2025 3:26:15 AM CST	2.3 KB	Estándar
<input type="checkbox"/>	sw.js	js	25 Jun 2025 3:26:15 AM CST	2.5 KB	Estándar

Configuración de Permisos

Después de subir los archivos, debemos garantizar que sean accesibles públicamente. Para hacerlo, nos dirigimos a la pestaña Permisos del bucket y habilitamos el acceso público, lo cual garantiza que cualquier usuario pueda acceder a los archivos del bucket sin restricciones. Esta configuración es crucial, ya que permite que la PWA sea accesible desde cualquier navegador sin necesidad de autenticación adicional.

cine-pwa Información

Objetos | Metadatos | Propiedades | **Permisos** | Métricas | Administración | Puntos de acceso

Información general sobre los permisos

Búsqueda de acceso
Los analizadores de acceso externos de IAM proporcionan los hallazgos de acceso. Obtenga más información sobre [cómo funcionan los hallazgos del analizador de IAM](#).
[Ver analizador para us-east-2](#)

Bloquear acceso público (configuración del bucket) Edit

Se concede acceso público a buckets y objetos a través de listas de control de acceso (ACL), políticas de bucket, políticas de puntos de acceso o todas las anteriores. A fin de garantizar que se bloquee el acceso público a todos sus buckets y objetos de S3, active Bloquear todo acceso público. Esta configuración se aplica en exclusiva a este bucket y a sus puntos de acceso. AWS recomienda activar Bloquear todo acceso público pero, antes de aplicar cualquiera de estos ajustes, asegúrese de que sus aplicaciones funcionarán correctamente sin acceso público. Si necesita cierto nivel de acceso público a sus buckets u objetos, puede personalizar los valores de configuración individuales a continuación para que se ajusten mejor a sus necesidades específicas de almacenamiento. [Más información](#)

Bloquear todo el acceso público
⚠ **Desactivado**

▼ Configuración de bloqueo de acceso público individual para este bucket

- ☐ **Bloquear el acceso público a buckets y objetos concedido a través de nuevas listas de control de acceso (ACL)**
S3 bloqueará los permisos de acceso público aplicados a objetos o buckets agregados recientemente, y evitará la creación de nuevas ACL de acceso público para buckets y objetos existentes. Esta configuración no cambia los permisos existentes que permiten acceso público a los recursos de S3 mediante ACL.
- ☐ **Bloquear el acceso público a buckets y objetos concedido a través de cualquier lista de control de acceso (ACL)**
S3 ignorará todas las ACL que conceden acceso público a buckets y objetos.
- ☐ **Bloquear el acceso público a buckets y objetos concedido a través de políticas de bucket y puntos de acceso públicas nuevas**
S3 bloqueará las nuevas políticas de buckets y puntos de acceso que concedan acceso público a buckets y objetos. Esta configuración no afecta a las políticas ya existentes que permiten acceso público a los recursos de S3.
- ☐ **Bloquear el acceso público y entre cuentas a buckets y objetos concedido a través de cualquier política de bucket y puntos de acceso pública**
S3 ignorará el acceso público y entre cuentas en el caso de buckets o puntos de acceso que tengan políticas que concedan acceso público a buckets y objetos.

⚠ **Desactivar el bloqueo de todo acceso público puede provocar que este bucket y los objetos que contiene se vuelvan públicos**
AWS recomienda que active la opción para bloquear todo el acceso público, a menos que se requiera acceso público para casos de uso específicos y verificados, como el alojamiento de sitios web estáticos.

☒ **Reconozco que la configuración actual puede provocar que este bucket y los objetos que contiene se vuelvan públicos.**

Activación de Sitio Web Estático

Una vez configurados los permisos, el siguiente paso es activar la opción de sitio web estático en el bucket. Esto lo hacemos desde la pestaña Propiedades del bucket, donde seleccionamos Hosting de sitio web estático y proporcionamos el nombre del archivo index.html como el archivo de índice (es decir, el archivo de inicio de la aplicación). También podemos configurar un archivo de error (por ejemplo, error.html) para manejar los casos en que los usuarios intenten acceder a una página no disponible. Al activar el hosting estático, Amazon nos proporcionará una URL pública para acceder a nuestra PWA.

index.html
Información
Copiar URI de S3
Descargar
Abrir
Acciones de objetos

Propiedades
Permisos
Versiones

Información general sobre el objeto

Propietario
811e2dc2dc65701e342f14afe2d50c409d459b5a0179c07ea01609a5f553982

Región de AWS
EE.UU. Este (Ohio) us-east-2

Última modificación
25 Jun 2025 3:26:14 AM CST

Tamaño
1.6 KB

Tipo
html

Clave
index.html

URI DE S3
s3://cine-pwa/index.html

Nombre de recurso de Amazon (ARN)
arn:aws:s3::cine-pwa/index.html

Etiqueta de entidad (Etag)
6b5e91065ab44ea8bb3cea136e4751c69

URL del objeto
https://cine-pwa.s3.us-east-2.amazonaws.com/index.html

Para ello realizamos una política de bucket que es la siguiente :

Se editó correctamente la política de buckets.
Cerrar

Configuración de bloques de acceso públicos individual para este bucket

Política de bucket

La política del bucket, escrita en JSON, proporciona acceso a los objetos almacenados en el bucket. Las políticas de bucket no se aplican a los objetos que pertenecen a otras cuentas. [Más información](#)

Editar
Eliminar

Copiar

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadAll",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3::cine-pwa/*"
    }
  ]
}

```

Acceso a la PWA

Una vez completados todos estos pasos, podemos acceder a nuestra PWA a través de la URL proporcionada por Amazon S3.

LINK: <https://cine-pwa.s3.us-east-2.amazonaws.com/index.html>

Con esto, nuestra PWA estará completamente desplegada y accesible a través de la web. Además, podemos configurar Amazon CloudFront para mejorar el rendimiento y la disponibilidad global, utilizando la red de entrega de contenido (CDN) de AWS.

A continuación, se muestra el resultado obtenido:

Configuración básica AWS RDS y carga de una base de datos MySQL

Para subir una base de datos MySQL a AWS RDS, lo primero que debemos hacer es crear una instancia de RDS en la consola de AWS. Accedemos a la consola de AWS desde el enlace: <https://console.aws.amazon.com/> y luego vamos a la sección de RDS. Una vez dentro, seleccionamos "Bases de datos" y hacemos clic en "Crear base de datos". En este paso, debemos elegir el motor de base de datos MySQL y seleccionar el tipo de uso que necesitamos, ya sea para producción o para desarrollo.

Crear base de datos [información](#)

Elegir un método de creación de base de datos

☒ **Creación estándar**
Puede definir todas las opciones de configuración, incluidas las de disponibilidad, seguridad, copias de seguridad y mantenimiento.

☐ **Creación sencilla**
Utilice las configuraciones recomendadas. Algunas opciones de configuración se pueden cambiar después de crear la base de datos.

Opciones del motor

Tipo de motor [información](#)

<input type="radio"/> Aurora (MySQL Compatible)	<input checked="" type="radio"/> Aurora (PostgreSQL Compatible)	<input type="radio"/> MySQL
<input type="radio"/> PostgreSQL	<input type="radio"/> MariaDB	<input type="radio"/> Oracle
<input type="radio"/> Microsoft SQL Server	<input type="radio"/> IBM Db2	

Si estamos en el nivel Free Tier, podemos optar por la opción gratuita. A continuación, configuramos la instancia, definiendo un nombre para la base de datos, eligiendo un usuario administrador y una contraseña, seleccionando la versión de MySQL que queremos usar, y configurando el almacenamiento, que generalmente se puede dejar en 20 GB.

Motor: MySQL

☒ MySQL



Uso: Producción o desarrollo (elegir gratuito si estás en Free Tier)

☒ **Capa gratuita**
Utilice el nivel gratuito de RDS para desarrollar nuevas aplicaciones, probar aplicaciones existentes o adquirir experiencia práctica con Amazon RDS.
[Información](#)

En la sección de conectividad, es importante habilitar el acceso público para poder conectarnos a la base de datos desde nuestra computadora. También debemos crear o seleccionar un grupo de seguridad (security group) que tenga abierto el puerto 3306, y asegurarnos de que permita conexiones desde nuestra IP. Después de completar esta configuración, podemos finalizar la creación de la instancia y esperar unos minutos hasta que se active.

▼ Configuración de credenciales

Nombre de usuario maestro [Información](#)

Escriba un ID de inicio de sesión para el usuario maestro de la instancia de base de datos.

admin

1 a 16 caracteres alfanuméricos. El primer carácter debe ser una letra.

Administración de credenciales

Puede usar AWS Secrets Manager o administrar sus credenciales de usuario maestro.



Administrado en AWS Secrets Manager - *más seguro*

RDS genera una contraseña y la administra durante todo su ciclo de vida mediante AWS Secrets Manager.



Autoadministrado

Cree su propia contraseña o pida a RDS q

☐ Generar contraseña automáticamente

Amazon RDS puede generar una contraseña en su nombre, o bien puede especificar su propia contraseña.

Contraseña maestra [Información](#)

Seguridad de la contraseña

Muy fuerte

Restricciones mínimas: al menos 8 caracteres ASCII imprimibles. No puede contener ninguno de los siguientes símbolos: / ' * @

Confirmar la contraseña maestra [Información](#)

Una vez que la instancia esté activa, necesitamos obtener los datos de conexión para poder interactuar con ella. Para esto, debemos ir nuevamente a la sección de RDS, seleccionar nuestra instancia recién creada y anotar los siguientes datos: el **Endpoint** (por ejemplo, database-1.xxxxxxxx.us-east-2.rds.amazonaws.com), el **Puerto** (que usualmente es 3306), y el **Usuario** que configuramos anteriormente.

Resumen

Identificador de base de datos halibri	Estado ⌚ Backing-up	Rol Instancia	Motor MySQL Community	Recomendaciones
CPU 0.00%	Clase db.t4g.micro	Actividad actual 0 Conexiones	Región y AZ us-east-2a	

[Conectividad y seguridad](#)

Supervisión

Registros y eventos

Configuración

Integraciones sin extracción, transformación y carga (ETL)

Mantenimiento y copias de seguridad

Conectividad y seguridad

Punto de enlace y puerto

Punto de enlace
halibri.cx8mis402qd1.us-east-2.rds.amazonaws.com

Puerto
3306

Redes

Zona de disponibilidad
us-east-2a

VPC
vpc-0bb4892d9ebaa9d5a

Grupo de subredes
default-vpc-0bb4892d9ebaa9d5a

Subredes
subnet-0d5ea1441d6885da
subnet-0fa103abb04c195b1
subnet-009de81fdcf4e730f

Tipo de red
IPv4

Seguridad

Grupos de seguridad de la VPC
launch-wizard-1 (sg-06c0e47f0cc4128)
⌚ Activo
MyWebServiceRule (sg-036617a19c932fcc1)
⌚ Activo
default (sg-0c24eb2cce5fc594c)
⌚ Activo

Accesible públicamente
Sí

Entidad de certificación [Información](#)
rds-ca-rsa2048-g1

Fecha de la entidad de certificación
May 21, 2061, 18:04 (UTC-06:00)

Fecha de expiración del certificado de instancia de base de datos

Con los datos de conexión a la mano, ya podemos conectarnos a la base de datos desde nuestra computadora. Para ello, utilizamos el siguiente comando en MySQL, reemplazando <ENDPOINT> por el endpoint de nuestra base de datos, y <USUARIO> por el usuario administrador que configuramos:

```
mysql -h <ENDPOINT> -P 3306 -u <USUARIO> -p
```

Una vez conectados a la base de datos, podemos proceder a subir el archivo .sql que contiene la base de datos. Si tenemos un archivo llamado basededatos.sql, podemos cargarlo a la instancia de RDS con el siguiente comando:

```
mysql -h <ENDPOINT> -P 3306 -u <USUARIO> -p <basededatos.sql>
```

Este comando cargará las tablas, los datos y los procedimientos almacenados desde el archivo .sql directamente a la base de datos en RDS. Finalmente, es importante verificar que los datos se hayan subido correctamente revisando las tablas y la información contenida en la base de datos.

Con estos pasos, habremos subido nuestra base de datos MySQL a AWS RDS y estaremos listos para empezar a trabajar con ella.

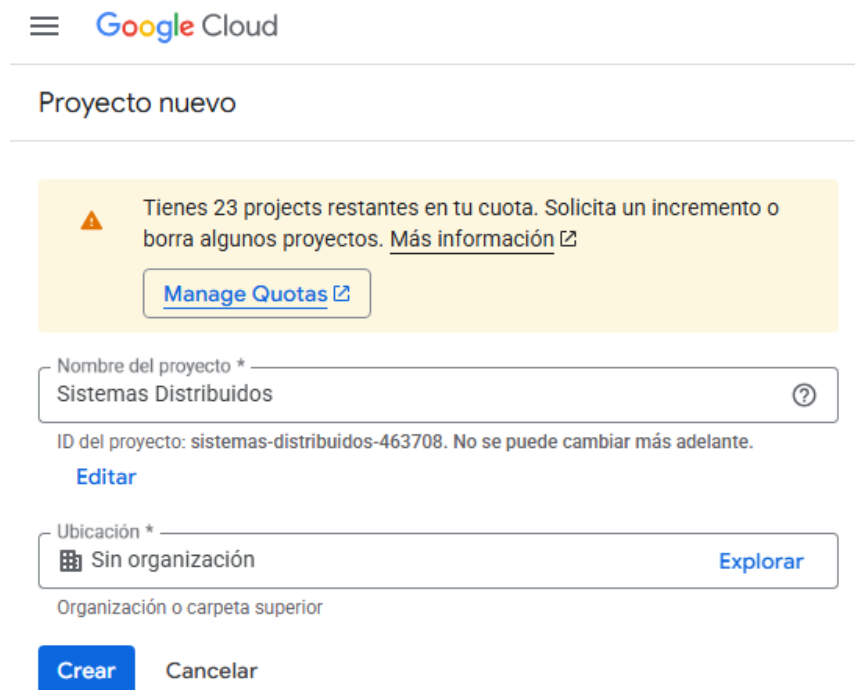
Google Cloud

Google Cloud es una plataforma de servicios en la nube proporcionada por Google, que ofrece una amplia gama de herramientas y soluciones para empresas y desarrolladores. Estos servicios permiten a las organizaciones almacenar datos, analizar grandes volúmenes de información, crear aplicaciones, gestionar infraestructuras, y mucho más, todo ello de manera flexible y escalable.

Creación de proyecto

Para comenzar a utilizar los servicios de Google Cloud, lo primero que debemos hacer es crear un nuevo proyecto en la consola. Esta acción es fundamental, ya que todos los recursos y configuraciones, como bases de datos, APIs, contenedores y aplicaciones, se organizan dentro de un proyecto específico. De este modo, podemos gestionar de manera independiente cada componente de nuestro sistema distribuido sin interferencias con otros entornos.

Desde la consola de GCP, accedemos al panel principal y seleccionamos el botón de proyectos ubicado en la parte superior de la interfaz. Al hacer clic en esta opción, se despliega un menú donde podremos ver los proyectos existentes y también crear uno nuevo. Al seleccionar “Nuevo proyecto”, se nos solicita asignar un nombre identificador, el cual debe reflejar el propósito del sistema que estamos desarrollando, en este caso le colocamos el nombre de “Sistemas distribuidos” debido a que lo utilizaremos para subir las últimas prácticas de la materia. Automáticamente se genera también un ID de proyecto único, el cual será utilizado internamente por los servicios de Google Cloud.



The screenshot shows the 'Proyecto nuevo' (New Project) form in the Google Cloud console. At the top, there's a header with the Google Cloud logo and a hamburger menu icon. Below the header, the title 'Proyecto nuevo' is displayed. A yellow warning box contains a triangle icon and the text: 'Tienes 23 projects restantes en tu cuota. Solicita un incremento o borra algunos proyectos. Más información'. Below this is a 'Manage Quotas' button. The main form has two sections: 'Nombre del proyecto *' with a text input field containing 'Sistemas Distribuidos' and a help icon, and 'Ubicación *' with a dropdown menu showing 'Sin organización' and an 'Explorar' button. Below the dropdown, it says 'Organización o carpeta superior'. At the bottom, there are two buttons: 'Crear' (Create) and 'Cancelar' (Cancel).

Google Cloud

Proyecto nuevo

Tienes 23 projects restantes en tu cuota. Solicita un incremento o borra algunos proyectos. [Más información](#)

[Manage Quotas](#)

Nombre del proyecto *
Sistemas Distribuidos

ID del proyecto: sistemas-distribuidos-463708. No se puede cambiar más adelante.

[Editar](#)

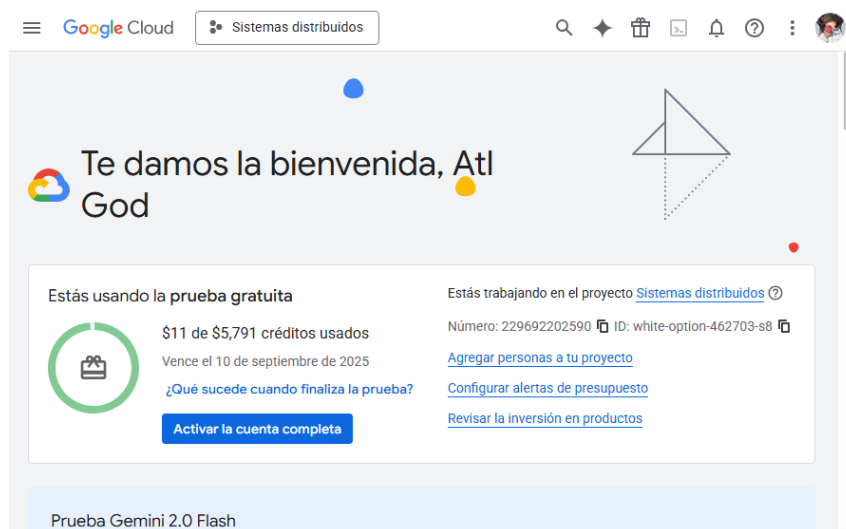
Ubicación *
Sin organización [Explorar](#)

Organización o carpeta superior

[Crear](#) [Cancelar](#)

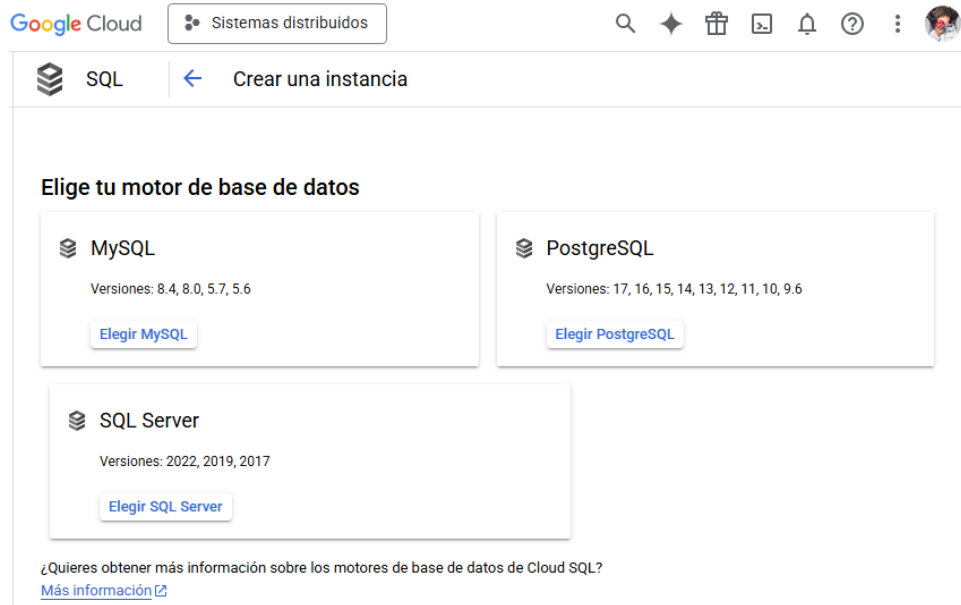
Durante la creación, también se nos ofrece la opción de asociar el proyecto a una organización, aunque si estamos trabajando desde una cuenta personal, esta configuración se asigna como “Sin organización” y no representa un inconveniente. Una vez completados los campos requeridos, procedemos a hacer clic en el botón “Crear”, lo cual inicia el aprovisionamiento automático del entorno. En pocos segundos, tendremos listo nuestro espacio de trabajo para desplegar y administrar los recursos de manera centralizada.

Esta estructura de proyectos no solo nos permite organizar los recursos, sino también establecer políticas de acceso, configurar entornos de desarrollo o producción, y realizar un seguimiento detallado del consumo. Así, logramos un entorno más seguro, escalable y controlado para nuestras aplicaciones distribuidas, lo cual es esencial en escenarios donde intervienen múltiples servicios y componentes.



Creación de base de datos en Google Cloud SQL

Una vez que contamos con un proyecto creado en Google Cloud, el siguiente paso clave para nuestra arquitectura distribuida es la creación de una base de datos. Para ello, utilizamos el servicio Cloud SQL, que nos permite implementar motores de base de datos como MySQL, PostgreSQL o SQL Server completamente administrados por Google. Este servicio resulta ideal para aplicaciones web o móviles, ya que nos garantiza alta disponibilidad, respaldo automático y escalabilidad sin necesidad de gestionar directamente el hardware.



Para iniciar el proceso, accedemos al panel de navegación lateral de la consola de Google Cloud y seleccionamos la opción SQL. Luego, hacemos clic en el botón “Crear instancia” y elegimos el motor de base de datos que deseamos utilizar; en este caso, seleccionamos MySQL, que es ampliamente compatible con todas nuestras aplicaciones y servicios web que realizamos en las prácticas anteriores. Posteriormente, debemos asignar un nombre a la instancia, definir una contraseña para el usuario root y seleccionar la región en la que se alojará la base de datos. Es recomendable elegir la misma región donde se desplegará nuestra aplicación para reducir la latencia en las comunicaciones.

Información de la instancia

Versión de la base de datos *
MySQL 8.0

✓ MOSTRAR VERSIONES SECUNDARIAS

ID de instancia *
cine1

Usa letras minúsculas, números y guiones. Comienza con una letra.

Contraseña *
••••••



GENERAR

Establece una contraseña para el usuario raíz. [Más información](#)

☐ Sin contraseña

✓ POLÍTICA DE CONTRASEÑAS

Durante la configuración, también es posible ajustar características como el tipo de máquina, la cantidad de almacenamiento y las opciones de respaldo automático. Estas opciones nos permiten adaptar el rendimiento de la base de datos según los requerimientos específicos de nuestro servicio. A continuación, se muestran las configuraciones que ocupamos para realizar nuestra instancia de base de datos, estas fueron elegidas principalmente por el precio y por las características que ofrecías.

Google Cloud Sistemas distribuidos Buscar (/) recursos, docum... Buscar

← Crea una instancia de MySQL

Elige una edición de Cloud SQL

Una edición de Cloud SQL determina las características fundamentales de tu instancia. Elige la mejor opción según tu presupuesto y necesidades de rendimiento. [Más información](#)

☐ Enterprise Plus

- ANS de disponibilidad del 99.99%
- Tiempo de inactividad por mantenimiento planificado en menos de un segundo
- Escalación vertical de instancias con tiempo de inactividad casi nulo
- Máquinas con rendimiento optimizado
- Hasta 35 días para la ventana de recuperación de un momento determinado
- Capacidad de procesamiento hasta 3 veces mayor con caché de datos
- Recuperación ante

☒ Enterprise

- ANS de disponibilidad del 99.95%
- Menos de 60 segundos de tiempo de inactividad por mantenimiento planificado
- Máquinas de uso general
- Hasta 7 días para la ventana de recuperación de un momento determinado

Resumen

Edición de Cloud SQL	Enterprise
Región	us-central1 (Iowa)
Versión de la base de datos	MySQL 8.0
CPU virtuales	1 CPU virtual(es)
RAM	3.75 GB
Caché de datos	Inhabilitada
Almacenamiento	10 GB HDD
Conexiones	IP privada IP pública
Copia de seguridad	Automatizada
Disponibilidad	Zona única
Recuperación de un momento determinado	Inhabilitada
Capacidad de procesamiento de la red (MB/s)	250 de 250
IOPS	Lectura: 1,000 de 1,000 Escritura: 3,015 de 10,000
Capacidad de procesamiento del disco (MB/s)	Lectura: 1.2 de 200.0 Escritura: 1.2 de 200.0

En la siguiente figura se muestra la tabla de precios estimados que nos da Google SQL en base a los recursos de procesamiento, memoria y almacenamiento establecidos en la instancia.

Precio estimado (sin descuentos)

Estos elementos representan únicamente los recursos de procesamiento, memoria y almacenamiento de Cloud SQL y reflejan la configuración que estableciste para tu instancia hasta el momento. No se incluyen los descuentos en la estimación.

[Más información](#)

Elemento	Costo por hora (estimado)
1 CPU virtuales (USD0.041 por CPU virtual por hora)	USD0.04
3.75 GiB de RAM (USD0.007 por GiB por hora)	USD0.03
10 GiB de HDD (USD0.09 por GiB al mes)	USD0.001
Total sin descuentos por uso	USD0.07

Costos de uso y tráfico

Estos costos varían según el uso de las funciones, el tráfico o la ubicación de los datos. [Más información](#)

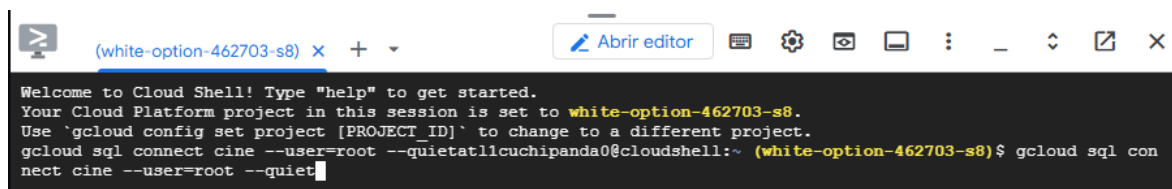
Copias de seguridad (USD0.08 por GiB)

Transferencia de datos externa de red (variable) ?

Una vez que completamos la configuración, hacemos clic en “Crear” y esperamos unos minutos mientras Google aprovisiona la instancia de Cloud SQL.

Una vez desplegada la instancia, podemos acceder a ella desde la consola y crear nuestras bases de datos personalizadas. Para esto, ingresamos a la consola Cloud Shell accedemos con los permisos correspondientes y colocamos el siguiente comando para poder acceder a nuestra base de datos que creamos anteriormente. Así como se muestra en la figura siguiente:

gcloud sql connect cine --user=root --quiet



```
(white-option-462703-s8) x + -
Abrir editor
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to white-option-462703-s8.
Use 'gcloud config set project [PROJECT_ID]' to change to a different project.
gcloud sql connect cine --user=root --quietatl1cuchipanda0@cloudshell:~ (white-option-462703-s8)$ gcloud sql connect cine --user=root --quiet
```

Posteriormente nos pedirá la contraseña que colocamos para nuestra instancia y una vez que accedamos a la instancia podremos crear la base de datos llamada “cineboletos” con los comandos que se encuentran en el archivo “cine.sql” que se encuentra anexo al final del documento.

A partir de este punto, ya podemos conectar nuestra aplicación backend a esta base de datos utilizando las credenciales definidas previamente.

Esta etapa es fundamental dentro de nuestra práctica, ya que asegura que el sistema distribuido cuente con una fuente de datos centralizada, confiable y administrada, lista para

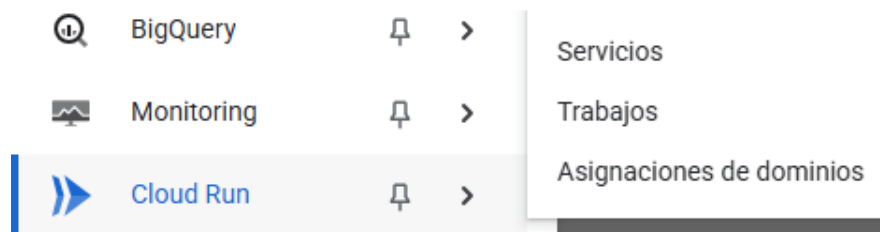
ser consumida desde cualquier microservicio, servicio web o PWA conectada. Además, al utilizar Cloud SQL, se minimizan los esfuerzos de administración, lo que nos permite enfocarnos en el desarrollo y la lógica de nuestro servicio.

Nombre de la conexión: ***white-option-462703-s8:us-central1:cine***

IP pública : ***34.136.247.46***

Despliegue de Servicio Web en Google Cloud Run

Una vez que contamos con nuestra base de datos en Cloud SQL y nuestro servicio web desarrollado en este caso con Node.js + Express.js, el siguiente paso consiste en desplegar este backend en la nube utilizando Google Cloud Run. Este servicio nos permite ejecutar aplicaciones contenedorizadas en una infraestructura totalmente gestionada por Google, con escalado automático y acceso público mediante una URL segura. Gracias a esto, podemos publicar nuestros servicios REST sin preocuparnos por servidores o configuración de red compleja.



Para iniciar el despliegue, primero debemos asegurarnos de que nuestro proyecto backend funcione correctamente de forma local. Posteriormente, es necesario crear un contenedor Docker que empaquete toda la lógica del servidor. Para ello, colocamos un archivo Dockerfile en la raíz del proyecto este archivo Docker lo podemos encontrar en el repositorio de Github “AtllGod”. Este archivo define cómo se construye la imagen del backend, especificando el entorno, las dependencias y el puerto de exposición. A continuación, se muestra esta configuración:

```

Dockerfile > ...
# Usa una imagen de Node.js
FROM node:18-slim
# Crea el directorio de la app
WORKDIR /app
# Copia los archivos necesarios
COPY package*.json ./
COPY server.js ./
# Instala dependencias
RUN npm install
# Expone el puerto
EXPOSE 3000
# Comando para iniciar la app
CMD ["npm", "start"]

```

Una vez que tenemos nuestro Dockerfile subido a Github junto con nuestro servicio web, así como se muestra en la siguiente figura:



Ahora si podremos crear nuestro servicio en Google Run para ello utilizaremos la opción de “Conectar con repositorio” alojado en Github. Esta opción nos permite automatizar completamente el proceso de despliegue, lo cual es ideal para proyectos de sistemas distribuidos donde la integración continua y el despliegue continuo (CI/CD) son fundamentales, lo cual es nuestro caso. Al habilitar esta integración, cualquier cambio que

realicemos en el repositorio, por ejemplo, al subir nuevas versiones del código o modificar parte de este, se generara automáticamente una nueva versión desplegada del servicio.

Para ello, al crear un nuevo servicio en Cloud Run, seleccionamos la opción "Implementar continuamente a partir de un repositorio (de origen o de función)". Esta opción aparece junto a otras como "Artifact Registry" o "Docker Hub", pero nos enfocamos en GitHub ya que proporciona un flujo más dinámico de trabajo. Una vez seleccionada esta opción, hacemos clic en el botón azul "Configuración con Cloud Build", lo cual nos llevará a vincular nuestra cuenta de GitHub con Google Cloud.

← Crear servicio Mostrar línea de comandos

Cada servicio expone un extremo único y ajusta automáticamente la escala de la infraestructura subyacente para controlar las solicitudes entrantes. No se puede cambiar el nombre del servicio ni la región más adelante.

Artifact Registry

Docker Hub

☐ Implementar una revisión desde una imagen de contenedor

GitHub

☒ Implementar continuamente a partir de un repositorio (de origen o de función)

Función

☐ Use an inline editor to create a function

Configuración con Cloud Build

Durante el proceso de vinculación, se nos solicitará autorizar a Google Cloud Build para que acceda a nuestros repositorios de GitHub. A continuación, elegimos el repositorio correspondiente al servicio web que deseamos desplegar, especificamos la rama que se usará en este caso es la rama main ya que no tenemos otra y configuramos el archivo Dockerfile que se encargará de construir la imagen del servicio. Es importante que el repositorio ya contenga este archivo, así como lo realizamos anteriormente, y que este ubicado en la raíz del proyecto, con las instrucciones necesarias para construir el contenedor de forma correcta.

1 Repositorio de código fuente

Proveedor del repositorio: GitHub

Repositorio *: ATL1GOD/Servicio-Web

¿No encuentras tu repositorio? [Administrar repositorios conectados](#)

☒ Entiendo que se transferirá el contenido de GitHub de los repositorios seleccionados a este proyecto de GCP para proporcionar el servicio conectado. Las principales que tengan acceso a este proyecto de GCP con permisos suficientes podrán crear y ejecutar activadores en estos repositorios en función del contenido transferido de GitHub. También comprendo que el contenido de todos los activadores de la app de GitHub del proyecto de GCP se podría transferir a GitHub a fin de proporcionar funcionalidades, como mostrar los nombres de los activadores en los resultados de la compilación de GitHub. Esta acción se aplicará a todos los activadores futuros y existentes de la app de GitHub en el proyecto. [Más información](#)

Los registros de compilación se enviarán a GitHub.

[Siguiente](#)

2 Configuración de compilación

Rama *: ^main\$

Usa una expresión regular que coincida con una rama específica [Más información](#)

Coincide con la rama: main

Tipo de compilación

☒ Dockerfile

Ubicación de origen *: /Dockerfile

Ubicación y nombre del Dockerfile. Este directorio también se usará como contexto de la compilación de Docker.

☐ Go, Node.js, Python, Java, .NET Core, Ruby o PHP, mediante los [paquetes de compilación de Google Cloud](#)

[Guardar](#)

Una vez completada la configuración, Cloud Build se encargará de clonar el repositorio, construir la imagen del contenedor y desplegarla automáticamente en Cloud Run. Al finalizar el proceso, se nos proporcionará una URL pública segura a través de la cual podremos acceder al backend. De esta manera, cualquier frontend que creemos va a poder consumir los servicios REST que hayamos implementado.

Configurar

Nombre de Servicio *
servicio-web

Región *
northamerica-south1 (México)

[¿Cómo se selecciona la región? \[?\]](#)

URL del extremo ⓘ
https://servicio-web-229692202590.northamerica-south1.run.app

Autenticación *
☒ Usar Cloud IAM para autenticar las solicitudes entrantes
Cloud IAM autorizará todas las invocaciones del extremo de este servicio.

☒ Permitir invocaciones no autenticadas
Configura una política de IAM que permite el acceso sin una credencial.

☐ Autenticación obligatoria
Administrar los usuarios autorizados con Cloud IAM.

Facturación ⓘ
☒ Basada en solicitudes
Se cobra solo cuando se procesan solicitudes. La CPU está limitada fuera de las solicitudes.

☐ Basada en instancias
Se cobra por todo el ciclo de vida de las instancias. CPU completa durante toda la vida útil de cada instancia.

Resumen de precios

Precios de Cloud Run

Nivel gratuito

Primeras 180,000 unidades de CPU virtuales segundo por mes

Primeros 360,000 GiB segundo por mes

2 millones de solicitudes por mes

→ Verificar los detalles de los niveles pagados

Para la configuración de nuestro servicio web, utilizaremos como región el servidor que se encuentra en México para evitar la latencia entre nosotros y el server, también configuraremos la autenticación para permitir la invocación sin necesidad de tener acceso mediante credenciales, esto es para fines más prácticos, pero para producción esto no se debe de realizar. Y por último como método de facturación o de pago elegiremos basado en solicitudes, ya que de esta manera el costo entra en el nivel gratuito y nos brindaran un límite de solicitudes gratis al mes, lo cual es bueno ya que al ser una práctica no realizaremos demasiadas, pero si es para producción se tendría que checar los costos y ver si nos conviene más por solicitudes o por instancias.

Creando servicio

^ Ocultar estado

Creando servicio

Completado

Configurando política de IAM

Completado

Creando activador de Cloud Build

Completado

Compilando e implementando a partir del repositorio (ver registros)

Completado

✓

servicio-web

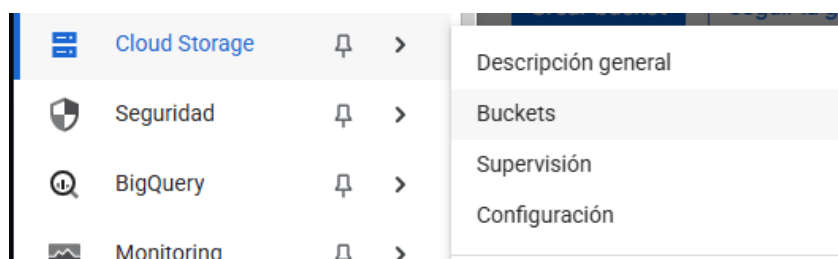
Región: northamerica-south1

Este método no solo nos facilita el despliegue inicial, sino que también nos permite establecer un flujo de integración continua, donde cada push al repositorio puede detonar automáticamente una nueva versión del servicio en producción. Esto reduce significativamente los tiempos de mantenimiento, aumenta la eficiencia del desarrollo y garantiza que nuestras aplicaciones distribuidas estén siempre actualizadas con la última versión de nuestro código.

Link de la API: <https://servicio-web-229692202590.northamerica-south1.run.app>

Despliegue de PWA en Google Cloud Storage

Una vez desarrollada nuestra aplicación web progresiva (PWA), es momento de hacerla accesible públicamente a través de internet. En nuestro caso, el despliegue de nuestro frontend se realiza utilizando Google Cloud Storage, el cual nos permite servir archivos estáticos como HTML, CSS, JavaScript e imágenes, de forma altamente escalable, segura y con baja latencia. Esto convierte al bucket de almacenamiento en una excelente solución para publicar nuestro PWAs dentro de un entorno distribuido.



Para llevar a cabo el despliegue, primero nos dirigimos al servicio Cloud Storage desde la consola de Google Cloud. Allí, seleccionamos la pestaña **Información útil**.

Elige cómo controlar el acceso a los objetos

Impedir el acceso público

Restringe el acceso público a los datos a través de Internet. Esto evitará que el bucket se use para el hosting web. [Más información](#)

☐ Aplicar la prevención de acceso público a este bucket

Información útil

Precios de ubicación

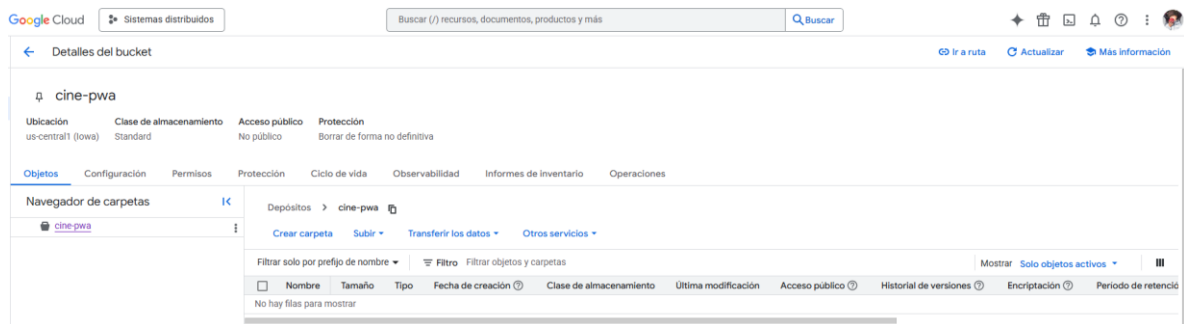
Las tarifas de almacenamiento varían según la clase de almacenamiento de los datos y la ubicación de los buckets. [Detalles de las tarifas](#)

Configuración actual: Region/Standard

Elemento	Costo
us-central1 (Iowa)	\$0.020 por GB al mes

[Estimar costo mensual](#)

Después de que hemos creado el bucket en Google Cloud Storage, el siguiente paso consiste en subir los archivos del frontend de la aplicación, es decir, todos los elementos que conforman nuestra PWA como index.html, hojas de estilo, scripts, imágenes, íconos, el manifest.json y el service-worker.js. En este caso, realizamos el procedimiento de forma sencilla y visual, utilizando directamente la interfaz web de Google Cloud.



Para comenzar, accedemos a la consola de Google Cloud y seleccionamos el bucket previamente creado “cine-pwa”. Una vez dentro, hacemos clic en el botón azul “Subir archivos” o “Subir carpeta”, dependiendo de cómo tengamos organizada nuestra PWA localmente. Si los archivos están sueltos, seleccionamos la opción “Subir archivos” y elegimos todos los elementos del frontend manualmente. Si están dentro de una carpeta, seleccionamos “Subir carpeta” y cargamos toda la estructura de la PWA con un solo clic.

Es fundamental asegurarnos de que el archivo `index.html` quede ubicado directamente en la raíz del bucket, ya que este será el punto de entrada de la aplicación cuando se acceda desde un navegador. Si `index.html` queda dentro de una subcarpeta, el despliegue no funcionará correctamente, y se mostrará un error al intentar acceder a la URL del bucket como sitio web.

Después de subir los archivos, verificamos que todos estén correctamente listados en el bucket, incluyendo el archivo `manifest.json`, los íconos y el `service-worker.js`, ya que estos son esenciales para que la aplicación se comporte como una PWA. En la siguiente figura se muestran los archivos subidos al bucket

Filtrar solo por prefijo de nombre									
Filtro Filtrar objetos y carpetas									
Mostrar Solo objetos activos									
<input type="checkbox"/>	Nombre	Tamaño	Tipo	Fecha de creación	Clase de almacenamiento	Última modificación	Acceso público	Historial de versiones	
<input type="checkbox"/>	icons/	—	Carpeta	—	—	—	—	—	⋮
<input type="checkbox"/>	index.html	1.6 KB	text/html	22 jun 2025 22:28:59	Standard	22 jun 2025 22:28:59	No público	—	⬇ ⋮
<input type="checkbox"/>	manifest.json	464 B	application/json	22 jun 2025 22:28:59	Standard	22 jun 2025 22:28:59	No público	—	⬇ ⋮
<input type="checkbox"/>	script.js	4.8 KB	text/javascript	22 jun 2025 22:28:59	Standard	22 jun 2025 22:28:59	No público	—	⬇ ⋮
<input type="checkbox"/>	style.css	2.2 KB	text/css	22 jun 2025 22:28:59	Standard	22 jun 2025 22:28:59	No público	—	⬇ ⋮
<input type="checkbox"/>	sw.js	2.6 KB	text/javascript	22 jun 2025 22:29:00	Standard	22 jun 2025 22:29:00	No público	—	⬇ ⋮

Finalmente, accedemos a la pestaña “Permisos” del bucket, y otorgamos el acceso público a todos los usuarios. Para hacerlo, hacemos clic en “Permitir acceso público” o agregamos manualmente a la entidad `allUsers` con el rol “Visualizador de objetos de almacenamiento”, en este caso para fines prácticos daremos acceso público. Esto garantiza que cualquier usuario pueda cargar el sitio desde su navegador sin restricciones.

Agregar principales

Las principales son usuarios, grupos, dominios o cuentas de servicio. [Más información sobre las principales de IAM](#)

⚠ Si agregas "allUsers", este recurso será público, y cualquier persona en Internet podrá acceder a él.

Principales nuevas * ?

Asignar roles

Los roles se componen de conjuntos de permisos y determinan lo que la principal puede hacer con este recurso. [Más información](#)

Rol *

Otorga acceso para ver los objetos y sus metadatos, excepto las LCA. También puede enumerar los objetos de un bucket.

Condición de IAM (opcional) ?

+ Agregar condición de IAM

+ Agregar otro rol

Guardar

Cancelar

Objeto publicado Historial de versiones

Descargar

Editar metadatos

Editar acceso

Borrar

Descripción general

Tipo	text/html
Tamaño	1.6 KB
Fecha de creación	22 Jun 2025 22:28:59
Última modificación	22 Jun 2025 22:28:59
Clase de almacenamiento	Standard
Tiempo personalizado	—
URL pública ?	https://storage.googleapis.com/cine-pwa/index.html
URL autenticada ?	https://storage.cloud.google.com/cine-pwa/index.html
URI de gsutil ?	gs://cine-pwa/index.html

Permisos

Acceso público ⚠ Público para Internet

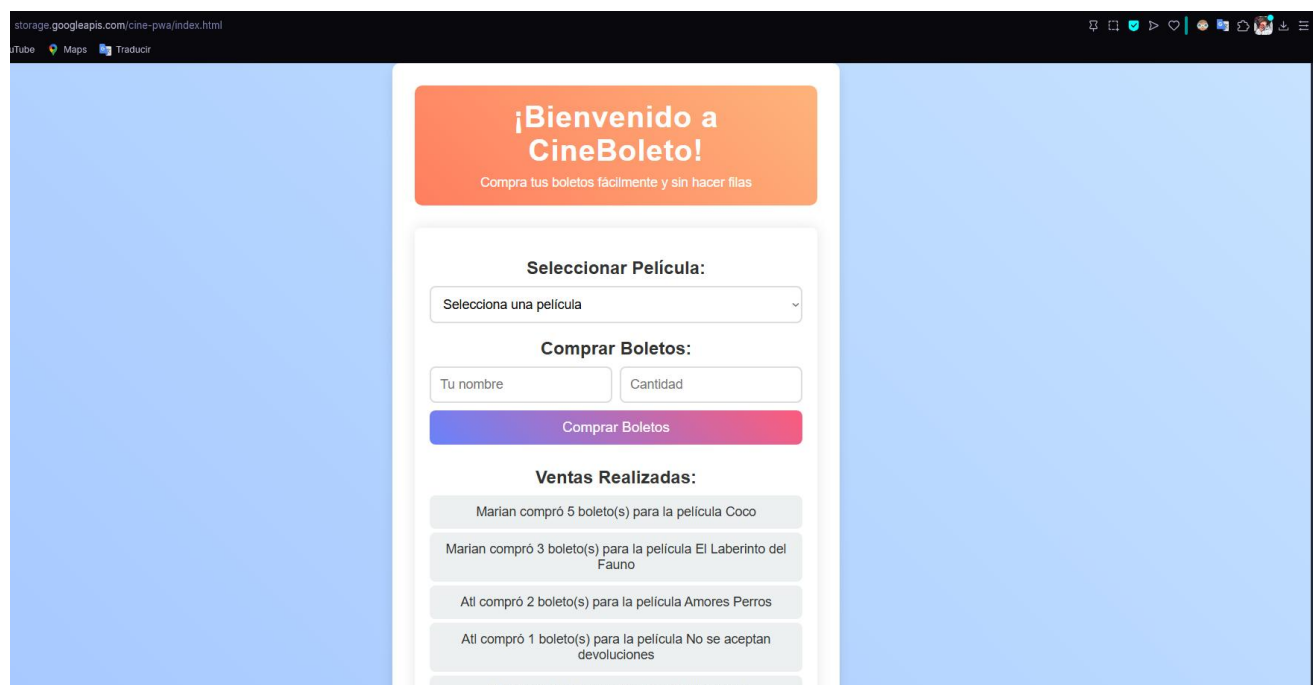
Protección

Historial de versiones ?	—
Fecha y hora de vencimiento de la retención ?	Ninguno
Periodo de retención del objeto	Ninguno
Periodo de retención del bucket	Ninguno
Estado de conservación	Ninguno
Tipo de encriptación	Administrada por Google

Una vez completado este paso, solo resta activar la opción de sitio web estático o url pública del objeto y Google nos proporcionará una URL pública donde se podrá visualizar y utilizar la PWA como si estuviera alojada en un servidor tradicional.

LINK: <https://storage.googleapis.com/cine-pwa/index.html>

A continuación, se muestra el resultado obtenido:



Conclusión

En conclusión, la práctica realizada sobre la configuración y despliegue de microservicios en la nube, utilizando plataformas como AWS, Azure y GCP, ha sido una experiencia enriquecedora que nos permitió fortalecer nuestras habilidades técnicas y adquirir una comprensión más profunda de los sistemas distribuidos. A través de esta práctica en parejas, no solo pudimos desarrollar y desplegar un microservicio utilizando Flask en una instancia virtual de AWS, sino que también aprendimos a gestionar y configurar los entornos necesarios para su funcionamiento adecuado.

El trabajo en la nube, utilizando modelos como IaaS, PaaS y SaaS, nos brindó la oportunidad de comparar las ventajas y características de cada proveedor, destacando tanto las similitudes como las diferencias clave en cuanto a su facilidad de uso, herramientas de administración y enfoque en la seguridad. Nos enfrentamos a la configuración de recursos virtuales, la gestión de redes, y la integración de medidas de seguridad, lo cual fue crucial para comprender cómo las plataformas en la nube facilitan la creación de soluciones escalables y seguras.

Además, el monitoreo de los recursos y el uso de herramientas como CloudWatch, Azure Monitor y Cloud Logging nos ayudaron a entender la importancia de una administración eficiente, lo que es fundamental en el contexto de sistemas distribuidos. De esta manera, la práctica no solo fortaleció nuestras competencias técnicas, sino que también nos preparó para tomar decisiones informadas y estratégicas al elegir un proveedor en función de las necesidades de un proyecto.

Por último, cabe mencionar que, esta experiencia ha sido esencial para comprender la relevancia de la infraestructura en la nube en el desarrollo de aplicaciones modernas y cómo las plataformas de la nube son fundamentales en la creación de soluciones distribuidas y escalables. Sin duda, nos ha proporcionado una ventaja competitiva al adquirir conocimientos clave sobre las tecnologías que dominan la industria de los sistemas distribuidos.

Referencias

- Amazon Web Services. (s.f.). Cloud Computing – Servicios de informática en la nube. <https://aws.amazon.com/es/>
- Mendieta, A. (17 de marzo de 2024). Qué es AWS: Un mundo de soluciones en la nube. Open Webinars. <https://openwebinars.net/blog/que-es-aws/>
- Microsoft Azure. (s.f.). Servicios de informática en la nube. <https://azure.microsoft.com/es-es>
- Google Cloud Platform. (s.f.). <https://cloud.google.com>
- Google Developers. (2024). Getting started with GCP. <https://developers.google.com>