Haute disponibilité et PostgreSQL

Haute disponibilité et PostgreSQL

TITRE : Haute disponibilité et PostgreSQL
SOUS-TITRE :

INTRODUCTION

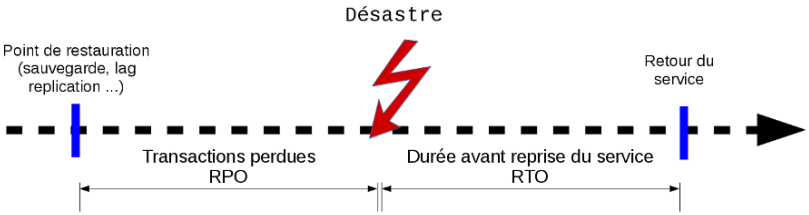
- définition de «Haute Disponibilité»
- contraintes à définir
- contraintes techniques
- solutions existantes

La haute disponibilité est un sujet complexe. Plusieurs outils libres coexistent au sein de l'écosystème PostgreSQL, chacun abordant le sujet d'une façon différente.

Ce document clarifie la définition de «haute disponibilité», des méthodes existantes et des contraintes à considérer. L'objectif est d'aider la prise de décision et le choix de la solution.

DÉFINITION DES TERMES

RTO / RPO



Deux critères essentiels permettent de contraindre le choix d'une solution: le RTO (*recovery time objectives*) et le RPO (*recovery point objective*).

Le RTO représente la durée maximale d'interruption de service admissible. Depuis la coupure de service jusqu'à son rétablissement. Il inclut le délais de détection de l'incident, le délais de prise en charge et le temps de mise en œuvre des actions correctives. Un RTO peut tendre vers 0 mais ne l'atteint jamais parfaitement. Une coupure de service est le plus souvent **inévitable**, aussi courte soit-elle.

Le RPO représente la durée maximale d'activité de production déjà réalisée que l'on s'autorise à perdre en cas d'incident. Contrairement au RTO, le RPO peut atteindre l'objectif de 0 perte.

Les deux critères sont complémentaires. Ils ont une influence **importante** sur le choix d'une solution **et** sur son coût total. Plus les RTO et RPO sont courts, plus la solution est complexe. Cette complexité se répercute directement sur le coût de mise en œuvre, de formation et de maintenance.

Le coût d'une architecture est exponentiel par rapport à sa disponibilité.

DÉFINITION DES TERMES

- haute disponibilité de service
- haute disponibilité de donnée

La **Haute Disponibilité de service** définit les moyens techniques mis en œuvre pour garantir une continuité d'activité suite à un incident sur un service.

La haute disponibilité de service nécessite de redonder tous les éléments nécessaires à l'activité du service: l'alimentation électrique, ses accès réseaux, le réseau lui-même, les serveurs, le stockage, les administrateurs, etc.

En plus de cette redondance, une technique de réplication synchrone ou asynchrone est souvent mise en œuvre afin de maintenir à l'identique ou presque les serveurs redondés.

La **Haute disponibilité des données** définit les moyens techniques mis en œuvre pour garantir une perte faible voire nulle de données en cas d'incident. Ce niveau de disponibilité des données est assuré en redondant les données sur plusieurs systèmes physiques distincts et en assurant que chaque écriture est bien réalisée sur plusieurs d'entre eux.

Dans le cas d'une réplication synchrone entre les systèmes, les écritures sont suspendues tant qu'elles ne peuvent être validées de façon fiable sur au moins deux systèmes. Autrement dit, la haute disponibilité des données et la haute disponibilité de service sont contradictoires, le premier nécessitant d'interrompre le service en écriture si l'ensemble ne repose que sur un seul système.

Par exemple, un RAID 1 fonctionnant sur un seul disque suite à un incident n'est PAS un environnement à haute disponibilité des données, mais à haute disponibilité de service.

La position du curseur entre haute disponibilité de service et la haute disponibilité de données guide aussi le choix de la solution. S'il est possible d'atteindre le double objectif, l'impact sur les solutions possibles et le coût est une fois de plus important.

SAUVEGARDES

- composant déjà présent
- travail d'optimisation à effectuer
- RTO de quelques minutes possibles
- RPO de quelques minutes facilement

Les différentes méthodes de sauvegardes de PostgreSQL sont souvent sous-estimées, c'est pourtant un élément essentiel de toute architecture qui est souvent déjà présent.

Investir dans l'optimisation des sauvegardes peut déjà assurer un certain niveau de disponibilité de votre service, à moindre coût.

Quoiqu'il en soit, la sauvegarde est un élément crucial de toute architecture. Ce sujet doit toujours faire partie de la réflexion autour de la disponibilité d'un service.

PITR

- sauvegarde incrémentale binaire
- optimiser la sauvegarde complète
- ajuster l'archivage au RPO désiré

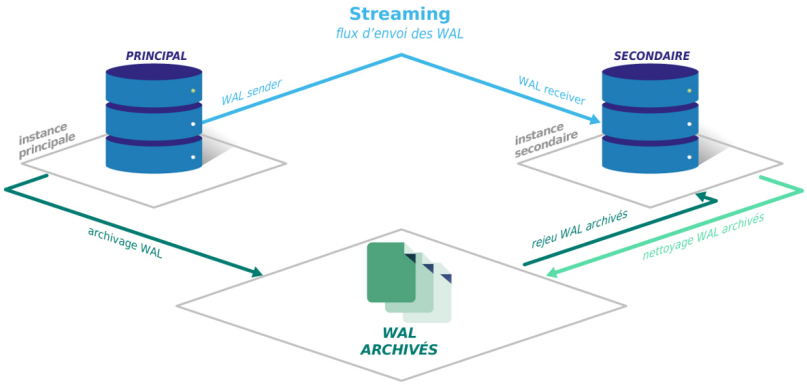
La sauvegardes PITR est une méthode permettant de restaurer une instance PostgreSQL à n'importe quel instant durant la fenêtre de rétention définie, eg. les dernières 24h. Le temps de restauration dépend de deux variables: le volume de l'instance et son volume d'écriture.

Avec le bon matériel, les bonnes pratiques et une politique de backup adaptée, il est possible d'atteindre un RTO de quelques minutes, dans la plupart des cas.

La maîtrise du RPO repose sur la fréquence d'archivage des journaux de transaction. Un RPO d'une minute est tout à fait envisageable. En dessous, nous entrons dans le domaine de la réplication de ces journaux, soit à destination des sauvegardes grâce à l'outillage fourni avec PostgreSQL (`pg_receivewal`), soit vers une instance secondaire. Nous abordons ce dernier sujet dans un futur chapitre.

PITR ET REDONDANCE RÉPLICATION

RÉPLICATION INTERNE POSTGRESQL



Enfin, il est possible d'utiliser les journaux de transaction archivés dans le cadre de la réplication physique. Ces archives deviennent alors un second canal d'échange entre l'instance primaire et ses secondaires, apportant une redondance à la réplication elle même.

BILAN PITR

- utiliser un outils libre issu de l'écosystème PostgreSQL
- fiabilise l'architecture
- facilite la mise en œuvre et l'administration
- couvre déjà certains besoins de disponibilité
- nécessite une bascule manuelle

Parmi les outils existants et éprouvés au sein de la communauté, nous pouvons citer:

- Barman
- pgBackRest
- pitrery

Le principal point faible de la sauvegarde PITR est le temps de prise en compte de

l'incident et donc d'intervention d'une administrateur.

RÉPLICATION PHYSIQUE

- réplique les écritures via les journaux de transaction
- entretient une ou plusieurs instances clone
- intégré à PostgreSQL
- facilité de mise en œuvre
- haute disponibilité des données

La réplication physique interne de PostgreSQL réplique le contenu des journaux de transaction. Les instances secondaires sont considérées comme des «clones» de l'instance primaire.

Avec peu de configuration préalable, il est possible de créer des instances secondaires directement à partir de l'instance primaire ou en restaurant une sauvegarde PITR.

La mécanique de réplication est très efficace car elle ne réplique que les modification binaires effectuées dans les tables et les index.

Cette étape permet déjà d'assurer une haute disponibilité de donnée, ces dernières étant présentes sur plusieurs serveurs distinct.

RÉPLICATION ET RPO

- réplication asynchrone ou synchrone
- nécessite un réseau très fiable et performant
- asynchrone: RPO dépendant du volume d'écriture
  - RPO < 1s hors maintenances
- synchrone: RPO = 0
  - 2 secondaires minimum
  - impact sur les performances

PostgreSQL supporte la réplication asynchrone ou synchrone.

La réplication asynchrone autorise un retard entre l'instance primaire et ses secondaires, ce qui implique un RPO supérieur à 0. Ce retard dépend directement du volume d'écriture envoyé par le primaire et de la capacité du réseau à diffuser ce volume de données, donc son débit. Une utilisation OLTP a un retard typique inférieur à la seconde. Ce retard peut cependant être plus important lors des périodes de maintenance (VACUUM, REINDEX, manipulation en masse de données, etc).

La réplication synchrone s'assure que chaque écriture soit présente sur au moins deux instances avant de valider une transaction. Ce mode permet d'atteindre un RPO de zéro,

mais impose d'avoir minimum 3 nœuds dans le cluster, autorisant ainsi la perte complète d'un serveur sans bloquer les écritures.

De plus, le nombre de transaction par seconde dépend directement de la latence du réseau: chaque transaction doit attendre la propagation vers un secondaire et le retour sa validation.

RÉPLICATION ET RTO

- bascule manuelle
- promotion d'une instance en quelques secondes

La réplication seule n'assure pas de disponibilité de service en cas d'incident.

Comme pour les sauvegardes PITR, le RTO dépend principalement du temps de prise en charge de l'incident par un opérateur. Une fois la décision prise, la promotion d'un serveur secondaire en production ne nécessite qu'une commande et ne prend que quelques secondes.

Reste ensuite à faire converger les connexions applicatives vers la nouvelle instance primaire. nous abordons ce sujet dans le chapitre Accès aux ressources.

BILAN RÉPLICATION

- $0 \leq \text{RPO} < \text{PITR}$
- RTO = prise en charge + 30s
- simple à mettre en œuvre

BASCULE AUTOMATISÉE

- détection d'anomalie et bascule automatique
- réduit le temps de prise en charge: HA de service
- plusieurs solutions en fonction du besoin
- beaucoup de contraintes

Une bascule automatique lors d'un incident permet de réduire le temps d'indisponibilité d'un service au plus bas, assurant ainsi une haute disponibilité de service.

Néanmoins, automatiser la détection d'incident et la prise de décision de basculer un service est un sujet très complexe, difficile à bien appréhender et maintenir. D'autant plus dans le domaine des SGBD.

PRISE DE DÉCISION

- la détection d'anomalie est naïve
- l'architecture doit pouvoir éviter les *split-brain*:
  - fencing: isoler un nœud ou une ressource
  - quorum: la majorité détient la ressource, la minorité l'arrête
  - watchdog: timeout reset, self-fencing

Quelque soit la solution choisie pour détecter les anomalies et déclencher une bascule, celle-ci est toujours très naïve. Contrairement à un opérateur humain, la solution n'a pas de capacité d'analyse et n'a pas accès aux mêmes informations. En cas de non réponse d'un élément du cluster, il lui est impossible de déterminer dans quel état il se trouve précisément. Sous une charge importante ? Serveur arrêté brutalement ou non ? Réseau coupé ?

Il y a une forte probabilité de se retrouver dans une situation de split-brain si le cluster se contente d'effectuer une bascule sans se préoccuper de l'ancien primaire. Dans cette situation, deux serveurs se partagent la même ressource (IP ou disque ou SGBD) sans le savoir. Il devient alors très long et fastidieux de corriger le problème et reconsolider les données. Ce qui entraîne une indisponibilité plus importante que le cas d'une simple bascule manuelle avec analyse et prise de décision humaine.

Il existe trois mécaniques complémentaires pour se prémunir des split-brain.

Le premier est le fencing (clôture) ou isolation d'incident. Suite à une anomalie, et **avant** la bascule vers le secours prévu, le composant fautif est isolé afin qu'il ne puisse plus interférer avec la production. Cette mécanique implique d'avoir un accès

d'administration à des ressources physiques (équipement réseau, UPS, PDU, SAN, IPMI, etc) ou à l'hyperviseur.

Le second est l'obtention d'un quorum avant de pouvoir effectuer toute action. Le quorum assure que seul le groupe de serveur majoritaire dans le cluster a le droit de posséder la ressource en haute disponibilité. En cas d'incident réseau sur un cluster à 3 nœuds, le serveur isolé arrête toute ressource et ne prend aucune initiative.

Le dernier est l'utilisation de watchdog, équipement désormais présent dans tous les ordinateurs, du portable ou serveur. Il peut être cependant nécessaire de l'activer explicitement à la création d'une machine virtuelle. Une fois activé, cet équipement effectue un reset de la machine si l'application chargée de le ré-armé n'est plus en capacité de le faire.

À minima, une architecture fiable peut se composer au choix:

- d'un fencing actif avec les services désactivés au démarrage
- d'un fencing et d'un quorum
- d'un watchdog par serveur et d'un quorum

IMPLICATION ET RISQUES DE LA BASCULE AUTO

- un opérateur de plus: un automate
- complexification importante
- administration importante
- formation importante
- erreurs humaines plus fréquente
- documentation et communication importante

L'ajout d'un mécanisme de bascule automatique implique quelques contraintes qu'il est important de prendre en compte lors de la prise de décision.

En premier lieu, l'automate chargé d'effectuer la bascule automatique a tout pouvoir sur vos instances PostgreSQL. Toute opération concernant vos instances de prêt ou de loin **doit** passer par lui. Il est vital que toutes les équipes soit informées de sa présence afin que toute intervention pouvant impacter le service en tienne compte (mise à jour SAN, coupure, réseau, mise à jour applicative, etc).

Ensuite, il est essentiel de construire une architecture aussi simple que possible. La complexification multiplie les chances de défaillance ou d'erreur humaine. Il est par ailleurs fréquent d'observer plus d'erreur humaine sur un cluster complexe que sur une architecture sans bascule automatique.

Pour palier à ces erreurs humaine, la formation d'une équipe est vitale. La connaissance

concernant le cluster doit être partagée par plusieurs personnes afin de toujours être en capacité d'agir en cas d'incident. Notez que même si la bascule automatique fonctionne convenablement, il est fréquent de devoir intervenir dessus dans un second temps afin de revenir à un état nominal (eg. reconstruire un nœud).

À ce propos, la documentation de l'ensemble et les procédures sont essentiels. En cas de maintenance planifiée ou d'incident, il faut être capable de réagir vite avec le moins d'improvisation possible. Quelque soit la solution choisie, assurez vous d'allouer suffisamment de temps au projet pour expérimenter, tester le cluster et le documenter.

SHARED STORAGE

- architecture simple
- redondance au niveau stokage
- cold standby matériel
- ou créer/déplacer une machine virtuelle

Une architecture de type *Shared Storage* repose essentiellement sur un disque partagé entre au moins deux serveurs. Les données de l'instance sont situées sur le disque partagé. En cas de panne sur le serveur actif, le ou les disques sont montés sur l'un des serveurs de secours et l'instance PostgreSQL y est démarrée.

L'avantage d'une telle architecture est son apparente simplicité. Les données étant habituellement situées dans un SAN, il est aisé en cas d'incident sur le serveur principal de monter les disques dans un serveur de secours et redémarrer PostgreSQL. La configuration de PostgreSQL reste au plus simple et l'espace occupé par les données de l'instance n'est pas dupliqué à travers plusieurs serveurs.

La haute disponibilité des données doit être prise en charge par une réplication au niveau disque (SAN, DRBD, etc).

La disponibilité du service dépend de la technologie utilisée: machine virtuelle, lame, serveur physique. Certaines technologies de virtualisation sont capables de "migrer" une machine virtuelle et ses disques en cas de panne sur un hyperviseur. D'autre cluster-ware (eg. Pacemaker, rgmanager) sont capable de basculer le disque et le service sur un serveur tiers du cluster.

L'un des points essentiel de cette architecture est de s'assurer que le disque ne peut être disponible que sur un seul serveur à la fois à tout instant. De façon excessivement stricte. La fiabilité des données en dépend.

L'autre architecture de clustering est appelée *Share Nothing*. Dans une telle architecture, chaque nœud du cluster est totalement indépendant. Dans le cadre de PostgreSQL, cela

signifie que les données sont situées sur plusieurs serveurs grâce à la réplication interne de ce dernier. Les solutions que nous proposons dans les prochains slides sont toutes de type Share Nothing.

PATRONI

- fiable: Quorum + Watchdog
- ne gère que PostgreSQL
- 1 cluster DCS (etcd) de 3 nœuds obligatoire
- le DCS peut gérer N cluster Patroni
- 2 nœuds PostgreSQL ou plus par cluster Patroni

Patroni assure l'exploitation d'un cluster PostgreSQL en réplication et est capable d'effectuer une bascule automatique en cas d'incident sur l'instance primaire.

Le projet repose sur un DCS externe (eg. etcd) comme stockage distribué de sa configuration et gestion du quorum. Le DCS nécessite au moins 3 nœuds pour assurer le quorum, mais un même cluster DCS peut ensuite gérer plusieurs cluster Patroni.

Se reposer sur un DCS fiable permet à Patroni plus de robustesse et de se focaliser sur l'expertise PostgreSQL uniquement. Le mécanisme de modification atomique de la base de configuration distribuée permet notamment une élection fiable en cas d'incident.

En plus du support du quorum et d'un mécanisme d'élection ne laissant aucune place aux race conditions, Patroni supporte l'utilisation d'un watchdog matériel accessible au travers du noyau Linux. Le couple quorum+watchdog permet donc d'éviter les situations de split-brain.

Patroni ne supporte pas de mécanisme de fencing. Les auteurs étudient cependant la question.

Patroni est un projet simple, rapide à déployer et prendre en main. Il est toutefois toujours hautement recommandé de former une équipe à son administration.

PACEMAKER

- référence de la HA sous Linux
- principaux contributeurs: RedHat et Suse
- empaqueté sur toutes les distributions principales
- quorum, watchdog et fencing supportés
- gère PostgreSQL et tout autre ressource
- cluster deux nœuds possible avec fencing

Pacemaker est la solution de haute disponibilité de référence sur les distributions Linux modernes. Plusieurs entreprises telle que RedHat, Suse ou Linbit investissent du temps de recherche et développement pour la maintenance et l'évolution du projet.

Le projet est très complet, supporte plusieurs types de fencing, avec escalade possible, le quorum, l'utilisation de watchdog, le clustering étendu (déployé entre deux datacenters). Il est capable de mettre en haute disponibilité plusieurs dizaines de services tels que des bases de données, des serveurs HTTP, mail, des machines virtuelles, des containers, etc.

Cette souplesse a néanmoins un prix en terme d'apprentissage et de maintenance. Il est largement recommandé d'être formé à l'utilisation et la maintenance de Pacemaker avant de le mettre en production.

Concernant la gestion de PostgreSQL, nous recommandons l'utilisation de l'agent PAF (PostgreSQL Automatic Failover). Ce dernier a été écrit afin de corriger ou contourner les problèmes et limitations de l'agent officiel existant. Il est entré au sein de l'organisme regroupant les différents projets liés à Pacemaker appelé ClusterLabs. Voir: https://clusterlabs.github.io/PAF/

ACCÈS AUX RESSOURCES

- IP virtuelle
- proxy
- intelligence dans l'application

Il existe plusieurs solutions pour gérer l'accès des applications à l'instance principale.

La plus simple est l'utilisation d'une IP virtuelle. Cette solution est à privilégier avec Pacemaker, ce dernier gérant alors à la fois la localisation de l'IP secondaire et de l'instance primaire. Cette solution est aussi envisageable avec Patroni, à condition d'utiliser un script en callback ou d'ajouter un service supplémentaire dédié à cette tâche, cette solution étant la plus recommandée.

Une autre solution est d'utiliser un tiers faisant office de proxy entre les clients et

l'instance principale. HAProxy est une solution populaire, mais il en existe bien d'autres, propriétaires ou non. La solution doit être capable de déterminer où se trouve l'instance primaire au sein du cluster.

Enfin, la dernière solution consiste à intégrer l'intelligence nécessaire directement dans les couche applicatives. La chaîne de connexion à l'instance peut par exemple comporter plusieurs destinataires et désigner le rôle recherché (`target_session_attrs=read-write`). Il est aussi possible d'utiliser un gestionnaire de configuration centralisé aux applications tel que `confd`.

CONCLUSION

	RTO	RPO	Qui	Complexité
PITR	humain	> 1 min	DBA	1
Réplication	humain	< 1 min	DBA	2
Shared disk	< 1 min	0-1 min	SYS	5
Patroni	> 5s	0-1 min	DBA	4
PAF	> 5s	0-1 min	DBA/SYS	5

La mise en œuvre de la haute disponibilité apporte un certain nombre de complications et complexifications à l'architecture. Avec une disponibilité exigée à 99.5% par an (soit 44h d'indisponibilité), nous déconseillons la mise en œuvre d'une telle architecture. Il est plus sage de reposer sur des procédures connues et éprouvées, outillées, avec une phase de prise de décision humaine optimisée pour être déclenchée le plus vite possible.

Une meilleure maîtrise de l'architecture évite souvent les appels d'astreintes impromptus et se substitue avantageusement à une bascule automatique beaucoup plus difficile à maîtriser.

Si une bascule automatisée est nécessaire, nous recommandons principalement l'utilisation de Patroni ou de Pacemaker/PAF. D'une part pour leur robustesse, d'autre part pour leur adoption importante au sein de la communauté. Évitez les solutions exotiques, les systèmes de fichiers distribués ou les projets peu populaires ou maintenus.

Quoiqu'il en soit, si vos instances sont critiques et nécessitent une haute disponibilité de service et/ou de données, il est vitale de porter un soin particulier:

- à la supervision
- aux sauvegardes
- à la formation des équipes
- à la documentation