Análisis Detallado de Plataformas en la Nube para Contenedores Serverless

A continuación, se presenta un análisis detallado de las principales plataformas de servicios en la nube (AWS, Azure, GCP e IBM Cloud) para determinar cuál es la más adecuada para implementar nuestra aplicación, que con el funcionamiento de los tres contenedores Docker creados (frontend, endpoint para detección de fraude y endpoint para clasificación de severidad y detección de partes dañadas) como funciones serverless; evaluando diferentes características como la facilidad de uso, la escalabilidad, los servicios específicos ofrecidos, los costos asociados y otros aspectos técnicos relevantes para poder efectuar la elección del proveedor más apropiado.

Identificación de servicios

Consultado las páginas web de cado uno de los proveedores, se identificaron los siguientes servicios clave en cada plataforma para la ejecución de contenedores Docker en un modelo serverless:

- **AWS**: AWS Fargate (con Amazon ECS o EKS), AWS Lambda (con soporte para imágenes de contenedores) y AWS App Runner (AWS, 2025).
- **Azure**: Azure Container Instances (ACI), Azure Container Apps y Azure Functions (con soporte para contenedores Docker), así como Azure Kubernetes Service (AKS) con nodos virtuales (ACI) (Microsoft, 2025).
- **GCP**: Google Cloud Run, Dataproc Serverless para Spark y App Engine Flexible Environment (Google Cloud, 2025).
- **IBM Cloud**: IBM Cloud Code Engine (IBM, 2024).

Cada una de estas ofertas presenta un enfoque distinto para la contenerización serverless, variando en su nivel de abstracción, flexibilidad y los casos de uso a los que mejor se adaptan. A continuación, se presenta un análisis comparativo de estas plataformas en función de estos criterios.

Comparación Detallada de Plataformas

Facilidad de Uso

La facilidad de uso es un pilar fundamental para la adopción de cualquier plataforma en la nube, impactando directamente la productividad del desarrollador y la velocidad de despliegue. En el ámbito de los contenedores serverless, la abstracción de la infraestructura es clave para simplificar la experiencia.

- AWS: La amplitud de servicios de AWS puede ser un arma de doble filo. Si bien ofrece una solución para casi cualquier necesidad, la curva de aprendizaje inicial puede ser pronunciada. Sin embargo, servicios como AWS App Runner están diseñados para la simplicidad, permitiendo el despliegue de aplicaciones de solicitud/respuesta con una configuración mínima. AWS Fargate, aunque abstrae la gestión de servidores, requiere una comprensión de los conceptos de ECS o EKS. AWS Lambda, aunque serverless, se centra en el modelo de funciones y su integración con imágenes de contenedores añade una capa de complejidad para quienes no están familiarizados con Docker. La consola de AWS, aunque potente, puede ser densa para los nuevos usuarios (Dubendorfer, 2024).
- Azure: Azure ofrece una gama de opciones que se adaptan a diferentes niveles de experiencia. Azure Container Instances (ACI) es notablemente sencillo, ideal para despliegues rápidos y cargas de trabajo de ráfaga sin la necesidad de gestionar VMs o clústeres. Azure Container Apps simplifica aún más el despliegue de microservicios y aplicaciones basadas en eventos, proporcionando una plataforma gestionada que maneja el escalado y la infraestructura. Azure Functions con soporte para contenedores es accesible para desarrolladores de funciones. Aunque AKS con nodos virtuales (ACI) combina la flexibilidad de Kubernetes con la simplicidad serverless de ACI, aún requiere conocimientos de Kubernetes. La interfaz de Azure Portal es generalmente intuitiva y bien organizada (Dubendorfer, 2024).
- GCP: Google Cloud Run es el líder indiscutible en facilidad de uso para contenedores serverless. Su modelo de despliegue de "cualquier contenedor, sin estado, basado en solicitudes" en un entorno totalmente gestionado es excepcionalmente sencillo. Los desarrolladores pueden desplegar una imagen de Docker y Cloud Run se encarga automáticamente del escalado a cero, el balanceo de carga y la gestión de dominios. La simplicidad de Cloud Run lo hace ideal para prototipos rápidos y aplicaciones que requieren una configuración mínima. Otros servicios como Dataproc Serverless o App Engine Flexible son más específicos o requieren más configuración, pero Cloud Run es el punto de entrada más fácil para la contenerización serverless en GCP (Dubendorfer, 2024).
- IBM Cloud: IBM Cloud Code Engine es una plataforma unificada y totalmente gestionada que busca simplificar el despliegue de contenedores, funciones y trabajos por lotes. Su diseño está orientado a la facilidad de uso, abstrayendo la complejidad de la infraestructura subyacente. Permite a los usuarios desplegar imágenes de contenedores de forma directa. La integración con el ecosistema de IBM Cloud es fluida, pero la plataforma en sí misma es menos conocida que sus contrapartes de AWS, Azure o GCP, lo que podría implicar una curva de aprendizaje para aquellos no familiarizados con IBM Cloud. Sin embargo, para usuarios dentro del ecosistema de IBM, Code Engine ofrece una experiencia simplificada (Bridgwater, 2023).

Por lo anteriormente expuesto, para una aplicación que busca la máxima simplicidad en el despliegue de contenedores Docker como funciones serverless, Google Cloud Run parece ser la opción más fácil de usar. Le siguen de cerca Azure Container Instances y Azure Container Apps, y luego AWS App Runner e IBM Cloud Code Engine. La elección dependería del nivel de abstracción deseado y la familiaridad con el ecosistema del proveedor.

Escalabilidad

La escalabilidad es la capacidad de una aplicación para manejar un aumento en la carga de trabajo, y en el contexto serverless, esto se logra de forma automática y elástica, adaptándose a la demanda en tiempo real.

- AWS: AWS ofrece una escalabilidad robusta y probada para contenedores serverless. AWS Lambda escala automáticamente en función del número de invocaciones, creando nuevas instancias de la función según sea necesario para manejar la carga. AWS App Runner escala según el número de solicitudes concurrentes, ajustando dinámicamente el número de instancias. AWS Fargate (utilizado con ECS o EKS) proporciona escalabilidad automática basada en métricas como el uso de CPU, memoria o solicitudes del balanceador de carga. La infraestructura global de AWS y su capacidad para manejar cargas de trabajo masivas garantizan que las aplicaciones serverless puedan escalar desde cero hasta miles de instancias sin intervención manual. La flexibilidad de elegir entre diferentes servicios permite optimizar el escalado para diversos patrones de tráfico (Dubendorfer, 2024).
- Azure: Azure proporciona opciones de escalado flexibles y eficientes. Azure Container Instances (ACI) permite un escalado rápido y bajo demanda, ideal para cargas de trabajo de ráfaga, ya que los contenedores se inician en segundos. Azure Container Apps gestiona automáticamente el escalado de microservicios y aplicaciones basadas en eventos, adaptándose a la demanda. Azure Functions, al ser un servicio serverless, escala automáticamente en función de la carga de trabajo. Azure Kubernetes Service (AKS) con nodos virtuales (ACI) combina la orquestación de Kubernetes con la elasticidad de ACI, permitiendo escalar rápidamente los pods sin la necesidad de aprovisionar VMs adicionales. Azure se enfoca en proporcionar un escalado elástico que se adapta a las necesidades cambiantes de la aplicación (Dubendorfer, 2024).
- GCP: Google Cloud Run es excepcionalmente escalable, con la capacidad de escalar automáticamente desde cero instancias hasta miles en función del tráfico entrante. Su modelo de escalado basado en solicitudes es altamente eficiente y permite un uso óptimo de los recursos, lo que se traduce en costos reducidos cuando la aplicación está inactiva. Dataproc Serverless para Spark también escala automáticamente para manejar cargas de trabajo de big data. La infraestructura global de Google y su enfoque en la eficiencia del escalado automático hacen de Cloud Run una opción muy atractiva para aplicaciones con patrones de tráfico variables y picos de demanda impredecibles. La capacidad de escalar a cero es una ventaja significativa para la optimización de costos (Dubendorfer, 2024).
- IBM Cloud: IBM Cloud Code Engine ofrece escalado automático para sus cargas de trabajo en contenedores, ajustándose a la demanda. Está diseñado para manejar picos de tráfico y reducir recursos cuando la demanda disminuye, incluyendo el escalado a cero cuando no hay actividad. Como plataforma serverless totalmente gestionada, Code Engine se encarga de la infraestructura subyacente para garantizar que las aplicaciones puedan escalar de manera eficiente. Si bien los detalles granulares de escalado pueden no ser tan públicamente detallados como en AWS o GCP, la naturaleza serverless de Code Engine implica una capacidad de escalado robusta para cargas de trabajo en contenedores (Bridgwater, 2023).

Como podemos ver, todos los proveedores ofrecen excelentes capacidades de escalabilidad para contenedores serverless. IBM Cloud Code Engine proporciona un escalado automático confiable para sus cargas de trabajo IBM Cloud Code Engine proporciona un escalado automático confiable para sus cargas de trabajo destaca por su eficiencia en el escalado a cero y su modelo de pago por uso que se alinea perfectamente con la elasticidad. AWS y Azure también ofrecen soluciones de escalado muy maduras y flexibles, adaptadas a diferentes escenarios de uso.

Servicios Específicos Ofrecidos

La elección de una plataforma en la nube para esta aplicación, no solo debe basarse en la capacidad de ejecutar contenedores serverless, sino también en el ecosistema de servicios complementarios que facilitan la construcción de aplicaciones completas y robustas.

- AWS: AWS posee el ecosistema de servicios más amplio y maduro del mercado. Para contenedores serverless, además de Fargate, Lambda y App Runner, ofrece una vasta gama de servicios de base de datos (DynamoDB para NoSQL, Aurora Serverless para relacionales), servicios de mensajería (SQS, SNS, EventBridge), almacenamiento (S3 para objetos, EFS para archivos), redes (VPC para redes privadas, API Gateway para APIs), monitoreo (CloudWatch, X-Ray) y seguridad (IAM, Cognito). Esta riqueza de servicios permite a los desarrolladores construir arquitecturas muy complejas y personalizadas, integrando fácilmente los contenedores serverless con otras funcionalidades de la nube. La madurez del ecosistema de AWS es una ventaja significativa para proyectos a gran escala o con requisitos específicos (Dubendorfer, 2024).
- Azure: Azure ofrece un ecosistema de servicios completo y bien integrado. Además de ACI, Container Apps y Functions, cuenta con una suite robusta de servicios de base de datos (Azure Cosmos DB para NoSQL multimodelo, Azure SQL Database para relacionales), servicios de mensajería (Azure Service Bus, Event Hubs), almacenamiento (Azure Blob Storage), redes (Azure Virtual Network, Azure API Management), monitoreo (Azure Monitor, Application Insights) y seguridad (Azure Active Directory, Azure Security Center). Azure se enfoca en proporcionar una experiencia unificada y herramientas que facilitan la integración entre sus servicios, lo que es beneficioso para equipos que buscan una plataforma cohesiva y fácil de gestionar. La integración con herramientas de desarrollo de Microsoft también es un punto fuerte (Dubendorfer, 2024).
- GCP: Google Cloud Platform se distingue por su enfoque en la simplicidad, la escalabilidad y la integración con tecnologías de código abierto. Su servicio principal para contenedores serverless es Cloud Run. Complementando esto, GCP ofrece servicios de base de datos (Cloud Firestore para NoSQL, Cloud Spanner para bases de datos distribuidas globalmente, Cloud SQL para relacionales), mensajería (Cloud Pub/Sub), almacenamiento (Cloud Storage), redes (VPC, Cloud Load Balancing, API Gateway), monitoreo (Cloud Monitoring, Cloud Logging) y seguridad (Cloud IAM, Cloud Armor). GCP es conocido por su infraestructura de red de baja latencia y sus capacidades avanzadas en inteligencia artificial y machine learning, lo que lo convierte en una opción atractiva para aplicaciones que requieren estas funcionalidades (Dubendorfer, 2024).
- IBM Cloud: IBM Cloud ofrece una plataforma robusta con un fuerte énfasis en la inteligencia artificial y la hibridación de la nube. Además de IBM Cloud Code Engine,

cuenta con una variedad de servicios de datos (IBM Cloudant para NoSQL, Db2 on Cloud para relacionales), mensajería (IBM MQ), almacenamiento (IBM Cloud Object Storage), redes y seguridad. Un diferenciador clave de IBM Cloud es su profunda integración con los servicios de IBM Watson para inteligencia artificial y aprendizaje automático. Esto permite a los usuarios desplegar y consumir modelos de IA en contenedores serverless de manera eficiente, lo que puede ser una ventaja significativa para aplicaciones que dependen fuertemente de capacidades de IA y procesamiento de lenguaje natural (Bridgwater, 2023).

Derivado de lo anterior, concluimos que IBM Cloud se distingue por su integración con los servicios de Watson, lo que lo convierte en una opción sólida para aplicaciones con un fuerte componente de IA como es el caso de nuestra aplicación. AWS ofrece la mayor diversidad y madurez de servicios complementarios, lo que lo hace ideal para arquitecturas complejas. Azure y GCP también tienen ecosistemas muy completos y bien integrados, con Azure destacando en la unificación de herramientas y GCP en la simplicidad y IA.

Características Técnicas

Las características técnicas subyacentes de cada plataforma determinan el rendimiento, la flexibilidad y el control que los desarrolladores tienen sobre sus aplicaciones en contenedores serverless.

- AWS: En AWS, los contenedores serverless se ejecutan en entornos gestionados por AWS Fargate, que proporciona aislamiento a nivel de tarea, lo que mejora la seguridad y la estabilidad. Fargate abstrae completamente la gestión de la infraestructura subyacente, liberando a los desarrolladores de la preocupación por VMs, parches o actualizaciones. AWS Lambda, cuando se utiliza con imágenes de contenedores, permite a los desarrolladores empaquetar sus aplicaciones con todas sus dependencias, ofreciendo un control granular sobre el entorno de ejecución. AWS App Runner automatiza el despliegue, el balanceo de carga y el escalado, y proporciona un endpoint HTTPS seguro. Para cargas de trabajo más complejas, ECS y EKS con Fargate ofrecen control sobre la definición de tareas, redes y almacenamiento, permitiendo configuraciones detalladas de CPU, memoria y volúmenes. AWS soporta una amplia gama de lenguajes de programación y runtimes, y ofrece integración profunda con su vasta red de servicios, incluyendo VPC para redes privadas y IAM para control de acceso granular. La arquitectura de AWS está diseñada para alta disponibilidad y tolerancia a fallos (Dubendorfer, 2024).
- Azure: Azure Container Instances (ACI) proporciona un entorno de ejecución aislado para contenedores, con la capacidad de especificar recursos de CPU y memoria de forma granular. ACI soporta contenedores Linux y Windows, y permite la ejecución de múltiples contenedores dentro de un mismo grupo de contenedores, compartiendo recursos y red. Azure Container Apps se basa en Kubernetes y Dapr (Distributed Application Runtime), lo que proporciona capacidades avanzadas para microservicios, como el enrutamiento de tráfico, la gestión de secretos, la observabilidad y la resiliencia. Azure Functions con contenedores permite a los desarrolladores utilizar cualquier lenguaje o runtime compatible con Docker. AKS con nodos virtuales (ACI) combina la orquestación de Kubernetes con la elasticidad de ACI, permitiendo el despliegue de pods en un entorno serverless. Azure ofrece opciones de red virtual para aislamiento y conectividad híbrida,

y una sólida integración con Azure Active Directory para la gestión de identidades y accesos. La plataforma de Azure está diseñada para cumplir con una amplia gama de estándares de cumplimiento y seguridad (Dubendorfer, 2024).

- GCP: Google Cloud Run se basa en Knative, lo que le permite ejecutar contenedores en un entorno serverless con escalado automático y rápido, incluyendo el escalado a cero. Soporta cualquier lenguaje de programación y biblioteca que pueda ejecutarse en un contenedor Docker. Cloud Run maneja automáticamente el balanceo de carga, la gestión de certificados SSL y la asignación de dominios. Ofrece un modelo de ejecución basado en solicitudes, lo que lo hace ideal para microservicios y APIs. Permite la configuración de variables de entorno, secretos y montajes de volúmenes. La integración con Google Cloud Build facilita la construcción de imágenes de contenedores. GCP también proporciona una red global de baja latencia y servicios de seguridad robustos. La capacidad de desplegar Cloud Run en GKE (Google Kubernetes Engine) ofrece a los usuarios la opción de tener un mayor control sobre la infraestructura subyacente si es necesario, manteniendo la experiencia de desarrollo de Cloud Run. La filosofía de diseño de GCP se centra en la simplicidad, la velocidad y la eficiencia (Dubendorfer, 2024).
- IBM Cloud: IBM Cloud Code Engine es una plataforma unificada que soporta contenedores, funciones y trabajos por lotes. Se basa en tecnologías de código abierto como Knative y Kubernetes, lo que proporciona flexibilidad y portabilidad. Permite a los usuarios desplegar imágenes de contenedores desde Docker Hub o IBM Cloud Container Registry. Code Engine gestiona automáticamente el escalado, el balanceo de carga y la gestión de dominios. Ofrece integración con otros servicios de IBM Cloud, incluyendo servicios de datos y de IA. Permite la configuración de recursos de CPU y memoria, y soporta la inyección de variables de entorno y secretos. IBM Cloud se enfoca en proporcionar un entorno seguro y compatible para cargas de trabajo empresariales, con opciones para redes privadas y gestión de identidades. La plataforma de IBM Cloud está diseñada para soportar cargas de trabajo de misión crítica y entornos híbridos (Bridgwater, 2023).

En resumen, todas las plataformas ofrecen características técnicas sólidas para la ejecución de contenedores serverless. Sin embargo, debido a que IBM Cloud proporciona una plataforma unificada basada en estándares abiertos, ideal para cargas de trabajo empresariales y de IA, con un enfoque en la hibridación, parece ser la opción que mejor se adapta al caso de negocio de nuestro proyecto. AWS destaca por su madurez y la profundidad de sus opciones de configuración, ofreciendo un control granular. Azure proporciona una buena combinación de simplicidad (ACI) y control (AKS), con un fuerte enfoque en la seguridad y el cumplimiento. GCP sobresale con Cloud Run por su simplicidad y eficiencia en el escalado a cero, y su integración con Knative.

Evaluación de Costos y Modelos de Precios

El costo es un factor determinante en la elección de una plataforma en la nube. Los modelos de precios para contenedores serverless suelen basarse en el consumo de recursos, como vCPU, memoria y almacenamiento, así como en el tiempo de ejecución.

Modelos de Precios de AWS

AWS Fargate, el motor de computación serverless para contenedores en AWS (utilizado con ECS y EKS), se factura en función de los recursos de vCPU y memoria que la aplicación solicita, desde el momento en que se inicia la descarga de la imagen del contenedor hasta que la tarea finaliza, redondeado al segundo más cercano con un mínimo de un minuto. También se cobra por el almacenamiento efímero utilizado por la tarea (AWS, 2025).

vCPU por hora: El costo por vCPU por hora varía según la región y el tipo de sistema operativo (Linux/X86, Linux/ARM, Windows/X86). Por ejemplo, en la región US East (Ohio), el precio es de aproximadamente $0.04048 por vCPU por hora para Linux/X86.

GB de memoria por hora: El costo por GB de memoria por hora también varía por región y sistema operativo. En la misma región, es de aproximadamente $0.004445 por GB por hora para Linux/X86.

Almacenamiento efímero: Se cobra por GB por hora, con un costo mínimo. Por ejemplo, $0.0001 por GB por hora.

AWS Lambda: Se cobra por el número de solicitudes y la duración de la ejecución (GB-segundos). También se cobra por el almacenamiento efímero y la transferencia de datos.

AWS App Runner: Se cobra por el tiempo que el servicio está activo (vCPU y memoria) y por el número de solicitudes. También hay un costo por el almacenamiento de la imagen del contenedor.

En general, el modelo de precios de AWS para contenedores serverless es de pago por uso, lo que significa que solo se paga por los recursos consumidos. Esto puede ser muy rentable para cargas de trabajo intermitentes o variables, pero requiere una buena comprensión de los patrones de uso para estimar los costos con precisión. AWS también ofrece opciones de precios Spot para Fargate, que permiten ejecutar cargas de trabajo tolerantes a interrupciones a un costo significativamente menor (Dubendorfer, 2024).

Modelos de Precios de Azure

Azure Container Instances (ACI) se factura a nivel de "grupo de contenedores", que son asignaciones de recursos de vCPU y memoria que pueden ser utilizadas por un solo contenedor o divididas por múltiples contenedores. Los grupos de contenedores se facturan en función del número de vCPU y GB de memoria solicitados para el grupo de contenedores, redondeados al segundo más cercano. También hay un cargo adicional por la duración del software de Windows en grupos de contenedores de Windows (Microsoft, 2025).

Memoria por GB: El costo por GB de memoria varía según el plan (Pago por uso, plan de ahorro de 1 año, plan de ahorro de 3 años, Spot) y el sistema operativo (Linux/Windows). Por ejemplo, para Linux, el precio de pago por uso es de aproximadamente $3.8909 por GB.

vCPU por vCPU: El costo por vCPU también varía según el plan y el sistema operativo. Para Linux, el precio de pago por uso es de aproximadamente $35.4780 por vCPU.

Contenedores confidenciales y GPU: Tienen modelos de precios específicos.

Azure Container Apps tiene un modelo de precios basado en el consumo, que incluye cargos por vCPU-segundos, GB-segundos de memoria y solicitudes HTTP. Azure Functions también se

basa en el consumo, cobrando por ejecuciones y GB-segundos. En general, Azure ofrece un modelo de pago por uso similar al de AWS, con opciones de ahorro a través de planes de ahorro y precios Spot para cargas de trabajo interrumpibles (Dubendorfer, 2024).

Modelos de Precios de GCP

Google Cloud Run se factura únicamente por los recursos utilizados, redondeados a los 100 milisegundos más cercanos. El costo total de Cloud Run es la suma del uso de recursos después de aplicar la capa gratuita. Esto lo convierte en una opción muy rentable para cargas de trabajo intermitentes o de bajo tráfico, ya que escala a cero y no se cobra cuando no hay solicitudes (Google Cloud, 2025).

CPU por vCPU-segundo: El costo por vCPU-segundo varía según la región (Tier 1 o Tier 2). Por ejemplo, en regiones de Tier 1, el precio es de $0.00002400 por vCPU-segundo más allá de la capa gratuita. Se incluyen 180,000 vCPU-segundos gratuitos por mes.

Memoria por GiB-segundo: El costo por GiB-segundo de memoria también varía por región. En regiones de Tier 1, el precio es de $0.00000250 por GiB-segundo más allá de la capa gratuita. Se incluyen 360,000 GiB-segundos gratuitos por mes.

Solicitudes: Se cobra por millón de solicitudes. En regiones de Tier 1, el precio es de $0.40 por millón de solicitudes más allá de la capa gratuita. Se incluyen 2 millones de solicitudes gratuitas por mes.

Transferencia de datos: La transferencia de datos saliente a Internet se cobra según los precios de red de Google Cloud, con una capa gratuita de 1 GiB por mes dentro de Norteamérica.

Cloud Run también ofrece descuentos por uso comprometido (CUD) para reducir el costo de uso continuo. El modelo de precios de Cloud Run es muy transparente y favorable para el modelo serverless, ya que solo se paga por el consumo real, lo que lo hace muy atractivo para aplicaciones con patrones de tráfico variables (Dubendorfer, 2024).

Modelos de Precios de IBM Cloud Code Engine

IBM Cloud Code Engine se diferencia de las tecnologías de computación en la nube tradicionales en que solo se paga por los recursos que se utilizan. Se factura por la memoria y la vCPU que consumen las cargas de trabajo, así como por las llamadas HTTP entrantes. Si la aplicación escala a cero o el trabajo no se está ejecutando, no se consumen recursos y, por lo tanto, no se cobra (IBM, 2024).

Aplicaciones: Se cobra por las solicitudes HTTP entrantes y por los recursos de CPU y memoria consumidos por las instancias en ejecución de la aplicación. Las llamadas HTTP internas dentro de un proyecto entre las cargas de trabajo están excluidas del total de llamadas HTTP facturables.

Trabajos: Se cobra por los recursos de CPU y memoria consumidos por el trabajo cuando se ejecuta. No se cobra por la configuración del trabajo.

Funciones: Similar a las aplicaciones, se cobra por las solicitudes HTTP entrantes y por los recursos de CPU y memoria consumidos por las instancias en ejecución de la función.

Compilaciones: Cuando se compila una imagen a partir del código fuente, se factura por el tiempo de uso de la memoria y la vCPU que consume la compilación. Este cargo es independiente de los cargos que se puedan incurrir al usar la imagen resultante en una aplicación o ejecución de trabajo.

IBM Cloud Code Engine incluye una capa gratuita para experimentar antes de comprometerse. El modelo de precios es de pago por uso, lo que lo hace rentable para cargas de trabajo variables. Los costos pueden variar según la geografía (Bridgwater, 2023).

Análisis comparativo de plataformas

Considerando la necesidad de implementar nuestra aplicación con tres contenedores Docker como funciones serverless, y evaluando la facilidad de uso, escalabilidad, servicios específicos y modelos de precios de AWS, Azure, GCP e IBM Cloud, se presenta la siguiente tabla comparativa

Característica	AWS	Azure	GCP	IBM Cloud
Facilidad de Uso	Media (App Runner es fácil, Fargate/Lambda más complejos)	Alta (ACI y Container Apps son muy fáciles)	Muy Alta (Cloud Run es excepcionalmente fácil)	Media (Code Engine es fácil, pero menos conocido)
Escalabilidad	Muy Alta (Lambda, Fargate, App Runner escalan a cero y a gran escala)	Alta (ACI, Container Apps, AKS con nodos virtuales escalan bien)	Muy Alta (Cloud Run escala a cero de forma muy eficiente)	Alta (Code Engine escala automáticamente a cero)
Servicios Específicos	Ecosistema más amplio y maduro (Lambda, Fargate, App Runner, S3, DynamoDB, etc.)	Ecosistema completo y bien integrado (ACI, Container Apps, Functions, Cosmos DB, etc.)	Enfoque en simplicidad y IA (Cloud Run, Pub/Sub, Firestore, etc.)	Fuerte integración con Watson AI (Code Engine, Watson services)
Características Técnicas	Madurez, control granular, aislamiento a nivel de tarea (Fargate), amplia gama de runtimes.	Flexibilidad (Linux/Windows), Dapr, AKS con ACI, seguridad y cumplimiento.	Basado en Knative, escalado a cero, cualquier lenguaje/runtime, integración con GKE.	Basado en Knative/Kubernetes, plataforma unificada, enfoque empresarial/híbrido.
Modelo de Precios	Pago por uso (vCPU, memoria, duración), opciones Spot.	Pago por uso (vCPU, memoria, duración), planes de ahorro, opciones Spot.	Pago por uso (vCPU, memoria, solicitudes), capa gratuita, escalado a cero.	Pago por uso (vCPU, memoria, solicitudes), capa gratuita, escalado a cero

Recomendación con base en el análisis comparativo

Para una aplicación que requiere el funcionamiento de tres contenedores Docker como funciones serverless, las plataforma más adecuadas son IBM Cloud, específicamente utilizando IBM Cloud Code Engine y Google Cloud Platform (GCP), específicamente utilizando Cloud Run.

La justificación se basa en los siguientes puntos clave:

Facilidad de Uso: Google Cloud Run es, con diferencia, la opción más sencilla para desplegar contenedores Docker como funciones serverless. Su modelo de "cualquier contenedor, sin estado, basado en solicitudes" permite a los desarrolladores enfocarse en el código sin preocuparse por la infraestructura. Esto es crucial para una implementación rápida y eficiente de los tres contenedores. Por otra parte, IBM Cloud Code Engine, aunque inicialmente puede tener

una mayor curva de aprendizaje, con la capacitación adecuada es posible utilizar la plataforma de manera eficiente.

Escalabilidad a Cero y Eficiencia de Costos: Tanto IBM Cloud Code Engine como Google Cloud Run escalan automáticamente a cero instancias cuando no hay tráfico, lo que se traduce en una optimización de costos significativa para cargas de trabajo intermitentes o de bajo volumen. Cuando la demanda aumenta, escalan para manejar la carga. Este modelo de pago por uso, combinado con una la capa gratuita por parte de ambos proveedores, los hace extremadamente rentables. Para tres contenedores Docker que operan como funciones serverless, la capacidad de escalar a cero es una ventaja económica considerable.

Flexibilidad Técnica: Tanto Google Cloud Run como IBM Cloud Code Engine, al estar basados en Knative, permiten ejecutar cualquier imagen de contenedor Docker, lo que significa que los tres contenedores pueden ser escritos en cualquier lenguaje o utilizar cualquier dependencia. Esto proporciona una gran libertad a los desarrolladores.

Ecosistema de Servicios Complementarios: GCP ofrece un ecosistema de servicios bien integrado que complementa a Cloud Run. Servicios como Cloud Pub/Sub para mensajería, Cloud Firestore para bases de datos NoSQL, y Cloud Storage para almacenamiento de objetos, se integran perfectamente con Cloud Run, permitiendo construir una aplicación completa y robusta alrededor de los contenedores serverless. Por otra parte, IBM Cloud Code Engine ofrece servicios específicos para inteligencia artificial empresaria como Watson Services y Watson AI.

Rendimiento y Fiabilidad: La infraestructura global de Google y su enfoque en la eficiencia garantizan un alto rendimiento y fiabilidad para las aplicaciones desplegadas en Cloud Run.

Implementación

Derivado del análisis anteriormente realizado, se decidió probar el funcionamiento de la aplicación en las plataformas de Amazon, Google e IBM. A continuación presentamos una breve reseña de las impresiones que nos dejó cada una de ellas.

Amazon AWS Fargate

El procedimiento para subir imágenes de contenedores e implementarlas como funciones serverless en Amazon está muy bien documentado y existen un amplio número de video tutoriales elaborados por terceros que explican paso a paso este procedimiento.

Si por algún motivo se presenta algún error, AWS posee una herramienta de diagnóstico que indica las probables soluciones. A continuación presentamos una imagen del diagnóstico que nos dio cuando se presentó un error por falta de configuración de Roles y Permisos de Acceso.

Diagnose console errors with Amazon Q

I'm diagnosing the following console error:

```
Se ha producido un error al crear el clúster fraud_detection_cluster. Resource handler returned message: "Invalid
request provided: CreateCluster Invalid Request: Unable to assume the service linked role. Please verify that th...
```

Expand

Analysis

The error indicates that the user is unable to create the fraud_detection_cluster due to an issue with the ECS service-linked role. The service-linked role is required for ECS to access other AWS services on your behalf, and if it does not exist or cannot be assumed, the cluster creation will fail.

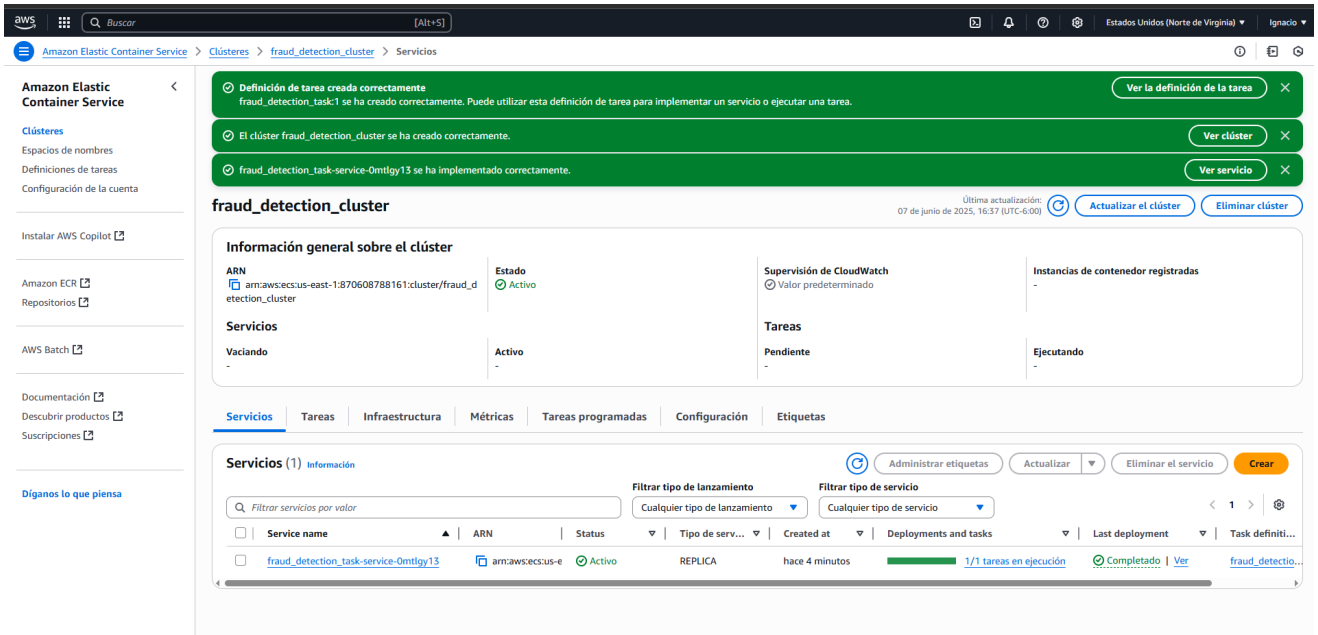
Resolution

Try the following steps to resolve your error:

1. Navigate to the IAM console in the AWS Management Console
2. In the left navigation pane, select 'Roles'
3. In the search bar, type 'AWSServiceRoleForECS' to find the ECS service-linked role
4. If the role doesn't exist:
   - Click on 'Create role'
   - Choose 'AWS service' as the trusted entity
   - From the list of AWS services, select 'Elastic Container Service'
   - Choose 'Elastic Container Service - Service-Linked Role'
   - Click 'Next' and then 'Create role'
5. If the role exists but there's an issue:
   - Click on the role name 'AWSServiceRoleForECS'
   - Review the role's permissions and trust relationship

Use of Amazon Q is subject to the AWS Responsible AI Policy. Outputs should be evaluated for accuracy and appropriateness for your use case.

Una vez resuelto el error, fue posible crear los clústers, subir las imágenes de los contenedores mediante la línea de comandos, configurar las definiciones de tarea y desplegar los servicios vinculados a los contenedores Docker de la aplicación como funciones Serverless.



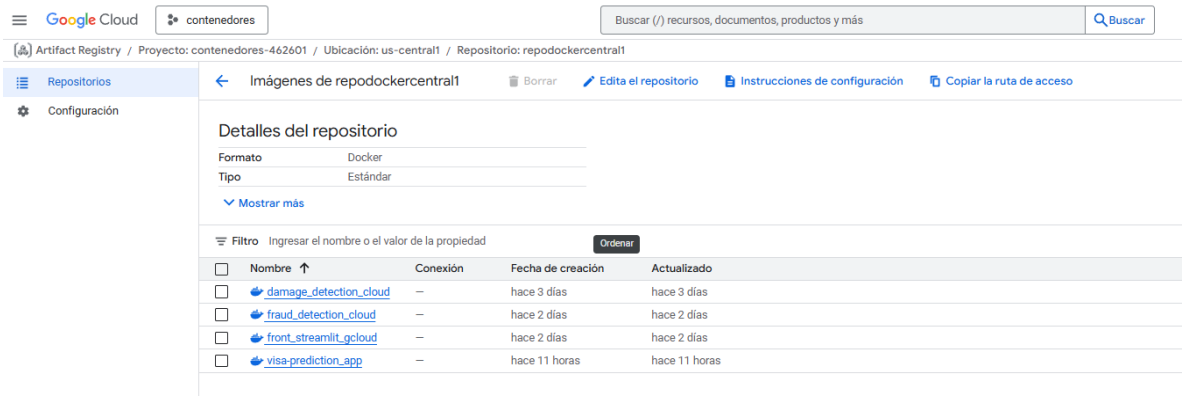
En cuanto al funcionamiento de la aplicación, debido a que las instancias no escalan a cero, la respuesta a las consultas es rápida, pero esto podría derivar en un mayor costo, dependiendo que necesidad de demanda tenga el uso de la aplicación, puesto que nunca se detiene la ejecución de estas instancias.

Google Cloud Run

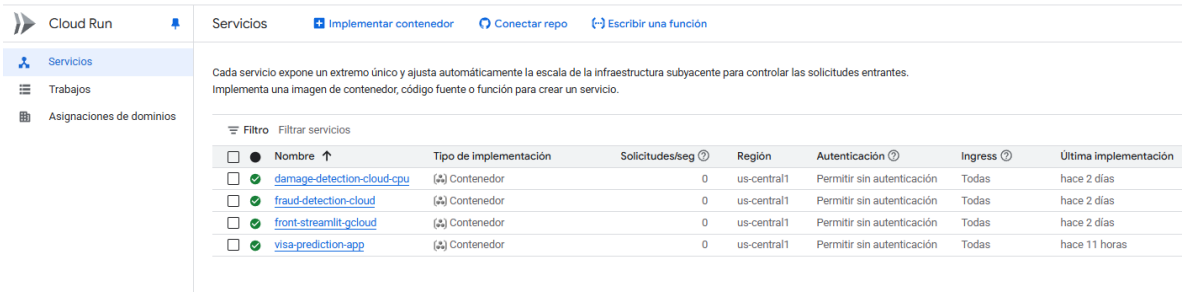
La documentación de Google Cloud Run para desplegar imágenes de contenedores Docker creadas por los usuarios, no es tan clara como la documentación de Amazon. Sin embargo, existen algunos video tutoriales creados por terceros que explican el procedimiento paso por paso.

Este procedimiento básicamente consiste en subir las imágenes de los contenedores a un repositorio creado en Artifact Registry mediante la línea de comandos, y posteriormente desplegarlas como funciones serverless mediante Cloud Run.

A continuación, se muestra una imagen de los contenedores Docker en el repositorio de Artifact Registry



Y una imagen de los servicios desplegados en Cloud Run:



En cuanto al funcionamiento de la aplicación, las instancias escalan a cero después de cierto tiempo de inactividad, pero están configuradas para utilizar una mayor cantidad de CPU en el arranque. Por tal motivo, la primera consulta a la aplicación tras un periodo de inactividad tarda aproximadamente 15 segundos, pero las siguientes consultas tardan la mitad de tiempo.

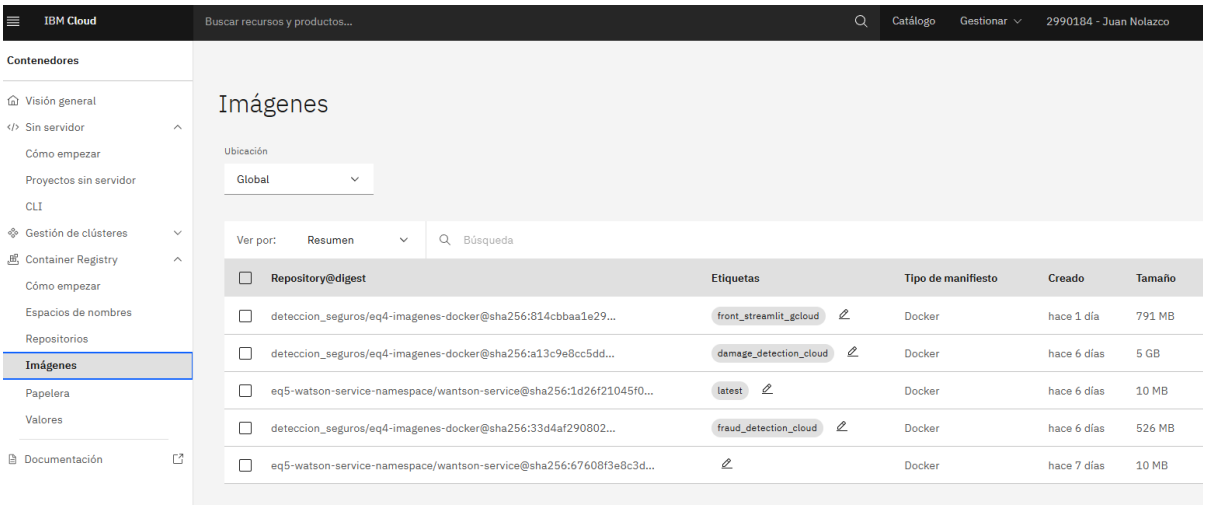
Por lo tanto, este escalado a cero contribuye a ahorrar recursos, pero hace que la experiencia del usuario sea un poco menos satisfactoria por el tiempo que toma la primera consulta tras un periodo de inactividad.

IBM Code Engine

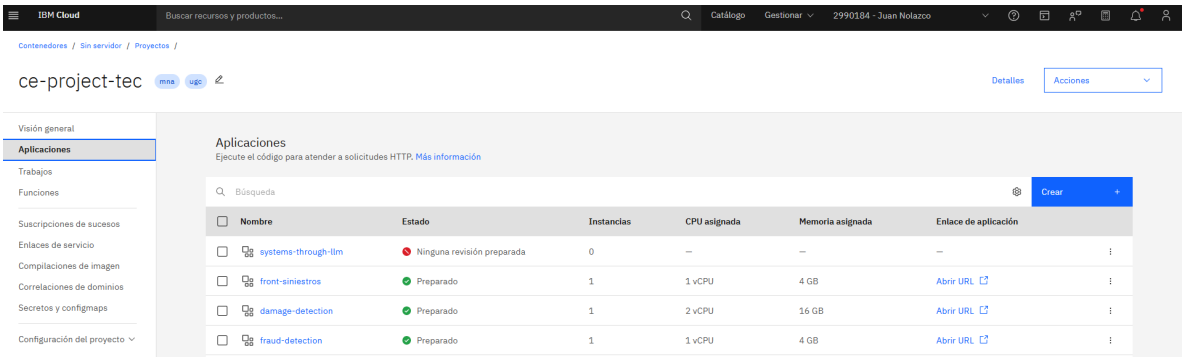
El procedimiento para ejecutar imágenes de contenedores Docker personalizadas como funciones serverless mediante IBM Code Engine se encuentra muy bien documentado en la plataforma de IBM. También hay algunos video tutoriales elaborados por el equipo de IBM que explican este procedimiento.

Al igual que en las otras plataformas, primero se debe crear un repositorio y luego subir a éste las imágenes de los contenedores mediante línea de comandos. Posteriormente, se crea una aplicación vinculada a estas imágenes, y la aplicación se encarga de hacer el despliegue y escalamiento de instancias de acuerdo con la configuración proporcionada.

A continuación, se muestra una imagen del repositorio de contenedores en IBM Code Engine



Y una imagen de las aplicaciones en ejecución:



En cuanto al funcionamiento de la aplicación, las instancias al igual que en Google Cloud escalan a cero después de un tiempo de inactividad. Sin embargo, no están configuradas para utilizar más CPU para un nuevo arranque, por lo que el arranque toma varios minutos. Por lo tanto, para lograr una buena experiencia de usuario, las aplicaciones se configuraron para no escalar a cero, por lo que siempre hay una instancia en ejecución para cada aplicación. Como se comentó anteriormente, el hecho de tener siempre una instancia en ejecución podría resultar más costoso dependiendo de las necesidades de demanda de uso de la aplicación.

Conclusiones

Cloud Run de GCP se destaca por su modelo de precios altamente eficiente para el caso de uso específico de contenedores Docker como funciones serverless que escalan a cero y utilizan una mayor cantidad de CPU para arrancar nuevamente. IBM Cloud Code Engine es una alternativa excelente, especialmente si ya se está en el ecosistema de IBM o se requiere una fuerte integración con los servicios de Watson y por requerimientos de uso de la aplicación, las instancias deben mantenerse en ejecución constantemente. Amazon Fargate, auqneues buena opción, en este caso no es tan atractiva puesto que al no escalar a cero se vuelve más costosa. Por lo tanto, para un despliegue general de contenedores serverless, Cloud Run ofrecería una propuesta de valor más atractiva, pero para el despliegue de contenedores serverless en un entorno empresarial, la integración con los servicios de Watson hace que IBM Cloud Code Engine se posicione como la mejor alternativa para el caso de negocio de nuestra aplicación.

Bibliografía

- AWS (2025). Go Serverless and containers - logical separation on AWS. https://docs.aws.amazon.com/whitepapers/latest/logical-separation/serverless-and-containers.html
- Bridgwater, A. (2023). IBM Cloud Code Engine: Serverless programming should be a default. Techzine Global. https://www.techzine.eu/blogs/analytics/105189/ibm-cloud-code-serverless-programming-should-be-a-default/
- Dubendorfer, D. (2024). AWS vs Azure vs Google: Cloud services comparison. Varonis.com. https://www.varonis.com/blog/aws-vs-azure-vs-google
- Google Cloud (2025). Cloud Run. https://cloud.google.com/run?hl=es

- IBM (2024). Cloud Code Engine. https://www.ibm.com/mx-es/products/code-engine
- Microsoft (2025). Containers on Azure. https://azure.microsoft.com/en-us/products/category/containers
- National Highway Traffic Safety Administration (NHTSA). (2023). Fatality and Injury Reporting System Tool (FIRST). https://cdan.dot.gov/query
- Rutecki (2022). Best techniques and metrics for Imbalanced Dataset. Kaggle. https://www.kaggle.com/code/marcinrutecki/best-techniques-and-metrics-for-imbalanced-dataset