Engineer

Haute disponibilité

Capacité de rester en service malgré la panne d'un élément

Ce que c’est

La haute disponibilité est la propriété d’un service qui continue de répondre lorsqu’un des éléments qui le soutiennent — machine, disque, liaison réseau ou processus — tombe en panne. Elle repose généralement sur un cluster : plusieurs hôtes capables d’exécuter le service, un logiciel qui les surveille et une procédure transférant le travail d’un hôte défaillant vers un hôte sain. Le projet ClusterLabs, qui maintient la pile logicielle de cluster open source Pacemaker et Corosync, décrit un tel cluster comme un système gérant les services sur plusieurs hôtes, de leur démarrage et du contrôle de leur santé à leur récupération après panne, « pour minimiser l’indisponibilité tout en préservant l’intégrité des données ».

La seconde moitié de cette phrase compte autant que la première. Un service qui revient vite avec des données erronées n’a pas été rendu hautement disponible. Il a été rendu dangereux.

La documentation de PostgreSQL distingue deux raisons de faire fonctionner plusieurs serveurs de bases de données ensemble. L’une est de permettre à un second serveur de prendre rapidement le relais si le principal tombe en panne : c’est la haute disponibilité. L’autre est de permettre à plusieurs machines de servir les mêmes données : c’est la répartition de charge. Les deux partagent souvent des mécanismes, mais répondent à des questions différentes : la haute disponibilité vise à survivre à une panne, la répartition de charge et la capacité de montée en charge à absorber davantage de travail.

La disponibilité elle-même est un nombre. La mesure traditionnelle est temporelle : la part d’une période pendant laquelle le service était utilisable. Le livre SRE de Google explique que, pour les services répartis dans le monde, dont

une partie fonctionne presque toujours quelque part, il mesure plutôt la part des requêtes auxquelles il est répondu avec succès. Les deux s'écrivent en pourcentages et se décrivent par un nombre de neuf.

Toute conception hautement disponible assemble les mêmes quelques éléments :

- **Redondance.** Un élément de secours pour chaque composant dont la panne arrêterait le service : second hôte, base de secours, second chemin réseau. Pacemaker accepte les paires actif/passif, les organisations N+1 où plusieurs nœuds actifs partagent un secours, et d'autres dispositions.
- **Détection.** Quelque chose doit remarquer la panne. Le chapitre de PostgreSQL sur la bascule décrit un signal de vie entre serveur principal et serveur de secours, vérifiant continuellement la connexion et le fonctionnement du principal.
- **Bascule.** Déplacer le travail : promouvoir le serveur de secours, déplacer l'adresse IP du service, démarrer le processus sur un autre hôte.
- **Isolement du nœud défaillant.** S'assurer qu'il s'est réellement arrêté avant de reprendre son travail ailleurs.
- **Reconstruction du secours.** Après une bascule, le système fonctionne sans redondance jusqu'à la mise en place d'un nouveau secours.

La haute disponibilité n'est ni une sauvegarde ni une reprise après sinistre. Elle protège de la panne d'un élément tandis que le site dans son ensemble fonctionne encore. Une sauvegarde protège de la perte ou de la corruption des données, y compris d'une corruption qu'un cluster recopie fidèlement sur chaque nœud ; la reprise après sinistre est le plan pour affronter la perte du site entier.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la haute disponibilité ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Une supervision et des alertes qui signalent les incidents tant qu'ils restent mineurs, avec les chiffres qui montrent à quelle fréquence ils le restent.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Premier tableau de bord d'observabilité — première vue de l'entreprise sur ses propres systèmes

Premier tableau de bord d'observabilité Contactez-nous →

Pourquoi c’est important

Chaque service a des éléments qui tomberont en panne. Le matériel casse, le noyau subit une défaillance fatale, une machine virtuelle est supprimée par erreur, une partition réseau isole un nœud sain du reste du cluster. Kubernetes les classe parmi les perturbations involontaires, à côté des perturbations volontaires planifiées, comme évacuer un nœud pour le réparer ou le mettre à jour. Sans redondance, chacune est une interruption de service, même planifiée, car appliquer un correctif à un serveur unique impose de l’arrêter.

Ce second point est souvent le premier bénéfice remarqué par l’équipe. Le chapitre de PostgreSQL sur la bascule juge utile d’alterner régulièrement principal et secours : chaque machine peut être arrêtée à son tour pour maintenance, et chaque échange teste le mécanisme qui devra fonctionner au besoin.

Les neuf se traduisent en temps. Pour une année de 365 jours (525 600 minutes) et un mois de 30 jours (43 200 minutes), le calcul est le suivant :

```
Disponibilité  Indisponibilité admise/an  Admise par mois de 30 jours
99%            3.65 jours                  7.2 heures
99.9%          8.76 heures                 43.2 minutes
99.95%         4.38 heures                 21.6 minutes
99.99%         52.56 minutes              4.32 minutes
99.999%        environ 5.26 minutes       environ 26 secondes
```

Chaque neuf supplémentaire divise la marge par dix, et le coût augmente encore plus vite. Le livre SRE prévient qu’un degré supplémentaire de fiabilité peut coûter cent fois plus que le précédent et qu’au-delà d’un certain point, les utilisateurs ne voient plus la différence : une personne sur un téléphone fiable à seulement 99 % ne remarquera pas si le service derrière fonctionne à 99,99 % ou à 99,999 %. Le livre traite donc la cible de disponibilité comme un plancher et un plafond. Un service beaucoup plus disponible que nécessaire a consommé de l’argent et du temps d’ingénierie qui auraient pu être consacrés ailleurs.

C’est pourquoi la cible est une décision métier, prise service par service. Un objectif de niveau de service consigne ce chiffre ; la haute disponibilité est l’ingénierie qui le rend atteignable.

À quoi cela ressemble en pratique

Prenons un service de réservation fictif à trois couches : une couche web, une couche applicative et une base PostgreSQL, chacune sur son propre hôte. Tous les nombres ci-dessous sont inventés pour l'exemple. Si chaque couche fonctionne 99,9 % du temps et que le service a besoin des trois, leurs disponibilités se multiplient. Doubler une couche fonctionne en sens inverse, car les deux copies doivent être en panne simultanément :

```
Trois couches en série, chacune disponible à 99.9% :
  0.999 × 0.999 × 0.999 ≈ 0.997      environ 26 heures d'arrêt/an

Une couche doublée, si ses deux copies tombent en panne indépendamment :
  1 − 0.001 × 0.001 = 0.999999       environ une demi-minute par an
```

Le second calcul repose sur un mot : indépendamment. Deux copies sur le même hôte, dans le même rack, derrière le même commutateur ou dans la même zone cloud partagent un destin, et le calcul ne s'applique plus. La formule suppose aussi que le passage d'une copie à l'autre est instantané et fonctionne toujours, ce que seuls les tests peuvent montrer.

Dans Kubernetes, c'est le rôle des contraintes de répartition topologique. L'exemple qui motive la documentation est une charge de travail pouvant se réduire à deux Pods : mieux vaut éviter qu'ils soient sur le même nœud, où une panne mettrait toute la charge hors ligne. Une contrainte dans le modèle du Pod répartit les répliques entre zones :

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

Avec `DoNotSchedule`, l'ordonnanceur laisse un Pod non planifié plutôt que de le placer là où le nombre de Pods entre zones différerait de plus d'un. La maintenance planifiée a besoin d'une seconde protection. Un PodDisruptionBudget limite le nombre de répliques d'une application pouvant

être indisponibles en même temps du fait de perturbations volontaires. `kubectl drain` respecte ce budget, car il demande à l'Eviction API d'évincer chaque Pod ; supprimer directement les Pods le contourne :

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

La documentation recommande `maxUnavailable` plutôt que `minAvailable`, car il suit automatiquement les changements du nombre de répliques.

La base de données est la couche la plus difficile, car elle conserve un état. PostgreSQL peut maintenir un serveur de secours à chaud (hot standby), qui accepte les requêtes en lecture seule, ou un serveur en attente (warm standby) qui n'accepte les connexions qu'après promotion. Choisir entre réplication synchrone et asynchrone revient à choisir ce qu'une bascule peut perdre. Avec la réplication synchrone, l'accusé de réception du commit attend le nombre configuré de serveurs de secours synchrones, au niveau d'accusé de réception configuré. Les paramètres `synchronous_standby_names` et `synchronous_commit` définissent ces conditions. Une bascule doit choisir un serveur de secours qui possède les données acquittées ; un serveur de secours asynchrone peut encore avoir du retard. Le prix de la première option est la vitesse : la documentation note que des liaisons lentes peuvent faire perdre plus de la moitié des performances à une configuration entièrement synchrone.

PostgreSQL ne décide pas quand basculer. Sa documentation indique qu'il ne fournit aucun logiciel système pour détecter un principal défaillant et prévenir le secours ; ce travail, comme le déplacement de l'adresse IP, appartient à un logiciel externe de cluster. La promotion tient en une étape, `pg_ctl promote` ou la fonction `pg_promote()`. La suite est la partie oubliée des plans. L'ancien principal disparu, le cluster ne fonctionne plus que sur un serveur, état qualifié de dégénéré par la documentation, jusqu'à la construction d'un nouveau secours. L'utilitaire `pg_rewind` peut accélérer cela sur de grands clusters.

Sur ce site, l'entrée sur le système de courriel avec bascule et journalisation des livraisons montre le même schéma appliqué à une dépendance extérieure. Trois fournisseurs de courriel se trouvent derrière une interface, la bascule est automatique plutôt qu'un changement de configuration sous pression, et un contrôle de démarrage envoie un vrai message : préproduction et production refusent donc de démarrer sans fournisseur fonctionnel. L'entrée rapporte qu'un fournisseur en difficulté pendant un après-midi ne provoque plus qu'un acheminement dégradé, plutôt qu'une panne, au prix de trois intégrations à maintenir. Le niveau de disponibilité requis est une décision à part entière : la page des niveaux de service explique que le chiffre est inscrit séparément dans l'accord de chaque système, car un rapport produit pendant la nuit et un service de paiement exigent des chiffres différents.

Où les problèmes surviennent

Une redondance au destin commun. Deux copies sur un même hôte, une même alimentation, un même commutateur ou une même zone cloud tombent ensemble. Remontez les dépendances de chaque secours et répartissez volontairement les répliques entre domaines de panne, plutôt que de compter sur la chance.

Un cluster scindé (split brain). Si les nœuds du cluster perdent contact mais continuent de fonctionner, chaque côté peut croire l'autre mort et démarrer le service lui-même. Pacemaker en explicite le résultat : activer la même adresse IP sur les deux hôtes rend imprévisible la destination des paquets entrants ; pour une base, les dégâts peuvent être une corruption ou une divergence des données. La protection est l'isolement du nœud, aussi appelé STONITH (Shoot The Other Node In The Head) : un dispositif, tel qu'un interrupteur d'alimentation commandé par le cluster, coupe le nœud défaillant avant que son travail reprenne ailleurs. La documentation indique que, sans ce dispositif, un cluster ne peut pas se rétablir en toute sécurité lorsqu'un nœud ne répond plus, et que le dispositif ne doit pas dépendre du réseau même du cluster : perdre ce réseau supprimerait aussi le moyen d'isoler le nœud.

Une bascule que personne n'a essayée. Un secours jamais promu est un espoir, pas un plan. Échangez les rôles à intervalles planifiés, comme le suggère PostgreSQL, écrivez la procédure et répétez-la comme l'ingénierie du chaos répète d'autres pannes.

L'oubli de l'état dégénéré. Après une bascule, il n'y a plus de secours. Si sa reconstruction est lente ou manuelle, la panne suivante, aussi petite soit-elle, devient une interruption de service. Traitez le rétablissement de la redondance comme une partie de l'incident, pas comme une tâche pour plus tard.

Supposer que la réplication asynchrone ne perd rien. Elle peut perdre les dernières transactions validées avant la panne. Décidez pour chaque base si c'est acceptable, et rendez ce compromis visible aux propriétaires des données.

Prendre la réplication pour une sauvegarde. Un secours recopie un `DELETE` erroné aussi fidèlement qu'une écriture correcte. La haute disponibilité maintient le service en fonctionnement ; seule une sauvegarde permet à l'équipe de retrouver les données telles qu'elles étaient.

Deux nœuds sans arbitre. Pacemaker souligne que la perte d'un nœud après une coupure de courant ressemble à celle de sa liaison réseau. Avec deux membres, chaque côté voit exactement cela. Kubernetes demande au moins trois nœuds de plan de contrôle dans son architecture empilée, et PostgreSQL mentionne un troisième serveur témoin qui peut éviter certaines bascules erronées, tout en prévenant que cette complexité ne vaut la peine qu'avec une configuration soignée et des tests rigoureux.

Payer des neuf invisibles. Chaque neuf supplémentaire coûte davantage que le précédent, et les utilisateurs passent par des réseaux et appareils moins fiables que la cible. Choisissez le nombre selon leurs besoins et ceux de l'activité, puis construisez pour l'atteindre.

Pour aller plus loin

- ClusterLabs — Pile logicielle open source de cluster à haute disponibilité, sa définition de ce que coordonne un cluster et de la nécessité d'isoler les nœuds.
- Haute disponibilité, répartition de charge et réplication — Chapitre de PostgreSQL sur les principaux et secours, les conceptions synchrones et asynchrones et leurs coûts.
- Bascule — PostgreSQL sur les signaux de vie, les serveurs témoins, STONITH, la promotion et l'état dégénéré.
- Isolement des nœuds — Pacemaker Explained sur la scission du cluster, les dispositifs d'isolement, leur rôle et le quorum.

- Accepter le risque — Livre SRE de Google sur la mesure de disponibilité, le coût de chaque neuf et le choix d'une cible.
- Perturbations — Perturbations volontaires et involontaires dans Kubernetes et limites imposées par un PodDisruptionBudget.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que la haute disponibilité ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/high-availability/