Engineer

Alta disponibilidad

Capacidad de seguir funcionando cuando falla un componente

Qué es

La alta disponibilidad es la propiedad de un servicio que sigue respondiendo cuando falla una de las máquinas, discos, conexiones de red o procesos que lo sostienen. Suele construirse como un clúster: varios hosts capaces de ejecutar el servicio, software que los vigila y un procedimiento que traslada el trabajo de un host averiado a uno sano. El proyecto ClusterLabs, que mantiene la pila de clústeres de código abierto Pacemaker y Corosync, describe un clúster de este tipo como un sistema que gestiona servicios en varios hosts, desde arrancarlos y comprobar su salud hasta recuperarlos tras un fallo, «para minimizar el tiempo de inactividad preservando la integridad de los datos».

La segunda mitad de esa frase importa tanto como la primera. Un servicio que vuelve pronto con datos incorrectos no se ha vuelto altamente disponible. Se ha vuelto peligroso.

La documentación de PostgreSQL separa dos razones para ejecutar varios servidores de bases de datos juntos. Una es que un segundo servidor tome el relevo con rapidez si falla el principal, lo que llama alta disponibilidad. La otra es que varios equipos sirvan los mismos datos, lo que llama balanceo de carga. A menudo comparten mecanismos, pero responden a preguntas distintas: la alta disponibilidad trata de sobrevivir a un fallo; el balanceo de carga y la escalabilidad, de soportar más trabajo.

La disponibilidad misma es un número. La medida tradicional se basa en el tiempo: la proporción de un período durante la que pudo usarse el servicio. El libro de SRE de Google explica que, para servicios repartidos por el mundo, donde casi siempre hay alguna parte activa en algún lugar, mide en cambio la proporción de solicitudes respondidas con éxito. Ambas se expresan en porcentajes y se describen como una cantidad de nueves.

Todo diseño de alta disponibilidad se compone de las mismas pocas piezas:

- **Redundancia.** Un repuesto para cada pieza cuyo fallo detendría el servicio: un segundo host, una base de datos en espera, una segunda ruta de red. Pacemaker admite pares activo/pasivo, disposiciones N+1 donde varios nodos activos comparten un repuesto y otras estructuras.
- **Detección.** Algo tiene que descubrir el fallo. El capítulo de PostgreSQL sobre conmutación por error describe una señal de vida entre el principal y el servidor en espera que comprueba continuamente la conexión y si el principal sigue funcionando.
- **Conmutación por error.** Trasladar el trabajo: promover el servidor en espera, mover la dirección IP del servicio, arrancar el proceso en otro host.
- **Aislamiento del nodo fallido.** Asegurar que el nodo averiado se ha detenido de verdad antes de iniciar su trabajo en otro lugar.
- **Reconstrucción del repuesto.** Tras una conmutación, el sistema funciona sin redundancia hasta que hay un repuesto nuevo.

La alta disponibilidad no es una copia de seguridad ni recuperación ante desastres. Protege frente al fallo de una pieza mientras el emplazamiento sigue funcionando en conjunto. Una copia de seguridad protege frente a la pérdida o corrupción de datos, incluida la corrupción que un clúster copia fielmente en todos los nodos, y la recuperación ante desastres es el plan para afrontar la pérdida del emplazamiento completo.

Utilice el concepto en una decisión

Aplique «¿Qué es la alta disponibilidad?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Monitorización y alertas que avisan cuando un fallo aún es pequeño, y cifras que muestran con qué frecuencia se mantiene así.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Primer panel de observabilidad de la empresa — la primera visión de la empresa sobre sus propios sistemas

Primer panel de observabilidad de la empresa Póngase en contacto →

Por qué importa

Todo servicio tiene piezas que fallarán. El hardware se rompe, los núcleos sufren errores fatales, una máquina virtual se elimina por accidente, una partición de red separa un nodo sano del resto del clúster. La documentación de Kubernetes las enumera como interrupciones involuntarias y las contrapone a las voluntarias

que alguien planifica, como evacuar un nodo para repararlo o actualizarlo. Sin redundancia, cada una es una caída, incluidas las planificadas, porque la única forma de aplicar parches a un servidor único es detenerlo.

Ese segundo punto suele ser el primer beneficio que nota un equipo. El capítulo de PostgreSQL sobre conmutación por error considera útil alternar regularmente entre principal y servidor en espera: permite detener cada máquina por turnos para mantenerla y cada cambio comprueba que el mecanismo funcionará cuando se necesite.

Los nueves se traducen en tiempo. Para un año de 365 días (525.600 minutos) y un mes de 30 días (43.200 minutos), el cálculo es:

Disponibilidad	Inactividad permitida/año	Permitida por mes de 30 días
99%	3.65 días	7.2 horas
99.9%	8.76 horas	43.2 minutos
99.95%	4.38 horas	21.6 minutos
99.99%	52.56 minutos	4.32 minutos
99.999%	unos 5.26 minutos	unos 26 segundos

Cada nueve adicional divide el margen entre diez y el coste crece aún más deprisa. El libro de SRE advierte que un paso más en fiabilidad puede costar cien veces lo que el anterior y que, pasado cierto punto, los usuarios no distinguen la diferencia: quien usa un teléfono que solo es fiable al 99 % no notará si el servicio de detrás funciona al 99,99 % o al 99,999 %. Por eso el libro trata el objetivo de disponibilidad como un mínimo y también un máximo. Un servicio mucho más disponible de lo necesario ha gastado dinero y tiempo de ingeniería que podrían haberse dedicado a otra cosa.

Por eso el objetivo es una decisión de negocio que se toma servicio por servicio. Un objetivo de nivel de servicio deja escrito el número; la alta disponibilidad es la ingeniería que permite alcanzarlo.

Cómo se aplica en la práctica

Tomemos un servicio de reservas inventado con tres capas: una web, una de aplicación y una base de datos PostgreSQL, cada una en su propio host. Todos los números siguientes son inventados para el ejemplo. Si cada capa está activa el

99,9 % del tiempo y el servicio necesita las tres, sus disponibilidades se multiplican. Duplicar una capa funciona al revés, porque ambas copias tienen que estar caídas a la vez:

```
Tres capas en serie, cada una disponible al 99.9%:
  0.999 × 0.999 × 0.999 ≈ 0.997      unas 26 horas de inactividad/año

Una capa duplicada, si sus dos copias fallan de forma independiente:
  1 − 0.001 × 0.001 = 0.999999       cerca de medio minuto al año
```

El segundo cálculo descansa en una palabra: independientemente. Dos copias en el mismo host, el mismo bastidor, detrás del mismo conmutador o en la misma zona de nube comparten destino y el cálculo deja de valer. La fórmula presupone también que el cambio de una copia a otra es instantáneo y siempre funciona, algo que solo las pruebas pueden demostrar.

En Kubernetes, para eso sirven las restricciones de distribución topológica. El caso que motiva la propia documentación es una carga de trabajo que puede reducirse a dos Pods: no conviene que ambos estén en el mismo nodo, donde un fallo dejaría la carga fuera de servicio. Una restricción en la plantilla del Pod reparte las réplicas entre zonas:

```
topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: topology.kubernetes.io/zone
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
          matchLabels:
              app: booking
```

Con `DoNotSchedule`, el planificador deja un Pod sin programar antes que situarlo donde el número de Pods entre zonas difiera en más de uno. El mantenimiento planificado necesita otra protección. Un PodDisruptionBudget limita cuántas réplicas de una aplicación pueden estar caídas a la vez por interrupciones voluntarias. `kubectl drain` respeta ese presupuesto porque pide a la Eviction API desalojar cada Pod; eliminarlos directamente lo elude:

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
    name: booking-pdb
spec:
    maxUnavailable: 1
    selector:
        matchLabels:
            app: booking
```

La documentación recomienda `maxUnavailable` frente a `minAvailable` porque se adapta automáticamente a los cambios en el número de réplicas.

La base de datos es la capa más difícil porque conserva estado. PostgreSQL puede mantener un servidor en espera en caliente (hot standby), que acepta consultas de solo lectura, o uno en espera templada (warm standby), que solo empieza a aceptar conexiones tras promoverse. Elegir entre replicación síncrona y asíncrona es elegir qué puede perder una conmutación. Con replicación síncrona, la confirmación del commit espera al número configurado de servidores síncronos en espera y al nivel de confirmación configurado. Los ajustes `synchronous_standby_names` y `synchronous_commit` definen esas condiciones. Una conmutación debe elegir un servidor en espera que tenga los datos confirmados; uno asíncrono aún puede ir por detrás. El precio de la primera es la velocidad: la documentación señala que conexiones de red lentas pueden hacer que una configuración totalmente síncrona pierda más de la mitad de su rendimiento.

PostgreSQL no decide cuándo conmutar. Su documentación dice que no proporciona software de sistema para detectar un principal averiado y avisar al servidor en espera; esa tarea, junto con mover la dirección IP, corresponde a software externo de clúster. La promoción misma es un solo paso, `pg_ctl promote` o la función `pg_promote()`. Lo que sigue es la parte que los planes olvidan. Sin el principal antiguo, el clúster funciona con un servidor, lo que la documentación llama un estado degenerado, hasta construir otro en espera. La utilidad `pg_rewind` puede acelerar esa tarea en clústeres grandes.

En este sitio, la entrada sobre correo con conmutación por error y registro de entregas muestra el mismo patrón aplicado a una dependencia externa. Tres proveedores de correo están detrás de una interfaz; la conmutación es automática, en lugar de un cambio de configuración bajo presión, y una comprobación de inicio envía un mensaje real, de modo que preproducción y producción se niegan a arrancar sin un proveedor que funcione. La entrada

informa de que una mala tarde de un proveedor ahora supone una vía degradada en vez de una caída, a cambio de mantener tres integraciones en funcionamiento. Cuánta disponibilidad necesita un sistema es una decisión propia: la página de niveles de servicio explica que la cifra se acuerda por separado para cada sistema, porque un informe nocturno y un servicio de pagos requieren cifras distintas.

Dónde surgen los problemas

Redundancia con un destino compartido. Dos copias en un host, una fuente de alimentación, un conmutador o una zona de nube fallan juntas. Siga las dependencias de cada repuesto y reparta las réplicas entre dominios de fallo deliberadamente, no por suerte.

Cerebro dividido. Si los nodos pierden contacto pero siguen funcionando, cada lado puede decidir que el otro ha muerto y arrancar el servicio por su cuenta. La documentación de Pacemaker detalla el resultado: activar la misma IP en ambos hosts vuelve impredecible el destino de los paquetes entrantes y, en una base de datos, puede corromper los datos o hacerlos divergir. La protección es el aislamiento del nodo fallido, también llamado STONITH (Shoot The Other Node In The Head): un dispositivo de aislamiento, como un interruptor de alimentación controlado por el clúster, desconecta el nodo averiado antes de iniciar su trabajo en otro lugar. La documentación establece que un clúster no puede recuperarse con seguridad de un nodo que no responde sin ese mecanismo, y que el dispositivo no debe depender de la propia red del clúster; de lo contrario, perderla también elimina la forma de aislarlo.

Una conmutación que nadie ha probado. Un servidor en espera que nunca se ha promovido es una esperanza, no un plan. Alterne los roles según un calendario, como sugiere PostgreSQL, documente el procedimiento y ensáyelo como la ingeniería del caos ensaya otros fallos.

Olvidar el estado degenerado. Tras una conmutación no hay repuesto. Si reconstruir el servidor en espera es lento o manual, el siguiente fallo, por pequeño que sea, se convierte en una caída. Trate la recuperación de la redundancia como parte del incidente, no como una tarea para después.

Suponer que la replicación asíncrona no pierde nada. Puede perder las últimas transacciones confirmadas antes del fallo. Decida para cada base de datos si eso es aceptable y haga visible el compromiso a los propietarios de los datos.

Confundir replicación con copia de seguridad. Un servidor en espera copia un `DELETE` equivocado con la misma fidelidad que una escritura correcta. La alta disponibilidad mantiene el servicio activo; solo una copia de seguridad permite volver a los datos tal como estaban.

Dos nodos y ningún árbitro. Pacemaker señala que perder un nodo por un corte eléctrico parece lo mismo que perder la conexión de red con él. Con dos miembros, cada lado ve exactamente eso. Kubernetes pide al menos tres nodos de plano de control en su disposición apilada, y PostgreSQL menciona un tercer servidor testigo que puede evitar algunas conmutaciones incorrectas, pero advierte que la complejidad adicional solo merece la pena con una configuración cuidadosa y pruebas rigurosas.

Pagar por nueves que nadie puede ver. Cada nueve adicional cuesta más que el anterior y los usuarios llegan al servicio a través de redes y dispositivos menos fiables que el objetivo. Elija la cifra a partir de las necesidades de usuarios y negocio y construya para alcanzarla.

Lecturas adicionales

- ClusterLabs — La pila de clústeres de alta disponibilidad de código abierto, su definición de lo que coordina un clúster y por qué necesita aislar nodos.
- Alta disponibilidad, balanceo de carga y replicación — El capítulo de PostgreSQL sobre servidores principales y en espera, y sobre diseños síncronos y asíncronos y sus costes.
- Conmutación por error — PostgreSQL sobre señales de vida, servidores testigo, STONITH, promoción y estado degenerado.
- Aislamiento de nodos — Pacemaker Explained sobre cerebro dividido, dispositivos de aislamiento y su relación con el cuórum.
- Aceptar el riesgo — El libro de SRE de Google sobre medir la disponibilidad, el coste de cada nueve y elegir un objetivo.
- Interrupciones — Interrupciones voluntarias e involuntarias en Kubernetes y cómo las limita un PodDisruptionBudget.



Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la alta disponibilidad?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/high-availability/