



Google Cloud

Diseña sistemas confiables

Stephanie Wong

Developers Advocate, Google Cloud

En este módulo, veremos cómo diseñar sistemas confiables.

Objetivos de aprendizaje

- Diseñar servicios para cumplir con los requisitos de disponibilidad, durabilidad y escalabilidad
- Implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada
- Prevenir fallas de sobrecarga mediante los patrones de diseño de disyuntor y retirada exponencial truncada
- Diseñar almacenamiento de datos resiliente con eliminación diferida
- Crear diseños para situaciones de estados operativos normales, degradados y de falla
- Analizar situaciones de desastres y planificar, implementar, probar y simular la recuperación ante desastres

Específicamente, explicaremos cómo diseñar servicios para que cumplan los requisitos de disponibilidad, durabilidad y escalabilidad. También analizaremos cómo implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada. La sobrecarga también puede ser un desafío en los sistemas distribuidos. Veremos cómo impedir las fallas por sobrecarga mediante patrones de diseño como el disyuntor y la retirada exponencial truncada.

También presentaremos cómo diseñar sistemas de almacenamiento de datos con eliminación diferida. Cuando ocurre un problema, queremos que nuestros sistemas sigan en control, por lo que estudiaremos decisiones de diseño para estados operativos diferentes, incluidas situaciones normales, degradadas y de falla. Por último, veremos cómo analizar situaciones de desastres y planificar, implementar y probar la recuperación ante desastres.

Objetivos de aprendizaje

- Diseñar servicios para cumplir con los requisitos de disponibilidad, durabilidad y escalabilidad
- Implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada
- Prevenir fallas de sobrecarga mediante los patrones de diseño de disyuntor y retirada exponencial truncada
- Diseñar almacenamiento de datos resiliente con eliminación diferida
- Crear diseños para situaciones de estados operativos normales, degradados y de falla
- Analizar situaciones de desastres y planificar, implementar, probar y simular la recuperación ante desastres

Específicamente, explicaremos cómo diseñar servicios para que cumplan los requisitos de disponibilidad, durabilidad y escalabilidad.

Objetivos de aprendizaje

- Diseñar servicios para cumplir con los requisitos de disponibilidad, durabilidad y escalabilidad
- Implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada
- Prevenir fallas de sobrecarga mediante los patrones de diseño de disyuntor y retirada exponencial truncada
- Diseñar almacenamiento de datos resiliente con eliminación diferida
- Crear diseños para situaciones de estados operativos normales, degradados y de falla
- Analizar situaciones de desastres y planificar, implementar, probar y simular la recuperación ante desastres

También analizaremos cómo implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada.

Objetivos de aprendizaje

- Diseñar servicios para cumplir con los requisitos de disponibilidad, durabilidad y escalabilidad
- Implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada
- Prevenir fallas de sobrecarga mediante los patrones de diseño de disyuntor y retirada exponencial truncada
- Diseñar almacenamiento de datos resiliente con eliminación diferida
- Crear diseños para situaciones de estados operativos normales, degradados y de falla
- Analizar situaciones de desastres y planificar, implementar, probar y simular la recuperación ante desastres

La sobrecarga también puede ser un desafío en los sistemas distribuidos. Veremos cómo impedir las fallas por sobrecarga mediante patrones de diseño como el disyuntor y la retirada exponencial truncada.

Objetivos de aprendizaje

- Diseñar servicios para cumplir con los requisitos de disponibilidad, durabilidad y escalabilidad
- Implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada
- Prevenir fallas de sobrecarga mediante los patrones de diseño de disyuntor y retirada exponencial truncada
- Diseñar almacenamiento de datos resiliente con eliminación diferida
- Crear diseños para situaciones de estados operativos normales, degradados y de falla
- Analizar situaciones de desastres y planificar, implementar, probar y simular la recuperación ante desastres

También presentaremos cómo diseñar sistemas de almacenamiento de datos con eliminación diferida.

Objetivos de aprendizaje

- Diseñar servicios para cumplir con los requisitos de disponibilidad, durabilidad y escalabilidad
- Implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada
- Prevenir fallas de sobrecarga mediante los patrones de diseño de disyuntor y retirada exponencial truncada
- Diseñar almacenamiento de datos resiliente con eliminación diferida
- Crear diseños para situaciones de estados operativos normales, degradados y de falla
- Analizar situaciones de desastres y planificar, implementar, probar y simular la recuperación ante desastres

Cuando ocurre un problema, queremos que nuestros sistemas sigan en control, por lo que estudiaremos decisiones de diseño para estados operativos diferentes, incluidas situaciones normales, degradadas y de falla.

Objetivos de aprendizaje

- Diseñar servicios para cumplir con los requisitos de disponibilidad, durabilidad y escalabilidad
- Implementar sistemas tolerantes a errores evitando puntos únicos de fallo, fallas correlacionadas y fallas en cascada
- Prevenir fallas de sobrecarga mediante los patrones de diseño de disyuntor y retirada exponencial truncada
- Diseñar almacenamiento de datos resiliente con eliminación diferida
- Crear diseños para situaciones de estados operativos normales, degradados y de falla
- Analizar situaciones de desastres y planificar, implementar, probar y simular la recuperación ante desastres

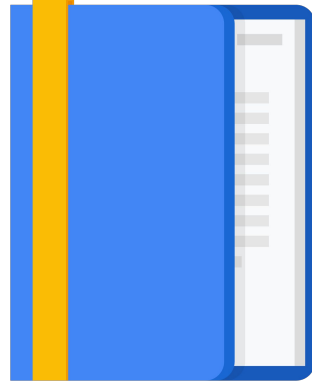
Por último, veremos cómo analizar situaciones de desastres y planificar, implementar y probar la recuperación ante desastres.

Temario

Métricas de rendimiento clave

Creación de diseños que
garanticen la confiabilidad

Planificación ante desastres



Primero, revisemos las métricas de rendimiento clave de los sistemas confiables.

Cuando cree diseños para garantizar la confiabilidad, considere estas métricas de rendimiento clave

Disponibilidad	Durabilidad	Escalabilidad
Es el porcentaje de tiempo en el que se ejecuta un sistema y puede procesar solicitudes.	Se refiere a las probabilidades de perder datos debido a una falla de hardware o del sistema.	Es la capacidad de un sistema para seguir funcionando cuando aumenta la carga de usuarios y datos.
<ul style="list-style-type: none">● Se obtiene con tolerancia a errores.● Cree sistemas de copia de seguridad.● Use verificaciones de estado.● Use métricas de caja blanca para medir las fallas y los aciertos reales del tráfico.	<ul style="list-style-type: none">● Se obtiene por medio de la replicación de datos en varias zonas.● Cree copias de seguridad frecuentes.● Practique el restablecimiento desde copias de seguridad.	<ul style="list-style-type: none">● Supervise el uso.● Use el ajuste de escala automático de la capacidad y quite servidores en respuesta a los cambios de carga.

Cuando cree diseños para garantizar la confiabilidad, considere la disponibilidad, durabilidad y escalabilidad como métricas de rendimiento clave. Explicaremos cada una de ellas:

- La disponibilidad es el porcentaje de tiempo en el que se ejecuta un sistema y puede procesar solicitudes. Para lograr una alta disponibilidad, la supervisión es fundamental. Las verificaciones de estado pueden detectar si una aplicación funciona correctamente. Supervisar con más detalle los servicios usando métricas de caja blanca para medir las fallas y los aciertos del tráfico ayudará a predecir problemas. Para mejorar la disponibilidad, también es fundamental aumentar la tolerancia a errores mediante acciones como quitar los puntos únicos de fallo. Los sistemas de copia de seguridad también son fundamentales para lograrlo.
- La durabilidad es la probabilidad de perder datos debido a fallas del hardware o del sistema. Se requiere una combinación de replicación y copias de seguridad para garantizar la conservación y disponibilidad de los datos. Los datos se pueden replicar en varias zonas. Se deben realizar restablecimientos desde copias de seguridad con frecuencia para confirmar que el proceso funciona correctamente.
- La escalabilidad es la capacidad de un sistema para seguir funcionando cuando aumenta la carga de usuarios y datos. Se deben usar la supervisión y el ajuste de escala automático para responder a las variaciones de carga. Las métricas del escalamiento pueden ser estándares, como la CPU o la memoria, o puede crear métricas personalizadas, como “cantidad de

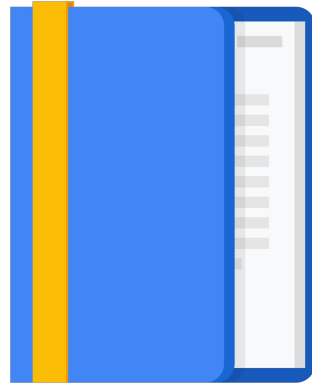
- jugadores conectados a un servidor”.

Temario

Métricas de rendimiento clave

Creación de diseños que
garanticen la confiabilidad

Planificación ante desastres



Ahora que explicamos las métricas de rendimiento clave, veamos cómo crear diseños que garanticen la confiabilidad.

Evite los puntos únicos de fallo

Dos instancias de respaldo ($N + 2$)

- Defina su unidad de implementación.
- $N + 2L$: Planifique tener una unidad disponible para hacer actualizaciones o pruebas, y garantizar la continuidad del servicio si falla otra.
- Asegúrese de que cada unidad pueda controlar la carga adicional.
- No cree ninguna unidad independiente que sea demasiado grande.
- Cree unidades que sean clones intercambiables sin estado.

Evite los puntos únicos de fallo replicando los datos y creando múltiples instancias de máquinas virtuales. Es importante que defina su unidad de implementación y comprenda sus capacidades. Para evitar los puntos únicos de fallo, debe implementar dos instancias adicionales ($N + 2$) a fin de abordar las fallas y las actualizaciones. Lo ideal es que estas implementaciones se ubiquen en zonas diferentes para mitigar las fallas zonales.

Explicaré la situación de actualización. Supongamos que tiene 3 VMs con balanceo de cargas para cumplir con la recomendación de $N + 2$. Si una se está actualizando y otra falla, tendrá un 50% menos de capacidad disponible para el procesamiento, lo que, potencialmente, duplica la carga que recibe la instancia restante y aumenta las probabilidades de que falle. Aquí es cuando es importante planificar la capacidad y conocer el potencial de su implementación. Además, para facilitar el escalamiento, una práctica recomendada es convertir las unidades de implementación en clones intercambiables sin estado.

Evite los puntos únicos de fallo

Dos instancias de respaldo ($N + 2$)

- Defina su unidad de implementación.
- $N + 2L$: Planifique tener una unidad disponible para hacer actualizaciones o pruebas, y garantizar la continuidad del servicio si falla otra.
- Asegúrese de que cada unidad pueda controlar la carga adicional.
- No cree ninguna unidad independiente que sea demasiado grande.
- Cree unidades que sean clones intercambiables sin estado.

Evite los puntos únicos de fallo replicando los datos y creando múltiples instancias de máquinas virtuales.

Evite los puntos únicos de fallo

Dos instancias de respaldo ($N + 2$)

- Defina su unidad de implementación.
- $N + 2L$: Planifique tener una unidad disponible para hacer actualizaciones o pruebas, y garantizar la continuidad del servicio si falla otra.
- Asegúrese de que cada unidad pueda controlar la carga adicional.
- No cree ninguna unidad independiente que sea demasiado grande.
- Cree unidades que sean clones intercambiables sin estado.

Es importante que defina su unidad de implementación y comprenda sus capacidades.

Evite los puntos únicos de fallo

Dos instancias de respaldo ($N + 2$)

- Defina su unidad de implementación.
- $N + 2L$: Planifique tener una unidad disponible para hacer actualizaciones o pruebas, y garantizar la continuidad del servicio si falla otra.
- Asegúrese de que cada unidad pueda controlar la carga adicional.
- No cree ninguna unidad independiente que sea demasiado grande.
- Cree unidades que sean clones intercambiables sin estado.

Para evitar los puntos únicos de fallo, debe implementar dos instancias adicionales ($N + 2$) a fin de abordar las fallas y las actualizaciones.

Evite los puntos únicos de fallo

Dos instancias de respaldo ($N + 2$)

- Defina su unidad de implementación.
- $N + 2L$: Planifique tener una unidad disponible para hacer actualizaciones o pruebas, y garantizar la continuidad del servicio si falla otra.
- Asegúrese de que cada unidad pueda controlar la carga adicional.
- No cree ninguna unidad independiente que sea demasiado grande.
- Cree unidades que sean clones intercambiables sin estado.

Lo ideal es que estas implementaciones se ubiquen en zonas diferentes para mitigar las fallas zonales.

Explicaré la situación de actualización. Supongamos que tiene 3 VMs con balanceo de cargas para cumplir con la recomendación de $N + 2$. Si una se está actualizando y otra falla, tendrá un 50% menos de capacidad disponible para el procesamiento, lo que, potencialmente, duplica la carga que recibe la instancia restante y aumenta las probabilidades de que falle.

Evite los puntos únicos de fallo

Dos instancias de respaldo ($N + 2$)

- Defina su unidad de implementación.
- $N + 2L$: Planifique tener una unidad disponible para hacer actualizaciones o pruebas, y garantizar la continuidad del servicio si falla otra.
- Asegúrese de que cada unidad pueda controlar la carga adicional.
- No cree ninguna unidad independiente que sea demasiado grande.
- Cree unidades que sean clones intercambiables sin estado.

Aquí es cuando es importante planificar la capacidad y conocer el potencial de su implementación.

Evite los puntos únicos de fallo

Dos instancias de respaldo ($N + 2$)

- Defina su unidad de implementación.
- $N + 2L$: Planifique tener una unidad disponible para hacer actualizaciones o pruebas, y garantizar la continuidad del servicio si falla otra.
- Asegúrese de que cada unidad pueda controlar la carga adicional.
- No cree ninguna unidad independiente que sea demasiado grande.
- Cree unidades que sean clones intercambiables sin estado.

Además, para facilitar el escalamiento, una práctica recomendada es convertir las unidades de implementación en clones intercambiables sin estado.

Tenga cuidado con las fallas correlacionadas

Las fallas correlacionadas suceden cuando elementos relacionados fallan al mismo tiempo.

- Si falla una máquina, fallarán todas las solicitudes que entrega.
- Si falla un conmutador de la parte superior del bastidor, fallará todo el bastidor.
- Si se pierde una zona o región, fallarán todos sus recursos.
- Los servidores que ejecuten el mismo software tendrán el mismo problema.
- Si falla un sistema de configuración global y muchos sistemas dependen de él, es posible que también fallen.

El grupo de elementos relacionados que pueden fallar al mismo tiempo se denomina **dominio con fallas**.

También es importante tener en cuenta las fallas correlacionadas, que suceden cuando fallan simultáneamente elementos que se relacionan entre sí. En el nivel más simple, si falla una máquina, fallarán todas las solicitudes que entrega. A nivel del hardware, si falla un conmutador de la parte superior del bastidor, fallará todo el bastidor. A nivel de la nube, si se pierde una zona o región, todos los recursos dejarán de estar disponibles. Los servidores que ejecutan el mismo software tienen el mismo problema: si hay una falta en el software, los servidores pueden fallar en tiempos similares.

Las fallas correlacionadas también se pueden aplicar a los datos de configuración. Si falla un sistema de configuración global y muchos sistemas dependen de él, es posible que también fallen. Cuando tenemos un grupo de elementos relacionados que pueden fallar juntos, lo llamamos un dominio con *fallas* o con *errores*.

Para evitar fallas correlacionadas...

Separe los servidores y use microservicios distribuidos entre varios dominios con fallas.

- Divida la lógica empresarial en servicios según los dominios con fallas.
- Realice implementaciones en varias zonas o regiones.
- Divida la responsabilidad en componentes y expándala en varios procesos.
- Diseñe servicios independientes, con acoplamiento bajo, pero que colaboran entre sí.

Se pueden usar varias técnicas para evitar fallas correlacionadas. Es útil tener en cuenta los dominios con fallas; luego, los servidores se pueden separar con microservicios distribuidos entre varios dominios con fallas. A fin de lograrlo, puede dividir la lógica empresarial en servicios según los dominios con fallas para realizar implementaciones en varias zonas o regiones.

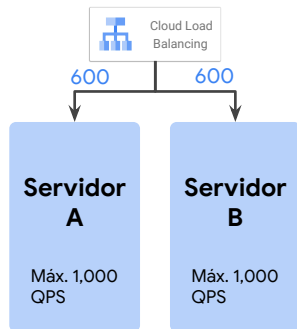
En un nivel más detallado, es recomendable dividir las responsabilidades en componentes y expandirlas en varios procesos. De este modo, si falla un componente, los demás no se verán afectados. Si todas las responsabilidades dependen de un componente, es muy probable que, si falla una, fallen todas las demás.

Cuando diseñe microservicios, debe generar servicios independientes, con acoplamiento bajo, pero que colaboren entre sí. Si falla un servicio, no se deben provocar fallas en los demás. Puede hacer que un servicio colaborativo tenga capacidad reducida o no pueda procesar por completo sus flujos de trabajo, pero el servicio debe mantener el control y no fallar.

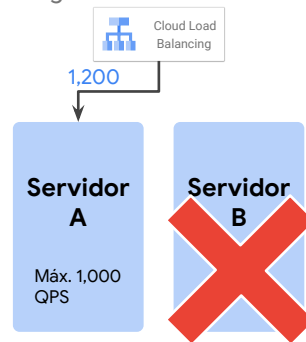
Tenga cuidado con las fallas en cascada

Las fallas en cascada suceden cuando falla un sistema y esto provoca que otros se sobrecarguen, como una cola de mensajes que se sobrecarga debido a la falla de un backend.

Los servidores A y B dividen la carga



El servidor B falla, lo que genera una sobrecarga en el A



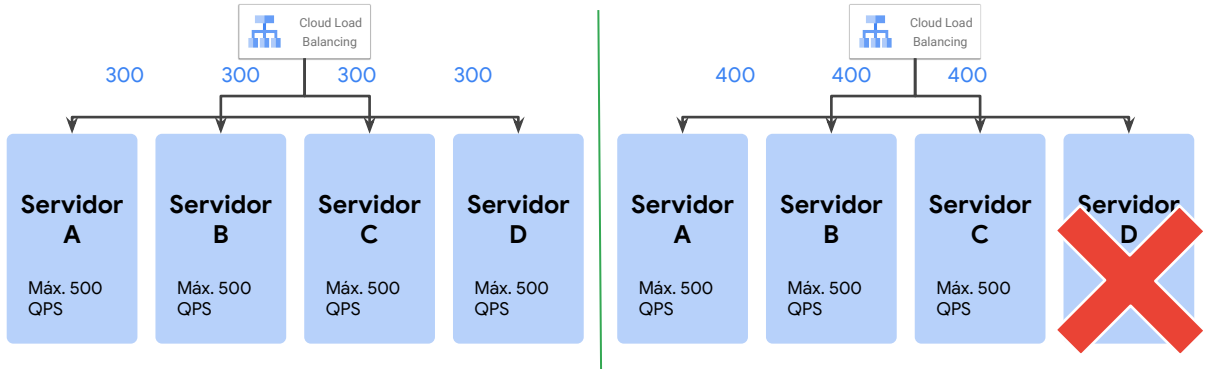
Las fallas en cascada suceden cuando falla un sistema y esto provoca que otros se sobrecarguen y, posteriormente, fallen. Por ejemplo, una cola de mensajes se puede sobrecargar porque un backend falla y no puede procesar los mensajes que están en la cola.

En el gráfico del lado izquierdo, se muestra un balanceador de cargas en la nube que distribuye la carga entre dos servidores de backend. Cada servidor puede admitir un máximo de 1,000 consultas por segundo. En este momento, el balanceador de cargas está enviando 600 consultas por segundo a cada instancia. Si el servidor B falla ahora, todas las 1,200 consultas por segundo se deben enviar al servidor A, como se muestra en el lado derecho. Esta cantidad es mayor que el máximo especificado y podría provocar una falla en cascada.

¿Cómo evitamos las fallas en cascada?

Cómo evitar las fallas en cascada

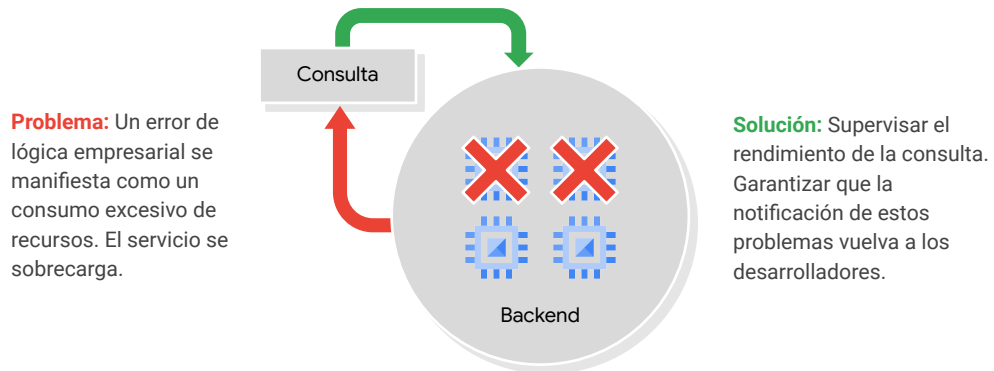
- Use verificaciones de estado en Compute Engine o pruebas de preparación y disponibilidad en Kubernetes para detectar y, luego, reparar las instancias en mal estado.
- Asegúrese de que las nuevas instancias del servidor se inicien rápido y de que, idealmente, no dependan de otros backends o sistemas para iniciarse.



Las fallas en cascada se pueden controlar con asistencia de la plataforma de implementación. Por ejemplo, puede usar verificaciones de estado en Compute Engine o pruebas de preparación y disponibilidad en GKE para habilitar la detección y reparación de las instancias en mal estado. Debe asegurarse de que las nuevas instancias se inicien rápido y que, idealmente, no dependan de otros backends o sistemas para iniciarse antes de estar listas.

En el gráfico de esta diapositiva, se ilustra una implementación con cuatro servidores que utilizan un balanceador de cargas. En función del tráfico actual, los tres servidores restantes pueden absorber una falla del servidor, como se muestra en el lado derecho. Si el sistema usa Compute Engine con grupos de instancias y reparación automática, el servidor con errores se reemplazaría por una instancia nueva. Como mencionamos, es importante que el nuevo servidor se inicie con rapidez para restablecer la capacidad completa tan rápido como sea posible. Además, esta configuración solo funciona con los servicios sin estado.

Sobrecarga causada por una “consulta de la muerte”



También debe planificar para proteger el servidor contra la “consulta de la muerte”, que puede generar una falla en el servicio. Se denomina así porque el error se manifiesta como un consumo excesivo de recursos, pero, en realidad, se debe a un error de lógica empresarial.

Puede ser difícil de descubrir y se requieren buena supervisión, observabilidad y registro para determinar la causa raíz del problema. Cuando se envían solicitudes, se deben supervisar la latencia, el uso de recursos y las tasas de errores para identificar el problema.

Falla de sobrecarga causada por un ciclo de retroalimentación positiva

Problema: Agregó reintentos para tratar de aumentar la confiabilidad del sistema, pero, en cambio, genera un potencial de sobrecarga.

Solución: Para impedir la sobrecarga, considere cuidadosamente las condiciones de sobrecarga cuando intente mejorar la confiabilidad con mecanismos de retroalimentación para invocar reintentos.



También debe planificar en caso de una falla de sobrecarga causada por un ciclo de retroalimentación positiva, que ocurre cuando se producen problemas por tratar de impedir otros.

Estos ocurren cuando agrega reintentos en caso de fallas para tratar de hacer que el sistema sea más confiable. En vez de corregir la falla, se genera un potencial de sobrecarga. También es posible que agregue más carga a un sistema que ya está sobrecargado.

La solución son los reintentos inteligentes que aprovechan la retroalimentación del servicio con fallas. Explicaremos dos estrategias para lograrlo.

Use un patrón de retirada exponencial truncada para evitar una sobrecarga por retroalimentación positiva en el cliente

- Si falla la invocación del servicio, vuelva a intentarlo:
 - Vuelva a intentarlo, pero espere un poco entre cada intento.
 - Espere un poco más de tiempo cada vez que falle la solicitud.
 - Establezca un tiempo máximo y una cantidad máxima de solicitudes.
 - Finalmente, deje de intentarlo.
- Ejemplo:
 - La solicitud falla. Espere 1 segundo + una_cantidad_aleatoria_de_milisegundos y vuelva a intentarlo.
 - La solicitud falla. Espere 2 segundos + una_cantidad_aleatoria_de_milisegundos y vuelva a intentarlo.
 - La solicitud falla. Espere 4 segundos + una_cantidad_aleatoria_de_milisegundos y vuelva a intentarlo.
 - Continúe así hasta un tiempo de retirada_máxima.
 - Siga con la espera y los reintentos hasta llegar a una cantidad máxima.

Está bien volver a intentarlo si el servicio falla. Sin embargo, se debe hacer de forma controlada. Una manera es usar el patrón de retirada exponencial, que genera un reintento, pero no inmediatamente. Debe esperar entre cada reintento y esperar un poco más cada vez que falle la solicitud a fin de que el servicio que falla tenga tiempo para recuperarse. La cantidad de reintentos y el tiempo antes de rendirse se deben limitar a un máximo.

A modo de ejemplo, considere una solicitud con errores a un servicio. Con la retirada exponencial, podemos esperar 1 segundo y una cantidad aleatoria de milisegundos antes de volver a intentarlo. Si la solicitud vuelve a fallar, esperamos 2 segundos y una cantidad aleatoria de milisegundos antes de volver a intentarlo. Si ocurre una nueva falla, esperamos 4 segundos y una cantidad aleatoria de milisegundos hasta llegar a una cantidad máxima.

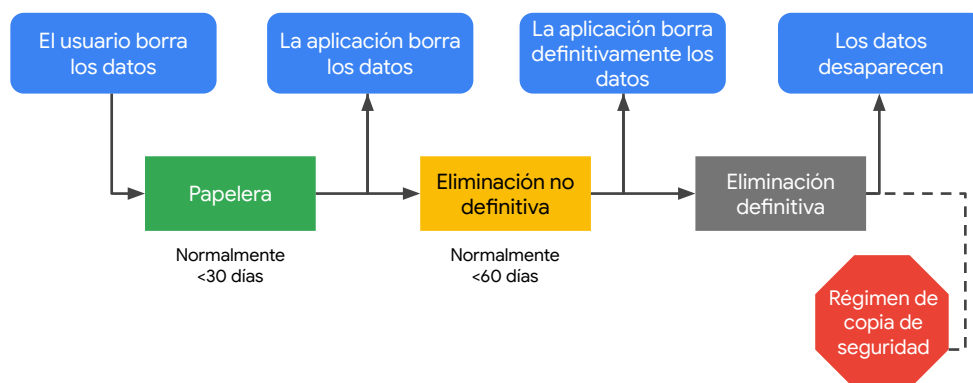
Use el patrón de disyuntor para proteger el servicio contra los reintentos excesivos

- Planifique para operaciones de estado degradado.
- Si un servicio dejó de funcionar y todos sus clientes están realizando reintentos, este aumento en las solicitudes puede empeorar la situación.
 - Proteja el servicio con un proxy que supervise la calidad del servicio (*el disyuntor*).
 - Si el servicio no está en buenas condiciones, no desvíe las solicitudes a él.
- Si usa GKE, aproveche Istio para implementar disyuntores automáticamente.

El patrón de disyuntor también puede impedir que se hagan demasiados reintentos en un servicio. El patrón implementa una solución en caso de que un servicio esté en un estado de operación degradada. Esto es importante porque, si un servicio falla o está sobrecargado y todos sus clientes están realizando reintentos, las solicitudes adicionales pueden empeorar la situación. El patrón de diseño de disyuntor protege el servicio con un proxy que supervisa el estado del servicio. Si el disyuntor determina que el servicio no está en buenas condiciones, no desviará las solicitudes a él. Cuando el servicio vuelva a estar en funcionamiento, el disyuntor comenzará a ingresar solicitudes nuevamente de forma controlada.

Si está usando GKE, la malla de servicios de Istio implementa disyuntores automáticamente.

Use la eliminación diferida para recuperar datos de forma confiable cuando los usuarios los borran por equivocación



La eliminación diferida es un método basado en la capacidad de recuperar datos de forma confiable cuando un usuario los borra por equivocación. Con la eliminación diferida, se inicia una canalización de eliminación parecida a la que se muestra en este gráfico y la eliminación avanza en fases. En la primera etapa, el usuario borra los datos, pero estos se pueden restablecer en un período predefinido, que, en este ejemplo, es de 30 días. El objetivo es proteger contra los errores del usuario.

Cuando se acaba el período predefinido, los datos ya no son visibles para el usuario y pasan a la fase de eliminación no definitiva. Aquí, los administradores o el equipo de atención al usuario pueden restablecer los datos. El objetivo es proteger contra los errores de la aplicación. Después del período de eliminación no definitiva de 15, 30, 45 o, incluso, 50 días, los datos se borran y ya no están disponibles. La única forma de restablecerlos es por medio de cualquier copia de seguridad o archivos que se hayan creado a partir de los datos.

Actividad 10: Diseñe aplicaciones confiables y escalables

Consulte el cuaderno de ejercicios de Design and Process.

- Dibuje un diagrama en el que se muestre cómo implementar su aplicación para que tenga alta disponibilidad, escalabilidad y durabilidad.



En esta actividad de diseño, dibujará un diagrama en el que se muestre cómo implementar su aplicación para que tenga alta disponibilidad, escalabilidad y durabilidad. Le mostraremos un ejemplo de lo que debe dibujar.

Actividad 10: Diseñe aplicaciones confiables y escalables

Consulte el cuaderno de ejercicios de Design and Process.

- Dibuje un diagrama en el que se muestre cómo implementar su aplicación para que tenga alta disponibilidad, escalabilidad y durabilidad.

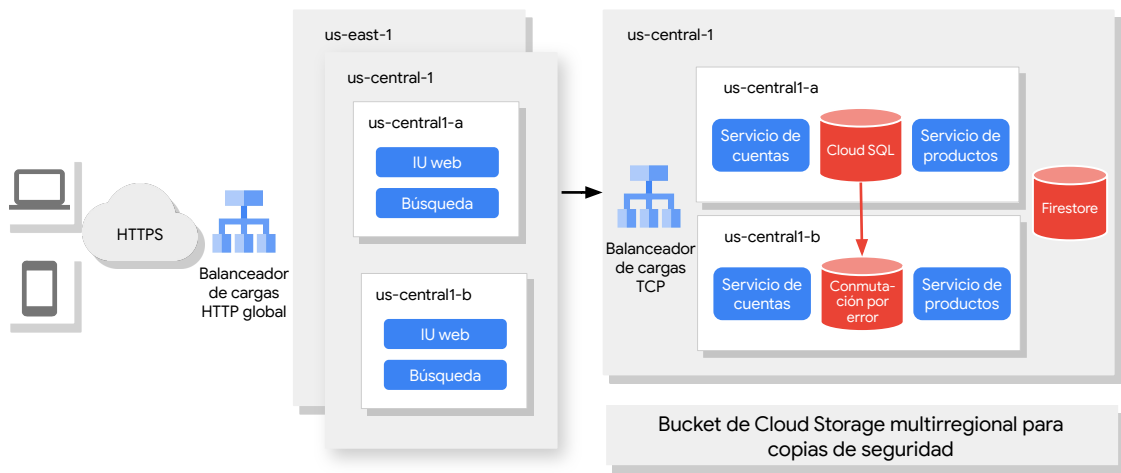
En esta actividad de diseño...

Actividad 10: Diseñe aplicaciones confiables y escalables

Consulte el cuaderno de ejercicios de Design and Process.

- Dibuje un diagrama en el que se muestre cómo implementar su aplicación para que tenga alta disponibilidad, escalabilidad y durabilidad.

dibujará un diagrama en el que se muestre cómo implementar su aplicación para que tenga alta disponibilidad, escalabilidad y durabilidad. Le mostraremos un ejemplo de lo que debe dibujar.



Este diagrama corresponde a una aplicación de banca en línea con clientes en Estados Unidos.

Deseo que la IU web tenga una alta disponibilidad, así que el gráfico indica que se implementa detrás de un balanceador de cargas HTTP global ubicado en múltiples regiones y zonas dentro de cada región. Elegí us-central-1 como la región principal porque está, más o menos, en el medio de EE.UU. También hay una región de copias de seguridad ubicada en us-east-1, que se encuentra en la costa este de EE.UU.

Implemento las cuentas y servicios de productos como backends solo en la región us-central1, aunque estoy usando varias zonas (us-central1-a y us-central1-b) para tener alta disponibilidad. Incluso tengo una base de datos de Cloud SQL de conmutación por error. La base de datos de Firestore para el servicio de productos es multirregional, por lo que no debo preocuparme de una conmutación por error.

En caso de desastres, almacenaré las copias de seguridad en un bucket de Cloud Storage multirregional. De esta forma, si hay una interrupción en la región, puedo restablecer el servicio en otra.

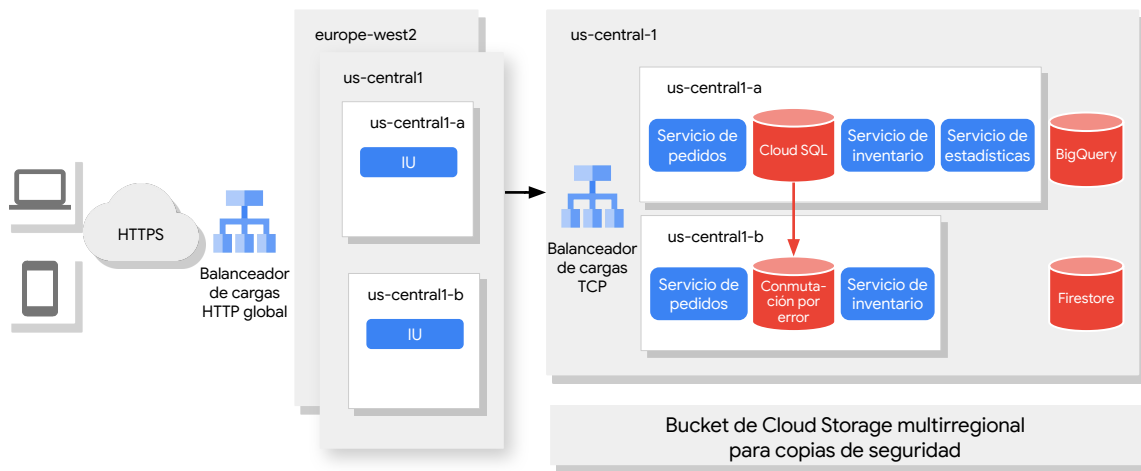
Consulte la actividad 10 del cuaderno de ejercicios a fin de crear un diagrama similar para sus servicios.

Revisar actividad 10: Diseñe aplicaciones confiables y escalables

- Dibuje un diagrama en el que se muestre cómo implementar su aplicación para que tenga alta disponibilidad, escalabilidad y durabilidad.



En esta actividad, se le solicitó que dibujara un diagrama en el que se mostrara cómo implementar su aplicación para que tuviera alta disponibilidad, escalabilidad y durabilidad.



Supongamos que TurisClic, nuestra aplicación de viajes en línea, pertenece a una empresa estadounidense, pero tiene un gran grupo de clientes en Europa.

Como quiero que la IU tenga una alta disponibilidad, la ubiqué en us-central1 y europe-west2 con un balanceador de cargas HTTP global. Este balanceador de cargas enviará las solicitudes de los usuarios a la región más cercana, a menos que esta no pueda admitir el tráfico.

También podría implementar los backends globalmente, pero estoy tratando de optimizar los costos. Puedo comenzar a hacerlo en us-central1. Si bien aumentará la latencia para los usuarios de Europa, puedo volver a revisar este punto más adelante y establecer un backend similar en europe-west2.

Para garantizar una alta disponibilidad, decidí implementar los servicios de pedidos y de inventario en múltiples zonas. Dado que el servicio de análisis no se orienta a los clientes, puedo ahorrar más dinero si lo implemento en una sola zona. Una vez más, tengo una base de datos de Cloud SQL de conmutación por error. Como la base de datos de Firestore y el almacén de datos de BigQuery son multirregionales, no debo preocuparme de su conmutación por error.

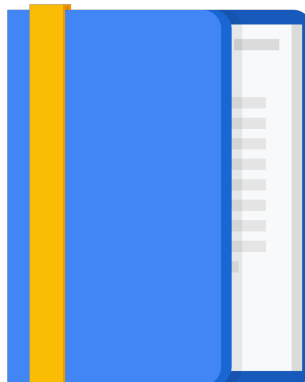
En caso de desastres, almacenaré las copias de seguridad en un bucket de Cloud Storage multirregional. De esta forma, si hay una interrupción en la región, puedo restablecer el servicio en otra.

Temario

Métricas de rendimiento clave

Creación de diseños que
garanticen la confiabilidad

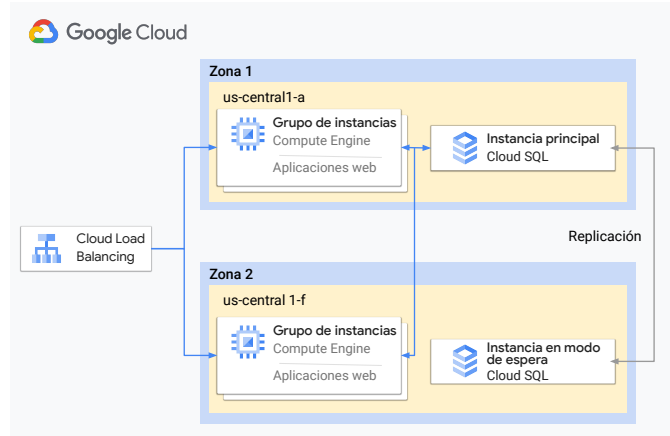
Planificación ante desastres



Ahora que creamos diseños para garantizar la confiabilidad, exploremos la planificación ante desastres.

Se puede obtener alta disponibilidad realizando implementaciones en varias zonas de una región

- Implemente múltiples servidores.
- Organice servidores con un grupo de instancias administrado regional.
- Cree una base de datos de conmutación por error en otra zona o use una base de datos distribuida como Firestore o Spanner.



Se puede obtener alta disponibilidad realizando implementaciones en varias zonas de una región. Para obtener alta disponibilidad cuando se usa Compute Engine, puede usar un grupo de instancias regionales, que entrega funcionalidad incorporada a fin de mantener las instancias en ejecución. Use la reparación automática con una verificación de estado de la aplicación y el balanceo de cargas para distribuir la carga.

Para los datos, la solución de almacenamiento seleccionada afectará lo que se necesita a fin de lograr alta disponibilidad.

En Cloud SQL, la base de datos se puede configurar a fin de tener alta disponibilidad, lo que entrega redundancia de datos y una instancia en espera para el servidor de base de datos en otra zona. En este diagrama, se muestra una configuración de alta disponibilidad con un grupo de instancias administrado regional para una aplicación web que utiliza un balanceador de cargas. La instancia de Cloud SQL principal está en us-central1-a con una instancia de réplica ubicada en us-central1-f.

Algunos servicios de datos, como Firestore o Spanner, entregan alta disponibilidad de forma predeterminada.

Los grupos de instancias administradas regionales distribuyen de forma uniforme las VMs entre zonas en la región especificada

Ubicación

Para asegurar una mayor disponibilidad, selecciona una ubicación de varias zonas para un grupo de instancias. [Más información](#)



Zona única



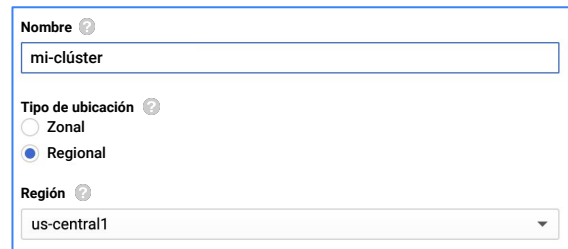
Varias zonas

Solo los grupos de instancias administrados pueden existir en varias zonas.

En el ejemplo anterior, el grupo de instancias administrado regional distribuía VMs entre zonas. Puede elegir entre configuración de zonas únicas y múltiples (o regionales) cuando crea su grupo de instancias, como puede ver en esta captura de pantalla.

Los clústeres de Kubernetes también se pueden implementar en zonas únicas o múltiples

- Los clústeres de Kubernetes se componen de un conjunto de grupos de nodos.
- Seleccionar el tipo de ubicación regional replica los grupos de nodos en múltiples zonas en la región especificada.



A screenshot of the Google Kubernetes Engine console configuration interface. It features three main sections: 'Nombre' with a text input field containing 'mi-clúster'; 'Tipo de ubicación' with two radio button options, 'Zonal' and 'Regional', where 'Regional' is selected; and 'Región' with a dropdown menu showing 'us-central1'.

Nombre ?
mi-clúster

Tipo de ubicación ?
☐ Zonal
☒ Regional

Región ?
us-central1

Los clústeres de Google Kubernetes Engine también se pueden implementar en una o varias zonas, como se muestra en esta captura de pantalla.

Un clúster consta de un controlador principal y conjuntos de grupos de nodos. Los clústeres regionales aumentan la disponibilidad del plano de control de un clúster y de sus nodos cuando se replican en varias zonas de una región.

Cree una verificación de estado cuando cree grupos de instancias para habilitar la reparación automática

- Cree un extremo de prueba en su servicio.
- Pruebe las necesidades del extremo para verificar que el servicio está funcionando y, además, que se puede comunicar con una base de datos y servicios de backend dependientes.
- Si falla la verificación de estado, el grupo de instancias creará un nuevo servidor y borrará el que no funciona.
- Los balanceadores de cargas también usan verificaciones de estado para asegurarse de que solo envíen solicitudes a instancias en buen estado.

The screenshot shows a configuration form for a health check endpoint. The form is titled 'mi-verificación-de-estado' and includes the following fields and options:

- Nombre:** mi-verificación-de-estado (Note: El nombre es permanente)
- Descripción (opcional):** (Empty text area)
- Protocolo:** HTTP (Dropdown menu)
- Puerto:** 80 (Text input)
- Protocolo de proxy:** NINGUNO (Dropdown menu)
- Ruta de la solicitud:** /test (Text input)
- ⌵ Más** (Expandable section)
- Criterios de buen estado:**
 - Definen cómo se determina un buen estado: con cuánta frecuencia verificar, cuánto tiempo esperar una respuesta y cuántos intentos exitosos o con errores son decisivos
 - Intervalo de verificación:** 10 segundos
 - Tiempo de espera:** 5 segundos
 - Umbral de buen estado:** 2 resultados correctos consecutivos
 - Umbral de mal estado:** 3 errores consecutivos

Si usa grupos de instancias para su servicio, debe crear una verificación de estado a fin de habilitar la reparación automática.

La verificación de estado es un extremo de prueba en su servicio. Debe indicar que su servicio está disponible y listo para aceptar solicitudes y no solo que el servidor se está ejecutando. Una dificultad al momento de crear un buen extremo de verificación de estado es que, si usa otros servicios de backend, debe verificar que están disponibles a fin de entregar la confirmación positiva de que su servicio está listo para ejecutarse. Si los servicios de los que depende no están disponibles, su servicio tampoco debe estarlo.

Si una verificación de estado falla en el grupo de instancias, se quitará la instancia con errores y se creará una nueva. Las verificaciones de estado se pueden usar en los balanceadores de cargas para determinar a qué instancias enviar solicitudes.

Use buckets de almacenamiento multirregionales para obtener alta disponibilidad si el impacto en la latencia es insignificante

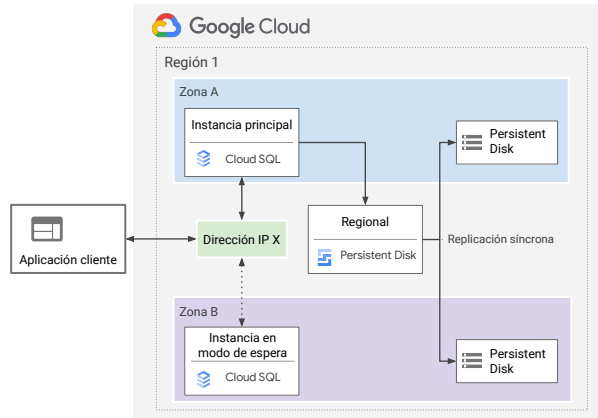
Tipo de bucket	Disponibilidad	Precio (us-central1)
Multirregional	99.95%	\$0.026 por GB
Unirregional	99.90%	\$0.020 por GB
Birregional	99.95%	\$0.020 por GB

Analicemos cómo obtener alta disponibilidad para los servicios de bases de datos y almacenamiento de datos de Google Cloud.

En Cloud Storage, puede obtener alta disponibilidad con buckets de almacenamiento multirregionales si el impacto en la latencia es insignificante. Como se ilustra en esta tabla, el beneficio de la disponibilidad multirregional es un factor de dos a medida que disminuye la falta de disponibilidad de un 0.1% a un 0.05%.

Si usa Cloud SQL, cree una réplica de conmutación por error para obtener alta disponibilidad

- La réplica se creará en otra zona ubicada en la misma región que la base de datos.
- Cambiará automáticamente a la de conmutación por error si la instancia primaria no está disponible.
- Duplica el costo de la base de datos.



Si está usando Cloud SQL y necesita alta disponibilidad, puede crear una réplica de conmutación por error. En este gráfico, se muestra la configuración en la que se configura una instancia primaria en una zona y se crea una réplica en otra, pero en la misma región. Si la instancia primaria no está disponible, la conmutación por error se cambiará automáticamente para controlar la instancia primaria. Recuerde que pagará por la instancia adicional con este diseño.

Spanner y Firestore se pueden implementar en 1 o varias regiones

Base de datos	ANS de disponibilidad
Firestore unirregional	99.99%
Firestore multirregional	99.999%
Spanner unirregional	99.99%
Spanner multirregional (nam3)	99.999%

Firestore y Spanner ofrecen implementaciones unirregionales y multirregionales. Una ubicación multirregional es un área geográfica general, como Estados Unidos. Los datos de una ubicación multirregional se replican en varias regiones. En una región, los datos se replican en varias zonas.

Las ubicaciones multirregionales pueden soportar la pérdida de regiones completas y mantener la disponibilidad sin perder datos. La configuración multirregional de Firestore y Spanner ofrece disponibilidad de “cinco nueves”, que es de menos de 6 minutos de tiempo de inactividad por año.

Realizar implementaciones para garantizar la alta disponibilidad aumenta los costos

Debe realizar un análisis de costos y riesgos.

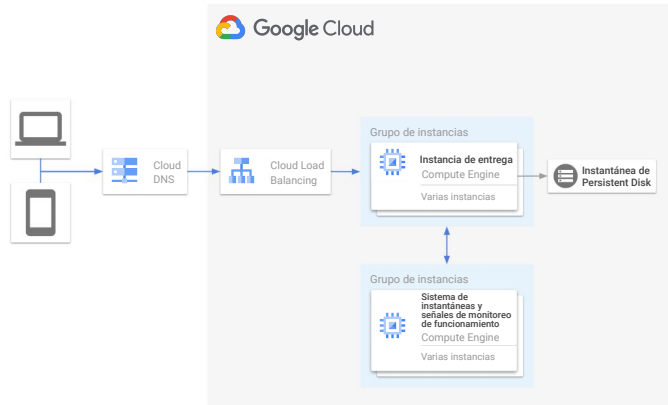
Implementación	Costo estimado	% de disponibilidad	Costo de inactividad
Zona única			
Varias zonas en una región			
Varias regiones			

Mencionamos que realizar implementaciones para garantizar la alta disponibilidad aumenta los costos porque se usan recursos adicionales.

Es importante que considere los costos de sus decisiones de arquitectura como parte del proceso de diseño. No solo hay que estimar el costo de los recursos que se usan, sino que también hay que considerar el costo de la inactividad de su servicio. Esta tabla es una forma muy eficaz de evaluar el riesgo frente al costo considerando diferentes opciones de implementación y equilibrándolas frente al costo de la inactividad.

Recuperación ante desastres: Espera pasiva

- Cree instantáneas, imágenes de máquina y copias de seguridad de datos en almacenamiento multirregional.
- Si falla la región principal, inicie servidores en la región de copia de seguridad.
- Enrute las solicitudes a una nueva región.
- Documente y pruebe el procedimiento de recuperación con frecuencia.

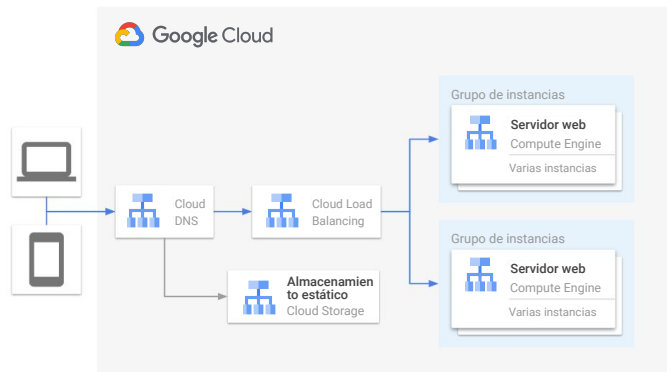


Ahora, presentaremos algunas estrategias de recuperación ante desastres.

Una estrategia de recuperación ante desastres simple puede ser una espera pasiva. Debe crear instantáneas de discos persistentes, imágenes de máquina y copias de seguridad de datos y guardarlas en el almacenamiento multirregional. En este diagrama, se muestra un sistema simple con esta estrategia. Se toman instantáneas que se pueden usar para recrear el sistema. Si falla la región principal, puede iniciar servidores en la región de copia de seguridad con imágenes de instantáneas y discos persistentes. Deberá enrutar las solicitudes a la nueva región. Es fundamental documentar y probar este procedimiento de recuperación con regularidad.

Recuperación ante desastres: Espera activa

- Cree grupos de instancias en varias regiones.
- Use un balanceador de cargas global.
- Almacene datos estructurados en buckets multirregionales.
- Para los datos estructurados, use una base de datos multirregional como Spanner o Firestore.



Otra estrategia de recuperación ante desastres es tener una espera activa en la que los grupos de instancias existen en varias regiones y el tráfico se envía a un balanceador de cargas global. En este diagrama, se muestra una configuración así.

Ya lo mencionamos, pero también puede implementarlo en servicios de almacenamiento de datos como buckets de Cloud Storage y servicios de bases de datos multirregionales como Spanner y Firestore.

Cuando planifique ante desastres, imagine situaciones que pueden provocar pérdida de datos o fallas del servicio

- ¿Qué situaciones podrían provocar fallas?
- ¿Cuál es el objetivo de punto de recuperación (cantidad de datos que sería aceptable perder)?
- ¿Cuál es el objetivo de tiempo de recuperación (cantidad de tiempo que puede tardar para que esté en funcionamiento)?

Servicio	Situación	Objetivo de punto de recuperación	Objetivo de tiempo de recuperación	Prioridad
Servicio de calificaciones de productos	El programador borró todas las calificaciones accidentalmente	24 horas	1 hora	Media
Servicio de pedidos	Falla del servidor de bases de datos	0	1 minuto	Alta

Cualquier planificación de recuperación ante desastres debe considerar sus objetivos en cuanto a dos métricas: el objetivo de punto de recuperación y el objetivo de tiempo de recuperación. El objetivo de punto de recuperación es la cantidad de datos que sería aceptable perder y el objetivo de tiempo de recuperación es la cantidad de tiempo que puede tardar para que esté en funcionamiento.

Debe imaginar situaciones que puedan provocar pérdida de datos o fallas del servicio y crear una tabla parecida a la que se muestra aquí, lo que puede ser útil para brindar estructura sobre diferentes situaciones y priorizarlas según corresponda. Creará una tabla como esta en la próxima actividad de diseño junto con un plan de recuperación.

Formule un plan de recuperación según sus situaciones de desastre

- Diseñe una estrategia de copia de seguridad basada en el riesgo, el punto de recuperación y los objetivos de tiempo.
- Informe sobre el procedimiento para la recuperación ante fallas.
- Pruebe y valide el procedimiento para recuperarse de las fallas con regularidad.
- Idealmente, la recuperación se convierte en un proceso optimizado, parte de las operaciones diarias.

Recurso	Estrategia de copia de seguridad	Ubicación de la copia de seguridad	Procedimiento de recuperación
Base de datos de MySQL de calificaciones	Copias de seguridad automáticas diarias	Bucket de Cloud Storage multirregional	Ejecutar secuencia de comandos de restablecimiento
Base de datos de Spanner de pedidos	Implementación multirregional	Región de copia de seguridad us-east1	Instantáneas y copias de seguridad en intervalos frecuentes, fuera de la infraestructura de entrega (p. ej., Cloud Storage)

Debe crear un plan de recuperación basado en las situaciones de desastre que usted defina.

Para cada situación, diseñe una estrategia en función del riesgo, el punto de recuperación y los objetivos de tiempo. Este no es un aspecto que olvidará después de documentarlo. Debe informar a todas las partes sobre el proceso de recuperación ante fallas.

Los procedimientos se deben probar y validar con regularidad, al menos, una vez al año. Además, idealmente, la recuperación será parte de las operaciones diarias, lo que ayuda a optimizar el proceso. En esta tabla, se ilustra la estrategia de copias de seguridad para diferentes recursos, junto con la ubicación de las copias de seguridad y el procedimiento de recuperación.

En esta vista simplificada, se ilustra el tipo de información que debe capturar.

Prepare al equipo para desastres con simulacros

Planificación

- ¿Qué puede salir mal con su sistema?
- ¿Cuáles son sus planes para resolver cada situación?
- Documente los planes.

Practique periódicamente

- Puede ser en la producción o en un entorno de pruebas, según corresponda.
- Evalúe los riesgos detenidamente.
- Equilibre con el riesgo de no conocer las debilidades de su sistema.



Antes de que pasemos a nuestra siguiente actividad de diseño, tenemos que destacar la importancia de la preparación de un equipo para los desastres por medio de simulacros.

Aquí decide lo que cree que puede salir mal en su sistema. Piense en los planes para resolver cada situación y documéntelos. Luego, practique estos planes con regularidad en un entorno de pruebas o de producción.

En cada etapa, evalúe los riesgos detenidamente y equilibre los costos de disponibilidad con el costo que genera la falta de disponibilidad. Con el costo de la falta de disponibilidad, podrá evaluar el riesgo de no conocer las debilidades del sistema.

Actividad 11: Planificación ante desastres

Consulte el cuaderno de ejercicios de Design and Process.

- Imagine situaciones de desastre.
- Cree un plan de recuperación ante desastres.



En esta actividad de diseño, imaginará situaciones de desastre para su caso de éxito y formulará un plan de recuperación ante desastres.

Actividad 11: Planificación ante desastres

Consulte el cuaderno de ejercicios de Design and Process.

- Imagine situaciones de desastre.
- Cree un plan de recuperación ante desastres.



En esta actividad de diseño...

Actividad 11: Planificación ante desastres

Consulte el cuaderno de ejercicios de Design and Process.

- Imagine situaciones de desastre.
- Cree un plan de recuperación ante desastres.



... imaginará situaciones de desastre para su caso de éxito...

Actividad 11: Planificación ante desastres

Consulte el cuaderno de ejercicios de Design and Process.

- Imagine situaciones de desastre.
- Cree un plan de recuperación ante desastres.



... y formulará un plan de recuperación ante desastres.

Servicio	Situación	Objetivo de punto de recuperación	Objetivo de tiempo de recuperación	Prioridad
Servicio de calificaciones de productos	El programador borró todas las calificación por accidente	24 horas	1 hora	Media
Servicio de pedidos	Ocurre una falla en la base de datos de pedidos	0 (no se debe perder ningún dato)	2 minutos	Alta

Aquí hay un ejemplo de la actividad. El objetivo de esta actividad es pensar en situaciones de desastre y evaluar la gravedad del impacto que tendrían en sus servicios.

El objetivo de punto de recuperación representa la cantidad máxima de datos que está dispuesto a perder. Obviamente, esto será diferente para su base de datos de pedidos, en la que no desea perder datos, y su base de datos de calificaciones de productos, en la que podría tolerar algunas pérdidas. El objetivo de tiempo de recuperación representa el tiempo que tardaría en recuperarse de un desastre. En este ejemplo, estimamos que podríamos recuperar la base de datos de servicio de calificaciones de productos desde una copia de seguridad en una hora. En el caso del servicio de pedidos, sería en 2 minutos.

Al igual que todos los aspectos de la vida, debemos establecer prioridades. Nunca hay tiempo para terminar todo, así que primero debe trabajar en los elementos que tienen la máxima prioridad.

Recurso	Estrategia de copia de seguridad	Ubicación de la copia de seguridad	Procedimiento de recuperación
Base de datos de calificaciones	Copias de seguridad automáticas diarias	Bucket de Cloud Storage multirregional	Ejecutar secuencia de comandos de restablecimiento
Base de datos de pedidos	Réplica de conmutación por error y copias de seguridad diarias	Implementación en varias zonas	Automatizado

Una vez que haya identificado estas situaciones, piense en planes para la recuperación. Estos serán diferentes en función de los objetivos de punto y tiempo de recuperación, al igual que las prioridades.

En este ejemplo, realizar copias de seguridad diarias de nuestra base de datos de calificaciones es suficiente para cumplir con los objetivos. Eso no será adecuado para la base de datos de pedidos, así que implementaremos una réplica de conmutación por error en otra zona.

Consulte las actividades 11a, b y c en su cuaderno de ejercicios de diseño a fin de documentar situaciones de desastre similares en su caso de éxito y formular un plan de recuperación ante desastres para cada una.

Revisión de la actividad 11: Planificación ante desastres

- Imagine situaciones de desastre.
- Cree un plan de recuperación ante desastres.



En esta actividad, se le solicitó que imaginara situaciones de recuperación ante desastres y planes para el caso de éxito en el que ha estado trabajando.

Servicio	Situación	Objetivo de punto de recuperación	Objetivo de tiempo de recuperación	Prioridad
Firestore del inventario	El programador borró todo el inventario por accidente	1 hora	1 hora	Alta
Base de datos de pedidos de Cloud SQL	La base de datos de pedidos falló	0 minutos	5 minutos	Alta
Base de datos de análisis de BigQuery	El usuario borró las tablas	0 minutos	24 horas	Media

Este es un ejemplo de algunas situaciones de desastre posibles para TurisClic, nuestro portal de viajes en línea.

Cada servicio utiliza una base de datos diferente y tiene objetivos y prioridades diferentes. Todo esto afecta la forma en que diseñamos nuestros planes de recuperación ante desastres.

Recurso	Estrategia de copia de seguridad	Ubicación de la copia de seguridad	Procedimiento de recuperación
Firestore del inventario	Copias de seguridad automáticas diarias	Bucket de Cloud Storage multirregional	Cloud Functions y Cloud Scheduler
Base de datos de pedidos de Cloud SQL	Copias de seguridad y registro de objetos binarios Réplica de conmutación por error en otra zona	N/A	Conmutación por error automática Ejecutar una secuencia de comandos de copia de seguridad si es necesario
Base de datos de análisis de BigQuery	No se requiere una copia de seguridad específica	N/A	Volver a importar los datos para reconstruir las tablas de análisis

Nuestro servicio de análisis alojado en BigQuery tiene la prioridad más baja, por lo que deberíamos poder volver a importar los datos para reconstruir las tablas de análisis si un usuario las borra.

Nuestro servicio de pedidos no puede admitir la pérdida de datos y debe volver a funcionar y ejecutarse casi inmediatamente. Para hacerlo, necesitamos una réplica de conmutación por error además de las copias de seguridad automáticas y el registro de objetos binarios.

Nuestro servicio de inventario utiliza Firestore, para el que podemos implementar copias de seguridad automáticas diarias en un bucket de Cloud Storage multirregional. Cloud Functions y Cloud Scheduler pueden facilitar el procedimiento de recuperación.

Repaso

Diseñe sistemas
confiables

En este módulo, explicamos cómo implementar nuestras aplicaciones para garantizar alta disponibilidad, durabilidad y escalabilidad. Para la alta disponibilidad, podemos implementar nuestros recursos en varias zonas y regiones. Para la durabilidad, podemos guardar varias copias de datos y realizar copias de seguridad con regularidad. En el caso de la escalabilidad, podemos implementar varias instancias de nuestros servicios y configurar escaladores automáticos.

También presentamos la planificación de recuperación ante desastres, que se define como la creación de situaciones en las que podría perder datos y la implementación de un plan de recuperación en caso de que ocurra un desastre.

Cuestionario

Una el término a su definición.

Disponibilidad

Es la probabilidad de no perder datos debido a una falla del hardware o del sistema.

Durabilidad

Es la capacidad de un sistema para seguir funcionando cuando aumenta la cantidad de usuarios y datos.

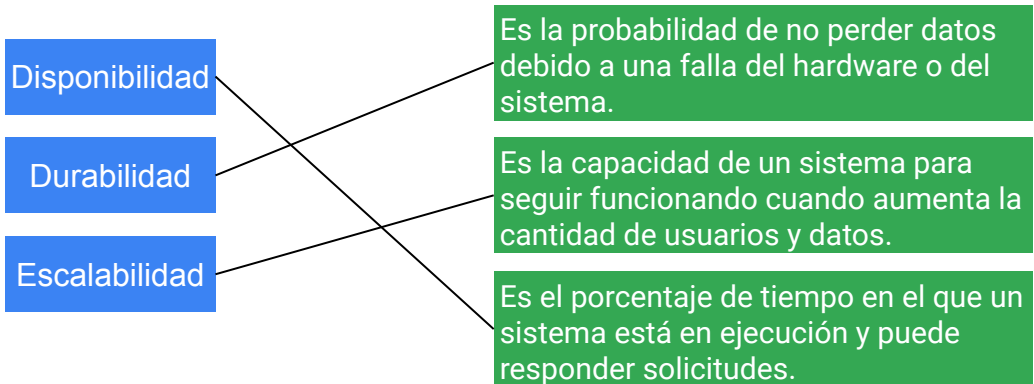
Escalabilidad

Es el porcentaje de tiempo en el que un sistema está en ejecución y puede responder solicitudes.

Una los términos disponibilidad, durabilidad y escalabilidad con sus definiciones.

Cuestionario

Una el término a su definición.



Estas son una verificación de las definiciones que presentamos en la diapositiva 4.

La disponibilidad es el porcentaje de tiempo en el que un sistema está en ejecución y puede responder solicitudes.

La durabilidad es la probabilidad de no perder datos debido a una falla del hardware o del sistema.

La escalabilidad es la capacidad de un sistema para seguir funcionando cuando aumenta la cantidad de usuarios y datos.

Cuestionario

¿Por qué no diseñaría todos los sistemas para garantizar la máxima disponibilidad?

¿Por qué no diseñaría todos los sistemas para garantizar la máxima disponibilidad?

Cuestionario

¿Por qué no diseñaría todos los sistemas para garantizar la máxima disponibilidad?

Una mayor disponibilidad aumenta los costos del sistema. Debe realizar un análisis de riesgos y costos a fin de determinar qué es lo adecuado para cada servicio.

No todos los sistemas necesitan alta disponibilidad. Un tiempo de inactividad puede ser aceptable y, si se puede lograr un tiempo de recuperación rápido, el beneficio económico que conlleva optimizar el tiempo de recuperación puede ser más beneficioso que la alta disponibilidad. La alta disponibilidad tiene un costo y, cuando se vuelve más alta, implica un mayor costo debido a los recursos adicionales y la mayor complejidad del software.

Mientras más disponibilidad, más costoso se vuelve un sistema. Debe realizar un análisis de riesgos y costos a fin de determinar qué es lo adecuado para cada servicio.

Cuestionario

Necesita una base de datos relacional para un sistema que requiere una disponibilidad extremadamente alta (99.999%). El sistema se debe ejecutar sin interrupciones incluso si ocurre una interrupción regional. ¿Qué base de datos elegiría?

- A. Cloud SQL
- B. Firestore
- C. Spanner
- D. BigQuery

Necesita una base de datos relacional para un sistema que requiere una disponibilidad extremadamente alta (99.999%). El sistema se debe ejecutar sin interrupciones incluso si ocurre una interrupción regional. ¿Qué base de datos elegiría?

- A. Cloud SQL
- B. Firestore
- C. Spanner
- D. BigQuery

Cuestionario

Necesita una base de datos relacional para un sistema que requiere una disponibilidad extremadamente alta (99.999%). El sistema se debe ejecutar sin interrupciones incluso si ocurre una interrupción regional. ¿Qué base de datos elegiría?

- A. Cloud SQL
- B. Firestore
- C. Spanner
- D. BigQuery

- A. Incorrecto. Cloud SQL proporciona una base de datos relacional y se puede configurar para que tenga alta disponibilidad. Sin embargo, incluso con esa opción de configuración, una interrupción regional causará cierto tiempo de inactividad porque la alta disponibilidad no es global.
- B. Incorrecto. Firestore no es una base de datos relacional.
- C. Correcto. Cloud Spanner cumple con todos los requisitos. Esta es una base de datos relacional global con alta disponibilidad. Las instancias multirregionales tienen un tiempo de actividad mensual igual o superior al 99.999%.
- D. Incorrecto. BigQuery no es una base de datos relacional.

C es la respuesta correcta. Cloud Spanner cumple con todos los requisitos. Esta es una base de datos relacional global con alta disponibilidad. Las instancias multirregionales tienen un tiempo de actividad mensual igual o superior al 99.999%.

La respuesta A no es correcta. Cloud SQL proporciona una base de datos relacional y se puede configurar para que tenga alta disponibilidad. Sin embargo, incluso con esa opción de configuración, una interrupción regional causará cierto tiempo de inactividad porque la alta disponibilidad no es global.

La respuesta B no es correcta. Firestore no es una base de datos relacional.

La respuesta D no es correcta. BigQuery no es una base de datos relacional.

Cuestionario

Está creando un servicio y quiere protegerlo para que no se sobrecargue si ocurre una interrupción parcial y los clientes realizan demasiados reintentos. ¿Cuál patrón de diseño implementaría?

- A. Disyuntor
- B. Retirada exponencial truncada
- C. Rechazo de retroalimentación por sobrecarga
- D. Almacenamiento en caché diferido

Está creando un servicio y quiere protegerlo para que no se sobrecargue si ocurre una interrupción parcial y los clientes realizan demasiados reintentos. ¿Cuál patrón de diseño implementaría?

- A. Disyuntor
- B. Retirada exponencial truncada
- C. Rechazo de retroalimentación por sobrecarga
- D. Almacenamiento en caché diferido

Cuestionario

Está creando un servicio y quiere protegerlo para que no se sobrecargue si ocurre una interrupción parcial y los clientes realizan demasiados reintentos. ¿Cuál patrón de diseño implementaría?

A. Disyuntor

B. Retirada exponencial truncada

C. Rechazo de retroalimentación por sobrecarga

D. Almacenamiento en caché diferido

- A. Correcto. El disyuntor intentará impedir la ejecución de una operación que posiblemente falle y, de esta forma, protegerá el recurso que se encuentra interrumpido de forma parcial para prevenir fallas en cascada.
- B. Incorrecto. La retirada exponencial truncada intentará ejecutar un reintento con la expectativa de que probablemente funcione. Esto causará problemas si hay una interrupción parcial porque el reintento no tendrá éxito.
- C. Incorrecto. El rechazo de retroalimentación por sobrecarga consiste en reducir la carga para evitar que se sobrecarguen los servicios. Por lo general, es una estrategia para el balanceo de cargas.
- D. Incorrecto. El almacenamiento en caché reduce la carga, pero no existe un control sobre la expulsión o el reabastecimiento de la caché, y es posible que no se reduzca la carga cuando se necesita.

A es la respuesta correcta. El disyuntor intentará impedir la ejecución de una operación que posiblemente falle y, de esta forma, protegerá el recurso que se encuentra interrumpido de forma parcial para prevenir fallas en cascada.

La respuesta B no es correcta. La retirada exponencial truncada intentará ejecutar un reintento con la expectativa de que probablemente funcione. Esto causará problemas si hay una interrupción parcial porque el reintento no tendrá éxito.

La respuesta C no es correcta. El rechazo de retroalimentación por sobrecarga consiste en reducir la carga para evitar que se sobrecarguen los servicios. Por lo general, es una estrategia para el balanceo de cargas.

La respuesta D no es correcta. El almacenamiento en caché reduce la carga, pero no existe un control sobre la expulsión o el reabastecimiento de la caché, y es posible que no se reduzca la carga cuando se necesita.

Más recursos

Guía de planificación para la recuperación ante desastres

<https://cloud.google.com/solutions/dr-scenarios-planning-guide>

En este vínculo, puede acceder a recursos útiles sobre planificación de recuperación ante desastres.