Google Cloud

Rediseña la arquitectura para una innovación sin límites

Aumenta la productividad y la escalabilidad, y reduce los costos para entornos nativos de la nube.

Índice

Capítulo

01 Arquitectura nativa de la nube

Capítulo

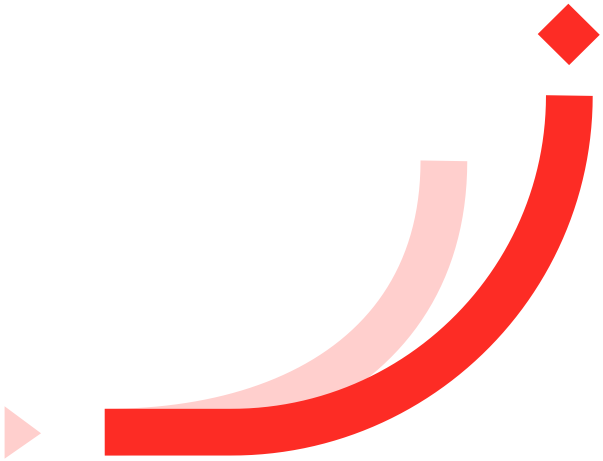
02 Principios y prácticas

Google Cloud

Índice

Capítulo

03 Comenzar



Capítulo

01 Arquitectura nativa de la nube

¿Por qué migrar a una arquitectura nativa de la nube?

Muchas empresas compilan sus servicios de software con arquitecturas monolíticas. Este enfoque tiene la ventaja de que los sistemas monolíticos son relativamente sencillos de implementar y diseñar (al menos al comienzo). Sin embargo, puede ser difícil mantener la productividad de los desarrolladores y la velocidad de implementación dado que, a medida que las aplicaciones crecen, se vuelven más complejas, lo que da como resultado sistemas costosos y lentos de cambiar, y riesgosos de implementar. En este artículo, encontrarás información para rediseñar tus aplicaciones a un paradigma nativo de la nube que te permitirá acelerar la entrega de funciones nuevas, incluso mientras haces crecer tus equipos, además de mejorar la calidad del software y alcanzar niveles más altos de estabilidad y disponibilidad.

A medida que crecen los servicios (y los equipos responsables de administrarlos), se vuelven más complejos y es más difícil evolucionarlos y operarlos. La implementación y las pruebas se tornan más complicadas, agregar funciones nuevas resulta más difícil, y mantener la confiabilidad y disponibilidad se vuelve todo un desafío.

Una investigación del equipo de DORA de Google demostró que es posible alcanzar altos niveles de capacidad de procesamiento de entrega de software, así como de estabilidad y disponibilidad del servicio en todas las organizaciones de todos los tamaños y dominios. Los equipos de alto rendimiento pueden realizar implementaciones varias veces al día, hacer cambios en la producción en menos de un día, restablecer el servicio en menos de una hora y conseguir tasas de fallos de cambios de entre un 0 y un 15%[1].

Además, las organizaciones de alto rendimiento pueden alcanzar niveles más altos de productividad de desarrolladores, que se mide en términos de implementaciones por desarrollador por día, incluso mientras aumentan el tamaño de sus equipos. Esto se muestra en la Figura 1.

[1] Conozca el rendimiento de su equipo según estas cuatro métricas clave en https://cloud.google.com/devops/

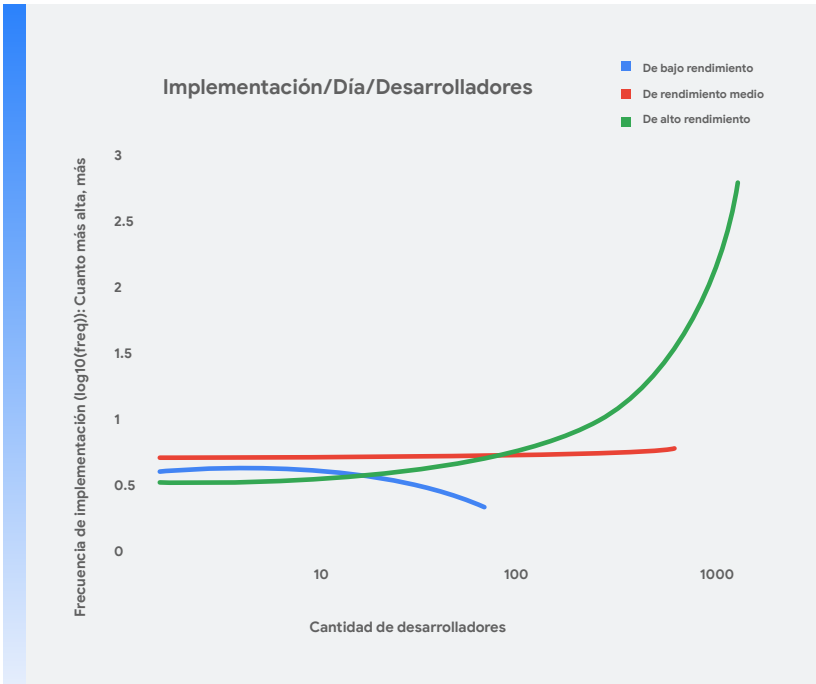


Figura 1: Impacto del rendimiento de la entrega de software de un equipo en su capacidad para escalar la productividad de los desarrolladores (Informe del estado de DevOps 2015)[2].

En el resto de este documento, encontrará información para migrar sus aplicaciones a un paradigma moderno nativo de la nube a fin de conseguir estos resultados. Si implementa las prácticas técnicas, podrá alcanzar los siguientes objetivos:

- **Aumento de la productividad de los desarrolladores**, incluso mientras hace crecer a su equipo.
- **Tiempo más rápido de salida al mercado**: Agregue nuevas funciones y corrija defectos con mayor rapidez.
- **Mayor disponibilidad**: Aumente el tiempo de actividad del software, reduzca la tasa de errores de implementación y disminuya el tiempo de restablecimiento en caso de que ocurran incidentes.
- **Mayor seguridad**: Reduzca la superficie de exposición a ataques de sus aplicaciones y facilite la detección y respuesta rápidas ante ataques y vulnerabilidades descubiertas recientemente.
- **Mejor escalabilidad**: Las aplicaciones y plataformas nativas de la nube facilitan el escalamiento horizontal cuando es necesario y, además, permiten reducir la escala verticalmente.
- **Reducción de costos**: Un proceso optimizado de entrega de software reduce los costos de entrega de funciones nuevas, y el uso eficaz de las plataformas en la nube reduce sustancialmente los costos operativos de sus servicios. 2015

[2] https://services.google.com/fh/files/misc/state-of-devops_2015.pdf

¿Qué es una arquitectura nativa de la nube?

Las aplicaciones monolíticas deben compilarse, probarse y, luego, implementarse como una única unidad. A menudo, el sistema operativo, el middleware y la pila de lenguaje de la aplicación se personalizan o se actualiza su configuración para cada aplicación. La compilación, la prueba y la implementación de secuencias de comandos y procesos también suelen ser únicas para cada aplicación. Esto resulta sencillo y eficaz para las aplicaciones nuevas; sin embargo, a medida que crecen, resulta más difícil probar, implementar y operar tales sistemas, así como implementar cambios en ellos.

Además, a medida que los sistemas crecen, los equipos que compilan, prueban, implementan y operan el servicio se vuelven más grandes y complejos. Un enfoque común, pero erróneo, es dividir a los equipos por función. Esto da como resultado transferencias entre los equipos, lo que suele incrementar los plazos de entrega y los tamaños de los lotes, y generar una cantidad significativa de trabajo repetido. En la investigación de DORA, se señala que los equipos de alto rendimiento tienen el doble de probabilidades de desarrollar y entregar software con un único equipo multifuncional.

Estos son algunos indicadores de este problema:

- Procesos de compilación extensos y, con frecuencia, incorrectos
- Ciclos de integración y de prueba poco frecuentes
- Mayor esfuerzo para admitir los procesos de prueba y compilación
- Pérdida de productividad de los desarrolladores
- Procesos de implementación difíciles que deben realizarse durante horas y que requieren de tiempo de inactividad programado
- Esfuerzo significativo para administrar la configuración de los entornos de prueba y producción

Por el contrario, el paradigma nativo de la nube representa los siguientes beneficios[3]:

- Los sistemas complejos se descomponen en servicios que pueden probarse y, luego, implementarse de forma independiente en un entorno de ejecución alojado en contenedores (una arquitectura orientada a servicios o de microservicios).
- Las aplicaciones usan servicios estándar proporcionados por la plataforma, como los sistemas de administración de bases de datos (DBMS), el almacenamiento de BLOB, la mensajería, la CDN y la terminación SSL.
- Una plataforma estandarizada en la nube se ocupa de numerosos asuntos operativos, como la implementación, el ajuste de escala automático, la configuración, la administración de secretos, la supervisión y las alertas. Los equipos de desarrollo de aplicaciones pueden acceder a estos servicios según la demanda.
- Se proporciona el sistema operativo estandarizado, el middleware y las pilas específicas de lenguaje a los desarrolladores de aplicaciones, y el proveedor de plataformas o un equipo independiente realiza el mantenimiento y la implementación de parches de estas pilas fuera de banda.
- Un único equipo multifuncional puede ser responsable de todo el ciclo de vida de la entrega de software de cada servicio.

[3] Esta información no pretende ser una descripción completa del paradigma nativo de la nube. Para conocer más sobre algunos de los principios de la arquitectura nativa de la nube, visite https://cloud.google.com/blog/products/ application-development/5-principles-for-cloud-native-architecture-what-it-is-and-how-to-master-it.

Este paradigma proporciona numerosos beneficios:

- **Entrega más rápida:** Dado que los servicios ahora son pequeños con acoplamiento bajo, los equipos asociados a estos servicios pueden funcionar de forma autónoma. Esto aumenta la productividad de los desarrolladores y la velocidad de desarrollo.
- **Actualizaciones confiables:** Los desarrolladores pueden compilar, probar y, luego, implementar servicios nuevos y existentes con rapidez en entornos de pruebas como los de producción. La implementación en el entorno de producción también es una actividad atómica sencilla. Esto acelera sustancialmente el proceso de entrega de software y reduce los riesgos de las implementaciones.
- **Costos más bajos:** El costo y la complejidad de los entornos de prueba y producción se reducen significativamente debido a que la plataforma proporciona servicios estandarizados compartidos y porque las aplicaciones se ejecutan en una infraestructura física compartida.
- **Mejor seguridad:** Los proveedores ahora son responsables de mantener los servicios compartidos, como los DBMS y la infraestructura de mensajería, actualizados, con parches y en cumplimiento. Además, es más sencillo mantener las aplicaciones actualizadas y con parches, ya que existe un método estándar para implementar y administrar aplicaciones.
- **Mayor disponibilidad:** La disponibilidad y confiabilidad de las aplicaciones aumenta gracias a la complejidad reducida del entorno operativo, la facilidad para hacer cambios en la configuración y la posibilidad de controlar el ajuste de escala automático y la reparación automática en el nivel de la plataforma.
- **Cumplimiento más sencillo y más económico:** Se puede implementar la mayoría de los controles de seguridad de la información en la capa de la plataforma, lo que hace que sea mucho más económico y sencillo implementar y demostrar el cumplimiento. Muchos proveedores de servicios en la nube mantienen el cumplimiento con frameworks de administración de riesgos como SOC2 y FedRAMP, lo que significa que las aplicaciones implementadas en ellos solo deben demostrar el cumplimiento con controles residuales no implementados en la capa de la plataforma.

Sin embargo, existen algunas desventajas asociadas al modelo nativo de la nube:

- Todas las aplicaciones ahora son sistemas distribuidos, lo que significa que realizan una cantidad significativa de llamadas remotas como parte de su operación. Esto implica analizar detenidamente cómo controlar las fallas de red y los problemas de rendimiento, y cómo depurar problemas en la producción.
- Los desarrolladores deben usar el sistema operativo estandarizado, el middleware y las pilas de aplicación que ofrece la plataforma. Esto dificulta el desarrollo local.
- Los arquitectos deben adoptar un enfoque controlado por eventos para el diseño de los sistemas, incluida la adopción de coherencia eventual.

Migre a un entorno nativo de la nube

Muchas organizaciones adoptaron un enfoque “lift-and-shift” para migrar servicios a la nube. En este enfoque, solo se requieren cambios menores en los sistemas; y aunque, básicamente, la nube se trata como un centro de datos tradicional, proporciona API, servicios y herramientas de administración mucho mejores, en comparación con los centros de datos tradicionales. Sin embargo, el enfoque lift-and-shift en sí mismo no proporciona ninguno de los beneficios que ofrece el paradigma nativo de la nube, señalados anteriormente.

Muchas organizaciones se estancan en el enfoque lift-and-shift debido a los gastos y la complejidad de la migración de las aplicaciones a una arquitectura nativa de la nube, que requiere un replanteamiento total, desde la arquitectura de la aplicación hasta las operaciones de producción y, por supuesto, todo el ciclo de vida de la entrega de software.

Este miedo es racional, ya que muchas organizaciones se han visto perjudicadas producto de años de esfuerzos fallidos por rediseñar completamente la plataforma.

La solución es implementar un enfoque progresivo, iterativo y evolutivo para rediseñar sus sistemas en un entorno nativo de la nube, lo que permitirá a los equipos aprender a trabajar eficientemente en este nuevo paradigma y seguir brindando nuevas funciones. A este enfoque llamamos “migrar y mejorar”.

Un patrón clave en la arquitectura evolucionaria es conocido como la **aplicación estranguladora**[4]. En lugar de reescribir complementamente los sistemas desde cero, escriba funciones nuevas con un estilo nativo moderno de la nube, y permítales comunicarse con la aplicación monolítica original para la funcionalidad existente. Cambie la funcionalidad existente a lo largo del tiempo de forma progresiva y según sea necesario para la integridad conceptual de los nuevos servicios, tal como se muestra en la Figura 2.

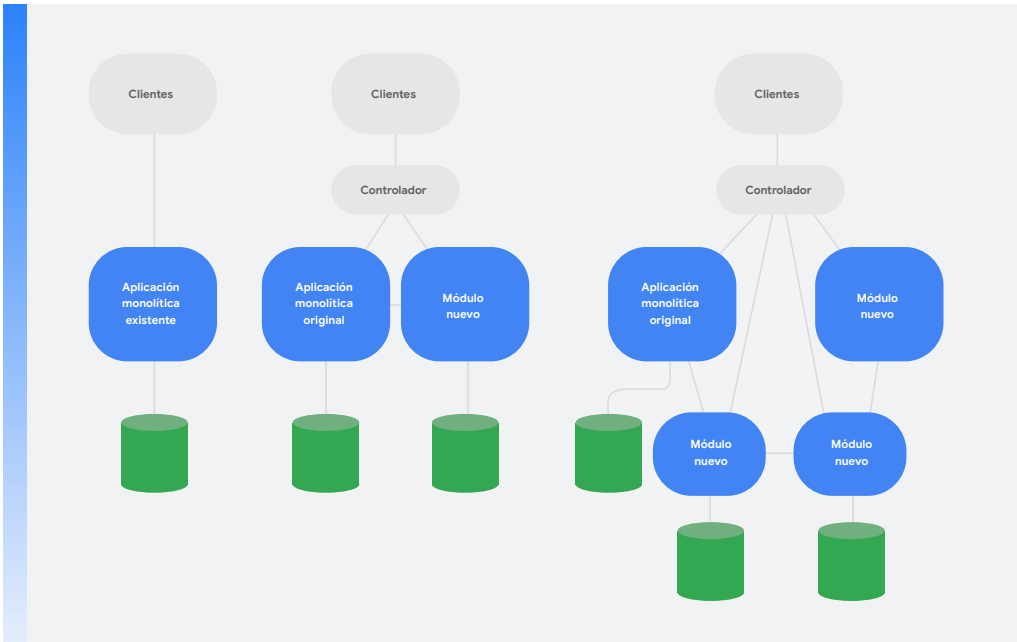


Figura 2: Uso del patrón de aplicación estranguladora para rediseñar de forma progresiva la aplicación monolítica.

[4] Se describe en https://martinfowler.com/bliki/StranglerFigApplication.html

Estos son tres lineamientos importantes para rediseñar la arquitectura de manera exitosa:

En primer lugar, **brinde una funcionalidad nueva con rapidez,** en lugar de reproducir la funcionalidad existente. La métrica clave consiste en qué tan rápido puede comenzar a brindar funcionalidades nuevas usando servicios nuevos, de modo que pueda aprender y comunicar con rapidez las prácticas obtenidas de su trabajo con este paradigma. Reduzca drásticamente el alcance a fin de entregar funcionalidades a usuarios reales en cuestión de semanas, no meses.

En segundo lugar, **diseñe para el entorno nativo de la nube**. Esto significa usar los servicios nativos de la plataforma en la nube para los DBMS, la mensajería, la CDN, las redes, el almacenamiento de BLOB y mucho más, y usar las pilas de aplicaciones estandarizadas proporcionadas por la plataforma cada vez que sea posible. Los servicios deben estar alojados en contenedores y usar el paradigma sin servidores siempre que sea posible, y los procesos de compilación, prueba y, también, implementación deben estar totalmente automatizados. Haga que todas las aplicaciones usen los servicios compartidos proporcionados por la plataforma para el registro, la supervisión y las alertas (vale la pena mencionar que puede resultar útil implementar este tipo de arquitectura de plataforma para cualquier plataforma de aplicación de multiusuario, incluido el entorno local de equipos físicos). En la Figura 3 a continuación, se muestra una imagen de alto nivel de una plataforma nativa de la nube.

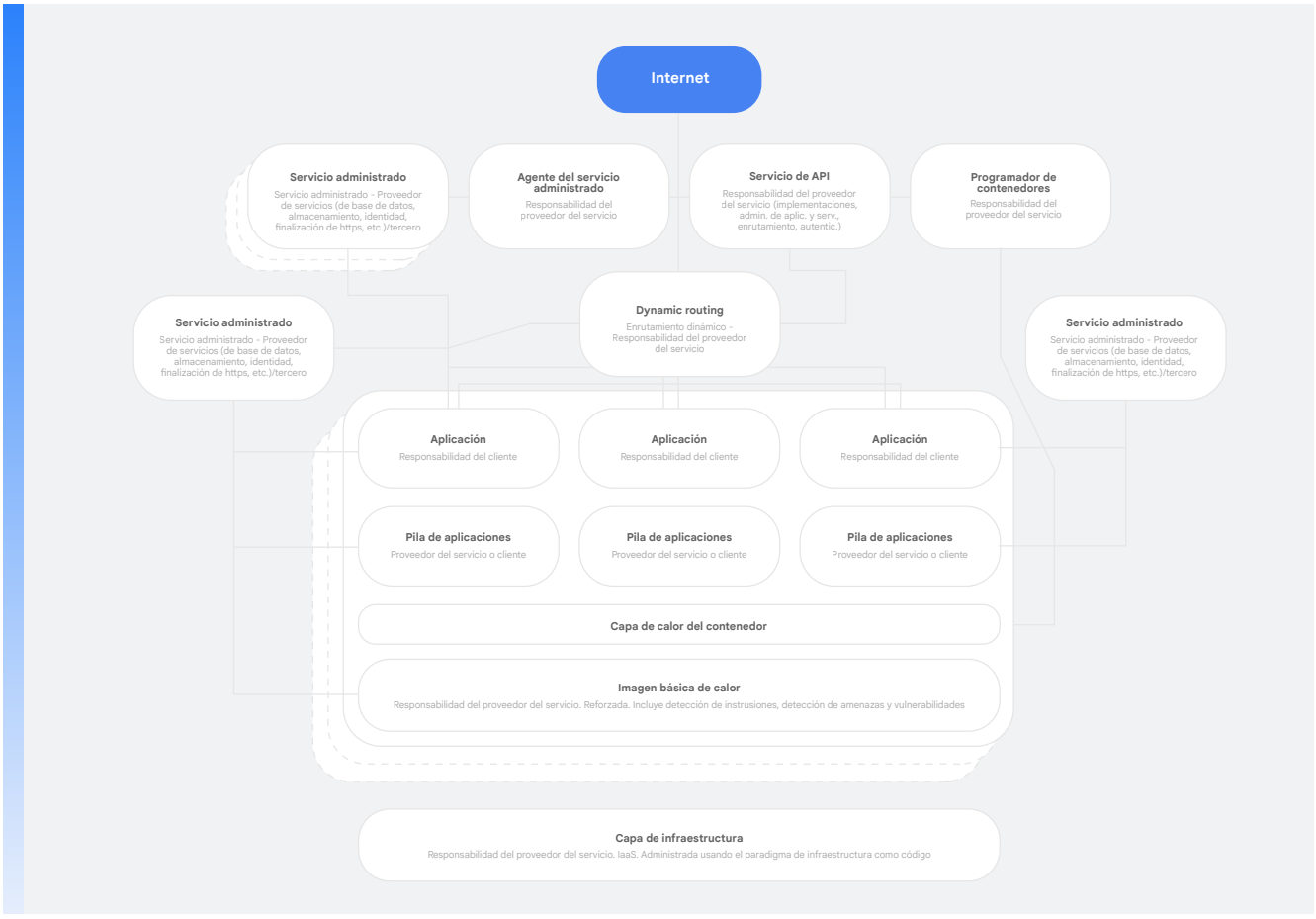


Figura 3: Anatomía de alto nivel de una plataforma en la nube

Por último, **diseñe para equipos autónomos con acoplamiento bajo que puedan probar y, también, implementar sus propios servicios**. Tal como se señala en nuestra investigación, los resultados más importantes en torno a la arquitectura se obtienen cuando los equipos de entrega de software pueden responder afirmativamente estas seis preguntas:

- ¿Podemos hacer cambios a gran escala en el diseño de nuestro sistema sin necesitar el permiso de nadie fuera del equipo?
- ¿Podemos hacer cambios a gran escala en el diseño de nuestro sistema sin depender de los cambios en los sistemas de otros equipos ni tener que crear un trabajo significativo para otros equipos?
- ¿Podemos completar nuestro trabajo sin comunicarnos ni coordinarnos con personas fuera del equipo?
- ¿Podemos implementar y lanzar nuestro producto o servicio según la demanda, independientemente de que dicho producto o servicio dependa de otros servicios?
- ¿Podemos realizar la mayor parte del proceso de prueba según la demanda, sin necesitar un entorno de pruebas integrado?
- ¿Podemos realizar implementaciones durante el horario laboral con un tiempo de inactividad mínimo?

Compruebe regularmente si los equipos trabajan en función de estos objetivos y si alcanzarlos es su prioridad. Esto suele implicar rediseñar tanto la arquitectura organizativa como la empresarial.

Más específicamente, es esencial organizar equipos de modo que todas las personas necesarias para compilar, probar y, también, implementar software, incluidos los gerentes de productos, colaboren y usen prácticas modernas de administración de productos para compilar y evolucionar los servicios en los que trabajan.
Esto no implica cambios en la estructura organizativa. El simple hecho de que trabajen en equipo diariamente (y que compartan un espacio físico, siempre que sea posible), en lugar de tener desarrolladores, verificadores y equipos de lanzamiento operando de forma independiente, hace una gran diferencia en el área de la productividad.

Según nuestra investigación, se puede predecir que los equipos que respondieron afirmativamente estas preguntas tienen un alto rendimiento del software, es decir, tienen la capacidad de entregar servicios confiables y con alta disponibilidad múltiples veces al día. Esto, a su vez, permite que los equipos de alto rendimiento aumenten la productividad de los desarrolladores (medida en términos de cantidad de implementaciones por desarrollador por día), incluso mientras aumenta la cantidad de equipos.

Capítulo

02 Principios y prácticas

Principios y prácticas para la arquitectura de microservicios

Cuando adopta una arquitectura orientada a los servicios o de microservicios, debe asegurarse de seguir algunos principios y prácticas importantes. Lo ideal es ser sumamente estrictos al respecto desde el comienzo, ya que resulta más costoso realizar actualizaciones posteriores.

- **Cada servicio debe tener su propio esquema de bases de datos.** Ya sea que use una base de datos relacional o una solución NoSQL, cada servicio debe tener su propio esquema al que no pueda acceder ningún otro servicio. Cuando múltiples servicios se comunican con el mismo esquema, con el tiempo, se vuelven altamente acoplados en la capa de base de datos. Estas dependencias impiden que los servicios se prueben y se implementen de forma independiente, lo que hace que sea más difícil cambiarlos y más riesgoso implementarlos.
- **Los servicios solo se deben comunicar a través de sus API públicas en la red**. Todos los servicios deben exponer su comportamiento a través de las API públicas, y solo deben hablar entre sí mediante estas API. No debe existir una "puerta trasera" de acceso, y los servicios no deben comunicarse directamente con las bases de datos de otros servicios. Esto evita que los servicios se vuelvan altamente acoplados y garantiza que la comunicación entre servicios sea con API compatibles y bien documentadas.
- **Los servicios son responsables de la retrocompatibilidad para sus clientes.** El equipo que compila y opera el servicio es responsable de garantizar que las actualizaciones a sus servicios no afecten negativamente a sus consumidores. Esto implica planificar los controles de versiones de la API y probar la retrocompatibilidad para que, cuando lance versiones nuevas, los clientes existentes no experimenten interrupciones del servicio. Los equipos pueden validar esto con el lanzamiento Canary. Esto también implica garantizar que las implementaciones no generen tiempo de inactividad mediante técnicas como las implementaciones azul-verde o los lanzamientos por etapas.

- **Cree una forma estándar de ejecutar servicios en estaciones de trabajo de desarrollo.** Los desarrolladores deben poder crear según la demanda cualquier subconjunto de servicios de producción en estaciones de trabajo de desarrollo usando un único comando. También debe ser posible ejecutar según la demanda versiones stub de servicios (asegúrese de usar versiones emuladas de los servicios en la nube. Existen numerosos proveedores de servicios en la nube que las ofrecen para ayudarlo). El objetivo es hacer que las pruebas y depuraciones de servicios de forma local sean más sencillas para los desarrolladores.
- **Invierta en supervisión y observabilidad del entorno de producción.** Muchos de los problemas en el entorno de producción, incluidos aquellos relacionados con el rendimiento, surgen de interacciones entre múltiples servicios. Nuestra investigación demuestra que es importante tener implementada una solución que dé cuenta del estado general de los sistemas (por ejemplo, si están funcionando o si tienen suficientes recursos disponibles), y que los equipos tengan acceso a herramientas y datos que los ayuden a comprender y diagnosticar problemas de infraestructura en los entornos de producción, incluidas las interacciones entre servicios, y a hacer un seguimiento de ellos.
- **Defina objetivos de nivel de servicio (SLO) para sus servicios y realice pruebas de recuperación ante desastres con regularidad.** Establecer SLO genera expectativas con respecto al rendimiento de sus servicios y lo ayuda a planificar el comportamiento de su sistema en caso de que un servicio deje de funcionar (una consideración clave a la hora de compilar sistemas distribuidos resilientes). Pruebe cómo se comporta su sistema de producción en la vida real usando técnicas como la inyección controlada de errores como parte de su plan de prueba de recuperación ante desastres (la investigación de DORA demuestra que las organizaciones que llevan a cabo pruebas de recuperación ante desastres con métodos como este tienen más probabilidades de tener altos niveles de disponibilidad del servicio). Mientras más pronto comience con este proceso, mejor, ya que podrá normalizar este tipo de actividad vital.

Son muchos aspectos que debe considerar, es por eso que es importante probar este tipo de trabajo con un equipo que tenga la capacidad de experimentar con la implementación de estas ideas. Habrá éxitos y fracasos, pero lo importante es aprender de ellos y aprovecharlos a medida que expande el paradigma de arquitectura nuevo en toda su organización.

Nuestra investigación demuestra que las empresas que tienen éxito utilizan pruebas de concepto y ofrecen a los equipos la oportunidad de compartir aprendizajes, por ejemplo, creando comunidades de práctica. Proporcione tiempo, espacio y recursos a miembros de diferentes equipos para que se reúnan con regularidad y puedan intercambiar ideas. Todos también necesitarán aprender nuevas habilidades y tecnologías. Invierta en el crecimiento de los equipos. Para ello, bríndeles presupuesto para la compra de libros, ofrézcales cursos de capacitación y permítales asistir a conferencias. Además, proporcione infraestructura y tiempo para que difundan conocimiento institucional y prácticas recomendadas a través de listas de distribución de la empresa, bases de conocimiento y reuniones presenciales.

Arquitectura de referencia

En esta sección, describiremos una arquitectura de referencia basada en los siguientes lineamientos:

- Uso de contenedores para servicios de producción y de un programador de contenedores como Cloud Run o Kubernetes para la orquestación
- Creación de canalizaciones eficaces de CI/CD
- Enfoque en la seguridad

Aloje servicios de producción en contenedores

La base de una aplicación alojada en contenedores en la nube consiste en una administración de contenedores y en un servicio de organización. Si bien hasta ahora se han creado numerosos servicios diferentes, hay uno que sobresale claramente de los demás: Kubernetes. De acuerdo con Gartner, “Kubernetes emergió como el estándar de facto para la organización de contenedores, con una comunidad animada y el apoyo de la mayoría de los principales proveedores comerciales”. En la Figura 4, se resume la estructura lógica de un clúster de Kubernetes.

Kubernetes define una abstracción denominada Pod. Por lo general, los Pods incluyen solo un contenedor (como los Pods A y B en la Figura 4), pero podrían contener más de uno (como en el caso del Pod C). Cada servicio de Kubernetes ejecuta un clúster que contiene una cantidad determinada de nodos, y cada uno de ellos suele ser una máquina virtual (VM). En la Figura 4 solo se muestran cuatro VM, pero un clúster real puede contener fácilmente cientos de ellos o incluso más. Cuando se implementa un Pod en un clúster de Kubernetes, el servicio determina en cuáles VM deben ejecutarse los contenedores de ese Pod. Dado que los mismos contenedores especifican los recursos que necesitan, Kubernetes puede tomar decisiones inteligentes sobre qué Pods se asignan a cada VM.

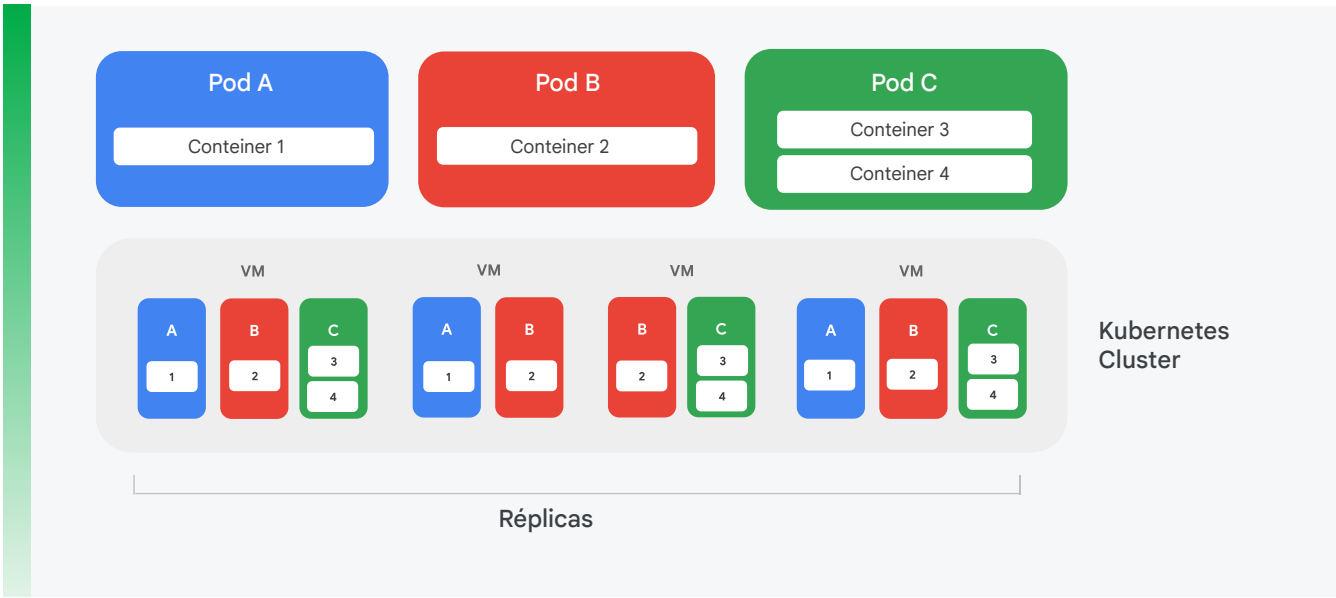


Figura 4: Un clúster de Kubernetes ejecuta cargas de trabajo, y cada Pod posee uno o más contenedores.

Parte de la información de implementación de un Pod indica cuántas instancias (réplicas) del Pod deben ejecutarse. Luego, el servicio de Kubernetes crea esa cantidad de instancias de los contenedores del Pod y las asigna a diferentes VM. En la Figura 4, por ejemplo, la implementación del Pod A solicitó tres réplicas, al igual que la implementación del Pod C. Sin embargo, la implementación del Pod B solicitó cuatro réplicas, es por eso que este clúster de ejemplo contiene en ejecución cuatro instancias del contenedor 2. Además, tal como se sugiere en la figura, un Pod con más de un contenedor, como el Pod C, siempre tendrá sus contenedores asignados al mismo nodo.

Kubernetes también proporciona otros servicios, como los que se indican a continuación:

- **Supervisión de Pods en ejecución**, de modo que si un contenedor falla, el servicio comenzará una instancia nueva. Esto garantiza que todas las réplicas solicitadas en la implementación de un Pod permanezcan disponibles.
- **Tráfico de balanceo de cargas**, con una divulgación inteligente de las solicitudes realizadas a cada Pod en las réplicas de un contenedor.
- **Lanzamiento automático sin tiempo de inactividad de contenedores nuevos**, con instancias nuevas que reemplazan de forma progresiva las instancias existentes hasta que se implementa totalmente una versión nueva.
- **Escalamiento automático**, con un clúster que agrega o borra VM de forma autónoma según la demanda.

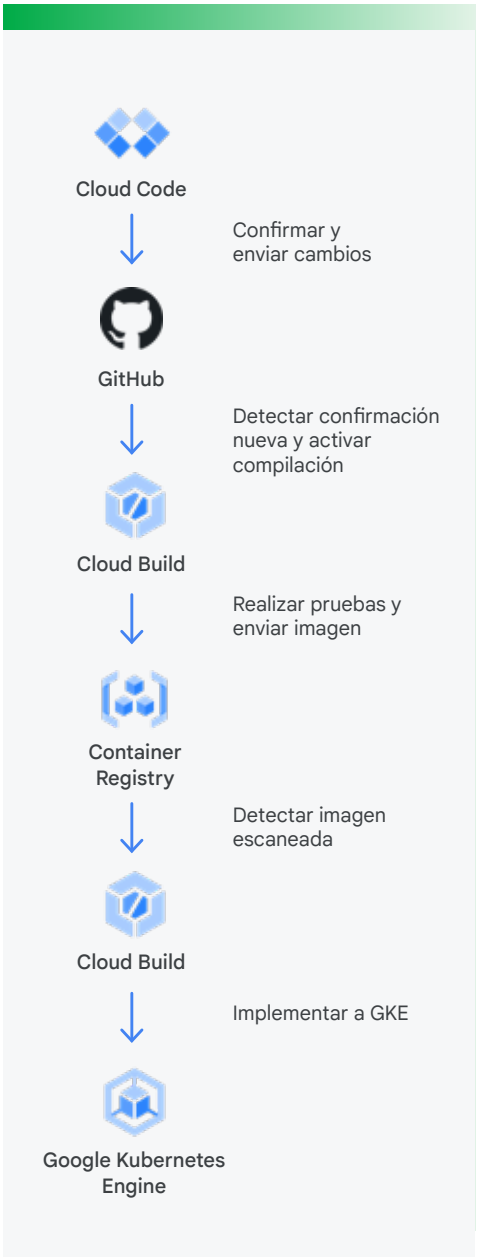


Figura 5: Una canalización de CI/CD consta de numerosos pasos, desde la escritura de código hasta la implementación de un contenedor nuevo.

Cree canalizaciones eficaces de CI/CD

Algunos de los beneficios de refactorizar una aplicación monolítica, como los costos inferiores, provienen directamente de la ejecución en Kubernetes. Sin embargo, uno de los beneficios más importantes, la habilidad de actualizar su aplicación con mayor frecuencia, solo es posible si cambia la forma en que compila y lanza software. Para obtener este beneficio, debe crear canalizaciones de CI/CD eficaces en su organización.

La integración continua (CI) se basa en la compilación y prueba automáticas de flujos de trabajo que proporcionan comentarios rápidos a los desarrolladores. Esto requiere que cada miembro de un equipo que trabaja en el mismo código (por ejemplo, el código para un único servicio) integre regularmente su trabajo en una línea principal o un enlace troncal compartidos. Esta integración debe suceder al menos una vez al día por desarrollador, y cada integración debe someterse a una verificación mediante un proceso de compilación que incluye pruebas automáticas. La entrega continua (CD) busca que la implementación de este código de integración sea rápida y menos riesgosa a través de la automatización del proceso de compilación, prueba y, luego, implementación, de modo que las actividades como el rendimiento, la seguridad y las pruebas exploratorias puedan realizarse de forma continua. En pocas palabras, la CI ayuda a los desarrolladores a detectar problemas en la integración con rapidez, mientras que la CD permite que la implementación sea confiable y rutinaria.

Es útil analizar un ejemplo en concreto para entender con mayor claridad este asunto. En la Figura 5, se muestra cómo sería una canalización de CI/CD que usa las herramientas de Google para los contenedores que se ejecutan en Google Kubernetes Engine.

Es útil pensar en el proceso como dos fragmentos, tal como se muestra en la Figura 6:

Desarrollo local: El objetivo aquí es acelerar un bucle de desarrollo interno y proporcionar a los desarrolladores herramientas para que obtengan comentarios rápidos relativos al impacto de los cambios de código locales. Esto incluye la compatibilidad con los análisis con lint, el autocompletado de YAML y compilaciones locales más rápidas.

Desarrollo remoto: Cuando se envía una solicitud de extracción (PR), se inicia el bucle de desarrollo remoto. El objetivo aquí es reducir drásticamente el tiempo que toma validar y probar la PR mediante la CI y realizar otras actividades, como el análisis de vulnerabilidades y la firma binaria, mientras se impulsan aprobaciones de versiones de manera automatizada.

Figura 6: Bucles de desarrollo local y remoto

Esta es la ayuda que pueden brindar las herramientas de Google Cloud durante este proceso:

Desarrollo local: Es esencial hacer que los desarrolladores sean productivos mediante el desarrollo local de aplicaciones. Este desarrollo local implica la compilación de aplicaciones que pueden implementarse en clústeres locales y remotos. Antes de realizar cambios en un sistema de administración de control de origen como GitHub, contar con un bucle de desarrollo local rápido puede garantizar que los desarrolladores prueben y, luego, implementen sus cambios en un clúster local.

Como resultado, Google Cloud proporciona Cloud Code. Cloud Code viene con extensiones a IDE, como IntelliJ y Visual Studio Code, para permitir a los desarrolladores iterar, depurar y ejecutar código con rapidez en Kubernetes. Además, usa herramientas populares, como Skaffold, Jib y Kubectl, para permitir a los desarrolladores obtener comentarios continuos sobre el código en tiempo real.

Integración continua: Con la nueva app de GitHub de Cloud Build, los equipos pueden activar compilaciones en diferentes eventos de repositorios, como eventos relacionados con solicitudes de extracción, ramas o etiquetas, directamente desde GitHub. Cloud Build es una plataforma sin servidores que aumenta o disminuye la escala verticalmente en función de la carga, y no requiere que se aprovisionen los servidores con anterioridad ni se realicen pagos por adelantado para obtener capacidad adicional. Las compilaciones activadas mediante la app de GitHub publican automáticamente el estado en GitHub. Los comentarios se integran directamente en el flujo de trabajo del desarrollador de GitHub, lo que reduce los cambios de contextos.

Administración de artefactos: Con Container Registry, su equipo puede administrar las imágenes de Docker, llevar a cabo análisis de vulnerabilidades y decidir quién accede a qué recursos con un control de acceso preciso, todo en un solo lugar. La integración del análisis de vulnerabilidades en Cloud Build permite a los desarrolladores identificar amenazas a la seguridad en cuanto Cloud Build crea una imagen y la almacena en Container Registry.

Entrega continua: Cloud Build usa pasos de compilación para que pueda definir pasos específicos que se desarrollarán como parte de los procesos de compilación, prueba y, también, implementación. Por ejemplo, una vez que se crea un contenedor nuevo y se envía a Container Registry, un paso de compilación posterior puede implementar ese contenedor en Google Kubernetes Engine (GKE) o Cloud Run, junto con la configuración y la política relacionadas. También puede implementarlo en otros proveedores de servicios en la nube si lo que desea es una estrategia de múltiples nubes. Por último, si lo que busca es una entrega continua tipo GitOps, Cloud Build le permite describir sus implementaciones de forma declarativa usando archivos (por ejemplo, manifiestos de Kubernetes) almacenados en un repositorio de Git.

No obstante, la implementación de código es solo una parte del proceso. Las organizaciones también deben administrar ese código mientras lo ejecutan. Para hacerlo, GCP les proporciona a los equipos de operaciones herramientas como Cloud Monitoring y Cloud Logging.

Ciertamente, no es obligatorio usar las herramientas de CI/CD de Google con GKE, por lo que es libre de usar cadenas de herramientas alternativas, si así lo desea; esto incluye Jenkins para CI/CD o Artifactory para la administración de artefactos.

Si, al igual que la mayoría de las organizaciones, tiene aplicaciones en la nube basadas en VM, es probable que actualmente no tenga un sistema de CI/CD bien estructurado. Es esencial tener uno para obtener los beneficios de una aplicación rediseñada, pero requiere trabajo. Las tecnologías necesarias para crear sus canalizaciones están disponibles, en parte, debido a la madurez de Kubernetes. Sin embargo, los cambios humanos pueden ser considerables. Los miembros de sus equipos de entrega deben volverse multifuncionales, con habilidades relativas al desarrollo, las pruebas y las operaciones. Cambiar la cultura toma tiempo, por lo que debe estar dispuesto a dedicar sus esfuerzos a cambiar el conocimiento y el comportamiento de los miembros a medida que avanzan hacia un sistema de CI/CD.

Enfóquese en la seguridad

Rediseñar aplicaciones monolíticas a un paradigma nativo de la nube es un gran desafío, y, como es de esperarse, hacerlo implica nuevos desafíos de seguridad que deberá abordar. Estos son dos de los más importantes:

- Proteger el acceso entre contenedores
- Garantizar una cadena de suministro de software segura

El primero de estos desafíos surge de un hecho evidente: para dividir su aplicación en servicios alojados en contenedores (y, tal vez, en microservicios), tales servicios deben poder comunicarse de alguna forma; y, aunque pudieran ejecutarse en el mismo clúster de Kubernetes, de todas formas necesitaría controlar el acceso entre ellos. Después de todo, es posible que esté compartiendo ese clúster de Kubernetes con otras aplicaciones que no deberían tener acceso a sus contenedores.

Para controlar el acceso a un contenedor, se deben autenticar sus emisores y, luego, determinar qué solicitudes tienen permitido realizar los demás contenedores. Hoy en día, es común resolver este problema (y varios otros) usando una malla de servicios. Un ejemplo claro de esto es Istio, un proyecto de código abierto creado por IBM y Google, entre otros. En la Figura 7, se muestra dónde encaja Istio en un clúster de Kubernetes.

Tal como se muestra en la figura, el proxy de Istio intercepta todo el tráfico entre los contenedores de su aplicación. Esto permite que la malla de servicios proporcione numerosos servicios útiles sin necesidad de hacer ningún cambio al código de su aplicación. Entre estos servicios, se incluyen los siguientes:

- **Seguridad**, tanto en la autenticación de servicio a servicio con TLS como en la autenticación del usuario final.
- **Administración del tráfico**, que le permite controlar la forma en que se enrutan las solicitudes entre los contenedores de su aplicación.
- **Observabilidad**, es decir, capturar registros y métricas de la comunicación entre sus contenedores.

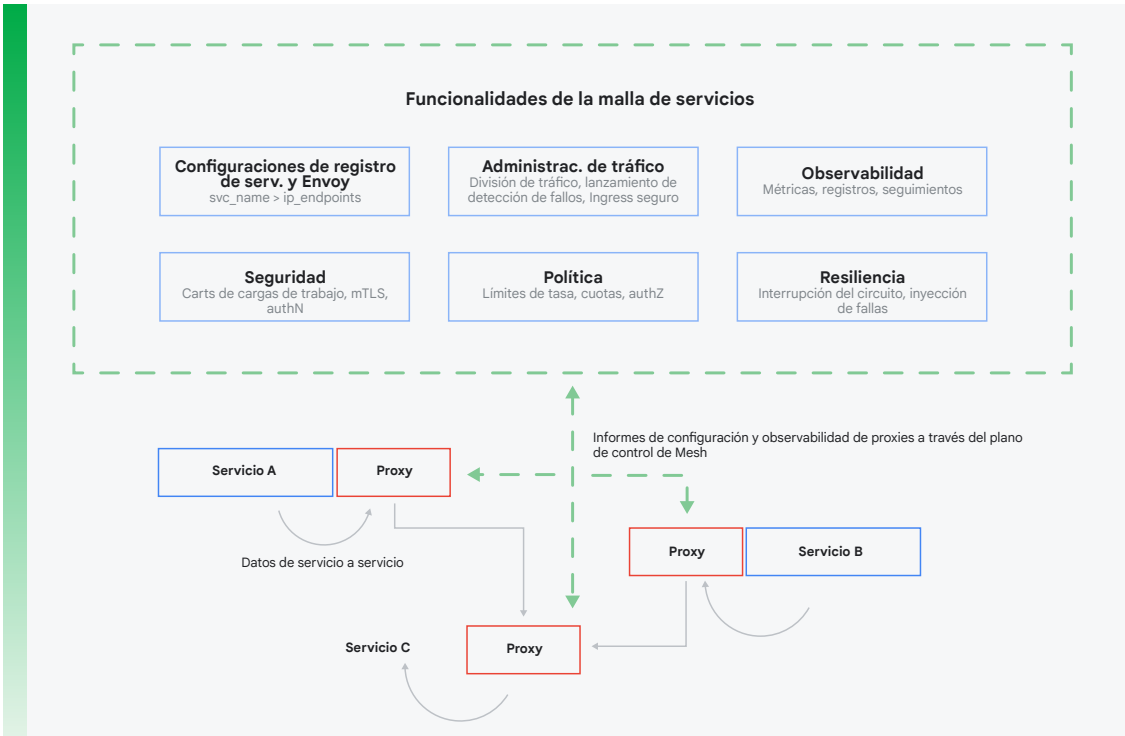


Figura 7: En un clúster de Kubernetes con Istio, todo el tráfico entre los contenedores pasa a través de esta malla de servicios.

GCP le permite agregar Istio a un clúster de GKE. Si bien no es obligatorio usar una malla de servicios, no se sorprenda si los clientes más conocedores que usan sus aplicaciones en la nube comienzan a preguntarle si su seguridad está al nivel de la de Istio. A los clientes les preocupa la seguridad, y en un mundo basado en contenedores, Istio es una parte fundamental para proporcionarla.

Además de ser compatible con Istio de código abierto, Google Cloud ofrece Traffic Director (un plano de control de la malla de servicios totalmente administrado por GCP que brinda balanceo de cargas global entre instancias de VM y clústeres en múltiples regiones), descarga verificaciones de estado de los proxies de servicio y proporciona administración de tráfico sofisticada, además de otras capacidades descritas anteriormente.

Una de las capacidades únicas de Traffic Director es la conmutación por error y el desbordamiento automáticos entre regiones para los microservicios en la malla (como se muestra en la Figura 8).

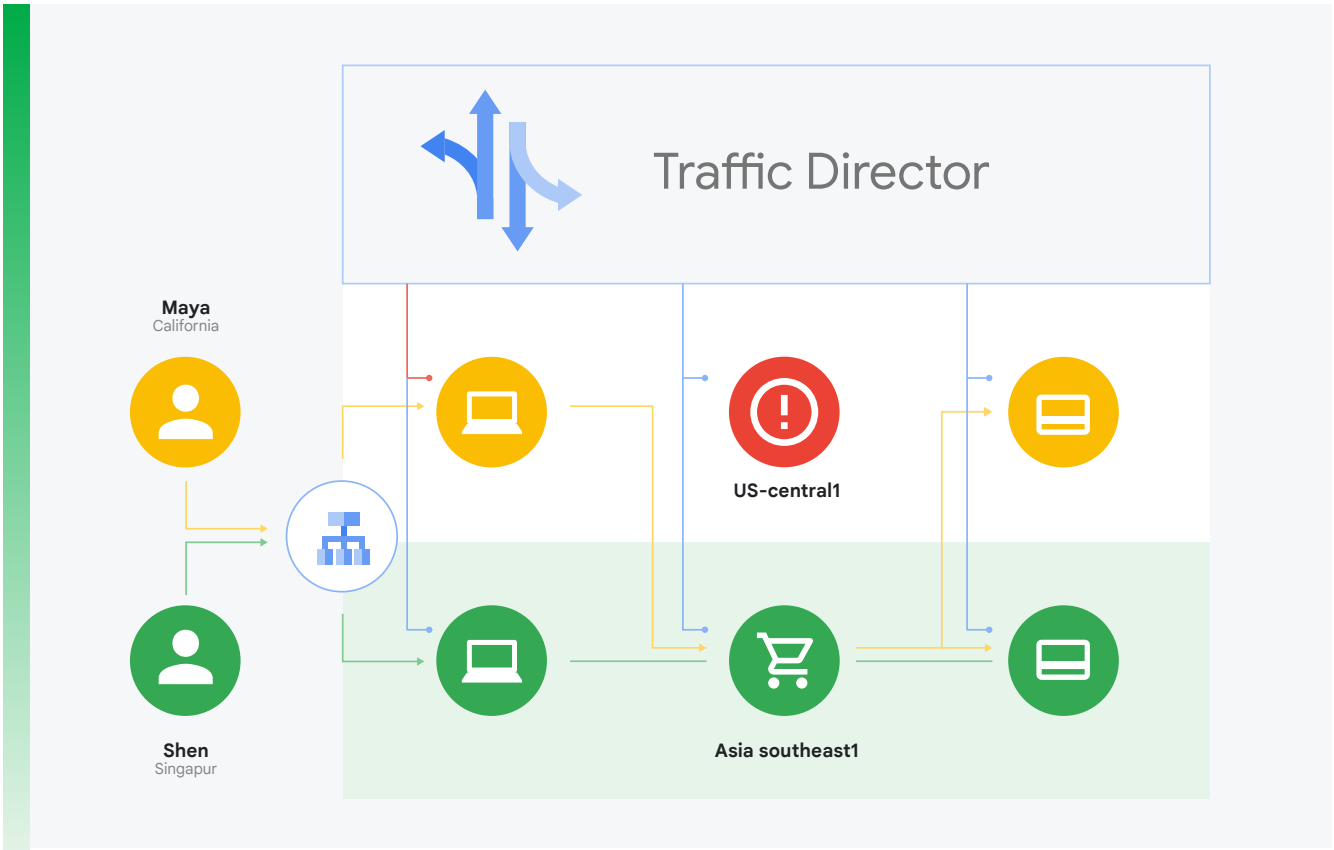


Figura 8: Traffic Director proporciona compatibilidad con la conmutación por error y el desbordamiento automáticos entre regiones

Con esta capacidad, puede combinar resiliencia global con seguridad para sus servicios en la malla de servicios.

Traffic Director ofrece numerosas funciones de administración de tráfico que pueden ayudarlo a mejorar la seguridad de su malla de servicios. Por ejemplo, la función de duplicación de tráfico que se muestra en la Figura 9 puede configurarse fácilmente como una política que permita que una aplicación paralela reciba una copia del tráfico real que se procesa en la versión principal de la app. Después del procesamiento, se descartan las respuestas de copia que recibió el servicio paralelo.

La duplicación de tráfico puede ser una herramienta eficaz para detectar anomalías en la seguridad y depurar errores en el tráfico de producción sin afectarlo.

Sin embargo, proteger las interacciones entre sus contenedores no es el único desafío de seguridad nuevo que genera la refactorización de una aplicación. Otro desafío es garantizar que las imágenes de contenedor que ejecuta sean confiables. Para ello, debe asegurarse de que la cadena de suministro de software tenga integrada la seguridad y el cumplimiento.

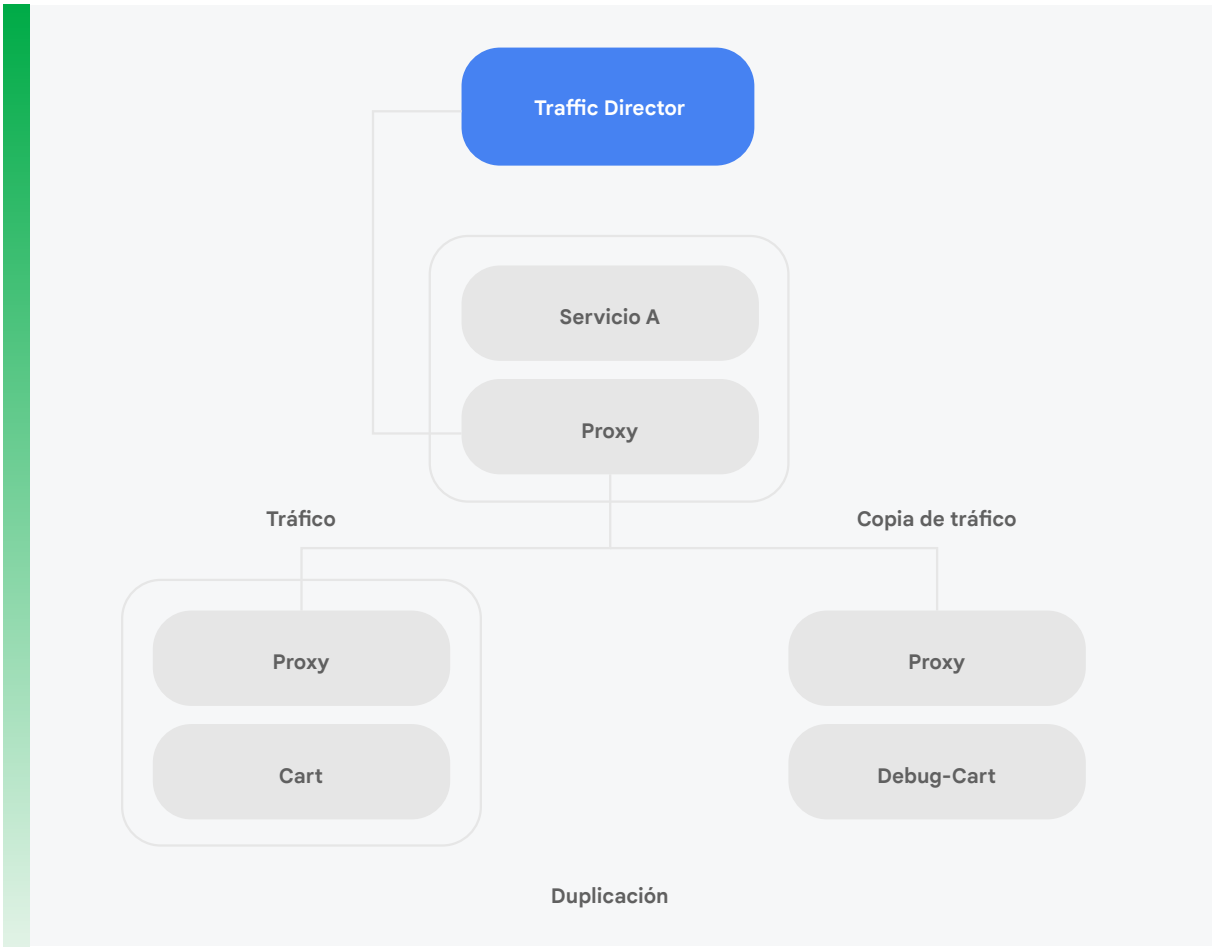


Figura 9: Duplicación de tráfico en Traffic Director

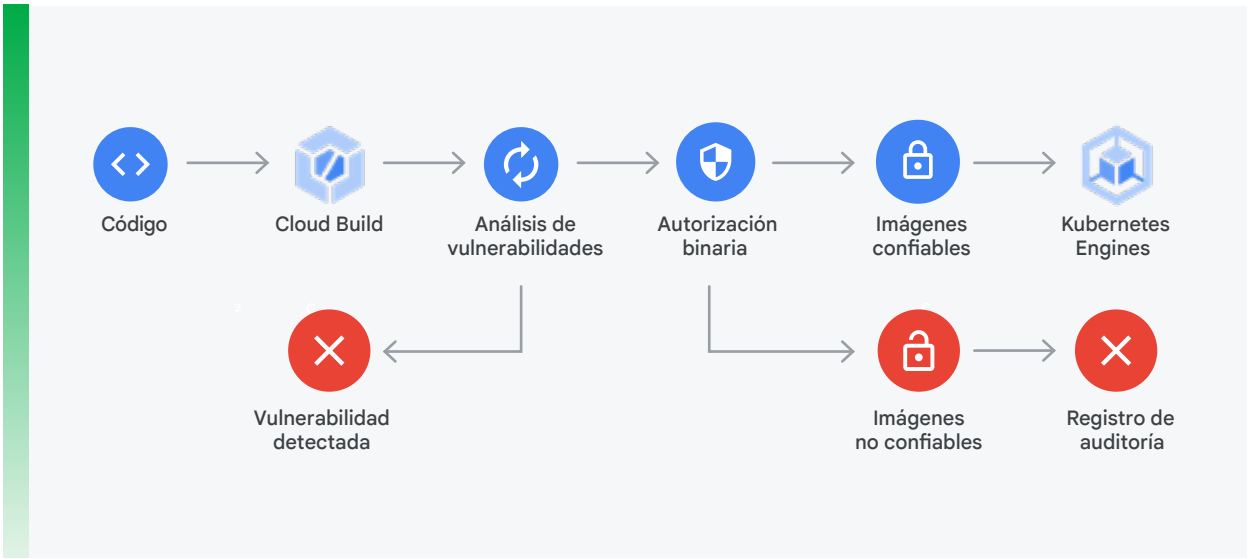


Figura 10: Flujo de trabajo del análisis de vulnerabilidades y la autorización binaria

Es decir, la cadena de suministro debe tener estas dos características (como se muestra en la Figura 10):

Análisis de vulnerabilidades: El análisis de vulnerabilidades de Container Registry le permite obtener comentarios rápidos con respecto a amenazas potenciales, así como identificar problemas en cuanto Cloud Build compila sus contenedores y los almacena en Container Registry. En el caso de Ubuntu, Debian y Alpine, las vulnerabilidades de los paquetes se identifican durante el proceso de desarrollo de aplicaciones. Próximamente, también estará disponible en CentOS y RHEL. Esto evita ineficiencias costosas y reduce el tiempo necesario para remediar vulnerabilidades conocidas.

Autorización binaria: Gracias a la integración de la autorización binaria y el análisis de vulnerabilidades de Container Registry, puede programar implementaciones según los resultados del análisis de vulnerabilidades como parte de la política de implementación general. La autorización binaria es un control de seguridad de tiempo de implementación que garantiza que se implementen imágenes de contenedor confiables en GKE sin ninguna intervención manual.

Proteger el acceso entre contenedores con una malla de servicios y garantizar una cadena de suministro de software segura son aspectos importantes para crear aplicaciones seguras basadas en contenedores. Si bien existen muchos otros aspectos, como verificar la seguridad de la infraestructura de la plataforma en la nube en la que está compilando, lo más importante es estar consciente de que pasar de una aplicación monolítica a un paradigma moderno nativo de la nube genera nuevos desafíos de seguridad. Para que esta transición sea exitosa, debe comprender cuáles son estos desafíos y, luego, crear un plan concreto para abordar cada uno.

Capítulo

03

Comenzar

No considere el cambio a una arquitectura nativa de la nube como un proyecto radical de varios años.

En su lugar, comience ahora buscando un equipo con la capacidad y la experiencia necesarias para elaborar una prueba de concepto, o bien busque uno que ya lo haya hecho.
Luego, comparta las lecciones aprendidas con toda la organización. Haga que los equipos adopten el patrón de aplicación estranguladora; para ello, migre los servicios a una arquitectura nativa de la nube de forma iterativa y progresiva mientras siguen brindando nuevas funcionalidades.

Para tener éxito, es esencial que los equipos tengan la capacidad, la autoridad y los recursos necesarios para hacer que la evolución de la arquitectura de sus sistemas forme parte de su trabajo diario. Defina objetivos claros con respecto a la arquitectura del nuevo trabajo siguiendo las 6 preguntas señaladas anteriormente, pero dé a los equipos la libertad de elegir cómo conseguirlos.

Y lo más importante de todo: no espere más para comenzar. Aumentar la productividad y agilidad de sus equipos, así como la seguridad y estabilidad de sus servicios, será cada vez más decisivo para el éxito de su organización. Los equipos que obtienen los mejores resultados son aquellos que consideran que la mejora y la experimentación disciplinada forman parte de su trabajo diario.

Google inventó Kubernetes con base en el software que usamos internamente desde hace años, es por eso que contamos con la mejor experiencia en cuanto a la tecnología nativa de la nube.

Google Cloud Platform está enfocada especialmente en las aplicaciones alojadas en contenedores, tal como se evidencia en nuestras ofertas de seguridad y CI/CD. La verdad es clara: GCP es el líder actual en lo que respecta a compatibilidad con aplicaciones alojadas en contenedores.

Visite cloud.google.com/devops para realizar una verificación rápida a fin de conocer en qué nivel se encuentra y obtener sugerencias de los pasos que debe seguir, incluida la implementación de los patrones descritos en este informe, como la arquitectura con acoplamiento bajo.

Muchos socios de GCP ya ayudaron a organizaciones como la suya a hacer esta transición. ¿Por qué trazar usted mismo un camino para rediseñar la estructura, cuando podemos brindarle un guía experimentado?

Si desea comenzar, comuníquese con nosotros para programar una reunión con un arquitecto de soluciones de Google. Podemos ayudarlo a comprender en qué consiste el cambio y, luego, trabajar con usted para hacerlo realidad.

Lecturas adicionales

cloud.google.com/devops: Seis informes anuales del estado de DevOps, un conjunto de artículos con información detallada sobre las capacidades que predicen el rendimiento de la entrega de software y una verificación rápida que lo ayuda a descubrir en qué nivel se encuentra y cómo mejorar.

Site Reliability Engineering: How Google Runs Production Systems (O'Reilly, 2016)

The Site Reliability Workbook: Practical Ways to Implement SRE (O'Reilly, 2018)

Building Secure and Reliable Systems: Best Practices for Designing, Implementing and Maintaining Systems (O'Reilly, 2020)

"How to break a Monolith into Microservices: What to decouple and when" de Zhamak Dehghani https://martinfowler.com/articles/break-monolith-into-microservices.html

"Microservices: a definition of this new architectural term" de Martin Fowler https://martinfowler.com/articles/microservices.html

"Strangler Fig Application" de Martin Fowler https://martinfowler.com/bliki/StranglerFigApplication.html