Atelier interne

Généralité sur la haute disponibilité

0.1

true

Généralité sur la haute disponibilité

Atelier interne

TITRE : Généralité sur la haute disponibilité

SOUS-TITRE : Atelier interne

REVISION: 0.1

DATE: 24 septembre 2020

Table des Matières

Introduction **6**

Définition des termes **7**

RTO / RPO 7
Définition des termes 8

Sauvegardes **9**

PITR 9
PITR et redondance réplication 10
Bilan PITR 10

Réplication physique **12**

Réplication et RPO 12
Réplication et RTO 13
Bilan Réplication 13

Bascule automatisée **14**

Prise de décision 14
Mécanique de fencing 15
Mécanique d'un Quorum 16
Mécanique du Watchdog 17
Storage Base Death 18
Bilan des solutions anti split-brain 19
Implication et Risques de la bascule auto 20

Solutions de HA **21**

Shared storage 21
Patroni 22
Pacemaker 23
Accès aux ressources 23

Conclusion **25**

INTRODUCTION

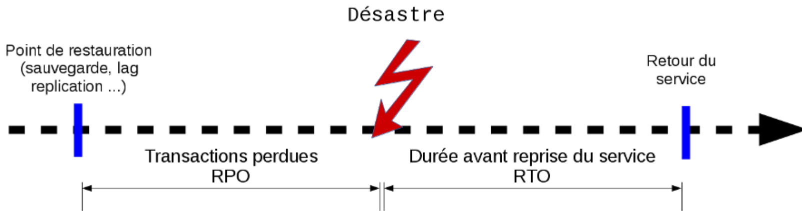
- définition de «Haute Disponibilité»
- contraintes à définir
- contraintes techniques
- solutions existantes

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 délai de détection de l'incident, le délai 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.

Quoi qu'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
- optimiser la restauration complète (RTO)
- ajuster l'archivage au RPO désiré

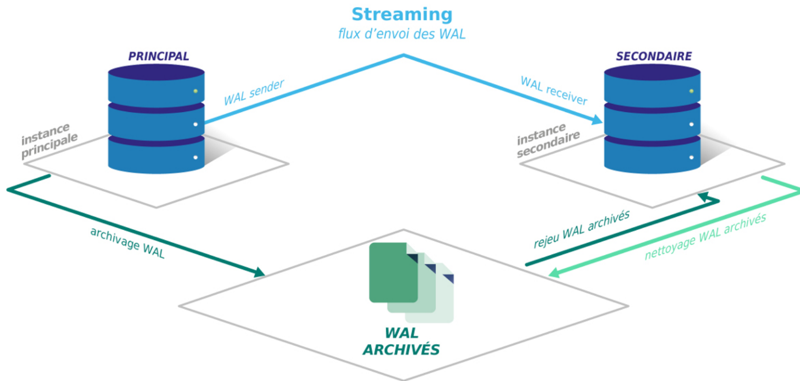
La sauvegarde PITR est une méthode permettant de restaurer une instance PostgreSQL à n'importe quel instant durant la fenêtre de rétention définie, eg. les dernières 24h. Le temps de restauration (RTO) dépend de deux variables: le volume de l'instance et son volume d'écriture.

Avec le bon matériel, les bonnes pratiques et une politique de sauvegarde 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 transactions. 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 sujet dans un futur chapitre.

PITR ET REDONDANCE RÉPLICATION

RÉPLICATION INTERNE POSTGRESQL



Enfin, il est possible d'utiliser les journaux de transactions 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
- nécessite une supervision fiable

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'un administrateur.

Enfin, la sauvegarde PITR doit être surveillée de très près par les équipes d'administration au travers d'une supervision adaptée.

RÉPLICATION PHYSIQUE

- réplique les écritures via les journaux de transactions
- entretient une ou plusieurs instances clone
- intégré à PostgreSQL
- facilité de mise en œuvre
- réduit le RPO par rapport au PITR
- plus de matériel
- architecture et maintenance plus complexes
- haute disponibilité des données

La réplication physique interne de PostgreSQL réplique le contenu des journaux de transactions. 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 modifications binaires effectuées dans les tables et les index.

Cette étape assure déjà une haute disponibilité de donnée, ces dernières étant présentes sur plusieurs serveurs distincts.

La réplication permet d'atteindre un RPO plus faible que celui du PITR, au prix d'investissement plus important (redondance du matériel), d'une complexification de l'architecture et de sa maintenance.

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 et chargement en masse
- 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, 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 transactions 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 typiquement 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

- $0 \leq$ RPO < PITR
- RTO = prise en charge + 30s
- simple à mettre en œuvre
- investissement en coût et humain plus important

La réplication nécessite donc au minimum deux serveurs, voir trois en cas de réplication synchrone. À ce coût s'ajoutent plusieurs autres plus ou moins cachés:

- le réseau se doit d'être redondé et fiable surtout en cas de réplication synchrone
- la formation des équipes d'administration
- la mise en œuvre des procédures de construction et de bascule
- une supervision plus fine et maîtrisée des équipes

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*
- solutions éprouvées: fencing, quorum, watchdog et SBD
- solutions le plus souvent complémentaires.

Quelle que 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 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. Corriger le problème et reconsolider les données est fastidieux et entraîne une indisponibilité plus importante qu'une simple bascule manuelle avec analyse et prise de décision humaine.

Quatre mécaniques permettent de se prémunir plus ou moins des split-brain: le fencing, le quorum, le watchdog et le Storage Based Death.

MÉCANIQUE DE FENCING

- isole un serveur (électriquement, du réseau, etc)
- ou isole une ressource (eg. disque)
- utile dans le cas d'un serveur muet ou fantôme («rogue node»)
- utile lorsque l'arrêt d'une ressource est perturbé
- déclenché depuis un des nœuds du cluster
- nécessite souvent une gestion fine des droits
- Supporté par Pacemaker, embryonnaire dans Patroni

Le fencing (clôture) isole un serveur ou une ressource de façon active. 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.

Il existe au moins deux anomalies où le fencing est incontournable. La première concerne le cas d'un serveur qui ne répond plus au cluster. Il est alors impossible de définir quelle est la situation sur le serveur. Est-il encore vivant ? Les ressources sont-elles encore actives ? Ont-elles encore un comportement normal ? Ont-elles encore accès à l'éventuel disque partagé ? Dans cette situation, la seule façon de répondre avec certitude à ces questions est d'éteindre le serveur. L'action définit avec certitude que les ressources y sont toutes inactives.

La seconde anomalie où le fencing est essentiel concerne l'arrêt des ressources. Si le serveur est disponible, communique, mais n'arrive pas à éteindre une ressource (problème technique ou timeout), le fencing permet « d'escalader » l'extinction de la ressource en extinction du serveur complet.

Il est aussi possible d'isoler un serveur d'une ressource. Le serveur n'est pas éteint, mais son accès à certaines ressources cruciales est coupé, l'empêchant ainsi de corrompre le cluster. L'isolation peut concerner l'accès au réseau Ethernet ou à un disque partagé par exemple.

L'opération de fencing prend donc plusieurs formes différentes, mais doit toujours être rapide et efficace. Pas de demi-mesures. Les plus connus agissent sur un UPS, un PDU, l'IPMI pour couper le courant, ou éteignent une machine virtuelle brusquement via son hyperviseur, coupe l'accès au réseau SAN ou Ethernet, etc.

Par conséquent, cette mécanique nécessite souvent de pouvoir gérer finement les droits d'accès à des opérations d'administration lourdes. C'est le cas par exemple au travers des communautés du protocole SNMP, ou la gestion de droits dans les ESX vmWare, les accès au PDU, etc.

MÉCANIQUE D'UN QUORUM

- chaque serveur possède un ou plusieurs votes
- utile en cas de partition réseau
- la partition réseau qui a le plus de vote détient le quorum
- la partition qui détient le quorum peut héberger les ressources
- la partition sans quorum doit arrêter toute ressource
- attention au retour dans le cluster
- supporté par Pacemaker et Patroni

La mécanique du quorum attribue à chaque nœud un (ou plusieurs) vote. Le cluster n'a le droit d'héberger des ressources que s'il possède la majorité absolue des voix. Par exemple, un cluster à 3 nœuds requiers 2 votes pour pouvoir démarrer les ressources, 3 pour un cluster à 5 nœuds, etc.

Lorsque qu'un ou plusieurs nœuds perdent le quorum, ceux-ci doivent arrêter les ressources qu'ils hébergent.

Il est conseillé de maintenir un nombre de nœuds impair au sein du cluster, mais plusieurs solutions existent en cas d'égalité (eg. par ordre d'id, par poids, serveur «témoins» ou arbitre, etc).

Le quorum permet principalement de gérer les incidents liés au réseau, quand "tout va bien" sur les serveurs eux-mêmes et qu'ils peuvent éteindre leurs ressources sans problème, à la demande.

Dans le cadre de PostgreSQL, il faut porter une attention particulière au moment où des serveurs isolés rejoignent de nouveau le cluster. Si l'instance primaire a été arrêtée par manque de quorum, cette dernière pourrait ne pas raccrocher correctement avec le nouveau primaire, voir corrompre ses propres fichiers de données. Effectivement, il est impossible de déterminer le type d'écriture ayant eu lieu sur l'ancien primaire entre la déconnexion réelle du reste du cluster, l'état de sa réplication, de ses backends et de son arrêt total.

Pacemaker intègre la gestion du quorum et peut aussi utiliser un serveur de gestion de vote appelé `corosync-qnetd`. Ce dernier est utile en tant que tiers pour gérer le quorum de plusieurs clusters Pacemaker à deux nœuds par exemple.

Patroni repose sur un DSC pour stocker l'état du serveur et prendre ses décisions. La responsabilité de la gestion du quorum est donc déléguée au DSC choisi, qui ont pour la plupart une architecture robuste pour toujours présenter des données fiables et de référence à ses clients (ici Patroni).

MÉCANIQUE DU WATCHDOG

- équipement matériel intégré partout
- compte à rebours avant reset complet du serveur
- doit être ré-armé par un composant applicatif du serveur
- permet de déclencher du self-fencing rapide et fiable
- complémentaire au quorum, «fencing du pauvre»
- certaines réactions du cluster plus rapides

Tous les ordinateurs sont désormais équipés d'un watchdog. Par exemple, sur un ordinateur portable Dell Latitude, nous trouvons:

```
iTCO_wdt: Intel TCO WatchDog Timer Driver v1.11
```

Sur un Raspberry pi modèle B:

```
bcm2835-wdt 20100000.watchdog: Broadcom BCM2835 watchdog timer
```

Au besoin, il est aussi possible d'ajouter plusieurs autres watchdog grâce à des cartes PCI par exemple, bien que ce ne soit pas nécessaire dans notre cas.

Concernant les machines virtuelles, une configuration supplémentaire est souvent nécessaire pour avoir accès à un watchdog virtualisé.

En tout dernier recours, il est possible de demander au noyau Linux lui-même de jouer le rôle de watchdog grâce au module `softdog`. Néanmoins, cette méthode est moins fiable qu'un watchdog hardware, car nécessite que le système d'exploitation fonctionne toujours correctement et qu'au moins un des CPU soit disponible. Voici un exemple d'incident avec softdog: http://www.beekhof.net/blog/2019/savaged-by-softdog.

Le principe du watchdog peut être résumé par: nourri le chien de garde avant qu'il ait faim et te mange. En pratique, un watchdog est un compte à rebours avant la réinitialisation brutale du serveur. Si ce compte à rebours n'est pas régulièrement ré-armé, le serveur est alors redémarré.

Un watchdog surveille donc passivement un processus et assure que ce dernier est toujours disponible et sain. Dans le cadre d'un cluster en haute disponibilité, le processus rendant compte de sa bonne forme au watchdog est le clusterware.

Notez qu'un watchdog permet aussi de déclencher un self-fencing rapide et fiable en cas de besoin. Il permet par exemple de résoudre rapidement le cas de l'arrêt forcé d'une ressource, déjà présenté dans le chapitre consacré au fencing.

Patroni et Pacemaker sont tous deux capables d'utiliser un watchdog sur chaque nœud. Pour Patroni, il n'est armé que sur l'instance primaire. Pour Pacemaker, il est armé sur tous les nœuds.

STORAGE BASE DEATH

- l'une des méthodes historique de fencing
- nécessite un ou plusieurs disques partagés et un watchdog par nœud
- les nœuds s'échangent des messages sur ce·s disque·s
- message `poison pill` pour demander à un nœud distant de s'auto-fencer
- self-fencing en cas de perte d'accès aux disques…
- …sauf si l'instance Pacemaker continue à présenter un cluster stable

Le Storage Base Death utilise un ou plusieurs disques partagés (pour la redondance), montés sur tous les nœuds du cluster à la fois. L'espace nécessaire sur chaque disque est très petit, de l'ordre de quelques méga octets pour plusieurs centaines de nœuds (`sbd` utilise 1 à 4 Mo pour 255 nœuds). Cet espace disque est utilisé comme support de communication entre les nœuds qui y échange des messages.

Le clusterware peut isoler un nœud en déposant un message `poison pill` à son attention. Le destinataire s'auto-fence grâce à son watchdog dès qu'il lit le message. De plus, un nœud s'auto-fence aussi s'il n'accède plus au stockage. Ce comportement défensif permet de s'assurer qu'aucun ordre de self-fencing ne peut se perdre.

Grâce au SBD, le cluster est assuré que le nœud distant peut effectuer son self-fencing soit par perte de son accès au disque partagé, soit par réception du poison pill, soit à cause d'une anomalie qui a empêché le clusterware d'assumer le ré-armement du watchdog.

Pacemaker supporte ce type d'architecture. Patroni ne supporte pas SBD mais a un comportement similaire vis-à-vis du DCS. D'une part les nœuds Patroni s'échangent des messages au travers du DCS. Aussi, Patroni doit attendre l'expiration du verrou `leader` avant de pouvoir effectuer une bascule, ce qui est similaire au temps de réactions d'une architecture SBD. Mais surtout, l'instance PostgreSQL est déchue en cas de perte de communication avec le DCS, tous le serveur peut même être éteint si le watchdog est actif et que l'opération est trop longue.

BILAN DES SOLUTIONS ANTI SPLIT-BRAIN

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

- d'un fencing actif ou d'un SBD
- d'un watchdog par serveur et d'un quorum
- l'idéal est de tous les configurer
- désactiver les services au démarrage

Le fencing seul est suffisant pour mettre en œuvre un cluster fiable, même avec deux nœuds. Sans quorum, il est néanmoins nécessaire de désactiver le service au démarrage du cluster, afin d'éviter qu'un nœud isolé ne redémarre ses ressources locales sans l'aval du reste du cluster.

Notez que plusieurs algorithmes existent pour résoudre ce cas, hors quorum, (eg. les paramètres `two_node`, `wait_for_all` et d'autres de Corosync). Néanmoins, dans le cadre de PostgreSQL, il n'est jamais très prudent de laisser une ancienne instance primaire au sein d'un cluster sans validation préliminaire de son état. Nous conseillons donc toujours de désactiver le service au démarrage, quelle que soit la configuration du cluster.

L'utilisation d'un SBD est une alternative intéressante et fiable pour la création d'un cluster à deux nœuds sans fencing actif. Le stockage y joue un peu le rôle du tiers au sein du cluster pour départager quel nœud conserve les ressources en cas de partition réseau. Le seul défaut de SBD par rapport au fencing est le temps d'attente supplémentaire avant de pouvoir considérer que le nœud distant est bien hors service. Aussi, attention au stockage iSCSI sur le même réseau que le cluster. En cas d'incident réseau généralisé, comme chaque machine perd son accès au disque ET aux autres machines via Pacemaker, toutes vont s'éteindre.

Une autre architecture possible est le cumul d'un quorum et du watchdog. Avec une telle configuration, en cas de partition réseau, la partition détenant le quorum attend alors la durée théorique du watchdog (plus une marge) avant de démarrer les ressources perdues. Théoriquement, les nœuds de la partition du cluster perdue sont alors soit redémarrés par leur watchdog, soit sains et ont pu arrêter les ressources normalement. Ce type d'architecture nécessite à minima trois nœuds dans le cluster, ou de mettre en place un nœud témoins, utilisé dans le cadre du quorum uniquement (eg. `corosync QNetd`).

Le cluster idéal cumul les avantages du Fencing, Quorum et Watchdog.

Comme nous l'avons vu, Pacemaker dispose de toutes les solutions connues. Reste à trouver la bonne combinaison en fonction des contraintes de l'architecture. Patroni quant à lui a une architecture similaire au SBD, mais ne force pas à utiliser le watchdog sur les nœuds. Pour avoir une architecture aussi fiable que possible, il est recommandé de toujours activer le watchdog sur tous les nœuds, au strict minima via softdog.

IMPLICATION ET RISQUES DE LA BASCULE AUTO

- un collègue peu parlant 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 soient 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 humaines, 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. Quelle que soit la solution choisie, assurez-vous d'allouer suffisamment de temps au projet pour expérimenter, tester le cluster et le documenter.

SOLUTIONS DE HA

- plusieurs architectures et solutions possibles
- Shared Storage: Pacemaker
- Shared Nothing
  - Pacemaker
  - Patroni

SHARED STORAGE

- architecture simple
- redondance au niveau stockage
- 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: SDB + Watchdog
- ne gère que PostgreSQL
- 1 cluster DCS (etcd) de 3 nœuds est recommandé
- un DCS peut gérer N clusters 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. Le DCS nécessite au moins 3 nœuds pour assurer son quorum propre, la sécurité et la viabilité de ses données. Néanmoins un même cluster DCS peut ensuite gérer plusieurs clusters 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.

Depuis la verison 2.0, Patroni permet aussi d'utiliser le protocol Raft, sans utiliser de DCS externe. Nous n'avons aucun recul sur cette technologie pour le moment.

En plus de ce DCS fiable et d'un mécanisme d'élection ne laissant aucune place aux race conditions, Patroni supporte l'utilisation d'un watchdog matériel. Le couple DCS+watchdog permet donc d'éviter les situations de split-brain.

Depuis la version 2.0 de Patroni offre la possibilité d'exécuter une action pré-promotion, capable d'annuler la promotion si nécessaire. Cette fonctionnalité a été ajoutée afin de permettre l'ajout d'une action de fencing de l'ancien primaire. Nous n'avons pour le moment aucun recul sur cette fonctionnalité. Il faut notamment étudier quelles informations sont accessibles depuis le callback pour décider d'un fencing ou non.

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 et cohérent sur toutes les distributions principales
- super fiable: 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. RedHat et Suse supportent officiellement les clusters Pacemaker (souvent au travers de souscriptions complémentaires) en imposant certaines contraintes de robustesse à leurs clients (eg. fencing obligatoire).

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 matière 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 n'est cependant par totalement robuste avec Patroni.

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 couches 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 et budgétiser.

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 qui vous exposent à des risques certains d'indisponibilité, voir de corruption (expériences plusieurs fois vécue au support Dalibo).

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

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