MEMORIA FINAL DE PROYECTO

Desarrollo, implementación y despliegue de una aplicación Web en un entorno DevOps, contenerización y servidores Cloud.

CICLO FORMATIVO DE GRADO SUPERIOR

Administración de Sistemas Informáticos en Red

AUTOR: ALBERTO MIGUEL PURIFICACIÓN PÉREZ

Tutor: José María Muñoz Morano

Coordinador: Pablo Palacio

Tabla de contenido

INDICE de tablas e ilustraciones

1 JUSTIFICACIÓN DEL PROYECTO

La cultura de trabajo DevOps cada vez está más extendida en los entornos de desarrollo y operaciones de las empresas que se dedican a la creación de aplicaciones. Las ventajas que se obtienen gracias a esta metodología de trabajo son muchas, y son aplicables a cada uno de los estados de la creación de la aplicación.

A su vez, el uso de plataformas de computación en la nube (AWS, Azure, Google Cloud, etc) es cada vez mayor gracias a las ventajas de escalabilidad y facilidad de uso y despliegue que ofrecen. Además, estos servicios solo cobran por los recursos utilizados, lo que permite a las empresas ahorrar costes innecesarios a la hora de adquirir servidores físicos sobredimensionados pensando en un futuro que quizás nunca llegue.

Sumando estas dos filosofías, se podría mejorar la infraestructura y el método de trabajo de cualquier empresa de software, de manera fácil y rápida, y obtener beneficios desde el primer momento de la vida de cualquier aplicación web.

2 OBJETIVOS

El propósito general de este proyecto es la creación de una infraestructura, basada en la computación en la nube, para la implementación de una plataforma web de una empresa con una infraestructura *“on-premise”.*

También se adoptará la filosofía de trabajo DevOps, mejorando el método de trabajo actual y acelerando todo el proceso de creación de la aplicación.

Durante todo este proyecto se explicarán las tecnologías, metodologías e infraestructura utilizadas para llevar a cabo la actualización de los servidores y la puesta en marcha de la metodología DevOps. Se explicarán los procesos de entrega continua y despliegue continuo y se explicaran las automatizaciones que el sistema tendrá para llevar a cabo la producción.

Con suerte, al final de este proyecto se habrá podrido demostrar que, tanto DevOps como la computación en la nube, son el futuro de las empresas de desarrollo de software y se habrá podido estudiar el proceso que tendrá que llevar a cabo una empresa tradicional para conseguir ser más funcional y rentable.

2.1 Requisitos de la empresa

Una vez evaluada la situación actual de la infraestructura de la empresa, se examinó cuáles son los requisitos de la misma para la implementación de la aplicación y si la situación se adecua a estos.

- La infraestructura de la empresa tiene que ser fácilmente escalable ya que se prevé que el tráfico vaya incrementando con el paso de los meses. En la situación de la empresa, esto significaba la adquisición de más recursos, aumentando el presupuesto dedicado a la aplicación en el futuro.
- La empresa no se podía permitir que la plataforma se quedase obsoleta con el paso del tiempo, debido a unos servidores anticuados. Cambiar todos los servidores podría ser algo que la empresa no podía asumir.
- La utilización de máquinas virtuales ralentizaba el desarrollo y mantenimiento de la aplicación, ya que los entornos de desarrollo e implementación podrían variar ente sí. Además, el uso de máquinas virtuales requiere una mayor potencia en los servidores ya que necesitan instalar el hipervisor en el propio servidor.
- Implementación de una cultura DevOps para que los equipos de desarrollo y operaciones trabajen juntos, con el fin de que la aplicación sea mejor, se entregue en menos tiempo y sea más fiable. Esta cultura funciona mejor bajo una infraestructura basada en la nube y la utilización de tecnologías como Docker, Amazon Elastic Container Service, CodePipeline o Git.

El sistema debía cumplir una serie de requisitos, entre los que se podría destacar, la fiabilidad, la rapidez y la funcionalidad. Pero también debía cumplir otra serie de requisitos, como una interfaz intuitiva y fácil de usar, que no contenga errores, etc.

Para explicar todos los requisitos detalladamente se dividieron en requisitos funcionales y requisitos no funcionales.

2.1.1 Requisitos funcionales

Estos requisitos son los que tienen que ver con el funcionamiento directo de la aplicación y corresponden, en su mayoría, al código e infraestructura de la misma. Estos requisitos son:

- La aplicación se trataba de una web dedicada a la gestión de colecciones de películas en diferentes formatos. Cada usuario podía buscar, en una base de datos, la película que quería añadir a su colección, agregarla y después llevar la cuenta de cuantas películas tiene en su colección.
- La aplicación debía tener un formulario de registro para que los usuarios de puedan registrar y personalizar su colección. Este formulario pediría al usuario un nombre de usuario, un email y una contraseña. Todos los campos eran obligatorios y no se podrán duplicar con los datos de otros usuarios
- Se creo una base de datos para cada usuario. De esta forma, cada usuario podía tener su colección privada siempre disponible.
- Una de las características de la ficha de cada película era la posibilidad de marcarla como vista o no vista. De esta forma, el usuario además llevaba la cuenta de cuantas películas de su colección ha visto y cuáles no.
- Tanto en la base de datos de películas, como en la colección de cada usuario, se creaban filtros por género, director, año de estreno, formato de la edición, etc.

2.1.2 Requisitos no funcionales

Estos requisitos son propiedades o cualidades que el sistema debe cumplir. Son en su mayoría, los objetivos que la aplicación debe cumplir para que los usuarios queden contentos con la misma. Estos requisitos son:

- Aplicación muy fácil de usar.
- Visualmente atractiva y de fácil entendimiento de todos los apartados.
- Web responsive con visualización para dispositivos móviles.
- Tiempo de carga muy bajo.

2.2 Posibles soluciones

A continuación, se exponen las diferentes opciones que mejorarían la situación de la empresa y ayudarían a un mejor desarrollo de la aplicación.

- Primero, se propone la utilización de entornos basados en computación en la nube. Esta opción proporcionaba una mejor escalabilidad, ya que los principales

proveedores de servidores en la nube facilitan la posibilidad de aumentar recursos, como procesadores o memoria, según sean las necesidades del servicio en cada momento.

Además, estos proveedores solo cobran en función de los recursos contratados, es decir, cuantos más recursos se utilicen, mayor será el precio del servicio. De esta manera solo se pagaría por lo que realmente se necesite.

- Los proveedores de servidores en la nube siempre dispondrán de las mejores tecnologías, por lo que podremos disponer de cualquier mejora al instante sin aumentar el gasto, proporcionando a la aplicación una ventaja importante.
- La utilización de contenedores, en lugar de máquinas virtuales, ya que se aumentaría la velocidad de la aplicación y se reduciría el uso de recursos. Los contenedores no necesitan un sistema hipervisor corriendo por encima de sistema operativo y solo usaría las dependencias necesarias para correr la aplicación.

  Además, el uso de contenedores posibilita el empaquetado de la aplicación y sus dependencias, a diferencia de una máquina virtual. De esta manera, el entrono necesario para el desarrollo y el despliegue de la aplicación siempre será el mismo y se reducirá el número de errores.
- Con este modelo de trabajo se aumenta la confiabilidad al aumentar la calidad y seguridad del código. Esto de hace gracias a la integración continua y la entrega continua (CI-CD), que permiten comprobar que cada actualización realizada sobre el producto es funcional, válida y segura gracias, no solo al feedback de desplegar los productos sino, al feedback instantáneo de las herramientas de calidad.

2.3 Solución elegida

Como se ha explicado en el apartado anterior, se eligió un entorno con filosofía DevOps para realizar la aplicación. Esta filosofía de trabajo contempla 5 fases que se van repitiendo durante la vida de la aplicación.

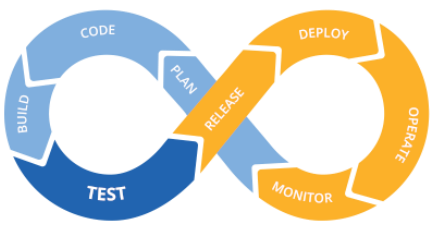

Ilustración 1. Ciclo de vida aplicación en Dev-Ops.

Existen diferentes soluciones dependiendo del entorno y la fase en la que se encuentre la aplicación. Las diferentes fases son:

- **Planificación**: Se realiza un estudio sobre que se quiere conseguir con la aplicación, que recursos se van a necesitar y se hace el reparto de tareas que se deben realizar. Este proyecto puede entrar dentro de esta fase.
- **Construcción**: En esta fase comienza a construirse el código de la aplicación. Se usarán tecnologías como Docker para crear un entorno de desarrollo idéntico al entorno de producción gracias a la facilidad de despliegue y uso de los contenedores.
- **Integración continua**: Se trata de integrar el código realizado con el ya existente o con el resto de código realizado por otras personas. Esta fase debe hacerse diariamente de forma automatizada. Para ello se eligió el uso de GitHub, como repositorio del código y gestión de versiones, y Amazon Codepipeline para automatizar las pruebas del código en un servidor de integración continua. De esta forma, esta fase se realizará de una forma muy rápida y se podrán solucionar los errores de código de manera más eficiente.
- **Despliegue**: Una vez comprobados los errores del código, mediante nuestro servidor de integración continua, se debe automatizar el despliegue del nuevo código. A esto se le denomina despliegue continuo.

  Se creo un entorno de despliegue previo al de producción, llamado entorno de integración, donde se realizaron pruebas exhaustivas a la aplicación, tanto por el equipo de desarrollo como por el cliente. De esta manera el feedback se obtuvo de una manera mucho más rápida y directa antes de hacer el despliegue final en el entorno de producción.

  De esta manera, nuestra aplicación seria 100% funcional antes de pasar al entorno de producción y los errores se solventarían de una manera rápida y eficaz sin necesidad de detener el servicio.
- **Monitorización**: Se realiza un seguimiento del funcionamiento de la aplicación. Se presta atención al uso de memoria, los tiempos de carga, etc. También se tiene en cuentas posibles avisos sobre fallos en el código y conflictos entre dependencias.

Tecnología	Planificación	Construcción	CI	Despliegue	Monitorización
Jira	X				
Docker		X		X	
Docker Hub			X		
GitHub		X	X		
CodePipeline			X	X	
Amazon ECS				X	
AWS Cloudwatch					X

Tabla 1. Tabla de tecnologías usadas en cada etapa

Para la realización de toda la infraestructura IT necesaria para el desarrollo de la aplicación se utilizaron los servidores físicos disponibles en la empresa como servidores de desarrollo y de integración continua. Mientras que para el entorno de integración y el de producción se usaron servidores en la nube de **Amazon Web Services**.

Se ha estudiado el precio de otras plataformas, como Azure o Google Cloud, y todas tienen un coste muy parecido. La ventaja de AWS es que proporcionan una capa gratuita de 12 meses de uso, siempre que no se sobrepasen ciertos niveles de recursos. Esto es una gran ventaja que abarata el coste final de la aplicación.

3 INTRODUCCION

3.1 Estado actual del sistema

- **Hardware**: Equipos físicos
- **Sistemas Operativos**: Software sobre el cual funcionan los diferentes servidores.
- **Redes**: Topología de la red empresarial, segmentación y conexión entre equipos.
- **Seguridad perimetral**: Equipos físicos de seguridad y todas las listas de acceso que forman las reglas de estos equipos.
- **Software**: Programas que usan los SO de los servidores.
- **Copias de seguridad**: Respaldo de toda la información almacenada en los servidores y configuración de los equipos de red.

3.1.1 Hardware

En el momento de hacer el estudio, el cliente disponía de un rack formado por tres servidores:

- Uno para la implementación de aplicaciones web virtualizado.
- Otro para la base de datos virtualizado.
- Otro para la organización de la empresa y para copias de seguridad. Se virtualizaban dos Windows Server 2019.

3.1.2 Sistemas Operativos

Los sistemas operativos instalados en los diferentes servidores eran:

- **CentOS**: Instalado en el servidor de la aplicación web y en el servidor de la base de datos.
- **Windows Server 2019**: Instalado en el servidor que proporciona organización a la empresa con Active Directory y en el servidor de copias de seguridad.

3.1.3 Redes

- **Router**: Elemento proporcionado por el proveedor IPS que dota de internet a la empresa mediante una fibra de 600 Mb simétricos. Estaba conectado al puerto WAN del firewall.
- **FortiSwitch**: Era el switch encargado de conectar todos los elementos y segmentar toda la red de la empresa. Esta red estaba segmentada en varias VLANs:
    - VLAN de Administración: Esta red era la utilizada para administrar todos los equipos de la red, como servidores, firewall, switchs y puntos de acceso.
    - VLAN DMZ publica: En esta red era donde se encontraba el servidor web para que se pudiera acceder a él desde el exterior, pero era inaccesible por los equipos de los usuarios de la empresa.
    - VLAN DMZ privada: Era donde se encontraba el servidor de la base de datos. Esta red era inaccesible desde el internet.
    - VLAN Usuarios: Era la red utilizada por los usuarios de la empresa.

  - VLAN Visitantes: Era la red pública para las visitas en la oficina de la empresa.

- **FortiGate**: Todos los switches de la empresa estaban conectados a cada una las interfaces del firewall. El firewall también era el encargado de gestionar la política de seguridad de la empresa, aplicando reglas para permitir o denegar el acceso a las diferentes VLANs e internet.

3.1.4 Seguridad perimetral

El firewall era el encargado de la seguridad de la empresa, internet y las DMZ públicas y privadas. Por defecto se deniega todo lo que no esté explícitamente permitido, para ello se configuran ciertas reglas. Las reglas de control de acceso del firewall eran las siguientes:

- Se permite el tráfico desde internet y la DMZ publica, a través del puerto 80, para que los usuarios puedan hacer peticiones a la aplicación.
- Se permite el tráfico desde la DMZ publica, a través del puerto 80, para que la aplicación pueda devolver los recursos necesarios a las peticiones de los usuarios.
- Se permite el tráfico desde la red VLAN de Administración hasta la DMZ pública y privada para los administradores de sistemas puedan tener acceso a los servidores.
- Se permite el acceso a la base de datos, a través del puerto 3306, entre la DMZ publica y la DMZ privada y viceversa. Con esto se permite que la aplicación tenga acceso a la base de datos.
- Se permite el tráfico entre la red de Visitantes e internet y se deniega el tráfico hacia el resto de VLANs. Con esto se impide que los visitantes tengan acceso a la red empresarial.

3.1.5 Software

- **Apache2 2.4.53**: Era el software encargado de crear el servidor web donde estaba alojada la aplicación.
- **PHP-FPM**: Versión de PHP para webs de alto rendimiento. Proporciona al servidor Apache la posibilidad de leer páginas escritas en este lenguaje de programación, creado para el desarrollo de aplicaciones web. El motor PHP-FPM lee el lenguaje PHP y lo traduce a HTML para enviarlo a los navegadores de los usuarios.

- **MySQL 8.0**: Es un sistema de gestión de bases de datos relacional donde se encontraban almacenados todos los datos necesarios para el funcionamiento de la aplicación.
- **VMware ESXi**: Hipervisor de tipo 1, o bare metal, profesional para ejecutar máquinas virtuales
- **SynckBlock**: Aplicación encargada de realizar las copias de seguridad en el servidor de backup.

3.1.6 Copias de seguridad

Se realizaban copias de seguridad del servidor web, del servidor de base de datos y del servidor con Active Directory. Se realizaban copias completas, incrementales y diferenciales que se guardaran en el servidor de backup durante 3 años. Pasado ese tiempo, las copias más antiguas eran reemplazadas por las más actuales.

- Se realizaba una copia incremental cada día de los datos que se hubieran modificado desde la última copia de seguridad incremental o completa en los 3 servidores.
- Se realizaba una copia diferencial una vez a la semana de los datos que se hubieran modificado desde la última copia de seguridad diferencial o completa en los 3 servidores
- Se realizaba una copia completa una vez al mes de todos los datos de los 3 servidores.

3.2 Planificación temporal de las tareas del proyecto

La realización de la aplicación duro un periodo de tiempo de 30 días, desde el 15 de mayo al 15 de junio. La jornada laborar era de 8 horas de lunes a viernes.

Se necesitaron 3 desarrolladores que se encargaron de realizar el código de la aplicación y de las pruebas necesarias en los entornos descritos. Se contrataron dos administradores de sistemas. Uno levanto toda la infraestructura en AWS y realizo la monitorización de esta una vez desplegada la aplicación. Otro administrador de sistemas era el encargado de mantener los equipos en las oficinas y de administrar Windows Server y el Directorio Activo.

Como se ha explicado anteriormente, la cultura DevOps se caracteriza por un flujo de trabajo continuo y cíclico por lo que, llegado a cierto punto del desarrollo, las tareas se solaparon.

3.3 Planificación de los recursos a utilizar

3.4 Herramientas

3.4.1 Docker

Se trata de una tecnología basada en contenedores, gracias a los cuales, todos los entornos por los que pasa la aplicación serán exactamente iguales. Si no se usasen estos contenedores, cada desarrollador tendría en su equipo local diferentes versiones de software o librerías, lo que podría dar lugar a posibles incompatibilidades a la hora de integrar el código en el entorno de producción.

Esto podría resolverse con el uso de máquinas virtuales, pero estas son bastante más pesadas al tener que tener instalado todo un Sistema Operativo junto con el hipervisor. Los contenedores solo contenían las dependencias del SO necesarias para la utilización de la aplicación. Esto dio lugar a imágenes de unos pocos megas, en comparación a las gigas que ocuparía una máquina virtual.

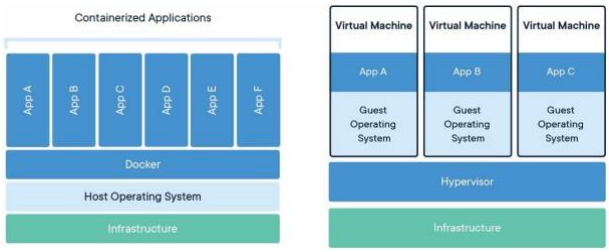

Ilustración 2. Diferencias entre contenedor y máquina virtual.

Estos contenedores se crearon a partir de imágenes, que a su vez se crearon a partir de ficheros, denominados Dockerfiles. En estos ficheros se especifica cada una de las características que formará el contenedor:

- **From:** Imagen a partir de la cual se crea el contenedor.
- **Expose**: Puertos que usa el contenedor
- **ADD**: Directorios del contenedor y de la maquina anfitriona que funciona como volúmenes del contenedor.
- **RUN**: Comandos que se deben ejecutar para la creación del contenedor.

Para el desarrollo de la aplicación se utilizaron imágenes de Alpine Linux, que es una distro de Linux muy ligera, por lo que las imágenes ocuparon pocos megas. Las imágenes también contenían Apache y PHP para el servidor de la aplicación y una imagen de Alpine Linux con MySQL para el servidor de la base de datos.

3.4.2 Docker-Compose

Es una tecnología que se usa para orquestar varios contenedores y crear entornos completos de forma fácil. Para ello se utiliza un archivo docker-compose.yml.

Este tipo de archivos cuenta con varias etiquetas que indican el nombre del servicio, la imagen que se usa para levantar el contenedor y las configuraciones de estos. Algunas de las etiquetas más importantes son:

- **Services**: Bajo esta etiqueta se fueron añadiendo servicios al entorno que vaya a crear.
- **Image**: Indica cual es la imagen que se utilizara para levantar el contenedor.
- **Enviroment**: Son las variables de entorno que se quieran configurar.
- **Ports**: Conecta el puerto de la maquina anfitriona con el puerto expuesto del contenedor.
- **Volumes**: Indica que carpetas de la maquina anfitriona se usara como volumen del contenedor.

3.4.3 Git

Es el software más popular de control de versiones, monitorizando y proporcionando control sobre los cambios en el código fuente. Guarda un registro de las modificaciones que se hacen permitiendo recuperar versiones antiguas del código. Este tipo de herramientas es esencial en un entorno colaborativo como lo es la filosofía de trabajo DevOps.

Este software tiene cierta terminología que es necesario nombrar:

- **Repositorio:** Espacio de Git donde se almacenan las diferentes versiones del código fuente.
- **Rama:** Un mismo código fuente puede ramificarse para que sea editado por dos personas al mismo tiempo de forma independiente.

- **Commit:** Confirmación de un cambio.
- **Push:** Copiar la nueva versión del código en el repositorio de Git.
- **Pull:** Descargar y fusionar los cambios en el repositorio de Git.
- **Merge**: Unificar, fusionar e integrar todos los cambios realizados por los desarrolladores en una nueva versión del código fuente.

3.4.4 Amazon Web Services

3.4.4.1 Amazon Elastic Compute Cloud

Amazon EC2 proporciona capacidad de computación escalable en la nube de Amazon Web Services. Con este tipo de computación, se evita tener que invertir dinero en adquirir más servidores físicos para escalar la infraestructura de una empresa.

Otra gran ventaja de los servidores EC2 es que si, por alguna razón, se necesitase menos capacidad de computación también sería posible escalar EC2 hacia abajo. A fin de cuentas, solo se pagará por los servicios y la computación contratados en cada momento.

Amazon EC2 ofrece las siguientes características:

- Servidores virtuales o instancias.
- Varias configuraciones predefinidas de recursos del servidor. Estos recursos podrían ser CPU, memoria, almacenamiento, etc.
- Volúmenes de almacenamiento que se eliminan cuando una instancia se detiene.
- Un firewall que permite especificar los protocolos, los puertos y los rangos de direcciones IP que pueden alcanzar las instancias mediante el uso de grupos de seguridad

3.4.4.2 Amazon Virtual Private Cloud

Una nube virtual privada (VPC) es una red virtual dedicada de AWS. Esta red se encuentra completamente aislada del resto de VPC de AWS. Dentro de la VPC se pueden configurar diferentes subredes, añadir grupos de seguridad y configurar tablas de ruteo.

La VPC se puede usar en cualquier tipo de instancia de AWS, como EC2, ECS, Load Balancer, etc.

- Una subred es un rango de direcciones IP en su VPC.

- Para proteger los recursos, puede utilizar varias capas de seguridad, como grupos de seguridad y listas de control de acceso a la red (ACL).
- Puede asociar una dirección IPv6 a la VPC.

3.4.4.3 Amazon Elastic Load Balancing

Distribuye automáticamente el tráfico entrante entre varias instancias, puertos, o contenedores. Comprueba el estado de los diferentes destinos para redirigir el tráfico al punto más optimo. Esto aumenta la disponibilidad de la aplicación.

Algunos de los componentes más importantes de un balanceador de carga son:

- **Agente de escucha**: Comprueba las solicitudes de conexión de los clientes mediante el protocolo y el puerto configurados. Las reglas que se definan para un agente de escucha determinan cómo el balanceador de carga va a direccionar las solicitudes a sus destinos registrados. Cuando se cumplen las condiciones de una regla, se llevan a cabo sus acciones.
- **Grupo de destino:** Direcciona las solicitudes a una o varias instancias utilizando el protocolo y el número de puerto que ha especificado. Las comprobaciones de estado se llevan a cabo en todos los destinos registrados en un grupo de destino especificado en la regla del agente de escucha del balanceador de carga.

3.4.4.4 Amazon Elastic Container Services

Es un servicio de administración de contenedores muy escalable y rápido. Se puede utilizar para ejecutar, detener y administrar contenedores en un clúster. Con Amazon ECS, los contenedores se definen en una definición de tareas que utiliza para ejecutar tareas individuales o tareas dentro de un servicio. En este contexto, un servicio es una configuración que puede usar para ejecutar y mantener un número determinado de tareas simultáneamente en un clúster. Las tareas y los servicios se pueden ejecutar en una infraestructura sin servidor administrada por AWS Fargate. Si desea más control sobre su infraestructura, puede ejecutar las tareas y los servicios en un clúster de instancias de Amazon EC2.

Las características principales de Amazon ECS son:

- Orquestación de contenedores administrados de AWS. Permite la creación, configuración y administración de contenedores Docker de manera fácil. Esta integración facilita a los equipos centrarse en crear las aplicaciones, no en el entorno.
- Implementación e integración continuas (CI/CD). Este es un proceso que se utiliza mucho en Dev-Ops y que se basan en contenedores Docker. Se puede crear una canalización de CI/CD que realice las siguientes acciones:
    - Comprobar los cambios en un repositorio fuente como GitHub.
    - Crea imágenes de contenedores Docker usando los archivos del repositorio.
    - Inserta la imagen en Docker Hub.
    - Desplegar las nuevas imágenes en Amazon ECS para actualizar los diferentes servicios.

3.4.4.5 Amazon Fargate

La tecnología AWS Fargate se puede utilizar en Amazon ECS para ejecutar contenedores sin tener que administrar servidores ni clústeres de instancias de Amazon EC2. De esta manera, se elimina la necesidad de elegir tipos de servidores, decidir cuándo escalar los clústeres u optimizar conjuntos de clústeres.

Al ejecutar las tareas y los servicios de Amazon ECS con el tipo de lanzamiento de Fargate, la aplicación se empaqueta en contenedores, se especifican los requisitos de sistema operativo, CPU y de memoria, se definen las políticas de IAM y las redes, y se lanza la aplicación. Cada tarea de Fargate tiene su propio límite de aislamiento y no comparte el kernel subyacente, los recursos de CPU, los recursos de memoria ni la interfaz de red elástica con otra tarea.

3.4.4.6 Amazon Codepipeline

Es un servicio de entrega continua que se utiliza para configurar los pasos necesarios para desplegar aplicaciones y servicios en diferentes instancias de AWS. Se pueden diseñar de forma simple las etapas por las que pasa el código de una aplicación, desde el repositorio hasta la implementación, pasando por la compilación del código de manera continua.

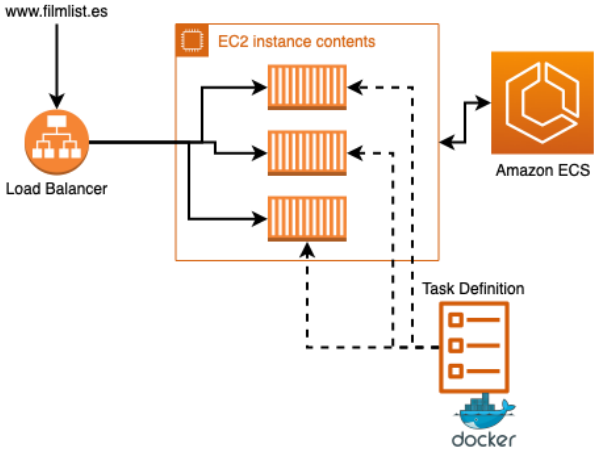

Ilustración 3. Esquema de infraestructura en AWS.

4 METODOLOGÍA Y DESARROLLO DEL PROYECTO

La aplicación, a grandes rasgos, cuenta con un servidor web basado en Apache2 con la extensión de PHP instalada. De esta manera, la web pudo ser creada en lenguaje PHP.

También se usan tecnologías como JavaScript, para el formulario, o HTML para la creación de la página web estática.

Por último, se usa una base de datos MySQL para la creación de todas las bases de datos necesarias, como las de usuarios, colecciones o películas.

Todo esto estará alojado en un servidor de Amazon Web Services que se encontrará en la nube, diseñado para cumplir alta disponibilidad y a prueba de fallos.

4.1 Estructura de la aplicación

Se trata de una aplicación web con un diseño atractivo. En su página de inicio debe aparecer una lista de películas recientes y el formulario de inicio de sesión.

Una vez logueado, el usuario ve su propia colección ordenada por orden alfabético. Tiene un buscador de películas para poder añadir más a la colección.

4.2 Componentes del sistema

- **Apache2 y PHP**: Corren en un contenedor de Docker. Este contenedor está formado por una imagen de Alpine Linux, Apache2 y PHP. Todo este contenedor está configurado gracias a un Dockerfile que se despliega en un clúster de Amazon ECS.
- **JavaScript**: Necesario para el formulario de registro e inicio de sesión.

- **HTML**: Toda la web estática esta creada en HTML para que los navegadores de los usuarios puedan cargar las páginas.
- **MySQL**: Corre en un contenedor de Docker. Este contenedor está formado por una imagen de Alpine Linux y MySQL. Está configurado en un archivo Dockerfile que se despliega en un clúster de Amazon.
- **Amazon Web Services**: Servidor en la nube que aloja el clúster de Amazon ECS donde están alojados todos los microservicios necesarios para que la página web funcione.

4.3 Arquitectura de la red

4.3.1 Servidores físicos

Estos servidores son los que se encuentran actualmente en el CPD de la empresa. Son reutilizados aprovechando su disponibilidad.

4.3.1.1 Servidor Windows Server

Se utilizará como servidor DNS, Servidor de DHCP y como servidor de Active Directory. Todos esto se hará gracias a la licencia de Windows server 2019 que la empresa tiene ahora mismo.

4.3.1.2 Servidor de Backup

En él se instalará Windows Server 2019 y la aplicación SynckBlock para realizar copias de seguridad de todos los servidores de la empresa.

También se duplicarán los servicios del controlador de dominio para tener redundancia del mismo en caso de fallo del primer servidor.

4.3.2 Servicios Amazon Web Services

Amazon Web Services, o AWS, es un conjunto de herramientas y servicios de computación en la nube creado por Amazon. Su lanzamiento oficial fue en 2006 y desde entonces se ha posicionado como el líder de los servidores y servicios basados en la nube. Su gran madurez y el gran abanico de características y herramientas disponibles, además de su menor coste, ha sido clave para elegir este servicio frente a otros muy similares.

De todos los servicios que ofrece AWS, se han usado los servidores Amazon Elastic Compute Cloud y Amazon Fargate. Proporciona un control completo sobre los recursos y permite configurar sus requisitos de una manera muy rápida y sencilla.

4.3.2.1 Integración continua

La integración continua se realiza usando el servicio Codepipeline que ofrece AWS. Se creo una tubería que está formada por tres etapas:

- **Source**: Automáticamente reconoce cualquier cambio que se realice en el repositorio GIT.
- **Build**: Después, crea las imágenes de Docker necesarias para correr la aplicación con el nuevo código ya implementado.
- **Deploy**: Una vez finalizada la etapa anterior, se despliegan los contenedores en el clúster de ECS, tanto en el de entorno de integración como en el de producción.

4.3.2.2 Servidor de entorno de Integración

Se utilizo una de las instancias EC2 de AWS para crear este servidor que es idéntico al de producción. Su función es el de realizar pruebas de estrés y capacidad de computación de la aplicación antes de pasar, finalmente, al servidor de producción.

En él, se creó un clúster de ECS, con los microservicios de Apache, PHP y MySQL, con un balanceador de carga para asegurar la alta disponibilidad.

4.3.2.3 Servidor de entorno de Producción

Este servidor es el destino final de la aplicación web. A este servidor es donde realizan las peticiones los usuarios para poder acceder al servicio.

Estará formado por una instancia EC2 con un clúster de Elastic Container Service, formado por los microservicios de Apache, PHP y MySQL. Este clúster asegura la alta disponibilidad y el rápido acceso a la web, gracias al balanceo de carga que ofrece ECS entre sus tareas.

4.3.3 Dispositivos de red

4.3.3.1 Dispositivos físicos

- **Router:** Se mantiene la configuración actual del router del CPD de la empresa.

- **Switch:** Se mantiene el FortiSwitch de la empresa, pero se varían las VLANs del mismo:
    - VLAN de Administración: Esta red es la utilizada para administrar todos los equipos de la red, como servidores, firewall, switchs y puntos de acceso.
    - VLAN Web: Desde esta red se puede acceder a los recursos de los servidores alojados en AWS.
    - VLAN Usuarios: Es la red utilizada por los usuarios de la empresa.
    - VLAN Visitantes: Es la red pública para las visitas en la oficina de la empresa.
- **Firewall:** Todos los switches de la empresa están conectados a cada una las interfaces del firewall. El firewall también es el encargado de gestionar la política de seguridad de la empresa, aplicando reglas para permitir o denegar el acceso a las diferentes VLANs e internet.

  Además, se crea un túnel Ipsec VPN, para conectar la red física de la empresa con la red virtual de AWS.

4.3.3.2 Dispositivos en la nube

Amazon proporciona una red virtual para conectar las diferentes instancias llamado Amazon Virtual Private Cloud (VPC). Funciona igual que una red física, pero tiene la ventaja que hace uso de la capacidad de escalabilidad de AWS.

Se usa este servicio para sustituir los dispositivos de red físicos que la empresa usaba para los servidores de sus aplicaciones web.

- **Red:** Se usa una red privada IPv4 /24, lo que dará un total de 255 direcciones IP privadas.
- **Instancias:** Se configuran las interfaces de red de cada una de las instancias EC2 con una IP de las 255 disponibles.
- **Grupo de seguridad:** Se configuran dos grupos de seguridad. Uno para el balanceador de carga, con una regla por la que solo acepta el paso de tráfico por el puerto 80 desde cualquier IP de internet. Y otro para las tareas de ECS, con una regla por la que solo acepta tráfico por el puerto 80 que proceda del balanceador de carga. Estos grupos funcionan como firewalls.

4.4 Entorno de desarrollo

Este entorno es el que usen cada uno de los desarrolladores de la aplicación para probar todos los cambios que vayan realizando. Tiene que ser un entorno exactamente igual en cada uno de los equipos para que a la hora de implementar los cambios en la versión final no haya fallos.

Con el fin de conseguir este tipo de entornos controlados se usó Docker y Docker-Compose para levantar todos los servicios necesarios. Una vez creados los archivos Dockerfile y docker-compose.yml necesarios, se subió al repositorio de GitHub para que todos los desarrolladores puedan clonarlo en su máquina e implementarlo.

Para la realización de este proyecto se creó una máquina virtual con Ubuntu Desktop 20.04 con el fin de simular un equipo de un desarrollador de la empresa.

Para crear el entorno deseado se han seguido los siguientes pasos:

1. Se creo un archivo llamado Dockerfile en la carpeta donde se encuentre todo el código de la aplicación. (**Anexo A**)
2. Con Visual Studio Code se editó el archivo indicando toda la configuración necesaria para crear la imagen de Docker.
    a. **FROM: php:7.2-apache** → Esta imagen estaba formada por un servidor web Apache2 con PHP 7.2 ya instalado.
    b. **ADD ./files /var/www/html/** → Con esta etiqueta se indicó que se van a copiar los archivos de la carpeta “file” en la ruta “/var/www/html/” del contenedor. De esta manera el servidor Apache puede acceder a ellos para hacer funcionar la página web.
    c. **RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf && a2enmod rewrite && service apache2 restart** → Se ejecutan los comandos dentro del contenedor.
    d. **EXPOSE 80** → Se expone el puerto 80 HTTP para que funcione la aplicación.
3. Después se creó el archivo docker-compose.yml. (**Anexo B**)
    a. **Services**: Se creo un servidor web, una base de datos MySQL y phpMyAdmin para administrar la base de datos.

b. **Image**: Se usaron las imágenes de Docker Hub de MySQL y phpMyAdmin. Para el servidor web, se cambió la etiqueta Image por Build para indicar que la imagen se crea a partir del archivo dockerfile de la carpeta raíz.

c. **Ports**: Se conectaron los puertos 80 del host y del servidor web, el puerto 3306 de host y contenedor de base de datos, y el puerto 8080 y el 80 del contenedor de phpMyAdmin.

d. **Volumes**: Se conecto la carpeta "./file" del host con "/var/www/html/" de Apache.

4. Para terminar de crear el entorno de desarrollo hubo que levantar todos los contenedores.

```
docker-compose up -d
```

Para ver todos los contenedores que están corriendo sé que ejecuto:

```
sudo docker ps
```

5. Se accedió desde un navegador al localhost:80 y apareció la aplicación web.

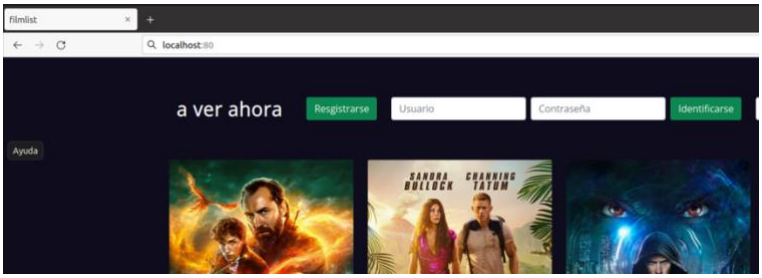

Ilustración 4. Aplicación en entorno de desarrollo.

6. Para finalizar se instaló Git en la consola .

```
sudo apt-get install git
```

4.5 Entorno de producción

En un ambiente profesional se crearían un entorno de integración y otro de producción, con el fin de realizar pruebas al código antes de desplegar definitivamente el código. Debido a las limitaciones de la capa gratuita de Amazon Web Service y, sobre todo, a que ambos servidores son exactamente idénticos, se ha decidido crear directamente el entorno de producción.

Este entorno de producción tiene que ser idéntico al entorno de desarrollo para que no se produzcan incompatibilidades o fallos en la aplicación. Para ello se usarían las mismas imágenes y las mismas configuraciones de los contenedores levantado en el entorno de desarrollo. La única diferencia es que no se usaría Docker-Compose, si no Amazon Elasctic Conteiner Service.

En los siguientes puntos se describe como se levantó el entorno de producción.

4.5.1 Grupos de seguridad

Para poder levantar el servicio lo primero que hubo que hacer es crear dos grupos de seguridad, uno para el balanceador de carga y otro para las tareas de ECS.

Las opciones que hay que se modificaron del grupo de seguridad para el balanceador de carga están definidas en la siguiente tabla. El resto de opciones se dejaron por defecto.

Nombre de ajuste	Configuración	
Nombre	ABL-filmlist-sg	
VPC	Default-VPC	
Regla de entrada	Tipo	HTTP
	Origen	0.0.0.0/0

Tabla 2. Configuration Security Group de Load Balancer.

Las opciones para el grupo de seguridad que se usaron en las tareas de ECS están indicadas en la siguiente tabla. El resto de opciones se dejaron por defecto.

Nombre de ajuste	Configuración		
Nombre	task-filmlist-sg		
VPC	Default-VPC		
Regla de entrada	1	Tipo	HTTP
		Origen	ABL-filmlist-sg
	2	Tipo	TCP
		Puerto	8080
		Origen	ABL-filmlist-sg

Tabla 3. Configuración Security Group instancia EC2.

Con esta configuración, el tráfico pasa primero por el balanceador de carga que recibe peticiones de cualquier IP y lo distribuye a los contenedores del clúster ECS por los puertos 80 y 8080. Estos contenedores están aislados de internet y solo pueden recibir el tráfico

que provenga del balanceador de carga. Esta configuración aumenta la seguridad de la aplicación web.

4.5.2 Grupos de destino

Antes del balanceador de carga, se crearon dos grupos de destino que apuntan a las tareas de ECS que se crearon. De este modo, cuando el tráfico entra por el puerto 80 del balanceador, este lo redirige a los grupos de destino creados que apuntan al puerto 80 y 8080 de las tareas de ECS.

La configuración del primer grupo de destino es la siguiente:

Nombre de ajuste	Configuración
Target type	IP
Name	grupo80
Protocolo	HTTP
Puerto	80
VPC	Default-VPC

Tabla 4. Configuración grupo de destino puerto 80.

La configuración del segundo grupo de destino es:

Nombre de ajuste	Configuración
Target type	IP
Name	grupo8080
Protocolo	HTTP
Puerto	8080
VPC	Default-VPC

Tabla 5. Configuración grupo de destino puerto 8080.

Una vez que la instancia este creada, se asociaran los grupos de destino con las tareas del clúster ECS.

4.5.3 Balanceador de Carga

Después, se creó un Balanceador de Carga de Aplicación en el panel de EC2. La configuración se describe en la siguiente tabla:

Nombre de ajuste	Configuración		
Nombre	ABL-Filmlist		
VPC	Default-VPC		
Mappings	us-east-1a us-east-1b		
Security Groups	ABL-filmlist-sg		
Listener and routing	1	Protocol	HTTP
		Port	80
		Grupo de destino	grupo80
	2	Protocol	HTTP
		Port	8080
		Grupo de destino	grupo8080

Tabla 6. Configuración Load Balancer.

Con esta configuración, el balanceador se encarga de alternar entre los puertos 80 y 8080 cuando se realiza el despliegue automático de los contenedores con el código actualizado.

4.5.4 Clúster ECS

Para poder levantar los diferentes contenedores primero se tuvo que crear un clúster donde se ejecutan. Este clúster está asociado a una instancia de EC2, que es donde se realiza todo el procesamiento necesario para ejecutar la aplicación web.

El clúster se creó en la consola de Elastic Conteiner Service y tiene la siguiente configuración:

Nombre de ajuste	Configuración
Clúster template	EC2 Linux + Networking
Clúster name	Filmlist
EC2 instance type	t3.micro
Number of instance	1
VPC	Default-VPC
Subnet	us-east-1a us-east-1b
Security group	task-filmlist-sg

Tabla 7. Configuración clúster ECS.

Una vez se cree el clúster se podrá interactuar con él, creando servicios y tareas.

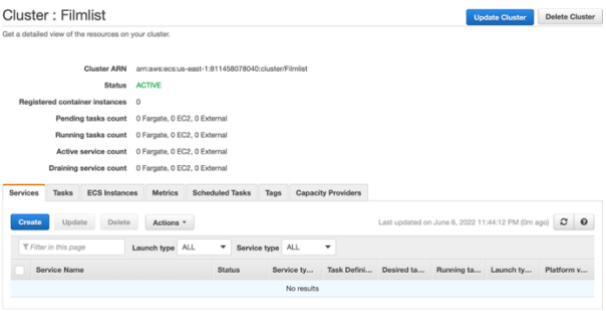

Ilustración 5. Clúster de Amazon ECS

Hay que observar que, junto al clúster, se creó una instancia EC2 donde se despliegan todos los contenedores y esta alojada la aplicación web.

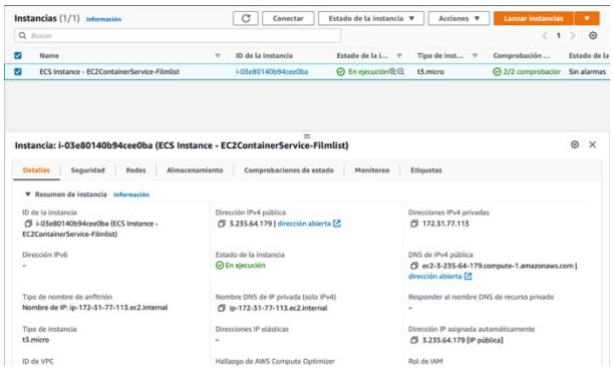

Ilustración 6. Instancia EC2 creada con clúster.

4.5.5 Task Definition

Una definición de tarea contiene toda la configuración de los contenedores que se van a levantar. Realiza las mismas tareas que Docker-Compose pero en los servidores de Amazon Web Service.

A continuación, se describen las configuraciones necesarias para levantar la aplicación:

Nombre de ajuste	Configuración
Tipo de lanzamiento	EC2
Nombre	Filmlist-Entorno
Task memory	1 GB
Task CPU	1 vCPU
Add container	• Se añaden las imágenes de los 3 contenedores que se necesitan. • Hard-limit → 500

	• Soft-limit → 300 • CPU units → 300 • Puertos → Los mismo que se definieron en el docker-compose.yml

Tabla 8. Configuración Task Definition.

Esta definición de tarea se puede cambiar, creando nuevas versiones con cada cambio que se realiza. Se creó un archivo JSON con todas las configuraciones de la Task Definition. (**Anexo C**)

4.5.6 Servicio

Para crear el servicio se entró dentro del clúster y seleccionó “créate” en la pestaña de Servicios.

La configuración del servicio es la siguiente:

Nombre de ajuste	Configuración
Launch Type	EC2
Task Definition	Filmlist-Entorno
Clúster	Filmlist
Service Name	Filmlist-Service
Number of task	2
Load Balancing	Aplication Load Balancer
Load Balancer name	ABL-Filmlist
Container to Load Balance	app:80:80 → Add to Load Balancer
Production Listener	80:HTTP
Target Group	Grupo80

Tabla 9. Configuración servicio ECS.

4.6 CodePipeline

Se creó una canalización para asegurar la integración continua y el despliegue continuo del nuevo código de la aplicación. A este proceso se le conoce como CI/CD.

Esta canalización se activa automáticamente cuando algún desarrollador realiza un “Push” en el repositorio de GitHub para subir nuevo código. Lee el nuevo código y pasa a la fase de “compilación” donde se crean las nuevas imágenes de los contenedores con los cambios ya integrados. Para finalizar, estos contenedores se despliegan en el clúster de

ECS y los antiguos son eliminados. Gracias al balanceador de carga este proceso es transparente para el usuario.

4.6.1 Etapa Source

Se conectó la tubería con el repositorio de GitHub para que esta pueda detectar los cambios realizados. Las configuraciones necesarias para esta etapa son las siguientes:

Nombre de ajuste	Configuración
Nombre de la canalización	Filmlist
Proveedor de origen	• GitHub • Conectarse a GitHub • Introducir credenciales de GitHub
Repositorio de GitHub	KataFurius/filmlist
Ramificación	Main

Tabla 10. Configuración etapa source.

4.6.2 Etapa Compilación

Aquí se construyen las imágenes necesarias de la aplicación con los cambios integrados, se etiquetan con el nombre de la última versión y se suben estas imágenes al repositorio de Docker Hub. Por último, se incluye, en el archivo de la definición de tarea del servicio que tenemos creado en ECS, el nombre y versión de las imágenes nuevas.

Las configuraciones para esta etapa son las siguientes:

Nombre de ajuste	Configuración
Proveedor de compilación	AWS CodeBuild
Nombre del proyecto	Crear proyecto
Nombre del proyecto	Filmlist-build
Imagen de entorno	Imagen administrada
Sistema Operativo	Ubuntu
Tiempo de ejecución	Standard
imagen	aws/codebuild/estándar:5.0
Especificaciones de la compilación	Utilizar un archivo de especificación de compilación. (**Anexo D**)
Tipo de compilación	Compilación única

Tabla 11. Configuración etapa compilación.

El resto de opciones se dejaron por defecto.

4.6.3 Etapa de implementación

Esta es la etapa final de la tubería y en ella se despliegan los contenedores en el clúster de Amazon Elastic Container Service, siguiendo la definición de tarea que se creó anteriormente.

Las configuraciones necesarias se indican en la siguiente tabla:

Nombre de ajuste	Configuración
Proveedor de la implementación	Amazon ECS
Nombre del clúster	Filmlist
Nombre del servicio	Filmlist-Service

Tabla 12. Configuración etapa implementación.

5 RESULTADOS Y DISCUSIONES

Son muchas las pruebas que pueden realizarse en un proyecto para eliminar los posibles errores y garantizar su correcto funcionamiento. Los casos de prueba establecen las condiciones/variables que permitirán determinar si los requisitos establecidos se cumplen o no.

A continuación, se detallan algunos de los casos de prueba que se ejecutaron para comprobar la correcta construcción de este proyecto.

5.1 Casos de prueba

5.1.1 En entorno de desarrollo

Una vez levantado todo el entorno se realizaron cambios en el código, usando Visual Studio Code. Los cambios que se hicieron se deberían ver reflejados automáticamente en la aplicación web gracias, a que se enlazo la carpeta con los archivos y el directorio de la aplicación en el contenedor.

5.1.2 Funcionamiento del servicio en producción

Una vez creado el clúster en Amazon ECS y creado el servicio de la aplicación web, se accedió al nombre de DNS público del balanceador de carga. Se uso el puerto 80, ya que este fue el que se configuro en la definición de tarea.

5.1.3 Funcionamiento de la CI/CD

Se comprobó que la canalización de integración continua, que creamos con Codepipeline, funcionaba perfectamente.

- Se realiza un cambio en el código de la aplicación dentro del entorno de desarrollo.
- Se realizó un “git push” para subir los cambios a GitHub.
- En la consola de Codepipeline, se accedió a la tubería de la aplicación.
- La etapa de Source tuvo que haberse disparado automáticamente.
- Una vez finalizada, empezó la etapa de Compilación
- Después, le siguió la etapa de Implementación.
- Una vez terminadas todas las etapas, se accedió al nombre de DNS del balanceador de carga por el puerto 80 y los cambios deberían de haberse implementado sin problemas.

5.2 Resultados obtenidos

En este apartado se comentan los resultados obtenidos al finalizar cada una de las pruebas que se han realizado para verificar la viabilidad del proyecto.

5.2.1 Entorno de desarrollo

En las siguientes imágenes se puede ver como el título de la página es “prueba”, tanto en el código como en la propia aplicación.

Ilustración 7. Código de aplicación.

Ilustración 8. Aplicación en localhost.

En la siguiente se editó el titulo por “filmlist” y al actualizar el navegador el título de la aplicación cambió.

Ilustración 9. Cambio en código fuente.

Ilustración 10. Cambio hecho en localhost.

Una vez que los cambios son correctos se subieron al repositorio de Github. De esta manera el código estará siempre actualizado y accesible para el resto de los desarrolladores.

5.2.2 Funcionamiento del servicio en producción

Una vez accedido a la URL publica que facilita el balanceador de carga, se observó como la aplicación se está ejecutando correctamente y sin fallos.

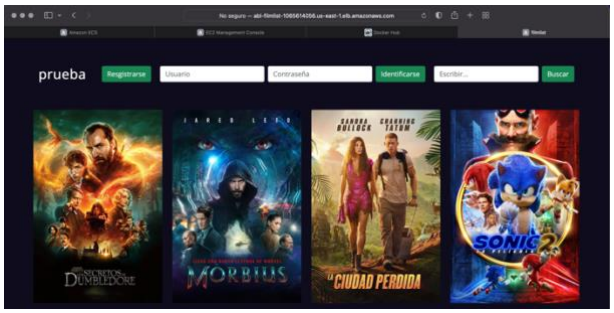

Ilustración 11. Prueba funcionamiento entorno producción

Al recargar la página muchas veces, esta no falló gracias al balanceador de carga que repartió la carga entre varios contenedores.

5.2.3 Funcionamiento de la CI/CD

A la hora de probar la canalización de integración continua fue imposible que funcionara. Cada vez que se realizaba un cambio en el código, la etapa de “Source” se iniciaba correctamente. Después, en la etapa de “Compilación”, se creaban las nuevas imágenes y se subían al repositorio de Docker Hub sin problemas. Donde siempre se obtuvo un error, que impedía finalizar la canalización, fue en la etapa de “Implementación”.

Cada vez que se intentaba realizar la implementación, el clúster devolvía el mismo fallo. Este consistía en que el balanceador de carga, por alguna razón, no era capaz de cambiar el puerto de los contenedores cuando había más de uno desplegado a la vez. De esta

manera, el puerto 80 del balanceador nunca era accesible por el nuevo contenedor porque ya estaba ocupado por el antiguo.

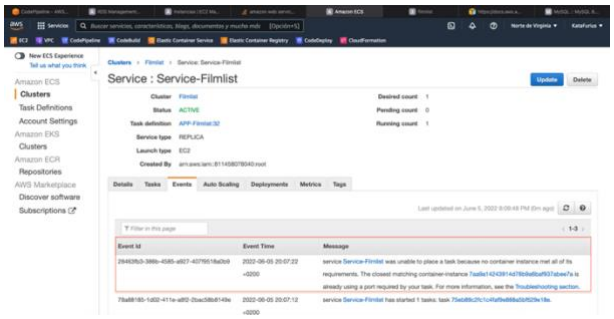

Ilustración 12. Fallo en la implementación.

Se eliminó el clúster, la definición de tareas, el balanceador de carga y la canalización. Es decir, se reinició todo el proyecto una y otra vez con el mismo resultado. También se estudiaron otras opciones que ofrece Amazon Web Service, pero ninguna presenta la facilidad de uso con contenedores que si tiene Amazon Elastic Container Service.

Se llegó a la conclusión que el problema estaba en la etapa de implementación de canalización. Esto es debido a que, si se levantaban los contenedores de forma manual, una vez subidas las imágenes necesarias, y recién construidas, a Docker Hub, el servicio funcionaba perfectamente y con todos los cambios visibles.

Para no dejar el proyecto sin acabar, y con mal sabor de boca, se decidió usar la misma metodología en el entorno de producción que se utilizó en el entorno de desarrollo. Esa solución es Docker-Compose.

AWS no ofrece ningún servicio capaz de levantar contenedores orquestados con Docker-Compose. Para salvar este escollo, se decidió crear una instancia EC2 fuera del clúster de ECS e instalar ahí Docker-Compose y Git.

De manera paralela, se creó un simple script en Bash que permite tirar abajo los contenedores del entorno y borrar la carpeta donde está alojada la aplicación y el archivo Dockerfile necesario para crear la imagen del contenedor de la misma.

El script clona el repositorio en la misma carpeta donde estaba anteriormente y ejecuta el comando “docker-compose up” para levantar el entorno de nuevo, esta vez con todos los cambios implementados. (**Anexo E**)

Se intento hacer una última prueba antes de dar por perdida la integración continua usando CodePipeline y se dio con la solución.

Al crear el servicio en el clúster de ECS se decidió cambiar el tipo de lanzamiento de EC2 a Fargate, un servicio de Amazon que permite subir contenedores sin necesidad de crear instancias EC2. Sorprendentemente todo funcionó a la primera y sin ningún problema.

Al pasar la canalización a la etapa de implementación, en el servicio Fargate de ECS se crearon dos nuevas tareas. Estas tareas se mantuvieron en ejecución junto con las dos tareas que ya estaban corriendo anteriormente. Pasado un tiempo, las dos tareas antiguas dejaron de funcionar y las dos tareas nuevas pasaron a funcionar por si solas, con los cambios realizados visibles en la página web.

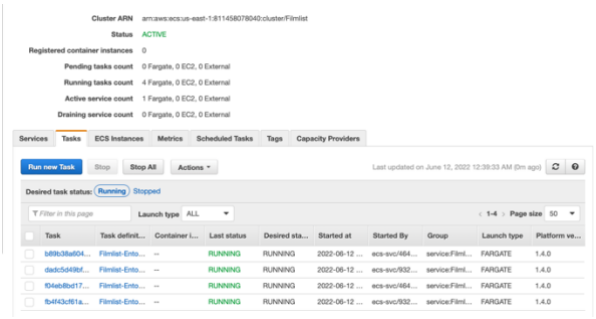

Ilustración 13. Cuatro tareas desplegadas.

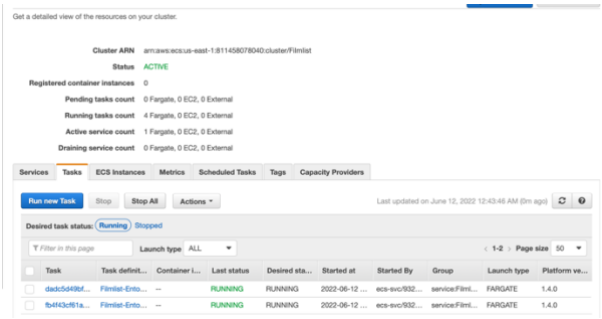

Ilustración 14. Dos tareas desplegadas.

Se tuvo que crear un clúster basado en Fargate para que la definición de tarea, también basada en Fargate, funcionase correctamente. A continuación, se describen las configuraciones que realizaron de nuevo para el clúster, la definición de tarea y la creación de servicios.

Nombre de ajuste	Configuración
Clúster template	Networking Only
Clúster name	Filmlist

Ilustración 15. Nueva configuración de clúster ECS.

Nombre de ajuste	Configuración
Tipo de lanzamiento	Fargate
Nombre	Filmlist-Entorno
Operating System Family	Linux
Task memory	2 GB
Task CPU	1 vCPU
Add container	• Se añaden las imágenes de los 3 contenedores que se necesitan. • Puertos → Los mismo que se definieron en el docker-compose.yml

Ilustración 16. Nueva configuración para Task Definition.

Nombre de ajuste	Configuración
Launch Type	Fargate
Operating System Family	Linux
Task Definition	Filmlist-Entorno
Clúster	Filmlist
Service Name	Filmlist-Service
Number of task	2
Clúster VPC	Default
Security Group Task	task-sg
Auto-assign public IP	Enabled
Load Balancing	Aplication Load Balancer
Load Balancer name	ABL-Filmlist
Container to Load Balance	app:80:80 → Add to Load Balancer
Production Listener	80:HTTP
Target Group	Grupo80

Ilustración 17. Nueva configuración para servicio ECS.

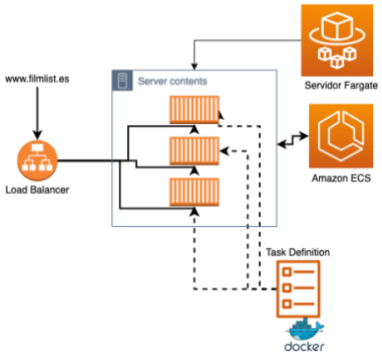

Ilustración 18. Esquema del nuevo entorno de producción.

6 CONCLUSIONES

A pesar de haber creado todos los entornos de forma fácil y rápida, el proceso de integración continua fue imposible de implementar. Una de las bases de la filosofía Dev-Ops es la automatización de tareas para ayudar a los desarrolladores y técnicos de infraestructura. En este proyecto, ese proceso de automatización se perdió durante gran parte del mismo, por lo que la cadena cíclica de Dev-Ops se rompía.

A pesar de estos problemas, se ha intentado buscar una solución lo más cercana posible a lo que se usaría en un entorno Dev-Ops. Al final, se intentó buscar una solución parecida, que consistía en ejecutar un solo comando. Unos días antes de la fecha de entrega se consiguió que toda la automatización funcionase a la perfección. Después del tiempo invertido buscando información para arreglar el problema, haber llegado por mí mismo a la solución del script me ha ayudado a darme cuenta de todo lo que he aprendido durante el curso. Pero, sobre todo, me ha ayudado a ser perseverante, a no dar nunca un problema por perdido y a usar el pensamiento lateral hasta conseguir solucionar el problema de la integración continua.

Mirar el problema desde otro ángulo y buscar soluciones un tanto diferentes a lo planteado en manuales y webs, ha sido la clave para no rendirme y poder completar todo lo que quería sin necesidad de ningún script.

7 BIBLIOGRAFIA

AWS CodeDeploy: Guía del usuario

- Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

web. https://docs.aws.amazon.com/es_es/es_es/codedeploy/latest/userguide/codedeploy-user.pdf#tutorials

- Amazon Elastic Compute Cloud: User Guide for Linux Instances

Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

web. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf

- Amazon Elastic Container Service: Guía para desarrolladores

web. https://docs.aws.amazon.com/es_es/AmazonECS/latest/developerguide/ecs-dg.pdf

- AWS CodePipeline: Guía del usuario

web. https://docs.aws.amazon.com/es_es/codepipeline/latest/userguide/codepipeline-user.pdf

- Elastic Load Balancing: Application Load Balancers

web. https://docs.aws.amazon.com/es_es/es_es/elasticloadbalancing/latest/application/elb-ag.pdf#introduction

8 ANEXOS

Anexo A. Archivo Dockerfile de la aplicación.

Anexo B. Archivo Docker-Compose del entorno.

Anexo C. Archivo de Taks Definition.

Anexo D. Archivo de la etapa de compilación.

Anexo E. Script de despliegue de entorno.

Anexo A. Archivo Dockerfile de la aplicación.

```
FROM php:7.2-apache

ADD ./files /var/www/html/

RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf && a2enmod rewrite && service apache2 restart

EXPOSE 80
```

Anexo B. Archivo Docker-Compose del entorno.

```
version: '3'

services:
  mysql:
    image: mysql:5.6
    container_name: mysql
    environment:
      MYSQL_DATABASE: filmlist
      MYSQL_USER: filmlist
      MYSQL_PASSWORD: qwerty123
      MYSQL ROOT PASSWORD: qwerty1234
    ports:
      - "3306:3306"
    restart: always

  web:
    build: .
    ports:
      - "80:80"
    volumes:
      - ./files:/var/www/html
    links:
      - mysql

  phpmyadmin:
    image: phpmyadmin
    container_name: phpmyadmin
    restart: always
    ports:
      - 8080:80
    environment:
      - PMA ARBITRARY=1
```

Anexo C. Archivo de Taks Definition.


```
{
    "ipcMode": null,
    "executionRoleArn": "arn:aws:iam::811458078040:role/ecsTaskExecutionRole",
    "containerDefinitions": [
      {
        "dnsSearchDomains": null,
        "environmentFiles": null,
        "logConfiguration": null,
        "entryPoint": null,
        "portMappings": [
          {
            "hostPort": 80,
            "protocol": "tcp",
            "containerPort": 80
          }
        ],
        "command": null,
        "linuxParameters": null,
        "cpu": 0,
        "environment": [],
        "resourceRequirements": null,
        "ulimits": null,
        "dnsServers": null,
        "mountPoints": [],
        "workingDirectory": null,
        "secrets": null,
        "dockerSecurityOptions": null,
        "memory": null,
        "memoryReservation": null,
        "volumesFrom": [],
        "stopTimeout": null,
        "image": "katafurius/filmlist:latest",
        "startTimeout": null,
        "firelensConfiguration": null,
        "dependsOn": null,
        "disableNetworking": null,
        "interactive": null,
        "healthCheck": null,
        "essential": true,
        "links": null,
        "hostname": null,
        "extraHosts": null,
        "pseudoTerminal": null,
        "user": null,
        "readonlyRootFilesystem": false,
        "dockerLabels": null,
        "systemControls": null,
        "privileged": null,
        "name": "APP-FILMLIST"
      }
    ],
    "placementConstraints": [],
    "memory": "128",
    "taskRoleArn": null,
```

```
      "compatibilities": [
        "EXTERNAL",
        "EC2"
      ],
      "taskDefinitionArn": "arn:aws:ecs:us-east-1:811458078040:task-definition/APP-Filmlist:15",
      "family": "APP-Filmlist",
      "requiresAttributes": [],
      "pidMode": null,
      "requiresCompatibilities": [
        "EC2"
      ],
      "networkMode": null,
      "runtimePlatform": null,
      "cpu": "1024",
      "revision": 15,
      "status": "ACTIVE",
      "inferenceAccelerators": null,
      "proxyConfiguration": null,
      "volumes": []
    }
```

Anexo D. Archivo de la etapa de compilación.

```
version: 0.2
phases:
  pre_build:
    commands:
      - echo Logging in to Docker Hub...
      - docker login -u katafurius -p Iniesta080116
      - REPOSITORY_URI=katafurius/filmlist
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"APP-FILMLIST","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
artifacts:
  files: imagedefinitions.jsonversion: 0.2
phases:
  pre_build:
```

```
      commands:
        - echo Logging in to Docker Hub...
        - docker login -u katafurius -p Iniesta080116
        - REPOSITORY_URI=katafurius/filmlist
        - IMAGE_TAG=${COMMIT_HASH:=latest}
    build:
      commands:
        - echo Build started on `date`
        - echo Building the Docker image...
        - docker build -t $REPOSITORY_URI:latest .
        - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
    post_build:
      commands:
        - echo Build completed on `date`
        - echo Pushing the Docker images...
        - docker push $REPOSITORY_URI:latest
        - docker push $REPOSITORY_URI:$IMAGE_TAG
        - echo Writing image definitions file...
        - printf '[{"name":"FILMLIST","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
artifacts:
    files: imagedefinitions.json
```

Anexo E. Script de despliegue de entorno.

```
#/bin/bash

cd /home/ubuntu/filmlist

sudo docker-compose down

cd /home/ubuntu

rm -rf /home/ubuntu/filmlist

git clone https://github.com/KataFurius/filmlist.git

cd /home/ubuntu/filmlist

sudo docker-compose up -d
```