2021-2022

Análisis técnico para la modernización y migración y su implementación a servicios cloud con AWS

TRABAJO DE FIN DE GRADO - GRADO DE INGENIERÍA INFORMÁTICA
AUTOR: JUAN CARLOS NADAL CASTELL

TUTOR: MANUEL JESÚS MENDOZA FLORES
ÁREA: ADMINISTRACIÓN DE REDES Y SISTEMAS OPERATIVOS

ENTREGA: JUNIO 2022

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Resumen

El trabajo propuesto en este TFG consiste en establecer un plan de empresa para la modernización y migración de los servicios de reservas y sistemas de gestión de disponibilidad de una empresa hotelera, a un sistema escalable sobre Kubernetes en Amazon Web Services usando los servicios que proporciona AWS.

El plan de migración comprende distintos escenarios que existen en la empresa y que deben migrarse coordinadamente ya que se interrelacionan unos con otros, esto supone migrar tanto las aplicaciones como las bases de datos y todos los elementos y servicios que puedan depender de dichas aplicaciones, intentando minimizar la pérdida de servicio de ninguno de los servicios críticos.

Se analizará en este TFG el motivo de la migración, que comprende el sistema de reservas de la empresa y todos sus servicios satelitales, y las distintas soluciones que se plantearon como alternativa, en las que se sopesarán los parámetros como la escalabilidad, simplicidad de la solución, costes e integración con los servicios existentes que no se migrarán, la mayoría por estar externalizados y se analizarán también los sistemas de monitorización y gestión de alarmas de las distintas soluciones.

Finalmente se analizará la implantación de la solución, planificación de la migración, los problemas y errores que han aparecido durante el proceso y cómo se han solucionado o se han tomado soluciones parciales.

Abstract

The work proposed in this TFM consists of establishing a business plan for the modernization and migration of the reservation services and availability management systems of a hotel company, to a scalable system on Kubernetes in Amazon Web Services using the services it provides AWS.

The migration plan includes different scenarios that exist in the company and that must be migrated in coordination since they are interrelated with each other, this means migrating both the applications and the databases and all the elements and services that may depend on said applications, trying minimize the loss of service of any of the critical services.

In this TFG, the reason for the migration will be analyzed, which includes the company's reservation system and all its satellite services, and the different solutions that were proposed as an alternative, in which parameters such as scalability, simplicity of solution, costs and integration with existing services that will not be migrated, most of them because they are outsourced, and the monitoring and alarm management systems of the different solutions will also be analysed.

Finally, the implementation of the solution, migration planning, the problems and errors that have appeared during the process and how they have been solved or partial solutions have been taken will be analyzed.

Índice

0.0 Licencia 2
0.1 Resumen 3
0.2 Abstract 4
0.3 Índice 5
0.4. Índice de figuras 7
1. Introducción 9
1.1 Justificación y contexto 10
1.2 Objetivos del TFG 10
1.3 Alcance del TFG 10
1.4 Planificación del TFG 10
2. Escenario original 12
2.1 Zona Mainframe 12
2.1.1. Descripción de la zona 12
2.1.2. Mantenimiento, monitorización y gestión de alarmas 14
2.1.3. Relación de costes 14
2.2. Zona Webservers 16
2.2.1. Descripción de la zona 16
2.2.2. Mantenimiento, monitorización y gestión de alarmas 19
2.2.3. Relación de costes 19
2.3. Zona Integraciones 19
2.3.1. Descripción de la zona 20
2.3.2. Monitorización y gestión de alarmas 25
2.3.3. Relación de costes 26
2.4. Motivación para un plan de migración 32
3. Estudio de soluciones para la migración 34
3.1. Zona Mainframe 34
3.2. Zona Webservers 37
3.3. Zona integraciones 39
3.4 Monitorización y gestión de alarmas 39
4. Solución final para la migración 41
4.1. Configuración general 41
4.1.1. AWS EKS 41
4.1.2. AWS WAF 50
4.1.3. AWS CloudFront 53
4.1.4. AWS RDS 53
4.2 Costes y asignación de recursos 55
4.3 Monitorización y gestión de alarmas 56
5. Implantación 58
5.1 Planificación de la migración 58

5.1.1 Cronología 59
5.2 Implantación 61
5.3 Resultado de la migración 61
6. Conclusiones 63
7. Bibliografía 65

Índice de figuras

Figura 1. Esquema de la zona Mainframe. 12
Figura 2. Esquema de la zona Webservers 18
Figura 3. Esquema de la Zona Integraciones 20
Figura 4. Esquema de una petición PULL. 21
Figura 5. Esquema de una petición PUSH. 22
Figura 6. Esquema de la zona webservers. 23
Figura 7. Esquema de la zona integraciones. 24
Figura 8. Servidor de alertas Zabbix. 25
Figura 9. Canal de Microsoft Teams para las alertas de Zabbix. 26
Figura 10. Instancia AWS EC2 m5.2xlarge. 27
Figura 11. Instancia AWS EC2 m5.xlarge y m5.large. 28
Figura 12. Instancia AWS RDS db.m5.8xlarge. 28
Figura 13. Instancia AWS RDS db.m5.4xlarge. 29
Figura 14. Instancia AWS EC2 c5.4xlarge. 29
Figura 15. Instancia AWS Redis cache.r5.2xlarge. 30
Figura 16. Resumen costes totales. 32
Figura 17. Página web de Zabbix. 40
Figura 18. Cústers de AWS EKS. 41
Figura 19. Nodos del cúster AWS EKS Live. 42
Figura 20. Configuración del node-group ONDEMAND. 42
Figura 21. Nodos del node-group ONDEMAND. 43
Figura 22. Configuración del clúster AWS EKS Live. 44
Figura 23. Configuración del node-group SPOT. 44
Figura 24. Nodos del node-group SPOT. 45
Figura 25. Plantilla de configuración de aplicación. 46
Figura 26. Página web de ArgoCD con aplicaciones. 46
Figura 27. Vista de aplicación de ArgoCD. 47
Figura 28. Pipeline de Jenkins para aplicar configuración. 48
Figura 29. Plantilla de Terraform para configuración de DNS de aplicación. 48
Figura 30. Esquema Kubernetes. 49
Figura 31. Plantilla de configuración de Ingress ALB. 49
Figura 32. Esquema AWS WAF. 50
Figura 33. AWS WAF. Web ACLs. 51
Figura 34. AWS WAF. Gráfica de peticiones. 51
Figura 35. AWS WAF. Reglas. 52
Figura 36. AWS WAF. Gráficas de Bots. 52
Figura 37. AWS WAF. Recursos asociados. 53
Figura 38. AWS CloudFront. 53
Figura 39. AWS RDS. Listado de algunas RDS por aplicación. 54
Figura 40. Esquema final de la solución con AWS EKS. 54
Figura 41. Kubecost. Coste total. 55

Figura 42. Kubecost. Relación de gastos por aplicación. 56
Figura 43. Kubecost. Desglose del gasto de una aplicación. 56

1. Introducción

En este apartado abordaremos la situación inicial en la que se encuentra la empresa analizando los motivos por el cual se acomete una migración de tal magnitud. Se analizarán las distintos entornos o situaciones separadamente para simplificar la problemática y su estudio, aunque en el proceso de implantación se realizará conjuntamente y al mismo tiempo.

1.1 Justificación y contexto

Este TFG contempla un análisis de un plan de migración en un cambio de paradigma de una gran empresa hotelera con sede en Palma de Mallorca fundada a mitad del siglo pasado y que cuenta con 350 hoteles repartidos por 40 países y con más de 45.000 empleados en 2019. Los servicios centrales y los servicios informáticos se encuentran totalmente centralizados en el edificio de oficinas centrales en Palma de Mallorca, que cuentan más de 1000 empleados en esta sede.

Este proyecto surge de la necesidad por parte de la empresa a la que pertenezco de establecer un plan de crecimiento a medio-largo plazo en el que se plantea un escenario de crecimiento de la empresa, tráfico de Internet, reservas de hoteles, servicios asociados mucho mayor del que hay en la actualidad. Dicho escenario empezó a plantear en los años 2014-2017 en los que la empresa tuvo un crecimiento muy elevado de reservas a través de su web y los buscadores como Booking, Expedia, Trivago, Ctrip y otras, empezaron a integrar los servicios de búsqueda de disponibilidad y gestión de reservas con nuestra empresa, hay que decir que hasta entonces el principal método de consulta de disponibilidad y reservas se hacía con canales conectados directamente a nuestros sistemas (punto a punto) de Amadeus, con agencias de viajes y el más importante era la propia agencia de viajes de El Corte Inglés (VECISA) y el uso particular de la web de reservas de la empresa era totalmente residual.

Con la expansión de Internet y de los metabuscadores (Booking, Expedia, Ctrip, Trivago, etc) el cliente particular empezó a usar canales alternativos a las agencias de viajes y usó más la web, tanto de dichos metabuscadores como la de la propia empresa, que implicó cambios significativos y muy rápidos en el modelo que había hasta entonces.

Los cambios comprendían la modernización de la web y la implantación de sistemas que pudieran absorber todo el tráfico de las consultas masivas de disponibilidad y precios de los metabuscadores, en ese momento se apostó por ampliar y escalar el número de servidores que alojaban la web de reservas para potenciarla, se separó el tráfico de los metabuscadores de la gestión de reservas que se centralizaba hasta entonces en un núcleo de extenso código en Cobol alojado en un mainframe de IBM y se creó una zona de Amazon Web Services

(AWS) con una serie de servidores y servicios para atender a dichos metabuscadores.

Con esto ya tenemos tres escenarios, el núcleo para la gestión de reservas en el mainframe de IBM, una zona con servidores dedicados en el hosting de IBM para la web y una zona para los metabuscadores en AWS, todas interconectadas en sí para compartir información.

1.2 Objetivos del TFG

El objetivo principal y único de este TFG es la realización de la migración con éxito, se entiende que el éxito de la migración es la implementación exitosa de los tres escenarios que deben migrarse al mismo tiempo y los servicios migrados deben proporcionar correctamente datos reales de reservas, disponibilidades y precios para los distintos servicios, que los requieren como pueden ser la web, la aplicación que se usa en el callcenter de reservas, metabuscadores, servicios pull-push, etc.

1.3 Alcance del TFG

El alcance de este TFG es la de analizar la motivación para proponer una migración de estas proporciones, analizar las posibles soluciones y alternativas que se han barajado para la migración, especificando el motivo por los cuales se descartaron o se llevaron a la práctica, proponer una planificación para la migración de los servicios y exponer cómo se realizó la migración especificando sus fallos y aciertos y mostrar los resultados obtenidos, además, también se especificará el sistema de monitorización y gestión de alarmas que se implantará, analizando algunas posibles soluciones que finalmente se descartaron.

Este TFG se centrará en otorgar una visión sobre la arquitectura de sistemas de la empresa sin entrar en detalles de las distintas redes, configuraciones de los servidores o desarrollos de software que se implementarán antes y durante la migración, aunque sí se expondrán algunos detalles interesantes sobre algunas configuraciones o plantillas de automatización que pueden servir para comprender el modelo de arquitectura que se adoptará en la empresa.

1.4 Planificación del TFG

Ya se ha comentado que existen tres escenarios que deben migrarse, así en este TFG se irán analizando separadamente para cada zona, el escenario pre-migración, con su análisis de costes y su sistema de monitorización y gestión de alarmas.

Posteriormente se analizarán las posibles soluciones de migración para cada zona, con su análisis de costes y su sistema de monitorización, aquí se hará ahondará en los pros y contras de cada solución.

Finalmente se expondrá la solución aceptada y la que se implantará finalmente en la migración.

2. Escenario original

Este apartado servirá para describir el escenario pre-migración, especificando detalladamente la descripción de la zona con su sistema de monitorización y su relación del coste de cada zona a migrar.

En el último subapartado se explicará los motivos para elaborar el plan de migración de la empresa, que, aunque los costes de mantenimiento del sistema pre-migración tienen mucho que ver no es el único motivo y se analizarán en dicho subapartado.

2.1. Zona Mainframe

Es la zona más importante de toda la migración ya que todo gira alrededor del gestor de reservas alojado en el Mainframe.

2.1.1. Descripción de la zona

En lo que se ha denominado "zona Mainframe" comprende una serie de servicios relacionados con el sistema gestor de reservas y disponibilidad, llamado Sirius, que es el programa principal, alojado en el host, una base de datos IBM DB2 alojada localmente en el host y un sistema de control de usuarios y seguridad IBM RACF (ver Figura 1).

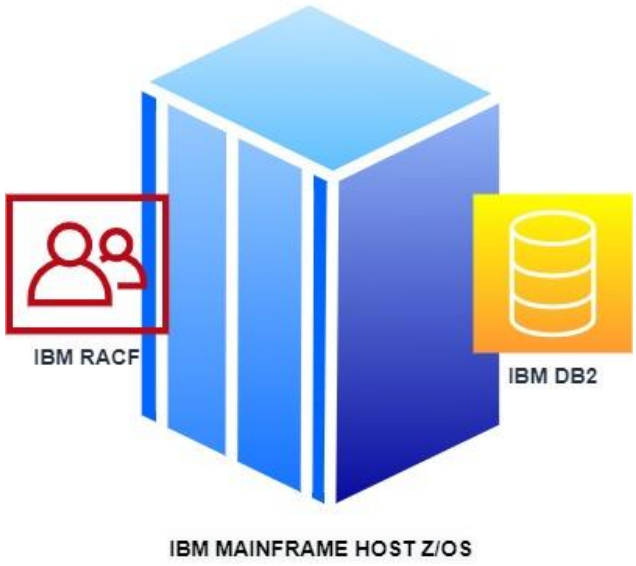

Figura 1. Esquema de la zona Mainframe.

Históricamente, mi empresa únicamente disponía de esta zona para toda su gestión de reservas y consultas de disponibilidad, estamos hablando de un mundo donde los sistemas se conectaban punto a punto mediante líneas RDSI, para reservas una habitación de hotel las únicas vías era ir a una agencia de

viajes o contactar directamente con el hotel vía telefónica o presencia. En un entorno así, no se necesitaba más que un servidor central con el programa gestor de reservas (Sirius) y conexiones punto a punto con las centrales de reservas, como son Pegasus o Amadeus. Con la llegada de Internet a los hogares el sistema tuvo que ampliarse para poder alojar servidores web que el mainframe no podía proporcionar, de aquí surge la "zona webservers" que ya se explicará más adelante.

Sin embargo, la llegada de Internet no supuso para la compañía una transformación de su gestor de reservas que siguió en el mainframe. La aparición de unos servidores web no suponía ningún obstáculo a seguir usando el gestor de reservas sin apenas tocar el código. Simplemente se debía adaptar el código de la web de la compañía, donde se incrustó un formulario en la web para consulta de disponibilidad y otro para la realización de las reservas (explicado muy resumidamente). Estos formularios enviaban la consulta al host (mainframe) y éste les devolvía los datos, igual para las reservas. Hay que decir que los servidores web estaban alojados en el datacenter de IBM donde también estaba el mainframe por lo que la comunicación era directa.

Con la llegada de los metabuscadores que ya se ha hablado en la introducción, se amplió el sistema del conjunto de mainframe+webservers a una "zona de integraciones". Debido a que los webservers no eran capaces de absorber tal cantidad de peticiones se creó dicha zona donde se alojaron unos servidores que atendían las consultas de los metabuscadores. La "zona de integraciones" consistía en otros servidores web específicos para atender las diferentes peticiones que los metabuscadores y partners de la compañía demandaban, como pasó con la zona webservers, las peticiones se enviaban directamente al mainframe para todas estas consultas y solamente supuso para el programa gestor en ampliaciones del código en subprogramas para adaptar dichas consultas.

Toda esta evolución que se ha comentado también ha venido acompañada con un aumento significativo de subprogramas en el gestor debido a otro factor a tener muy en cuenta. Este factor es la especificación de las reservas. Al inicio, cuando solo existía la zona mainframe, una reserva se componía de un tipo de habitación, número ocupantes, fecha de entrada, fecha de salida y régimen de alojamiento. Hoy en día existen multitud de variantes para una reserva, empezando que las cadenas de hoteles han visto que a los usuarios nos gusta tener la posibilidad de elegir entre muchas opciones. Por eso se han creado muchos tipos de habitaciones, algunos ejemplos son la Cyclist Room, Twin Room, Family Room, con vistas al mar, a la piscina, a la montaña, Suite, Suite Executive, etc. Y no solo tipos de habitaciones, también complementos, como el pack de bienvenida de cava y chocolate, toallas de piscina, servicio de distintas

almohadas disponible, etc. Y servicios adiciones: servicio de recogida en el aeropuerto, late checkout, early checkin, etc.

Todas estas opciones (y muchas más) que se han comentado repercuten en el gestor de reservas (Sirius) en forma de líneas de código adicionales. En concreto, cuando se empezó a plantearse la migración de este gestor a otros entornos tenía más de 15 millones de líneas de código, incrementándose día a día a medida que se iban añadiendo funcionalidades y especificidades.

2.1.2. Mantenimiento, monitorización y gestión de alarmas

Un asunto importante es el tema del mantenimiento del sistema, la monitorización y la gestión de alarmas puesto que de ello depende el buen funcionamiento del sistema y evitar pérdidas de servicio.

En el caso de esta zona el mantenimiento, la monitorización y la gestión de alarmas corre totalmente a cargo del personal de IBM, encargándose de todo lo necesario para el buen funcionamiento del sistema. Esto es porque el mainframe está alojado en el centro de datos que IBM tiene en Alcalá de Henares, Madrid, teniendo un BRS (Business Recovery Service) en su centro de datos de Barcelona.

Poniéndonos en el lugar de la empresa, en una época pre-internet doméstico, este escenario era el más adecuado y simple para cumplir los objetivos de servicio. Ante una posible caída del sistema, el personal de IBM se encargaba de monitorizar permanentemente el sistema en un centro de control donde las alertas se centralizan. Además, dispone de personal de todas las áreas permanentemente de guardia y disponible en todo momento. Para mi empresa, se evitaba tener que disponer de todo ese personal ni de establecer sistemas de monitorización que en aquel entonces hubieran sido difíciles de implementar.

2.1.3. Relación de costes

Este apartado es muy importante porque es el detonante de la migración que es tema de este TFM.

El sistema de cobro del servicio del mainframe de IBM se basa en una tarifa de facturación por MIPS (Millones de Instrucciones Por Segundo) consumidos, tanto por el gestor de reservas como por el gestor de RACF o la base de datos DB2, que recordamos que estaban alojadas localmente en el propio mainframe, y por tanto consumen CPU que se traduce en MIPS que se facturan al cliente, en este caso mi empresa.

Los servicios de mantenimiento, monitorización y gestión de alarmas están incluidos en el precio del MIPS. Lo que resulta una tarificación sencilla para mi empresa, tanta CPU consumes tanto pagas. Sin embargo, no es tan fácil. IBM establece una línea base por contrato, que suele ser una media a la baja del consumo de MIPS, así si mi empresa un día no consume MIPS no le representa un ahorro ya que ha pagado por adelantado los MIPS de la línea base (con un ahorro por pronto pago). En general, si se consumen MIPS por debajo de la línea base representa un desaprovechamiento por parte de mi empresa, pero si consume MIPS por encima de la línea base, se facturan por MIPS consumidos diariamente.

Existe un matriz a los consumos de MIPS diarios y es que existe un horario de facturación de MIPS y un horario en los que no se facturan los MIPS por muchos que se consuman, si se sobrepasa la línea base. El horario de facturación de MIPS es de lunes a viernes de 9.00h a 20.00h. Este horario coincide con el horario laboral de las operaciones bursátiles y viene motivado porque los bancos son los principales clientes de mainframe y principales consumidores de MIPS.

En el momento de la migración que es objeto de este TFM, la línea base del último contrato con IBM, se establecía en 3000 MIPS diarios y el precio por MIPS por encima de la línea base era de 72,15 euros. Durante los últimos años anteriores la pandemia de 2020 causada por la COVID, mi empresa consumía una media de 400 MIPS diarios por encima de la línea base. Es fácil calcular la factura fuera de contrato:

Facturación extra =	72,15 euros/MIPS x 400 Mips consumidos x 260 días facturables al año =	7.503.600 euros

Así pues, mi empresa pagaba 7.503.600 euros al año extra del contrato anual de IBM, que eran unos 4.000.000 euros.

Resumiendo, solamente con esta zona mainframe, IBM debía cobrar a mi empresa 11.503.600 euros aproximadamente al año. Al final IBM aplicaba un descuento del 30% sobre el precio, por lo que queda en 8.052.520 euros, que tampoco es una factura despreciable.

2.2. Zona Webservers

Ya se ha comentado como surgió esta zona de webservers anteriormente. Un resumen es que cuando internet empezó a ser accesible a la población, surgió

como necesitad de las empresas disponer de páginas web, al principio solamente informativas, pero poco a poco añadiéndoles más funcionalidades.

2.2.1. Descripción de la zona

Con anterioridad a la aparición de esta zona, la principal fuente de reservas de la compañía provenía de las agencias de viajes y se mantuvo como principal fuente de ingresos hasta la llegada de los metabuscadores. Aunque la página web permitía la realización de reservas online casi desde el principio de su implantación eran pocas las reservas que se materializaban por este canal. Internet ya estaba plenamente introducido y asimilado en la mayoría de los hogares, sin embargo, había una cierta desconfianza al pago online. Por eso los canales más usados eran el de agencias de viajes y el callcenter (venta telefónica) de la propia compañía. Así la web se mantenía en un plano informativo que los clientes usaban para conseguir información de los destinos, consultaban precios, fotografías de los hoteles y posteriormente recurrían a la venta a través del callcenter o acudían a una agencia de viajes para el paso de efectuar la reserva.

Con la llegada masiva y explosiva de los metabuscadores, se convirtieron en muy poco tiempo en el principal canal de ventas de la compañía, y en concreto un metabuscador que estaba inundando de publicidad todos los canales de televisión, periódicos, radio, etc. Este buscador era Booking.com y pasó a ser el principal partner de la compañía durante un tiempo.

Sin embargo, mi compañía estableció rápidamente un plan de empresa para potenciar los canales de venta propios y ahorrarse las comisiones que exigían los metabuscadores. En concreto quería potenciar el uso de la página web porque suponía el canal más económico. El callcenter necesita mucho mantenimiento, muchos servidores (web, ldap, de aplicaciones, de almacenamiento, etc), mucho personal y grandes oficinas, en contraste de unos pocos servidores web. Hablando del plan de empresa que se puso en marcha, nada original, se basaba en igualar el precio que ofrecía Booking pero añadiendo opciones que no estaban disponibles en el metabuscador. Por ejemplo, las habitaciones ofertadas en Booking.com no incluían el desayuno y si se reservaba a través de la web propia sí era incluido. Aparte de esto último, la compañía realizó campañas muy fuertes de publicidad en televisión y radio anunciando el precio más bajo en la web, cosa que no era del todo cierta, puesto que el precio de Booking era el mismo, aunque sin desayuno. Hay que decir que el contrato inicial con Booking establecía que ellos debían tener el precio más bajo, incluso del de la propia compañía, y esto era posible por el gran volumen de reservas que se realizaban desde este buscador. Mi compañía respetó el contrato con Booking, solo que añadía un extra al precio que lo hacía más suculento al cliente.

Con lo dicho anteriormente, la web propia empezó a tener mayor peso en la facturación de la compañía, en detrimento de las agencias de viajes y del callcenter. Con el tiempo la web propia se ha convertido en el principal canal de ventas de la compañía, detrás están las ventas de los buscadores (Booking, Expedia, Ctrip, etc), en tercer puesto el callcenter propio y por último lugar las agencias de viajes. Esto propició que se tuviera que ampliar la potencia de esta zona. De dos servidores (lo mínimo para tener HA) web y cuatro servidores de aplicación (Jboss), se pasó a cuatro servidores web con mucho incremento de cpu y memoria, y siete servidores de aplicación de los que se duplicó tanto la cpu y memoria. En los siguientes tres años estos servidores pasarían a duplicar otra vez los recursos.

Pasando a la descripción técnica de esta zona, se comprende de unos servidores web en los que está alojada la web, otros servidores web separados de los anteriores dedicados a los cobros, que son PCI Compliance, y un servidor con una base de datos Oracle (ver Figura 2).

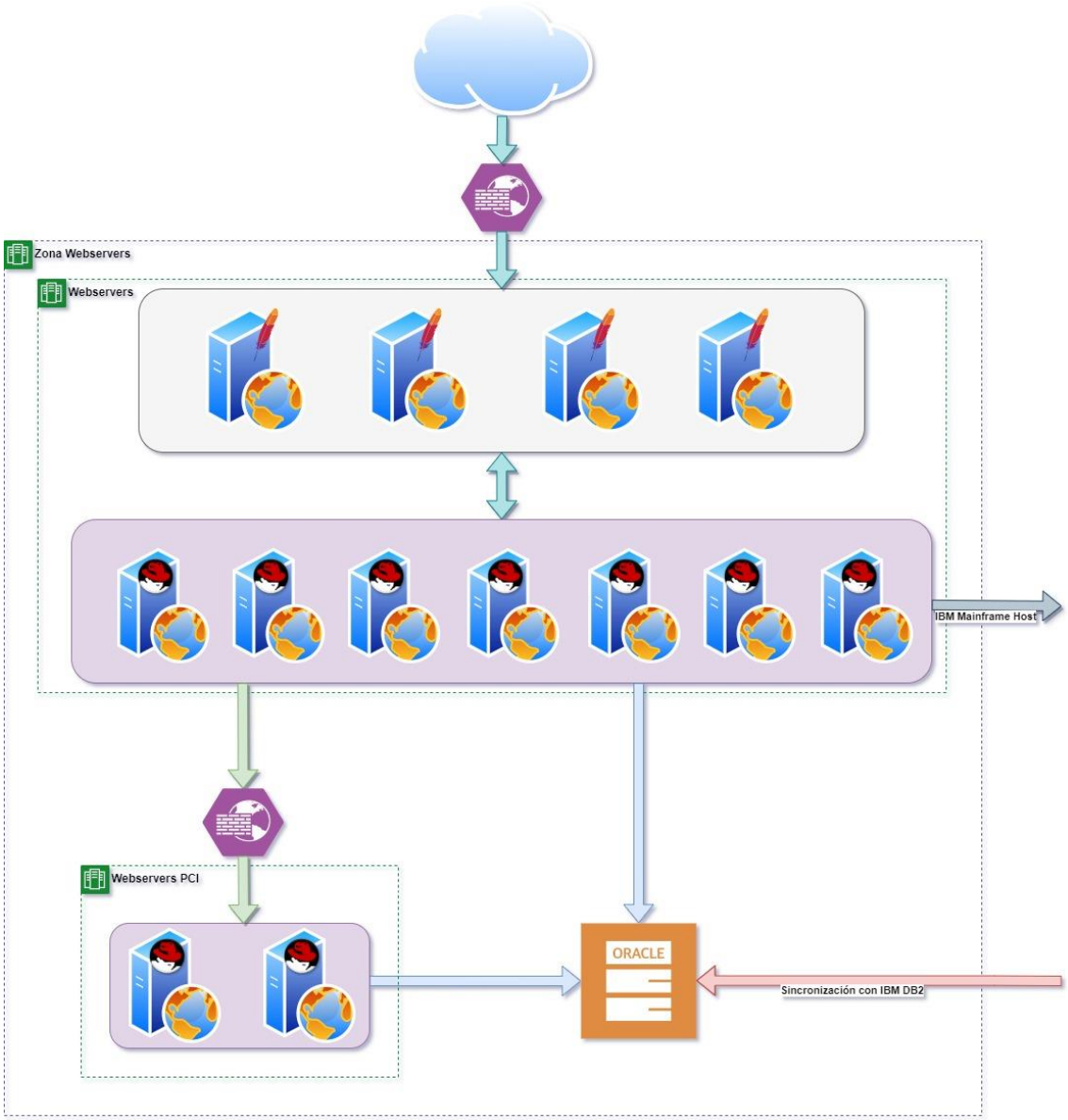

Figura 2. Esquema de la zona Webservers

En la imagen anterior podemos ver que la zona a migrar se compone de unos servidores web Apache que sirven el contenido estático de la web, imágenes, vídeos y html principalmente. Unos servidores que componen un cluster de Jboss EAP 7.2 en modo dominio para el contenido dinámico de la web, programado en Java, que consulta la disponibilidad y las operaciones sobre reservas hacia el Mainframe, mediante colas IBM MQ. Para el pago con tarjeta de crédito, un framework incrustado en la web proveniente de los webservers PCI garantizan que se cumpla en todo momento la normativa PCI Compliance para los pagos por medios electrónicos. Estos servidores también forman un cluster Red Hat Jboss EAP 7.2 en modo dominio.

El servidor de Oracle es otro elemento a migrar, contiene los datos de los usuarios, hoteles, habitaciones, etc., que necesita la web para su correcto funcionamiento. Estos datos se mantienen actualizados por existe una enlace de

sincronización continuo desde la base de datos de IBM DB2 alojada en el Mainframe, la transferencia de estos datos se realiza mediante IBM Datamirror y nos permite mantener tablas actualizadas en el Oracle sin necesitad de acceder directamente al DB2, cosa que aparte de ser más difícil tanto para los desarrolladores, como para los administradores de base de datos, representa un coste adicional, ya que los accesos al DB2 consumen MIPS.

2.2.2. Mantenimiento, monitorización y gestión de alarmas

Respecto a estos servidores, todos son servidores virtualizados alojados en el datacenter de IBM. Por lo que el mantenimiento corre a cargo del personal de IBM, que se encarga de tener el software actualizado (todo lo que no sea la web) y de que los sistemas funciones correctamente. Además, como ocurre con el Mainframe, dispone de un sistema de monitorización de alarmas en el centro de control y personal cualificado para atender y resolver dichas alarmas, tanto de los servidores web como del Oracle. Ya que ha salido el tema en el punto anterior, como es de suponer el IBM MQ y el IBM Datamirror también corre a cargo de IBM el mantenimiento, monitorización y solución de problemas que puedan ocurrir.

2.2.3. Relación de costes

En esta zona los servidores tienen un coste fijo pero dependiendo del tipo de servidor tienen un coste u otro porque la gestión y mantenimiento van incluido.

Los servidores de aplicación tienen un coste de 4500 euros mensuales fijos, aunque cada cambio que se realice por parte del cliente representa un cargo aparte, por ejemplo, subir memoria, cpu o espacio en disco. Los servidores web tienen un coste de 7500 euros mensuales.

Los servidores de bases de datos tienen un coste de 4700 euros porque la gestión y mantenimiento de la base de datos (Oracle) tiene un coste adicional de 3500 euros.
Así el coste de esta zona será:

Coste= (7200 euros x 7 Jboss) + (4500 euros x 4 Apache) + 4700 euros Oracle + 3500 euros BDA = 76.600 euros mensuales

Lo que hace un total de 919.200 euros anuales.

2.3. Zona Integraciones

2.3.1. Descripción de la zona

Ya se ha hablado de cómo surgió esta zona, con la llegada de los metabuscadores, que al inundar de consultas los webservers, encarecían la factura de Mainframe, ya que todas esa consultas se dirigían allí al no disponer de otro gestor de reservas.

La zona consta de una serie de clusters de servidores AWS EC2 (Elastic Compute Cloud) con un balanceador de carga delante de cada cluster. Estos balanceadores pueden ser AWS Classic Load Balancer, ALB (Application Load Balancer) o ELB (Elastic Load Balancer).
La zona también consta de una serie de bases de datos independientes para cada aplicación, todas en AWS RDS, y pueden ser MySQL u Oracle dependiendo de las características de la aplicación que la usa. Estas bases de datos están sincronizadas mediante AWS DMS (Database Migration Service) al Oracle de la zona Webservers.

En la imagen siguiente (figura 3) se puede ver el esquema de esta zona:

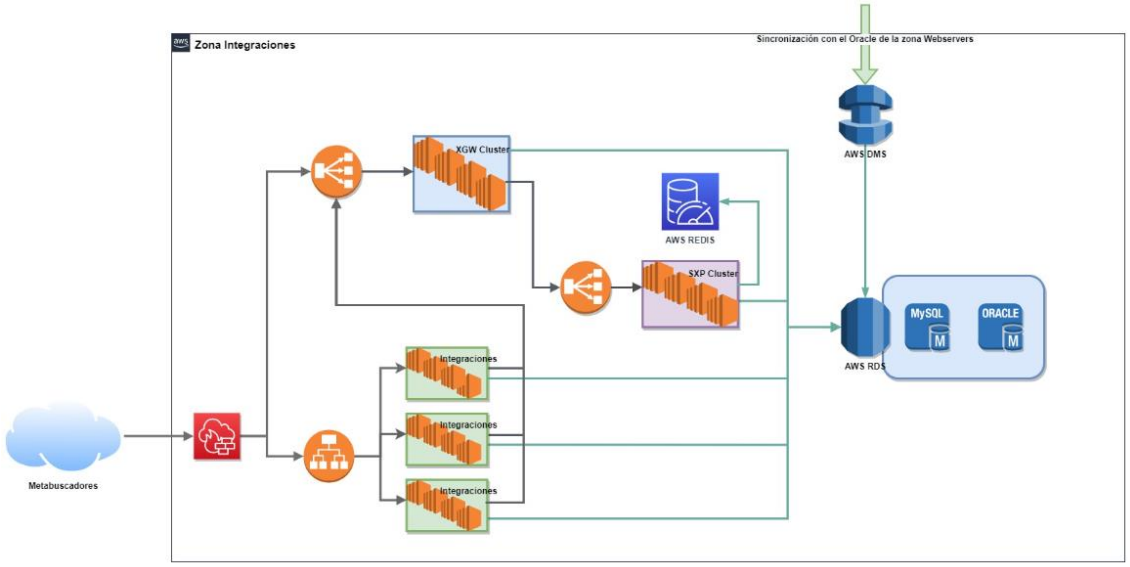

Figura 3. Esquema de la Zona Integraciones

Los clusters de EC2 son de tres tipos:

- XGW: Acrónimo de XML Gateway. Se trata de una aplicación escrita en java y servida por un Apache Tomcat que se encarga de la traducción de los diferentes tipos de peticiones que realizan los metabuscadores y traduce estas peticiones en un formato único.
- SXP: Acrónimo de Smart XML PULL. Se trata de una aplicación que imita a Sirius alojado en el Mainframe, pero solo para consultas de disponibilidades. Cuando un metabuscador realiza una petición, el XGW se encarga de traducir esa petición en un formato que SXP es capaz de

interpretar, SXP atiende esa petición y le devuelve la respuesta al XGW, que vuelve a traducir la respuesta en el formato que el metabuscador había realizado y devuelve la petición al metabuscador en formato XML.

- Servidores de Integraciones: Son servidores específicos para cada metabuscador, sirven para enviar las disponibilidades a los metabuscadores (notificaciones PUSH). La diferencia entre PULL y PUSH, es que en el PULL es el metabuscador quien nos reclama las disponibilidades directmente, en cambio, en las PUSH, somos nosotros quien les enviamos las disponibilidades, la diferencia principal es quien inicia la consulta. Así estos servidores crean una petición de disponibilidad como lo harian los metabuscadores PUSH, que van directamente al XGW, le envían la petición al XGW que la transforma en el formato que el SXP necesita y éste les devuelve la consulta, al XGW y éste al servidor que ha iniciado la transacción, cuando recibe la consulta, la envia al metabuscador específico.

En los siguientes gráficos se pueden ver los esquemas de como funcionan las peticiones PULL (figura 4) y PUSH (figura 5):

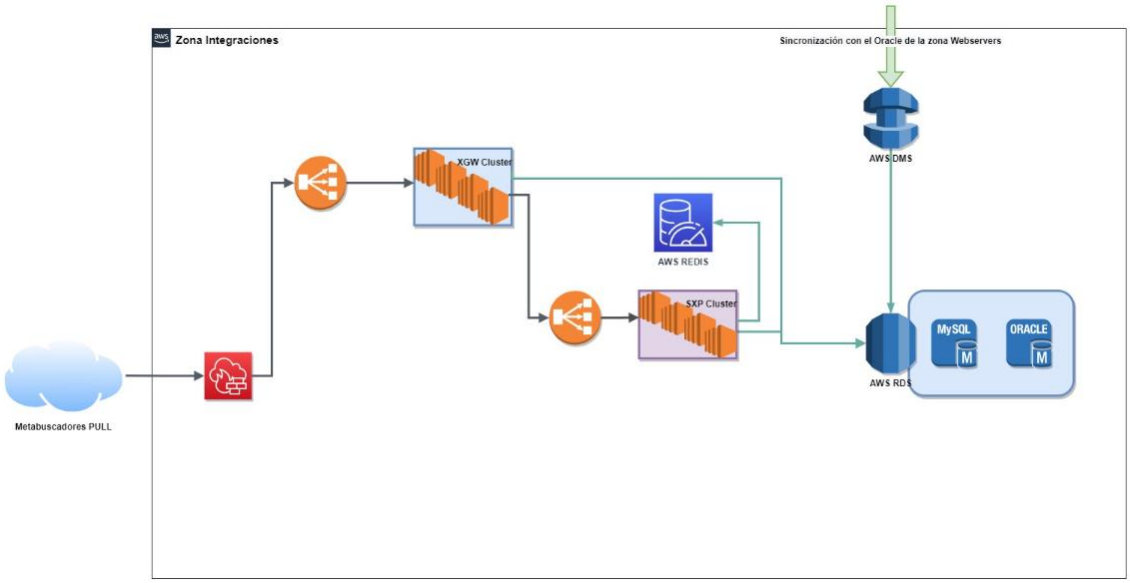

Figura 4. Esquema de una petición PULL.

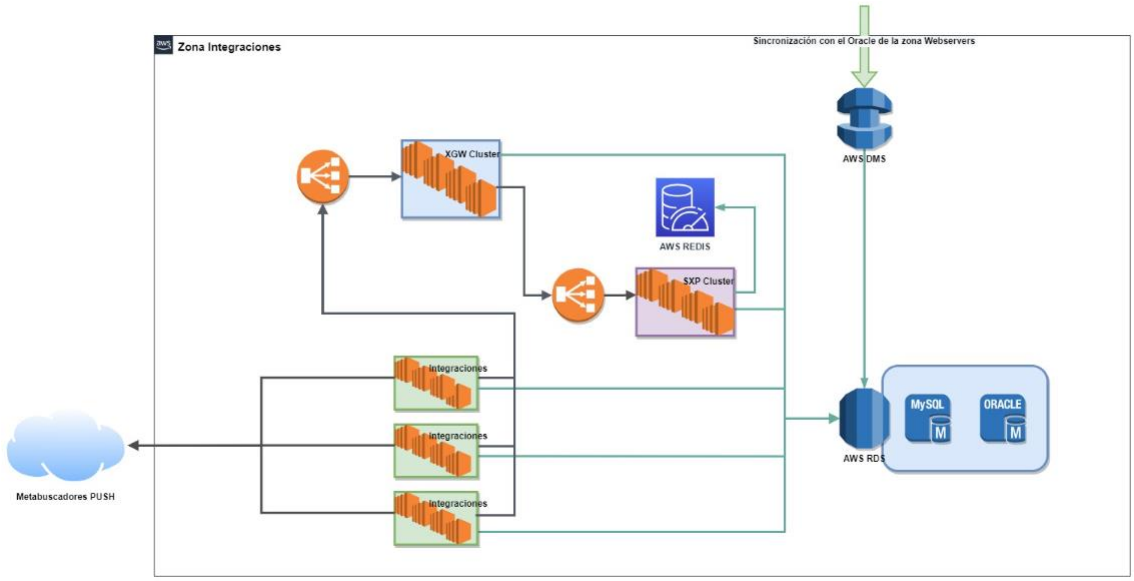

Figura 5. Esquema de una petición PUSH.

Una mejora que se realizó posteriormente en la zona webservers fue la de usar el cluster de XGW para las consultas de disponibilidad y seguir usando el mainframe para la gestión de reservas (ver figura 6 y 7). Esta mejora produjo un ahorro de MIPS que fue anulada por el incremento de petición y consumo de MIPS, en el momento de la mejora el consumo de MIPS facturables estaba por encima de 1000 MIPS al día y con la mejora volvimos a estar 300-500 MIPS por encima de la linea base.

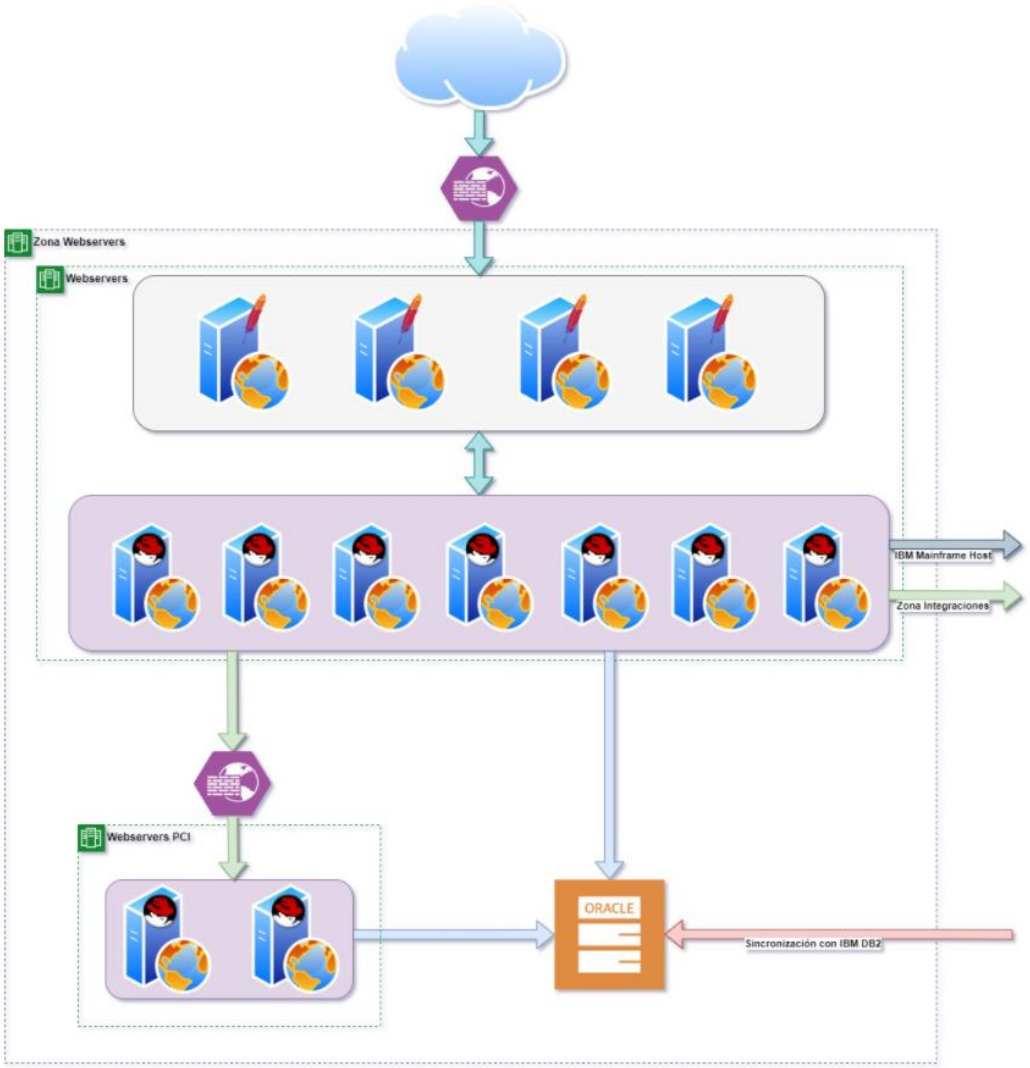

Figura 6. Esquema de la zona webservers.

Si volvemos a ver el gráfico de la figura 3, podemos ver que todos los clusters y servidores hacen uso de los dos clusters de XGW y SXP, pues son la pieza clave de toda la zona. Sin embargo, existe una pieza más crítica y es la base de datos que usa SXP pues de ello dependen todas las respuestas de disponibilidades y precios que proporciona SXP. SXP también utiliza una base de datos Redis a modo de caché para tablas de precios. Existe una singularidad en las consultas de los metabuscadores y es que repiten muchas veces las mismas consultas en el mismo día, por poner un ejemplo, un hotel puede tener el mismo precio para una determinada habitación y ocupación hoy que mañana, sin embargo, los metabuscadores piden en una misma consulta los precios de las habitaciones para todo el mes, así, cacheando el precio para determinadas consultas se "ahorra" el tiempo de ir a la base de datos relacional MySQL y las consultas son más rápidas. Los datos en Redis tienen una expiración de ocho horas por lo que si SXP accede a Redis para conseguir un precio y se ha eliminado de la base de datos, SXP accede a Mysql para obtener el dato y luego lo inserta en Redis para su posterior utilización.

Esta base de datos, se trata de una AWS RDS Mysql, es alimentada por un AWS DMS proveniente del Oracle de la zona de Webservers pero tiene más particularidades, la tarea de DMS es en modo batch, es decir, se realiza de forma programada durante la madrugada y actualiza las tablas de forma diferencial, solo actualiza los cambios. Estos datos en las tablas que usa SXP son extraidas del mainframe mediente IBM Datamirror y replicadas al Oracle de la zona de webservers y de allí mediante AWS DMS a la zona de integraciones para poder usarse con SXP. Todo el proceso se realiza cada día de madrugada porque así no se facturan MIPS y porque los hoteles suelen actualizar los datos de su ocupación antes de las 21.00h de cada día, por lo tanto, la disponibilidad y los precios que se les ofrece a los metabuscadores es bastante exacta, exceptuando algunas pequeñas discrepancias que pueden solventarse porque al realizar la reserva, esas discrepancias pueden corregirse.

Con la mejora que se hizo de conectar la zona webservers con el XGW de la zona de integraciones (Figuras 6 y 7) para la consulta de disponibilidades y precios, se tuvo que adaptar el procedimiento de cambios de precios para que afectara lo mínimo, antes de la mejora los precios de una habitación de actualizaban a las 9.00h y con la mejora pasaron a actualizarse a las 21.00h del día anterior para que el precio ofrecido en la web propia y en los metabuscadores fuera lo más exacto posible, siempre teniendo en cuenta que el mercado mayoritario de la compañía es Europa y el mayor número de reservas y consultas se realiza de 9:00h a 23:00h.

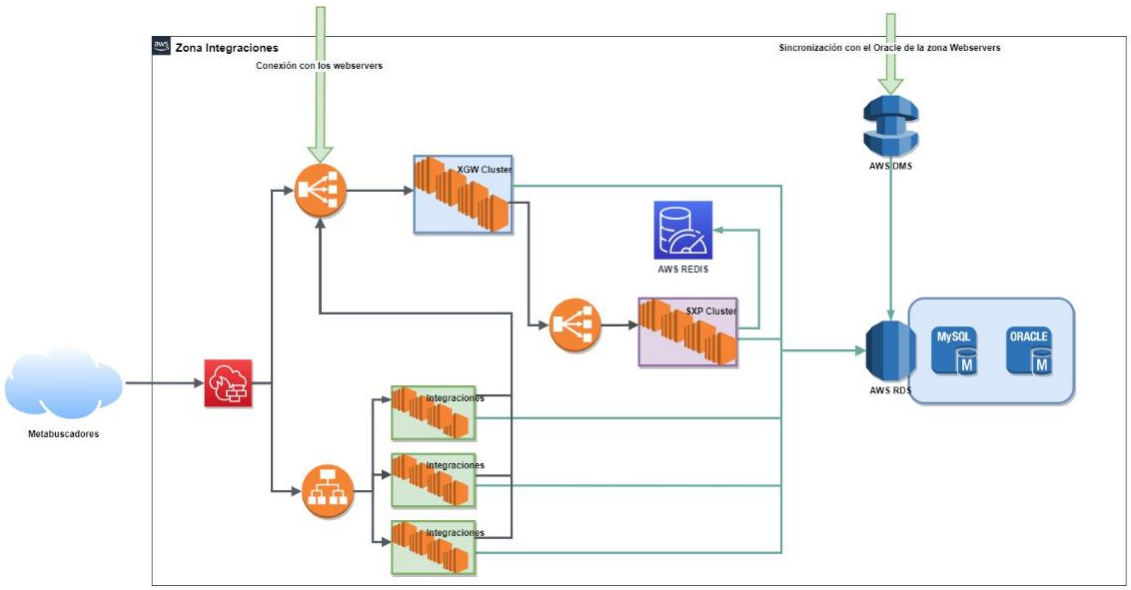

Figura 7. Esquema de la zona integraciones.

Resumiendolo mucho, podemos decir, que existe una répica en Mysql en esta zona que contiene muchas de las tablas que están en el IBM DB2 y que contiene

todos los datos necesarios para que el SXP reproduzca la disponibilidad de forma análoga a cómo lo hace el Sirius en el mainframe.

2.3.2. Monitorización y gestión de alarmas

La monitorización de los servidores y aplicaciones de esta zona corre a cargo de la empresa y del personal de administración de sistemas encargados de mantener esta zona de integraciones. Para ello se ha instalado un sistema de alertas Zabbix sobre una máquina AWS EC2 y en cada servidor de la zona de integraciones se le ha instalado un agente de Zabbix. El agente se comunica con el servidor de Zabbix y le manda todas las alertas que se han configurado en el servidor de Zabbiz. En cuanto a estas alertas, el servidor ya cuenta con algunos templates, alertas y triggers predefinidos que conviene revisar y ajustar a nuestras necesidades. Aparte de estas plantillas predefinidas se pueden añadir todas las que se requieran e incluso programar autoreparaciones. Estas alertas se pueden configurar avisos a varios sistemas, por ejemplo, correo electrónico, sms y como es nuestro caso, aun un canal de teams (ver Figura 9). En la figura 8 se puede ver como en el apartado de Problems una alerta ha pasado a Resolved.

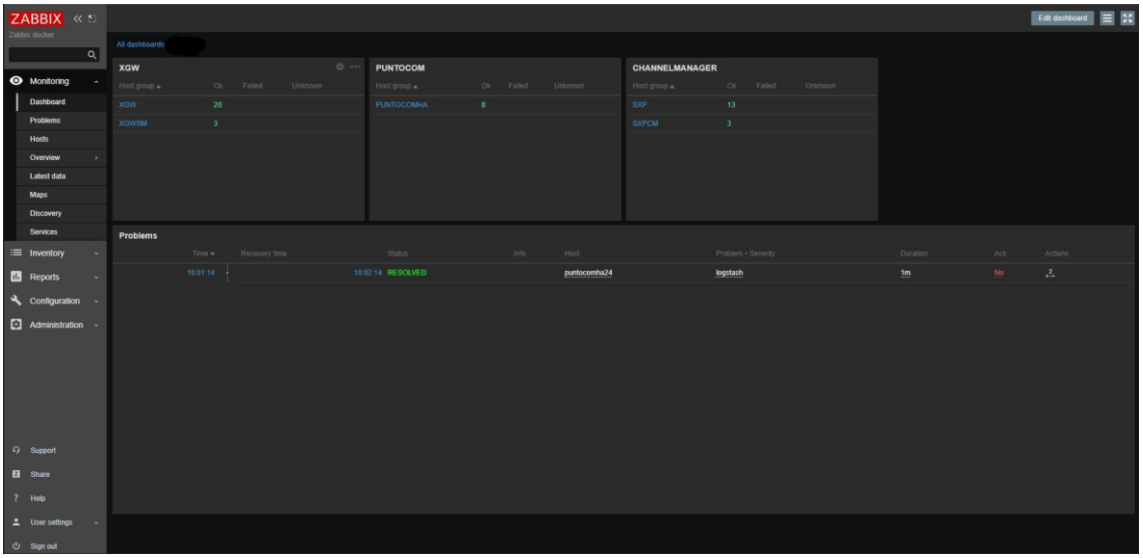

Figura 8. Servidor de alertas Zabbix.

En el canal de Microsoft Teams correspondiente (Figura 9) podemos observar cómo esta alerta es recurrente y se va notificando el problema y su resolución cada poco tiempo. En este caso viene dado por la sonda de logstash que envía los logs a un servidor Elastic Logstah para su almacenamiento y tratamiento. Respecto a este problema en concreto parece ser que la sonda de logstash está usando mucha memoria y el propio sistema mata el proceso, la sonda tiene programado un autoencendido por lo que al arrancar el Zabbix entiende que el problema se ha resuelto.

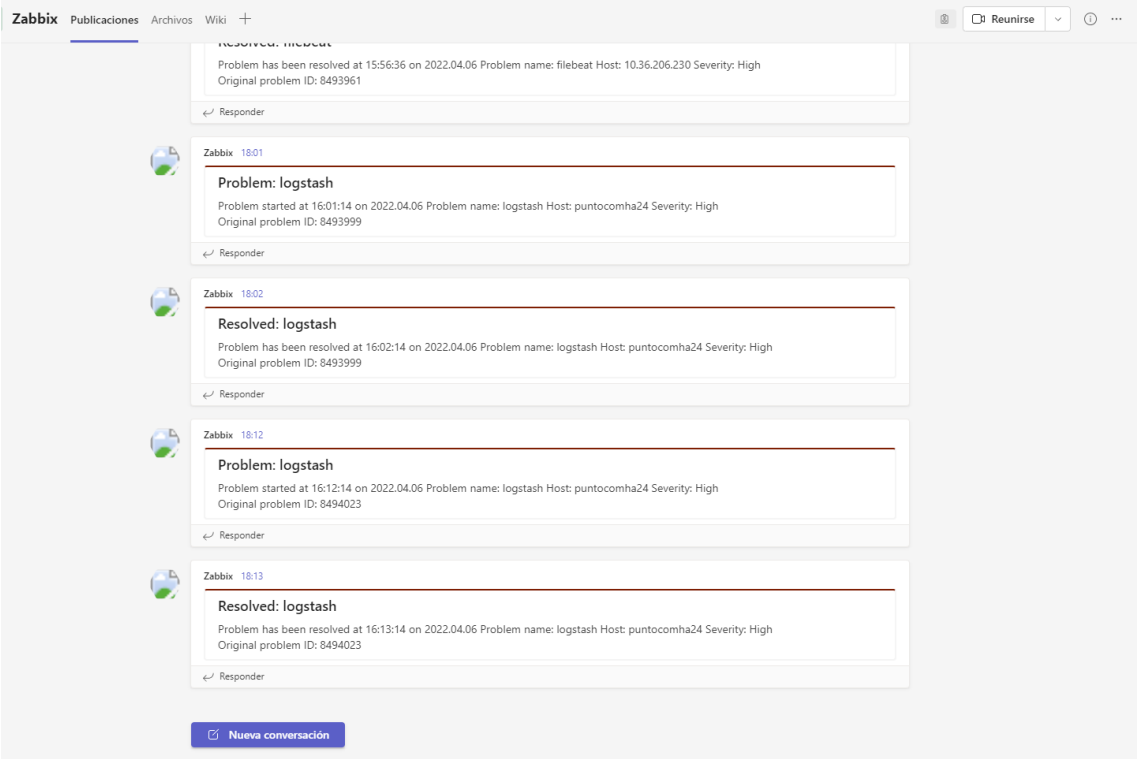

Figura 9. Canal de Microsoft Teams para las alertas de Zabbix.

Como se puede apreciar en la figura 8, también es posible en Zabbix organizar los clusters en pequeñas ventanitas llamadas widgets. Estos widgets pueden ser de varios tipos, en la imagen vemos de dos tipos, para mostrar en general el estado del cluster y para especificar los problemas, existen más tipos de widgets pero no se adaptan a nuestras necesidades y por lo tanto, no los usamos.

2.3.3. Relación de costes

Para la relación de costes de esta zona podemos desglosarla por tipos de servicios para facilitar su comprensión y al final sumar todos los conceptos en una hoja de cálculo. En esta zona tenemos los servicios de EC2, RDS, DMS, REDIS, ALB y WAF**.**

Hay que tener en cuenta que todos los servicios se encuentran en AWS en la región de Irlanda, esto es debido a que cuando se empezó a usar esa región era la única disponible para Europa (posteriormente AWS ha añadido varias regiones más, París, Milán, Londres, Fráncfort y Estocolmo). Además, no todos los servicios están disponibles en todas las regiones, algo que sí ocurre en Irlanda ya que fue la primera, y tiene la particularidad de que la mayoría de los servicios son más baratos en esa región (AWS calcula los costes de cada servicio de forma local en cada región).

Todos los precios se pueden consultar a través de la web de AWS https://aws.amazon.com/es/ec2/pricing/on-demand/.

EC2:

- XGW: El clúster se compone de 20 máquinas EC2 de propósito general com sistema operativo Linux de AWS m5.2xlarge.

Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Almacenamiento	Rendimiento de la red
m5.2xlarge	0,428 USD	8	32 GiB	Solo EBS	Hasta 10 gigabits

Figura 10. Instancia AWS EC2 m5.2xlarge.

El uso de estas máquinas es de 24x7 por lo que el coste total del clúster anualmente es de:

Coste anual: 0,428 USD/h x 24 h x 365 días/año x 20 máquinas = 74.985,6 USD/año

- SXP: El clúster se compone de 13 máquinas EC2 de propósito general com sistema operativo Linux de m5.4xlarge.

Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Almacenamiento	Rendimiento de la red
m5.4xlarge	0,856 USD	16	64 GiB	Solo EBS	Hasta 10 gigabits

Figura 11. Instancia AWS EC2 m5.4xlarge.

El uso de estas máquinas es de 24x7 por lo que el coste total del clúster anualmente es de:

Coste anual: 0,856 USD/h x 24 h x 365 días/año x 13 máquinas = 97.481,28 USD/año

- Integraciones: Este clúster se compone de aplicaciones más pequeñas que requieren menos potencia tanto de cpu como de memoria. Sin embargo, la cantidad de diversas aplicaciones tiene el inconveniente de requerir muchas máquinas, pero de menor tamaño que los anteriores clústers.

Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Almacenamiento	Rendimiento de la red	Número de instancias
m5.xlarge	0,214 USD	4	16 GiB	Solo EBS	Hasta 10 gigabits	20
m5.large	0,107 USD	2	8 GiB	Solo EBS	Hasta 10 gigabits	75

Figura 11. Instancia AWS EC2 m5.xlarge y m5.large.

Coste anual: 0,214 USD/h x 24 h x 365 días/año x 20 máquinas
+
0,107 USD/h x 24 h x 365 días/año x 75 máquinas =
107.791,8 USD/año

El coste total de todas las instancias es:

Coste total = 74.985,6 USD + 97.481,28 USD + 107.791,8 USD =
280.258,68 USD/año

RDS:

- MySQL: Se trata de una instancia de RDS Multi-AZ, esto significa que aunque visiblemente sea una instancia realmente existe una réplica actualizada al momento para casos de fallas o procesos de backups, y se van conmutando sin que los usuarios o las aplicaciones se den cuenta. El Multi-AZ es muy útil en los casos en que se precise realizar backups o snapshots sin pérdida de servicio, ya que estos se realizan sobre la copia sin servicio, también es muy útil en el caso de necesidad de un reinicio de la base de datos, ya sea por actualizaciones, upgrades o cambios de configuración, en esos casos las aplicaciones solo perciben el corte de las transacciones que tienen con la base de datos pero en unos segundos el Multi-AZ conmuta a la otra copia y ya se puede operar normalmente mientas una de las copias se está reiniciando.
  Los precios actuales se pueden consultar en AWS RDS MySQL Price.

Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Implementación	Rendimiento de la red	Número de instancias
db.m5.8xlarge	6,05 USD	32	128 GiB	Multi-AZ	10 gigabits	1

Figura 12. Instancia AWS RDS db.m5.8xlarge.

Coste anual = 6,05 USD/h x 24 h x 365 días/año = 52.998 USD

- Oracle: Amazon AWS da la posibilidad de elegir dos opciones que afectan al precio. La primera es que el cliente debe aportar la licencia de Oracle gestionada independientemente de Amazon, esto es que el cliente debe adquirir previamente la licencia con Oracle y luego introducirla en AWS para operar estas bases de datos. La segunda opción es con el coste de la licencia integrada en el precio final, la ventaja de esta modalidad es que el cliente no tiene que gestionar la licencia de Oracle por su cuenta, pero tiene ciertas limitaciones como pueden ser el tamaño de las instancias y los paquetes de administración de la base de datos que no se pueden usar. La que tenemos adquirida es con la segunda opción, también con Multi-AZ.

Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Implementación	Rendimiento de la red	Número de instancias
db.m5.4xlarge	7,856 USD	32	128 GiB	Multi-AZ	Hasta 10 gigabits	1

Figura 13. Instancia AWS RDS db.m5.4xlarge.

Coste anual = 7,856 USD/h x 24 h x 365 días/año = 68.818,56 USD/año

Las instancias db.m5.4xlarge son las instancias más grandes que se pueden obtener bajo el contrato de licencia incluida.

El coste total de las dos instancias es:

Coste Total = 52.998 USD + 68.818,56 USD = 121.816,56 USD/año

DMS:

AWS DMS se factura como si de una instancia de EC2 se tratara, sin embargo, no se tratan de instancias de propósito general sino orientadas al cómputo (instancias de tipo c) o a memoria elevada (instancias de tipo r). Sefacturan exactamente igual que si fuera una Ec2.

Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Implementación	Rendimiento de la red	Número de instancias
c5.4xlarge	2,808 USD	32	128 GiB	Multi-AZ	Hasta 10 gigabits	1

Figura 14. Instancia AWS EC2 c5.4xlarge.

Si el la instancia de replicación DMS actúa sobre una base de datos RDS con Multi-AZ es recomendable usar también la instancia con Multi-AZ, esta particularidad difiere de una instancia EC2 normal.

Coste anual = 2,808 USD/h x 24 h x 365 días/año = 24.598,08 USD/año

Redis:

El nombre del servivio es AWS Elasticache for Redis. Esta instancia tiene la particularidad que está optimizada para el uso de memoria.

Nombre de la instancia	Tarifa por hora bajo demanda	vCPU	Memoria	Uso	Rendimiento de la red	Número de instancias
cache.r5.2xlarge	0,961 USD	8	52,82 GiB	Optimizada para memoria	Hasta 10 gigabits	1

Figura 15. Instancia AWS Redis cache.r5.2xlarge.

Coste anual = 0,961 USD/h x 24 h x 365 días/año = 8.418,36 USD/año

Load Balancers:

También son un producto de Elastic que Amazon AWS ha adaptado e integrado como si de un producto suyo se tratara. AWS dispone de balanceadores de distintos tipos, de aplicación, de red, de puerta de enlace (Gateway) y balanceadores llamado clásicos pero que se están deprecando y son sustituidos por los de aplicación.

En nuestro caso se trata de ALB (Apllication Load Balancer) que mediante la selección de Target Groups podemos ir creando clusters e ir añadiendo máquinas EC2 para su balanceo de carga.

La facturación de los balanceadores de carga tiene un componente fijo que se factura por horas de utilización y un componente que depende del tráfico. AWS nombra un concepto que son los LCU (Unidad de Capacidad del Balanceador) que pueden ser por el número de nuevas conexiones por minuto, por conexiones activas en el balanceador, por bytes procesados y por el número de reglas aplicadas. Esto hace que el coste de los balanceadores varíe mes a mes debido al ajuste de los LCU y que no todos los balanceadores cuesten lo mismo porque irá en función del tráfico que soporten.

En concreto tienen un coste de 0,0252 USD/h más 0,008 USD por cada LCU máxima, esto es, el número de LCU máximo que alcanza el balanceador en una hora de todos los conceptos que suponen LCU. Como es un concepto muy cambiante y depende de cada balanceador en este supuesto se hará una media de los LCU de todos los balanceadores para calcular el coste.

En total son 47 balanceadores, uno por aplicación que necesita se accedida para simplificar la gestión, y tienen una media de 6 LCU por hora cada uno, así el coste total será:

Coste anual fijo= 47 x 0,0252 USD/h x 24 h x 365 días/año = 10.375,344 USD/año

Coste anual LCU = 47 x 0,008 USD/h x 24 h x 365 días/año = 3.293,76 USD/año

Coste total= 10.375,344 + 3.293,76 = 13.669,104 USD/año

WAF:

El servicio básico de AWS WAF es bastante asequible ya que se factura a razón de 5,00 USD al mes por cada web, 1,00 USD por regla administrativa al mes y 0,60 USD al mes por millón de solicitudes. Sin embargo, al poner muchas reglas administrativas que condicionan el acceso (se permite el acceso a las IPs autorizadas de los metabuscadores y algunos accesos públicos) y tener muchas peticiones de disponibilidad, resulta tener un coste considerable por mantener un sistema casi cerrado al tráfico de Internet.

Así tenemos 47 webs de 47 aplicaciones, unas 2000 reglas administrativas en total y unos 3000 millones de solicitudes al mes en total.

El coste total será:

Cargos por web = 47 webs x 5,00 UDS/web x 12 meses/año = 2.820 USD

Cargos por reglas = 2000 reglas x 1,00 USD/regla x 12 meses/año = 24.000 USD

Cargos por tráfico = 3000 millones x 0,60 USD por millón de solicitudes x 12 = 21.600 USD

Coste total WAF = 2.820 + 24.000 + 21.600 = 48.420 USD/año

Sumando todos los conceptos de la Zona Integraciones tenemos un coste total anual de 497.180,78 USD.

La siguiente tabla es un resumen de los costes que hemos ido viendo hasta ahora para esta zona.

EC2	RDS	DMS	REDIS	ALB	WAF	TOTAL
280.258,68	121.816,56	24.598,08	8.418,36	13.669,10	48.420,00	497.180,78

Figura 16. Resumen costes totales.

2.4. Motivación para un plan de migración

Como ya se habrá intuido durante todo este punto 2 en el que se describe el escenario, el principal motivo para la presente migración que es motivo de este TFM es un motivo principalmente económico.

En principio, ninguna de las zonas expuestas tiene a priori problemas para dimensionarse.

La Zona Mainframe, debido a que estos servidores tienen mucha capacidad de cálculo, tiene la limitación muy lejos del nivel de cálculo exigido por nuestra empresa; por decirlo de otro modo, la capacidad de cálculo del mainframe es muy superior a la que necesitados, de hecho, el mainframe está compartido por muchas empresas e instituciones y el propio servidor es constantemente ampliado.

La Zona Webservers tampoco presenta límites de capacidad y a priori no es necesario ningún sistema de autoescalado porque, debido a que hay muy pocos servidores y la diferencia entre el uso máximo y mínimo (de cpu o de memoria) es muy pequeña, deben estar siempre encendidos.

El problema principal de esta zona en la poca versatilidad y poca movilidad que tienen los sistemas. Cualquier cambio profundo en la zona, por ejemplo, un servidor nuevo (ya sea de servidores Apache o Jboss), requiere una planificación de meses por parte de IBM, firmas de contratos, pruebas tanto por parte de IBM como del cliente, etc. Resulta un proceso tedioso y largo que impide realizar cambios de infraestructura con la soltura y dinamismo que los nuevos sistemas de hoy en día tienen (autoescalado, arranques automáticos, kubernetes, etc).

La Zona Integraciones tampoco presenta límites de capacidad, la migración de las aplicaciones de debe a un tema de estandarización de la infraestructura para reducir la gestión y el mantenimiento. Aprovechando la migración de las otras dos zonas, también se procederá a la migración que esta zona para usar los mismos recursos que se usarán para las otras dos zonas y conseguir un acercamiento de todos los servicios y servidores para reducir latencias y recursos.

Así pues, el motivo principal para la migración es el alto coste de mantenimiento del gestor de reservas. Después los motivos secundarios son la poca flexibilidad que tenemos en la zona webservers y la poco nivel de estandarización que tenemos en la Zona Integraciones donde tenemos cuarenta y siete aplicaciones distintas. Ya que se van a migrar estas tres zonas se requiere un sistema económico, fácil de mantener y con adaptación a la carga.

3. Estudio de soluciones para la migración

3.1. Zona Mainframe

Como ya se ha visto hasta ahora, el gestor de reservas presenta unas cuantas singularidades que se resumen es:

- Código extenso, 15M de líneas.
- Programado en Cobol
- Base de datos en local
- Imposibilidad de parada de servicio por periodos prolongados.

De todos estos puntos, el más complicado de salvar es el tema de la base de datos local en el mainframe como veremos a continuación.

La migración del sistema gestor de reservas se planteó desde un principio hacia un sistema en Cloud. Por ello se pidieron presupuestos y ensayos a grandes empresas, que ya eran proveedores de nuestra compañía, para que nos ayudaran.

1. Microsoft:
   - azure.microsoft.com
   - Plataforma Cloud: Microsoft Azure
   - Sistema Operativo de los servidores: Windows Server 2016
   - Base de Datos: SQL Server 2017
   - Precio estimado para la migración: 5.000.000 euros más el mantenimiento mensual de los recursos del cloud.

     Microsoft fue el proveedor que más posibilidades tenía de realizar la migración. Los asesores de Microsoft realizaron pruebas en un servidor local con programa de Cobol transformado en Java. Inicialmente las pruebas fueron satisfactorias, pero al incorporar datos reales en la base de datos, las pruebas arrojaban unos tiempos pésimos. Solamente se lograban tiempos medianamente aceptables si incluían la base de datos en el mismo servidor donde estaba el programa con lo que las consultas se realizaban en local. Al separar la base de datos la latencia de red hacía imposible cualquier consulta en un tiempo aceptable. La opción de juntar base de datos en el mismo servidor tampoco era una opción ya que al incrementar el volumen de consultas se necesitaba un servidor mucho más potente del que se disponía en Azure. La realización de clústers tampoco era una opción ya que representaba tener muchos nodos con

base de datos replicándose una cantidad inmensa de datos continuamente y saturaban el SQL Server.
Al final, Microsoft admitió que no podía realizar el proyecto y desistió.

2. Google:
   - cloud.google.com
   - Plataforma Cloud: Google Cloud
   - Sistema Operativo de los servidores: Red Hat Enterprice Linux
   - Base de Datos: SQL Server 2017
   - Precio estimado para la migración: -

   Tras estudiar los datos y las pruebas que Microsoft había realizado, los asesores de Google Cloud ni siquiera llegaron a presentar ninguna propuesta aludiendo que su cloud, y sus características, eran muy similares al de Microsoft y no tenían otro enfoque para el proyecto. Así que básicamente se limitarían a repetir las pruebas con muy probable igual resultado. Por ello desistieron del proyecto antes de proponer una propuesta económica.

3. Amazon AWS:
   - aws.amazon.com
   - Plataforma Cloud: Amazon AWS
   - Sistema Operativo de los servidores: Amazon Linux 2
   - Base de Datos: MySQL
   - Precio estimado para la migración: -

   Amazon AWS no tiene entre su personal, especialistas dedicados a la asesoría de empresas ni técnicos dedicados a éstas, sí dispone de asesores para sus clientes a modo de orientación en la utilización de sus productos, pero no dedicarán tiempo ni recursos a ninguna operativa como la que teníamos en mente. No obstante, Amazon nos puso en contacto con varios colaboradores suyos, que declinaron el proyecto sin llegar a entrar en detalles.

4. IBM:
   - www.ibm.com
   - Plataforma Cloud: IBM Cloud
   - Sistema Operativo de los servidores: Red Hat Enterprice Linux (RHEL)
   - Base de Datos: MySQL
   - Precio estimado para la migración: 4.000.000 euros más el mantenimiento mensual de los recursos del cloud.

IBM realizó su propuesta de transformación de Sirius a su cloud conociendo bastante bien la infraestructura y el volumen de datos que manejaba el gestor de reservas. Por ello, mi empresa pensaba que tendría un enfoque distinto y que creía que IBM tenía una solución al problema. Esto ocurrió a medias, la solución propuesta no era sacar Sirius del mainframe sino trasladar el gestor de reservas de un mainframe físico a un mainframe cloud. Esta solución no convenció mucho a los gerentes de mi empresa debido básicamente a que por muy cloud que sea, Sirius seguía estando en un mainframe, programado en Cobol y con la base de datos localmente en el servidor. Si embargo IBM realizó pruebas con el mismo código recompilado en Java que había probado Microsoft, viendo que, con el código en Java, el programa no era rápido. IBM terminó por retirarse el proyecto puesto que no tenía nada más que ofrecer.

5. Blue Age:
   - www.bluage.com
   - Plataforma Cloud: Amazon AWS
   - Sistema Operativo de los servidores: Amazon Linux 2
   - Base de Datos: MySQL
   - Precio estimado para la migración: 20.000.000 euros con un pago inicial de 1.000.000 euros a fondo perdido para realizar el estudio de viabilidad.

     La empresa Blue Age está especializada en migraciones de mainframe a cluod. De hecho, en su página web tienen colgado un vídeo sobre la transformación de Sirius a AWS Mainframe (https://www.bluage.com/site_data/video/bluage_rex_credit_logement_ensub_f1_opt.mp4).
     Después de inspeccionar las pruebas que Microsoft e IBM habían realizado, pidieron 1.000.000 euros a fondo perdido para realizar el estudio de viabilidad, que estimaban poder realizar por 20.000.000 euros en un plazo de dos años. En reuniones posteriores reconocieron que era muy poco probable que pudieran cumplir los plazos ni que el resultado fuera exitoso. Ante esta perspectiva, mi empresa decidió abandonar el proyecto con Blue Age.

Tras los fracasos en los proyectos de migración mi empresa se tuvo que replantear la última opción que quedaba. No migrar Sirius sino realizar otro Sirius desde cero, programado directamente en Java y orientado a microservicios, escalable (autoescalado sería lo ideal) y contenizable (mejor sobre AWS EKS que sobre otros sistemas).

El proyecto para el nuevo gestor de reservas tendría los siguientes recursos asignados:

- Equipo humano:
  - 1 Arquitecto de soluciones informáticas
  - 2 desarrolladores senior en Java
  - 3 desarrolladores junior en Java
  - 2 desarrolladores senior en Cobol, con experiencia en Sirius
  - 1 QA, pruebas de software
  - 1 Scrum Master

- Equipo material:
  - Servidor de AWS RDS Mysql, aunque se hicieron pruebas con Aurora y PostgreSQL.
  - AWS MemoryDB for Redis
  - Servidor de pruebas, AWS EC2.
  - Servidor de pruebas AWS EC2 con servidor de Zipkin instalado como gestor de trazas y de log distribuido.

- Duración del proyecto: 2 años.

- Partida presupuestaria asignada: 600.000 euros

3.2. Zona Webservers

Primero de todo, exponemos un resumen de las características de la zona:

- Gestión IBM
- Servidores dedicados RHEL
- Servidor de aplicaciones Red Hat Jboss EAP 7.2
- Aplicación en Java

Vistas las características mayoritarias de la zona podemos ver que es fácilmente exportable a otro entorno ya que la aplicación en java permite ser puesta en servicio en otros servidores de aplicaciones sin muchos problemas. Algunas de las propuestas de migración van en ese sentido. Veamos las propuestas que se valoraron y por cual se decantó.

1. AWS AppConfig + AWS EC2:

AWS AppConfig facilita a los clientes la configuración, validación e implementación rápida y segura de indicadores de características y configuración de aplicaciones. AWS AppConfig, puede realizar cambios en las aplicaciones de forma segura, evitar errores en los cambios de configuración, implementar cambios en un conjunto de objetivos rápidamente y controlar la implementación de cambios en sus aplicaciones.
La aplicación podría estar en máquinas EC2 sobre un Apache Tomcat, que sustituye al Jboss, y solamente habría que adaptar la aplicación en formato .war para introducir todas las librerías que necesita. Jboss incorpora muchas librerías que la aplicación usa, y usar Apache Tomcat (que es el servidor de aplicación básico) requiere que metamos todas esas librerías en la aplicación. De hecho, Jboss que un conjunto de servidores ya que incorpora Apache como servidor web, Apache tomcat para contenido java, incorpora librerías para conexiones SSL, etc.

El motivo por el que se descartó esta opción fue que AppConfig no se adapta a nuestra web y se deberían hacer cambios muy profundos para conseguir gestionarla con AppConfig.
Además, el hecho de que haya que introducir librerías en la aplicación, resulta tedioso. No por el echo de meterlas en sí, sino porque no podrían ser de la misma versión de las que usa el Jboss y muchas entrarían en conflicto, con dependencias unas de otras, librerías ya deprecadas, etc.

2. AWS EC2:

Similar opción que la anterior alternativa, pero sin usar AppConfig. Se descartó porque adaptar las librerías de Jboss para que pudieran usarse en Apache Tomcat suponía hacer un esfuerzo extra que el equipo no tenía.
Al no haber partida presupuestaria no se podían destinar recursos a este fin con lo que se descartó esta opción.

3. AWS EKS + Docker con Wildfly:

Wildfly es la solución Community (mantenida por la comunidad de usuarios y es gratuita) de Jboss EAP. La solución pasa por no modificar nada de la aplicación y empaquetarla en un Wildfly que ira en un contenedor Docker. Se desplegarán tantos contenedores (pods) como servidores Jboss existen en la Zona Webservers.

El sistema general de gestión de contenedores será AWS EKS (Elastic Kubernetes Service).

Esta solución es la que se adoptará por varios motivos:

- Permite realizar la migración al mismo tiempo minimizando las modificaciones de software en el aplicación de la web de reservas.
- Wildfly es gratuito y su estructura es similar a la de Jboss por lo que los cambios de configuración son mínimos.
- El cambio a AWS EKS permitirá homogeneizar la web con las demás aplicaciones ya que EKS será el estándar a partir de la migración.

3.3. Zona Integraciones

En esta zona no se planteó otra posibilidad que kubernetizar todas las aplicaciones. Algunas aplicaciones se pasaron a Spring Boot (que ya lleva el servidor de aplicación integrado) con los que el contenedor Docker era aún más sencillo. Las aplicaciones más complejas se pasaron tal cual a un contenedor Docker con un Apache Tomcat instalado.

Por lo tanto, la única alternativa pasa por usar AWS EKS para todas las aplicaciones de la Zona Integraciones.

3.4. Monitorización y gestión de alarmas

Ya se ha comentado que la estandarización es importante porque reduce complejidad y recursos. La monitorización y la gestión de alarmas también debe estandarizarse, es por ello por lo que se adapta a AWS EKS el servidor Zabbix que ya existe, por lo que muchas de las alarmas pueden ser exportadas/importadas al nuevo servidor.

Recordamos que Zabbix es un gestor de alarmas gratuito y desde su página web oficial existen ya creadas imágenes de Docker listas para usarse con simplemente cambiar algunos parámetros y con algunas posibilidades en cuanto a bases de datos, nosotros siempre usaremos MySQL por tenemos todas las bases de datos con este motor, por lo tanto, nuestra opción será Zabbix Server con MySQL.

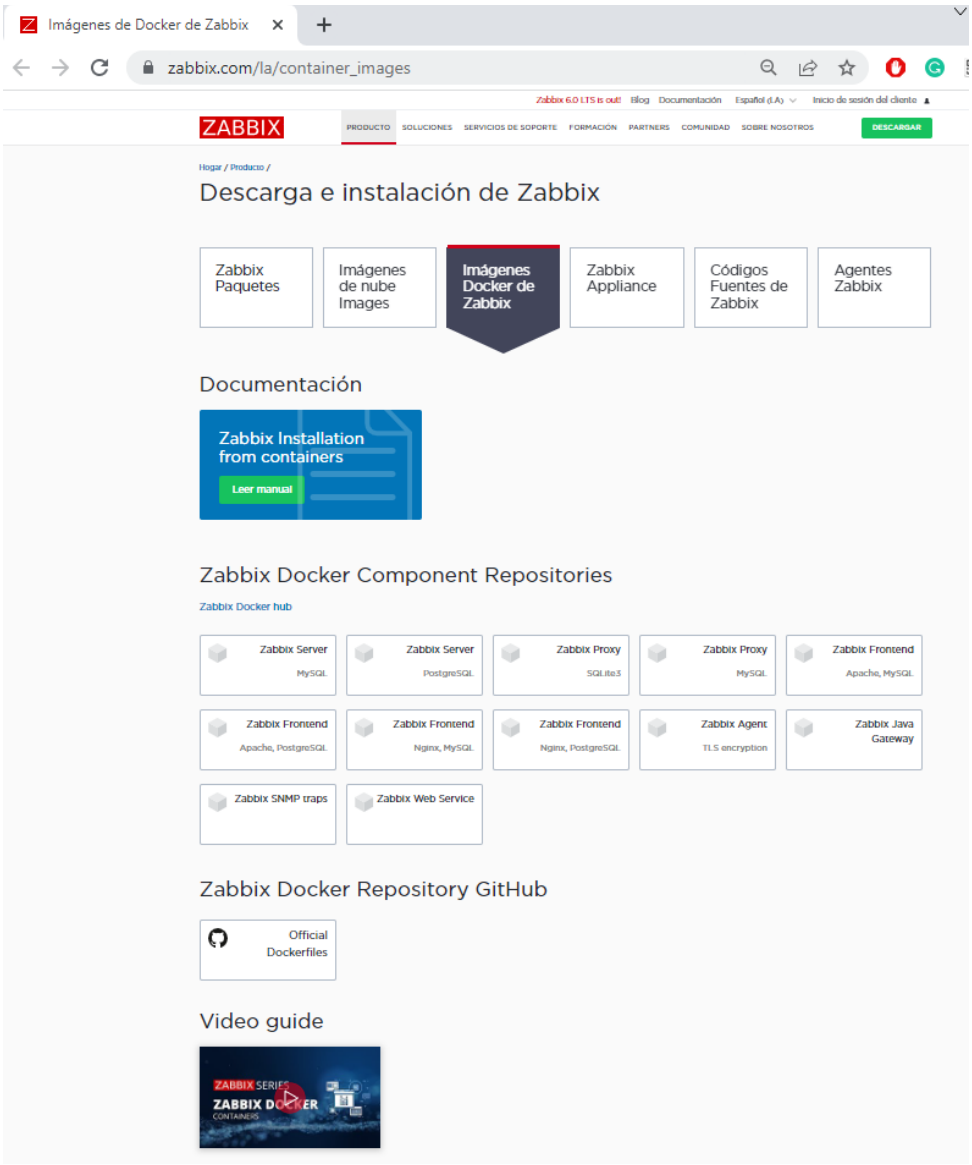

Figura 17. Página web de Zabbix.

Tal como ya se ha explicado en el punto 2.3.2 Las alarmas, aparte de salir en la web de Zabbix se envían a un canal de Teams donde las personas encargadas del soporte son avisados, también se reciben en un canal específico para el personal de guardia 24x7.

4. Solución final para la migración

4.1. Configuración general

4.1.1 AWS EKS

Ya se ha comentado que la solución final de todas las zonas es una migración de todas las aplicaciones a AWS EKS. En este apartado se explicará la estructura y organización que se adoptará de forma general. Puesto que el nuevo gestor de reservas, llamado ahora Vega, como las aplicaciones de la web corporativa y la aplicación web del callcenter y todas las aplicaciones de la zona de integraciones se empaquetan en contenedores, solo es necesario un único clúster para gestionarlas ya que eso reduce complejidad y recursos.

En todo momento se ha hablado del entorno productivo, sin embargo, no se debe olvidar que siempre se necesita un entorno pre-productivo lo más similar al productivo posible. Es por ello que se han diseñado dos clústers de kubernetes (aws eks) iguales en esencia, aunque puede que los servidores (workers) no sean iguales debido a las diferencias de potencia de un clúster y otro.

También debemos recordar que la Zona Webservers PCI se encuentra en otro clúster de otra cuenta de AWS separada. Esto es para poder definir accesos bien diferenciados a las personas que tienen acceso a la Zona PCI, en el que solo algunos administradores deben poder acceder.

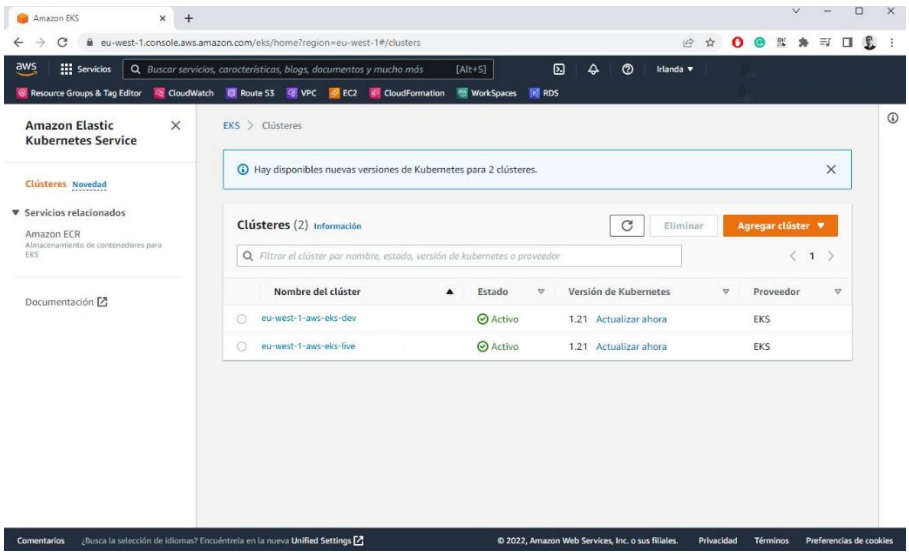

Figura 18. Cúsers de AWS EKS.

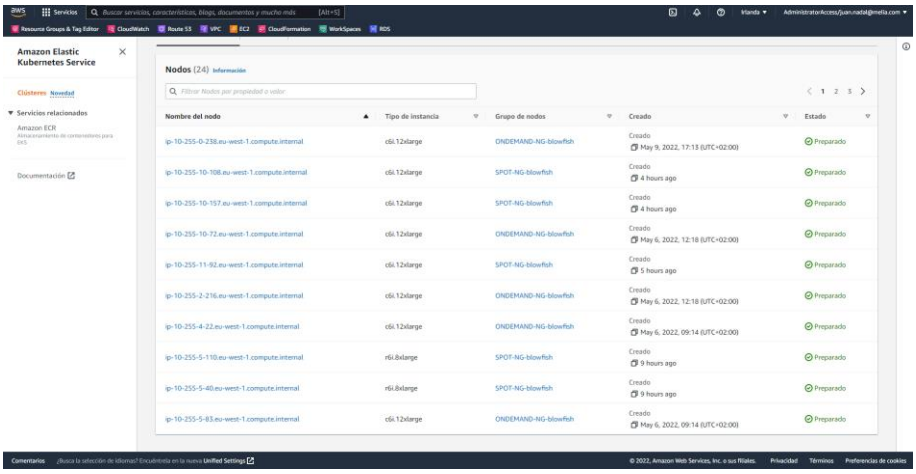

Figura 19. Nodos del cúster AWS EKS Live.

El cluster productivo (live) está formado por dos node-group ONDEMAND-NG-blowfish y SPOT-NG-blowfish (-blowfish es una notación por defecto de Terraform). Los node-group son agrupaciones de nodos para los que se definen unas políticas, por ejemplo, de autoescalado, seguridad, etc.

En las imágenes siguientes se pueden apreciar algunos detalles del node-group ONDEMAND-NG-blowfish. Por ejemplo, que está compuesto por 8 nodos (según la imagen) de tipo c6i.12xlarge (48 vcpu, 96 GiB de memoria y un ancho de banda de red de 18,75 Mbps) y que están todos activos, que pertenecen a un grupo de autoescalado donde el mínimo son 8 nodos y el máximo 12, por lo tanto, en el momento de realizar la captura el node-group estaba con los nodos mínimos.

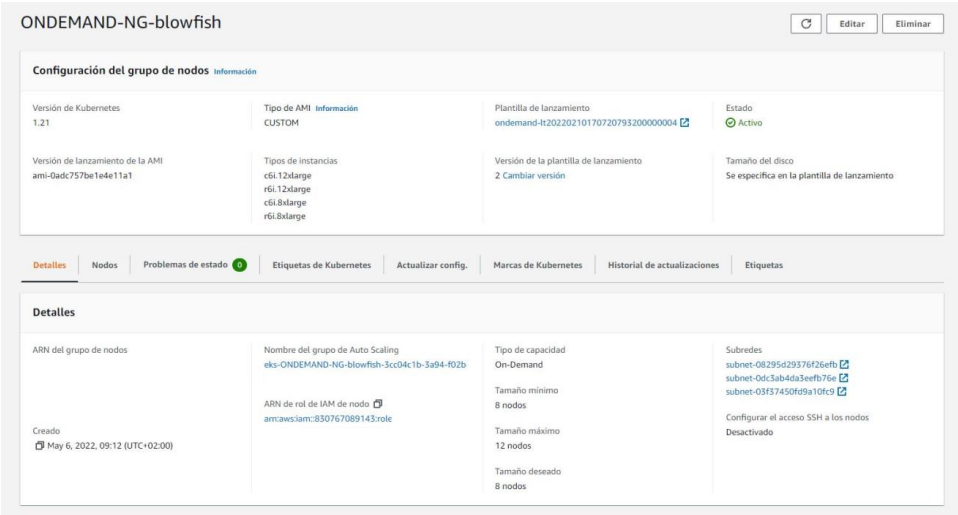

Figura 20. Configuración del node-group ONDEMAND.

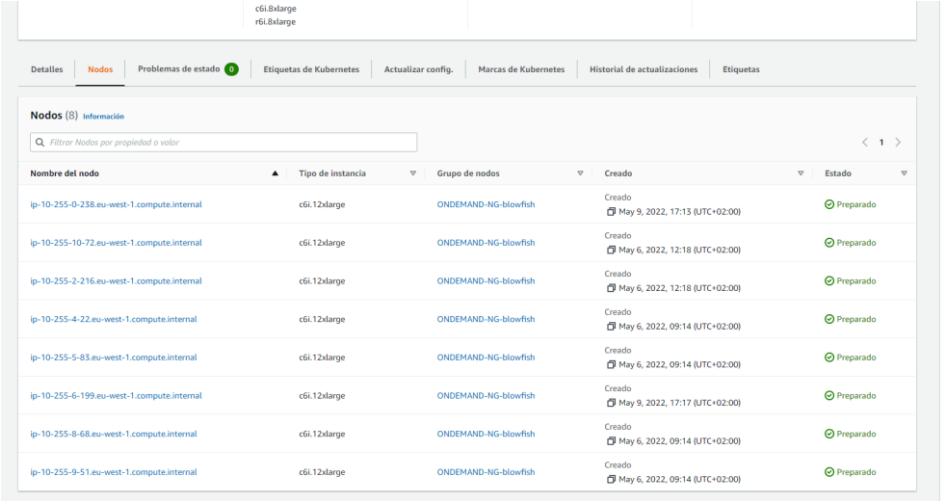

Figura 21. Nodos del node-group ONDEMAND.

La diferencia de los node-group de ONDEMAND a SPOT es que en el de ondemand las instancias están reservadas y en el de SPOT usa instancias spot. Las spot son unas instancias iguales, pero mucho más baratas pero que se pagan conforme a un mercado de subastas, eso significa que en cualquier momento algún cliente puede pagar más dinero por esa instancia y puede desaparecer de tu cuenta. AWS avisa unos minutos antes y existen scripts para cuando se recibe un aviso de que una instancia va a desaparecer, comprar otra, o bien más cara o de otro tipo distinto. En cualquier caso significa sobredimensionar un poco el clúster para que no sea una cuestión crítica.

Las aplicaciones más críticas se encuentran en el node-group ONDEMAND, porque no podemos permitirnos que una aplicación se quede sin sitio donde EKS pueda colocar el pod. Sin embargo, tenemos aplicaciones que incluso pueden estar varias horas sin funcionar, es el caso de aplicaciones PUSH en el que enviarán las disponibilidades cuando la aplicación vuelva a estar disponible, en estos casos se situarán en el node-group SPOT. Ya hemos dicho que las aplicaciones PCI estaban en otra cuenta, pero la configuración del cluster es similar, solo que no existe el node-group SPOT, son solamente un node-group con de 3 a 6 máquinas, más pequeñas y son ondemand porque son muy críticas porque de ellas dependen los pagos de las reservas.

En la siguiente imagen podemos ver que el node-group de spots está degradado. Nada está mal, simplemente, no ha podido encontrar ninguna máquina spot del tipo de instancia predeterminado y ha arrancado una instancia de otro tipo.

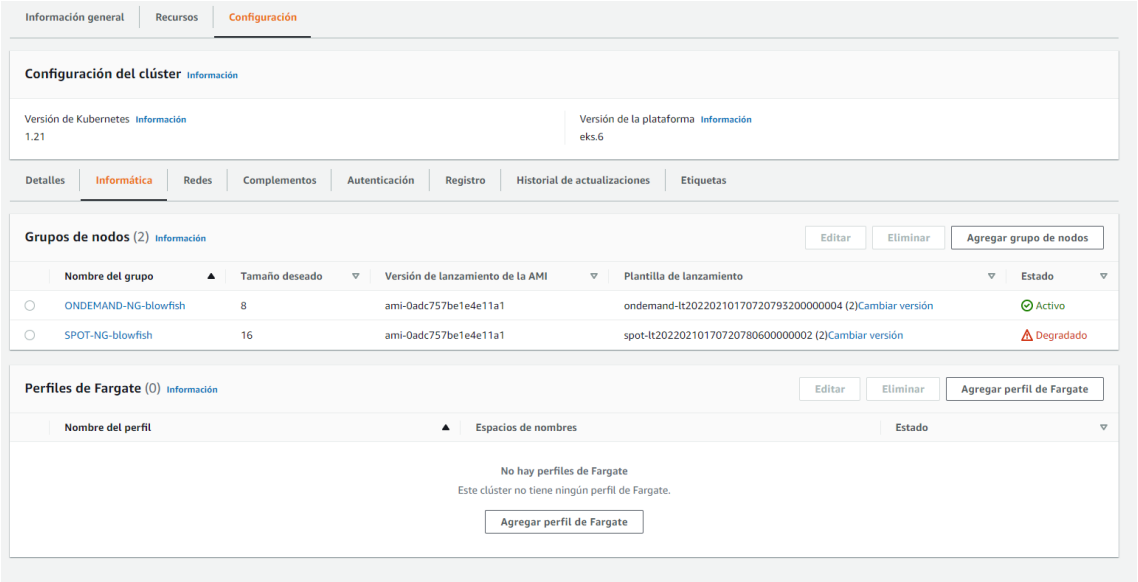

Figura 22. Configuración del clúster AWS EKS Live.

En las imágenes siguientes vemos que el node-group tiene 16 instancias preparadas y correctas, que es el mínimo que debe tener este node-group con máximo de 20 nodos/instancias.

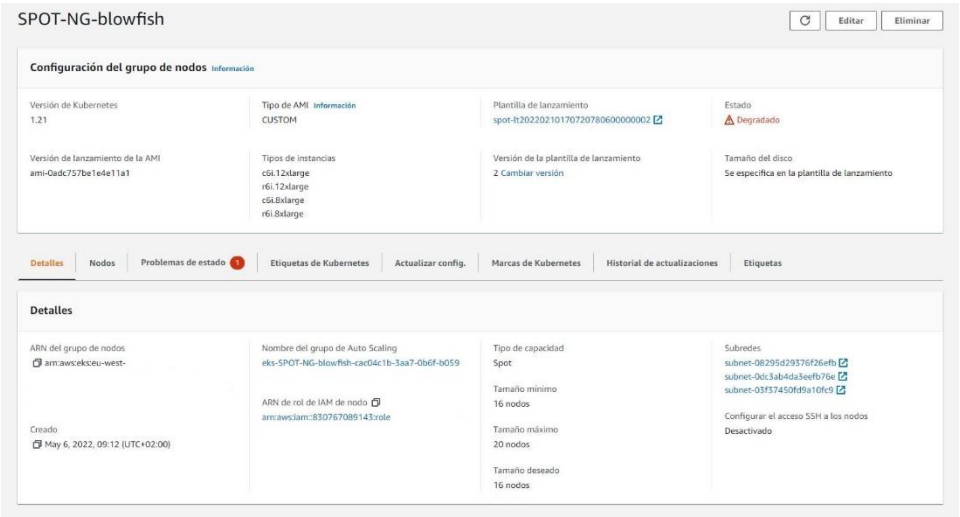

Figura 23. Configuración del node-group SPOT.

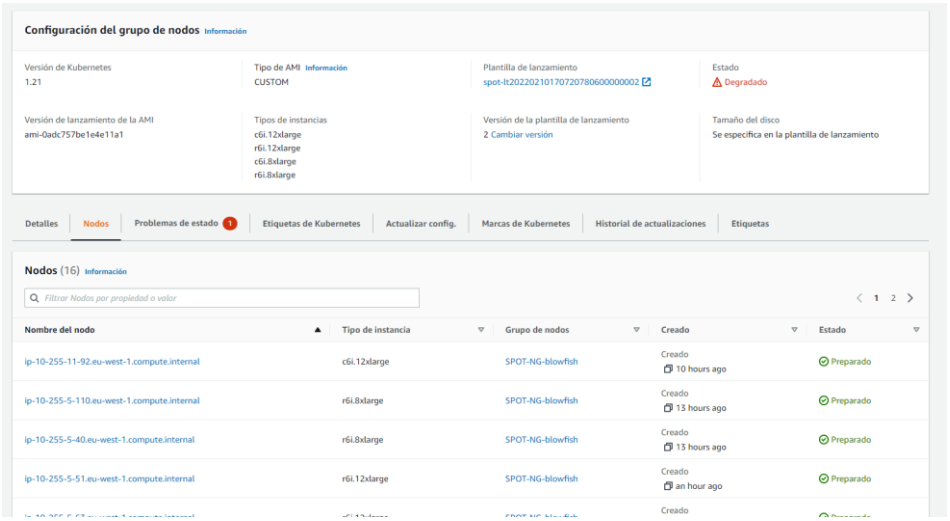

Figura 24. Nodos del node-group SPOT.

Hay que decir que estos nodos no son gestionables directamente por nosotros, sino que es el propio eks quien los configura y prepara. Cada nodo puede contener de cero a n pods y un pod es la unidad mínima gestionable, es decir un contenedor. EKS se encarga también de la gestión de los pods en los nodos, aunque podemos configurar algunas cosas, por ejemplo, que reparta los pods de cierta aplicación tanto como pueda por los nodos, para que no se agrupen muchos en el mismo y si este cae o se reinicia suponga un grave problema, al mismo tiempo se pueden crear asociaciones de pods de distintas aplicaciones que deben estar en el mismo nodo, etc.

En cuanto a la cpu y memoria de cada pod se configura por aplicación. En cada aplicación se debe especificar el número de vcpu y de memoria que necesita cada aplicación y el número de pods que requerirá esa misma aplicación. EKS se encarga de gestionar la cpu, memoria y pods automáticamente.

En la siguiente imagen podemos ver un ejemplo de configuración de la aplicación. En ella se puede ver el número de réplicas (pods) la configuración de memoria y cpu y el node-group al que irá la aplicación.

Otra cosa que se ve en la figura es la imagen, se aprecia que se usa el sistema de AWS ECR (Elastic Container Registry), que no es más que un repositorio de contenedores, donde se almacenan todas las versiones de las imágenes que se van generando. Este punto se explicará en breve.

La figura se trata de un fichero .yaml que contiene la configuración de la aplicación y que se aplica directamente sobre el clúster de eks. Esto es debido a que el fichero .yaml está almacenado en un repositorio de Git, en nuestro caso Bitbucket, que al modificarse produce un cambio en ArgoCD que lo traslada al clúster de EKS.

```yaml
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: smartxmlpull
  labels:
    app: smartxmlpull
spec:
  replicas: 15
  template:
    metadata:
      labels:
        app: smartxmlpull
        version: latest
    spec:
      containers:
      - name: smartxmlpull
        image: 1256985214.dkr.ecr.eu-west-1.amazonaws.com/smartxmlpull:master.latest
        env:
        - name: JAVA_OPTS
          value: -XX:MaxRAMPercentage=70.0 -Dspring.profiles.active=PRO
        resources:
          limits:
            memory: 18Gi
            cpu: 8
          requests:
            memory: 18Gi
            cpu: 8
      nodeSelector:
        eks.amazonaws.com/capacityType: ON_DEMAND
```

Figura 25. Plantilla de configuración de aplicación.

ArgoCD es una herramienta de despliegue continuo que se puede instalar en un clúster de Kubernetes y puede desplegar aplicaciones en el mismo clúster o en otros clústers, permitiendo así gestionar de forma centralizada y automática el despliegue de software. Se puede configurar de tal forma que este siempre chequeando un repositorio de Git (como lo tenemos nosotros) y ante cualquier cambio en los ficheros, redesplegar las aplicaciones o aplicar los cambios necesarios.

En la siguiente imagen, la web de ArgoCD con algunas de las aplicaciones desplegadas.

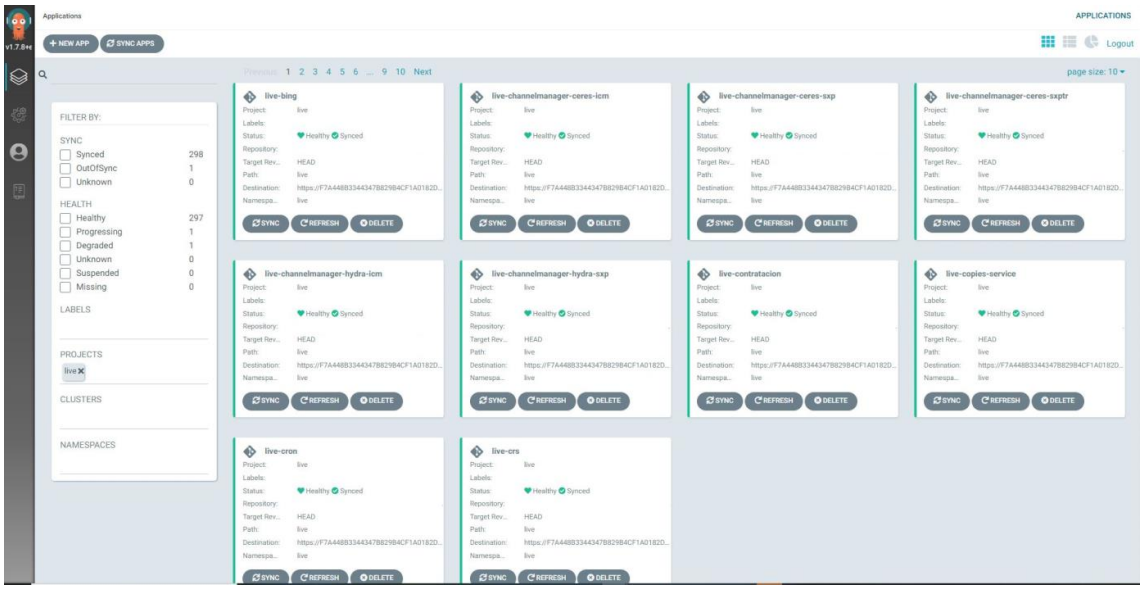

Figura 26. Pácgina web de ArgoCD con aplicaciones.

En la imagen siguiente el detalle de una aplicación.

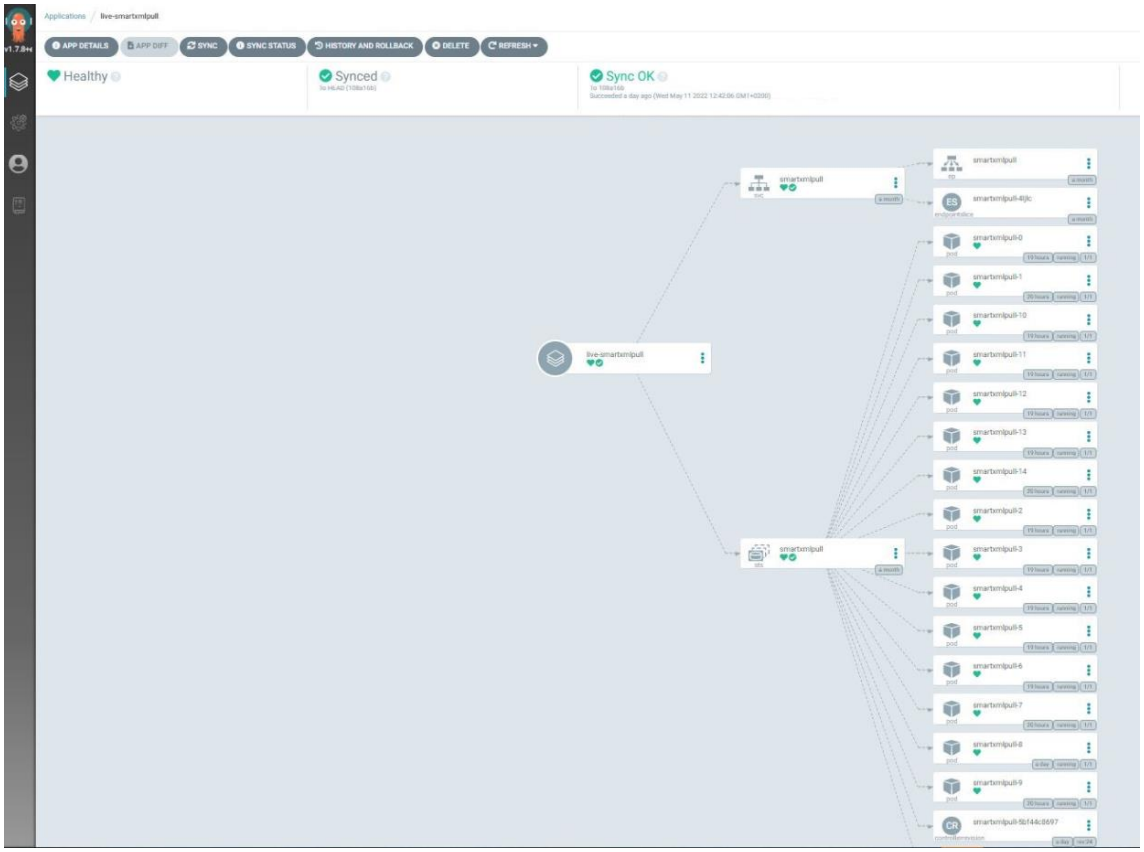

Figura 27. Vista de aplicación de ArgoCD.

En la imagen anterior vemos el estado de la aplicación que está sincronizada con el Git, es decir, que tiene aplicados todos los cambios de la aplicación, se aprecian todos los nodos que están en estado healthy y el servicio también es correcto.

En cuanto a infraestructura de cada aplicación se refiere, accesos a base de datos, dns, configuración del balanceador, etc., se hace todo con plantillas de Terraform. Se ha desplegado un servidor de Jenkins, que es un servidor opensource para la integración continua, que es el encargado de gestionar los cambios.

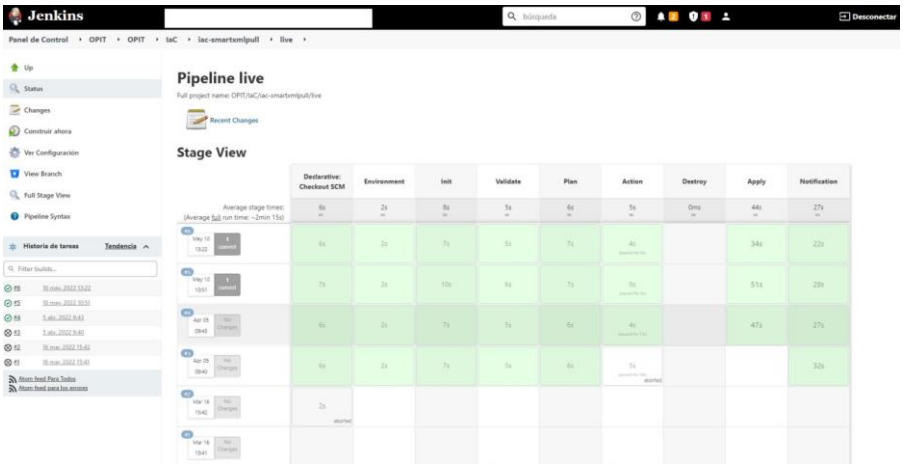

Figura 28. Pipeline de Jenkins para aplicar configuración.

Dicho servidor de Jenkins, que es un contenedor oficial, por lo que la configuración es mínima, también tiene instalado Terraform para la traducción de las plantillas en recursos de AWS.

La imagen siguiente pertenece a una plantilla para smartxmlpull de terraform para la creación de su vhost en el balanceador ingress.

Una vez creada la plantilla, Jenkins reconoce que se ha modificado la rama que se le ha configurado escuchar y ejecuta el pipeline que se puede ver en la anterior imagen en el que Terraform analiza la plantilla y la translada a AWS. Los pasos están bien marcados en el pipeline en el que por ejemplo, “Validate” es el paso en el que Terrraform analiza la sintaxis de la plantilla por si existen errores, en “Action” el pipeline se detiene unos minutos para que el operador pueda ver las acciones que se realizarán o los cambios y cancelar el proceso si es necesario, en “Apply” Terraform aplica los cambios y posteriormente se notifica en un canal de Teams específico.

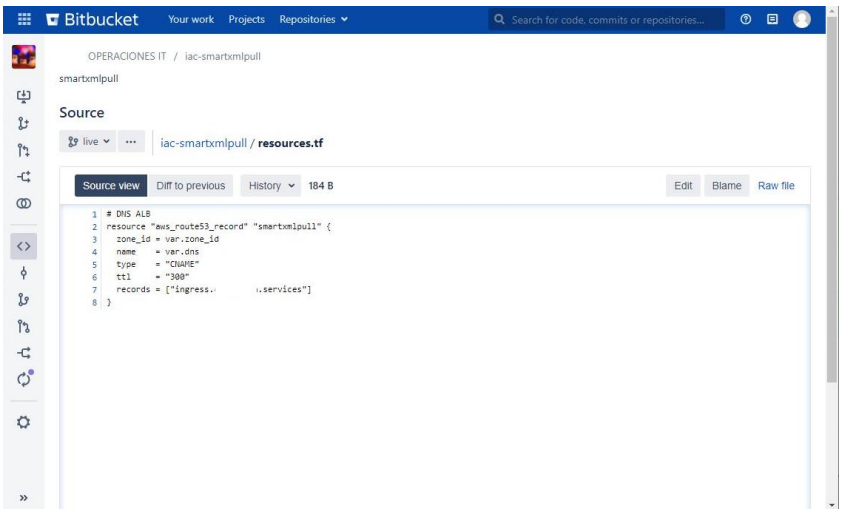

Figura 29. Plantilla de Terraform para configuración de DNS de aplicación.

Llegados a este punto hay que explicar que un Ingress es un balanceador de carga específico para Kubernetes, internamente AWS arranca un AWS ALB, que ya hemos hablado de ellos. En la imagen siguiente vemos la estructura que sigue un clúster de Kubernetes (en AWS EKS es lo mismo):

Figura 30. Esquema Kubernetes.

Entonces lo que hemos visto en la plantilla de Terraform anterior es simplemente para que cree un registro dns que apunte al balanceador Ingress con un CNAME. En la configuración del Ingress tendremos un contexto /smartxmlpull para que desvíe todo el tráfico que proviene desde fuera con ese contexto a los pods de la aplicación.

En la imagen siguiente vemos la configuración del ingress con varios contextos/aplicaciones. Se puede apreciar también que hay una configuración de certificado almacenado en AWS ACM (AWS Certificate Manager) y algunas otras configuraciones de red interna, logs, entorno, …

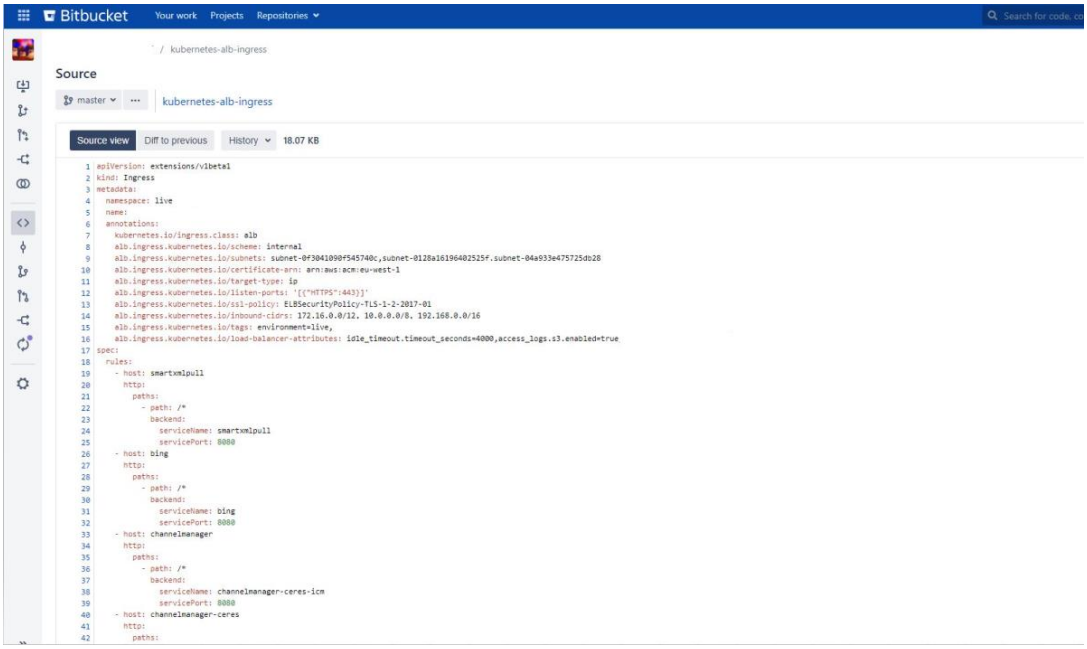

Figura 31. Plantilla de configuración de Ingress ALB.

4.1.2 AWS WAF

Según la definición que nos ofrece Amazon de su sistema, AWS WAF (Web Application Firewall) es un firewall para aplicaciones web que ayuda a proteger las aplicaciones web o API contra ataques web y bots comunes que pueden afectar la disponibilidad, poner en riesgo la seguridad o consumir demasiados recursos.

AWS WAF permite crear reglas de seguridad que controlan el tráfico de bots, bloquean los patrones de ataque comunes, como la inyección de SQL o el scripting entre sitios. También se puede personalizar las reglas que filtran patrones de tráfico específicos.

En nuestro caso, y debido a las muchas peticiones, AWS recomienda implementar AWS WAF en Amazon CloudFront como parte de la solución CDN, y con el balanceador de carga de aplicaciones que se encuentra delante de sus servidores web, AWS EKS o servidores de origen ejecutados en EC2.

La cuestión de usar un CDN, en nuestros caso AWS CloudFront, es un tema de rendimiento. Parece ser que AWS WAF es un servicio global y como a nuestros sistemas deben acceder desde todo el mundo, es mejor utilizar el sistema de AWS CloudFront para acercar el contenido a los clientes y viceversa, incluso aunque no usemos contenido estático almacenado en él. El motivo es que el tráfico entre un nodo lejano de AWS CloudFront y nuestros balanceadores se realiza a través del tráfico interno de AWS, que es mucho más rápido y óptimo que si el tráfico fuera por Internet. Además, AWS CloudFront ya cuenta con funcionalidades anti DDoS con lo que añade un extra de protección al sistema.

El esquema siguiente resumen las anteriores palabras:

Figura 32. Esquema AWS WAF.

En el sistema WAF se han creado dos Web ACL, que son como dos casos de uso, el uso público, para accesos sin restricciones y el uso restringido, solamente dar acceso a los partners que deben acceder a nuestros sistemas para consultas de disponibilidad o reservas.

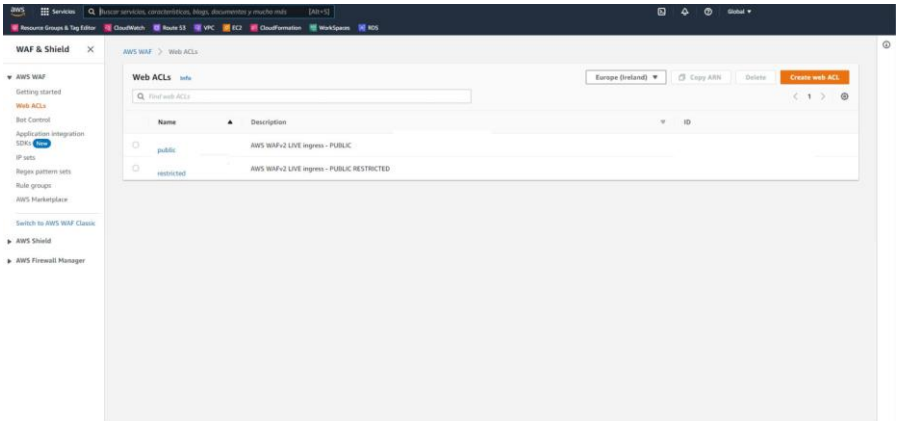

Figura 33. AWS WAF. Web ACLs.

Dentro de cada Web ACL tenemos algunas pestañas interesantes. Por ejemplo:

Pestaña Overview: Gráfica de las peticiones por cada cinco minutos, las permitidas y las bloqueadas.

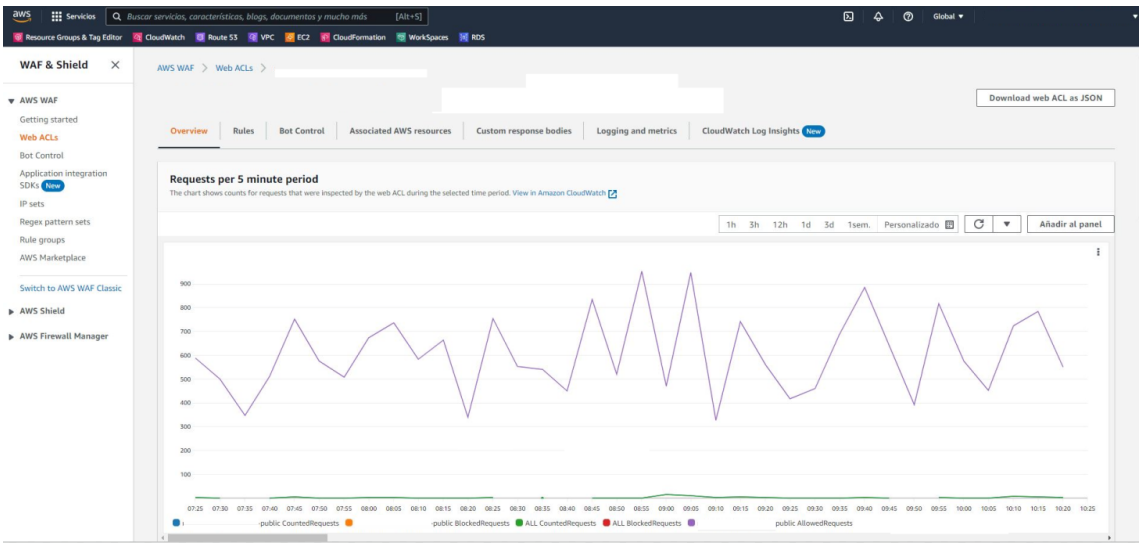

Figura 34. AWS WAF. Gráfica de peticiones.

Pestaña Rules: Donde se especifican las reglas que deben usarse en este Web ACL. Como estamos en el público debemos dejar pasar todo el tráfico pero con condiciones. Vemos en la imagen que existe una regla que incorpora AWS por defecto y son un conjunto de reglas anti-scripting y otras vulnerabilidades.

También tenemos una regla con un rate_limit, si un usuario realiza demasiadas peticiones en el tiempo establecido se le bloquea temporalmente. Y finalmente la regla por defecto que es permitir cuando se hayan pasado por las dos anteriores reglas.

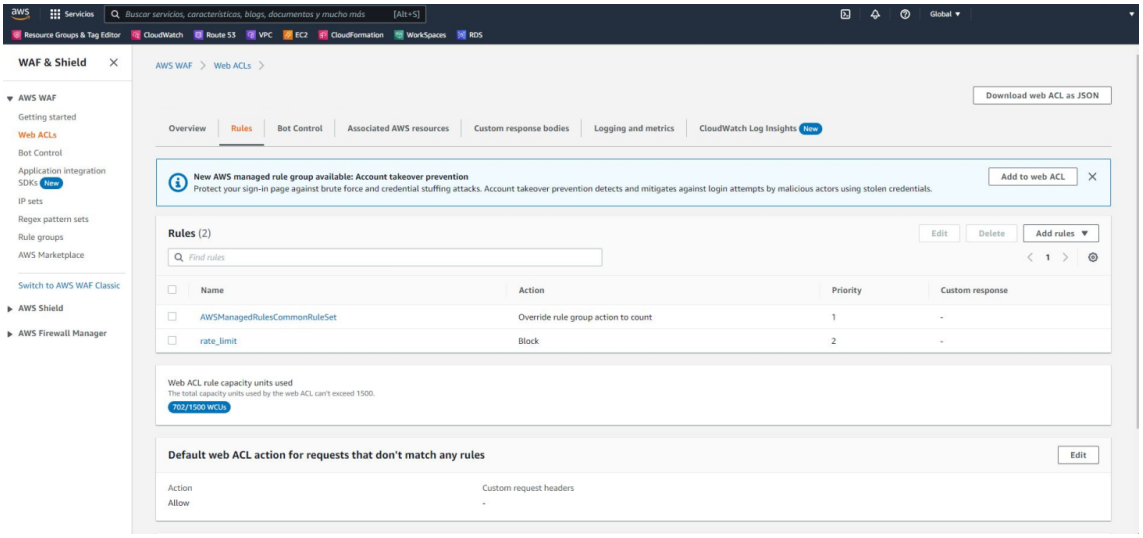

Figura 35. AWS WAF. Reglas.

La pestaña Bot Control: Para activar la función para bloquear bots.

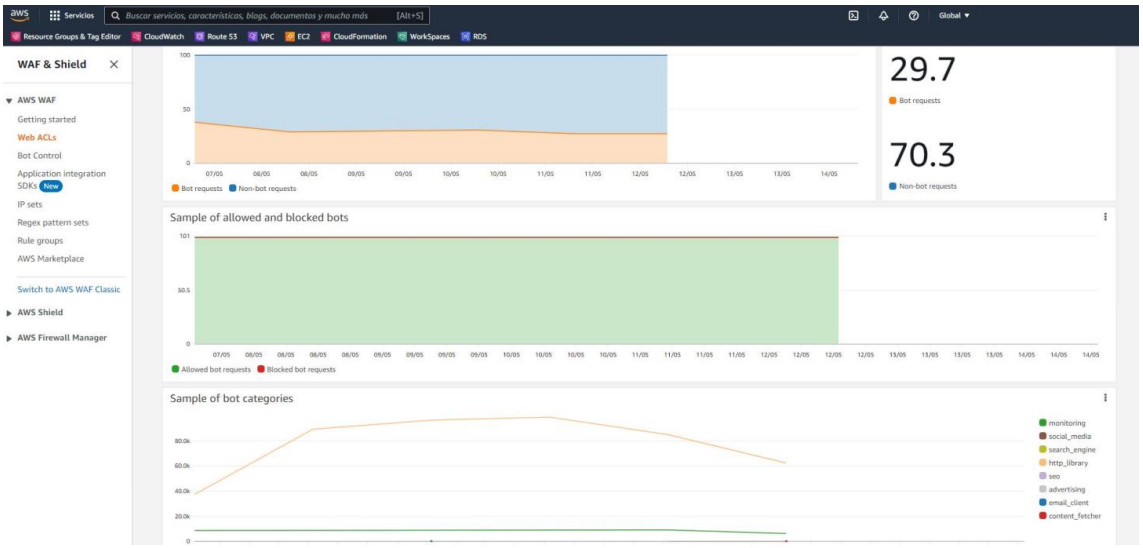

Figura 36. AWS WAF. Gráficas de Bots.

Pestaña Recursos AWS asociados: En esta pestaña asociamos los recursos donde irá el tráfico, en nuestro caso es un cdn que se ha creado expresamente.

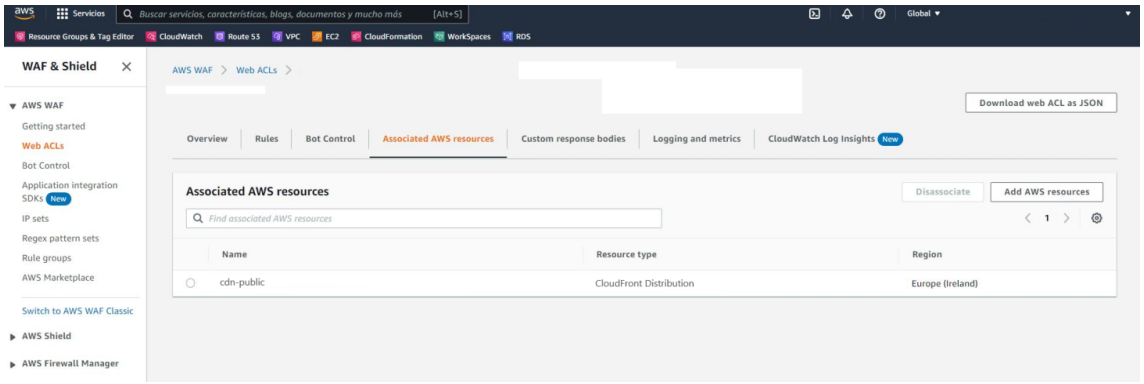

Figura 37. AWS WAF. Recursos asociados.

4.1.3 AWS CloudFront.

Hay poco que explicar de AWS Cloudfront, AWS WAF le manda el tráfico de Internet y éste se lo manda al Ingress público de AWS EKS. Podemos configurar una regla en el Ingress para que solo acepte tráfico proveniente de nuestro AWS CloudFront.

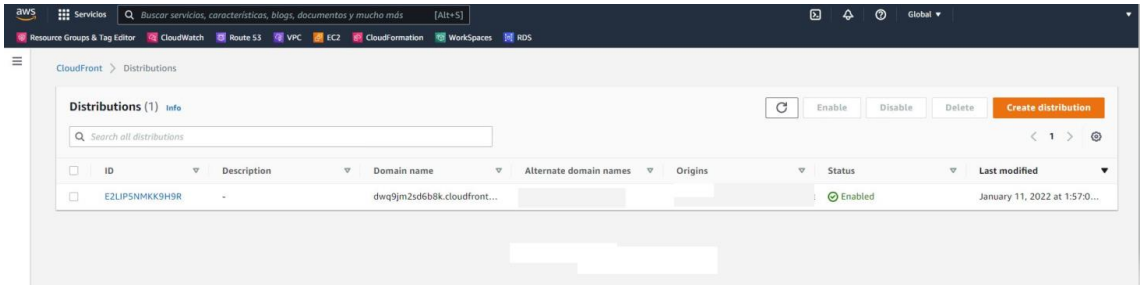

Figura 38. AWS CloudFront.

4.1.4 AWS RDS.

Todas las bases de datos se han pasado a AWS RDS. La migración de los datos se ha hecho con AWS DMS para que los datos estén sincronizados en las bases de datos origen y destino, el DMS en estado mirroring asegura que el día de la migración las bases de datos de las Zonas Webservers e Integraciones tendrán los datos correctos en las nuevas bases de datos que se han creado en la cuenta de AWS donde está AWS EKS. Una vez migradas las aplicaciones, el DMS se para y se apagan las bases de datos antiguas.

Para simplificar el mantenimiento de las bases de datos se ha creado una por cada aplicación. En la siguiente imagen se pueden ver algunas de las RDS en el entorno de producción.

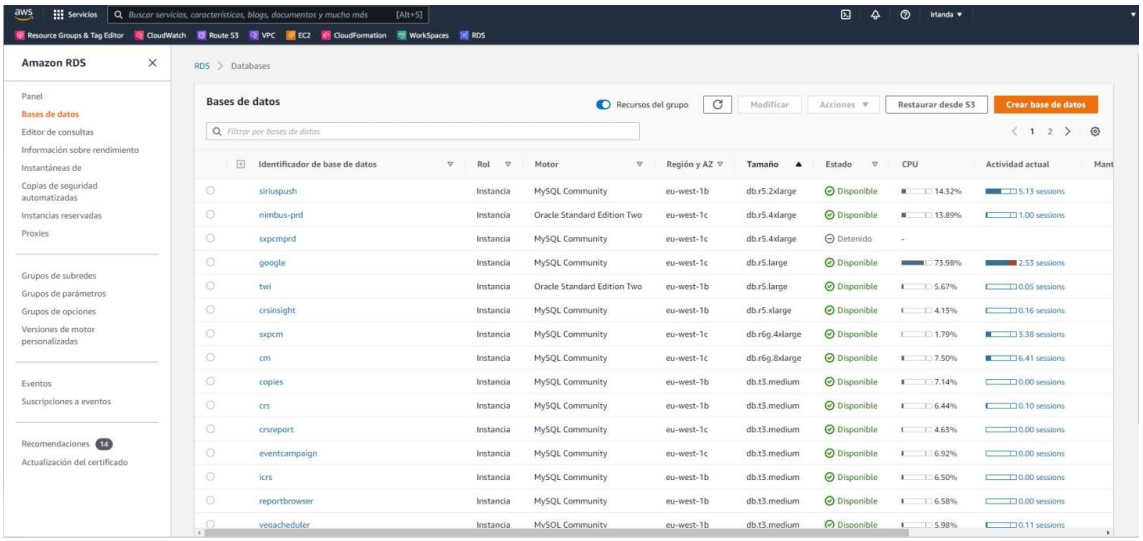

Figura 39. AWS RDS. Listado de algunas RDS por aplicación.

Con todo esto tenemos configurado todo el sistema de la empresa. Todas las aplicaciones irán sobre AWS EKS ofreciendo seguridad con AWS WAF y AWS CloudFront además de un mejor rendimiento eliminando latencias ente Zonas.

En la siguiente imagen tenemos un esquema general de como queda la nueva cuenta con todos los elementos que hemos nombrado

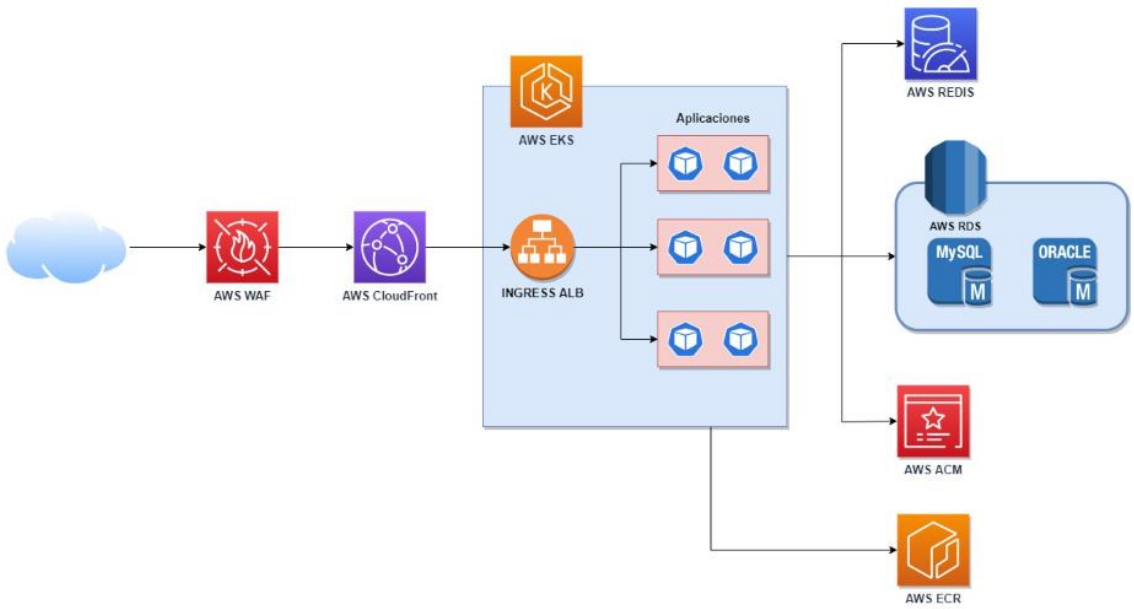

Figura 40. Esquema final de la solución con AWS EKS.

4.2. Costes y asignación de recursos

En anteriores apartados ya hemos visto algunos costes, por ejemplos de las RDS, del WAF, etc. Sabemos que cuesta una máquina EC2 sobre los que estarán los pods de las aplicaciones, sin embargo, cuando añadimos autoescalado en AWS EKS, gestión de máquinas Spot y la posibilidad de que el clúster tenga distintos tipos de instancias dependiendo de la disponibilidad, calcular es coste es complicado. Otra complicación es saber exactamente cuesta una cierta aplicación en EKS.

Afortunadamente existen herramientas para calcular los costes de forma desglosada por aplicación, clúster o grupo de aplicaciones (Apptio, Cast AI, SPOT Clous Analyzer, etc). Una de las más simples y la que usamos en la empresa es Kubecost.

Kubecost nos permite saber el coste de nuestros clúster de EKS globalmente y por aplicación. Por ejemplo, en la imagen siguiente, se nos indica el coste mensual del clúster de producción (live).

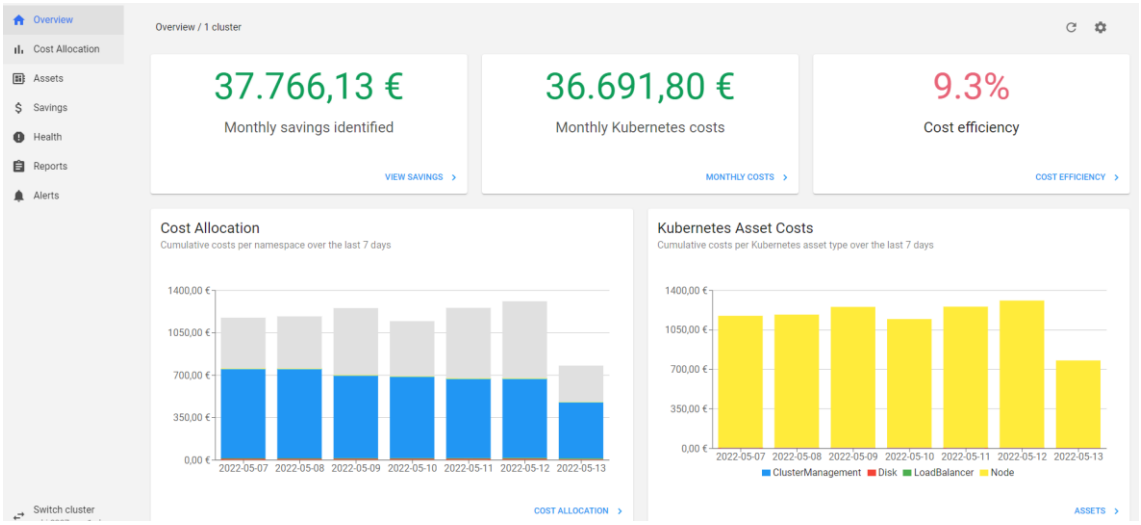

Figura 41. Kubecost. Coste total.

Nos indica qué aplicaciones son las que más consumen por cpu y memoria y las que más gastan.

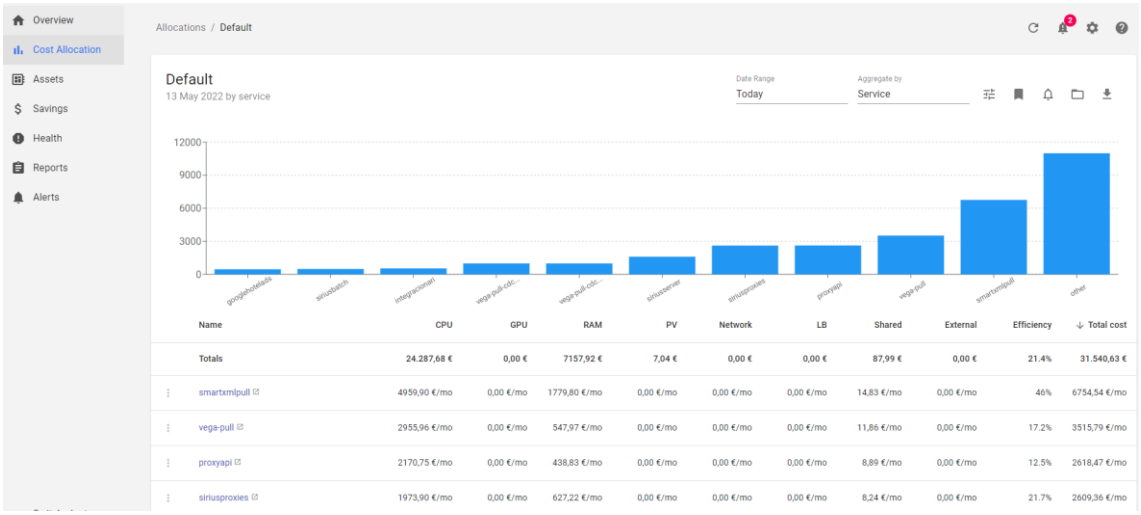

Figura 42. Kubecost. Relación de gastos por aplicación.

Podemos seleccionar una aplicación determinada y ver los gastos diarios y como ha ido cogiendo recursos a medida que los necesitaba.

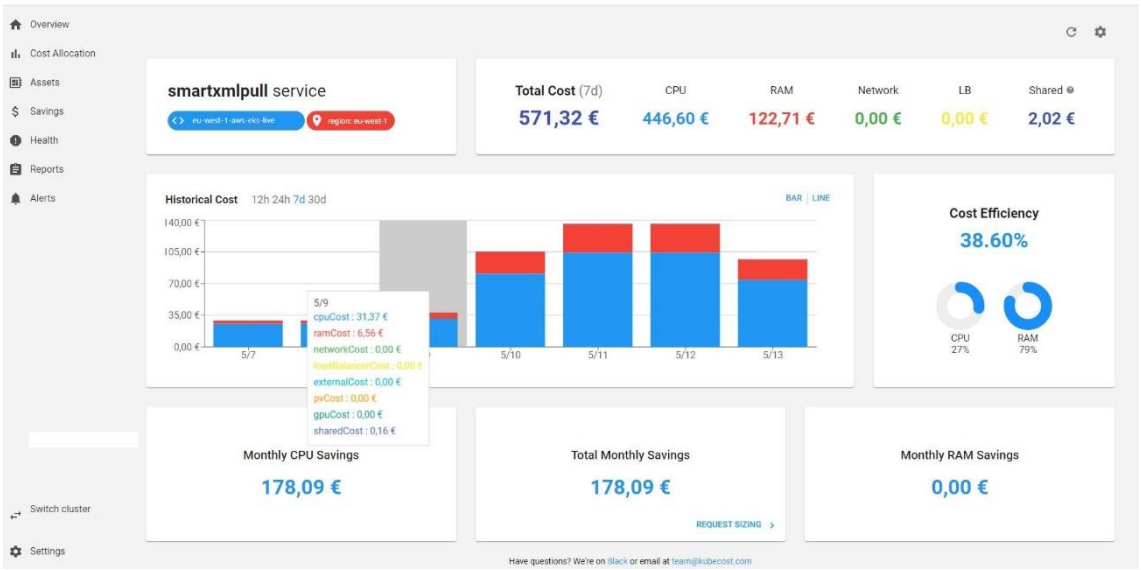

Figura 43. Kubecost. Desglose del gasto de una aplicación.

4.3. Monitorización y gestión de alarmas

En cuanto a la monitorización y sistema de alarmas, se ha adoptado el mismo sistema que ya se venía usando en la Zona Integraciones, Zabbix con notificaciones a canales de Teams.

Ahora, como todos los sistemas se encuentran integrados en AWS EKS y los desarrollos de todas las zonas se encuentran ejecutándose juntos, se han creado grupos de notificaciones en Zabbix que, dependiendo el host que genera la alarma se notifica en un canal u otro. Esto facilita que distintos grupos de trabajo y soporte reciban la alerta en su canal correspondiente. Las alertas críticas de todos los hosts se reciben también en el canal que tiene a su disposición el equipo que conforma la guardia 24x7 para que sean atendidas fuera del horario laboral o de oficina.

Se ha creado un Dashboard más amplio para poder abarcar tanto el entorno productivo Live como el de desarrollo Dev para poder obtener en un solo panel uns vista general del correcto funcionamiento del sistema.

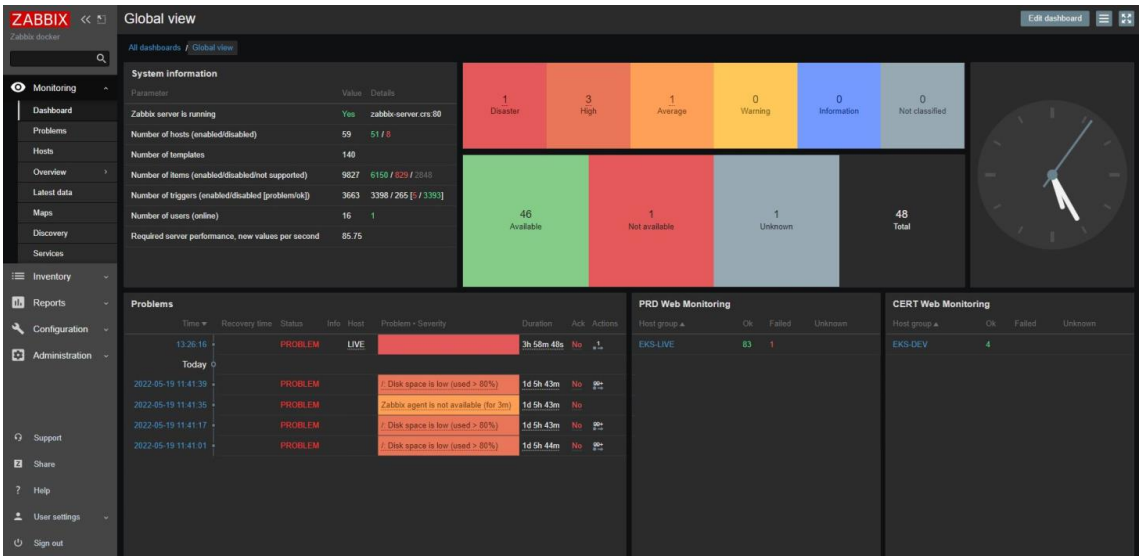

Figura 44. Zabbix. Dashboard general.

5. Implantación

5.1. Planificación de la migración

Para realizar la migración el día convenido, existe antes un trabajo preparatorio que debe realizarse con la antelación justa para que no se solape con la propia migración, pero lo suficientemente ajustado como para que el impacto sea mínimo. Estamos hablando mayoritariamente de los datos.

Las bases de datos son una pieza muy importante en la migración porque su integración nos asegurará que, al arrancar las aplicaciones en el nuevo entorno, no se habrán perdido datos. En este caso perder datos de reservas no es lo más importante, los hoteles cuentan con sistemas con los que recuperar las reservas pendientes, además de otros sistemas de Backup que, aunque son manuales permitirían recuperar el estado anterior a la migración, lo más crítico sería perder datos de los propios hoteles.

Imaginemos que con el cambio las tablas correspondientes a los hoteles no son consistentes, eso pone en peligro todo el sistema de reservas pasado, presente y futuro ya que podría ser que, al recuperar las reservas, el número de habitaciones, personas alojadas u otros datos con casaran con los del hotel. Esto podría provocar cancelaciones automáticas por errores que no son reales, un overbooking que no es tal, consultas de disponibilidad agotadas cuando aun queda sitio en el hotel, hoteles ofreciendo servicios que no tiene, etc. Recomponer todo el sistema es una tarea que no queremos tener que hacer y la única forma es asegurar la consistencia de los datos.

Para asegurar la consistencia de los datos, se analizaron distintas vías:

- Usar AWS DMS en modo “mirror” continuo de todas las bases de datos: Esta era la gran esperanza de la empresa para no tener que parar las aplicaciones durante mucho tiempo. Las aplicaciones seguirían en marcha, ofreciendo disponibilidad y operativa de reservas normalmente. Mientras los datos se actualizan de la base de datos origen a la que será la base de datos destino en la nueva cuenta. Sin embargo, el alto número de actualizaciones, recordemos que los borrados en bases de datos relacionales penalizan mucho en los tiempos, creaban una latencia en la actualización que no aseguraba que cuando se apagaran las aplicaciones para arrancarlas en la cuenta cuenta, con otra configuración, los datos de la nueva base de datos fueran exactamente iguales que los que se estaban usando antes de parar. Además, el gran número de procesos de actualización (AWS DMS) en marcha a la vez, otorgaba mucha complejidad al proceso.

- Usar AWS DMS en modo “mirror” continuo en grupos de bases de datos: Se pensó entonces en realizar la migración por grupos de aplicaciones, y por tanto, mantener los datos en mirror pero en grupos. Esta opción se descartó enseguida al ver que muchas aplicaciones consultaban más de una base de datos, es decir, todo el repertorio de datos que necesitaba una cierta aplicación estaba deslocalizado en varias bases de datos, por lo tanto, era imposible hacer funcionar con normalidad una aplicación si no se migraban todas las bases de datos que necesitaba. El problema era que las aplicaciones cruzaban datos unas con otras haciendo que las fuera imposible establecer ningún grupo de aplicaciones aisladas con sus bases de datos.

- Parada total de todo el sistema: No quedaba más opción que establecer una hora de parada para todo el sistema, aplicaciones y bases de datos, copiar las bases de datos del origen al destino con AWS DMS en modo carga completa sin replicación continua, es decir, como si fuera un export-import de la base de datos.
  Después de analizar todas las bases de datos y estimar la duración de estas tareas se concertó en parar todo el sistema ocho horas antes de empezar la migración de las aplicaciones.

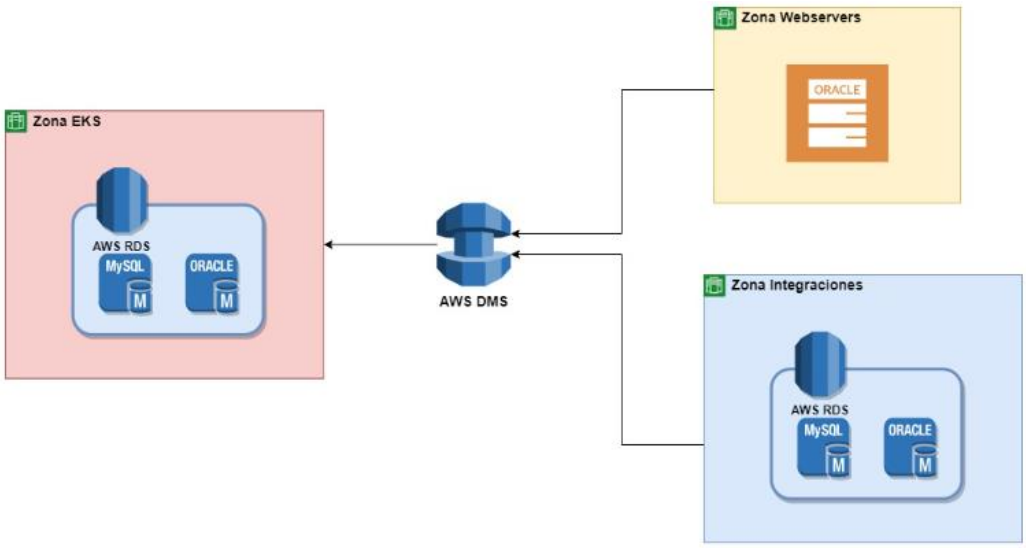

Figura 45. Esquema migración de bases de datos

5.1.1 Cronología

Día	Hora	Recurso		Descripción	Localización
-1	21:00	Sirius	Aplicación	Parada del aplicativo gestor de reservas	Zona Mainframe
	21:00	Web Corporativa	Aplicación	Inhabilitar gestión de reservas y usuarios para ofrecer solo disponibilidad (sin modificación en bbdd)	Zona Webservers
	21:00	Callcenter	Aplicación	Inhabilitar gestión de reservas y usuarios para ofrecer solo disponibilidad (sin modificación en bbdd)	Zona Webservers
	21:15	BBDD Oracle	BBDD	Migración tablas para Sirius, Web y Callcenter	Zona Webservers
	21:15	BBDD Oracle	RDS	Migración tablas para XWG	Zona Integraciones
	21:15	BBDD MySQL	RDS	Migración tablas para XWG	Zona Integraciones
	21:15	BBDD MySQL	RDS	Migración tablas para SXP	Zona Integraciones
	21:15	BBDD MySQL	RDS	Migración tablas para resto de aplicaciones Integraciones	Zona Integraciones
0	5:00	Sirius	Aplicación	Arrancar el aplicativo de gestor de reservas apuntando a las nuevas bbdd	Zona EKS
	5:30	Sirius	Aplicación	Prueba del gestor de reservas	Zona EKS
	6:30	Web Corporativa	Aplicación	Arrancar el aplicativo apuntando al nuevo Sirius	Zona EKS
	7:00	Web Corporativa	Aplicación	Prueba del aplicativo apuntando al nuevo Sirius	Zona EKS
	7:30	Callcenter	Aplicación	Arrancar el aplicativo apuntando al nuevo Sirius	Zona EKS
	8:00	Callcenter	Aplicación	Prueba del aplicativo apuntando al nuevo Sirius	Zona EKS
	8:30	XGW	Aplicación	Parada y arranque de la aplicación apuntando al nuevo Sirius y a las nuevas bbdd	Zona EKS
	9:00	XGW	Aplicación	Prueba del aplicativo apuntando al nuevo Sirius y a las nuevas bbdd	Zona EKS
	8:30	SXP	Aplicación	Parada y arranque de la aplicación apuntando al nuevo Sirius y a las nuevas bbdd	Zona EKS
	9:00	SXP	Aplicación	Prueba del aplicativo apuntando al nuevo Sirius y a las nuevas bbdd	Zona EKS
	9:30	Aplicaciones Integraciones	Aplicación	Parada y arranque de los aplicativos apuntando al nuevo Sirius y a las nuevas bbdd	Zona EKS
	12:00	Aplicaciones Integraciones	Aplicación	Prueba de los aplicativos apuntando al nuevo Sirius y a las nuevas bbdd	Zona EKS
	14:00	Seguimiento		Revisión con los distintos grupos para informar de los errores e incidencias ocurridas durante la	Zona EKS
	16:00	Seguimiento		Revisión con los distintos grupos para informar de los errores e incidencias desde la última reunión	Zona EKS
	17:00			**Fin de la migración con éxito**	**Zona EKS**
	21:00	Seguimiento		Revisión con los distintos grupos para informar de los errores e incidencias desde la última reunión	Zona EKS
1	9:00	Seguimiento		Revisión con los distintos grupos para informar de los errores e incidencias desde la última reunión	Zona EKS
	14:00	Seguimiento		Revisión con los distintos grupos para informar de los errores e incidencias desde la última reunión	Zona EKS
	21:00	Seguimiento		Revisión con los distintos grupos para informar de los errores e incidencias desde la última reunión	Zona EKS

Figura 46. Cronología de la migración

5.2 Implantación

En el anterior punto se establece la cronología de la migración, que comprende un día de preparación, el día de la migración y el siguiente para seguimiento, monitorización y resolución de errores.

La migración empezó un viernes (día -1) a las 21:00h y se desarrolló durante un fin de semana escogido por su bajo número de reservas. Históricamente el fin de semana anterior al puente de la Constitución (6-8 de diciembre) es un fin de semana de poca actividad en cuanto a reservas y disponibilidad a través de la web y el callcenter. Esa fecha fue la escogida para realizar la migración.

Durante la noche del viernes y madrugada del sábado un equipo de soporte, que habitualmente trabaja con las bases de datos implicadas, realizó las “releases” de las aplicaciones y puso en marcha los DMS de las bases de datos. Periódicamente se monitorizaban y se establecieron ciertas alertas que eran

atendidas por el personal de guardia que debía avisar al personal de soporte que empezó la tarea.

A las 5:00h de la madrugada del sábado todo el personal implicado de alguna u otra forma estaba presente en las instalaciones de la oficina. Esto comprende al personal de soporte que debía realizar la migración, desarrolladores de las aplicaciones implicadas, DBA (Database Administrator) de las bases de datos implicadas y un técnico del proveedor de servicios de Internet que también nos proporciona soporte y mantenimiento de la red corporativa.
El personal de soporte realizaba la tarea marcada en el cronograma y una vez realizada se daba aviso al equipo desarrollador pertinente para que realizara las pruebas necesarias.

A las 14:00 se dio por concluida la migración y se realizó la primera reunión de seguimiento en el que el responsable de los distintos grupos implicados exponía los problemas que se habían encontrado y si se habían resuelto y cómo.

La migración se dio por concluida después de la segunda reunión se seguimiento al constatar que no había grandes errores en las aplicaciones y que todo iba sobre la marcha y se operaba con normalidad. Todo el personal pudo abandonar las instalaciones corporativas, aunque debían estar disponibles por teletrabajo por si surgía algún error importante no detectado y debía solventarse de inmediato.

A partir de entonces se iban realizando reuniones de seguimiento periódicas ya que los equipos de soporte y guardia 24x7 seguía monitorizando activamente las aplicaciones, bases de datos y recursos implicados.

Después del domingo, la monitorización se dejó únicamente para el personal de guardia 24x7.

5.3 Resultado de la migración

La migración resulto un éxito rotundo, aunque como es de esperar, hubo algunos problemas que se fueron resolviendo más o menos rápidamente.

Los errores más comunes fueron los relacionados con las bases de datos. Estos errores se resolvieron con bastante rapidez ya que las aplicaciones vuelcan errores bastante aclarativos, ya sea por permisos, por tablas inexistentes, violaciones de claves o constraints o por índices y vistas que no existen. En estos casos, los DBA actuaron rápido y permitieron solventar los errores con prontitud.

El otro error frecuente y con una resolución más complicada y lenta, fueron los errores de código, falta de librerías o funcionalidades que la aplicación, que estos eran más difíciles porque se necesitaba re-evaluar la funcionalidad y aplicar un código corrector. Aun así, la mayoría se resolvieron dentro del tiempo establecido para la migración.

6. Conclusiones

En este proyecto se ha expuesto una migración desde diferentes entornos surgidos en su momento por diversas necesidades y motivado por la necesidad actual de dotar a los sistemas de la suficiente agilidad, flexibilidad y dinamismo de la que carecían hasta entonces.

Usando herramientas de automatización y de desarrollo continuo, y utilizando los recursos que nos ofrece nuestro proveedor de servicios Amazon AWS, se ha creado un entorno robusto en cuanto a seguridad y fiabilidad se refiere pero a la vez ágil y flexible en lo referente a posibilidades de configuración y escalabilidad.

El cambio desde un sistema monolítico como es el Mainframe a Kubernetes, donde la manera de desarrollar, operar y gestionar el software ha cambiado fundamentalmente. No se trata solo de medidas para introducir el desarrollo ágil, sino de la conexión de distintas tecnologías y métodos que permiten a la empresa lograr un alto nivel de dinamismo y flexibilidad durante todo el ciclo de vida del software.

En este proyecto también se evalúa la necesidad de mejora económica por parte de la empresa y las posibilidades que se presentaron para sustituir sobre todo la parte alojada en el mainframe. En este sentido Kubernetes ha permitido, no solamente bajar muchísimo el coste anual del sistema sino permite también optimizar, escalar y crecer a un menos coste respecto a los sistemas que se estaban usando anteriormente en la empresa.

La migración realizada fue un éxito total. Las aplicaciones y la web se migraron correctamente en el tiempo estimado y el tiempo de indisponibilidad en la gestión de reservas fue de tan solo doce horas y el tiempo de indisponibilidad de consultas fue tan solo de cuatro horas con lo que cumplirían los requisitos de indisponibilidad mínima.

A nivel personal, participar activamente en un proyecto de esta magnitud, me ha permitido conocer distintas tecnologías que de otra forma no tendría acceso a ellas.

Estudiar estas tecnologías, herramientas y procesos me ha proporcionado un buen conocimiento de las mayores funcionalidades de Amazon AWS y me ha permitido, sobre todo, adquirir conocimientos sobre Kubernetes aplicados sobre la integración continua y las herramientas necesarias para lograr una buena automatización de procesos, tanto en la aplicación de la configuración de la infraestructura como en el proceso de aplicación de cambios en las aplicaciones.

Respecto a la migración, estoy plenamente satisfecho por haber podido formar parte de uno de los grupos que participaron más activamente en la operación y el hecho de que se ejecutara con éxito, y solamente aparecieran algunos

pequeños errores que pudieron ser salvados casi al instante y no aparecieran errores graves ni durante el día ni en días sucesivos.

7. Bibliografía

- Ingress. (n.d.). Kubernetes. Retrieved May 13, 2022, from https://kubernetes.io/docs/concepts/services-networking/ingress/
- Kubecost. (n.d.). Kubecost.Com. Retrieved May 14, 2022, from https://www.kubecost.com/?gclid=Cj0KCQjwpv2TBhDoARIsALBnVnlZKa3TCdwhD1aC6ZFH9SePaJPzSqhCL-ZAaE3LuvQtaaUe62PvVloaArP-EALw_wcB
- (N.d.-a). Amazon.Com. Retrieved May 13, 2022, from https://aws.amazon.com/es/blogs/opensource/kubernetes-ingress-aws-alb-ingress-controller/
- (N.d.-b). Amazon.Com. Retrieved May 13, 2022, from https://docs.aws.amazon.com/es_es/eks/latest/userguide/alb-ingress.htm
- (N.d.-c). Amazon.Com. Retrieved May 13, 2022, from https://docs.aws.amazon.com/es_es/elasticloadbalancing/latest/application/introduction.html
- (N.d.-d). Amazon.Com. Retrieved May 13, 2022, from https://aws.amazon.com/es/ec2/pricing/on-demand
- (N.d.-e). Amazon.Com. Retrieved May 13, 2022, from https://aws.amazon.com/es/rds/pricing/
- (N.d.-f). Amazon.Com. Retrieved May 13, 2022, from https://aws.amazon.com/es/waf/pricing/
- (N.d.-g). Amazon.Com. Retrieved May 13, 2022, from https://aws.amazon.com/es/elasticloadbalancing/pricing/
- (N.d.-h). Amazon.Com. Retrieved May 13, 2022, from https://aws.amazon.com/es/certificate-manager/
- (N.d.-i). Amazon.Com. Retrieved May 14, 2022, from https://aws.amazon.com/es/waf/
- (N.d.-j). Amazon.Com. Retrieved May 14, 2022, from https://aws.amazon.com/es/blogs/aws-spanish/protegiendo-su-pagina-web-con-aws-waf-y-amazon-cloudfront/