IMPLEMENTACIÓN DE UNA ARQUITECTURA CLOUD COSTO EFICIENTE

David Mejia Restrepo

Ingeniero de sistemas

Asesor

Maria Bernarda Salazar Sánchez, Doctora en Ingeniería Electrónica

Jaime Alejandro Álvarez Valencia, Ingeniero Físico y Ms Student

Universidad de Antioquia

Facultad de ingeniería

Ingeniería de sistemas

Medellín

2023

TABLA DE CONTENIDO

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

AWS	Amazon Web Services
CI	Integración continua
CD	Despliegue continuo
WAF	Firewall de aplicación web
CMP	Plataforma de administración de contenedores
USD	Dólar estadounidense
EC2	Elastic Compute Cloud
ECS	Elastic Container Service
ECR	Elastic Container Register
VPN	Red privada virtual
DDoS	Ataque de denegación de servicio
NS	Nombre de servidor

RESUMEN

Este proyecto presenta el desarrollo e implementación de una mejora en la arquitectura sobre AWS para la plataforma Auco de la empresa Inversiones Cubit SAS. El objetivo fue optimizar la escalabilidad, disponibilidad, seguridad y eficiencia en los despliegues de este sistema ya consolidado. La solución involucró el uso de AWS ECS para lograr un sistema auto escalable capaz de responder a la demanda, la implementación de procesos de CI/CD mediante CodePipeline y CodeBuild para automatizar los nuevos despliegues, la adición de un firewall de aplicación web de Cloudflare para prevenir ataques DDoS y la integración con SonarQube para monitoreo de calidad de código. Se presentan los desafíos en la implementación sobre la arquitectura existente y la solución más adecuada para el contexto de Auco, los resultados muestran mejoras significativas que fortalecen técnicamente la plataforma en escalabilidad, eficiencia, seguridad y disponibilidad.

Palabras clave — **Arquitectura Cloud, AWS, ECS, CI/CD, WAF.**

ABSTRACT

This project presents the development and implementation of an architectural enhancement on AWS for Auco platform of Inversiones Cubit SAS. The objective was optimizing scalability, availability, security and efficiency in deployments on this consolidated system. The solution involved leveraging AWS ECS for an autoscaling system to meet demand, implementing CI/CD processes with CodePipeline and CodeBuild to automate new deployments, adding a Cloudflare web application firewall to prevent DDoS attacks, and integrating SonarQube for code quality monitoring. Challenges faced in implementation on the existing architecture are discussed along with the most suitable solution for Auco's context, results display significant improvements that technically strengthen the platform in scalability, efficiency, security and availability.

Keywords — **Cloud Architecure, AWS, ECS, CI/CD, WAF.**

I. INTRODUCCIÓN

En la era de la cuarta revolución industrial las empresas dependen cada vez más de implementar soluciones orientadas a la computación en la nube para potenciar su capacidad de crecimiento y adaptabilidad con respecto a la constante y emergente competencia del mercado, algunas empresas emergentes que se encuentran en etapas tempranas de desarrollo ven necesario reducir sus costos, pero sin opacar factores importantes como su seguridad, escalabilidad y eficiencia. Inversiones Cubit SAS es una empresa en etapa temprana de desarrollo que se enfoca principalmente en uno de sus productos llamado Auco, una solución para los procesos de gestión documental y validación de identidad se enfrenta a grandes competidores en este mercado, por tal motivo aborda la necesidad de optimizar su solución desde un nivel de arquitectura de la misma, de aquí surge la necesidad de buscar alternativas e implementar una arquitectura costo-eficiente de acuerdo a los servicios y necesidades del producto.

Para abordar esta necesidad, se han planteado enfoques y mejoras en múltiples áreas, no solo a nivel de infraestructura, sino también en procesos y requisitos. Uno de los cambios fundamentales fue la migración a un sistema auto escalable que permitiera a la plataforma mantener su rendimiento incluso ante un aumento significativo de usuarios. Este sistema se implementó sobre los proyectos de frontend que se encuentran en un monorepo lo cual representó un desafío a la hora de realizar despliegues óptimos de los módulos que fueron afectados.

La migración a esta nueva arquitectura trajo consigo desafíos adicionales, como la implementación de un sistema de despliegue eficiente siguiendo las prácticas de integración y despliegue continuo. También se llevó a cabo un proceso de pruebas de caja blanca para minimizar posibles despliegues fallidos.

Además de la migración y la optimización de procesos, se añadió una capa de seguridad adicional implementando un firewall de aplicación web proporcionado por una plataforma externa a AWS. Este paso fue esencial para controlar el flujo de peticiones no deseadas y garantizar la eficiencia en el proceso.

II. OBJETIVOS

A. Objetivo general

Desarrollar una nueva configuración de la arquitectura de los módulos frontend y backend de la empresa Inversiones Cubit SAS para el producto Auco, enfocado en implementar soluciones que aporten a la escalabilidad, disponibilidad y seguridad de la plataforma y a la eficiencia en los procesos de despliegue dentro de la compañía.

B. Objetivos específicos

1. Analizar los diferentes servicios y arquitecturas de contenedores que permite implementar los servicios de Amazon web services.
2. Implementar la arquitectura de contenedores seleccionada dentro Amazon web services, para permitir al sistema tolerar alto flujo de usuarios, asegurando además tener un sistema auto escalable y que sea costo eficiente.
3. Establecer un sistema de integración y despliegue continuo para los servicios alojados en bajo contenedores de la arquitectura implementada.
4. Implementar un servicio de firewall de aplicación web que permita filtrar y bloquear el acceso de usuarios no deseados no deseados desde los servicios expuestos.
5. Implementar la herramienta de análisis de caja blanca seleccionada e integrar esta misma a los sistemas de integración y despliegue continuo.

III. MARCO TEÓRICO

Para poder cumplir con los objetivos planteados en el proyecto, es necesario entender múltiples conceptos. A continuación, se presentan estos conceptos explicados con mayor detalle (Ilustración 1)

Ilustración 1. Servicios en la nube

Nota: Fuente https://bit.ly/48KHqU2

Computación en la nube y Amazon Web Services (AWS)

La computación en la nube es una tecnología que permite el acceso a recursos de computación, como redes, servidores, almacenamiento, aplicaciones y servicios, a través de Internet. Esta tecnología se ha convertido en una de las más importantes de la industria 4.0 y es clave para la transformación digital de las organizaciones. La computación en la nube ofrece varios modelos de servicio, como Software como Servicio (SaaS), Plataforma como Servicio (PaaS) e Infraestructura como Servicio (IaaS), que permiten a las empresas elegir el modelo que mejor se adapte a sus

necesidades. Algunas de las ventajas de la computación en la nube son la reducción de costos, la escalabilidad, la flexibilidad y la disponibilidad de los recursos. Sin embargo, también existen algunas desventajas, como la dependencia de la conexión a Internet y la posible falta de seguridad de los datos almacenados en la nube [1]. AWS es uno de los proveedores más importantes de computación en la nube y se define a sí misma como la nube líder en el mercado con la mayor adopción alrededor del mundo [2].

Ilustración 2. Kubernetes administrando contenedores Docker.

Nota: Fuente https://bit.ly/46njfJW

Contenedores y arquitectura de contenedores

Las *Virtual Machine Images* juegan un papel fundamental para la computación en la nube, permitiendo personalizar y crear réplicas de ambientes de ejecución garantizando el correcto funcionamiento de los aplicativos independientemente a donde se ejecuten estos mismos. Esta elasticidad fue pilar fundamental de la computación en la nube y de esto surgen los contenedores Docker como una alternativa ligera y dando inicio al patrón de arquitectura de microservicios que permiten desacoplar aplicaciones en múltiples y pequeños componentes independientes. Los contenedores son una tecnología que permite empaquetar aplicaciones y sus dependencias en un paquete portátil y autónomo que se puede ejecutar en cualquier entorno. Esto permite una mayor

eficiencia en el uso de los recursos y una mayor flexibilidad en la implementación de aplicaciones. Para crear estos sistemas distribuidos se requiere de una forma de administrar los contenedores y de esto surgen los CMPs como Kubernetes, Docker Swarm y Amazon ECS que permiten orquestar y definir reglas sobre los contenedores para tener un sistema adaptable y que puede aumentar o disminuir la cantidad de máquinas escalando el sistema dependiendo a las necesidades [3]-[4].

Ilustración 3. Proceso de DevOps.

Nota: Fuente https://bit.ly/3FavCgk

Integración continua (CI) y despliegue continuo (CD)

El CI/CD es una estrategia de desarrollo de software que surge de la cultura emergente de DevOps, la cual se ve reflejada en la empresa Inversiones Cubit SAS como una oportunidad de mejora frente a la competencia permitiéndoles crear de manera más ágil mejores y más estables aplicaciones. Esta estrategia de CI/CD se divide en dos procesos que se realizan mediante instrucciones automatizadas, integración continua que permite verificar la integridad del código antes en etapas tempranas de desarrollo, ejecutando análisis de código y pruebas, y por otro lado la entrega

continua que permite mediante un conjunto de instrucciones o pipeline de despliegue, desplegar el software a producción en cualquier momento. La implementación de CI/CD permite una mayor eficiencia en el desarrollo de software, una mayor calidad en el producto final y una mayor rapidez en la entrega del software [5].

Análisis de caja blanca

Este es un tipo de prueba que se realiza sobre el código fuente en lugar del producto compilado. Esta técnica permite encontrar defectos en el código y ayuda a determinar que este cumpla con los estándares de diseño definidos. Además de estos beneficios, la implementación de análisis de caja blanca también puede proporcionar otros beneficios, como la identificación temprana de errores, la mejora de la calidad del software y el ahorro de tiempo y costos. El análisis de caja blanca busca identificar problemas en el código fuente, como errores de sintaxis, errores de lógica, vulnerabilidades de seguridad y problemas de rendimiento [6].

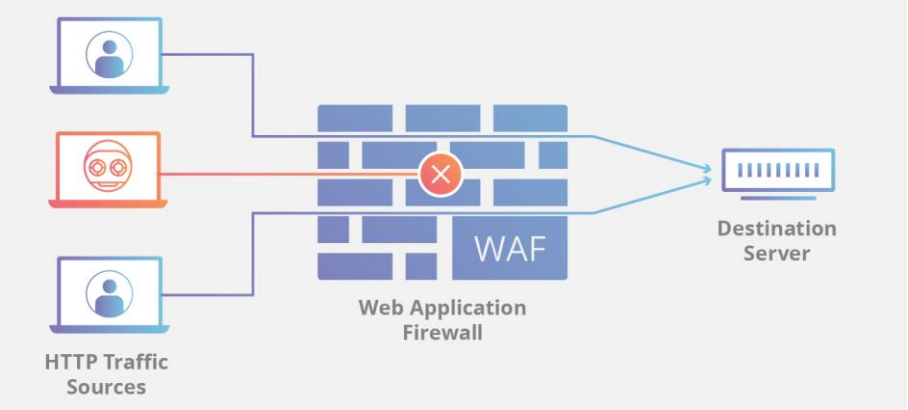

Ilustración 4. Representación de un firewall de aplicación web.

Nota: Fuente https://bit.ly/48MKxe9

WAF o firewall de aplicación web

Esta es una herramienta que ayuda a filtrar y monitorear el tráfico HTTP que ingresa a las aplicaciones web expuestas y que obliga al tráfico a tener que pasar antes por esta herramienta y así blindar el sistema ante algunas amenazas. Los WAF son una medida de seguridad importante para las aplicaciones web, ya que pueden proteger contra ataques comunes, como inyecciones SQL,

ataques de cross-site scripting (XSS) y ataques de denegación de servicio (DoS). Uno de los WAF más populares y que se explorará en el proceso de implementación es Cloudflare [7].

IV. METODOLOGÍA

Para la elaboración del presente proyecto se emplea una metodología mixta en la cual se definen unos objetivos sobre los cuales se realiza un proceso por etapas para explorar diferentes alternativas y se implementa una metodología más ágil para las entregas periódicas, esta se realiza bajo metodología Scrum y sobre las actividades definidas con el asesor interno. Se plantearon las siguientes actividades para identificar un estimado de tiempos de acuerdo a los hallazgos obtenidos:

Revisión bibliográfica

Se realiza una investigación exhaustiva de literatura relacionada con arquitecturas eficientes, escalables y seguras, así como herramientas y prácticas recomendadas para la automatización de despliegues y la revisión de calidad de código.

Evaluación de alternativas

Se analizan diferentes enfoques y tecnologías que puedan mejorar la capacidad de carga de los aplicativos como lo sería realizar un escalado vertical o uno horizontal, se revisaron diferentes administradores de contenedores como Kubernetes, ECS y EKS optando por ECS dado a su precio y ser un servicio propio de AWS, se analizaron diferentes herramientas para eliminar los despliegues manuales como lo fueron Jenkins, CircleCI y CodePipeline optando por esta última dado a la facilidad de integración al manejar los repositorios sobre otro servicio de AWS, se plantearon dos herramientas para fortalecer la seguridad frente a conexiones inusuales como el WAF de AWS y Cloudflare optando por este último después de un proceso de consultoría con un arquitecto externo y finalmente mejorar la calidad del código para lo cual se planteó realizar un análisis de caja blanca y se optó una implementación de uno de los servicios de SonarQube.

Diseño de la arquitectura

Se diseña una arquitectura optimizada que cumple con los requerimientos identificados y considerando las diferentes soluciones evaluadas previamente.

Implementación de un sistema escalable

Se lleva a cabo la implementación del sistema escalable para alojar los aplicativos de acuerdo con la arquitectura propuesta.

Implementación de la herramienta de CI/CD

Se lleva a cabo la implementación de la herramienta de integración continua (CI) y despliegue continuo (CD) acordada.

Migración al sistema escalable

Se lleva a cabo la migración del sistema actual a sistema escalable en un ambiente de desarrollo y luego en un ambiente productivo.

Implementación de un sistema de revisión de código

Se lleva a cabo la implementación del sistema de revisión de código aprobado y se integra al pipeline de CI/CD.

Implementación de un WAF

Se implementa un firewall de aplicación web **(**WAF) para limitar el tráfico no deseado en los aplicativos expuestos.

Documentación y presentación

Se elabora un informe detallado y una presentación que describe los resultados obtenidos y las soluciones implementadas.

V. ANÁLISIS Y RESULTADOS

Se diseña una arquitectura enfocada en soportar altas cargas de trabajo sobre AWS ECS y que facilite el proceso de despliegues automáticos usando la herramienta AWS CodePipeline (Ilustración 5), sobre está arquitectura se implementan dos versiones con la única diferencia del tipo de instancia asignada para la el alojamiento y ejecución de los contenedores Docker. Para ambas versiones se configura un cluster sobre AWS ECS para soportar 4 servicios con características similares, cambiando únicamente la configuración de infraestructura de cada una, enfocando la primera de estas en un sistema serverless mediante instancias AWS Fargate y la segunda utilizando un servidor dedicado con AWS EC2.

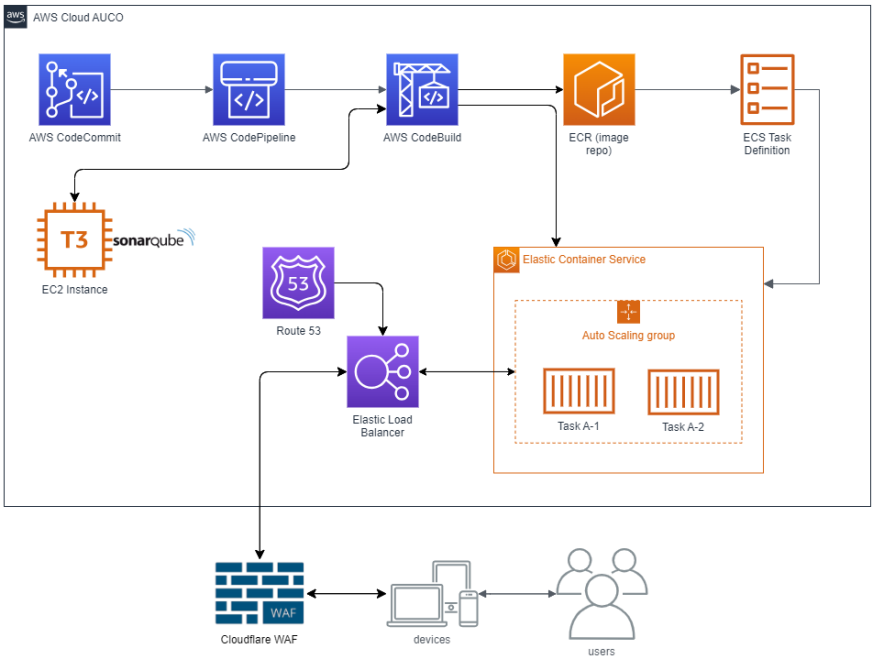

Ilustración 5. Diagrama de arquitectura de despliegue en ECS de Auco

El enfoque serverless sobre el cluster que implementa AWS Fargate asigna una instancia Fargate a cada servicio con una configuración de recursos de 0.5 vCPU y 1GB de memoria RAM y unas reglas de auto escalado con validaciones cada minuto, aumentando las instancias cuando se presente un uso de memoria o vCPU superior a 80% por 3 medidas consecutivas y disminuyendo el número de instancias cuando se presente un uso de memoria o vCPU inferior a 72% durante 15

medidas consecutivas. Durante la monitorización del estado y consumo de las instancias asignadas se evidencia que los recursos estimados para cada servicio son superiores a lo requerido por cada uno como se muestra en la Ilustración 6 e Ilustración 7, esto provoca que los servicios no requieran la ejecución de la tarea de escalado. Se realiza el balance del costo diario después de un mes de uso de esta configuración con dos cluster cada uno con 4 servicios y separados como producción y desarrollo, este balance arroja un costo diario de $4.74 USD para un aproximado de $142 USD mensual (Ilustración 8).

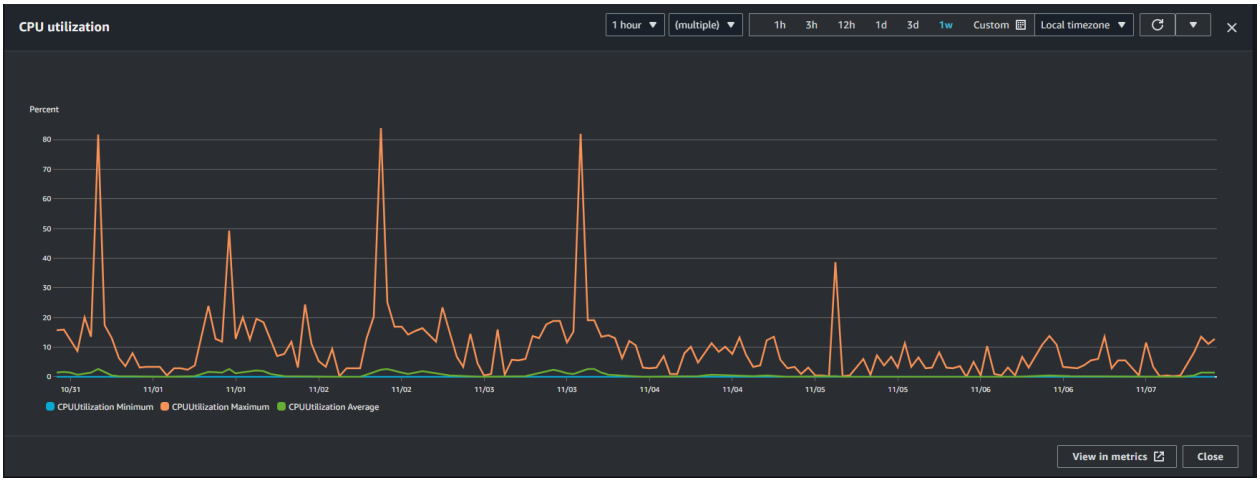

Ilustración 6. Uso de CPU de un servicio en el periodo de 31 de octubre a 7 de noviembre.

Ilustración 7. Uso de memoria de un servicio en el periodo de 31 de octubre a 7 de noviembre.

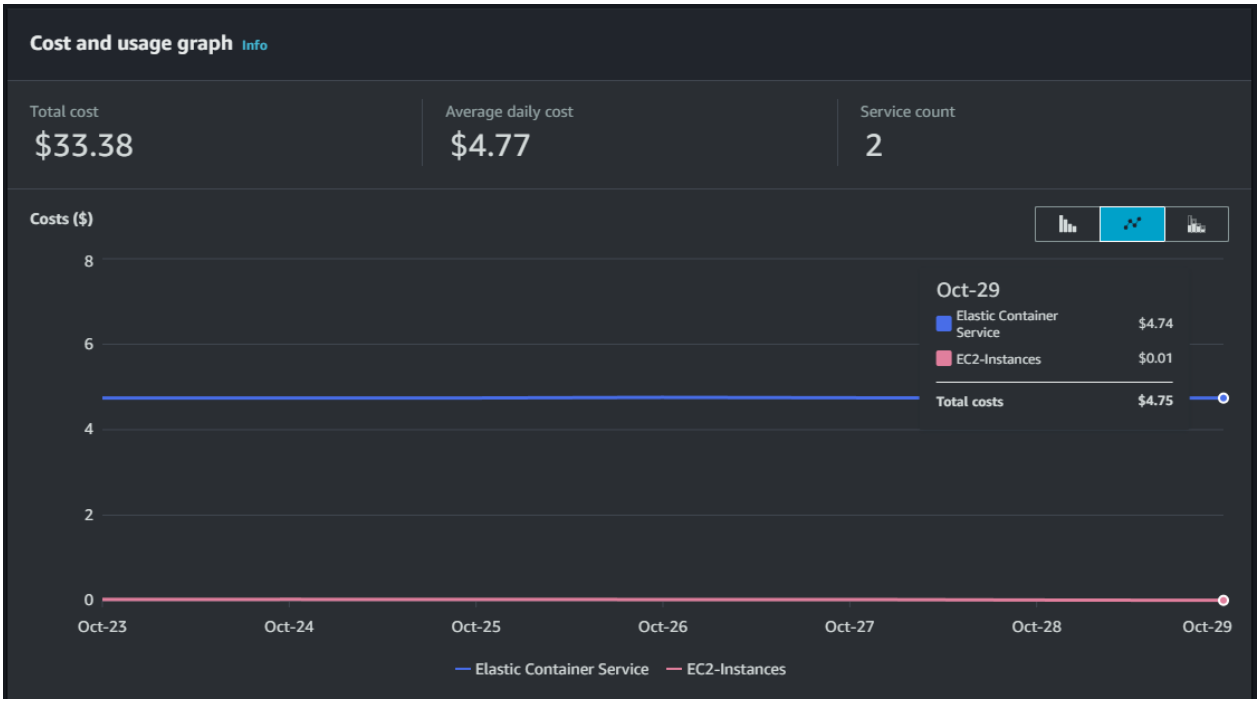

Ilustración 8. Costo de Elastic Container Service y EC2 entre octubre 23 y octubre 29.

En el caso de la implementación con servidor dedicado para la cual se asignan cuatro instancias de AWS EC2 de tipo t3a micro (2 vCPUs y 1 GiB de memoria) sobre las cuales se ejecutan y distribuyen todos los servicios del cluster. En este enfoque de instancias reservadas se redujeron la cantidad de recursos requeridas a cada servicio, para este caso se configuró cada servicio con 0.25 vCPU y 0.5 GiB de memoria RAM dando la posibilidad de escalar los servicios dentro de las instancias EC2 asignadas, este mismo se aplica únicamente sobre el ambiente de pruebas disminuyendo los costos el servicio de ECS en un total diario de $2.37 y aumentando el costo de las instancias EC2 en un total diario de $0.91 (Ilustración 9). El balanceador de cargas del cluster solicita nuevas instancias EC2 cuando cumple las reglas de auto escalado asignadas, este proceso se evidencia en la Ilustración 10.

Ilustración 9. Costo de Elastic Container Service y EC2 entre noviembre 6 y noviembre 12.

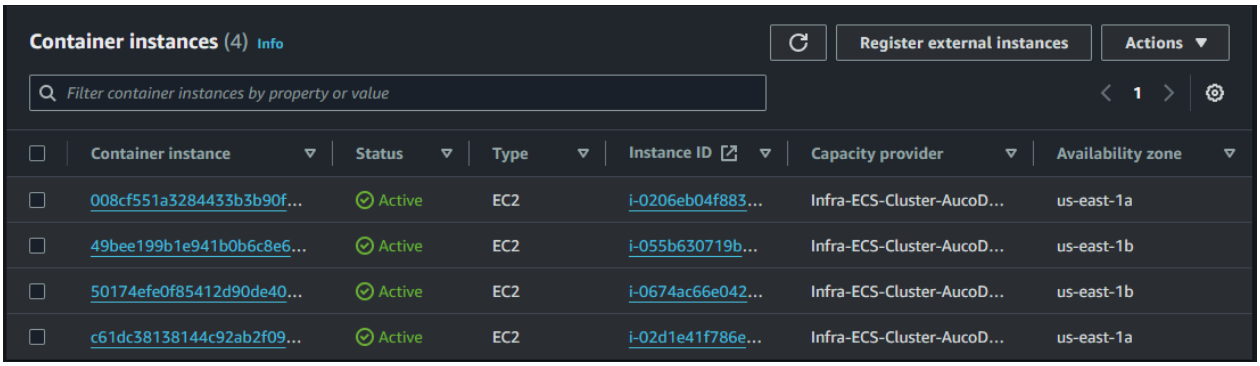

Ilustración 10. Escalado del cluster con instancias EC2.

Se utilizan diferentes herramientas de AWS enfocadas en el CI/CD como CodeCommit, CodeBuild y CodePipeline para automatizar los despliegues de los aplicativos sobre el clustes de ECS a nivel de infraestructura y a nivel de repositorio se utilizaron las diferentes utilidades que aporta Nx sobre la administración de mono-repositorios. De Nx se utiliza la personalización de comandos para poder ejecutar acciones sobre cada aplicativo utilizando un comando único, y se aprovecha la utilidad de affected para ejecutar los comandos únicamente sobre las aplicaciones que

sufrieron cambios en la última actualización del código. Sobre CodePipeline se configuran dos ambientes diferentes para ejecutar las instrucciones cuando se actualizan las ramas stage y prod, esta configuración como se muestra en la Ilustración 11. Proceso del CodePipeline sobre ac-monorepo., requiere que un usuario realice la aprobación de despliegue antes de realizar los respectivos cambios para lanzar una instancia de CodeBuild con la configuración adecuada para el despliegue sobre el ambiente que corresponde.

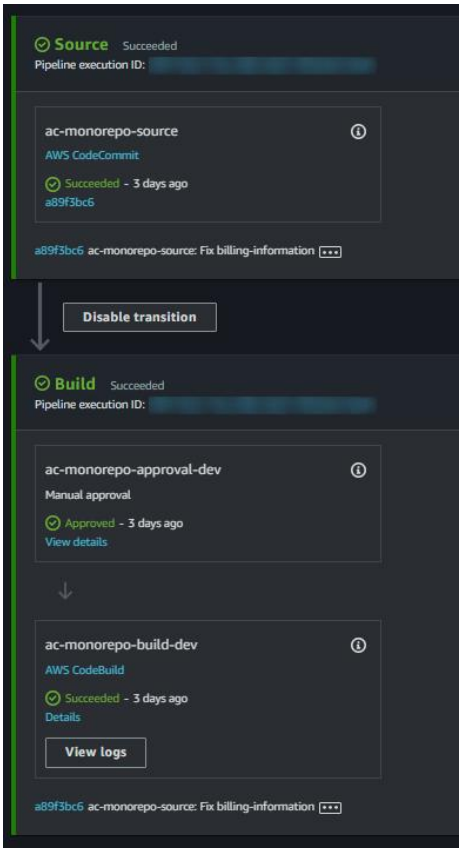

Ilustración 11. Proceso del CodePipeline sobre ac-monorepo.

La instancia de CodeBuild cuenta con una configuración que habilita git-credential-helper el cual permite acceder al histórico de Git del repositorio durante la ejecución de los diferentes comandos que permiten crear y desplegar los artefactos de cada aplicativo, estos procesos se componen de una etapa de instalación, una etapa de pre construcción y una etapa de construcción como se muestra en la Ilustración 12, esta primera etapa permite validar desde los logs sobre que rama se realiza el despliegue, instala las demencias necesarias y se asegura de generar el grafo de

dependencias de NX para asegurar su correcto funcionamiento, en la segunda etapa de pre construcción el sistema crea las variables necesarias para los diferentes comandos y establecer la conexión a ECR y Docker para la publicación de las imágenes, en la tercera etapa se realiza la ejecución de comandos propios y personalizados de cada proyecto dentro de NX con la herramienta affected encargada de ejecutar estos comandos solo si los proyectos sufrieron cambios en su última actualización, estos comandos personalizados realizan en etapas diferentes tareas como la creación de las imágenes Docker de cada proyecto, la publicación de cada imagen sobre ECR y finalmente un llamado de actualización sobre el aplicativo en ECS para desplegar la última versión publicada de la imagen Docker (Ilustración 13).

```
{
  "version": 0.2,
  "env": {
    "git-credential-helper": "yes"
  },
  "phases": {
    "install": {
      "runtime-versions": {
        "nodejs": 16
      },
      "commands": [
        "git status",
        "npm install --silent",
        "npx nx reset"
      ]
    },
    "pre_build": {
      "commands": [
        "NX_HEAD=origin/${GIT_BRANCH}",
        "NX_BASE=origin/${GIT_BRANCH}~1",
        "REPOSITORY_BASE=[redacted]dkr.ecr.us-east-1.amazonaws.com/",
        "COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)",
        "IMAGE_TAG=${COMMIT_HASH:=latest}",
        "aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $(echo $REPOSITORY_BASE)"
      ]
    },
    "build": {
      "commands": [
        "echo Building affected projects...",
        "npx nx affected --target=ci:build --parallel=2",
        "echo Building docker images...",
        "npx nx affected --target=docker:build --parallel=2 --repository_base=$(echo $REPOSITORY_BASE) --image_tag=$(echo $IMAGE_TAG) --stage=$(echo $STAGE)",
        "echo Deploy images to ecr...",
        "npx nx affected --target=docker:deploy --parallel=2 --repository_base=$(echo $REPOSITORY_BASE) --image_tag=$(echo $IMAGE_TAG) --stage=$(echo $STAGE)",
        "npx nx affected --target=aws:deploy --parallel=2 --stage=$(echo $STAGE_CAP)"
      ]
    }
  }
}
```

Ilustración 12. Configuración de CodeBuild.

```
"docker:build": {
  "executor": "nx:run-commands",
  "options": {
    "commands": [
      "docker build -f Dockerfile.app . -t {args.repository_base}ac-app-{args.stage}:latest --build-arg APP=dashboard-auco",
      "docker tag {args.repository_base}ac-app-{args.stage}:latest {args.repository_base}ac-app-{args.stage}:{args.image_tag}"
    ],
    "parallel": false
  }
},
"docker:deploy": {
  "executor": "nx:run-commands",
  "options": {
    "commands": [
      "docker push {args.repository_base}ac-app-{args.stage}:latest",
      "docker push {args.repository_base}ac-app-{args.stage}:{args.image_tag}"
    ],
    "parallel": false
  }
},
"aws:deploy": {
  "executor": "nx:run-commands",
  "options": {
    "commands": [
      "aws ecs update-service --cluster Auco{args.stage}2 --service App{args.stage}Service --force-new-deployment"
    ],
    "parallel": false
  }
},
```

Ilustración 13. Configuración de los proyectos en NX.

La creación de cada imagen Docker de las aplicaciones dentro del mono repositorio se realiza con ayuda de un Dockerfile genérico que permite mediante un parámetro de entrada crear la imagen adecuada para proyecto (Ilustración 14), como se muestra en la Ilustración 12 el comando de construcción de las aplicaciones se realiza antes de construir las imágenes Docker, este proceso se realiza de esta manera para evitar el proceso de instalación dentro de la construcción de cada imagen, reduciendo el peso de las mismas y aprovechando una herramienta experimental para la creación de aplicaciones standalone que genera los archivos necesarios para ejecutar el aplicativo y todas sus dependencias.

```
FROM public.ecr.aws/docker/library/node:lts-alpine3.17

WORKDIR /app

RUN apk add --no-cache libc6-compat
ARG APP=dashboard-auco

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --chown=nextjs:nodejs ./dist/apps/${APP}/.next/standalone .
COPY --chown=nextjs:nodejs ./dist/apps/${APP}/.next/static ./dist/apps/${APP}/.next/static
COPY --chown=nextjs:nodejs ./dist/apps/${APP}/public ./apps/${APP}/public

USER nextjs

EXPOSE 3000
ENV NODE_ENV=production
ENV APP_PATH=apps/${APP}/server.js
ENV PORT 3000

CMD node $APP_PATH

```

Ilustración 14. Configuración de Dockerfile.

Para proteger los servicios expuestos en los servicios de ECS, se implementa el WAF de Cloudflare enfocando su configuración en la prevención de ataques DDoS, para este proceso se actualizan los NS dentro de Route 53 de AWS para dirigir el tráfico de nuestra ruta principal sobre el servicio de Cloudflare y en el panel de administración se habilitan las reglas de firewall para IPs provenientes fuera de América.

Se configura una instancia de EC2 t3 medium (2 vCPU y 4 GiB de memoria) para ejecutar el servicio de SonarQube, este se encuentra en una red privada dentro de AWS para la cual se requiere la conexión mediante AWS VPN para poder acceder a la misma. Está instancia cuenta con una copia del mono repositorio lo cual le permite descargar las actualizaciones que se realicen sobre este, estas se acuerdan realizar de manera manual y no en integración con el pipeline de despliegue debido al estado del código a la hora de generar la solución, en lugar de esto se plantea como herramienta facilitando realizar revisiones en los cierres de sprint para así definir puntos de mejora sobre el actual código y correcciones a realizar sobre el código de las nuevas actualizaciones.

VI. CONCLUSIONES

La implementación de esta nueva arquitectura en AWS logra mejoras sobre la plataforma, estas permiten mantener un sistema escalable y que después de un monitoreo a las diferentes estrategias implementadas una selección adecuada a un sistema mantenible por su costo, este análisis sobre las diferentes implementaciones ayuda a plantear una solución más optima a la propuestas inicial de una infraestructura serverless para el cluster ECS debido a los requisitos sobre los servicios desplegados ayuda a definir que la solución más rentable sin afectar el desempeño de los sistemas corresponde al utilizar instancias reservadas y con capacidad de soportar los servicios, comparativamente ahorrando un 50% de los costos de operación en comparación de ambas infraestructuras.

Se minimizan los errores humanos durante los procesos de despliegue al desarrollar el sistema CI/CD con CodePipeline y CodeBuild, luego de un análisis y optimización de los procesos definidos para un despliegue exitoso y realizando una estandarización y codificación permitiendo minimizar la intervención humana en este proceso también reduciendo la carga sobre el equipo de desarrollo y permitiendo que se puedan agregar nuevos servicios al mono repositorio haciendo de esta solución de despliegue continuo escalable y mantenible en el tiempo.

Buscando fortalecer la seguridad y la calidad de los servicios prestados, se implementa una solución para prevenir ataques DDoS que podrían causar intermitencias en el servicio y/o aumento en los costos de infraestructura, permitiendo generar confianza en la solución de ECS generada. Este proceso se siendo complementado con la implementación de SonarQube, que se explora como herramienta para realizar revisiones de código de los nuevos desarrollos y para integrar puntos de mejora en la proyección de futuros desarrollos aumentando gradualmente el grado de confianza en seguridad y eficiencia de los servicios presentados.

REFERENCIAS

[1] S. Shilpashree, R. P. Renuka y C. Parvathi, «Cloud computing an overview,» *International Journal of Engineering & Technology*, vol. 7, nº 4, pp. 2743-2746, 2018.

[2] Amazon Web Services, Inc, «What is AWS,» [En línea]. Available: https://aws.amazon.com/what-is-aws/. [Último acceso: 12 Agosto 2023].

[3] A. Pérez, G. Moltó, M. Caballer y A. Calatrava, «Serverless computing for container-based architectures,» *Future Generation Computer Systems*, vol. 83, pp. 50-59, 2018.

[4] B. Burns, J. Beda, K. Hightower y L. Evenson, Kuernetes: Up & Running. Dive into the Future of Infrastructure, 3rd ed., O'Reilly Media, 2022.

[5] M. Soni, «End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery,» de *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Bangalore, India, 2015.

[6] S. Nidhra y J. Dondeti, «BLACK BOX AND WHITE BOX TESTING,» *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, nº 2, pp. 29-50, 2012.

[7] Cloudflare, Inc., «What is a WAF? | Web Application Firewall explained,» [En línea]. Available: https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/. [Último acceso: 12 Agosto 2023].