UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Gutiérrez Rodríguez, Daniel; Pons Aroztegui, Jordi, tut. Desenvolupament i desplegament d'una aplicació amb React a Google Cloud Platform. 2025. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/308747

under the terms of the BY-NC-ND license

Desenvolupament i desplegament d'una aplicació amb React a Google Cloud Platform

Daniel Gutiérrez Rodríguez

Resum

Aquest projecte s'ha centrat en la resolució d'un cas pràctic típic en el món de l'informàtica on una empresa vol migrar una aplicació en local al núvol. S'ha volgut enfocar des de la perspectiva d'un arquitecte del núvol, plantejant l'anàlisi de les diverses opcions que la plataforma Google Cloud Platform (GCP) ofereix per a desplegar aplicacions, a més de seguir el procés del desenvolupament de l'aplicació web Global Track Trends i els passos del seu desplegament amb la finalitat de mostrar la rellevància del treball d'un arquitecte al núvol.

Paraules clau: Artifact Registry, Cloud Logging, Contenidor, DevOps, Google Cloud Platform (GCP), Google Compute Engine (GCE), Google Kubernetes Engine (GKE), Migration Instance Group (MIG), Minikube, Orquestració.

Abstract

This project has focused on solving a typical practical case in the computing environment, where a company wants to migrate a local application to the cloud. We wanted to focus from the perspective of a cloud architect, proposing the analysis of the various options that Google Cloud Platform (GCP) offers for deploying applications, in addition to following the development process of the Global Track Trends web application and its deployment steps in order to show the relevance of the work of a cloud architect.

Index Terms: Artifact Registry, Cloud Logging, Container, DevOps, Google Cloud Platform (GCP), Google Compute Engine (GCE), Google Kubernetes Engine (GKE), Migration Instance Group (MIG), Minikube, Orchestration.

1 Introducció - Context del treball

El creixement de les aplicacions al núvol en els darrers anys ha estat exponencial [1]. Empreses de totes les mides han migrat cap a arquitectures en el núvol (IaaS, SaaS, etc) [2] aprofitant els beneficis que aquestes ofereixen, com la flexibilitat, l'escalabilitat i la reducció de costos. Google Cloud Platform (GCP) és una de les principals plataformes de serveis al núvol, oferint múltiples solucions per al desplegament d'aplicacions entre molts altres serveis.

La motivació d'aquest treball és explorar les diferents tecnologies de còmput que ofereix Google Cloud Platform i profunditzar en el servei que s'ajusta millor a les necessitats d'una aplicació web en procés de migració al núvol, reflectint la feina d'un arquitecte al núvol i posant de manifest la seva rellevància. Aquesta anàlisi es basa en un cas pràctic inspirat en situacions que es donen en el dia a dia on una empresa busca optimitzar els seus costos operatius i millorar l'escalabilitat del seu servei a través del desplegament al núvol.

Global Track Trends es presenta com un exemple pràctic per desenvolupar i desplegar una aplicació web que permet als usuaris veure i escoltar les llistes Top 50 de cançons de Spotify dels països on està disponible. Aquesta aplicació també incorpora una base de dades MongoDB per gestionar l'històric de cançons escoltades pels usuaris.

El projecte posa l'accent en un escenari habitual dins d'equips informàtics: la migració d'un software ja funcional en local cap a una infraestructura al núvol per aprofitar millor els recursos disponibles, garantir la disponibilitat i poder adaptar-se a fluctuacions de demanda abaratint els costos, sense la necessitat de preocupar-se per la infraestructura necessària.

Al llarg d'aquest informe es tractaran els objectius principals del projecte, quins canvis hi ha hagut respecte als objectius inicials i la metodologia emprada. Es parlarà més en detall del procés de desenvolupament de l'aplicació, les tecnologies que s'han utilitzat, s'explicarà l'estudi dels diferents serveis seguint els passos que Google recomana i com es va realitzar la fase de desplegament.

2 Objectius

Amb aquest projecte l'objectiu és reflectir la feina d'un arquitecte al núvol a través de l'estudi i comparativa de les diferents solucions que ofereixen serveis com GCP per al desplegament d'una aplicació web, identificant quina s'ajusta millor a les necessitats de l'aplicació de cada client.

Tot i que inicialment es volia llançar l'aplicació en tres serveis diferents de GCP: Google Compute Engine (GCE), Migration Instance Groups (MIG) i Google Kubernetes Engine (GKE), per posteriorment fer una anàlisi de costos, consum de recursos i rendiment, a causa d'alguns contratemps durant la fase de desenvolupament, problemes amb el llançament en local en un entorn de contenidors i un augment imprevist de costos, vam decidir modificar l'estratègia i dur a terme, previ al desplegament, una anàlisi

dels diferents serveis. Sent aquest un apropament més semblant al treball real d'un arquitecte al núvol, qui generalment analitza i defineix la plataforma de desplegament més adequada abans de realitzar el llançament.

Els objectius específics del projecte un cop decidits els canvis són:
OB 1: Desenvolupar l'aplicació web Global Track Trends.
OB 2: Analitzar els diferents serveis de còmput Infrastructure as a Service (IaaS) que ofereix GCP per determinar la més adient segons l'estructura de l'aplicació i les "necessitats del client".
OB 3: Desplegar l'aplicació sobre el servei escollit, garantint una implementació eficient que s'ajusti a aquestes necessitats.

3 Desenvolupament del cas pràctic: Global Track Trends

Global Track Trends és una aplicació web que permet als usuaris escoltar les cançons de les playlist top 50 de tots els països en els quals aquesta funcionalitat està disponible a Spotify. L'objectiu principal de la plataforma és proporcionar als usuaris un accés fàcil i ràpid a la música més popular en temps real i poder fer una comparativa dels rànquings de les cançons més escoltades en tot el planeta.

3.1 Requisits funcionals

1. **Inici de sessió:** l'aplicació permet als usuaris autenticar-se amb el seu compte de Spotify.
2. **Visualització de Llistes:** els usuaris poden veure les llistes top 50.
3. **Històric de les cançons escoltades per l'usuari:** els usuaris poden veure les cançons que han escoltat amb anterioritat en un ordre de més a menys recent.
4. **Playlists:** els usuaris poden veure les seves pròpies playlists.
5. **Reproducció de cançons:** els usuaris amb compte prèmium de Spotify poden escoltar les cançons directament des de l'aplicació.

3.2 Arquitectura del sistema

Per al desenvolupament del front-end es va decidir utilitzar React [3] per la seva facilitat en la creació d'interfícies d'usuari dinàmiques i modulars. Alternatives com Angular i Vue van ser considerades, però es van descartar a favor de React pel seu gran ecosistema de biblioteques i comunitat, així com per la familiaritat que ja es tenia amb la llibreria.

React és una biblioteca JavaScript per a la construcció d'interfícies d'usuari interactives que proporciona una estructura modular que facilita el desenvolupament de components reutilitzables i eficients.

El back-end de l'aplicació es basa en l'API de Spotify [4], la qual es consulta per obtenir les dades de les llistes d'èxits de cada país i la informació de perfil d'usuari, fet que elimina la necessitat d'una infraestructura de back-end pròpia per guardar usuaris o llistes. Tot i això, s'ha decidit afegir una part de back-end a l'aplicació per gestionar l'històric de cançons escoltades. S'ha integrat MongoDB com a base de dades NoSQL que emmagatzema les cançons escoltades per l'usuari.

Estructura:
Com es pot veure a la figura 1, els components que formen l'estructura de l'aplicació són:

- **Front-end o interfície d'usuari (React):** desenvolupat en components que gestionen la interfície d'usuari i la lògica de l'aplicació.
- **API de Spotify**: proporciona les dades de les llistes i usuaris en temps real.
- **Base de dades:** per millorar l'experiència de l'usuari, una base de dades NoSQL com MongoDB és útil per emmagatzemar preferències d'usuari o històrics de consulta.
- **Servidor Node.js**: s'encarrega de la connexió entre el Front i la base de dades.

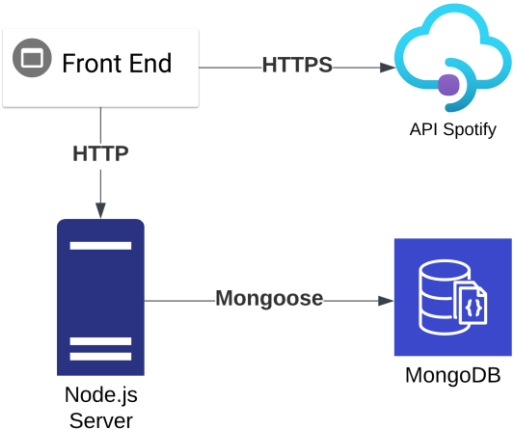

Fig. 1. Arquitectura de l'aplicació

3.3 Metodologia

El desenvolupament de l'aplicació i anàlisi de les opcions de desplegament a GCP s'han dut a terme seguint una metodologia DevOps [5]. Aquesta és una metodologia iterativa que permet intercalar les etapes de desenvolupament amb les d'operacions de desplegament. Aquesta metodologia assegura una col·laboració contínua entre el desenvolupament i les operacions, cosa que permet anar testejant l'aplicació a mesura que es realitzen diferents iteracions en el codi. Per abordar el projecte, es va estructurar en les següents fases:

Investigació i disseny: Durant aquesta fase es va investigar quines tecnologies podien ser utilitzades per al desenvolupament de l'aplicació i dissenyar l'arquitectura de l'aplicació.

Desenvolupament de l'aplicació: Durant aquesta fase es va dur a terme tota la codificació de l'aplicació mentre alhora es feien tests en local.

Anàlisi dels serveis: Fase on es van analitzar les diferents opcions de còmput i determinar quina s'adequava més a

l'aplicació.

Desplegament: En aquesta etapa del projecte, amb GKE ja escollit com a servei per al llançament de l'aplicació, es va realitzar tot el procés de desplegament.

3.4 Procés i proves del desenvolupament

Inicialment, totes les proves s'han dut a terme en un entorn local, utilitzant la comanda "*npm start*", proporcionada per l'entorn de desenvolupament de React. Aquesta comanda inicia un servidor local que permet visualitzar i interactuar amb l'aplicació a través d'un navegador, de manera que es poden fer proves i canvis en temps real.

Gràcies a la funcionalitat *hot reloading* de React, es poden veure els canvis de manera instantània sense necessitat de reiniciar manualment el servidor. Això agilitza el procés de desenvolupament, ja que permet fer iteracions ràpides i provar el flux complet de l'aplicació.

Un cop va estar verificat el correcte funcionament en local, es va desplegar l'aplicació a Vercel [6], una plataforma d'allotjament en el núvol.

3.5 Interfície i funcionalitats de l'aplicació

La interfície d'usuari de l'aplicació Global Track Trends s'ha dissenyat amb l'objectiu de ser intuïtiva i fàcil d'usar. Aquesta es divideix principalment en tres columnes que organitzen la informació del perfil d'usuari i finestres, llistat de cançons i reproductor.

Pantalla principal
Com es pot apreciar a la figura A1.1 situada a l'Annex 1, la pantalla principal es presenta inicialment amb dues columnes:

- **Columna esquerra:** Mostra la informació del perfil de l'usuari, incloent-hi dades com el nom i la foto de perfil (si estan disponibles) i les pàgines de l'aplicació a les quals pot navegar, com llistes personals o preferències d'usuari.
- **Columna central:** Conté el llistat de les llistes top 50 de Spotify de cada país.

Interacció amb una playlist
En prémer una playlist, aquesta s'expandeix mostrant les 10 primeres posicions de la llista, oferint una vista ràpida de les cançons més populars d'aquell país. A més, aquesta acció activa la tercera columna situada a la dreta, on s'indica la informació addicional de la playlist seleccionada (figura A1.2).

Reproductor
Observant la figura A1.3 es pot veure que quan l'usuari prem qualsevol cançó dins de la llista estesa, la columna de la dreta on es mostra la informació de la llista canvia per la informació de la cançó junt amb un reproductor i la lletra en cas d'haver-se trobat. En el cas que l'usuari no tingui un compte prèmium es mostrarà tota la informació d'igual manera, però no es reproduirà a causa de limitacions de l'API de Spotify.

Mini reproductor
La interfície inclou una funcionalitat addicional per millorar l'experiència d'usuari, com es pot apreciar a la figura A1.4 En cas d'haver seleccionat una altra playlist o haver tancat l'actual, la cançó pot continuar reproduint-se, aturar i tancar amb un petit reproductor situat a la columna dreta.

Registre històric
Finalment, com es mostra a la figura A1.5, per millorar l'experiència de l'usuari, l'aplicació ofereix dues finestres addicionals. Una similar a la ja mostrada on es veuen les llistes que l'usuari té guardades al seu compte de Spotify i una segona finestra que conté l'històric de cançons que l'usuari ha escoltat a la web.

3.6 Problemes sorgits durant el desenvolupament

Al llarg del desenvolupament del projecte ens vam trobar amb diferents problemes o imprevistos que vam haver de solucionar i que van causar canvis en la planificació. A continuació s'exposen quins van ser:

Canvis en l'API de Spotify: Durant els dies de proves de desplegament es van realitzar modificacions sobre l'API de Spotify que afectaven directament l'aplicació [7]. Els canvis afectaven principalment els endpoints d'algunes funcions d'àudio i obtenció de llistes destacades, cosa que va requerir aplicar alguns canvis al codi.

Desplegament en contenidors: Tot i funcionar de manera correcta en entorns locals llençant servidor i front de forma independent, la integració de MongoDB en un entorn de contenidors va presentar algunes dificultats amb la connexió entre serveis. Com un dels objectius inicials era llançar l'app en GKE, calia modificar el codi per a poder executar-lo en un entorn de contenidors. Amb aquesta finalitat es va voler dur a terme primer proves en local amb Minikube [8]. En aquest context com que Mongo ja no corria en local, sinó dins d'un contenidor Docker, aquest requeria la configuració de variables d'entorn on s'indiqués la IP del servidor Mongo. Per aquest motiu, requeria una IP per poder llençar-lo en un entorn de contenidors.

Augment de costos: Per poder tenir una IP disponible i poder connectar els serveis dins de contenidors, es va decidir llençar l'app directament a GKE i saltar el pas de l'entorn contenidoritzat en local. Durant les proves amb GKE, es va apreciar l'ús i consum de serveis com Cloud Logging i Virtual Private Connection necessaris per tenir connexió entre els serveis i un registre de logs.

Degut a tots aquests imprevistos, com s'ha comentat anteriorment als objectius d'aquest informe, tot i inicialment proposar el llançament en 3 serveis diferents, es va decidir realitzar l'estudi dels serveis i definir la plataforma de llançament de forma prèvia al mateix.

4 ANÀLISIS DELS SERVEIS DE CÒMPUT

En el cas pràctic que proposem, el client vol utilitzar Google Cloud Platform per llançar l'aplicació Global Track Trends amb la finalitat d'abaratir costos i evitar haver de fer-se càrrec del manteniment ús de la infraestructura.

Per determinar l'arquitectura d'infraestructura d'una aplicació cal conèixer com aquesta funciona, quines expectatives de tràfic es tenen, siguin d'una zona geogràfica o diverses, si esta codificada en micro-serveis, etc.

Global Track Trends està programada en tres parts (front, server, mongo), però no descarta afegir noves funcionalitats en un futur. L'objectiu del client és llençar-la de la forma més senzilla possible, sense que requereixi fer canvis en el codi. A més, el motiu principal pel qual el client va triar GCP, va ser el no haver d'estar al corrent de l'escalabilitat de la seva app, vol un servei que sigui manegat i escalat de forma automàtica, tenint en compte que esperen altes demandes d'usuaris.

4.1 Opcions de còmput

Inicialment, és essencial considerar les diferents opcions que GCP ofereix per llançar una aplicació al núvol, veure quines opcions aporta cadascuna d'elles i quina s'ajusta més a les necessitats del client. Dins de les plataformes de computació disponibles trobem:

Serveis serverless:
El concepte serverless fa referència al model de computació al núvol on les aplicacions s'executen sense que els desenvolupadors hagin de gestionar directament els seus servidors. Tot i que el terme "serverless" pot semblar que implica que no hi ha servidors, en realitat significa que els servidors són gestionats pel proveïdor. De manera que els desenvolupadors es poden centrar exclusivament en el codi de l'aplicació [9]. Dins d'aquesta secció existeixen diferents serveis:

1. Cloud run [10]: Cloud Run és un exemple de plataforma serverless on els recursos es proveeixen en funció de la demanda, que permet executar contenidors sense estat, activats mitjançant sol·licituds web o esdeveniments Pub/Sub. Aquest servei està dissenyat per a càrregues de treball intensives en càlcul, però no per càrregues destinades a estar sempre actives. Pot escalar de 0 (sense trànsit) permetent no consumir recursos quan no hi ha demanda fins al volum necessari. A més permet definir la CPU i memòria de les tasques i serveis.

2. Cloud Functions [11]: és un servei que actualment forma part de Cloud Run i permet executar codi en resposta d'esdeveniments sense necessitat d'administrar els servidors. Permet desenvolupar funcions aïllades que encapsulen la lògica en petites peces de codi. És una bona opció per casos on s'executen funcions de curta durada, escalables i que s'activin per esdeveniments concrets com ara sol·licituds HTTP, missatges de Pub/Sub, canvis en fitxers de Cloud Storage, entre d'altres.

3. App Engine [12]: és un servei de Platform as a Service (PaaS) totalment gestionat per Google que permet desenvolupar i allotjar aplicacions web a gran escala sense preocupar-te per la infraestructura subjacent. És a dir, Google s'encarrega de tot el que està relacionat amb els servidors, el balanceig de càrrega i l'escalabilitat, cosa que permet que els desenvolupadors puguin centrar-se en el codi a desenvolupar. Permet triar entre diversos llenguatges i frameworks populars per desenvolupar l'aplicació depenent del tipus d'entorn per adaptar-se el màxim possible a les necessitats dels seus clients:

- L'**entorn estàndard** és ideal per aplicacions que requereixen una configuració ràpida i escalat automàtic. Utilitza contenidors preconfigurats i ofereix una experiència de desenvolupament senzilla. Tot i això, la seva flexibilitat és limitada, ja que com he comentat són entorns predefinits.
- Per una altra banda, l'**entorn flexible** ofereix més control sobre l'entorn d'execució, permetent als desenvolupadors personalitzar o crear les imatges Docker.

Màquines virtuals
És una de les solucions més clàssiques dins del món de la informàtica que permet executar càrregues de treball especialitzades i de propòsit general.

1. **Google Compute Engine (GCE) [13]:** proporciona màquines virtuals (VM) individuals que pots configurar a mida. És ideal per a càrregues de treball altament personalitzades i quan es vol tenir un control granular sobre el maquinari i software.
2. **Migration Instance Group (MIG) [14]:** són grups d'instàncies homogènies que pots definir amb un template permetent crear flotes senceres que escalen de forma automàtica segons la teva configuració.

Google Kubernetes Engine (GKE)
GKE és un servei gestionat que permet desplegar, gestionar i escalar aplicacions contenidoritzades de forma pràctica i eficient [15]. És una plataforma que es podria dir que s'encarrega, en gran mesura, de la complexitat associada a l'orquestració de contenidors, permetent centrar-te en el desenvolupament de l'aplicació.

Dins de GKE, hi ha dos tipus principals de clústers que es poden configurar segons les necessitats de cada projecte: GKE Standard i GKE autopilot.

1. GKE Standard permet un control total sobre la infraestructura del clúster. L'usuari és responsable de gestionar els nodes del clúster incloent-hi la configuració dels nodes, actualitzacions i seguretat, escalat, etc.
2. Els clústers autopilot són una opció més recent que ofereix una experiència "serverless", on gran

part de la gestió del clúster és automatitzada per Google. Aquesta automatització incorpora l'aprovisionament i manteniment dels nodes, configuració de la infraestructura de xarxa i seguretat i escalat automàtic.

5 Quina opció escollir?

Com es pot apreciar a la figura 2, els serveis es poden classificar segons si són més o menys configurables i/o el control que tens sobre el mateix.

Fig. 2. Distribució de serveis segons el manteniment i l'opció a configurar-lo

GCP ofereix una gamma de serveis de còmput que van des de donar als usuaris un control total (GCE) fins a elements més abstractes (Cloud Functions), deixant que Google s'ocupi cada cop més de la gestió i les operacions al llarg del camí. Google proporciona un arbre de decisions per facilitar als clients de GCP el fet de decidir quin servei de còmput escollir [16].

5.1 Escollint la millor opció de còmput seguint l'arbre de decisions

Per a determinar el servei de còmput més adequat per a desplegar l'aplicació Global Track Trends, s'ha seguit l'arbre de decisions de GCP. Basant-nos en l'anàlisi feta, GKE es presenta com la solució òptima. A continuació, es detallen les raons d'aquesta elecció seguint l'arbre de decisions esmentat. Cal comentar que aquest arbre de decisions no és una guia definitiva per a trobar l'opció de còmput ideal, sinó una sèrie de preguntes que Google recomana fer-se quan un equip treballa en la creació o migració d'una aplicació a GCP. En alguns casos, més d'una opció podria ser vàlida, i és quan cal tenir més en compte els detalls de l'arquitectura del codi i les necessitats del client. A continuació es mostren les diferents preguntes i respostes, així com l'arbre de decisions a la figura 3:

1 Ets un desenvolupador d'aplicacions mòbils/HTML5?
Google: Si esteu creant un client gruixut que només es basa en un backend per a la sincronització i/o l'emmagatzematge, Firebase és una opció fantàstica. Firebase us permet emmagatzemar documents NoSQL complexos i fitxers mitjançant una API i un client molt fàcils d'utilitzar disponibles per a iOS, Android i Javascript.

En cas d'estar planejant començar a desenvolupar una aplicació web amb GCP, Firebase podria ser una opció viable. No obstant això, com que és una aplicació ja existent hi ha certes limitacions que no encaixen del tot amb aquest servei.

Firebase és molt útil quan s'utilitzen Cloud Functions com a backend, però en el nostre cas és un backend ja existent en Node.js. A més, Firebase no està dissenyat per treballar amb MongoDB sinó Firestore, la seva pròpia base de dades NoSQL. Això implica que caldria fer una migració de la BD. Com en el nostre cas tenim una aplicació ja existent i les necessitats del client són migrar l'aplicació realitzant els mínims canvis i aconseguint uns costos reduïts, però mantenint sempre la disponibilitat, Firebase no sembla la millor opció tenint en compte que caldria modificar el codi del servidor i migrar la base de dades.

2 Estàs desenvolupant una aplicació dirigida per esdeveniments?
Google: Busqueu solucions "sense servidor" o "Funcions com a servei"? Cloud Functions permet escriure funcions en Javascript que s'executen a Node.js. Amb Cloud Functions, podeu crear funcions individuals complexes que s'exposen com a microserveis per aprofitar tots els nostres serveis sense haver de mantenir sistemes i enganxar-los tots junts.

Encara que l'aplicació interactua amb l'API de Spotify, la seva naturalesa no és ser dirigida per esdeveniments, com en el cas d'aplicacions basades en IoT o processament de dades en temps real.

Llançar l'aplicació completament en Cloud Functions és inviable per la seva pròpia naturalesa. Es tracta d'un servei de contenidors efímers sense persistència de dades, impedint l'ús directe d'una BD i caldria utilitzar altres serveis per llençar l'app completament. Cal afegir també, que aquesta classe de serveis com Cloud Functions o Cloud Run, no estan pensats per aplicacions d'un ús intensiu o amb una alta demanda d'usuaris. Són serveis que generen instàncies o contenidors efímers que s'eliminen quan la demanda és zero. En cas de tenir un tràfic sostingut com és el nostre cas pràctic, el cost podria augmentar significativament respecte a altres serveis.

3 La teva aplicació utilitza o necessites algun dels següents requisits?
- **Especificació d'un sistema operatiu o nucli:** No.
- **Ús de GPU:** No.

Google: Si heu respost "no", val la pena fer-hi una ullada a App Engine. App Engine és una solució sense servidor que executa el vostre codi a la nostra infraestructura i només us cobra pel que feu servir.

L'aplicació no requereix capacitats de processament especialitzades, com les que ofereixen les GPUs, per la qual cosa no aplica l'ús de Google Compute Engine (GCE) amb configuracions específiques de maquinari. Com el mateix Google comenta, en aquest cas App Engine és una opció que podria ser viable.

Efectivament pel que fa al front i al servidor App Engine podria ser útil, ja que aquest servei executa contenidors tant amb imatges estàndard com personalitzades depenent de l'entorn, però no gestiona bases de dades directament. Una opció seria l'ús de l'entorn flexible, pel fet que utilitzem imatges personalitzades, tenint en compte que caldria

realitzar algunes modificacions al codi del servidor per mantenir l'ús de la base de dades.

4 La teva aplicació utilitza o necessites algun dels següents requisits?
- **Desplegament híbrid o multicloud:** No
- **Llicències de programari específiques:** No
- **Ús de protocols diferents d'HTTP/S:** No
- **Aplicació ja existent:** Si

Encara que Google Compute Engine podria considerar-se per a gestionar manualment la infraestructura, aquesta opció augmenta la complexitat operativa i no és ideal per a les necessitats actuals. A més és un servei que no ofereix opcions d'escalada automatitzades i caldria dur a terme una configuració molt extensa de diferents màquines virtuals per a poder donar suport a una alta demana o utilitzar màquines d'un cost molt més elevat.

Una altra opció amb relació a les màquines virtuals és l'ús d'un Migration Instance Group. Aquest es tracta d'un servei que facilita la creació d'una o més plantilles de màquines virtuals les quals es poden configurar per executar més o menys màquines homogènies en funció de la demanda. Això solucionaria el problema d'escalabilitat però segueix requerint d'una complexitat operativa.

5 La teva aplicació estarà en contenidors?
Google: Si esteu creant una solució de diversos contenidors, l'orquestració es converteix en una consideració. L'orquestració de contenidors és un servei que gestiona el desplegament, la redundància i la distribució de càrrega dels vostres contenidors. Si esteu pensant en utilitzar Kubernetes a GCP, només hauríeu d'utilitzar Container Engine. Escala automàticament els membres del clúster de Kubernetes, cosa que us permet crear solucions de Kubernetes que creixen i es redueixen en funció de la demanda.

Sí, l'aplicació està dividida en tres serveis principals (front-end, back-end i base de dades MongoDB), els quals es poden contenidoritzar. Això fa que l'orquestració mitjançant Kubernetes sigui una opció natural, i a diferència d'una màquina virtual en cas de tractar-se d'un clúster manegat per Google no cal tenir en compte les especificacions dels nodes que s'utilitzaran.

6 Es planeja utilitzar Kubernetes com a orquestrador?
Si és necessari sí. Kubernetes és l'estàndard de la indústria per a l'orquestració de contenidors, i el seu ús permet aprofitar funcionalitats avançades com l'escalabilitat automàtica, balanceig de càrrega, desplegaments continus i alta disponibilitat.

Quina opció escollir?
En aquest punt, un cop analitzats els diferents serveis de l'arbre de decisions hi ha tres serveis que podrien ajustar-se a les nostres necessitats: App Engine, GCE amb MIG i GKE. Per escollir la millor de les tres, caldria fer un estudi de costos i decidir quina opció és més adequada per al que busquem.

5.3 Estimació dels costos

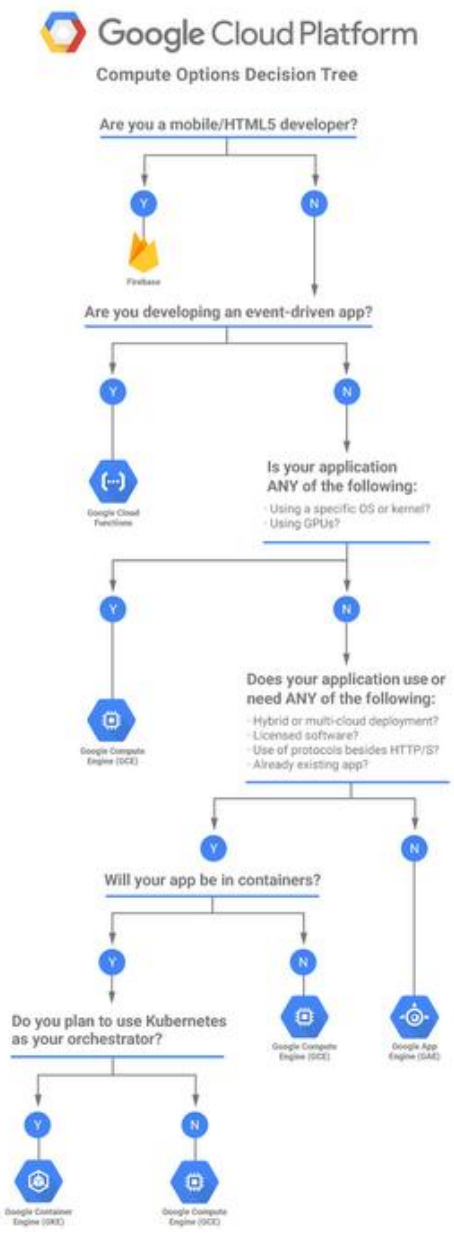

Fig. 3. Arbre de decisions de Google

GCP també ofereix una calculadora de costos [17] perquè els clients puguin fer una estimació abans de llançar una aplicació (visualitzar figures A2.1 – A2.3). Malgrat no poder calcular-ho tot de manera exacta com seria ideal, ajuda a fer-se una idea de quin pot ser el cost mensual del servei que volem provar.

Normalment en un cas com el nostre on una aplicació ja funciona en local, però es vol migrar al núvol, hi ha dades suficients per a fer una aproximació molt més fidel a la realitat, però com es tracta d'un cas hipotètic improvisarem les dades intentant que siguin tant reals com sigui possible. A continuació es mostren els càlculs i costos hipotètics de cadascun dels serveis esmentats:

Costos App Engine

Per calcular el cost mensual d'App Engine en l'entorn flexible, entorn que permet l'ús d'imatges personalitzades als contenidors d'App Engine, la calculadora de Google demana: vCPUs/hora, memòria per hora, mida del disc i dades de sortida.

Per obtenir el càlcul del tràfic hem suposat que l'aplicació té un flux de 100.000 usuaris al dia, una mitjana de 10 peticions per sessió i que cada petició pesa uns 50 KB. A continuació es mostren els càlculs:

Usuari per sessió: 50 KB/petició x 10 peticions = 500 KB per usuari
Diari total: 500 KB/usuari x 100.000 usuaris = 50.000.000 KB/dia
Mensual total: 50 GB/dia x 30 dies = 1500 GB/mes

Suposant un consum de:
- 0.5 vCPUs per hora
- 1 GB de memòria per hora
- 10 GB de disc
- 1500 GB de tràfic

Obtenim un cost de **125,77** euros cada mes.

Costos GCE
Per calcular el cost mensual d'una màquina virtual de GCE la calculadora demana que introduïm la següent informació:
- **Nombre d'instàncies i ús total:** 1 instància encesa les 730 hores del mes
- **Sistema operatiu:** gratuït
- **Model d'aprovisionament:** Fa referència a si són màquines Spot (que poden ser aturades per motius externs) o Regular. Seleccionarem Regular.
- **Grup de màquines:** Propòsit general
- **Sèrie:** N1
- **Màquina configurada:** 1 vCPU + 1 GiB de RAM
- **Mida del disc d'arrancada:** 10 GiB
- **SSD:** El mínim és 375 GB

Amb aquesta configuració obtenim un cost mensual de **58.48** euros cada més.

Costos GKE
En el cas de GKE cal configurar la calculadora d'una forma semblant a la de GCE indicant el tipus de màquina i les seves característiques, ja que fa el càlcul sobre els clústers que el client manega i no permet calcular quin és el cost d'un clúster manegat per Google, però ens ajuda a fer-nos una idea.

El cost final que dona configurant els nodes igual que a l'apartat del càlcul del cost de GCE és de **101.48** euros per mes.

5.4 Justificació de l'elecció de GKE en lloc de GCE

Després de seguir l'arbre de decisions per a la selecció de la infrestructura de còmput més adequada, es va concloure que les tres opcions que millor s'adaptaven a les necessitats del projecte Global Track Trends eren GCE amb MIG, GKE i App Engine. No obstant, l'elecció final degut als requisits del projecte i les seves característiques va ser GKE com a l'opció més adequada per diferents raons. Com hem vist a l'apartat anterior dels costos, App Engine no només requereix una migració de base de dades i alguns canvis en el codi del servidor, sinó que és un servei que surt més car a mesura que va augmentant la demanda, ja que no només es paguen els recursos sinó també les dades de sortida. Per aquests motius es va descartar com la infraestructura adequada.

Un cop descartat el servei d'App Engine, ens queda determinar si utilitzar GCE amb MIG o GKE.

Taula 1. Comparativa entre GKE i GCE

Tema	GKE	GCE amb MIG
Escalabilitat	Escalabilitat automàtica de pods i nodes	Escalabilitat automàtica d'instàncies
Gestió de la infraestructura	Totalment gestionat per Google en cas d'Autopilot	Requereix configuracions manuals a la VM
Disponibilitat	Alta disponibilitat integrada	Dependència de la configuració manual
Costos operatius	L'automatització té un cost de $0.10 l'hora per cada clúster	Cal tenir a alguna persona designada que s'encarregui de la configuració manual

Avantatges específics de GKE per a Global Track Trends
Com hem vist en les respostes anteriors, GKE destaca com la millor opció per a desplegar l'aplicació Global Track Trends. A més, basant-nos en els requisits inicials del client, aquest servei s'ajusta a la perfecció pels motius següents:

- **Escalabilitat automàtica:** GKE permet escalar automàticament tant els pods com els nodes en funció de la càrrega de treball. Un pod és la unitat bàsica d'execució de Kubernetes, es podria considerar a un equivalent a un contenidor tot i que això no és del tot així, ja que cada pod executa un o més contenidors a dins seu. Que GKE tingui aquest nivell d'escalabilitat és crucial per a una aplicació com a Global Track Trends, que podria experimentar pics de demanda significatius en determinats moments.
- **Compatibilitat amb arquitectures modernes de microserveis:** la divisió de l'aplicació en diversos serveis (front-end, back-end i base de dades) es beneficia de la capacitat de GKE per a orquestrar i gestionar microserveis. Això permet una major modularitat i facilitat de manteniment.
- **Optimització de costos:** tot i que altres opcions com Cloud Run poden semblar més econòmiques inicialment, GKE permet una configuració més eficient

dels recursos i optimització de costos a llarg termini mitjançant l'ús d'escalat automàtic i configuracions adequades, a més d'estar pensat per a aplicacions d'alta demanda.

- **Alta disponibilitat i recuperació davant desastres:** GKE ofereix distribució multi-node i multi-zona, garantint que l'aplicació sigui resistent davant fallades de maquinari o interrupcions en una regió. A part d'això, també compta amb funcions com l'Horizontal Pod Autoescaler que s'encarrega d'aprovisionar o eliminar pods segons la demanda o en cas d'haver caigut el servei.

Conclusió de les opcions de còmput

Com s'ha anat mostrant fins aquest punt, la infraestructura de GKE Autopilot proporciona una sèrie de beneficis clau que la converteixen en la millor opció per a Global Track Trends.

Encara que existeixen altres opcions vàlides, aquestes impliquen una major complexitat operativa en forma de reestructuració de codi, canvi de tecnologies o costos operatius de manteniment i configuració.

Per aquests motius i per totes les facilitats que GKE aporta a l'hora de llançar noves versions de l'aplicació, la facilitat d'afegir altres serveis a la web de forma independent i fàcil i el fet de poder oblidar-te per complet de la part d'escalabilitat i configuració dels nodes es va escollir GKE com a infraestructura per a l'aplicació.

6 Desplegament de l'aplicació de GKE

Inicialment, per tal d'optimitzar el desplegament de l'aplicació i reduir costos en les fases inicials de prova, com ja s'ha comentat anteriorment en aquest informe, es va decidir començar llençant l'aplicació en un clúster local utilitzant Minikube. Aquesta aproximació va permetre identificar i solucionar problemes potencials abans d'invertir en recursos del núvol de GCP.

6.3 Creació d'imatges Docker

El primer pas per desplegar l'aplicació en un entorn de contenidors va ser crear les imatges Docker corresponents a cadascun dels components:

1. **Front-end:** l'aplicació desenvolupada en React.
2. **Back-end:** el servidor de Node.js.
3. **Base de dades:** ús de la imatge oficial de MongoDB disponible al seu repositori públic [18].

Durant les proves locals amb Minikube, com s'ha fet notar al punt 2.5, es va detectar un problema de connexió entre els contenidors. Després de diverses proves, la conclusió va ser que MongoDB requeria una configuració específica en l'URI de connexió quan es tracta de contenidors, ja que cal indicar la IP del contenidor a les variables d'entorn. Mitjançant algunes proves més es va corroborar que tant el back-end com la base de dades funcionaven correctament de forma independent, verificant així que es tractava d'un error de connexió entre serveis.

Aquest fet va portar a la decisió de passar directament amb el desplegament a GKE per aprofitar les eines natives de Kubernetes i poder configurar una IP en compres de localhost com fins al moment.

6.4 Emmagatzemar les imatges Docker: Artifact Registry

Artifact Registry és un servei de Google Cloud que actua com un dipòsit centralitzat per emmagatzemar, gestionar i desplegar artefactes utilitzats en el desenvolupament de programari [19]. Això inclou imatges de contenidors (com les fetes servir per Kubernetes o Docker), biblioteques de paquets, binaris i altres components essencials del programari.

Un cop creades les imatges del front i del back amb la comanda *Docker build*, es van pujar a Artifact Registry. Això permet tenir les imatges accessibles de manera centralitzada i segura, simplificant i agilitzant el procés de desplegament. En indicar aquestes imatges directament des d'Artifact Registry als manifestos de Kubernetes, es pot realitzar els deployments d'una forma ràpida, sense haver de preocupar-se per gestionar manualment les imatges.

6.5 Creació del clúster a GKE

Un cop les imatges havien estat creades i guardades a Artifact Registry, es va procedir a la creació del clúster de kubernetes on s'executarà l'aplicació. Tenint en compte que l'objectiu del client era minimitzar la gestió de la infraestructura, es va optar per la creació d'un clúster autopilot.

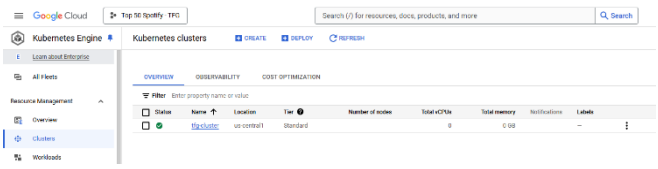

Fig. 4. Creació del clúster autopilot

6.6 Configuració dels deployments/workloads i resolució d'errors

Un Deployment a Kubernetes defineix la configuració dels pods. Els deployments permeten gestionar l'escalabilitat, actualitzacions i recuperació davant d'errors, així com configurar els seus paràmetres o variables d'entorn.

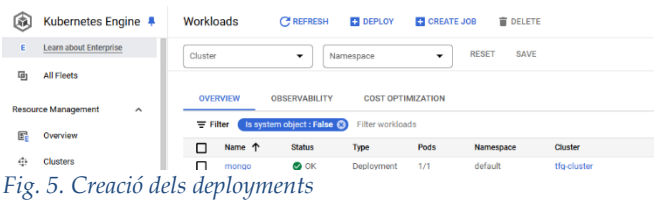

Fig. 5. Creació dels deployments

Cada Deployment es defineix amb un fitxer YAML que especifica els detalls necessaris com la imatge Docker a utilitzar, el nombre inicial de rèpliques o còpies, variables

d'entorn, ports d'escolta, etc.

Desplegament del fron-end
Inicialment es va crear un Deployment bàsic per al front-end, definit per un fitxer YAML amb la configuració mínima necessària per posar-lo en funcionament. Aquest fitxer incloïa:

- La imatge Docker del front de l'Artifact Registry.
- L'exposició del port.
- Una rèplica inicial, ja que en aquesta fase no es requeria escalabilitat.

En intentar iniciar el Deployment del front, els pods no es van poder inicialitzar correctament registrant un error en la imatge Docker i el seu format.

Durant la investigació de l'error, es va veure que l'origen de l'error era el fet d'haver creat la imatge Docker amb un ordinador Mac forçant una arquitectura de la imatge ARM64, mentre que els nodes del clúster GKE utilitzen arquitectura x86-64 (AMD64). Per solucionar-ho es va utilitzar la comanda *Docker Buildx* [20] per construir les imatges en comptes de *Docker Build*. Aquesta és una eina que permet crear imatges Docker compatibles amb diverses arquitectures.

Exposició del front-end: Load Balancer
Un cop solucionat el problema de les imatges, el Deployment del front es va iniciar correctament. Tot i això, encara no era accessible des d'internet. Per fer-lo accessible, es va crear un servei del tipus Load Balancer [21] proporcionant el següent:

1. Una IP pública que permet als usuaris externs accedir a l'aplicació.
2. Una connexió entre el port d'escolta del Deployment i el port extern del LoadBalancer.

El servei es va configurar amb el port 80, connectant amb el port 3000 del pod del front, ja que React utilitza aquest port per defecte.

Desplegament del back-end
El Deployment del back-end es va realitzar seguint un procés similar al del front:

1. Crear la imatge Docker multiarquitectura del servidor Node.js i pujar-la a Artifact Registry.
2. Definir el fitxer YAML del Deployment especificant el port d'escolta del servidor.
3. Crear un servei del tipus LoadBalancer per exposar el servidor a internet. Això permet que el front pugui enviar peticions al back des del navegador dels usuaris.

Desplegament de Mongo
Per a la base de dades MongoDB es va utilitzar la imatge oficial de Mongo disponible al repositori públic de Docker Hub. En aquest cas no era necessari exposar MongoDB a internet, ja que únicament el servidor es comunica amb la BD. En lloc d'això, es va configurar un servei del tipus ClusterIP, que assigna una IP interna dins del clúster millorant la seguretat i simplificant la configuració.

Connexió entre els components:
Amb els tres components (front-end, back-end i MongoDB) desplegats i els seus serveis configurats, es va verificar la connexió entre ells:

- El front-end és accessible des d'internet i envia peticions HTTP al back-end mitjançant la IP pública del LoadBalancer associat.
- El back-end accedeix a MongoDB utilitzant la IP interna proporcionada pel servei ClusterIP.

```
PS C:\Users\danie\Documents> kubectl get services
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)          AGE
back-service    LoadBalancer   34.118.231.220   34.44.159.6      8080:31757/TCP   3d17h
front-service   LoadBalancer   34.118.225.165   34.136.173.235   80:30752/TCP     3d17h
kubernetes      ClusterIP      34.118.224.1     <none>           443/TCP          5d17h
mongo-service   ClusterIP      34.118.231.249   <none>           27017/TCP        3d16h
PS C:\Users\danie\Documents> kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
mongo-6b967f586f-z4fqk           1/1     Running   0          12m
spotify-back-797bbcc785-tgmht    1/1     Running   0          12m
spotify-front-6fbc85d6bd-r4wzs   1/1     Running   0          12m
PS C:\Users\danie\Documents>
```

Fig. 6. Verificant les connexions del serveis al clúster

7 COSTOS FINALS

Com a referència real per als costos de l'aplicació en funcionament un cop desplegada a GKE, el desglossament dels serveis i els seus costos en euros són:

- **Networking (VPC):** 18.02
- **Cloud Monitoring:** 1.93
- **Artifact Registry:** 1.72
- **Kubernetes Engine:** 57.77

Com podem veure, tot i tractar-se d'un clúster i ser una infraestructura típicament gran, gràcies al servei autopilot i la gestió de Google per augmentar o reduir els recursos d'aquest, després d'un mes amb l'aplicació en execució de forma constant ha suposat un cost de 57.77 euros, el qual considero que és un molt bon preu per totes les facilitats que aporta.

8 RESULTATS ASSOLITS

En aquest apartat de l'informe revisarem els objectius plantejats en un inici per veure el seu assoliment.

OB1 Desenvolupar l'aplicació web funcional "Global Track Trends" - Assolit

L'aplicació ha estat completament desenvolupada incloent les següents funcionalitats: autenticació d'usuaris a través de l'API de Spotify, visualització de les llistes Top 50, accés a les playlists personals i emmagatzematge de l'historial de reproduccions. Recentment, s'ha aconseguit arreglar un problema en la reproducció de les cançons degut a les restriccions de Spotify amb els canvis més recents de l'API, ja que restringeixen fortament l'ús d'aquesta per als desenvolupadors més amateur, necessitant tenir una finestra d'Spotify oberta per a poder detectar un dispositiu on reproduir una cançó.

OB2 Fer una anàlisi comparativa de les opcions de desplegament - Assolit

Inicialment es va planificar una comparativa entre GKE, GCE i MIGs amb desplegaments simultanis, però aquesta opció es va descartar degut als imprevistos esmentats a l'apartat 2.5 i degut a l'augment de cost que hagués suposat. En lloc d'això, es va seguir un procés d'anàlisi previ al desplegament, utilitzant l'arbre de decisions de GCP i els criteris d'arquitectura recomanats per Google. Aquest enfocament va permetre determinar que GKE en mode Autopilot era la millor opció per satisfer els requisits d'escalabilitat, manteniment i cost operatiu.

Basant-se en l'anàlisi elaborada, es va concloure que Google Kubernetes Engine (GKE) era l'opció més adequada per al desplegament de l'aplicació. Les seves característiques clau són:

Escalabilitat automàtica: ajusta automàticament els recursos segons la demanda.
Compatibilitat amb microserveis: facilita la gestió de l'aplicació modular en diversos components.
Gestió operativa mínima: Gràcies al mode Autopilot que redueix la gestió manual.
Alta disponibilitat i recuperació davant desastres: gràcies a la distribució multinode i multizona i l'opció de rèpliques en els deployments dels diferents serveis.

OB3 Desplegar l'aplicació a Google Cloud Platform (GCP) - Assolit

Després d'analitzar les diferents plataformes i determinar que Google Kubernetes Engine (GKE) era la millor opció de desplegament per a Global Track Trends, es va finalitzar el llançament en GKE de manera exitosa.

9 Conclusions

El desenvolupament i desplegament de l'aplicació Global Track Trends ens ha permès obtenir una visió més propera de les diferents opcions de desplegament disponibles a Google Cloud Platform (GCP), amb un enfocament particular en Google Kubernetes Engine (GKE). Aquest projecte ha posat de manifest la importància de conèixer el projecte que es té entre mans alhora de prendre decisions en la selecció d'infraestructures al núvol, tenint en compte aspectes com l'escalabilitat, la gestió operativa, els costos associats, etc. S'ha pogut mostrar la rellevància de determinar la infraestructura adequada per poder evitar costos excessius causats per una infraestructura mal escollida.

En conclusió, aquest projecte ha assolit els objectius plantejats inicialment, mostrant un cas habitual del procés de migració d'una aplicació al núvol i exemplificant la importància de la feina dels arquitectes al núvol.

Bibliografia

[1] IT Reseller. (2024, Agost). Las aplicaciones SaaS i habilitadas en la nube continuan con su crecimiento. [En línia] Disponible a: https://www.itreseller.es/cloud/2024/08/las-aplicaciones-saas-y-habilitadas-en-la-nube-continuan-su-crecimiento

[2] PaaS vs. IaaS vs. SaaS vs. CaaS: How are they different? [En línia]. Disponible a: https://cloud.google.com/learn/paas-vs-iaas-vs-saas?hl=en

[3] React Documentation. [En línia]. Disponible a: https://react.dev/learn

[4] Web API | Spotify for Developers. [En línia]. Disponible a: https://developer.spotify.com/documentation/web-api

[5] What is DevOps? [En línia]. Disponible a: https://aws.amazon.com/devops/what-is-devops/

[6] What does Vercel do? [En línia]. Disponible a: https://vercel.com/blog/what-is-vercel

[7] Introducing some changes to our Web API. [En línia]. Disponible a: https://developer.spotify.com/blog/2024-11-27-changes-to-the-web-api

[8] Using Minikube to Create a Cluster. [En línia]. Disponible a: https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/

[9] Sin servidores. [En línia]. Disponible a: https://cloud.google.com/serverless?hl=es-419

[10] What is Cloud Run. [En línia]. Disponible a: https://cloud.google.com/run/docs/overview/what-is-cloud-run

[11] Cloud Run functions overview. [En línia]. Disponible a: https://cloud.google.com/functions/docs/concepts/overview

[12] An overview of App Engine. [En línia]. Disponible a: https://cloud.google.com/appengine/docs/an-overview-of-app-engine

[13] Compute Engine overview. [En línia]. Disponible a: https://cloud.google.com/compute/docs/overview

[14] Instance groups. [En línia]. Disponible a: https://cloud.google.com/compute/docs/instance-groups

[15] GKE overview. [En línia]. Disponible a: https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview

[16] GCP decision tree. [En línia]. Disponible a: https://cloud.google.com/blog/products/compute/choosing-the-right-compute-option-in-gcp-a-decision-tree

[17] Welcome to Google Cloud's pricing calculator. [En línia]. Disponible a: https://cloud.google.com/products/calculator?hl=es-419

[18] Mongo image. [En línia]. Disponible a: https://hub.docker.com/_/mongo

[19] Artifact Registry. [En línia]. Disponible a: https://cloud.google.com/artifact-registry/docs/overview

[20] Docker Buildx. [En línia]. Disponible a: https://docs.docker.com/reference/cli/docker/buildx/

[21] Exposing apps in GKE. [En línia]. Disponible a: https://cloud.google.com/kubernetes-engine/docs/how-to/exposing-apps

APÈNDIX
A1. Imatges de l'aplicació Global Track Trends

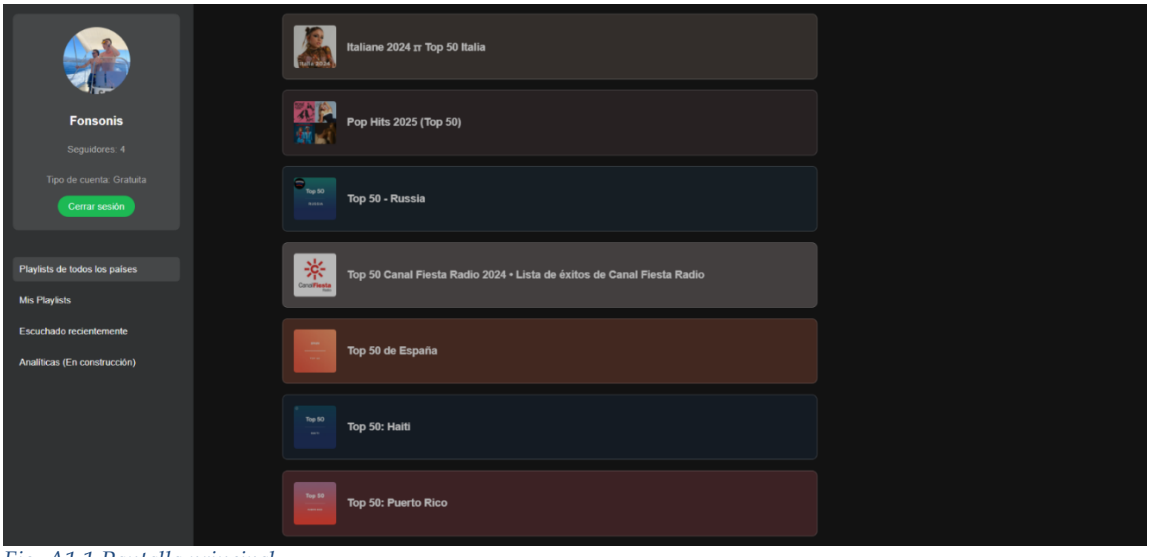

Fig. A1.1 Pantalla principal

Fig. A1.2 Pantalla principal amb playlist estesa

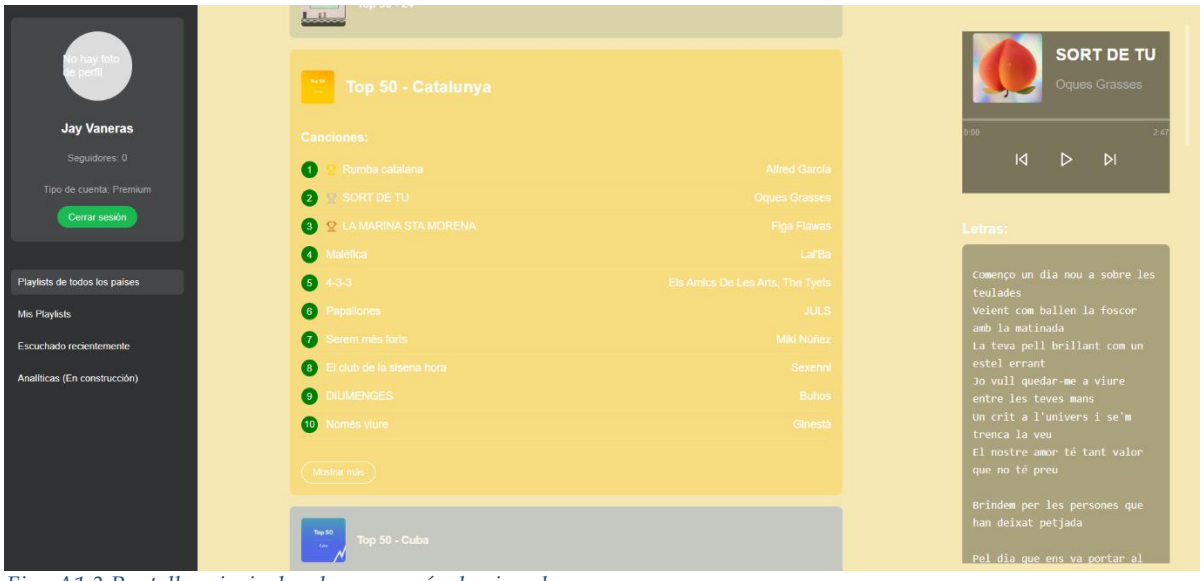

Fig. A1.3 Pantalla principal amb una cançó seleccionada

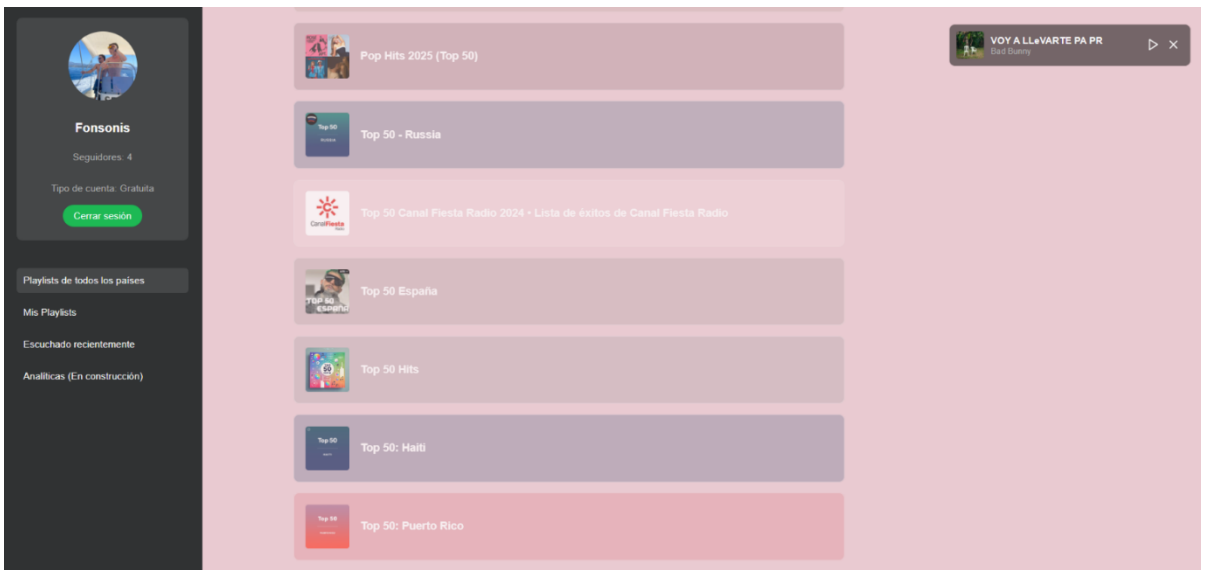

Fig. A1.4 Pantalla principal amb una cançó reproduint-se al mini reproductor

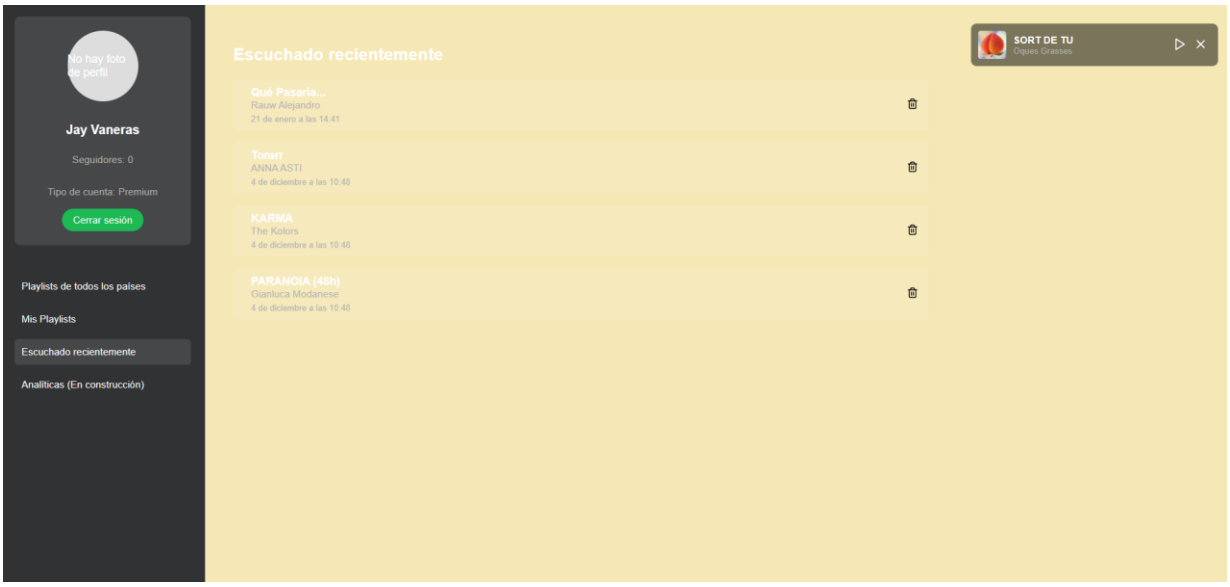

Fig. A1.5 Pantalla del registre de cançons escoltades

A2. Imatges de la Google Price Calculator

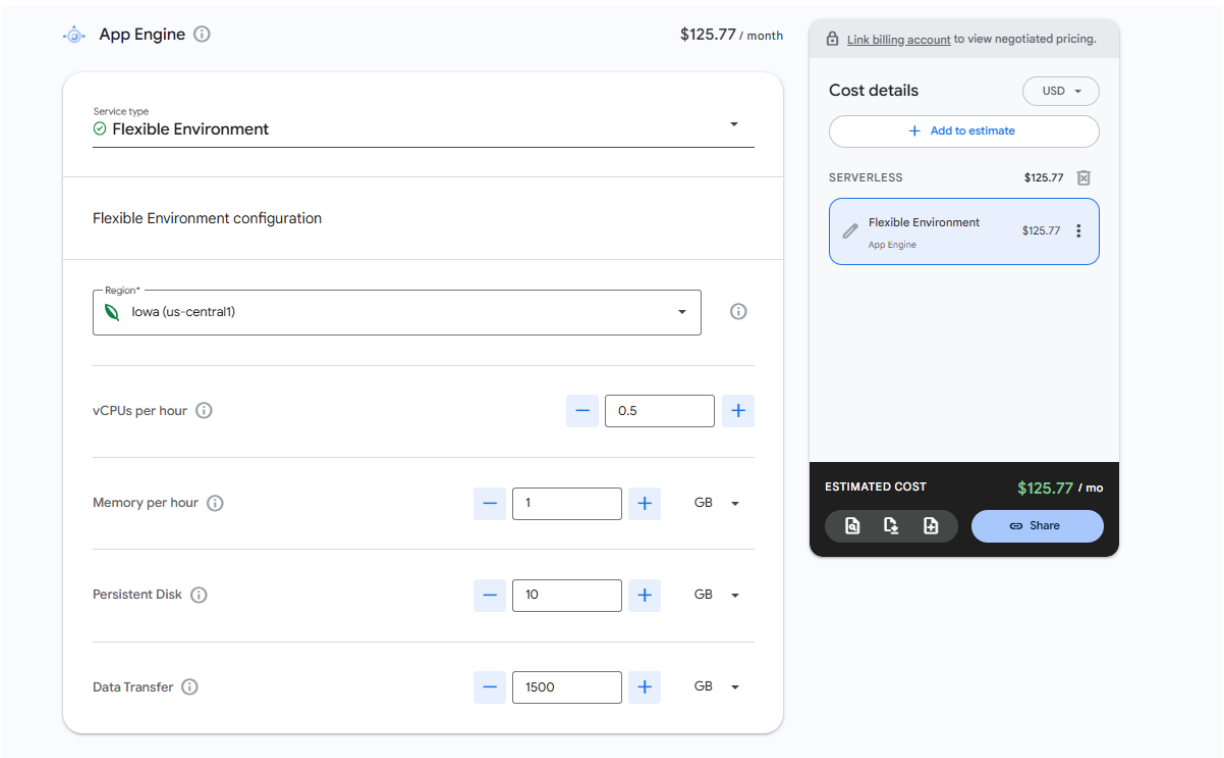

Fig. 9. Google Price Calculator App Engine

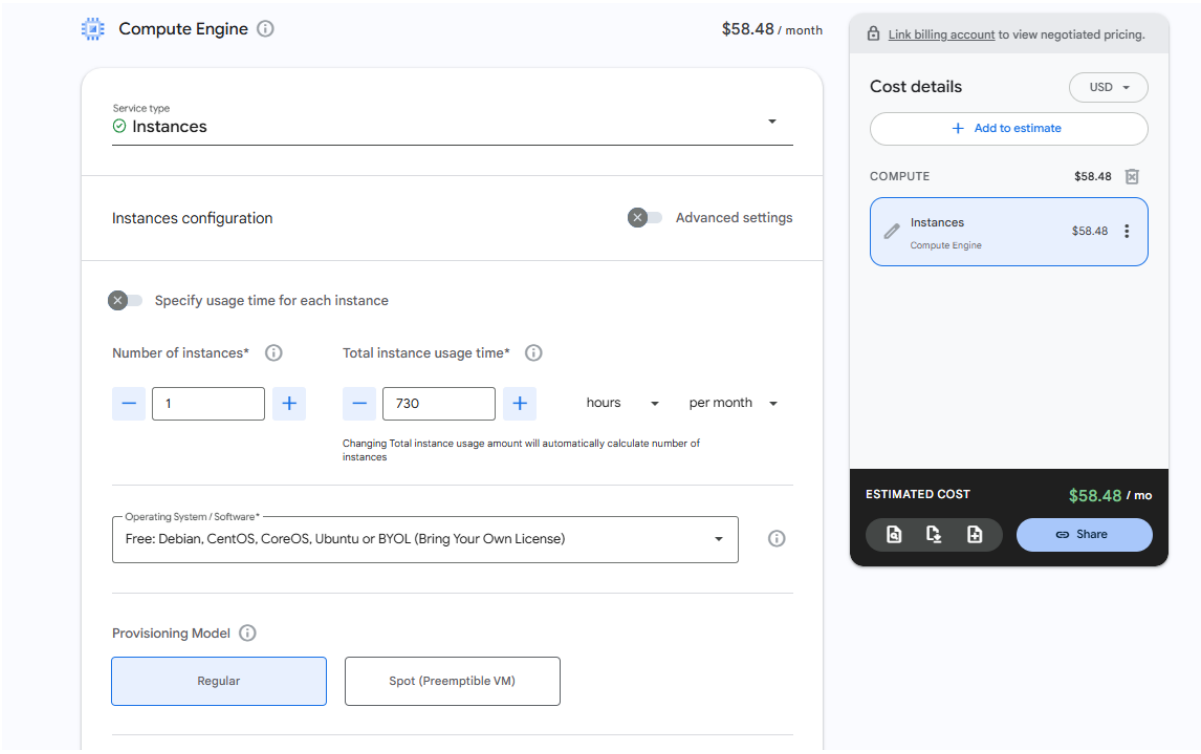

Fig. 10. Google Price Calculator App Engine

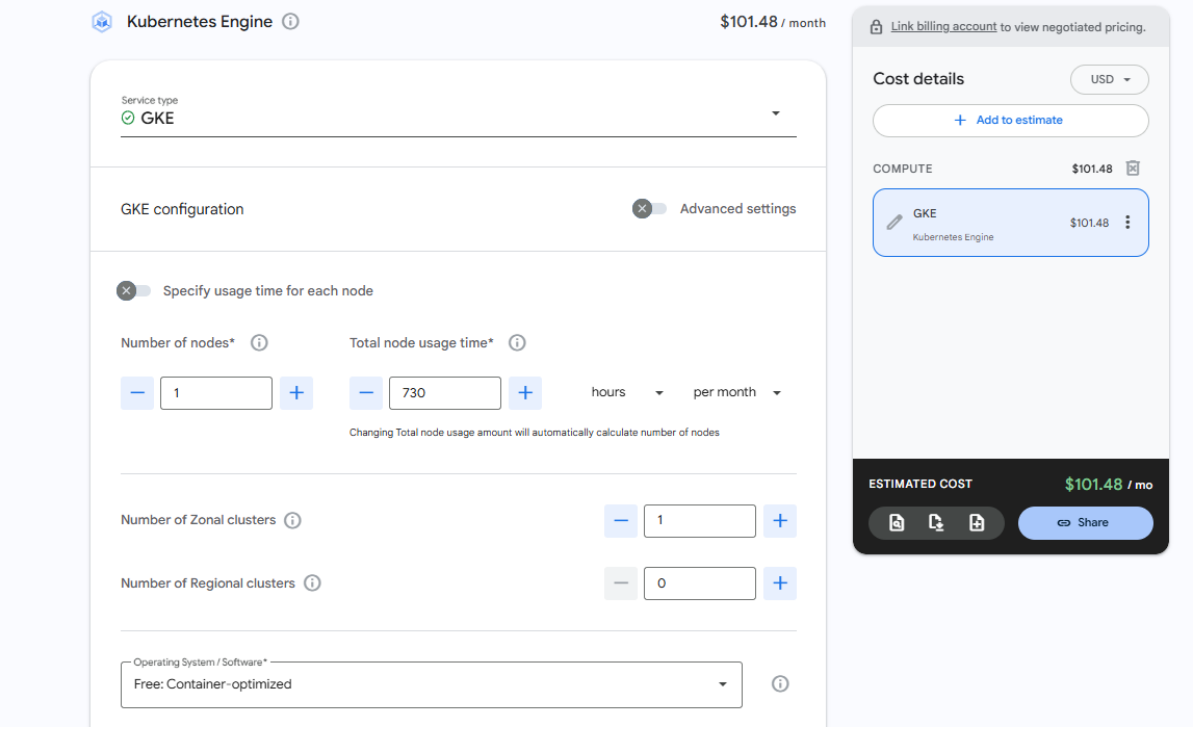

Fig. 11. Google Price Calculator App Engine