Proyecto Fin de Carrera

Grado en Ingeniería de las Tecnologías de la Telecomunicación

Diseño y despliegue de una aplicación de integración de solicitudes de soporte en nube pública sobre clúster Kubernetes.

Autor: Juan Ariza Toledano

Tutor: Francisco Javier Muñoz Calle

Dpto. Ingeniería Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Grado
Ingeniería de Tecnologías de la Telecomunicación

Diseño y despliegue de una aplicación de integración de solicitudes de soporte en nube pública sobre clúster Kubernetes.

Autor:
Juan Ariza Toledano

Tutor:
Francisco Javier Muñoz Calle
Profesor Colaborador Doctor

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Grado: Diseño y despliegue de una aplicación de integración de solicitudes de soporte en nube pública sobre clúster Kubernetes.

Autor: Juan Ariza Toledano

Tutor: Francisco Javier Muñoz Calle

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis amigos

A mis compañeros y profesores

AGRADECIMIENTOS

En primer lugar, dar las gracias a mis padres y mi hermana por el cariño, esfuerzo y dedicación hacia mí.

También me gustaría agradecer a mis amigos, por ser un grupo tan desastroso y aún así quererlos; a mis compañeros de piso, por la difícil tarea de soportarme día a día; y a María, por ser mi compañera de vida.

Merece un agradecimiento especial el Dr. Javier J. Salmerón García, coautor del diseño/implementación de la aplicación, sin el cual este trabajo no habría sido posible.

Para finalizar, reconocer el apoyo incondicional a mis abuelos, a los cuales dedico este trabajo.

Juan Ariza Toledano

Sevilla, 2020

RESUMEN

Bitnami basa su prosperidad en torno a cinco valores, siendo uno de ellos promover el éxito de sus usuarios (del inglés: “Empower users”). El soporte que Bitnami ofrece a sus usuarios, tanto de ventas (consultas sobre los planes de subscripción disponibles, gestión de pagos o renovación de licencias) como técnico (resolución de incidencias con los productos de Bitnami, consultas sobre configuraciones avanzadas o reporte de errores y fallos de seguridad), es el pilar fundamental de dicho valor.

Con un gran número de usuarios (ya que se despliegan más de 1.5 millones de instancias basados en productos de Bitnami al mes) y hasta cinco plataformas web distintas dónde atender las preguntas de estos, es fundamental disponer de la metodología, automatismos y herramientas necesarias para hacer frente a la gran cantidad de soporte solicitado.

Este proyecto nace de la necesidad de integrar automáticamente el soporte solicitado por los usuarios en las diferentes plataformas en las que Bitnami los atiende bajo un mismo portal, que permita a los agentes de soporte de la compañía realizar su trabajo de una forma eficiente. Se decide utilizar la plataforma web de gestión de tareas denominada Trello, como portal único para la gestión del soporte.

Para alcanzar el objetivo se decide desarrollar una aplicación, denominada Trello Updater, que canalice la información procedente de las plataformas en las que Bitnami ofrece soporte, la adapte y la envíe a la plataforma Trello en el formato adecuado.

Una vez desarrollada la aplicación, se crea un plan el despliegue y puesta en producción de esta. Dicho plan consta de dos fases: el despliegue provisional de la aplicación en una máquina virtual en nube pública; y la migración de la aplicación a un clúster basado en la plataforma Kubernetes en nube pública, acompañada de la implementación de un sistema de integración/despliegue continuo.

ABSTRACT

Bitnami has based his growth on decisions guided by its values, being “Empower Users” one of them. The support provided to the users is the cornerstone of this value.

In order to attend the big amount of users (as a consequence of over 1.5 millions deployed servers based on Bitnami products per month) dropping their questions on five different platforms, it is indispensable to have the proper methods, automations and tools.

This project starts from the need of automatically integrating the support requests received on the different platforms where Bitnami attends its users under an unique portal, to allow the company support agents to work more efficiently. Trello, a task management web platform, will be used to manage the support duties.

To reach this goal, a new application, named Trello Updater, is developed to process and redirect the information received on each platform where Bitnami offers support to Trello.

Once the application is developed, a plan to deploy it on a production environment is designed. The plan is devided into two phases: provisionally deploying the application in a web server in the cloud; and migrating the application to a Kubernetes cluster in the cloud and implementing a CI/CD pipeline.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

What helps people, helps business
- Leo Burnett, Advertising Executive-

Tanto por el normal desarrollo de la inmensa mayoría de compañías del sector de la telecomunicación como por motivos estratégicos, el soporte es una pieza fundamental para lograr cumplir los objetivos empresariales y afrontar nuevas metas.

Los usuarios han de tener un rápido acceso a los servicios de soporte y recibir una respuesta efectiva a sus problemas en el menor tiempo posible. Sólo así se garantiza una buena experiencia de uso que sustente la confianza depositada en un producto y la buena reputación de una compañía.

Por tanto, podríamos asegurar que todo producto software necesita de dos complementos extra para ser considerado como tal:

- Documentación.
- Soporte Técnico.

A través de esta introducción recorreremos como toman forma estos elementos en Bitnami. Además, analizaremos cuáles son los retos que nos llevan a plantearnos la necesidad de un cambio de enfoque en los servicios de soporte y la posterior realización de este trabajo.

1.1 Importancia de la Comunidad

Hace ya 8 años, Bitnami creó un foro llamado Community [1] con la finalidad de que los usuarios de sus diferentes productos pudieran acceder de una forma sencilla al soporte técnico que los ingenieros de Bitnami ofrecían en él.

Con este foro se perseguía dar el mayor uso posible a cada una de las respuestas

ofrecidas, puesto que cualquier usuario puede navegar a través de él y encontrar de forma sencilla la respuesta a sus problemas en las preguntas que otros usuarios hicieron anteriormente.

Sin embargo, con la creación de un foro abierto no sólo se pretendía ofrecer soporte técnico sino crear una verdadera comunidad. Es decir, fomentar la colaboración entre los distintos usuarios de las soluciones de Bitnami y ofrecer un canal en el que éstos pudieran proponer mejoras y discutir sobre ellas.

1.2 Documentación y Soporte: retroalimentación

Previamente se mencionaba la vital importancia de la documentación y el soporte técnico como complementos que todo producto software ha de tener. Hay que entender que ambos están íntimamente relacionados.

Una buena documentación, bien organizada y con la suficiente cobertura debería ser capaz de dar respuesta a la mayoría de las dudas que surjan a los usuarios a la hora de utilizar un producto. Podría incluso afirmarse que el primer agente de soporte no es quién atiende en primera instancia al usuario a través de un canal de soporte, sino quién escribe la documentación. Por consiguiente, se puede reducir sustancialmente la cantidad de soporte solicitado a través de una continua mejora de la documentación, así como de los algoritmos de búsqueda de esta.

Por otro lado, son los agentes de soporte, con su trabajo diario, aquellos que detectan erratas, inconsistencias y lagunas en la documentación, al proveer soluciones a los usuarios basadas en ésta. Dotando a dichos agentes de los mecanismos adecuados para aportar "*feedback"* a los documentalistas, se completa un bucle de retroalimentación que permite mejorar ambos complementos del producto.

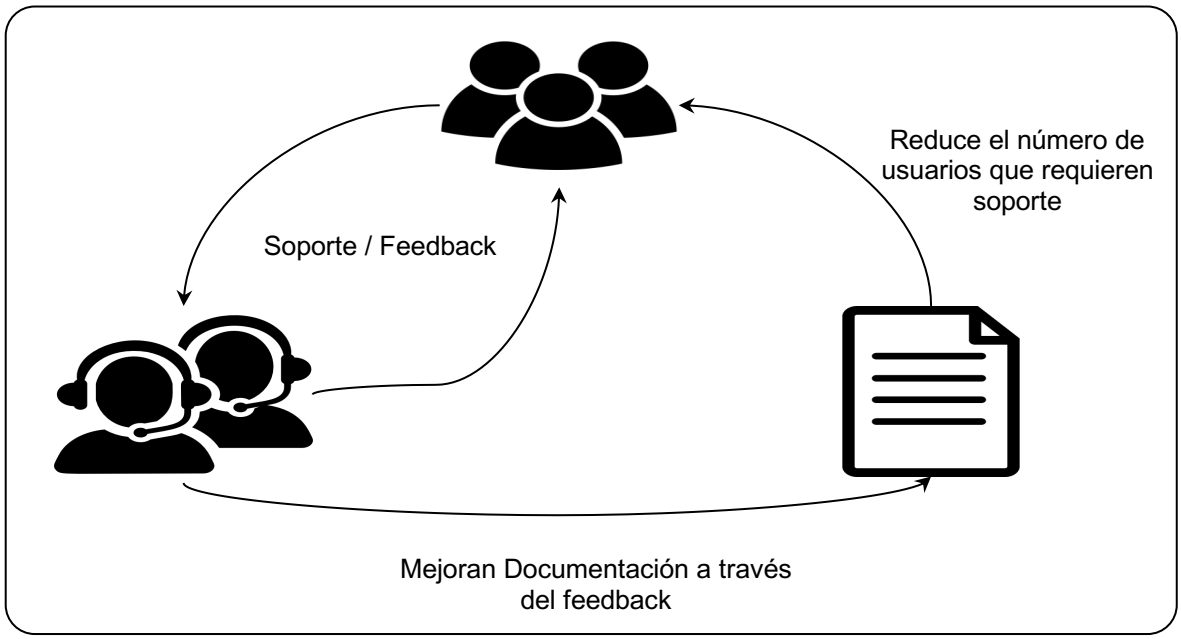


Figura 1 - Retroalimentación Documentación-Soporte

1.3 Necesidad de nuevas plataformas

Tradicionalmente Bitnami ha atendido las solicitudes de soporte de sus usuarios en dos plataformas: Community y HelpDesk [2].

La primera es el medio habitual en el que cualquier usuario puede preguntar cualquier duda técnica o de carácter general sobre el uso de los diferentes productos del catálogo de Bitnami.

Por otro lado, la plataforma HelpDesk (basada en el servicio ofrecido por la compañía Zendesk [3]) permite ofrecer a los usuarios un soporte personalizado a través de correo electrónico, garantizando al usuario la total privacidad en aquellos asuntos que así la requieran. Esta característica lo convierte en el medio utilizado para dar respuesta a problemas relacionados con cuentas de usuario, pagos, recuperación de contraseñas, etc.

1.3.1 Stackoverflow y GitHub

Debido a la creciente popularidad de la plataforma online Stackoverflow [4], cada vez son más los usuarios que acuden a ésta para preguntar dudas técnicas sobre los productos de Bitnami.

Por otro lado, en los últimos dos años son muchos los proyectos open-source que Bitnami ha publicado a través de la plataforma GitHub [5] (actualmente la organización Bitnami cuenta con más de 300 repositorios públicos en dicha plataforma). Al tratarse de código abierto, cualquier usuario puede contribuir al mismo a través de un “Pull Request” así como abrir una incidencia (más conocida como “issue”) para que otros puedan aportar una solución.

En vista de la necesidad de atender a los usuarios que hacen uso de estas plataformas, se decide añadirlas al ámbito de soporte. Es decir, atender a los usuarios que soliciten ayuda en ellas pasa a formar parte del trabajo diario que el equipo de soporte de Bitnami realiza con un ANS (Acuerdo de Nivel de Servicio) similar al de Community.

1.3.2 Incremento de los procesos manuales: caos

La incorporación de nuevas plataformas al ámbito de soporte supone un significante aumento de la carga a responder, así como un incremento de las tareas manuales en el trabajo diario. El agente tiene que familiarizarse con nuevas plataformas y cambiar continuamente de contexto para realizar su trabajo lo que reduce sustancialmente su eficiencia.

Los continuos cambios de contexto, la dificultad para localizar las peticiones de soporte y el constante aumento de la carga; hacen que aumente el estrés y la desorganización interna dando a lugar a escenarios caóticos con altos niveles de intensidad laboral y escasa productividad.

1.4 Alcance y objetivos

El objetivo de este proyecto es contribuir a la mejora de las herramientas y procedimientos internos utilizados por el equipo de soporte de Bitnami.

Debido al constante incremento de las peticiones de soporte recibidas, así como la necesidad de extender los servicios ofrecidos a nuevas plataformas, es trascendental disponer de herramientas que automaticen tantos procesos manuales como sean posibles. Además, dichas herramientas deben ayudar a incrementar la eficiencia de aquellas tareas que no puedan ser automatizadas. Todo ello ha de permitir escalar los servicios ofrecidos de manera sostenible, sin impactar en el normal desarrollo de la actividad no relacionada con soporte que los ingenieros de Bitnami realizan.

Para ello, se propone el diseño y despliegue de una aplicación web que permita colectar, clasificar y adaptar las solicitudes de soporte recibidas en las distintas plataformas. Tras la colecta y procesado de las solicitudes, la aplicación ha de enviar información sobre estas a una plataforma de gestión de tareas que permita organizarla bajo un mismo tablón. El propósito de dicho tablón no es otro que ofrecer un único interfaz que permita acceder de una forma sencilla a cada una de las solicitudes de soporte recibidas, repartir la carga de trabajo entre los agentes disponibles y clasificarla en función a su prioridad, antigüedad, complejidad, etc.

La información almacenada en dicho tablón ha de ser fundamental para la extracción de una serie de estadísticas que permitan tomar las medidas correctoras oportunas en el futuro, así como presentar resultados más objetivos y detallados sobre las labores de soporte realizadas a los órganos de dirección de empresa.

Por otro lado, es fundamental que el diseño proponga una solución que minimice los costes de mantenimiento de esta, y permita que el desarrollo de nuevas funcionalidades, así como la integración del soporte solicitado en posibles nuevas plataformas, sea sencillo y suponga el menor coste posible. Es por ello por lo que surgen una serie de requisitos técnicos, así como otros intrínsecos a la problemática planteada, que han de ser tenidos en consideración. La siguiente tabla recoge dichos requisitos:

Nº	Requisito	Descripción	Justificación
1	Operatividad 24/7.	La solución debe estar operativa todos los días de la semana, las 24 horas del día.	Bitnami ofrece sus productos de forma global. Por tanto, una solicitud de soporte puede darse en cualquier momento y esta ha de quedar registrada.
2	Escalabilidad Horizontal.	Debe ser posible aumentar o disminuir el número de réplicas que componen la solución de forma dinámica.	Para realizar un uso eficiente de los recursos y disminuir el coste de operatividad de la solución, esta debe tener la capacidad de adaptarse en función del tráfico recibido.

4	Kubernetes [6] como plataforma.	Uso de la plataforma Kubernetes para el despliegue de la aplicación.	Requisito impuesto por dirección cuya justificación se desarrollará más adelante.
5	Contenerización de la aplicación desarrollada.	La aplicación debe ser desplegada a través de un contenedor.	Intrínseca al uso Kubernetes, puesto que es una plataforma de despliegue de contenedores.
6	Uso de buenas prácticas en Kubernetes.	La solución ha de seguir las guías de buenas prácticas para el diseño y despliegue de aplicaciones en K8s.	Impuesta por el equipo de SRE de Bitnami.
7	Implementación de un sistema de integración y despliegue continuo.	Proveer de mecanismos que automaticen el testeo de nuevos cambios y permitan realizar ciclos cortos en el proceso de desarrollo.	Minimizar los costes de mantenimiento de la solución.
8	Monitorización.	Disponer de un sistema que recoja métricas del uso de la solución y los recursos que esta consume.	Obtener datos que permitan analizar el comportamiento de la solución y realizar mejoras en el mismo.
9	Seguridad.	La solución debe tener los mecanismos necesarios que garanticen los pilares de la seguridad: confidencialidad, integridad y disponibilidad.	Requisito de obligado cumplimiento para cualquier servicio puesto en producción.

Tabla 1 - Requisitos Técnicos de la solución

2 ESTUDIO PREVIO

Jugez un homme par ses questions plutôt que par ses réponses - Voltaire

Una vez se ha introducido los motivos que conllevan a la realización de este proyecto, se procede a un estudio más detallado de la situación, las tecnologías a utilizar y las soluciones existentes en el mercado.

2.1 Características de las plataformas de soporte

Tal como se comentó en la introducción, son cuatro las plataformas en las que los agentes de soporte de Bitnami atienden a los usuarios. Dichas plataformas tienen diferentes características, recogidas en la siguiente tabla:

	Prioridad	Información Sensible	Servicio Propio	Detección Automática
Community	Normal.	No.	Sí.	Sí.
GitHub	Normal.	No.	No.	No.
HelpDesk	Alta.	Sí.	No.	Sí.
Stackoverflow	Baja.	No.	No.	No.

Tabla 2 - Características de las plataformas de soporte

Descripción extendida de cada una de las características:

- Prioridad: importancia por necesidad de negocio que tiene dar respuesta a los casos de soporte recibidos en una plataforma de soporte concreta.
- Información sensible: determina si la información recibida de los usuarios que solicitan soporte ha de ser considerada como privada. Por ejemplo, la información relativa a contraseñas, credenciales o pagos se considera información sensible.
- Servicio propio: indica si la plataforma es un servicio propio gestionado por Bitnami o se hace el uso del servicio que ofrece un tercero.
- Detección Automática: señala si la plataforma dispone de métodos que permitan obtener de forma sencilla aquellas solicitudes de soporte que aún no han sido atendidas.

Como se puede observar, cada plataforma es distinta. Este proyecto pretende aunar las solicitudes recibidas en cada una de ellas bajo un mismo interfaz teniendo en cuenta sus diferencias.

El principal hándicap es la falta de un sistema de detección automática de solicitudes en todas las plataformas. Por tanto, es necesario utilizar un enfoque distinto que permita obtener dicha información con un método común a todas. Afortunadamente, todas las plataformas disponen de mecanismos de notificación de eventos. En otras palabras, podemos configurar todas ellas para enviar una notificación cada vez que se produce un cambio en una petición de soporte.

De ahí la necesidad de una aplicación que reciba y procese los eventos emitidos cada vez que se produce un cambio en un caso de soporte, para posteriormente enviar información a una plataforma de gestión de tareas que permita organizar la asignación de solicitudes de soporte a un agente concreto.

2.2 Soluciones existentes en el mercado

Son múltiples las soluciones existentes en el mercado que permiten dar respuesta a los retos y requisitos planteados en la introducción (ver apartado 1.4). En este apartado repasamos con detalle algunas de ellas.

2.2.1 Soluciones de gestión de soporte

Podemos encontrar decenas de soluciones en el mercado que permiten ofrecer soporte, a la par que gestionar el mismo de una manera eficiente. Sin ir más lejos, Bitnami hace uso de una de ellas, Zendesk. Otras plataformas con servicios similares pueden ser Freshdesk, Kayako, o Zoho [7].

Sin embargo, estas plataformas ofrecen soluciones completas y autocontenidas. Es decir, proveen de todas las herramientas necesarias para gestionar el soporte a través de ellas mismas, pero no están diseñadas para ser integradas con otras plataformas. Por tanto, no son válidas para nuestra problemática.

2.2.2 Soluciones de gestión de tareas

Respecto a herramientas de gestión de tareas, necesarias para optimizar el trabajo de los agentes de soporte, encontramos de nuevo un mercado saturado con muchas herramientas disponibles basadas en diferentes metodologías, que incluyen tanto planes gratuitos como planes de pago [8].

Debido a su popularidad y su potente API que nos permite automatizar procesos, la elegida para gestionar el trabajo de los agentes de soporte será Trello. Además, esta plataforma fue recomendada por compañeros que habían trabajado con ella anteriormente.

Trello [9] es una plataforma online diseñada para la gestión de tareas. Basada en la metodología Kanban [10], ofrece un sistema colaborativo ideal para la coordinación de equipos.

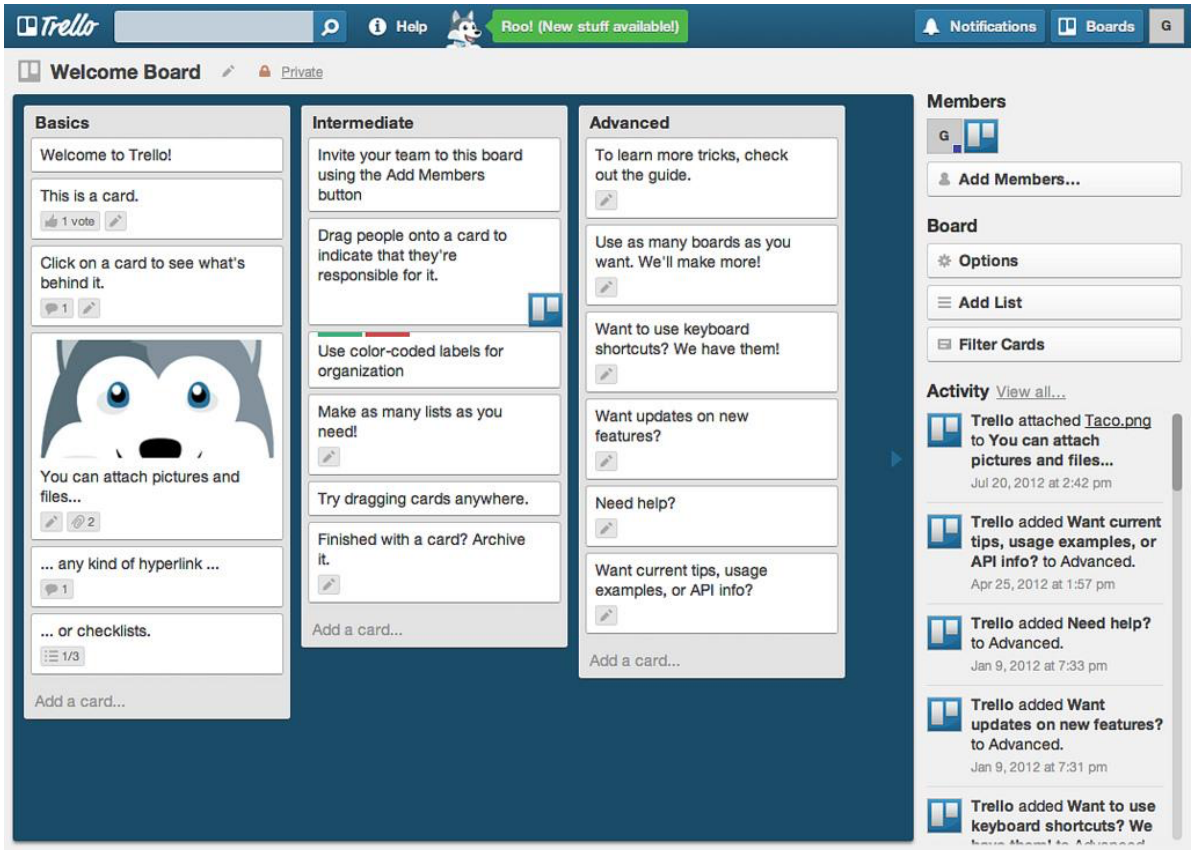


Figura 2 - Interfaz de la plataforma Trello

Reúne todas las características deseadas para gestionar el trabajo que los agentes de soporte realizan pues proporciona:

- Un único interfaz en forma de tablón que permite clasificar las peticiones de soporte recibidas, en tarjetas; y las diferentes plataformas de soporte de las que proceden, en columnas.
- Capacidad de añadir etiquetas a cada tarjeta que permita priorizar y catalogar cada petición de soporte.

- Colaboración en tiempo real entre los diferentes agentes de soporte, permitiendo asignar de forma inequívoca cada petición con el agente que está a cargo de ella.
- Gestión de fechas límite y antigüedad de las peticiones recibidos.

Por otro lado, esta plataforma ofrece una API REST que nos permite realizar operaciones de tipo CRUD, por lo que es ideal para su automatización. Además, toda la funcionalidad de dicha API está extensamente detallada en la documentación de Trello [11].

El uso de Trello da pie a que la aplicación a desarrollar, introducida en los objetivos del proyecto (ver apartado 1.4), sea bautizada como "Trello Updater" ("Actualizador de Trello", en inglés), ya que sintetiza de forma escueta su principal función.

2.2.3 Soluciones de contenerización

La solución que prácticamente monopoliza el mercado como motor de ejecución de contenedores es Docker [12]. Hay otras soluciones como CoreOS rkt [13] o Apache Mesos [14], pero gozan de mucha menos popularidad y disponen de un ecosistema de tecnologías complementarias menos amplio.

Por otro lado, Bitnami ofrece su amplio catálogo de aplicaciones en formatos diferentes como instaladores, máquinas virtuales o contenedores; siendo este último uno de los formatos que mayor popularidad está alcanzando por las numerosas ventajas que nos ofrece. En la actualidad, todas las soluciones de contenedores publicadas por Bitnami se basan en Docker. Por tanto, dicha tecnología es muy familiar a nivel interno puesto que es de uso diario.

Debido a la citada familiaridad, así como por ser la tecnología más popular a nivel mundial, se usa Docker para contenerizar nuestra solución (requisito #5).

2.2.4 Soluciones de orquestación de contenedores

De nuevo encontramos un mercado en el que una tecnología se ha impuesto sobre las demás, convirtiéndose en el estándar "de facto". Dicha tecnología es Kubernetes.

Otras soluciones como Docker Swarm o Mesos han sido relegadas a un prácticamente insignificante segundo plano, sin apenas cuota de mercado ni un ecosistema de tecnologías que complementen la solución. Por otro lado, el uso de Kubernetes es un requisito impuesto por dirección (requisito #4).

Kubernetes, abreviado como "K8s", es una plataforma open-source de orquestación de contenedores para su despliegue en producción. En otras palabras, permite realizar de una forma sencilla operaciones con contenedores como:

- Despliegue automático.
- Definición de mecanismos de recuperación ante fallos (requisito #1 y #9).
- Escalado (requisito #2).
- Balanceo de carga.
- Definición de políticas de control de acceso y bloqueo de tráfico (requisito #9).
- Monitorización (requisito #8).
- ...

Tiene su origen en un proyecto desarrollado por Google conocido como Borg, el cual

inspiró los CGroups [15]. Durante muchos años fue mantenido en secreto y finalmente fue dado a conocer públicamente en 2015 [16].

Kubernetes es un proyecto open-source desde 2014 y tiene a VMware, RedHat y Google como principales compañías contribuidoras del proyecto.

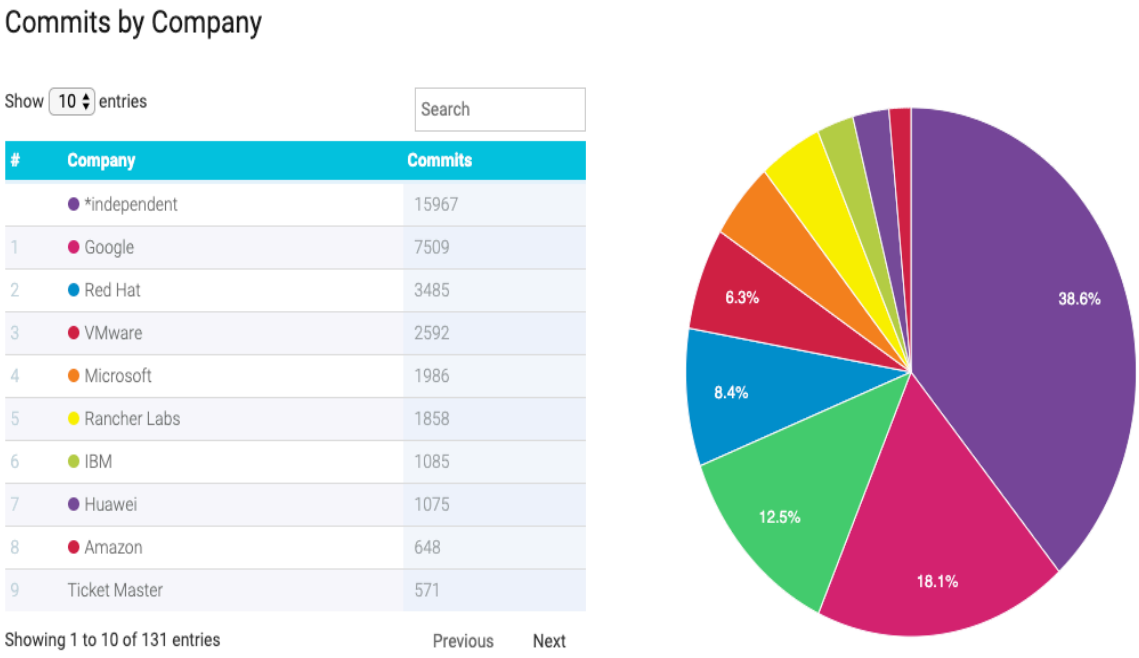


Figura 3 - Kubernetes, contribución por compañía [83].

Actualmente está gobernado por la CNCF (del inglés Cloud Native Computing Foundation) [17], una organización perteneciente a la Linux Foundation [18].

En los últimos años su popularidad se ha extendido de forma exponencial, contando en la actualidad con:

- Más de 2500 contribuidores.
- Más de 80.000 commits.
- Más de 25.000 usuarios en la herramienta de comunicación Slack [19].

Bitnami ha decidido apostar firmemente por la tecnología Kubernetes contribuyendo en algunos de los proyectos de su ecosistema.

En la actualidad, es uno de los principales contribuidores en el proyecto Helm [20], un gestor de "paquetes" denominados *charts* para Kubernetes. Por otro lado, ha aportado una serie de proyectos de creación propia tales como:

- Kubeapps [21] y Kubeless [22].
- Kubecfg [23] y Sealed Secrets [24].
- Bitnami Kubernetes Production Runtime [25].

Debido a esta apuesta tan ambiciosa en el proyecto tanto como por las numerosas ventajas que proporciona, es una gran prioridad a nivel de compañía la adopción interna

de dicha tecnología.

Esta idea se vea aún más reforzada con la adquisición en mayo de 2019 de Bitnami por parte de VMware [26], puesto que Kubernetes es una pieza fundamental de su porfolio.

2.2.5 Soluciones de integración y despliegue continuo

Al igual que ocurre con las herramientas de gestión de tareas, hoy en día encontramos un mercado muy variado de soluciones de integración y despliegue continuo (también denominadas herramientas CI/CD). Algunas de las soluciones open-source más populares son GitLab, CircleCI o TeamCity [27].

Para implementar el sistema CI/CD (requisito #7) se decide utilizar la herramienta de automatización de código abierto Jenkins [28]. Esta herramienta consiste en un servidor que permite definir y ejecutar “trabajos” (del inglés “jobs”) que automatizan una serie de operaciones.

Son múltiples los usos que se hacen de Jenkins en Bitnami, siendo la familiaridad con esta solución uno de los motivos por los que se opta por la misma. Además, el equipo de SRE ha desarrollado una serie de librerías que hacen más fácil la creación de trabajos que gestionan el ciclo de vida de una aplicación en Kubernetes.

3 DISEÑO GENERAL DE LA SOLUCIÓN

It's necessary sometimes to take one step backward
to take two steps forward – Vladimir Lenin

En este apartado, hablaremos de forma genérica de la solución que se plantea para lograr los objetivos definidos. No es de esperar mucho detalle en este apartado, pues sólo se busca plantear una visión global del proyecto, para hacer así más sencillo seguir el resto de la memoria.

También se desglosará el proyecto en las distintas fases que lo comprenden, incluyendo detalle de cuál fue mi participación personal en cada una de ellas, y en cuáles participaron compañeros de Bitnami.

3.1 Necesidad de una aplicación de integración de solicitudes de soporte

En el estudio previo, se destacaba la posibilidad de usar las notificaciones de eventos, disponibles en las diferentes plataformas en la que Bitnami ofrece soporte a sus usuarios, como método de obtención de información sobre las solicitudes de soporte recibidas. Por otro lado, se eligió Trello, como herramienta de gestión de tareas. El uso de dicha plataforma está condicionado a poder utilizar la información obtenida en las citadas notificaciones para poder realizar llamadas al API que Trello expone.

Sin embargo, cada plataforma implementa dichas notificaciones de manera diferente y tan sólo la plataforma Community es un servicio propio que pueda ser modificado conforme a nuestros intereses.

Es por esto por lo que se requiere desplegar una aplicación que recolecte las notificaciones procedentes de cada una de las plataformas, las procese y realice las correspondientes llamadas a la API de Trello.

3.2 Diseño general

La idea tras Trello Updater, no es más que la de utilizar un intermediario que traduzca las notificaciones recibidas de cada una de las plataformas de soporte a un lenguaje que Trello comprenda.

En el siguiente esquema de caja negra, podemos ver reflejado cómo nuestra aplicación actúa como intermediario:

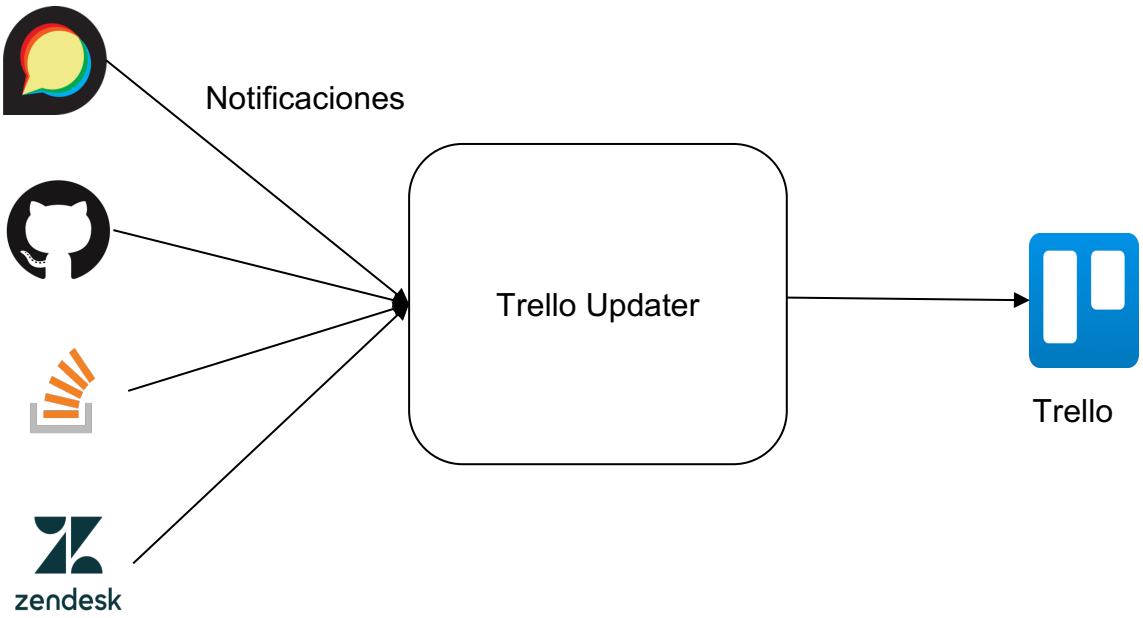


Figura 4 - Trello Updater: diagrama de caja negra.

Tal como se detalla más adelante es muy importante que Trello Updater no guarde información sobre las operaciones que realiza en ninguna base de datos. Esto nos permitirá simplificar su implementación sustancialmente. Este detalle de diseño no ha de resultar en la pérdida de ningún tipo de dato, puesto que la plataforma destino, Trello, será la encargada de almacenar toda la información con sus propios mecanismos.

No obstante, si que será útil para el diagnóstico y prevención de errores, que la aplicación genere y exponga una serie de logs y métricas. Este asunto será extensamente tratado en el apartado 6.3.

La figura 4 nos muestra una simplificación extrema de la solución. La siguiente figura nos muestra un esquema más completo de todos los elementos a desarrollar (resaltados en rojo), incluyendo elementos para garantizar alta disponibilidad como balanceadores de carga o mecanismos de seguridad, como la gestión de certificados TLS. Otros elementos incluidos en la solución final, como aquellos relativos a la monitorización, son excluidos en esta figura.

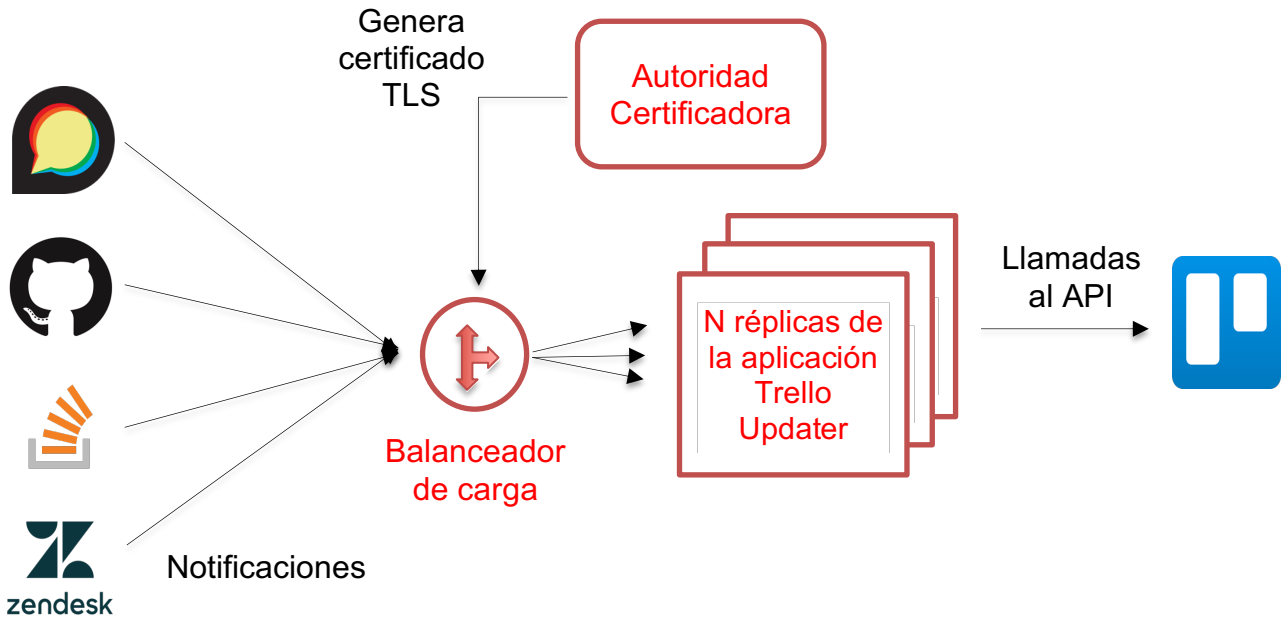


Figura 5 - Trello Updater: arquitectura

3.3 Etapas del proyecto

El proyecto se divide en dos etapas principales, que dividen a su vez en varias fases:

- Etapa 1: Diseño e implementación de la aplicación Trello Updater en estado beta. El principal objetivo de esta etapa es tener una solución operativa a disposición de los agentes de soporte para que estos puedan evaluarla. Esta fase también incluye implementar mejoras en Trello Updater en base al feedback obtenido del equipo de soporte.
- Etapa 2: Puesta en producción de la aplicación conforme a los requisitos planteados para el proyecto.

Es importante resaltar que la segunda etapa parte del trabajo realizado en la primera, pero desecha algunos de los elementos implementados en ésta. Esto se debe a que, en la primera etapa, no se tienen en cuenta algunos de los requisitos del proyecto al desplegar la solución para así reducir los tiempos de desarrollo. Al hacer esto, el equipo de soporte puede evaluar la solución mucho antes y, de forma paralela a esta evaluación, se puede avanzar en la segunda etapa.

El siguiente diagrama muestra las fases de la etapa 1 según su duración temporal:

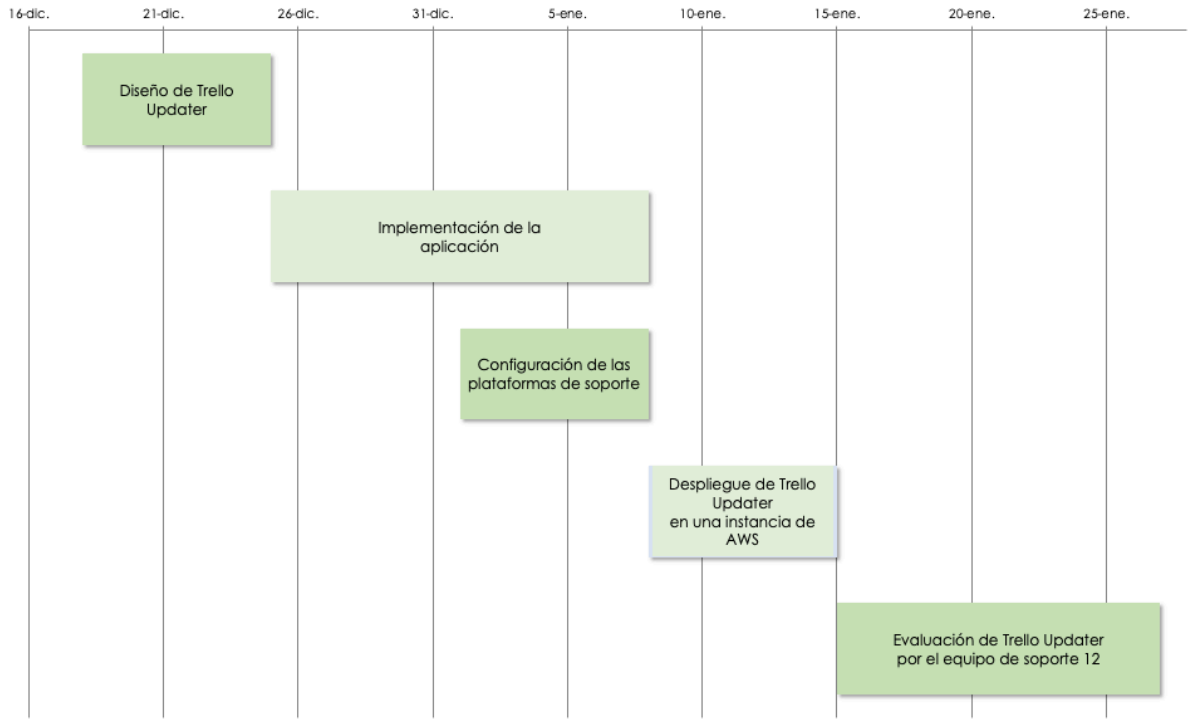


Figura 7 - Fases de la etapa 1.

El siguiente diagrama muestra las fases de la etapa 2 según su duración temporal:

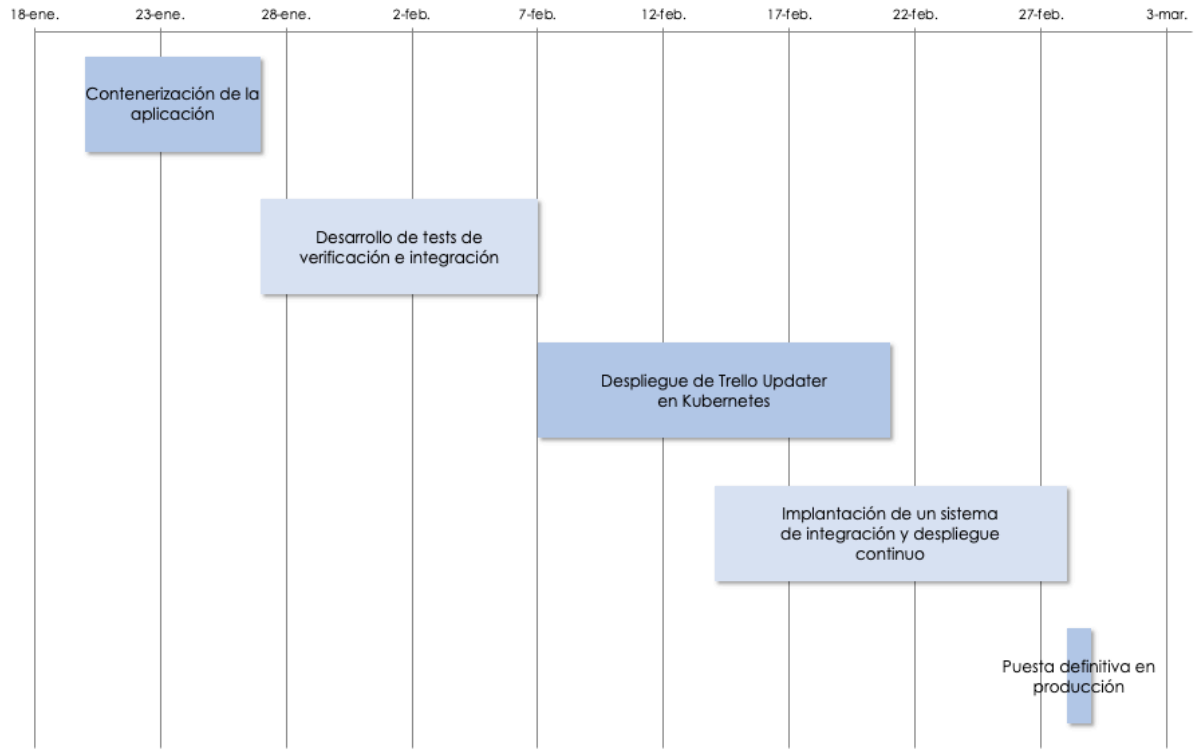


Figura 6 - Fases de la etapa 2.

En la siguiente tabla puede obtenerse información sobre las principales tareas a realizar en cada una de las fases. En verde, se listan aquellas fases pertenecientes a la etapa 1; en azul, las pertenecientes a la etapa 2:

Fase.	Tareas a realizar.
Diseño de Trello Updater.	• Diseñar la arquitectura. • Elegir el patrón de diseño. • Elegir las tecnologías a utilizar.
Implementación de la aplicación.	• Programar la solución previamente diseñada.
Configurar las plataformas de soporte.	• Configurar las plataformas de soporte para enviar webhooks a Trello Updater. • Crear un script para enviar webhooks manualmente.
Despliegue de Trello Updater en una instancia de AWS.	• Desplegar una instancia en AWS. • Instalar y configurar Trello Updater en dicha instancia. • Monitorización básica.
Evaluación de Trello Updater por el equipo de soporte.	• Formar al equipo de soporte sobre cómo utilizar Trello. • Evaluación de la solución por parte del equipo de soporte. • Programar las mejoras en Trello Updater en base a las sugerencias del equipo.
Contenerización de la aplicación.	• Adaptar la aplicación para ser contenerizada. • Crear una imagen de Docker que contenga la aplicación instalada y lista para ser desplegada.
Desarrollo de tests de verificación e integración.	• Programar tests de verificación e integración para Trello Updater. • Crear un script que permita ejecutar los tests en batería. • Crear una imagen de Docker que contenga los tests.
Despliegue de Trello Updater en Kubernetes.	• Crear los diferentes objetos que permiten desplegar Trello Updater en Kubernetes. • Configurar los servicios de Kubernetes que nos permitan exponer Trello Updater, obtener sus logs y monitorizarlo.
Implantación de un sistema de integración y despliegue continuo.	• Configurar una herramienta de CI/CD para desplegar cambios en Trello Updater en un entorno de preproducción, realizar tests, y desplegar los cambios en el entorno de producción una vez validados.
Puesta definitiva en producción.	• Desplegar Trello Updater en el clúster de Kubernetes de producción. • Reconfigurar las plataformas de soporte para enviar los

	webhooks a la URL definitiva. • Eliminar la instancia de AWS con la solución beta.

Tabla 3 - Tareas a realizar en cada fase.

En la siguiente tabla puede obtenerse información sobre las tecnologías y herramientas (incluyendo plataformas) utilizadas en las fases del proyecto, así como las personas involucradas:

Fase.	Personas involucradas.	Tecnologías utilizadas.	Herramientas utilizadas.
Diseño de Trello Updater.	• Javier J. Salmerón García. • Juan Ariza Toledano.		• Microsoft Excel. • Microsoft Word.
Implementación de la aplicación.	• Juan Ariza Toledano.	• JavaScript. • Node.js. • JSON. • Express.js. • Let's Encrypt. • Git.	• VSCode. • Trello.
Configurar las plataformas de soporte.	• Javier J. Salmerón García. • Juan Ariza Toledano.	• JavaScript. • Ruby on Rails. • Git. • Tampermonkey.	• VSCode. • Discourse. • GitHub. • ZenDesk. • Stackoverflow.
Despliegue de Trello Updater en una instancia de AWS.	• Juan Ariza Toledano.	• MEAN stack.	• AWS.
Evaluación de Trello Updater por el equipo de soporte.	• Equipo de soporte de Bitnami. • Juan Ariza Toledano.	• JavaScript. • Node.js. • JSON. • Express.js. • Git. • Tampermonkey.	• VSCode. • Trello. • Discourse. • GitHub. • ZenDesk. • Stackoverflow.
Contenerización de la aplicación.	• Juan Ariza Toledano.	• Docker.	• VSCode. • Docker Engine.
Desarrollo de tests de verificación e integración.	• Juan Ariza Toledano.	• JavaScript. • Node.js. • JSON. • Mocha. • Docker.	• VSCode. • Docker Engine.

Despliegue de Trello Updater en Kubernetes.	• Juan Ariza Toledano.	• Kubernetes. • Kubecfg. • Jsonnet. • Sealed Secrets. • Cert Manager. • Ingress Controller. • External DNS. • EFK. • Grafana. • Prometheus.	• VSCode. • GCP/GKE.
Implantación de un sistema de integración y despliegue continuo.	• Juan Ariza Toledano.	• Groovy. • Kubernetes.	• VSCode. • Jenkins. • GCP/GKE.
Puesta definitiva en producción.	• Juan Ariza Toledano. • Juan José Ciarlante.	• Kubernetes. • EFK. • Grafana. • Prometheus.	• Jenkins. • GCP/GKE. • Trello.

Tabla 4 - Tecnologías utilizadas en cada fase.

4 TRELLO UPDATER

No tengas miedo de la perfección, nunca la alcanzarás - Salvador Dalí

Tanto en el estudio previo, como el diseño general de la solución, se desglosaban las razones por las que desarrollar una aplicación que sirva de enlace entre las distintas plataformas en las que se atienden solicitudes de soporte y la plataforma Trello.

En esta sección se detallan los aspectos de implementación dicha aplicación, Trello Updater, así como los detalles de cómo se configuran las plataformas en las que Bitnami ofrece soporte a sus usuarios para enviar notificaciones a la aplicación.

4.1 ¿Qué es y para que se utiliza Trello Updater?

Trello Updater es una aplicación web que expone un API REST. Está desarrollada en JavaScript utilizando el entorno de ejecución Node.js [29] y el framework Express [30].

Esta aplicación se utiliza para colectar las notificaciones procedentes de diferentes plataformas de soporte y extraer su información relevante, para luego adaptarla, clasificarla y reenviarla a un tablón en la plataforma Trello. En otras palabras, Trello Updater realiza tres funciones:

- Traducir las notificaciones a un lenguaje compatible con la plataforma Trello.
- Eliminar aquella información recibida en las notificaciones que no es interesante para la gestión del soporte.
- Catalogar de manera apropiada la información recibida en la plataforma Trello.

4.2 Configuración de las plataformas de soporte para enviar notificaciones a Trello Updater.

Las notificaciones de eventos enviados a Trello Updater tendrán forma de peticiones HTTP de tipo POST conocidas como webhooks [31]. Se procederá a configurar cada plataforma de soporte para que envíen webhooks a nuestra aplicación por medio de los mecanismos que estas ofrecen tal y cómo se detalla en la siguiente figura:

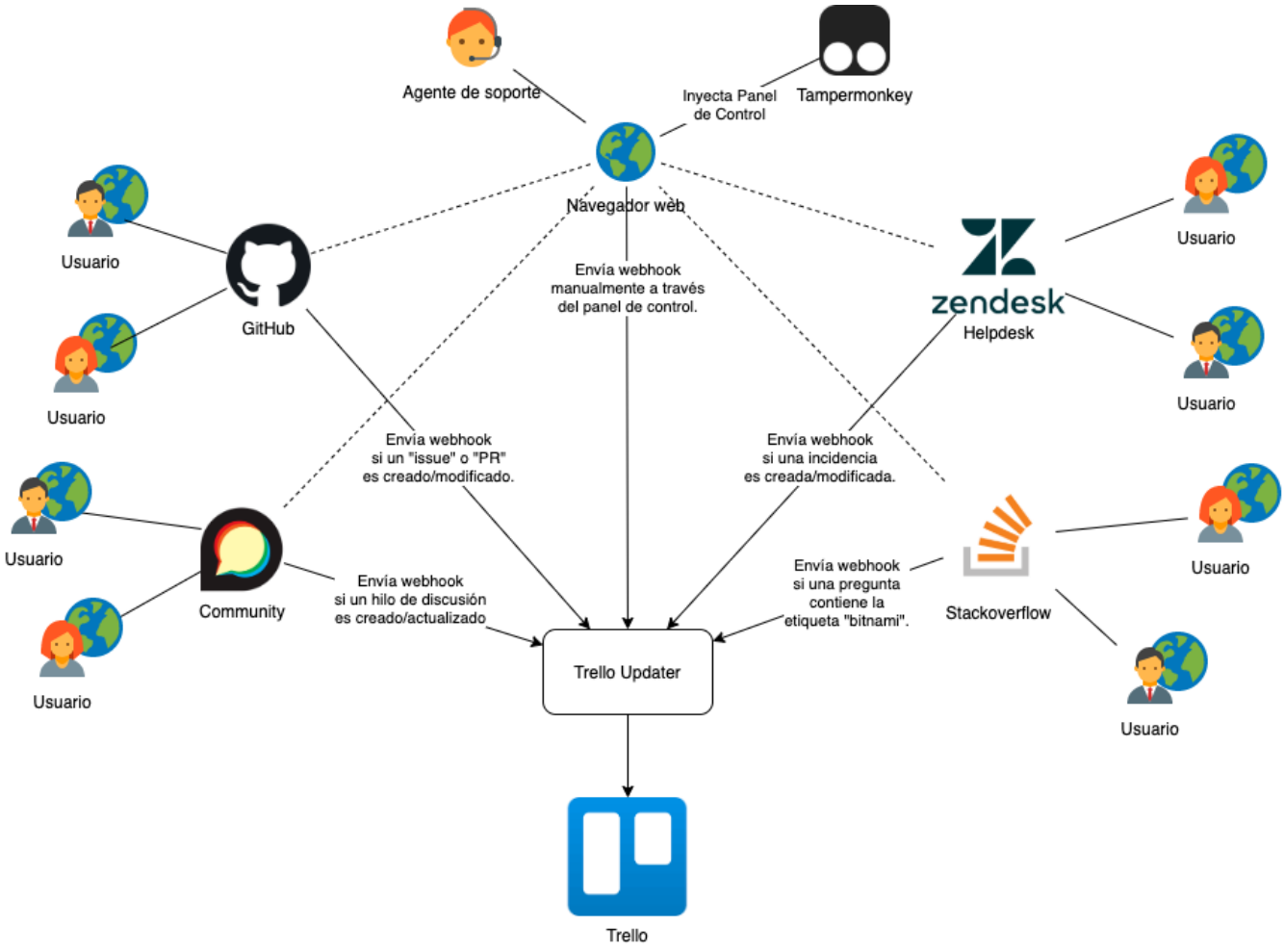


Figura 8 - Envío de webhooks desde las plataformas de soporte.

En esta sección se detalla cómo se implementa dicha configuración en cada una de las plataformas.

4.2.1 Configuración de Community.

Community está basado en la aplicación web Discourse [32]. Dicha aplicación es un proyecto open-source basado en el lenguaje de programación Ruby [33] y el framework Ruby on Rails [34].

Por defecto, Discourse no ofrece una forma sencilla de generar webhooks pero se cuenta con la ventaja de estar desplegada en un sistema gestionado por Bitnami. Para poder adaptar Community a nuestras necesidades, se decide implementar un plugin para Discourse, al que se llama community-trello. Dicho plugin es open-source y está disponible en la plataforma GitHub [35].

Gracias al plugin desarrollado, se configura Community para enviar webhooks cada vez que se abre un nuevo hilo de discusión en el foro, o hay una modificación en alguno existente (ver anexo A).

4.2.2 Configuración de HelpDesk.

Zendesk nos permite generar webhooks a través de dos elementos: “triggers” y “extension targets” [36].

- Los “triggers” son acciones que son ejecutadas cada vez que una solicitud de soporte es creada o modificada.
- Los “extension targets” nos permiten definir los destinos a los que enviar el resultado de un “trigger” concreto.

En nuestro caso, configuraremos un “extension target” que represente a la aplicación Trello Updater. Así, un webhook es enviado a Trello Updater cada vez que una solicitud de soporte es creada o modificada (ver anexo A).

4.2.3 Configuración de GitHub.

GitHub permite configurar notificaciones vía webhooks en todos los repositorios pertenecientes a una organización de forma sencilla.

Usando esta funcionalidad, configuramos GitHub de forma que un webhook es enviado cada vez que un “Pull Request“ o “Issue” es creado o modificado en cualquier repositorio perteneciente a la organización Bitnami (ver anexo A).

4.2.4 Configuración de Stackoverflow.

Se procede a crear una cuenta en esta plataforma, y se configura de tal forma que se envíe un webhook cada vez que una pregunta en StackOverFlow es creada o modificada conteniendo la etiqueta “bitnami” (ver anexo A).

4.2.5 Tampermonkey.

Si comparamos los webhooks en formato JSON [37] generados por cada una de las plataformas de soporte, observamos que el utilizado en HelpDesk tiene algunos campos extra que son de gran interés para la organización del trabajo de los agentes de soporte: “tags”, “phtask” y “docsReason”. Estos campos se rellenan en base a nuevos elementos añadidos al interfaz de Zendesk que nos permiten añadir información extra sobre una solicitud de soporte en cuestión.

Para disponer de una correcta clasificación y etiquetado en Trello, se necesita proveer al

agente de soporte de la capacidad de añadir información sobre estos campos en las demás plataformas de soporte. Sin embargo, se cuenta con el hándicap de no poder modificar a nuestro antojo GitHub y Stackoverflow, puesto que son servicios externos.

Como respuesta a estar problemática, se propone el uso de herramientas que permitan inyectar código JavaScript [38] en el navegador utilizado por los agentes de soporte. Es decir, al visitar la plataforma de soporte, el navegador utiliza un plugin para alterar la web mostrada en el lado del cliente. Para este propósito se decide utilizar la herramienta Tampermonkey [39].

Por tanto, se crea un script en lenguaje JavaScript que se inyecta al visitar a través del navegador GitHub, Community y StackOverFlow y añade un panel que permita tener los citados campos extra. El agente de soporte podrá utilizar este panel para enviar webhooks a Trello Updater con información extra para su correcta clasificación y etiquetado (ver anexo A).

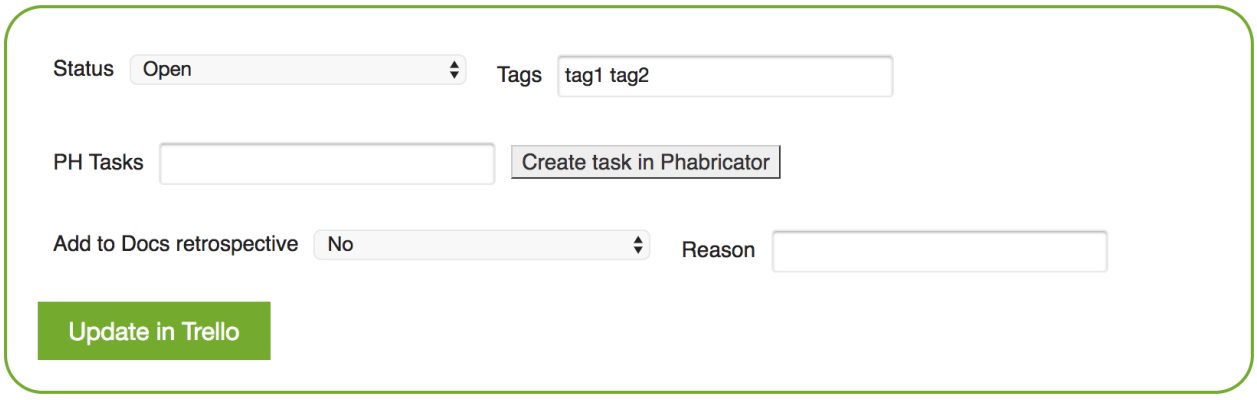


Figura 9 - Panel inyectado vía Tampermonkey.

4.3 Arquitectura y Funcionamiento.

En vista de que Trello Updater no necesita almacenar ninguna información, es decir, tan sólo procesa los webhooks recibidos y realiza la llamada correspondiente a la API de Trello; podemos optar por usar una aplicación sin estado (del inglés stateless). Este tipo de aplicaciones son mucho más sencillas y como se detallará más adelante, simplifican significativamente su migración a Kubernetes.

Trello Updater expondrá una API REST con un sólo método que acepte peticiones de tipo POST. El envío de webhooks a dicha API supondrá el procesamiento de estos y la posterior sincronización con Trello.

En la siguiente figura, podemos observar el diseño del flujo desde que se produce el envío de un webhook a Trello Updater hasta que este procesa la información recibida, y procede a actualizar la plataforma Trello a través de su API:

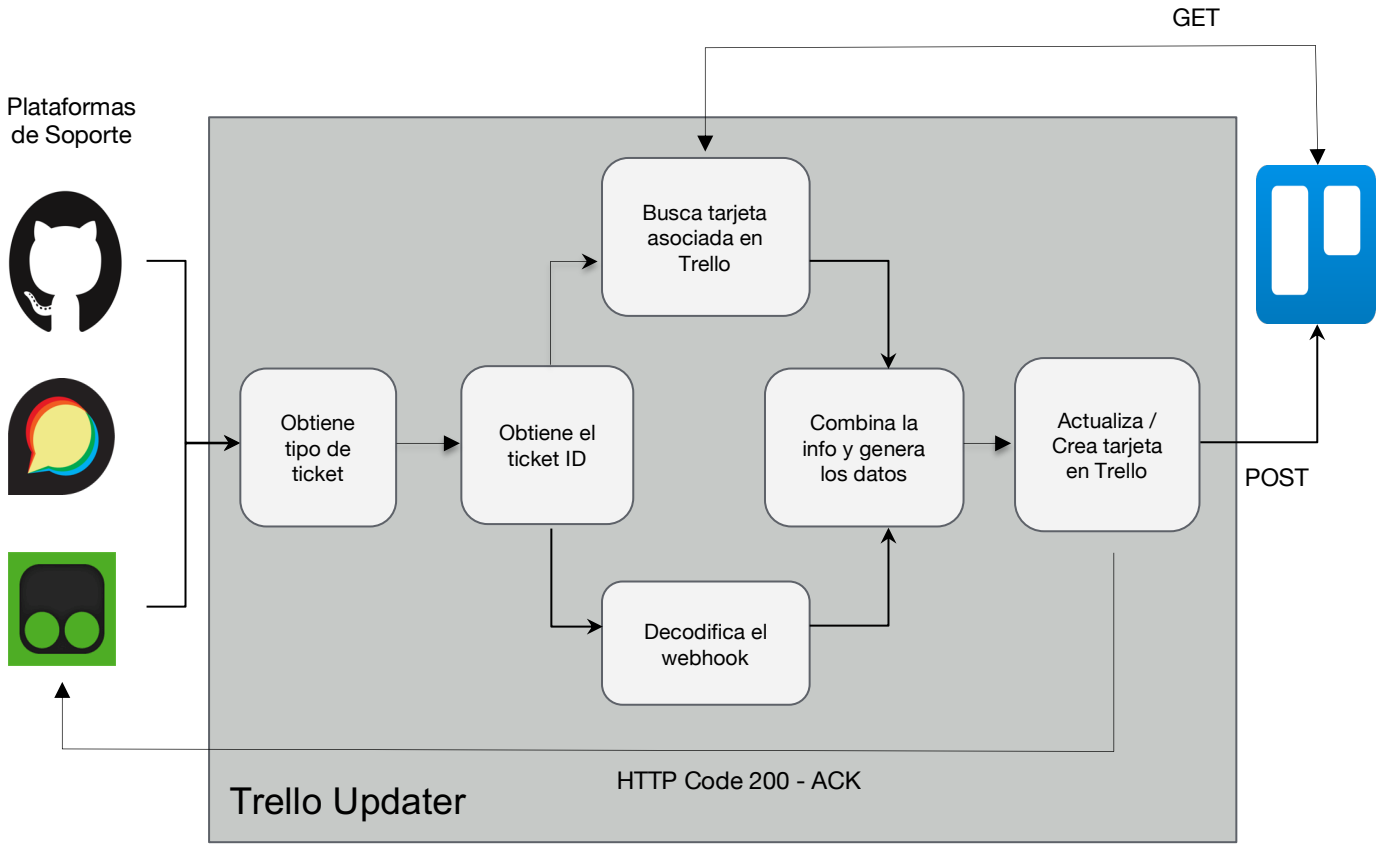


Figura 10 - Trello Updater: diagrama de caja blanca.

En la siguiente tabla analizamos con más detalle cómo funciona la aplicación, explicando con detalle qué ocurre en cada una los pasos del diagrama anterior en orden cronológico:

Paso	Funcionamiento
1. Trello Updater recibe una notificación tipo POST a través de su API	Al recibir una notificación de tipo POST en la API REST se crea un objeto (utilizando POO, Programación Orientada a Objetos) denominado “ticket” que contiene la información recibida en el webhook en formato JSON. Este “ticket” representa una solicitud de soporte.

2. Obtención del tipo de ticket.	En función a los campos que contiene el JSON recibido, se puede distinguir el tipo de ticket del que se trata. El tipo de ticket viene condicionado, principalmente, por la plataforma de soporte de procedencia. Siguiendo el patrón de diseño Factory Method [40], el objeto creado en el paso 1 utilizará una clase diferente en función del tipo de ticket, ver anexo B.3 para mayor detalle sobre las clases utilizadas.
3. Obtención del ID del ticket.	Cada ticket posee un identificador único. Este se usará para crear tarjetas en Trello que puedan ser localizadas de manera inequívoca a través de su ID. Dicho identificador es obtenido en función a un campo de valor único incluido en cada webhook recibido.
4. Búsqueda de una tarjeta asociada en Trello.	Una vez obtenido el ID del ticket se realiza una consulta, enviando una petición tipo GET al API de Trello, para determinar si ya existe una tarjeta asociada a ese ticket. Este caso puede darse cuando se recibe una notificación que actualiza el estado de un caso de soporte previamente creado.
5. Decodificación del webhook.	Usando un proceso paralelo al paso 4, se procesa el resto de información obtenida del webhook para adaptarla a un formato compatible con la API de Trello.
6. Combinación de la información obtenida y generación de los datos a enviar a Trello.	En función a la respuesta obtenida en el paso 4 pueden darse dos casos: • Caso A) No existe ninguna tarjeta en Trello asociada al ID del ticket. En este caso, se utiliza la información procedente del paso 5 tal cual para generar los datos a enviar a Trello que permitan crear una nueva tarjeta. • Caso B) Existe una tarjeta en Trello asociada al ID del ticket. En este caso se combina la información procedente del paso 5 con la información obtenida en el paso 4 sobre la tarjeta existente, para generar los datos a enviar a Trello que permitan actualizar el estado de la tarjeta.
7. Actualización / Creación de una tarjeta en Trello.	Usando los datos generados en el paso 6 se crea o actualiza una tarjeta en Trello, enviando una petición de tipo POST a su API. En este paso se propaga a la plataforma que envió la notificación el código de estado recibido de Trello al realizar la petición tipo POST.

Tabla 5 - Funcionamiento de Trello Updater.

4.4 Implementación.

Una vez sabemos cual es la arquitectura utilizada en Trello Updater y como este gestiona las notificaciones recibidas, veamos los detalles de implementación de la solución.

4.4.1 Node.js y Express.

Puesto que se pretende desarrollar la aplicación en el menor tiempo posible, es esencial usar una tecnología con la que se esté bien familiarizado. Es por esto, como por sus numerosas ventajas, que se decide implementar la aplicación utilizando la tecnología Node.js, un entorno del lado del servidor basado en el lenguaje de programación JavaScript.

Node.js posee una serie de características que son ideales para el propósito de la solución a implementar:

- Programación asíncrona.
- Bajo coste de infraestructura.
- Buena gestión de paquetes.
- Alta escalabilidad.
- Rápido desarrollo.

Por otro lado, se elige el framework Express, por su sencillez y popularidad, para organizar la estructura del código. Además, este framework dispone de librerías que permiten crear servidores web sencillos con soporte para http y https, por lo que es perfecto para crear el servidor web que exponga la API de Trello Updater (ver anexo B.1).

En el fragmento de código siguiente se puede observar cuan sencillo resulta exponer un método “ticket” a través de una API con Express. Dicho método acepta peticiones tipo POST en formato JSON:

```
const bodyParser = require('body-parser');
const express = require('express');
...
const app = express();
...
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
...
app.use('/healthcheck', require('express-healthcheck')());
...
// Add headers
app.use(function(req, res, next) {
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS');
  ...
});
// Ticket method definition
app.post('/ticket', (req, res) => {
  ...
});
...
app.listen(port);
```

Como se puede observar, también se añade un mecanismo de comprobación de salud

(del inglés “Health Check”) de la aplicación. Este será muy útil en el futuro para detectar fácilmente anomalías en su correcto funcionamiento.

4.4.2 Instalación de Trello Updater.

Una de las grandes ventajas que ofrece Node.js es su gestor de paquetes NPM [41]. Este nos permite gestionar de una forma muy sencilla todas las dependencias que usa nuestra aplicación (ver anexo B.2). Sólo es necesario generar un fichero “package.json” siguiendo las especificaciones oficiales [42].

Una vez se han indicado las dependencias, la instalación de la aplicación y todas sus dependencias se resume a la ejecución del siguiente comando:

```
$ npm install
```

Tras instalar todas las dependencias, basta con ejecutar el comando siguiente para iniciar la aplicación:

```
$ npm start
```

4.4.3 Configuración de Trello Updater.

Trello Updater se configura a través de un fichero configuración ***'settings.json'.*** Este fichero tiene el siguiente formato:

```
{
  "allowedOrigins": [
    "https://bitnami.zendesk.com",
    "https://community.bitnami.com",
    "https://stackoverflow.com",
    "https://serverfault.com",
    "https://github.com"
  ],
  "runMode": "production",
  "port": 3000
}
```

Como se puede observar, podemos configurar el puerto en el que se expondrá la API, las URL(s) de origen desde las cuales se aceptan conexiones y el modo de funcionamiento.

Son dos los modos de funcionamiento soportados: “production” y “testing”. El primero se utilizará para la puesta en marcha de la aplicación Trello Updater, mientras que el segundo será utilizado para someter a la aplicación a una batería de tests de validación e integración. En el apartado 5.2 estudiaremos en mayor detalle los tests citados.

4.5 Despliegue de Trello Updater en una instancia de AWS.

En un principio, y a pesar de que el objetivo final es que Trello Updater sea desplegada en un torno de Kubernetes, se decide desplegar la aplicación en una máquina virtual en nuble pública.

La razón de hacer esto es tener una solución, en estado beta, operativa para que el equipo de soporte la evalúe y aporte feedback en el menor tiempo posible. Esto nos permitirá trabajar en el resto de los pasos necesarios para desplegar Trello Updater en un entorno de Kubernetes de manera paralela a la evaluación de la aplicación por aquellos que harán uso de esta.

Por tanto, este despliegue inicial no es más que una solución provisional. Para llevarlo acabo, se crea una máquina virtual o instancia en la nube pública de Amazon, AWS (de su acrónimo en inglés, Amazon Web Services).

Este despliegue resulta muy sencillo gracias a las soluciones que Bitnami ofrece en diferentes formatos conocidas como “stacks”, siendo uno de estos formatos las imágenes de nube que permiten crear máquinas virtuales en plataformas como AWS. Las “stacks” empaquetan bajo la misma solución todas las herramientas necesarias para un propósito concreto. Por ejemplo, un desarrollador web que quisiera crear un sitio web basado en PHP [43], usando Apache [44] como servidor web y una base de datos MySQL [45]; puede utilizar la “stack” conocida como LAMP [46] que le proporciona todas esas herramientas ya instaladas.

Puesto que buscamos desarrollar una aplicación Node.js, creamos la instancia en AWS basándonos en la MEAN stack [47], consultar anexo C.1 para mayor detalle. Esta nos ofrece una serie de componentes instalados y configurados que nos serán esenciales tales como:

- Node.js.
- NPM.
- Express.
- Lego [48]. Un cliente de Let’s Encrypt utilizado para la generación de los certificados TLS usados en Trello Updater [49].
- GIT [50]. Utilizado para el control de versiones de código.

MEAN incluye otros componentes tales como Apache, Angular.js [51] o MongoDB [52] los cuáles serán desactivados al no ser utilizados en este proyecto:

- MongoDB no es necesario ya que, como hemos mencionado previamente, Trello Updater será una aplicación sin estado y no guardará datos en ninguna base de datos.
- Apache tampoco es requerido, puesto que el framework Express proporciona librerías propias para levantar un servidor web.
- Angular.js es un framework para desarrollar interfaces de usuario y también podemos desactivarlo, ya que Trello Updater sólo expondrá una API.

Una vez tenemos la instancia en AWS lista, nos conectaremos por SSH a esta y subimos el código y ficheros de configuración de la aplicación Trello Updater. Luego, instalamos la aplicación como se indica en el apartado 4.4.2.

Una vez el entorno está listo para ejecutar la aplicación, se realizan las siguientes acciones previas a iniciarla:

- Se añade un nuevo servicio al gestor de servicios de Linux systemd [53], que nos permita realizar las operaciones básicas (start, stop, status, restart) con nuestra aplicación, Trello Updater.
- Se configuran una monitorización muy básica de la aplicación haciendo uso de la herramienta de monitorización de servicios monit [54], disponible en cualquier distribución de Linux.

Consultar el anexo C.2 para mayor detalle sobre la instalación de Trello Updater.

Por último, se inicia Trello Updater, y se configuran las plataformas de soporte tal como se detalla en el apartado 4.2 para que estas comiencen a enviar webhooks a la IP asociada a la instancia de AWS en la que hemos desplegado la aplicación. A partir de momento Trello Updater comienza a actualizar de manera automática un tablón creado en Trello, que usará el equipo de soporte durante este periodo provisional en el que se familiarizan con la nueva forma de trabajo y compartirán sus impresiones sobre esta.

5 CONTENERIZACIÓN Y TESTEO

Simplicity is complex. It's never simple to keep things simple. Simple solutions require the most advanced thinking. - Richie Norton

Una vez se tiene desplegada la aplicación Trello Updater en una instancia de AWS para su evaluación por los agentes de soporte, se pone el foco en desarrollar el resto de los elementos deseados en nuestra solución definitiva: monitorización, sistema CI/CD, uso de Kubernetes, etc.

Uno de los requisitos consecuencia de usar la plataforma Kubernetes para albergar la solución definitiva, era la contenerización de la aplicación. Por otro lado, disponer de una serie de tests que permitan realizar cambios el código y su puesta en producción con las garantías de que el servicio no sea interrumpido, es algo intrínseco a la implementación de un sistema de integración y despliegue continuo, otro de los requisitos.

La contenerización y la creación de los tests serán los primeros pasos en nuestro camino hacia la solución definitiva.

5.1 Contenerización.

Tal como se introdujo en el estudio de soluciones del mercado (ver apartado 2.2.3), se elige Docker como herramienta para contenerizar nuestra aplicación, Trello Updater.

De manera similar a las imágenes ISO utilizadas para desplegar una máquina virtual, Docker requiere de unas instantáneas del estado de un contenedor denominadas imágenes de Docker para desplegar un contendor.

La definición de una imagen de Docker se realiza a través de unos ficheros denominados

Dockerfile. Siguiendo las guías buenas prácticas en la escritura de estos ficheros [55], se crea el siguiente Dockerfile:

```
FROM bitnami/node:10

# Install required system packages
RUN install_packages procps
# Mount application files
COPY app /app
WORKDIR /app
# Install application and start it
RUN npm install
CMD ["npm",  "start"]
```

Como se puede observar es muy sencillo. Tomando con imagen base la imagen de Node.js de Bitnami, la cual nos proporciona Node.js ya instalado, simplemente es necesario montar el código de nuestra aplicación e instalarla utilizando NPM. Por tanto, el único requerimiento es contar con el código de la aplicación en el mismo directorio que el Dockerfile tal y como se describe en el siguiente árbol de ficheros:

```
.
├── Dockerfile
└── app
    ├── ...
    ├── package.json
    └── server.js
```

Una vez se tiene la definición de la imagen de Docker, basta con usar el comando “docker build” para construirla. Podemos usar un sencillo script para crear imágenes únicas basándonos en las etiquetas asignadas:

```
#!/bin/bash

imageTag=$(date +"%Y%m%d%H%M%S")
imageName="trello-updater:$imageTag"
docker build . --tag "$imageName"
```

Tras la construcción de la imagen de Docker, esta pasa a formar parte del repositorio local de imágenes y podemos utilizarla para crear un contenedor en el que nuestra aplicación sea ejecutada. El comando “docker run” nos permite realizar dicha acción de forma sencilla.

Por su diseño, Trello Updater espera que una serie de ficheros de configuración/credenciales en formato JSON se encuentre en las rutas adecuadas. En el siguiente ejemplo, podemos observar como crear un contenedor que ejecute la aplicación

montando cada uno de los ficheros en la ruta adecuada:

```
docker run -it trello-updater -p 3000:3000 \
-v ./settings.json:/app/settings/settings.json \
-v ./agentstrello.json:/app/lib/trello/agentstrello/agentstrello.json \
-v ./comunityagents.json:/app/lib/trello/comunityagents/comunityagents.json \
-v ./githubagents.json:/app/lib/trello/githubagents/githubagents.json \
-v ./helpdeskagents.json:/app/lib/trello/helpdeskagents.json
```

Como se puede observar, el puerto 3000 es expuesto al crear el contenedor para así poder acceder a la API de Trello Updater desde el exterior.

El hecho de poder montar de forma sencilla ficheros en el contenedor que es creado, nos permite algo esencial como es la separación del código de la aplicación de las credenciales que esta utiliza. Por motivos de seguridad, dichos ficheros no han de ser parte del repositorio que contiene el código ya que contienen las credenciales de acceso de los agentes de soporte a la plataforma Trello (entre otras).

5.2 Testeo.

Tras contenerización de la aplicación, el siguiente paso en busca de una solución de garantías es disponer de una buena cobertura de tests que nos permita introducir cambios en el código (y su posterior puesta en producción) con la seguridad de que ninguna funcionalidad existente se vea afectada.

Para ello, realizaremos una serie de pruebas de validación, funcionalidad e integración [40], detallados en las secciones 5.2.1 y 5.2.2, que nos permitan testear tanto el correcto funcionamiento de la aplicación como su interactuación con el resto de los servicios implicados: las plataformas de soporte y la plataforma Trello.

Se utilizará el framework Mocha para desarrollar dichos tests ya que está desarrollado en JavaScript y utiliza Node.js como motor [56]. Es decir, utiliza la misma tecnología utilizada para el desarrollo de la aplicación.

Este framework es ideal para el desarrollo de tests sencillos sobre la API que nuestra aplicación expone. Haciendo uso del gestor de paquetes NPM podremos instalar Mocha y el resto de los paquetes utilizados en el desarrollo de los tests.

Una vez desarrollados todos los tests se crea un script para poder ejecutarlos en batería (ver anexo D.1).

Para la ejecución de estos, se decide crear un contenedor basado en la imagen de Node.js de Bitnami, tal y como se hizo para el despliegue de Trello Updater. Este contenedor ejecuta durante su inicialización el script que lanza la batería de tests y redirige los resultados de los tests a la salida estándar donde son recogidos por Docker.

5.2.1 Tests de validación.

Cada uno de los elementos de un tablón en la plataforma Trello (columnas, tarjetas, etiquetas, etc.) se modela a través de su API con un identificador único. Dichos identificadores y el elemento al que representan son usados por Trello Updater a la hora de realizar llamadas a la API. Por tanto, es necesario comprobar que éstos son correctos y se desarrolla un test que recorre todos los identificadores disponibles en la aplicación para comprobar que son validos consultando el API de Trello.

Una de las tareas pendientes en este apartado es aumentar la cobertura de tests que comprueben que los ficheros tienen los permisos adecuados para garantizar que la aplicación sea segura.

5.2.2 Tests de funcionalidad e integración.

En el momento de desarrollar los tests de integración se plantean varias alternativas para simular los dos elementos con los que Trello Updater interactúa: las plataformas de soporte y la plataforma Trello.

- Plataformas de soporte:
    - Modelado (conocido en inglés como “mock-up”) de las mismas como un generador aleatorio de webhooks. Este generador simulará el comportamiento de las plataformas de soporte creando los mismos webhooks que estas utilizan para notificar a la plataforma Trello. Esta opción es descartada finalmente por su complejidad.
    - Crear manualmente diferentes webhooks cubriendo todas las combinaciones posibles de notificaciones que las plataformas de soporte pueden realizar. Para ello, se utiliza un generador de webhooks que, basándose en plantillas de los weebhoks generados por cada plataforma de soporte, nos devuelve el deseado en función a los argumentos utilizados al invocarlo. Posteriormente se utilizará un cliente HTTP que utilice los webhooks generados para hacer llamadas a la API de nuestra aplicación. Esta es la alternativa elegida finalmente.
- Plataforma Trello:
    - Modelado de la API de Trello utilizando alguna tecnología de API mock-up como node-mock-server [57]. Debido a su complejidad y a que esta opción requiere actualizar constantemente el modelo en función a los nuevos cambios introducidos en la API, esta opción es excluida.
    - Uso de un segundo tablón en Trello cuyo único propósito sea la realización de tests. Este tablón ha de ser una copia exacta del utilizado por los agentes de soporte en su día a día, es decir, ha de tener las mismas columnas y etiquetas disponibles. De esta forma, se puede interactuar con Trello para realizar tests sin afectar al tablón utilizado en producción. Por su sencillez, se opta por esta alternativa.

En la siguiente figura se ilustra el funcionamiento de los tests de integración:

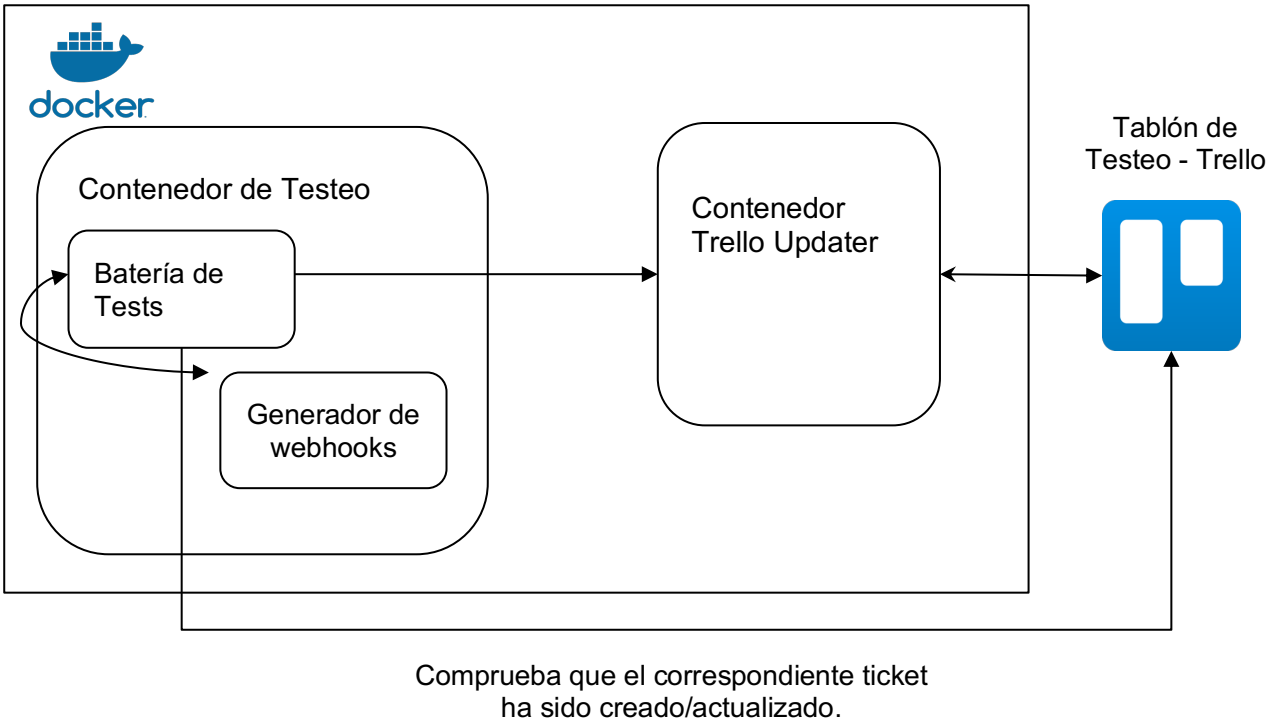


Figura 11 - Trello Updater: testeo.

Los tests de funcionalidad e integración creados simulan como las plataformas de soporte envían notificaciones a Trello Updater cuando un usuario solicita soporte o un agente de soporte atiende a dicho usuario. Posteriormente, mediante consultas al API de Trello se comprueba que Trello Updater crea o actualiza la correspondiente tarjeta en el tablón de Trello en concordancia a las notificaciones recibidas.

Se crea un test por cada tipo de ticket existente en Trello Updater y se simulan las posibles formas de interactuar entre un usuario que solicita soporte y un agente que lo atiende.

En el siguiente ejemplo se detalla como funciona uno de estos tests. Este ejemplo simula el escenario desde que un usuario crea una incidencia en la plataforma HelpDesk hasta que esta queda resuelta por un agente de soporte:

- Paso 1:
  a. Escenario real: un usuario crea un nuevo ticket en HelpDesk ya que tiene problemas para acceder por SSH a una instancia en la nube creada con un producto de Bitnami. HelpDesk envía un weebhook con información sobre el nuevo ticket a Trello Updater.
  b. Simulación: el test genera un webhook idéntico al que generaría HelpDesk y lo envía a Trello Updater.
  c. Comprobaciones: el test comprueba que Trello Updater devuelva el código de estado correcto, y posteriormente realiza una consulta a Trello para confirmar que se ha creado una tarjeta en el tablón de testeo con la información del ticket de HelpDesk en la columna “HelpDesk tickets” y contiene las etiquetas adecuadas.

- Paso 2:
  a. Escenario real: un agente de soporte es asignado para responder el ticket asociado a la tarjeta creada en Trello. Posteriormente el agente procede a responder al usuario con las instrucciones para acceder por SSH. Tras la respuesta, HelpDesk envía un weebhook actualizando el estado del ticket a Trello Updater.
  b. Simulación: el test genera un webhook idéntico al que generaría HelpDesk para actualizar el estado del ticket y lo envía a Trello Updater.
  c. Comprobaciones: el test comprueba que Trello Updater devuelva el código de estado correcto, y posteriormente realiza una consulta a Trello para confirmar que se ha actualizado la tarjeta en el tablón de testeo moviéndola a la columna “Pendiente de respuesta del usuario” y contiene las etiquetas adecuadas.
- Paso 3:
  a. Escenario real: el usuario logra acceder por SSH, y decide cerrar el ticket como “resuelto” agradeciendo al agente de soporte su ayuda. HelpDesk envía un weebhook actualizando el estado del ticket a Trello Updater.
  b. Simulación: el test genera un webhook idéntico al que generaría HelpDesk y lo envía a Trello Updater.
  c. Comprobaciones: el test comprueba que Trello Updater devuelva el código de estado correcto, y posteriormente realiza una consulta a Trello para confirmar que se ha actualizado la tarjeta en el tablón de testeo moviéndola a la columna “Tickets resueltos” y contiene las etiquetas adecuadas.

6 MIGRACIÓN A KUBERNETES

You've got to think about big things while you're doing small things, so that all the small things go in the right direction - Alvin Toffler

Tras la contenerización de la aplicación, el siguiente paso ha de ser migrar la misma a Kubernetes, y eliminar la instancia provisional creada en AWS.

En esta sección se cubre el diseño de los distintos objetos de Kubernetes que compondrán la solución de nuestra aplicación para esta plataforma, así cómo su empaquetado usando el lenguaje de plantilla Jsonnet [58]. Posteriormente, se usará la herramienta "kubecfg", desarrollada por Bitnami, para desplegar la solución empaquetada en un clúster en la nube pública de Google, GCP (de su acrónimo en inglés, Google Cloud Platform), haciendo uso del servicio GKE (de su acrónimo en inglés, Google Kubernetes Engine).

6.1 Introducción a los objetos de Kubernetes

Para un mayor entendimiento de los diferentes objetos de Kubernetes, su funcionamiento y especificaciones se recomienda consultar la documentación oficial de Kubernetes [59].

En este proyecto se hará simplemente una breve introducción a cada uno de los objetos utilizados para desplegar Trello Updater en K8s:

- Pod: Es la unidad mínima de computación en Kubernetes. Representa uno o más contenedores que comparten recursos de almacenamiento y un mismo espacio de puertos.
- Replicaset: Permite especificar el número de réplicas de un determinado tipo de pod que K8s ha de garantizar.
- Deployment: Abstracción de alto nivel que define una serie de operaciones de gestión sobre un replicaset.
- Horizontal Pod Autoscaler: Objeto que permite definir una serie de reglas para aumentar o disminuir el número de réplicas de un determinado tipo de pod automáticamente.

- Secret: Objeto que nos permite definir una serie de datos sensibles (credenciales, contraseñas, etc.) y ponerlos a disposición de las aplicaciones que los requieran.
- Service: Recurso que expone uno o más pods a través de una interfaz virtual. Puede ser utilizado para exponer recursos tanto dentro como fuera del clúster de K8s.
- Ingress: Facilita la configuración de un NGINX proxy [60] para acceder a un servicio a través de un dominio específico.

Todo recurso en Kubernetes es modelado como un objeto de API; desde entidades abstractas, como el "ReplicaSet", a aquellos que representan algún recurso real, como los "volumes". Esto algo crucial ya que podemos modelar cualquier solución a desplegar como una serie de objetos de API (descritos en formato JSON o YAML [61]). En el siguiente esqueleto YAML, se observa la estructura que sigue todo objeto K8s:

```yaml
apiVersion: v1  # Versión de API que soporta el objeto en cuestión
kind: Pod  # Tipo de objeto
metadata:  # Metadatos que permiten identificar/clasificar el objeto
  name: nginx
  namespace: default
  labels:
    key1: value1
  ...
spec:  # Especificaciones del objeto en cuestión
  containers:
  - image: myImage:latest
  ...
status:  # Estado del objeto una vez desplegado en Kubernetes
  hostIP: X.X.X.X
  phase: Running
```

6.2 Trello Updater en Kubernetes

Trello Updater es una aplicación "stateless", es decir, sin estado. Esta característica simplifica mucho el diseño del despliegue de esta en Kubernetes puesto que no es necesario tener en cuenta algunos de los problemas comunes como la persistencia de datos o la replicación del estado al escalarla horizontalmente.

Otra propiedad a tener en cuenta es el hecho de que proporciona un servicio que atiende peticiones HTTP y se desea exponer públicamente. En otras palabras, es necesario que el clúster permita acceder al API que expone Trello Updater desde el exterior, en un dominio concreto. Este requerimiento supone una serie de implicaciones:

- El clúster de Kubernetes ha de tener una IP pública y, al menos, un dominio DNS asociado a la misma. Trello Updater ha de usar un subdominio de este.
- El clúster ha de redirigir el tráfico dirigido al subdominio asociado a Trello Updater a las distintas réplicas de Trello Updater (balanceo de carga).
- Es necesario configurar un certificado TLS para garantizar conexiones seguras con la aplicación.

Para un mayor entendimiento de la solución diseñada, en la siguiente figura se muestran los diferentes elementos que componen el despliegue de Trello Updater y su interacción:

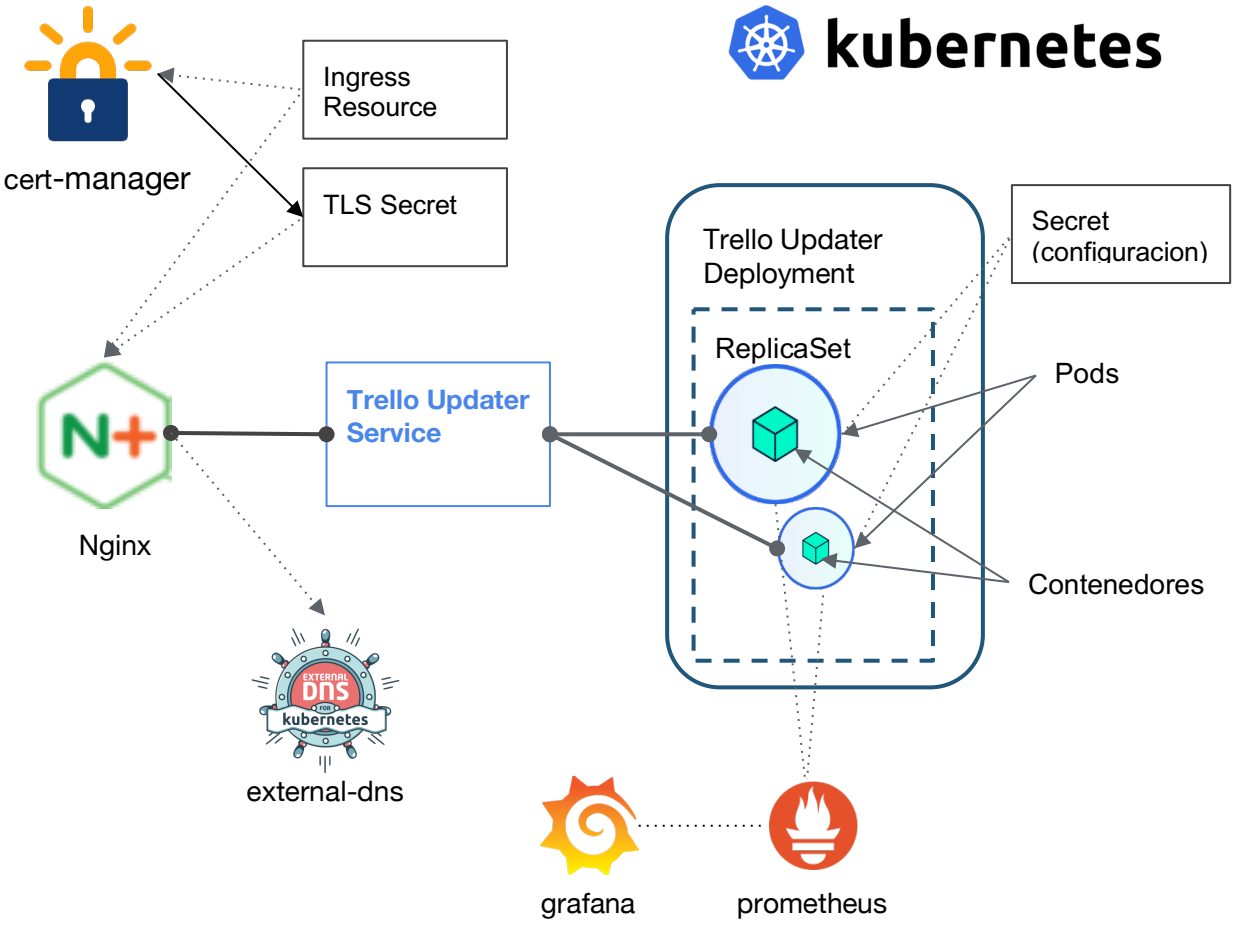


Figura 12 - Trello Updater en Kubernetes

Como se puede observar, la aplicación Trello Updater esta replicada horizontalmente en una serie de “pods” que son configurados a través de los ficheros de configuración disponibles en un “secret”. Estos ficheros se presentan codificados en el citado “secret”.

Para poder disponer de un histórico de los cambios realizados, así como para definir políticas de despliegue automático, se utiliza un “deployment”. Este a su vez, define el “replicaSet”, los “pods” y contenedores que lo componen (ver anexo E).

El escalado horizontal de la aplicación es automático haciendo uso de los mecanismos de autoescalado que Kubernetes ofrece desde la versión 1.11 [62]. Para ello se definen unas reglas a través de un “Horizontal Pod Autoscaler” para aumentar o disminuir el número de réplicas a desplegar en el clúster, basándose en el consumo de CPU y memoria RAM de estas (ver anexo E). De esta forma, podemos reducir el número de réplicas de Trello Updater al mínimo en periodos de poca actividad, y aumentarlo en aquellos periodos en los que se reciben muchas notificaciones. Todo ello, de forma automática y sin intervención humana.

La aplicación se expone a través de un servicio interno (es decir, sólo accesible desde dentro del propio clúster de Kubernetes). Sin embargo, al disponer nuestro clúster de un proxy NGINX de tipo “ingress” accesible desde el exterior, se define una regla en este que permite redirigir las peticiones web destinadas a un subdominio concreto a dicho servicio interno (ver anexo E).

Haciendo uso del controlador de Kubernetes denominado external-dns [63], se sincronizan aquellos subdominios DNS definidos dentro del clúster con el servidor DNS público de Google, Cloud DNS [64]. Esto permite acceder a los mismos de forma pública. Por ejemplo, si el clúster tiene asociado el domino “cluster.com” y se define una regla de “ingress” para el subdominio “trello.cluster.com”, external-dns creará un registro DNS para dicho subdominio y se podrá acceder al mismo públicamente.

Como respuesta a la necesidad de disponer certificados TLS, se utiliza un controlador de Kubernetes llamado cert-manager [65]. Este controlador monitoriza las distintas reglas de “ingress” definidas y genera/renueva de forma automática certificados para aquellas que así lo soliciten utilizando un cliente de Let’s Encrypt (ver anexo E). Nótese que esto supone una gran ayuda, pues sólo se requiere una solicitud y automáticamente nuestro NGINX proxy quedará correctamente configurado y será capaz de ofrecer contenidos de forma segura usando el protocolo HTTPS.

En la siguiente figura se muestra el flujo de una petición HTTPS al API de Trello:

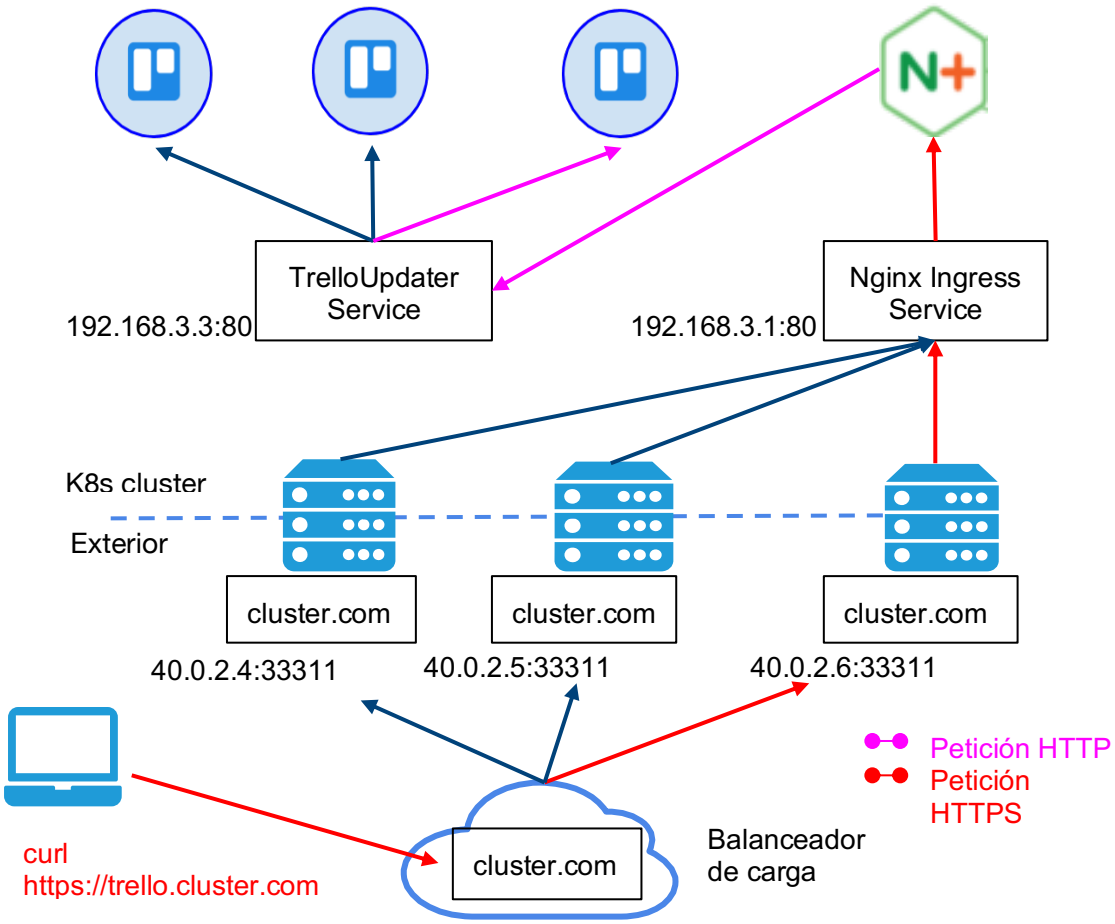


Figura 13 - Flujo de una petición HTTPS.

6.2.1 Sealed Secrets

Los secretos en Kubernetes no están encriptados, simplemente están codificados (utilizando el sistema de codificación base64 [66]). A pesar de ello, son utilizados para almacenar información sensible como contraseñas u otro tipo de credenciales. Esto se debe a que los secretos están diseñados para ser protegidos utilizando políticas de control de acceso por rol (del inglés Role Base Access Control) [67].

El hecho de no estar encriptados supone un problema de seguridad cuando se quiere almacenar la definición de todos los objetos de Kubernetes utilizados en un repositorio Git, como es el caso que nos ocupa. Para dar respuesta a este problema, se utiliza “Sealed Secrets”.

Esta solución se basa en tres elementos:

- La definición de un objeto customizado (del inglés Custom Resource Definition [68])

en Kubernetes.
- Un nuevo controlador de Kubernetes encargado de la gestión de este nuevo objeto en K8s.
- Un cliente de línea de comandos, denominado “kubeseal”, que se comunica con el controlador para la creación de objetos encriptados.

El funcionamiento es el siguiente:

- El controlador dispone de una clave privada que le permite desencriptar cualquier objeto de tipo “SealedSecret”.
- El cliente, kubeseal, obtiene la clave pública del controlador y haciendo uso de esta, convierte objetos de tipo “Secret” en tipo “SealedSecret”, los cuales están encriptados.
- Al desplegar la solución, sólo se indica la definición del objeto “SealedSecret” y será el controlador el encargado de crear el objeto tipo “Secret” correspondiente desde dentro del clúster.

Así, usando esta nueva herramienta, se puede guardar las definiciones encriptadas de los distintos objetos Sealed Secrets en nuestro repositorio.

6.3 Observabilidad en Kubernetes

Un aspecto crucial de cualquier servicio en producción es la observabilidad de dicho servicio. Este es un concepto muy amplio y son diversas las herramientas y tecnologías que dan solución a los problemas que plantea. Sin embargo, todos tienen un objetivo en común: obtener información relevante sobre el estado de un servicio y su interacción con otros servicios del sistema [69] [70].

Dos de los elementos principales de la observabilidad son:

- Registros o “logging”.
- Monitorización.

Los clústeres de Kubernetes utilizados en Bitnami utilizan las siguientes tecnologías para dar solución a dichos elementos:

- EFK (ElasticSearch [71] + Fluentd [72] + Kibana [73]), para “logging”.
- Prometheus [74] y Grafana [75], para monitorización.

A continuación, se analiza cada uno y cómo como se configura la aplicación Trello Updater para que ésta pueda ser integrada con los mismos.

6.3.1 Registros o “logging”

Para obtener los registros o “logs” de las aplicaciones que están siendo ejecutadas en el clúster, estas han de redirigir los mismos a la salida estándar o a la salida de errores (conocidas en inglés cómo “stdout” o “stderr”). Trello Updater fue diseñada originalmente de tal forma, por lo que no requiere ser modificada.

Kubernetes almacena los registros de cada contenedor. Para acceder a ellos, basta con

usar el cliente de línea de comandos kubectl [76] y solicitar los registros de un contenedor específico como se muestra en el siguiente comando:

```
$ kubectl logs [-f] POD [-c CONTAINER] [options]
```

Este es un método bastante pobre para obtener los registros ya que no nos permite analizar y/o filtrar los mismos. Por ello, es necesario buscar una solución más compleja, como EFK.

Actualmente todos los clústeres de Kubernetes utilizados en Bitnami constan de esta solución ya configurada y lista para su uso en producción, siendo el equipo de SRE de Bitnami el encargado de dicha tarea.

EFK se compone de tres elementos:

- Elasticsearch. Base de datos que permite realizar búsquedas muy eficientes gracias mediante técnicas de indexado de los mismos.
- Fluentd. El cual actúa como colector de datos. Es decir, se encarga de obtener los registros de cada uno de los nodos de Kubernetes y encolarlos para su posterior inyección en la base de datos Elasticsearch.
- Kibana. Se trata de panel de control o "dashboard" que permite realizar búsquedas en Elasticsearch a través de una interfaz de usuario web. También posee herramientas para la elaboración de gráficos basados en los registros almacenados en Elasticsearch.

En la siguiente figura se muestra cómo interactúan los distintos elementos de la solución entre sí:

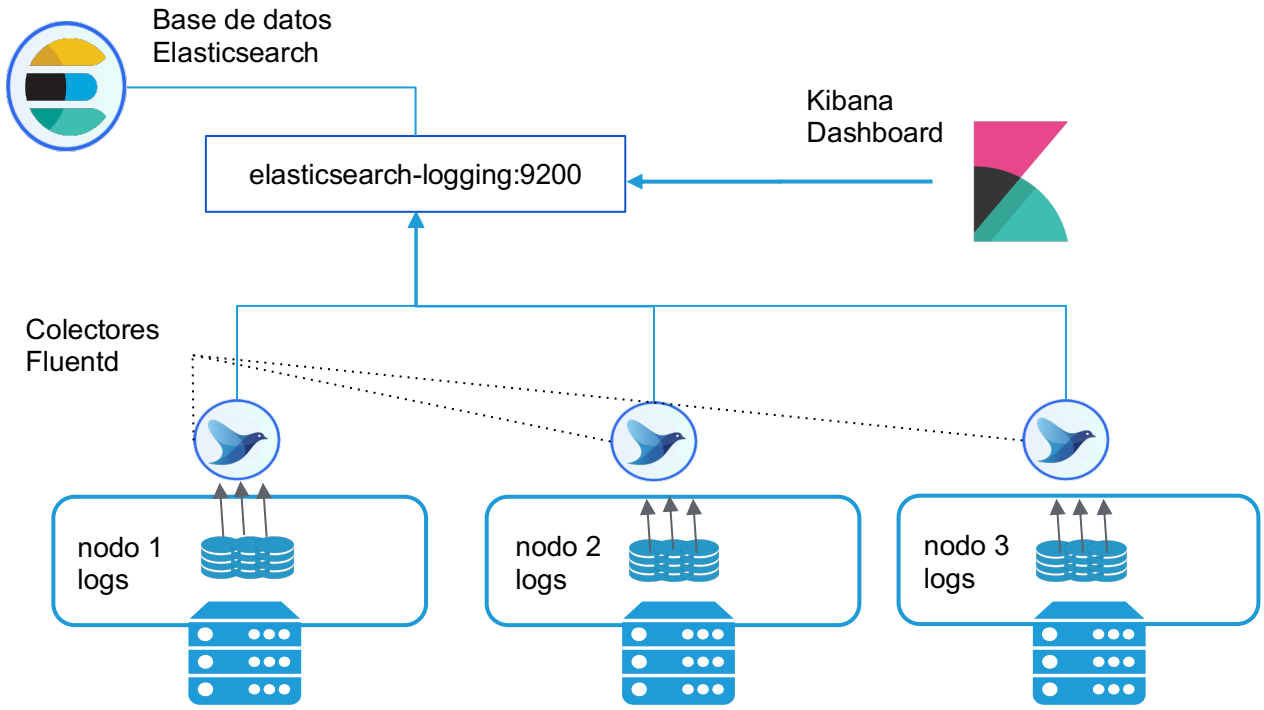


Figura 14 - EFK.

6.3.2 Monitorización

La obtención de métricas relevantes resulta crucial para analizar el comportamiento de un servicio en producción.

Un sistema que permita obtener/agregar métricas de las aplicaciones del clúster dota al administrador de estas una serie de ventajas:

- Implementación de alertas. Por ejemplo, se pueden implementar una alerta cuando una determinada métrica supera cierto valor.
- Análisis de tendencias.
- Depuración de errores.
- Estimación de recursos.

Los clústeres utilizados en Bitnami utilizan Prometheus y Grafana como sistema de monitorización:

- Prometheus se encarga de colectar métricas periódicamente, así como de su procesamiento y almacenamiento.
- Grafana es una herramienta que permite crear gráficos basados en múltiples fuentes de datos, siendo Prometheus una de ellas.

Hay dos tipos de métricas en todo clúster de Kubernetes:

- Métricas expuestas por el propio clúster: como el consumo de CPU o el consumo de recursos de red.
- Métricas propias de las aplicaciones. Estas son diferentes en cada aplicación y es responsabilidad del desarrollador de estas decidir cuáles son relevantes y exponerlas públicamente. Por ejemplo, en un servidor web, podría exponerse una métrica con el número de peticiones web recibidas.

La monitorización de Trello Updater es utilizada en este proyecto para el estudio de la evolución del consumo de CPU y memoria de la aplicación Trello Updater. Basándonos en dicho estudio se ajustan los recursos que se solicitan al clúster cuando se despliegan los contenedores asociados a la misma.

En la siguiente imagen se muestran gráficos de Grafana sobre el consumo de CPU y memoria RAM en Trello Updater:

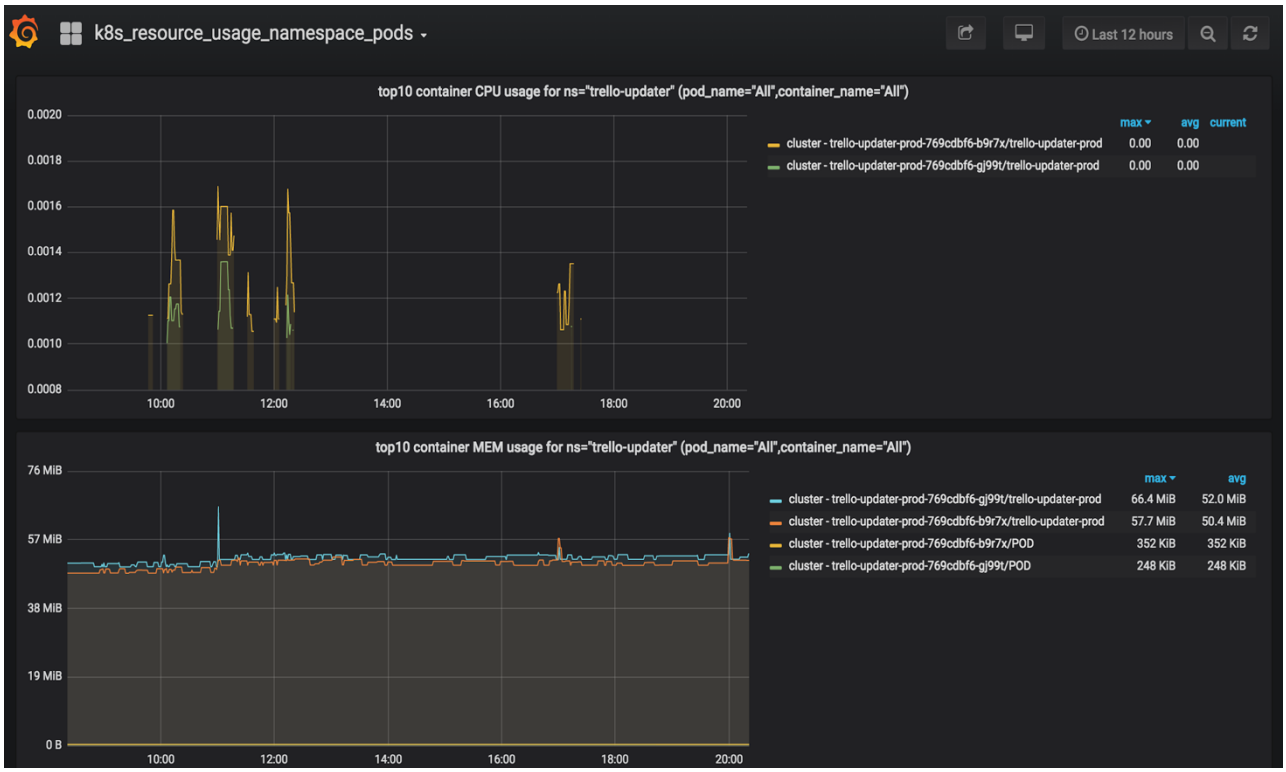


Figura 15 - Monitorización: consumo de memoria y CPU de Trello Updater.

6.4 Empaquetado y despliegue de la solución

Previamente, en el apartado 5.1, se mencionaban los distintos objetos de Kubernetes necesarios para desplegar Trello Updater en un clúster. Todos estos objetos comparten algo en común, formar parte de una misma solución: Trello Updater. Dicho de otro modo, pertenecen al mismo “paquete”.

Este escenario no es exclusivo de Trello Updater, sino de toda solución que se despliega en Kubernetes. Son varias las soluciones que se han propuesto en el ecosistema Kubernetes para dar respuesta al concepto de “empaquetado”, siendo Helm la solución más extendida.

Helm resuelve el problema del empaquetado presentando un sistema de plantillas que permiten agrupar las diferentes definiciones de los objetos de K8s bajo un mismo paquete, denominado “chart” en su terminología. A la hora de desplegar un chart, se modifican una serie de valores en las plantillas, permitiendo reutilizar el mismo para realizar tantos despliegues como se deseen sin que haya colisión en el espacio de nombres.

A pesar de su popularidad y de la familiaridad que se tiene con esta tecnología, al ser Bitnami el principal contribuidor de charts, se descarta por motivos de seguridad. Esta solución requería en su versión 2 la instalación de un controlador en el clúster denominado “Tiller” que ha de tener privilegios de administrador en el clúster, suponiendo esto una

importante brecha de seguridad (nótese que esta vulnerabilidad es conocida por la comunidad y será corregida en próximas versiones de Helm)

Cómo alternativa a Helm, se opta por utilizar otro lenguaje de plantilla: Jsonnet.

6.4.1 Jsonnet

Jsonnet es un lenguaje muy flexible desarrollado por Google que permite realizar operaciones (suma, resta, sustitución, concatenación, etc.) con datos en formato JSON. La idea de empaquetar definiciones de Kubernetes con Jsonnet consiste en utilizar distintos ficheros con pequeñas modificaciones que hacen uso de librerías genéricas que nos definen cada uno de los objetos de Kubernetes.

Para empaquetar aplicaciones en Jsonnet, se utiliza una librería de código abierto que ha sido desarrollado por compañeros de Bitnami y se encuentra en disponible en Github, kube-libsonnet [77]. Esta librería define la mayoría de los objetos de Kubernetes disponibles en la actualidad.

Una de las principales ventajas de utilizar Jsonnet es lo simple que resulta modificar cualquier valor de un objeto ya definido. Esta característica se aprovecha para, partiendo de la misma definición jsonnet, crear dos soluciones diferentes: una para desplegar Trello Updater para su testeo y otra para desplegarlo para su puesta en producción. Para mayor detalle sobre las definiciones Jsonnet utilizadas, ver anexo F.

6.4.2 Kubecfg

Para desplegar la solución empaquetada en el clúster de Kubernetes se usará kubecfg, una herramienta desarrollado por compañeros de Bitnami que está diseñada para desplegar soluciones en K8s a partir de plantillas Jsonnet.

Esta herramienta sólo necesita kubectl esté correctamente configurado para administrar el clúster donde se quiere desplegar una solución. Para desplegar la solución basta con ejecutar el siguiente comando donde ‘***solution.jsonnet’*** ha de contener la solución empaquetada en formato Jsonnet:

```
$ kubecfg update solution.jsonnet
```

7 CI/CD: INTEGRACIÓN Y DESPLIEGUE CONTINUO

Do the best you can until you know better. Then when you know better, do better - Maya Angelou

Uno de los requisitos designados a este proyecto era la implantación de un sistema de integración y despliegue continuo que permita reducir los tiempos de cada ciclo de desarrollo.

Este sistema persigue la automatización de cuántos procesos sean posibles en el ciclo de desarrollo de forma que cualquier cambio introducido en la aplicación esté listo para su despliegue en producción en el menor tiempo posible.

En este capítulo se expone la implementación del citado sistema para la aplicación web Trello Updater, así como las herramientas utilizadas en este.

7.1 Ciclo de Desarrollo

El ciclo de desarrollo del software suele definirse como una secuencia estructurada de etapas que componen el desarrollo de un producto software determinado [40]. Dichas etapas suelen incluir el estudio de los requisitos del sistema, así como el análisis de este y su diseño. Sin embargo, en el ámbito de la integración continua, no suele envolver a estas etapas iniciales pues se busca la automatización de los procesos implicados en la introducción de nuevos cambios en un proyecto ya creado.

Volviendo a la definición más académica del ciclo de desarrollo del software, son las etapas del despliegue de cambios, pruebas e integración con otros componentes/servicios las cuales serán automatizadas.

En el caso de este proyecto, dichas etapas se componen de los siguientes procesos:

- Despliegue de cambios. En esta solución son dos los tipos de elementos sujetos a modificación: las definiciones de los objetos de Kubernetes a desplegar o la imagen

de Docker que contiene la aplicación contenerizada.
    - En el caso de cambios que afecten a las definiciones de Kubernetes, se utiliza una funcionalidad de la herramienta kubecfg que permite detectar cambios entre una solución ya desplegada en un clúster y definiciones locales. En el caso de existir diferencias, permite modificar aquellos aspectos de la solución existente que hayan sido modificados.
    - En el caso de cambios en la imagen de Docker, ya sea por cambios en la aplicación en sí o en la propia definición de la imagen, se sigue un sistema de versionado. Es decir, cada vez que se introduce un cambio es posible utilizar un script para construir una nueva imagen que es etiquetada con una versión diferente. De esta forma basta con editar las definiciones de Kubernetes garantizando que estas usan la versión deseada de la imagen de Docker.
- Tests o pruebas. Tal cómo se expone en el apartado 5.2, se hace uso de un contenedor que contiene tanto los tests a realizar como las herramientas para ejecutarlos para realizar pruebas de validación e integración. En Kubernetes existe un tipo de objeto denominado "job", que permite ejecutar un pod determinado y destruirlo una vez la acción que este realiza ha concluido con éxito. Este objeto es perfecto para ejecutar los tests, pues permite lanzar la batería de tests contra la aplicación desplegada en el clúster, reportar el resultado y eliminar los recursos utilizados para los mismos.
- Integración con otros componentes/servicios. En el apartado 5.2.1 ya se introduce el uso de un tablón de testeo, el cuál es una réplica del utilizado en producción. Bastan con garantizar acceso desde la aplicación desplegada en el clúster a este para poder realizar las pruebas de integración pertinentes.

Un aspecto que todo sistema CI/CD debe garantizar es evitar interferir en el normal desarrollo del servicio de producción. Para ello, se hacen uso de dos clústeres diferentes, un clúster de preproducción y otro de producción.

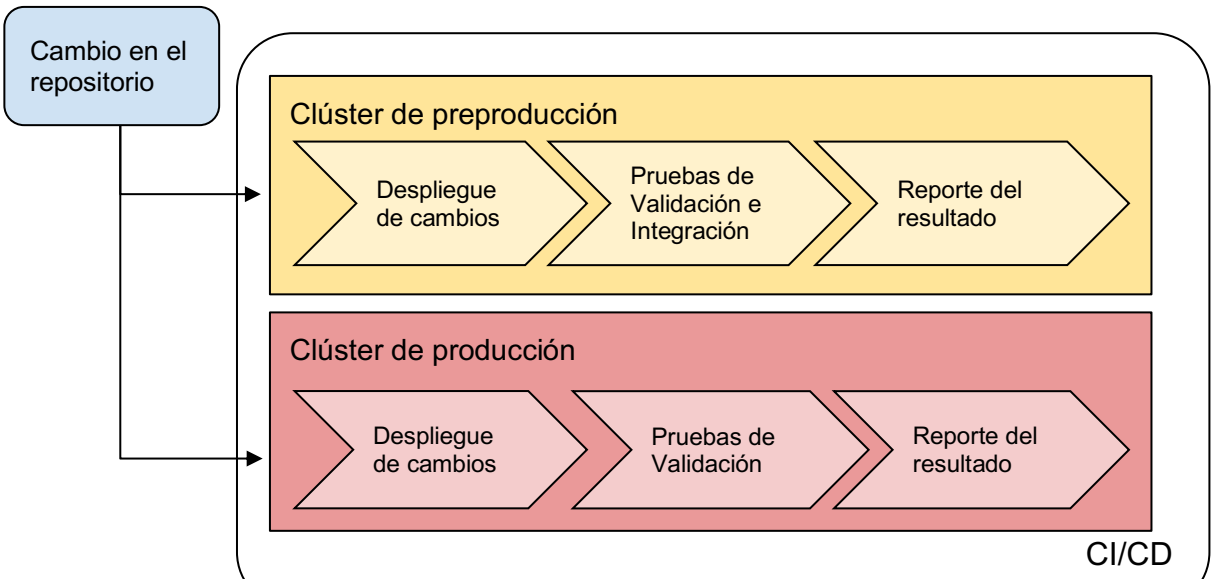


Figura 16 - Sistema CI/CD: diagrama de flujo.

Cómo se observa en la anterior figura, es el repositorio el encargado de garantizar un control de versiones. Es decir, en caso de que un cambio afecte al correcto funcionamiento del servicio y este escape a las tests disponibles, basta con revertir dicho cambio en el repositorio y el sistema es capaz de reproducir y desplegar la solución en su estado anterior. Esto resulta crucial si se tiene en cuenta que, una vez automatizado, se requiere de tan sólo un minuto aproximadamente en completar el ciclo.

7.2 Jenkins

Como se introdujo en el estudio de mercado (ver apartado 2.2.5), se usa la herramienta de automatización de código abierto Jenkins para implementar el sistema de integración y despliegue continuo. Esta herramienta consiste en un servidor que permite definir y ejecutar "trabajos" (del inglés "jobs") que automatizan una serie de operaciones.

El sistema CI/CD se implementa pues a través de un trabajo de Kubernetes que en función a dos parámetros de entrada gestiona la integración continua, en el clúster de preproducción; o el despliegue continuo, en el clúster de producción. Estos parámetros de entrada son:

- Confirmación (del inglés "commit") del repositorio.
- Clúster de destino.

El citado trabajo se crea en un servidor de Jenkins ya disponible y cuya gestión es responsabilidad del equipo de SRE de Bitnami, con el que se colabora estrechamente durante la creación de este. Este equipo se encarga, por ejemplo, de proveer a Jenkins de las credenciales necesarias para realizar cambios en los clústeres de Kubernetes utilizados. En el anexo D.2 se detalla la configuración del trabajo de Jenkins creado.

Jenkins nos permite acceder al resultado de los trabajos realizados a través de su interfaz gráfica. Para más detalle, ver anexo D.3.

8 VALIDACIÓN

The purpose of validating yourself is to enable you validate others - Meiz Ezra

En esta sección se procede a validar si los objetivos establecidos al inicio de este proyecto han sido logrados con la ejecución de este.

Recordemos que este proyecto perseguía mejorar la productividad y los procedimientos utilizados por el equipo de soporte de Bitnami. Por otro lado, la solución a implementar estaba sujeta a una serie de requisitos para su puesta en producción, y era muy importante que fuese una solución sencilla con costes de mantenimiento muy bajos.

8.1 Robustez de la solución

La mejor muestra de cuan robusta es la solución desarrollada en este proyecto proviene del tiempo que lleva la misma puesta en producción: desde el 28 de febrero de 2018 hasta la actualidad.

Durante este tiempo, las incidencias registradas han sido mínimas y la mayoría de carácter leve: duplicado de tarjetas y problemas de rendimiento puntuales debido a sobrecarga de tarjetas en Trello. Sin embargo, apenas han ocurrido incidencias que hayan afectado a su operatividad. El servicio ha operado de forma ininterrumpida con una disponibilidad superior al 99%, ya que de las más de 17.500 horas de vida de la aplicación, en tan sólo 20 de ellas se vio afectada su disponibilidad. La siguiente tabla recoge información sobre las incidencias que han afectado a Trello Updater:

Carácter	Ejemplos	Incidencias registradas
Grave – interrupción del servicio o pérdida de datos.	• Trello Updater es inaccesible. • Trello Updater no puede contactar con el API de Trello.	Nº total: 2 • Problema con el certificado TLS usado para las conexiones HTTPS. Aprox. 18 horas desde que ocurre hasta que se detecta/corrige.

	• Trello Updater no puede creare nuevas tarjetas en Trello.	• Corrupción de las credenciales al realizar una actualización. El servicio estuvo 2 horas inaccesible. Se creo un test para evitar que esto pasará de nuevo.
Medio – Interrupción puntual del servicio sin pérdida de datos	• Los agentes de Trello no pueden acceder a Trello, pero este continúa almacenando la información procedente de Trello Updater.	Nº total: 2 • En un par de ocasiones la UI de Trello ha permanecido inaccesible por periodos de varias horas. Sin embargo, su API siempre ha estado disponible por lo que no se han perdido datos.
Leve – Problemas de rendimiento o datos incorrectos	• La navegación a través de la UI de Trello es lenta. • Una tarjeta ha sido creada por duplicado. • Una tarjeta es movida a la columna incorrecta.	Nº total: >50 (el número es aproximado ya que las incidencias leves no quedan registradas) • La más común reportada es la navegación lenta en la UI de Trello. Suele producirse cuando el número de tarjetas en el tablón es elevado (>200). Pare evitarlo, se creó un cronjob que archiva las tarjetas resueltas diariamente. • El duplicado de tarjetas suele producirse cuando el tiempo entre la creación de una tarjeta y su primera actualización es muy corto. Con diferentes mejoras se ha conseguido reducir el intervalo entre eventos que provocan la incidencia de 5 minutos a unos 10 segundos, reduciendo así sustancialmente el número de estas.

Tabla 6 - Incidencias registradas en Trello Updater.

En este tiempo se han añadido pequeñas mejoras, como por ejemplo añadir nuevas etiquetas para el catalogado de las tarjetas, incluir nuevas columnas para nuevos tipos de soporte, o adaptar la aplicación a los nuevos formatos de los webhooks. Aparte de estas mejoras, se ha actualizado la imagen de Trello Updater para incluir actualizaciones en los paquetes de sistema, la versión de Node.js, etc. En total, han sido más de 60 las veces en las que se han desplegado cambios en el clúster de producción, sin que el despliegue de estos interrumpiera el servicio.

Además, gracias al sistema del integración y despliegue continuo, los tiempos para implementar una nueva funcionalidad, testearla y desplegarla en producción son realmente bajos. En ninguna ocasión, ya sea para resolver una incidencia o implementar una nueva funcionalidad, se ha querido más de una jornada laboral, 8 horas, para implementar los cambios y desplegarlos en producción.

8.2 Requisitos Técnicos

En el apartado 1.4, se introducían una serie de requisitos técnicos a implementar que quedan listados en la tabla 1. La siguiente tabla refleja el cumplimiento de estos:

Requisito	Cumplimiento	Validación
Operatividad 24/7	✓	Como se menciona en el anterior apartado, la solución tiene una disponibilidad superior al 99%.
Escalabilidad Horizontal	✓	La solución es 100% escalable, y se han implementado mecanismos que automatizan el proceso de escalado, ve apartado 6.2.
Kubernetes como plataforma	✓	Trello Updater corre en Kubernetes, como se detalla en el apartado 6.
Contenerización de la aplicación desarrollada	✓	Trello Updater es una aplicación contenerizada, ver apartado 5.
Uso de buenas prácticas en Kubernetes	✓	La solución sigue las guías de buenas prácticas para el diseño y despliegue de aplicaciones sobre K8s. Además, hace uso de librerías mantenidas por el equipo de SRE como se detalla en el apartado 6.4.1
Implementación de un sistema CI/CD	✓	Se implementa un sistema de integración y despliegue continuo usando Jenkins tal y como se detalla en el apartado 7.
Monitorización	✓	Haciendo uso de Grafana y Prometheus, se implementa un sistema que permite monitorizar la aplicación, ver apartado 6.3.2
Seguridad	✓	La aplicación garantiza conexiones seguras a través de HTTPS, y es obligatorio autenticarse vía TOKEN para poder acceder a su API. Por otro lado, no se han detectado comportamientos sospechosos a través de su monitorización. Su alta disponibilidad es, de nuevo, un buen indicador de su seguridad.

Tabla 7 - Validación de requisitos técnicos.

8.3 Reducción de la carga de trabajo

Gracias a los automatismos introducidos por Trello Updater y la posibilidad de ofrecer a los agentes de soporte una única interfaz a través de la que gestionar las diversas plataformas de soporte, la carga de trabajo en soporte se ha reducido significativamente.

En los meses previos a la implementación de Trello Updater, de forma rotativa (cada dos semanas) se dedicaba el trabajo de dos ingenieros a tareas de soporte. Esto suponía un coste de 80 horas semanales o, aproximadamente, 350 horas mensuales dedicadas.

Tras la implementación de Trello Updater, se cambia el modelo de trabajo. Se introduce una nueva etiqueta que permite catalogar cada caso de soporte en 3 niveles de complejidad: L1, L2 y L3. A mayor nivel, mayor la complejidad. Por otro lado, se pasa de 2 a 4 los ingenieros dedicados a tareas de soporte de forma rotativa. El reparto de las tareas de soporte pasa a ser el siguiente:

- 3 ingenieros responden los casos de soporte de nivel L1 y L2, además de encargarse de la clasificación de estos según su complejidad.
- El ingeniero restante se encarga de dar respuesta a los casos de nivel L3.

Con este nuevo modelo, cada ingeniero dedica 3 horas al día a tareas de soporte en lugar de hacerlo a jornada completa. Es decir, 3 horas x 4 ingenieros x 5 días laborales = 60 horas semanales. Aproximadamente, 260 horas mensuales.

Cómo se puede observar el ahorro de tiempo es más que significativo, en torno al 25%. Además, se reduce significativamente el nivel de frustración del ingeniero, puesto que las tareas de soporte suelen resultar tediosas.

En junio de 2019, se reestructura de nuevo el reparto de tareas debido a la creciente demanda de soporte en GitHub con el siguiente modelo:

- 2 ingenieros responden los casos de soporte de nivel L1 y L2 procedentes de StackoverFlow, Community y HelpDesk, además de encargarse de la clasificación de estos según su complejidad.
- 2 ingenieros responden los casos soporte de nivel L3 procedentes de StackoverFlow, Community y HelpDesk.
- 4 ingenieros responden todo el soporte (nivel L1, L2 y L3) procedente de GitHub.

Con este nuevo modelo, se reduce el tiempo que cada ingeniero dedica a tareas de soporte a 2 horas al día. Es decir, 2 horas x 8 ingenieros x 5 días laborales = 80 horas semanales. Aproximadamente, 350 horas mensuales. Este dato puede parecer una regresión, pero como veremos en el siguiente gráfico no lo es ya que la cantidad de soporte solicitado no para de crecer:

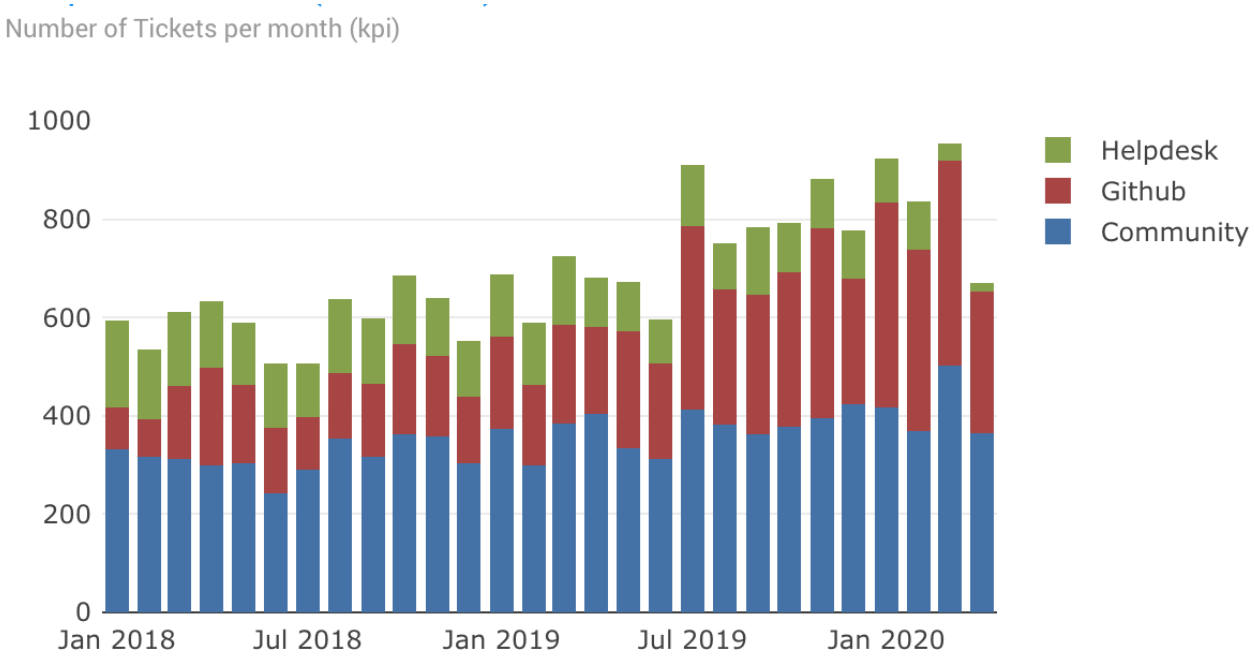


Figura 17 - Nº de solicitudes de soporte por mes.

Nota: instantánea tomada el 19 de abril del 2020. El soporte procedente de Stackoverflow no es representado ya que es despreciable en proporción.

El récord mensual de solicitudes de soporte hasta ese momento corresponde a marzo de 2020 con 956 solicitudes (36 en HelpDesk, 417 en Github, y 501 en Community). Todo esto dedicando 350 horas mensuales a soporte, lo que supone unos 22 minutos dedicados a cada solicitud de soporte. Si comparamos este valor con las recibidas en enero de 2018, 594 (177 en HelpDesk, 85 en Github, y 332 en Community), lo que supone unos 35 minutos dedicados a cada solicitud de soporte, vemos que se ha reducido casi un 40% el tiempo dedicado por cada solicitud.

8.4 Retroalimentación Soporte-Documentación

En la introducción se abordaba este aspecto como un elemento clave para mejorar la experiencia de uso.

Desde que Trello Updater se puso en producción se habilito un campo que permitía al agente de soporte marcar un caso de soporte para su posterior revisión con el equipo de documentación. Básicamente este campo añade una etiqueta a la tarjeta asociada en Trello que indica que ha de ser revisado por los documentalistas.

Basados en aquellas tarjetas que poseen dicha etiqueta, se vienen haciendo reuniones bisemanales en las que se analizan aquellos casos en los que los agentes de soporte consideran que la documentación ha de ser mejorada o extendida. De esta reunión se sacan una serie de tareas a implementar por el equipo de documentación en la siguiente interacción, cerrando así el bucle de retroalimentación.

Se han realizado considerables avances en la documentación como refleja las mejoras en indicadores como el NPS (de su acrónimo en inglés, Net Promoter Score), que permiten medir la satisfacción de los usuarios [78]. Las siguientes figuras muestran la evolución del NPS entre octubre de 2018 y octubre de 2019:

- Octubre 2018:

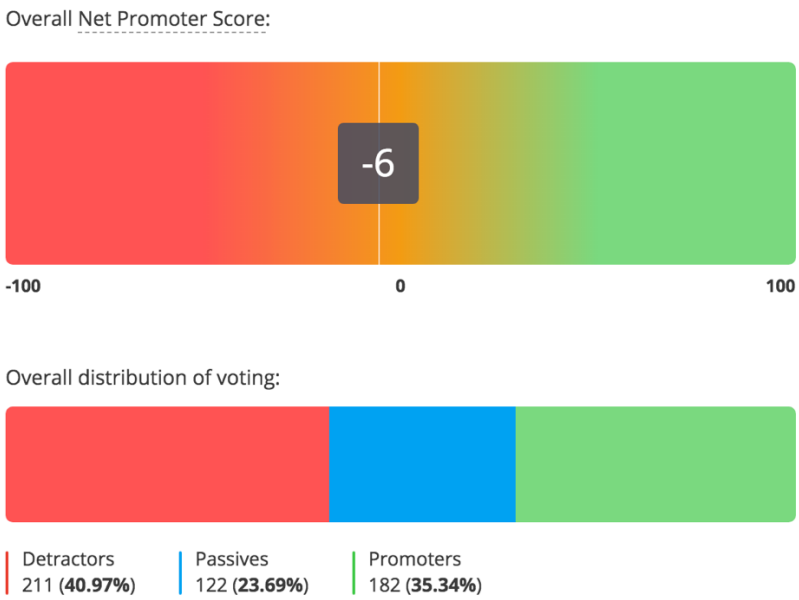


Figura 18 - NPS de la documentación: octubre 2018.

- Octubre 2019:

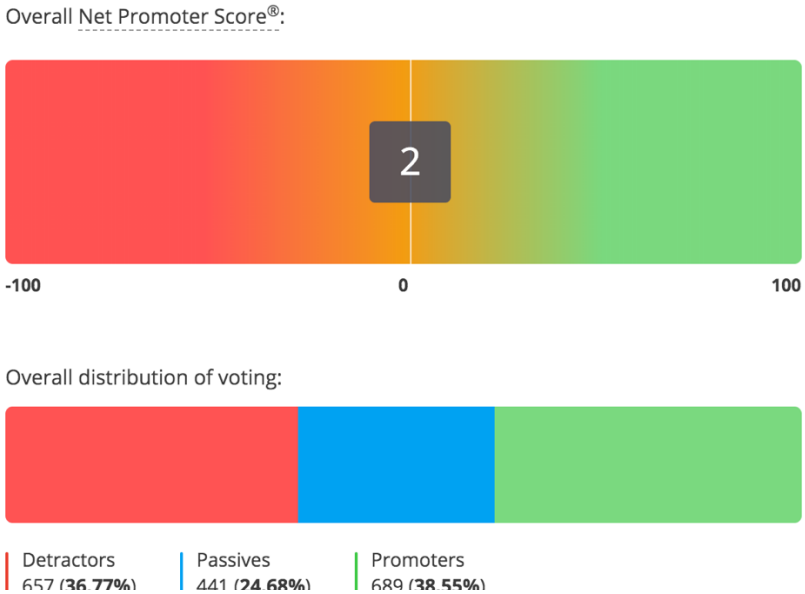


Figura 19 – NPS de la documentación: octubre 2019.

También se ha observado cómo algunas de las preguntas más comunes en soporte han disminuido considerablemente en frecuencia gracias a la restructuración de la documentación y a la creación de secciones de resolución de problemas (del inglés “troubleshooting”).

8.5 Métricas de Soporte y SLAs

Paralelo a la migración de Trello Updater a Kubernetes, se comenzó a trabajar en un “dashboard” que presente métricas basándose en la información almacenada en Trello. Es decir, un sistema que analice las tarjetas y las etiquetas asociadas a las mismas para así sacar información relevante sobre el soporte proporcionado.

Este dashboard se implementa utilizando la herramienta de código abierto Redash [79]. Esta herramienta ofrece una interfaz que permite realizar consultas avanzadas a diferentes bases de datos y generar una serie de gráficos en función a las mismas.

Para extraer información de Trello Updater, se utilizan unos “Cron Jobs” en Kubernetes. Estos no son más que unos trabajos que se ejecutan de forma periódica y son eliminados tras finalizar. Estos trabajos obtienen información consultando el API de Trello y la almacenan en una base de datos usando el servicio de bases de datos relaciones RDS de AWS [80].

Finalmente, se configura Redash para realizar consultas sobre la base de datos de RDS y se crean una serie de gráficos para presentar resultados de cara a dirección sobre el trabajo realizado en soporte. La siguiente figura muestra algunos de los gráficos creados:

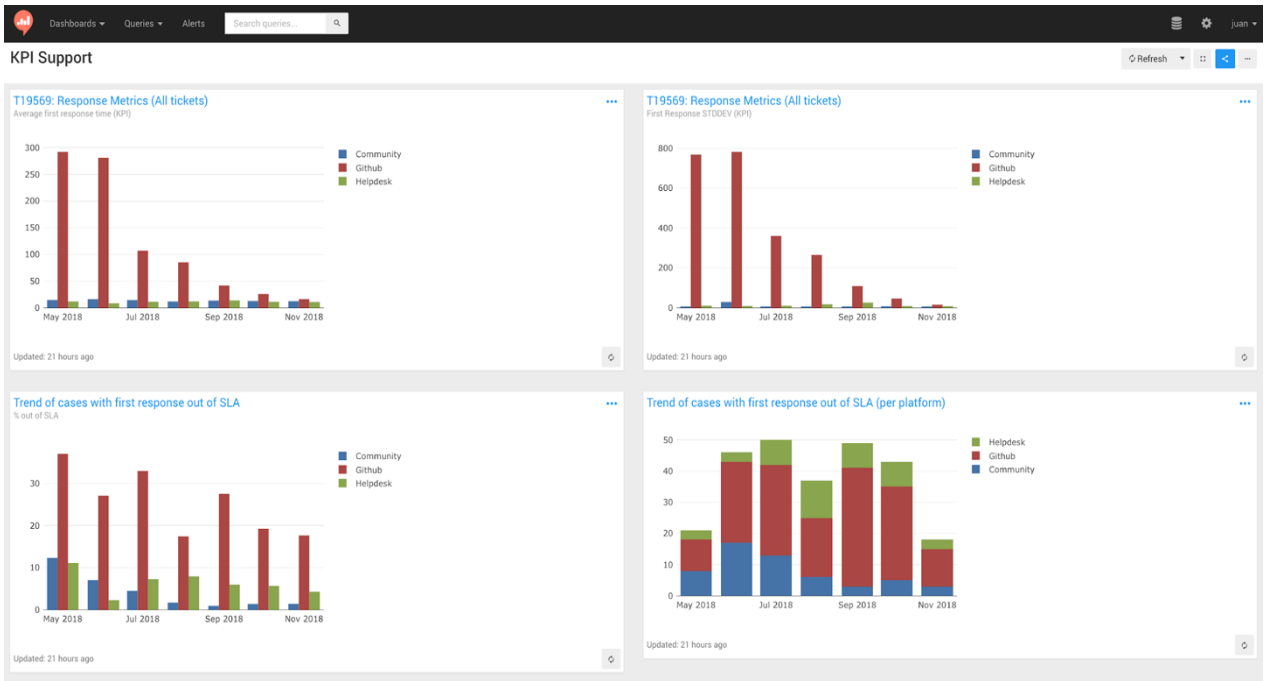


Figura 20 - Dashboard de soporte.

Basado en la información obtenida en ReDash, se decide desde dirección implementar una serie de acuerdos de nivel de servicio, conocidos como SLA (del inglés, Service Level Agreement), para garantizar unos estándares de calidad en cuánto al soporte que se proporciona a los usuarios [81].

Estos acuerdos de nivel de servicio se focalizan principalmente en los tiempos desde que un usuario acude a una plataforma de soporte hasta que el usuario recibe una respuesta por parte de un agente. Además, dichos tiempos son diferentes según qué plataforma, puesto que se prioriza en función del tipo de usuario. Por ejemplo, aquellos casos de soporte procedentes de usuarios de pago que hacen uso de la plataforma HelpDesk, tienen un SLA más restrictivo.

Estos SLAs permitirán además a la empresa alcanzar acuerdos concretos en cuánto al soporte proporcionado a la hora de negociar nuevos contratos, o la hora de renovar los existentes, con los proveedores de cloud en los que se ofertan las soluciones de Bitnami.

En los siguientes gráficos observamos como han mejorado algunas de estos SLAs:

- Tiempo medio en cerrar una solicitud de soporte:

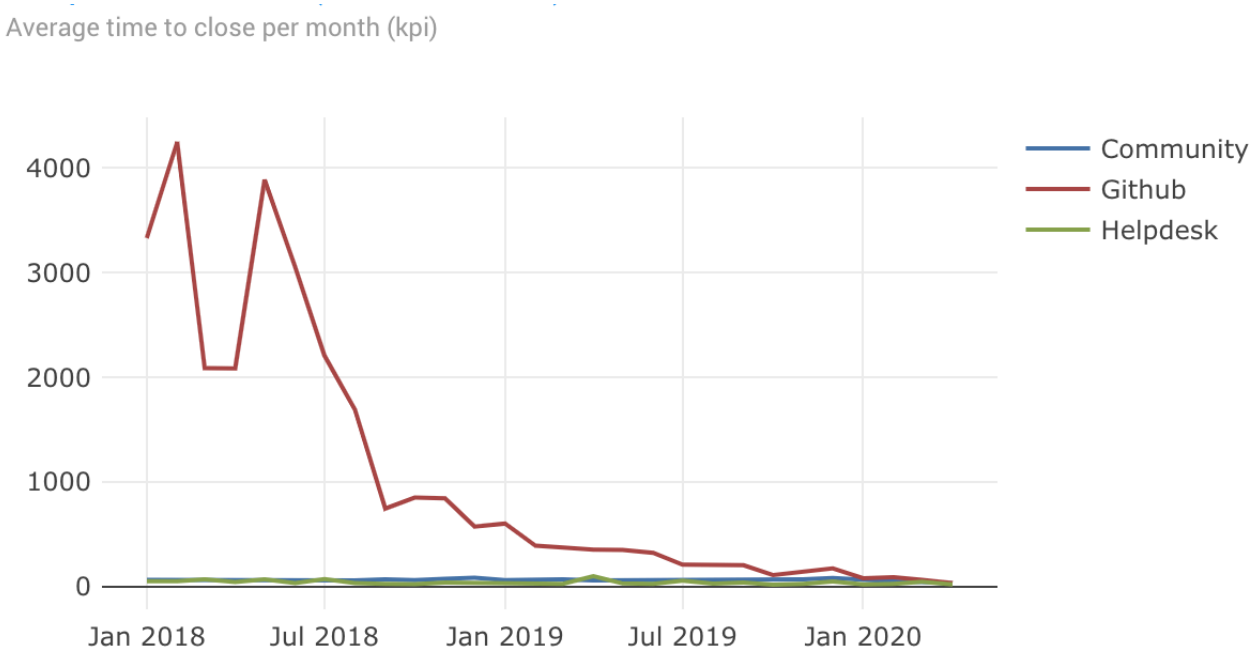


Figura 21 - Tiempo medio para cerrar una solicitud de soporte pore mes.

- Tiempo medio en dar una primera respuesta a una solicitud de soporte:

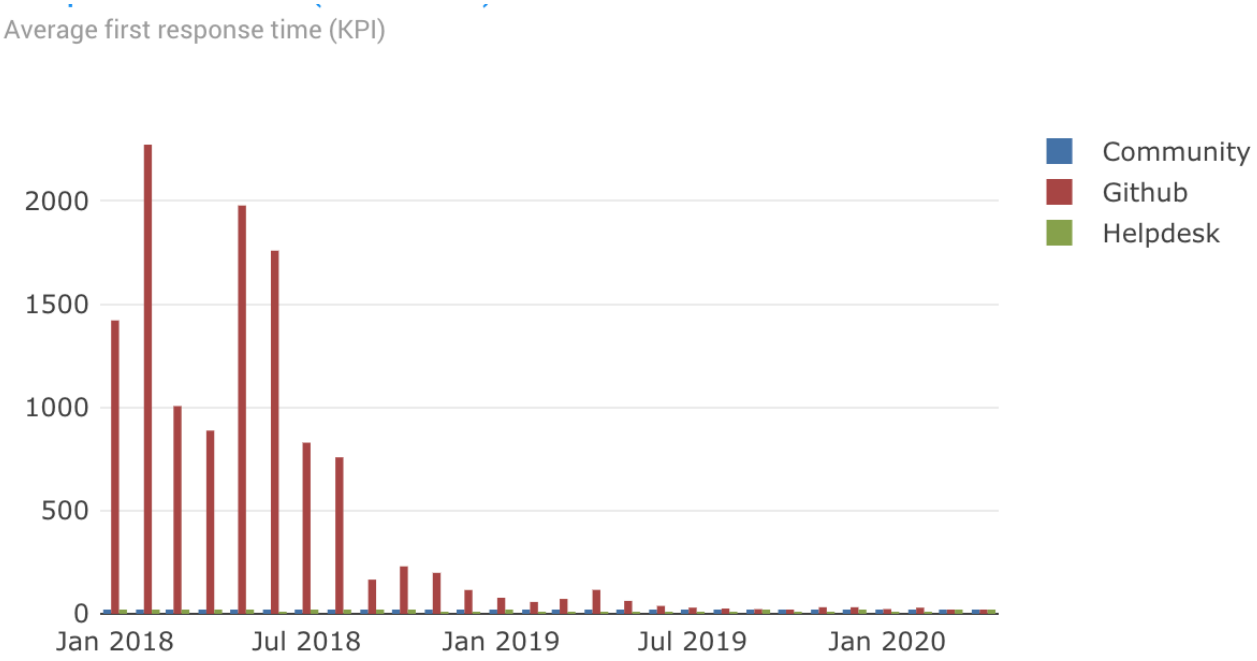


Figura 22 - Tiempo medio en dar una primera respuesta a una solicitud de soporte por mes.

Nota: instantáneas tomada el 19 de abril del 2020. El soporte procedente de Stackoverflow no es representado ya que es despreciable en proporción.

9 CONCLUSIONES Y LÍNEAS DE MEJORA

Information is just bits of data. Knowledge is putting them together. Wisdom is transcending them. - Ram Dass

Este trabajo ha supuesto una gran experiencia a nivel personal. El esfuerzo realizado me ha permitido profundizar significativamente mis conocimientos sobre Kubernetes, gracias también a la colaboración con otros equipos con mayor recorrido en la materia.

Respecto a los resultados, se puede afirmar que son realmente satisfactorios, ya que la solución responde a los retos que se planteaban al inicio del proyecto. De hecho, el trabajo realizado ha sido valorado muy positivamente por dirección.

En este apartado analizaremos, a modo de conclusión, el grado de cumplimiento de los objetivos iniciales. Por otro lado, se propondrán algunas posibles líneas de mejora.

9.1 Conclusiones

El principal objetivo de este proyecto radicaba en mejorar la productividad en el trabajo realizado por los agentes de soporte de Bitnami. En otras palabras, reducir los recursos humanos que Bitnami dedica a atender las solicitudes de soporte que recibe de sus usuarios. Este objetivo queda cumplido ya que, como podemos observar en el apartado 8.3 de la valoración, se reduce cerca del 40% el tiempo dedicado por los agentes de soporte por cada solicitud de soporte recibida.

Otro aspecto importante del proyecto era poder de extender y escalar los servicios de soporte ofrecidos a nuevas plataformas. Podemos concluir, haciendo lectura del apartado 8.3, que este reto también se cumple ya que, a pesar del importante aumento de la demanda de soporte en la plataforma GitHub, la solución ha permitido seguir atendiendo a los usuarios de dicha plataforma. De hecho, si observamos la evolución del tiempo medio para dar una primera respuesta y el tiempo medio para cerrar un ticket (ver apartado 8.5), se puede observar cómo, partiendo de una situación de descontrol en el soporte proporcionado a los usuarios de GitHub, se ha ido normalizando hasta tener en la actualidad una situación controlada y similar a la de otras plataformas.

Uno de los aspectos en los que más se insistió por parte de dirección fue en crear una solución sencilla y robusta, cuyos costes de mantenimiento fueran muy bajos. Como

podemos inferir del apartado 8.1 de la validación, este aspecto se ha cumplido con creces. La robustez queda acreditada por el bajo número de incidencias, el alto porcentaje de disponibilidad y el gran número de actualizaciones realizadas sin afectar la operatividad; mientras que la sencillez y los bajos coste de mantenimiento, quedan refrendados por el bajo coste que conlleva añadir una nueva funcionalidad a la aplicación. Por otro lado, la solución cumple con la totalidad de los requisitos técnicos listados en la tabla 1, cómo se puede validar en el apartado 8.2.

Por último, cabe mencionar que un objetivo secundario del proyecto era el poder tener métricas sobre el trabajo de soporte realizado en Bitnami. Antes de la existencia de Trello Updater y tener toda la información bajo un mismo portal, esto resultaba muy complejo. Gracias a este proyecto y el trabajo realizado de forma paralela en el "Dashboard de Soporte", se ha podido dar respuesta al deseo de dirección de tener datos sobre los que definir acuerdos de nivel de servicio y presentar informes objetivos a los proveedores de cloud (ver apartado 8.5).

9.2 Líneas de mejora

Son varios los aspectos a mejorar en el ámbito del soporte y la documentación partiendo de este primer elemento, Trello Updater, que permite automatizar y organizar el trabajo de una manera eficiente. Algunas de estos aspectos a mejorar son propios a la implementación de la aplicación y otros son proyectos complementarios para seguir mejorando en el servicio ofrecido y en el control e información que se tiene sobre el mismo. Los más importante son:

- El sistema de retroalimentación soporte-documentación puede mejorarse significativamente. En lugar de marcar casos de soporte para su posterior análisis con el equipo de documentación de forma bisemanal, se puede implementar un sistema que permita al agente de soporte crear una incidencia para ser resuelta por el equipo de documentación.
- Asignado automático de los agentes de soporte disponibles a cada una de las solicitudes recibidas. Actualmente este proceso es manual, cada agente se asigna tarjetas en Trello en función a la carga de trabajo que tiene.
- Utilizar técnicas de Machine Learning para detectar peticiones de soporte comunes y proveer respuestas automáticas basadas en el análisis de las respuestas anteriores dadas por los agentes de soporte a casos similares.

Por otro lado, hay una serie de mejoras técnicas que podemos implementar en la solución.

9.2.1 Mejoras sobre la gestión de credenciales

Cómo se detalla en el apartado 6.2.1, las credenciales se gestionan usando Sealed Secrets actualmente. Esta solución nos permite guardar las credenciales encriptadas en el repositorio de GIT. Sin embargo, encriptar las credenciales sigue siendo un proceso manual. Es decir, cada vez que hay que modificarlas, manualmente hay que ejecutar un comando de "kubeseal" que permita actualizar el objeto Sealed Secret asociado.

Una posible mejora en este aspecto es usar algún servicio de gestión de credenciales para Kubernetes, como por ejemplo Hashicorp Vault [82]. Una solución de este estilo, nos permitiría actualizar las credenciales a través de dicho sistema, y sería el propio sistema el encargado de actualizar los secretos asociados en el clúster, evitando así procesos manuales.

9.2.2 Mejoras sobre la monitorización

En la actualidad, Trello Updater no exporta métricas propias. Prometheus recoge métricas que el propio Kubernetes exporta sobre el uso de CPU y memoria en los pods, el volumen de tráfico dirigido hacia los mismos, etc. Sin embargo, es posible implementar métricas propias de la aplicación y exponerlas para que Prometheus pueda obtenerlas y posteriormente sacar gráficos de los que obtener información relevante en Grafana.

Hay dos alternativas para implementar estas métricas propias de la aplicación:

- Implementar las métricas en la propia aplicación siendo éstas expuestas en un "endpoint" concreto de la API. Por ejemplo "/metrics".
- Crear un servidor dedicado exclusivamente a la extracción de métricas relevantes de Trello Updater para su posterior exposición en el formato de métrica que Prometheus espera.

Aparte de implementar nuevas métricas, sería posible implementar alertas. Prometheus posee un componente, denominado Alertmanager, que permite analizar las métricas obtenidas y lanzar alertas si se cumplen ciertos criterios.

En la actualidad, se usa un servicio denominado "Pingy" que comprueba periódicamente que los distintos servicios de Bitnami devuelven una respuesta correcta al acceder a ellos. Este servicio está configurado para comprobar que Trello Updater está activo, entre otros servicios. Si se implementaran alertas más avanzadas usando Alertmanager, se podría prescindir de Pingy.

ANEXO A: WEBHOOKS

Los siguientes webhooks son utilizados por las diferentes plataformas de soporte para notificar nuevos cambios a Trello Updater (formato JSON):

Plataforma	Webhook
Community	<pre>{ "add": true/false, "ticket": "topic.id", "title": "topic.title", "status": "status", "type": "CommunitySupportTicket", "url": "topic.url", "agent": "user.username", "user": "user.username", "category": "category_name" }</pre>
HelpDesk	<pre>{ "ticket": "{{ticket.id}}", "title": "{{ticket.title}}", "status": "{{ticket.status}}", "type": "HelpdeskSupportTicket", "tags": "{{ticket.tags}}", "phtask": "{{ticket.ticket_field_33316948}}", "url": "https://{{ticket.url}}", "organization": "{{ticket.organization.name}}", "lastupdate": "{{ticket.latest_public_comment}}", "docsReason": "{{ticket.ticket_field_360002428733}}" }</pre>
GitHub	<pre>{ "login": "user123", "id": 1234567, "avatar_url": "https://avatars3.githubusercontent.com/u/1234567?v=4",</pre>

	`` ... "url": "https://api.github.com/users/user123", ... "organizations_url": "https://api.github.com/users/user123/orgs", "repos_url": "https://api.github.com/users/user123/repos", ... "type": "User", "site_admin": false } ```
StackoverFlow	``` { "add": true, "ticket": "", "title": "Serverfault Test Ticket", "status": "Open", "type": "StackExchangeSupportTicket", "url": "https://serverfault.com/questions" } ```
Tampermonkey	``` { "agent": "", "repo": "", // Sólo en GitHub "category": "", // Sólo en Community "overrideData": true, // Sólo en StackOverFlow "docsReason": "", "phtask": "ticket": "", "type": "", "title": "", "tags": "", "updateTags": true, "status": "", "url": "" } ```

ANEXO B: ESTRUCTURA DE FICHEROS, PAQUETES UTILIZADOS Y DIAGRAMA DE CLASES

En este anexo se detalla en profundidad la estructura de ficheros utilizada en el fichero, así como las clases utilizadas en el código.

B.1 Estructura de ficheros

En el siguiente árbol de directorios se puede observar los ficheros que conforman la aplicación Trello Updater.

```
.
├── lib
│   ├── factory
│   │   ├── helpdesk-tags.json
│   │   ├── HelpdeskTicketFactory.js
│   │   └── SupportTicketFactory.js
│   ├── tickets
│   │   ├── applications.json
│   │   ├── AutomaticHelpdeskSupportTicket.js
│   │   ├── communityapps.json
│   │   ├── CommunitySupportTicket.js
│   │   ├── customers.json
│   │   ├── GithubSupportTicket.js
│   │   ├── HelpdeskSupportTicket.js
│   │   ├── ISVHelpdeskSupportTicket.js
│   │   ├── SalesHelpdeskSupportTicket.js
│   │   ├── StackExchangeSupportTicket.js
│   │   ├── StacksmithHelpdeskSupportTicket.js
│   │   ├── SubscriptionSupportTicket.js
│   │   ├── SupportTicket.js
│   │   ├── tickets-lists-ids.json
│   │   └── WebdevHelpdeskSupportTicket.js
│   └── trello
│       ├── boards-ids.json
│       ├── communityapps.json
│       ├── githubapps.json
│       ├── githubfilter.json
│       ├── labelsFromTags.json
```

```
│       ├── TrelloClient.js
│       └── TrelloHandler.js
├── package.json
└──  server.js
```

Nótese que la estructura anterior no incluye los ficheros y directorios generados durante la instalación de las diferentes dependencias (obtenidas con el gestor de paquetes NPM) requeridas por la aplicación.

En el siguiente árbol de directorios se puede observar los ficheros que conforman los diferentes tests utilizados a aplicación Trello Updater.

```
.
└── test
    ├── helpers
    │   ├── clean-trello-board.js
    │   ├── keys-to-lists.json
    │   ├── trello-updater-api-handler.js
    │   ├── webhook-generator.js
    │   └── webhook-templates
    │       ├── community-tampermonkey-webhook.json.tpl
    │       ├── community-webhook.json.tpl
    │       ├── github-bitnami-user.json.tpl
    │       ├── github-non-bitnami-user.json.tpl
    │       ├── github-tampermonkey-webhook.json.tpl
    │       ├── github-webhook.json.tpl
    │       ├── helpdesk-webhook.json.tpl
    │       ├── serverfault-tampermonkey-webhook.json.tpl
    │       ├── serverfault-webhook.json.tpl
    │       ├── stackoverflow-tampermonkey-webhook.json.tpl
    │       └── stackoverflow-webhook.json.tpl
    ├── integration
    │   ├── community-basic-test.js
    │   ├── community-bnsupport-test.js
    │   ├── community-level-l3-test.js
    │   ├── community-with-task-test.js
    │   ├── github-basic-test.js
    │   ├── github-charts-test.js
    │   ├── github-k8s-squad-test.js
    │   ├── github-level-l3-test.js
    │   ├── github-with-task-test.js
    │   ├── helpdesk-automatic-test.js
    │   ├── helpdesk-basic-test.js
    │   ├── helpdesk-bnsupport-test.js
    │   ├── helpdesk-isv-test.js
    │   ├── helpdesk-level-l3-test.js
    │   ├── helpdesk-paid-customer-test.js
```

```
│   ├── helpdesk-password-recovery-test.js
│   ├── helpdesk-review-test.js
│   ├── helpdesk-sales-test.js
│   ├── helpdesk-stacksmith-squad-test.js
│   ├── helpdesk-stacksmith-test.js
│   ├── helpdesk-webdev-test.js
│   ├── serverfault-test.js
│   └── stackoverflow-test.js
├── test.sh
└── validation
    ├── check-no-duplicates.js
    └── check-trello-ids.js
```

B.2 Paquetes utilizados

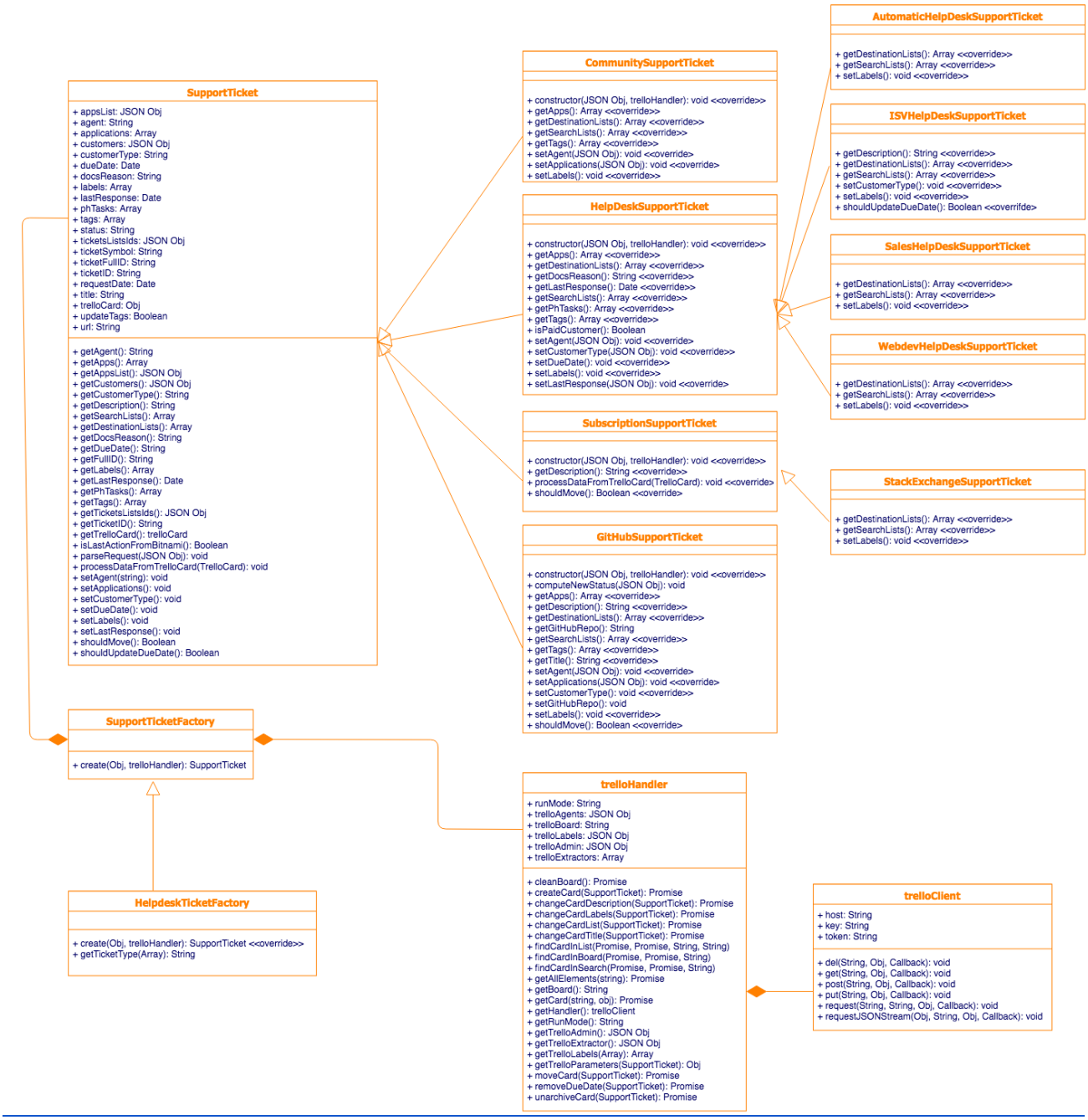
Los siguientes paquetes NPM han sido utilizados en desarrollo de la aplicación y en los distintos tests desarrollados:

	Aplicación	Tests
Paquetes NPM	• body-parser • canduit • debug • event-stream • express • express-healthcheck • googleapis • http • JSONStream • lodash • markdown-table • moment-timezone • process • promise-any • request • showdown • tableify	• chai • dateformat • eslint • eslint-config-airbnb-base • eslint-plugin-import • fake • mocha • path • request-promise

B.3 Diagrama de Clases

La figura siguiente muestra el diagrama de clases del proyecto siguiendo el lenguaje de

modelado UML:

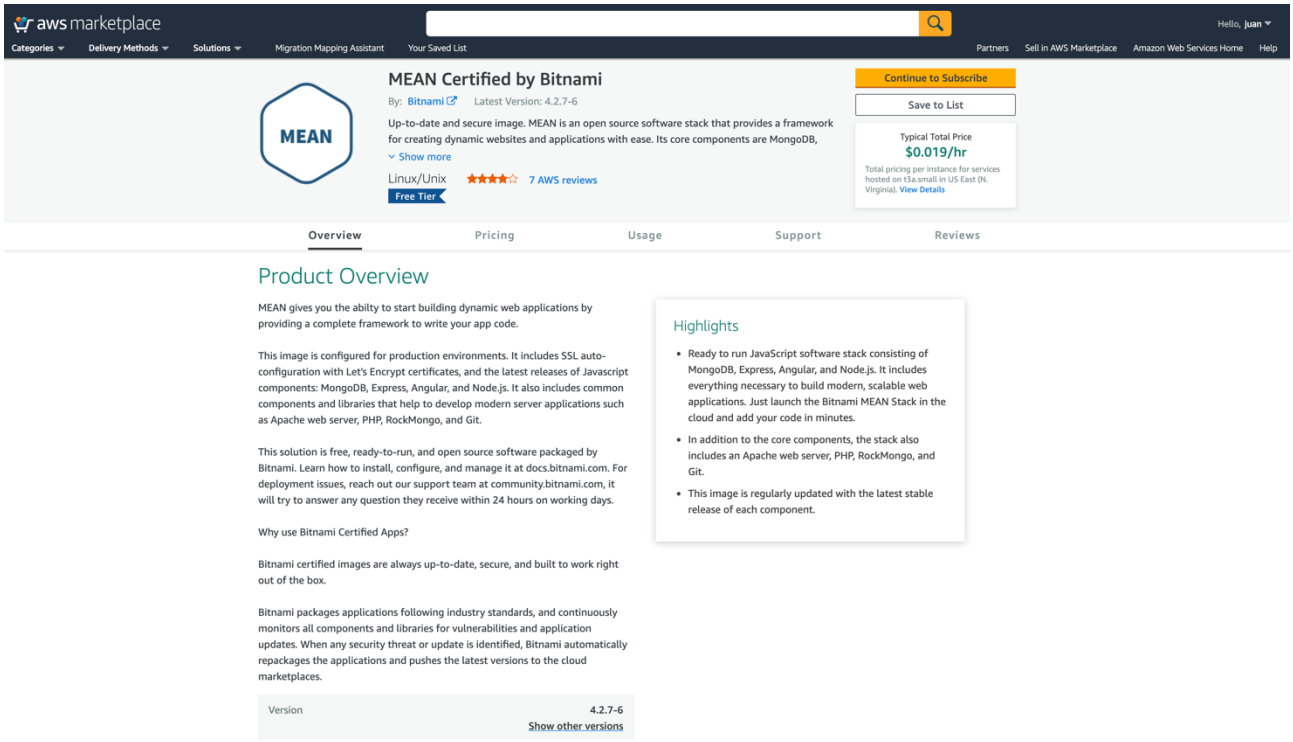


ANEXO C: DESPLIEGUE DE TRELLO UPDATER EN UNA INSTANCIA DE AWS

En este anexo se detalla el proceso de instalación de Trello Updater en una instancia de AWS.

C.1 Despliegue de una instancia basada en MEAN stack

La MEAN stack está disponible en el Marketplace de AWS, cómo se muestra en la siguiente captura de pantalla:



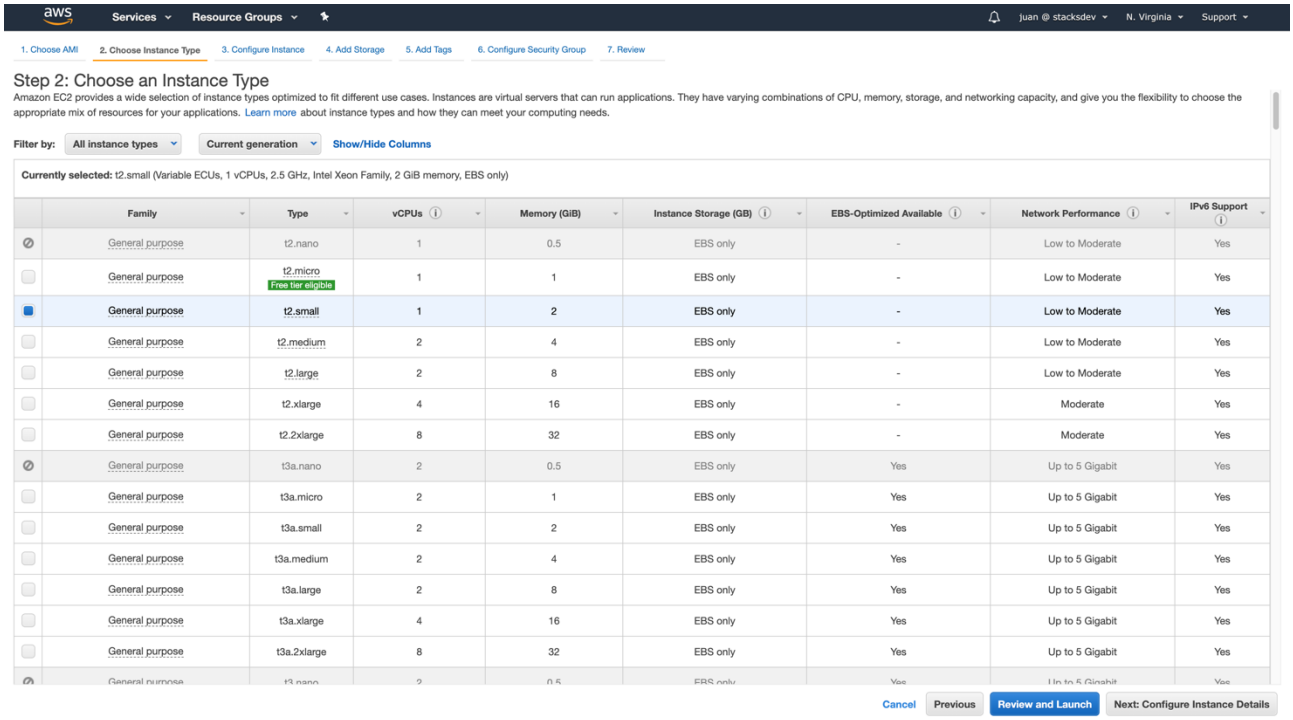
Desplegar una instancia basada en ella es muy sencillo. Basta con seguir los pasos indicados en la consola EC2 de AWS que consisten en:

- Elegir la región geográfica donde desplegar la instancia.
- Elegir el tipo de instancia (incluye el número de vCPUs, la memoria RAM, el tipo de almacenamiento, etc.)
- Elegir detalles de networking y permisos de usuarios.
- Elegir el "Security Group" a utilizar. Este nos permite elegir qué puertos están abiertos en la instancia y desde qué rangos de IPs aceptan conexiones.

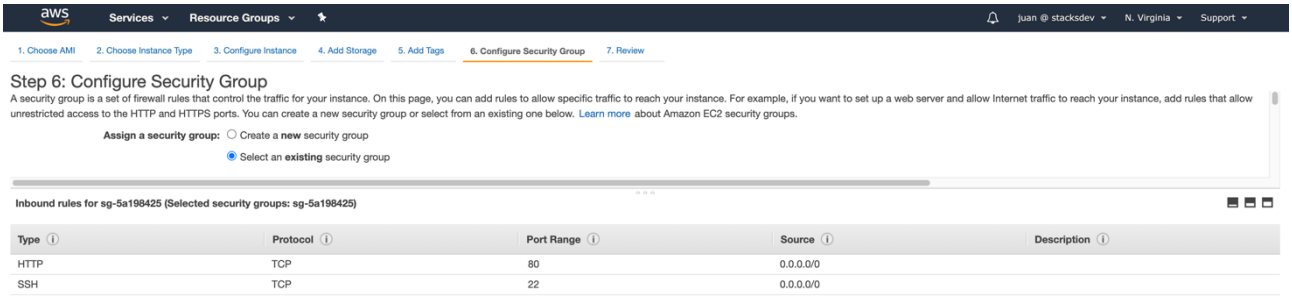
- La clave SSH pública a incluir como “authorized key” para poder acceder a la instancia vía SSH posteriormente.
- Desplegar definitivamente la instancia.

Las siguientes capturas muestran algunos de estos pasos:

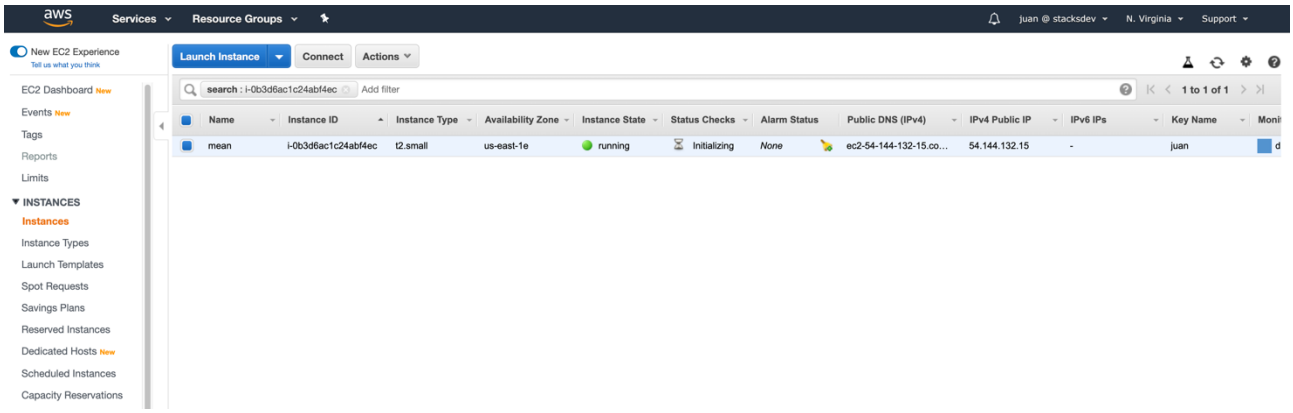
- Tipo de instancia:



- Security Groups:



Una vez lanzada, se puede obtener información sobre el estado de esta, su IP pública, etc. A través de la consola de EC2:



Podemos acceder vía SSH usando la clave privada y la IP obtenida:

```
juan:~$ ssh -i [redacted] bitnami@54.144.132.15
Linux ip-172-31-58-90 4.19.0-9-cloud-amd64 #1 SMP Debian 4.19.118-2 [redacted] x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
      ___ _ _                   _
     | _ |_) |_ _ _  __ _ _ __ (_)
     | _ \ |  _| ' \/ _` | '  \| |
     |___/_|\__|_|_|\__,_|_|_|_|_|

  *** Welcome to the Bitnami MEAN 4.2.7-6                    ***
  *** Documentation:  https://docs.bitnami.com/aws/infrastructure/mean/ ***
  ***                 https://docs.bitnami.com/aws/          ***
  *** Bitnami Forums: https://community.bitnami.com/         ***
bitnami@ip-172-31-58-90:~$
```

C.2 Instalación de Trello Updater

Puesto que MongoDB y Apache no son necesarios, se pueden parar dichos servicios y desactivarlos:

```
bitnami@ip-172-31-58-90:~$ sudo /opt/bitnami/ctlscript.sh status
apache already running
mongodb already running
bitnami@ip-172-31-58-90:~$ sudo /opt/bitnami/ctlscript.sh stop apache
Stopped apache
bitnami@ip-172-31-58-90:~$ sudo /opt/bitnami/ctlscript.sh stop mongodb
Stopped mongodb
bitnami@ip-172-31-58-90:~$ sudo mv /etc/monit/conf.d/mongodb.conf /etc/monit/conf.d/mongodb.conf.disabled
bitnami@ip-172-31-58-90:~$ sudo mv /etc/monit/conf.d/apache.conf /etc/monit/conf.d/apache.conf.disabled
```

Tras subir el código de la aplicación comprimido vía SFTP, es necesario descomprimirlo e instalar las dependencias:

```
bitnami@ip-172-31-58-90:~$ sudo mkdir -p /opt/bitnami/trello-updater && sudo chown $USER /opt/bitnami/trello-updater
bitnami@ip-172-31-58-90:~$ tar xfz trello-updater.tar.gz -C /opt/bitnami/trello-updater --strip-components=1
bitnami@ip-172-31-58-90:~$ ls /opt/bitnami/trello-updater/
lib  package.json  server.js  test
bitnami@ip-172-31-58-90:~$ cat > /opt/bitnami/trello-updater/settings.json << 'EOF'
> {
>   "allowedOrigins": [
>     "https://bitnami.zendesk.com",
>     "https://community.bitnami.com",
>     "http://stackoverflow.com",
>     "http://serverfault.com",
>     "https://stackoverflow.com",
>     "https://serverfault.com",
>     "https://github.com"
>   ],
>   "runMode": "production",
>   "port": 3000
> }
> EOF
bitnami@ip-172-31-58-90:~$ cd /opt/bitnami/trello-updater/ && npm install
[redacted]
added 434 packages from 288 contributors and audited 436 packages in 17.185s
```

Por último, se configura Monit para monitorizar la aplicación y enviar alertas en caso de que algo vaya mal:

```
bitnami@ip-172-31-58-90:~$ cat /etc/monit/conf.d/trello-updater.conf
check program trello-updater with path /opt/bitnami/trello-updater/monit/health.sh
  start program = "npm --prefix /opt/bitnami/trello-updater/ start"
  stop program = "npm --prefix /opt/bitnami/trello-updater/ stop"
  if status != 0 for 2 cycles then restart
  if 3 restarts within 3 cycles then exec "/opt/bitnami/trello-updater/monit/slack_failure.sh"
bitnami@ip-172-31-58-90:~$ cat /opt/bitnami/trello-updater/monit/health.sh
#!/bin/bash


[ $(curl -o -I -L -s -w %{http_code} http://127.0.0.1:3000/healthcheck) -eq 200 ] || exit 1
bitnami@ip-172-31-58-90:~$ cat /opt/bitnami/trello-updater/monit/slack_failure.sh
#!/bin/bash

curl -X POST \
     -H 'Content-type: application/json' \
     --data '{"text": "@support-team *Trello Updater server* is down!!", "link_names": 1}' \
     https://hooks.slack.com/services/[redacted]
bitnami@ip-172-31-58-90:~$ sudo monit reload
```

Con este último paso, se completa el proceso y podemos iniciar la aplicación y comenzar a enviar webhooks a la misma en el momento en que se desee.

ANEXO D: TESTS E INTEGRACIÓN CONTINUA

En este anexo se describe con mayor detalle los tests utilizados y la solución propuesta para la implementación de un sistema de Integración Continua.

D.1 Batería de Tests

El siguiente script, utilizado para lanzar la batería de tests disponible, nos muestra la totalidad de estos y cómo han de ser ejecutados:

```
#!/bin/bash -m

## Check Trello IDs
mocha --timeout 20000 ./validation/check-trello-ids.js

if [ "$TESTING_MODE" == "testing" ]; then
  ## Clean Testing Board
  node ./helpers/clean-trello-board.js
  ## Test Community
  mocha --timeout 20000 ./integration/community-basic-test.js
  mocha --timeout 20000 ./integration/community-with-task-test.js
  mocha --timeout 20000 ./integration/community-level-l3-test.js
  mocha --timeout 20000 ./integration/community-bnsupport-test.js
  ## Test HelpDesk
  mocha --timeout 20000 ./integration/helpdesk-basic-test.js
  mocha --timeout 20000 ./integration/helpdesk-level-l3-test.js
  mocha --timeout 20000 ./integration/helpdesk-paid-customer-test.js
  mocha --timeout 20000 ./integration/helpdesk-review-test.js
  mocha --timeout 20000 ./integration/helpdesk-isv-test.js
  mocha --timeout 20000 ./integration/helpdesk-sales-test.js
  mocha --timeout 20000 ./integration/helpdesk-automatic-test.js
  mocha --timeout 20000 ./integration/helpdesk-password-recovery-test.js
  mocha --timeout 20000 ./integration/helpdesk-webdev-test.js
  mocha --timeout 20000 ./integration/helpdesk-stacksmith-test.js
  mocha --timeout 20000 ./helpdesk-stacksmith-squad-test.js
```

```
  mocha --timeout 20000 ./integration/helpdesk-bnsupport-test.js
  ## Test StackExchange
  mocha --timeout 20000 ./integration/stackoverflow-test.js
  mocha --timeout 20000 ./integration/serverfault-test.js
  ## Test GitHub
  mocha --timeout 20000 ./integration/github-basic-test.js
  mocha --timeout 20000 ./integration/github-with-task-test.js
  mocha --timeout 20000 ./integration/github-level-l3-test.js
  mocha --timeout 20000 ./integration/github-k8s-squad-test.js
  mocha --timeout 20000 ./integration/github-charts-test.js
  ## Test No Duplicates
  mocha --timeout 20000 ./validation/check-no-duplicates.js
fi
```

Se observa que hay una serie de tests que sólo son ejecutados cuando el modo de testeo es preproducción (del inglés “staging”). Esto se debe a que el mismo script es utilizado tanto para ejecutar los tests de validación en integración en el entorno de preproducción, como para realizar unas mínimas pruebas de validación en el entorno de producción.

El hecho de utilizar el mismo código para ejecutar los tests en dos modos distintos es muy importante puesto que permite reutilizar el mismo contenedor, disminuyendo así la complejidad y el coste de mantenimiento.

Hay 4 grupos de tests, uno por cada plataforma de soporte. Dentro de cada grupo vemos distintos tipos de tests, algunos de los más importantes son:

- -basic- tests: Son tests que simulan lo que ocurre cuando los usuarios de una plataforma de soporte concreta y el agente que les responde interactúan.
- -level-l3- tests: Simulan lo que ocurre cuando un agente de soporte decide escalar un caso de soporte cuando este es muy complejo.
- -with-task- tests: Simulan lo que ocurre cuando un agente de soporte crea una tarea al equipo de ingeniería para resolver una incidencia reportada y la asigna a un caso de soporte concreto.

Otros tests interesantes son “clean-trello-board.js” y “check-no-duplicates.js”. El primero, limpia el tablón de Trello utilizado para los tests para que el resultado de los tests previamente ejecutados no afecten a la nueva batería de tests. Por otro lado, el segundo comprueba que como resultado de haber realizados la batería de tests no se hayan generado tarjetas duplicadas en el tablón.

D.2 CI/CD: Configuración

El sistema de integración y despliegue continuo se implementa a través de la herramienta Jenkins.

Para ello se utiliza un “trabajo” dentro de un servidor Jenkins ya existente y gestionado por el equipo de SRE de Bitnami. Dicho servidor Jenkins posee las credenciales necesarias para realizar cambios en dos clústeres de Kubernetes (preproducción y producción).

El trabajo implementado consta las siguientes etapas:

- Obtención de parámetros de entrada:
  - Clúster destino
  - Rama o confirmación del repositorio.
- Testeo del código Jsonnet en busca de fallos de sintaxis y validando que los objetos definidos cumplen las especificaciones del API de Kubernetes.
- Detección de cambios entre la solución ya desplegada en el clúster y aquella que se quiere testear.
- Despliegue de los nuevos cambios a testear si procediese.
- Ejecución de los tests de validación e integración por medio de un job de Kubernetes.
- Obtención de los resultados y eliminación del job utilizado para los tests.

El siguiente fichero de configuración de Jenkins refleja las etapas descritas en lenguaje Groovy:

```
#!groovy
@Library(['jenkins-pipeline-kubernetes', 'jenkins-pipeline-internal-kubernetes']) _

properties([
  parameters([
    choice(
      name: 'DEPLOY_ENV',
      choices: 'auto\nproduction\nstaging',
      description: 'Target K8s cluster',
      ),
    stringParam(
      name: 'BRANCH_NAME',
      defaultValue: 'master',
      description: 'git@endor:trello-updater branch',
)])])

kube = new com.bitnami.kubernetes.KubeTools()
kube_clusters = new com.bitnami.internal.kubernetes.Clusters()

// Constants
APP_NAME = 'trello-updater'
TESTING_JOB_NAME = 'testing-job'
env.BRANCH_NAME = params.BRANCH_NAME

def setEnvironment(String branch_name) {
  config = [:]
  config.branch_name = branch_name
  ...
  switch (env.DEPLOY_ENV) {
    case 'production':
      config.name = 'g.web'
```

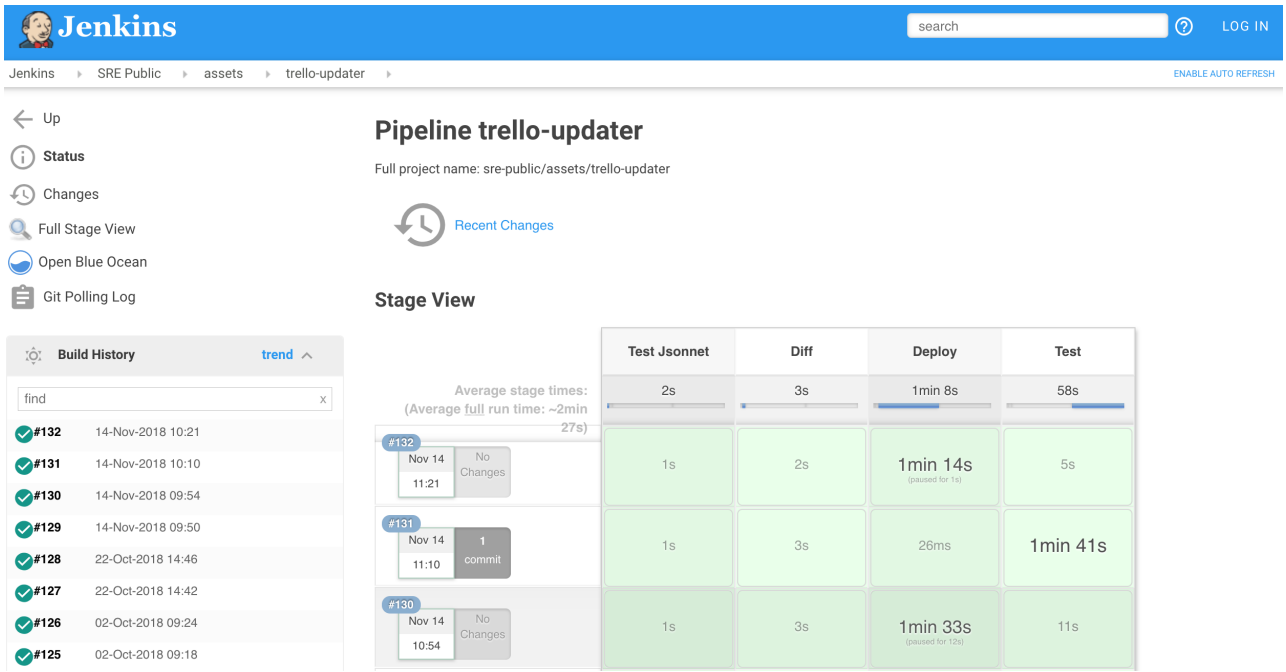
```
      config.jsonnet_file = "manifests/${APP_NAME}-production.jsonnet"
      config.jsonnet_testing = "manifests/${TESTING_JOB_NAME}-production.jsonnet"
    default:
      config.name = 'g.dev'
      config.jsonnet_file = "manifests/${APP_NAME}-staging.jsonnet"
      config.jsonnet_testing = "manifests/${TESTING_JOB_NAME}-staging.jsonnet"
    }
  config.auth_token_id = config.name + '-jenkins-trello-updater-serviceaccount-token'
  config.kube_ctx = kube_clusters.getContext(config.name, config.auth_token_id)
  return config
}

try {
  timeout(time: 30, unit: 'MINUTES') {
    node('master') {
      …
        stage('Test Jsonnet') {
          parallel(
            fmt: {
              if (env.DEPLOY_ENV == 'staging') {
                kube.testFmt(myenv.jsonnet_testing)
              }
              kube.testFmt(myenv.jsonnet_file)
            },
            validate: {
              if (env.DEPLOY_ENV == 'staging') {
                kube.testValid(myenv.jsonnet_testing)
              }
              kube.testValid(myenv.jsonnet_file)
        })},
        stage('Diff') {
          diff_rc = kube.kubecfgDiff(myenv.kube_ctx, myenv.jsonnet_file)
          diff_test = kube.kubecfgDiff(myenv.kube_ctx, myenv.jsonnet_testing)
        }
        stage('Deploy') {
          if (diff_rc) {
            if (env.DEPLOY_ENV == 'production') {
              ...
            kube.kubecfgUpdate(myenv.kube_ctx, myenv.jsonnet_file)
          } else {
            echo "No diffs, not deploying"
        }}
        stage('Test') {
          if (diff_test) {
            kube.kubecfgUpdate(myenv.kube_ctx, myenv.jsonnet_testing)
            def job_name = "${TESTING_JOB_NAME}-dev"
            if (env.DEPLOY_ENV == 'production') {
              job_name = "${TESTING_JOB_NAME}-prod"
```

```
            }
            kube.withContext(myenv.kube_ctx) {
              sh "jenkins/wait-for-job.sh ${job_name} ${APP_NAME}"
            }}
            kube.kubecfgDelete(myenv.kube_ctx, myenv.jsonnet_testing)
          } else {
            echo "No diffs, not deploying job"
  }}}}}
} catch (e) {
  msg = "Hello @support-team, Trello Updater deployment to ${env.DEPLOY_ENV}
failed.\nCheck <${env.BUILD_URL}|this link> for more information.\n"
  slackSend channel: "#assets-alerts", color: 'danger', message: msg
  throw e
}
```

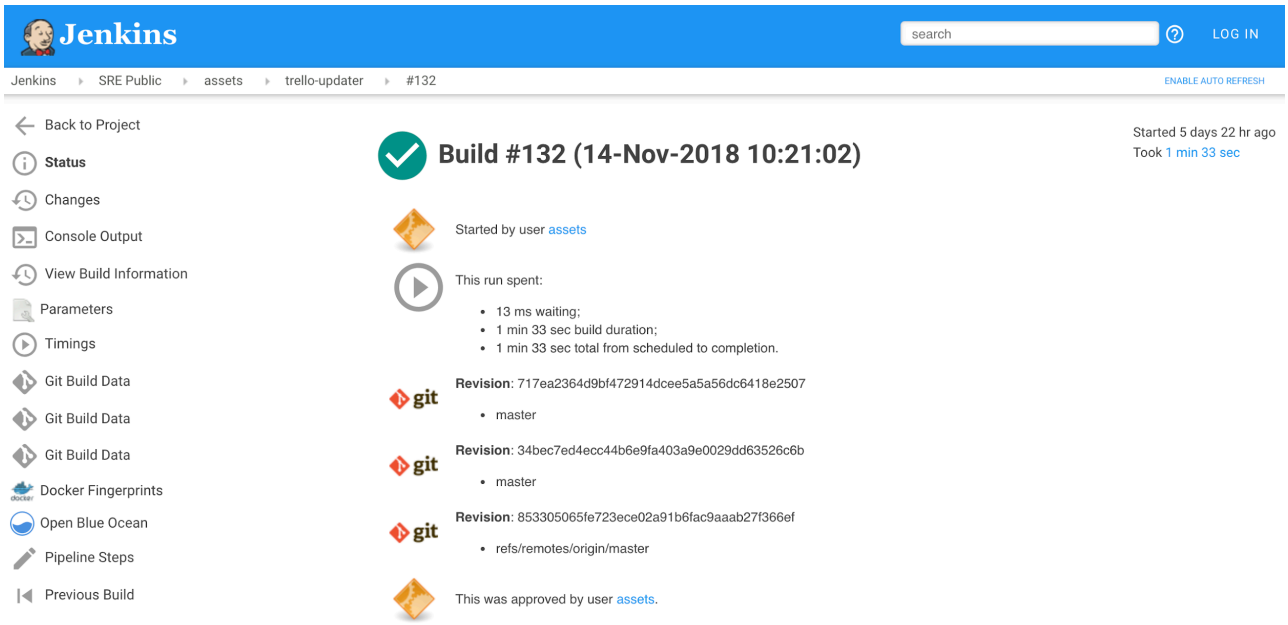
D.3 CI/CD: Interfaz Gráfica

Jenkins ofrece una interfaz gráfica muy intuitiva que permite a sus usuarios lanzar una ejecución de un trabajo concreto y acceder a sus resultados.

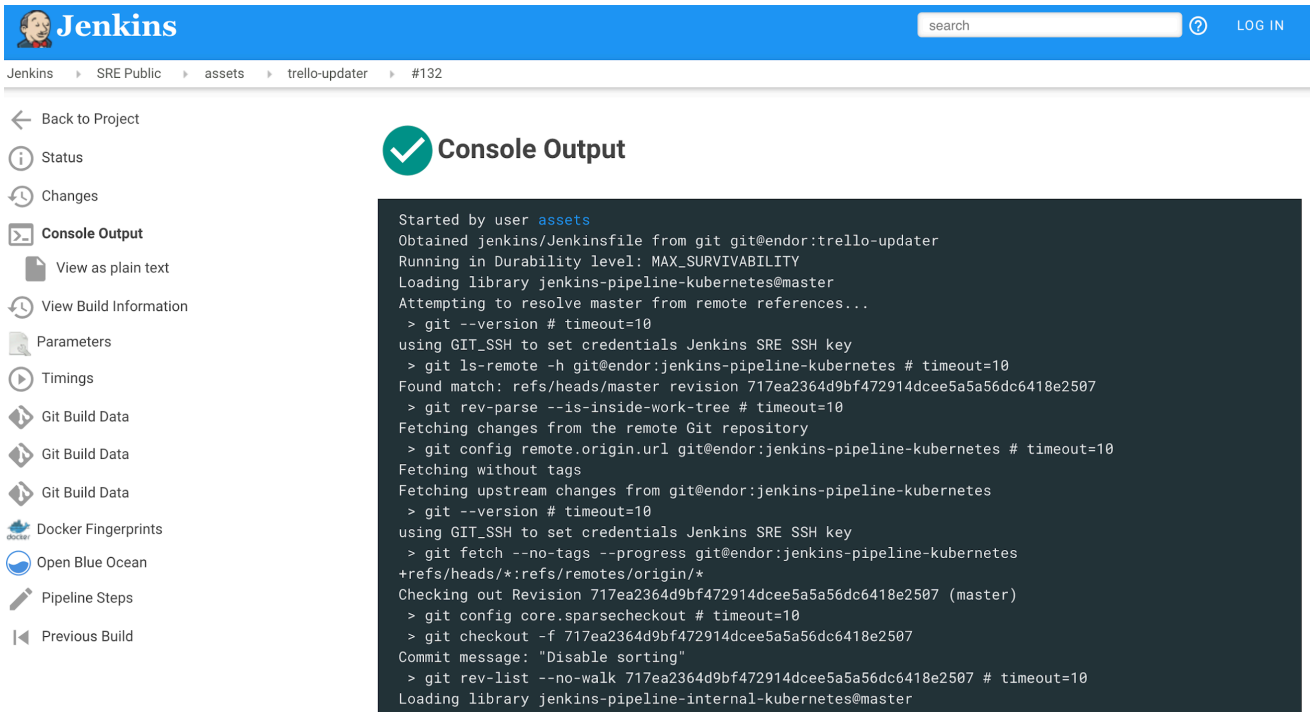
En la siguiente captura de pantalla se puede observar la vista general del trabajo utilizado para implementar el sistema CI/CD:



Se puede acceder al resultado de una ejecución concreta para obtener más información sobre la misma y acceder a los “logs” de los tests. La siguiente captura muestra la información detallada de una ejecución concreta:



Se observa que existe la posibilidad de acceder a la salida por consola (basta con hacer click en el campo “Console Output”). Un ejemplo de esta casuística puede observarse en la siguiente imagen:



ANEXO E: OBJETOS DE KUBERNETES

La solución que nos permite desplegar en Kubernetes está compuesta por los siguientes objetos de Kubernetes:

- 1 Deployment (define a su vez el ReplicaSet y Pod(s) asociados).
- 1 Horizontal Pod Autoscaler
- 1 Service.
- 1 SealedSecret y su Secret asociado.
- 1 Ingress Rule.
- 1 TLS Secret.

Todos estos objetos pueden se definen como un objeto de API en Kubernetes, y pueden ser creados en el clúster usando el cliente de línea de comandos ***kubectl*** tal como se muestra en el siguiente comando, donde '***manifest.yaml***' ha de contener la definición en formato YAML del objeto correspondiente:

```
$ kubectl create -f manifest.yaml
```

El uso de este tipo de comandos no será necesario, puesto que en este proyecto se usa jsonnet y kubecfg para empaquetar y desplegar la solución, respectivamente.

La solución está complementada por los servicios gestionados por el equipo de SRE de Bitnami siguientes:

- NGINX Ingress Controller.
- External DNS.
- CertManager.
- ElasticSearch, Logstash y Kibana.
- Prometheus y Grafana.

E.1 Definición en formato YAML de los objetos de Kubernetes

Objeto	Definición
Deployment	apiVersion: apps/v1 kind: Deployment metadata: labels:

```
    app: trello-updater
    name: trello-updater
  name: trello-updater
spec:
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: trello-updater
      name: trello-updater
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: trello-updater
        name: trello-updater-prod
    spec:
      containers:
      - env:
        - name: DEBUG
          value: TrelloUpdaterServer
        image: gcr.io/bitnami-images/trello-updater:{TAG_NAME}
        imagePullPolicy: IfNotPresent
        name: trello-updater
        ports:
        - containerPort: 3000
          name: http
          protocol: TCP
        resources:
          limits:
            cpu: 100m
            memory: 100Mi
          requests:
            cpu: 50m
            memory: 80Mi
        volumeMounts:
        - mountPath: /app/lib/trello/agentstrello
          name: agentstrello-creds
        - mountPath: /app/lib/trello/communityagents
```

```
    name: communityagents-creds
  - mountPath: /app/lib/trello/githubagents
    name: githubagents-creds
  - mountPath: /app/lib/trello/helpdeskagents
    name: helpdeskagents-creds
  - mountPath: /app/settings
    name: settings
securityContext:
  fsGroup: 1001
  runAsUser: 1001
volumes:
- name: agentstrello-creds
  secret:
    defaultMode: 420
    items:
    - key: agentstrello.json
      path: agentstrello.json
    secretName: trello-updater-prod
- name: communityagents-creds
  secret:
    defaultMode: 420
    items:
    - key: communityagents.json
      path: communityagents.json
    secretName: trello-updater-prod
- name: githubagents-creds
  secret:
    defaultMode: 420
    items:
    - key: githubagents.json
      path: githubagents.json
    secretName: trello-updater-prod
- name: helpdeskagents-creds
  secret:
    defaultMode: 420
    items:
    - key: helpdeskagents.json
      path: helpdeskagents.json
    secretName: trello-updater-prod
- name: settings
  secret:
    defaultMode: 420
```

	items: - key: production-settings.json path: settings.json secretName: trello-updater-prod
Horizontal Pod Autoscaler	apiVersion: autoscaling/v2beta1 kind: HorizontalPodAutoscaler metadata: labels: app: trello-updater name: trello-updater name: trello-updater spec: scaleTargetRef: apiVersion: apps/v1 kind: Deployment name: trello-updater minReplicas: 2 maxReplicas: 10 metrics: - type: Resource resource: name: cpu targetAverageUtilization: 70 - type: Resource resource: name: memory targetAverageUtilization: 70
Service	apiVersion: v1 kind: Service metadata: labels: app: trello-updater name: trello-updater name: trello-updater spec: ports: - port: 3000 protocol: TCP targetPort: 3000 selector: app: trello-updater name: trello-updater

	<pre> sessionAffinity: None type: ClusterIP</pre>
SealedSecret	<pre>apiVersion: bitnami.com/v1alpha1 kind: SealedSecret metadata: labels: app: trello-updater name: trello-updater name: trello-updater spec: encryptedData: ...</pre>
Ingress Rule	<pre>apiVersion: extensions/v1beta1 kind: Ingress metadata: annotations: certmanager.k8s.io/acme-challenge-type: dns01 certmanager.k8s.io/acme-dns01-provider: default certmanager.k8s.io/cluster-issuer: letsencrypt-dns ingress.kubernetes.io/whitelist-source-range: 0.0.0.0/0 nginx.ingress.kubernetes.io/whitelist-source-range: 0.0.0.0/0 stable.k8s.psg.io/kcm.email: sre@bitnami.com stable.k8s.psg.io/kcm.provider: route53 labels: app: trello-updater name: trello-updater stable.k8s.psg.io/kcm.class: default name: trello-updater spec: rules: - host: trello-updater.g.bitnami.net http: paths: - backend: serviceName: trello-updater servicePort: 3000 path: / tls: - hosts: - trello-updater.g.bitnami.net secretName: trello-updater-cert</pre>

TLS Secret

```
apiVersion: v1
data:
  ca.crt: ""
  tls.crt: ...
  tls.key: ...
  tls.pem: ...
kind: Secret
metadata:
  annotations:
    certmanager.k8s.io/alt-names: trello-
updater.g.bitnami.net
    certmanager.k8s.io/common-name: trello-
updater.g.bitnami.net
    certmanager.k8s.io/ip-sans: ""
    certmanager.k8s.io/issuer-kind: ClusterIssuer
    certmanager.k8s.io/issuer-name: letsencrypt-dns
  labels:
    certmanager.k8s.io/certificate-name: trello-updater-prod-
cert
    stable.k8s.psg.io/kcm.class: default
    stable.k8s.psg.io/kcm.domain: trello-
updater.g.bitnami.net
  name: trello-updater-cert
type: kubernetes.io/tls
```

ANEXO F: EMPAQUETADO DE LA SOLUCIÓN, JSONNET

Haciendo uso del lenguaje de plantilla Jsonnet y su versatilidad, se empaqueta los distintos objetos de Kubernetes a desplegar en el siguiente modelo genérico:

```
local kube = import "lib/kube.libsonnet";
local bitnami = import "lib/bitnami.libsonnet";

local labels = {
  app: "trello-updater",
};
local secretVolPath(secret, path) = kube.SecretVolume(secret) {
  secret+: { items: [{ key: path, path: path } ]},
};

local trello_updater = {
  namespace:: "trello-updater",
  app_name:: labels.app,
  settings_file:: error "Must specify settings file",
  hostname:: error "Must specify hostname",
  ingress: bitnami.Ingress($.app_name) {
    host:: $.hostname,
    metadata+: {
      namespace: $.namespace,
      labels+: { app: labels.app, name: $.app_name },
      annotations+: {
        "nginx.ingress.kubernetes.io/whitelist-source-range": "0.0.0.0/0",
      },
    },
    paths: [
      { path: "/", backend: $.service.name_port },
    ],
  },
  service: kube.Service($.app_name) {
    metadata+: {
      namespace: $.namespace,
      labels+: { app: labels.app, name: $.app_name }
    },
    target_pod: $.deployment.spec.template,
  },
  deployment: kube.Deployment($.app_name) {
```

```
    metadata+: {
      namespace: $.namespace,
      labels+: { app: labels.app, name: $.app_name }
    },
    spec+: {
      template+: {
        spec+: {
          containers_+: {
            default: kube.Container($.app_name) {
              image: "gcr.io/bitnami-images/trello-updater:20190514095459",
              ports_+: {
                http: { containerPort: 3000 },
              },
              env_+: { "DEBUG": "TrelloUpdaterServer" },
              volumeMounts_+: {
                settings: { mountPath: "/app/settings" },
                agentstrello_creds: { mountPath: "/app/lib/trello/agentstrello" },
                communityagents_creds: { mountPath: "/app/lib/trello/communityagents" },
                githubagents_creds: { mountPath: "/app/lib/trello/githubagents" },
                helpdeskagents_creds: { mountPath: "/app/lib/trello/helpdeskagents" },
              },
              resources: {
                requests: { memory: "80Mi", cpu: "50m" },
                limits:  { memory: "100Mi", cpu: "100m" },
              },
            },
          },
          volumes_+: {
            settings: kube.SecretVolume($.sealedsecret) {
              secret+: { items: [{ key: $.settings_file, path: "settings.json" } ]},
            },
            agentstrello_creds: secretVolPath($.sealedsecret, "agentstrello.json"),
            communityagents_creds: secretVolPath($.sealedsecret, "communityagents.json"),
            githubagents_creds: secretVolPath($.sealedsecret, "githubagents.json"),
            helpdeskagents_creds: secretVolPath($.sealedsecret, "helpdeskagents.json"),
          },
        },
      },
    },
  },
};

kube.List() {
  items_+:
    trello_updater,
}
```

Una vez se tiene el modelo genérico, se crean dos nuevos modelos basados en él:

- Uno se utiliza para desplegar la solución en el entorno de testeo o preproducción. En este entorno se realizarán una serie de tests para garantizar que los últimos cambios introducidos en la aplicación están listos para ser desplegados en producción sin afectar el servicio.
- El segundo modelo, será utilizado para desplegar la aplicación en el entorno de

producción.

Modelo de testeo o preproducción:

```
local trello_updater = import "trello-updater.jsonnet";
local sealed_secret = import "trello-updater-ss-staging.json";

trello_updater {
  items_+: {
    app_name: "trello-updater-dev",
    hostname: "trello-updater.g.dev.bitnami.net",
    settings_file: "testing-settings.json",
    deployment+: { spec+: { replicas: 2 } },
    sealedsecret: sealed_secret,
  },
}
```

Modelo de testeo o preproducción:

```
local trello_updater = import "trello-updater.jsonnet";
local sealed_secret = import "trello-updater-ss-production.json";

trello_updater {
  items_+: {
    app_name: "trello-updater-prod",
    hostname: "trello-updater.g.web.bitnami.net",
    settings_file: "production-settings.json",
    deployment+: { spec+: { replicas: 2 } },
    sealedsecret: sealed_secret,
  },
}
```

REFERENCIAS

[1] **VMware, Inc.** *Bitnami Community forum*. Bitnami. Disponible en: https://community.bitnami.com [Último acceso: Mayo 2020].

[2] **VMware, Inc.** *Bitnami HelpDesk platform*. Bitnami. Disponible en: https://helpdesk.bitnami.com [Último acceso: Febrero 2020]

[3] **Zendesk, Inc.** *Zendesk Support Suite*. Zendesk. Disponible en: https://www.zendesk.com/support-suite [Último acceso: Enero 2020].

[4] **Stack Exchange Inc.** *Stackoverflow forum*. Stackoverflow. Disponible en: https://stackoverflow.com/questions [Último acceso: Abril 2020].

[5] **Microsoft Corporation.** *GitHub Platform*. GitHub. Disponible en: https://github.com [Último acceso: Mayo 2020].

[6] **CNCF**, *Kubernetes site*. Kubernetes. Disponible en: https://kubernetes.io [Último acceso: Enero 2020].

[7] **J. DeMuro y B. Turner**. *Best helpdesk software for ticketing and support.* Techradar. Disponible en: https://www.techradar.com/best/best-helpdesk-software [Último acceso: Marzo 2019].

[8] **B. Turner y N. Fearn**. *Best task management apps*. Techradar. Disponible en: https://www.techradar.com/best/best-task-management-apps [Último acceso: Marzo 2019].

[9] **Atlassian Inc**. *Trello Platform*. Trello. Disponible en: https://trello.com [Último acceso: Mayo 2020]

[10] **Shore Labs**. *Metodología Kanban*. Kanban Tool. Disponible en: https://kanbantool.com/es/metodologia-kanban [Último acceso: Enero 2018].

[11] **Atlassian, Inc**. *Trello API Documentation*. Trello. Disponible en: https://developers.trello.com/v1.0/reference [Último acceso: Febrero 2019].

[12] **Docker Inc.** *Docker Engine*. Docker . Disponible en: https://www.docker.com/products/container-runtime [Último acceso: 2020].

[13] **Red Hat, Inc.** *CoreOS rkt*. CoreOS. Disponible en: https://coreos.com/rkt [Último acceso: Marzo 2019].

[14] **The Apache Software Foundation.** *Apache Mesos*. Apache. Disponible en:

http://mesos.apache.org/documentation/latest/mesos-containerizer [Último acceso: Marzo 2019].

[15] **P. Menage.** *CGroups - Kernel Documentation*. Kernel. Disponible en: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt [Último acceso: Marzo 2019].

[16] **A. Verma.** *Large-scale cluster management at Google with Borg*. Google. Disponible en: https://ai.google/research/pubs/pub43438 [Último acceso: Marzo 2019].

[17] **The Linux Foundation®.** *Cloud Native Computing Foundation*. CNCF. Disponible en: https://www.cncf.io [Último acceso: Abril 2020].

[18] **The Linux Foundation®.** *The Linux Foundation Project.* Linux Foundation. Disponible en: https://www.linuxfoundation.org [Último acceso: Abril 2020].

[19] **Slack Technologies, Inc.** *Slack inroduction*. Slack. Disponible en: https://slack.com [Último acceso: Mayo 2020].

[20] **The Linux Foundation®.** *Helm package manager*. Helm. Disponible en: https://helm.sh [Último acceso: Mayo 2020].

[21] **VMware, Inc.** *Kubeapps site*. Kubeapps. Disponible en: https://kubeapps.com [Último acceso: Mayo 2020].

[22] **VMware, Inc.** *Kubeless site*. Kubeless. Disponible en: https://kubeless.io [Último acceso: Enero 2020].

[23] **VMware, Inc.** Kubecfg GitHub repository. GitHub. Disponible en: https://github.com/bitnami/kubecfg [Último acceso: Febrero 2020].

[24] **VMware, Inc.** Sealed Secrets GitHub repository. GitHub. Disponible en: https://github.com/bitnami-labs/sealed-secrets [Último acceso: Febrero 2020].

[25] **VMware, Inc.** *Bitnami Kubernetes Production Runtime.* Kubeprod . Disponible en: https://kubeprod.io [Último acceso: Febrero 2020].

[26] **M. Desai y P. Fazzone.** *VMware to Acquire Bitnami.* VMware. Disponible en: https://cloud.vmware.com/community/2019/05/15/vmware-to-acquire-bitnami [Último acceso: Mayo 2019].

[27] **Katalon LLC.** *Best 14 CI/CD Tools You Must Know, 2020*. Katalon. Disponible en: https://www.katalon.com/resources-center/blog/ci-cd-tools [Último acceso: Febrero 2018].

[28] **CloudBees**. *Jenkins site.* Jenkins . Disponible en: https://jenkins.io [Último acceso: Febrero 2018].

[29] **OpenJS Foundation.** *NodeJS site*. Nodejs. Disponible en: https://nodejs.org [Último acceso: Marzo 2019].

[30] **Node.js Foundation.** *Express site*. Express. Disponible en: https://expressjs.com [Último acceso: Marzo 2019].

[31] **R. Jin.** *Webhook vs API: What's the difference?*. Hackernoon. Disponible en: https://hackernoon.com/webhook-vs-api-whats-the-difference-8d41e6661652 [Último acceso: Enero 2018].

[32] **Civilized Discourse Construction Kit, Inc.** *Discourse site.* Discourse. Disponible en: https://www.discourse.org [Último acceso: Enero 2019].

[33] **Ruby Community**. *Documentación de Ruby*. Ruby Lang. Disponible en: https://www.ruby-lang.org/es/documentation [Último acceso: Marzo 2019].

[34] **Ruby Community.** *Ruby on Rails guides*. Ruby on Rails. Disponible en: https://guides.rubyonrails.org [Último acceso: Marzo 2019].

[35] **J. Salmerón García y J. Ariza Toledano.** *Community Trello repository*. GitHub. Disponible en: https://github.com/bitnami/community-trello [Último acceso: Enero 2019].

[36] **C. Nadeau.** *Creating webhooks with the HTTP target*. Zendesk. Disponible en: https://support.zendesk.com/hc/en-us/articles/204890268-Creating-webhooks-with-the-HTTP-target [Último acceso: Enero 2018].

[37] **IETF**, *The JavaScript Object Notation (JSON) Data Interchange Format*. IETF. Disponible en: https://tools.ietf.org/html/rfc7159 [Último acceso: Marzo 2019].

[38] **Wikipedia contributors.** *JavaScript wiki page*. Wikipedia. Disponible en: https://en.wikipedia.org/wiki/JavaScript [Último acceso: Marzo 2019].

[39] **Jan Biniok**. *Tampermonkey Plugin site*. Tampermonkey. Disponible en: https://tampermonkey.net [Último acceso: Abril 2020].

[40] **I. Román Martínez.** *Apuntes de Ingeniería del Software*.

[41] **Microsoft Corporation.** *NPM site*. NPM. Disponible en: https://www.npmjs.com [Último acceso: Marzo 2019].

[42] **NPM community.** *Package.json specifications*. NPM documentation. Disponible en: https://docs.npmjs.com/files/package.json [Último acceso: Marzo 2019].

[43] **The PHP Group.** *PHP site.* PHP. Disponible en: http://php.net [Último acceso: Marzo 2019].

[44] **The Apache Software Foundation.** *Apache Server Project*. Apache. Disponible en: https://httpd.apache.org [Último acceso: Marzo 2019].

[45] **Oracle Corporation.** *MySQL site*. MySQL. Disponible en: https://www.mysql.com [Último acceso: Marzo 2019].

[46] **VMware, Inc.** *LAMP Stack*. Bitnami. Disponible en: https://bitnami.com/stack/lamp [Último acceso: Marzo 2020].

[47] **VMware, Inc.** *Mean Stack*. Bitnami. Disponible en: https://bitnami.com/stack/mean [Último acceso: Marzo 2020].

[48] **Grav and Hugo.** *Lego documentation*. Lego. Disponible en: https://go-acme.github.io/lego [Último acceso: Enero 2020].

[49] **Internet Security Research Group (ISRG).** *Let's Encrypt. site.* Let's Encrypt. Disponible en: https://letsencrypt.org [Último acceso: Enero 2020].

[50] **Software Freedom Conservancy, Inc.** *Git version control system*. Git. Disponible en: https://git-scm.com [Último acceso: Marzo 2019].

[51] **Google LLC.** *AngularJS framework*. Angular. Disponible en: https://angularjs.org [Último acceso: Marzo 2019].

[52] **MongoDB, Inc.** *MongoDB database site.* MongoDB. Disponible en: https://www.mongodb.com [Último acceso: Marzo 2019].

[53] **The Linux man-pages project**. *systemd man page*. Linux man pages. Disponible en: http://man7.org/linux/man-pages/man1/init.1.html [Último acceso: Enero 2018].

[54] **J.-H. Haukeland, M. Pala, C. Hopp y R. Toma.** *Monit utility*. Linux man pages. Disponible en: https://linux.die.net/man/1/monit [Último acceso: Enero 2018]

[55] **J. Ariza Toledano.** *Best Practices writing a Dockefile*. Bitnami Engineering Blog. Disponible en: https://engineering.bitnami.com/articles/best-practices-writing-a-dockerfile.html [Último acceso: Abril 2020]

[56] **The OpenJS Foundation**. *Mocha test framework site*. Mocha. Disponible en: https://mochajs.org [Último acceso: Diciembre 2019]

[57] **S. Mollweide**. *Api mock server GitHub repository*. GitHub. Disponible en: https://github.com/smollweide/node-mock-server [Último acceso: Febrero 2018]

[58] **Google LLC**. *Jsonnet - The data templating language*. Jsonnet. Disponible en: https://jsonnet.org [Último acceso: Diciembre 2019]

[59] **CNCF**. *Kubernetes Official documentation*. Kubernetes. Disponible en: https://kubernetes.io/docs [Último acceso: Abril 2020]

[60] **NGINX, Inc.** *NGINX Ingress Controller for Kubernetes*. GitHub. Disponible en: https://github.com/kubernetes/ingress-nginx [Último acceso: Abril 2020]

[61] **O. Ben-Kiki, C. Evans y I. döt Net.** *YAML specification*. YAML . Disponible en: https://yaml.org/spec/1.2/spec.html [Último acceso: Marzo 2019]

[62] **CNCF.** *Horizontal Pod Autoscaler, Kubernetes documentation*. Kubernetes. Disponible en: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale [Último acceso: Abril 2020]

[63] **CNCF.** *ExternalDNS GitHub repository*. GitHub. Disponible en: https://github.com/kubernetes-sigs/external-dns [Último acceso: Abril 2020]

[64] **Google LLC.** *Cloud DNS.* Google Cloud. Disponible en: https://cloud.google.com/dns [Último acceso: Septiembre 2019]

[65] **Jetstack Ltd.** *cert-manager site*. Cert Manager. Disponible en: https://cert-manager.io. [Último acceso: Marzo 2019]

[66] **Wikipedia contributors.** *Base64 wiki page*. Wikipedia. Disponible en: https://es.wikipedia.org/wiki/Base64. [Último acceso: Marzo 2019]

[67] **VMware Inc.** *Configure RBAC in your Kubernetes cluster guide.* Bitanmi. Disponible en: https://docs.bitnami.com/kubernetes/how-to/configure-rbac-in-your-kubernetes-cluster. [Último acceso: Marzo 2019]

[68] **CNCF.** *Custom Resources, Kubernetes documentation*. Kubernetes Disponible en: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources. [Último acceso: Marzo 2019]

[69] **CNCF.** *Logging using Elasticsearch and Kafka, Kubernetes documentation*. Kubernetes. Disponible en: https://kubernetes.io/docs/tasks/debug-application-cluster/logging-elasticsearch-kibana. [Último acceso: Febrero 2018]

[70] **B. Brazil.** *The Classes of Container Monitoring.* The New Stack. Disponible en: https://thenewstack.io/classes-container-monitoring. [Último acceso: Marzo 2019]

[71] **Elasticsearch B.V.** *Elasticsearch* product site. Elasticsearch. Disponible en: https://www.elastic.co/products/elasticsearch. [Último acceso: Marzo 2019]

[72] **CNCF.** *Fluentd data collector*. Fluentd. Disponible en: https://www.fluentd.org. [Último acceso: Marzo 2019]

[73] **Elasticsearch B.V.** *Kibana* product site. Disponible en: https://www.elastic.co/kibana. [Último acceso: Marzo 2019]

[74] **The Linux Foundation®.** *Prometheus site*. Prometheus. Disponible en: https://prometheus.io. [Último acceso: Marzo 2019]

[75] **raintank Inc.** *Grafana site*. Grafana. Disponible en: https://grafana.com. [Último acceso: Marzo 2019]

[76] **CNCF**. *kubectl, Kubernetes documentation*. Kubernetes. Disponible en: https://kubernetes.io/docs/reference/kubectl/kubectl. [Último acceso: Abril 2020]

[77] **VMware, Inc.** *Bitnami's jsonnet library for building Kubernetes manifests*. GitHub. Disponible en: https://github.com/bitnami-labs/kube-libsonnet. [Último acceso: Enero 2020]

[78] **Wikipedia contributors.** *Net Promoter Score wiki page.* Wikipedia. Disponible en: https://en.wikipedia.org/wiki/Net_Promoter. [Último acceso: Marzo 2019]

[79] **Redash Company.** *Redash site*. Redash. Disponible en: https://redash.io. [Último acceso: Marzo 2019]

[80] **Amazon Web Services. Inc.** *Amazon Relational Database Service.* Amazon Web Services. Disponible en: https://aws.amazon.com/es/rds. [Último acceso: Mayo 2020]

[81] **Wikipedia contributors.** *Acuerdo de Nivel de Servicio*. Wikipedia. Disponible en: https://es.wikipedia.org/wiki/Acuerdo_de_nivel_de_servicio. [Último acceso: Marzo 2019]

[82] **HashiCorp**. *Vault for Kubernetes documentation*. Vault Project. Disponible en: https://www.vaultproject.io/docs/platform/k8s. [Último acceso: Marzo 2019]

[83] **Stackalytics**. *K8s Contribution by Companies*. Stackalytics. Disponible en: http://stackalytics.com/?project_type=kubernetes-group&metric=commits. [Último acceso: Marzo 2019]

GLOSARIO

API: Interfaz de Programación de Aplicaciones (del inglés: Application Programming Interface).

AWS: Plataforma de Servicios en nube de Amazon (del inglés: Amazon Web Services).

CI/CD: Integración Continua/Despliegue Continuo (del inglés: Continuous Integration/Continous Delivery).

ConfigMap: Recurso de Kubernetes que presenta una serie de datos de configuración a disposición de las aplicaciones.

Contenedor: Tecnología similar a las máquinas virtuales que, en lugar de virtualizar la infraestructura completa de computación, utilizan el sistema operativo de su host.

Deployment: Recurso de Kubernetes. Representa un conjunto de pods añadiendo información sobre su replicación y control de versiones.

DNS: Sistema de nombres de dominio (del inglés: Domain Name System).

GCP: Plataforma de Servicios en nube de Google (del inglés: Google Cloud Platform).

GKE: Servicio de GCP que ofrece el despliegue/administración de Kubernetes clústeres en nube (del inglés: Google Kubernetes Engine).

Horizontal Pod Autoscaler: Recueros de Kubernetes que permite definir reglas para aumentar o disminuir el número de réplicas de un determinado tipo de pod.

HTTP: Protocolo de comunicación web (del inglés: HyperText Transport Protocol).

HTTPS: Versión segura del protocolo de comunicación web HTTP.

Ingress: Recurso de Kubernetes. Representa una regla de redirección o proxy.

K8s: Abreviatura de Kubernetes.

POD: Recurso de Kubernetes. Representa un grupo de uno o más contenedores que comparten recursos de red y almacenamiento.

PV: Recurso de Kubernetes. Representa un volumen persistente (del inglés: Persitent Volume).

ReplicaSet: Recurso de Kubernetes. Representa un grupo de pods.

Secret: Recurso de Kubernetes que representa una serie de datos de información sensible.

SS: Recurso de Kubernetes (do inglés: Sealed Secret).

SVC: Recurso de Kubernetes. Representa un servicio. (del inglés: Service).

TLS: Protocolo de seguridad de la capa de Transporte (del inglés: Transport Layer Security).

Webhook: Petición HTTP asociado a un determinado evento.