

Quels outils pour quels problèmes ?

Supervision graphique en PostgreSQL



Dalibo & Contributors

<https://dalibo.com/formations>

Supervision graphique en PostgreSQL

Quels outils pour quels problèmes ?

TITRE : Supervision graphique en PostgreSQL
SOUS-TITRE : Quels outils pour quels problèmes ?

REVISION: 19.11
LICENCE: PostgreSQL

Table des Matières

Supervision en PostgreSQL	6
Au programme	7
Superviser ?	7
Que superviser ?	7
Politique de supervision	8
pgBadger	13
temBoard	24
Conclusion	30
Travaux Pratiques	31

SUPERVISION EN POSTGRESQL



Figure 1: PostgreSQL

Photographie de Rad Dougall, licence [CC BY 3.0¹](https://creativecommons.org/licenses/by/3.0/deed.en) , obtenue sur [wikimedia.org²](https://commons.wikimedia.org/wiki/File:The_Big_Boss_Elephant_(190898861).jpeg) .

¹<https://creativecommons.org/licenses/by/3.0/deed.en>

²[https://commons.wikimedia.org/wiki/File:The_Big_Boss_Elephant_\(190898861\).jpeg](https://commons.wikimedia.org/wiki/File:The_Big_Boss_Elephant_(190898861).jpeg)

AU PROGRAMME

- Supervision : quoi et pourquoi ?
 - Trois outils de supervision graphique :
 - PoWA
 - pgBadger
 - temBoard
-

SUPERVISER ?

| sy.pex.vi.ze |

« Se placer au-dessus pour voir, remarquer, prendre des mesures »

La supervision est la *surveillance du bon fonctionnement d'un système ou d'une activité*.

Elle permet de surveiller, rapporter et alerter les fonctionnements normaux et anormaux des systèmes informatiques.

Elle répond aux préoccupations suivantes :

- technique : surveillance du réseau, de l'infrastructure et des machines ;
 - applicative : surveillance des applications et des processus métiers ;
 - contrat de service : surveillance du respect des indicateurs contractuels ;
 - métier : surveillance des processus métiers de l'entreprise.
-

QUE SUPERVISER ?

- Superviser PostgreSQL et le système
- Deux types de supervision :
 - automatique,
 - occasionnelle.

Superviser un serveur de bases de données consiste à superviser le SGBD lui-même mais aussi le système d'exploitation et le matériel. Ces deux derniers sont importants pour connaître la charge système, l'utilisation des disques ou du réseau, qui pourraient expliquer

des lenteurs au niveau du SGBD. PostgreSQL propose lui aussi des informations qu'il est important de surveiller pour détecter des problèmes au niveau de l'utilisation du SGBD ou de sa configuration.

Un système de supervision automatique est primordial. Il permet de toujours être notifié en cas de dysfonctionnement. Couplé à un outil de visualisation graphique, il fournit un aperçu de l'évolution du système dans le temps.

Il peut également être intéressant, en cas de fonctionnement anormal, ou pour un besoin d'audit, de venir ponctuellement superviser un système en utilisant un outil précis.

POLITIQUE DE SUPERVISION

- Pour quoi ?
- Pour qui ?
- Quels critères ?
- Quels outils ?

Il n'existe pas qu'une seule supervision. Suivant la personne concernée par la supervision, son objectif, les critères de la supervision seront différents.

Lors de la mise en place de la supervision, il est important de se demander l'objectif de cette supervision, à qui elle va servir, les critères qui importent à cette personne.

Répondre à ces questions permettra de mieux choisir l'outil de supervision à mettre en place, ainsi que sa configuration.

OBJECTIFS DE LA SUPERVISION

- Améliorer / mesurer les performances
- Améliorer l'applicatif
- Anticiper / prévenir les incidents
- Réagir vite en cas de crash

Généralement, les administrateurs mettant en place la supervision veulent pouvoir anticiper les problèmes qu'ils soient matériels, de performance, de qualité de service, etc.

Améliorer les performances du SGBD sans connaître les performances globales du système est très difficile. Si un utilisateur se plaint d'une perte de performance, pouvoir

corroborer ses dires avec des informations provenant du système de supervision aide à s'assurer qu'il y a bien un problème de performances et peut fréquemment aider à résoudre le problème. De plus, il est important de pouvoir mesurer les gains obtenus après une modification matérielle ou logicielle.

Une supervision des traces de PostgreSQL permet aussi d'améliorer les applications qui utilisent une base de données. Toute requête en erreur est tracée dans les journaux applicatifs, ce qui permet de trouver rapidement les problèmes que les utilisateurs rencontrent.

Un suivi régulier de la volumétrie ou du nombre de connexions permet de prévoir les évolutions nécessaires du matériel ou de la configuration : achat de matériel, création d'index, amélioration de la configuration.

Prévenir les incidents peut se faire en ayant une sonde de supervision des erreurs disques par exemple. La supervision permet aussi d'anticiper les problèmes de configuration. Par exemple, surveiller le nombre de sessions ouvertes sur PostgreSQL permet de s'assurer que ce nombre n'approche pas trop du nombre maximum de sessions configuré avec le paramètre `max_connections` dans le fichier `postgresql.conf`.

Enfin, une bonne configuration de la supervision implique d'avoir configuré finement la gestion des traces de PostgreSQL. Avoir un bon niveau de trace (autrement dit ni trop ni pas assez) permet de réagir rapidement après un crash.

ACTEURS CONCERNÉS

- Développeur
 - correction et optimisation de requêtes
- Administrateur de bases de données
 - surveillance, performance
 - mise à jour
- Administrateur système
 - surveillance, qualité de service

Il y a trois types d'acteurs concernés par la supervision.

Le développeur doit pouvoir visualiser l'activité de la base de données. Il peut ainsi comprendre l'impact du code applicatif sur la base. De plus, le développeur est intéressé par la qualité des requêtes que son code exécute. Donc des traces qui ramènent les requêtes en erreur et celles qui ne sont pas performantes sont essentielles pour ce profil.

L'administrateur de bases de données a besoin de surveiller les bases pour s'assurer de

la qualité de service, pour garantir les performances et pour réagir rapidement en cas de problème. Il doit aussi faire les mises à jours mineures dès qu'elles sont disponibles.

Enfin, l'administrateur système doit s'assurer de la présence du service. Il doit aussi s'assurer que le service dispose des ressources nécessaires, en terme de processeur (donc de puissance de calcul), de mémoire et de disque (notamment pour la place disponible).

INDICATEURS CÔTÉ SYSTÈME D'EXPLOITATION

- Charge CPU
- Entrées/sorties disque
- Espace disque
- Sur-activité et inactivité du serveur
- Temps de réponse

Voici quelques exemples d'indicateurs intéressants à superviser pour la partie du système d'exploitation.

La charge CPU (processeur) est importante. Elle peut expliquer pourquoi des requêtes, auparavant rapides, deviennent lentes. Cependant, la suractivité comme la non-activité sont un problème. En fait, si le service est tombé, le serveur sera en sous-activité, ce qui est un excellent indice.

Les entrées/sorties disque permettent de montrer un souci au niveau du système disque : soit PostgreSQL écrit trop à cause d'une mauvaise configuration des journaux de transactions, soit les requêtes exécutées utilisent des fichiers temporaires pour trier les données, ou pour une toute autre raison.

L'espace disque est essentiel à surveiller. PostgreSQL ne propose rien pour cela, il faut donc le faire au niveau système. L'espace disque peut poser problème s'il manque, surtout si cela concerne la partition des journaux de transactions.

Il est possible aussi d'utiliser une requête étalon dont la durée d'exécution sera testée de temps à autre pour détecter les moments problématiques sur le serveur.

INDICATEURS CÔTÉ BASE DE DONNÉES

- Nombre de connexions
- Requêtes lentes et/ou fréquentes
- Nombre de transactions par seconde
- Ratio d'utilisation du cache
- Retard de réplication

Il existe de nombreux indicateurs intéressants sur les bases :

- nombre de connexions : en faisant par exemple la différence entre connexions inactives, actives, en attente de verrous,
 - nombre de requêtes lentes et / ou fréquentes,
 - nombre de transactions par seconde
 - volumétrie : en taille, en nombre de lignes,
 - ratio de lecture du cache (souvent appelé *hit ratio*)
 - retard de réplication
 - nombre de parcours séquentiels et de parcours d'index
 - etc.
-

INFORMATIONS INTERNES

- PostgreSQL propose deux canaux d'informations :
 - les statistiques d'activité
 - les traces
- Mais rien pour les conserver, les historiser

PostgreSQL propose deux canaux d'informations :

- les statistiques d'activité. À ne pas confondre avec les statistiques sur les données, à destination de l'optimiseur de requêtes.
- les traces applicatives (ou « logs »), souvent dans un fichier dont le nom varie avec la distribution et l'installation.

PostgreSQL stocke un ensemble d'informations dans des tables systèmes. Il peut s'agir de métadonnées des schémas, d'informations sur les tables et les colonnes, de données de suivi interne, etc. PostgreSQL fournit également des vues combinant des informations puisées dans différentes tables systèmes. Ces vues simplifient le suivi de l'activité de la base.

Supervision graphique en PostgreSQL

PostgreSQL est aussi capable de tracer un grand nombre d'informations qui peuvent être exploitées pour surveiller l'activité de la base de données.

Pour pouvoir mettre en place un système de supervision automatique, il est essentiel de s'assurer que les statistiques d'activité et les traces applicatives sont bien configurées et il faut aussi leur associer un outil permettant de sauvegarder les données, les alertes et de les historiser.

OUTILS EXTERNES

- Nécessaire pour conserver les informations
- ... et exécuter automatiquement des actions dessus :
 - Génération de graphiques
 - Envoi d'alertes

Pour récupérer et enregistrer les informations statistiques, les historiser, envoyer des alertes ou dessiner des graphiques, il faut faire appel à un outil externe.

Cela peut être fait grâce à des outils de supervision génériques comme Icinga, munin ou Zabbix. On utilisera des agents ou plugins spécifiques pour ces outils comme pg-monz, check_pgactivity ou check_postgres.

Nous nous intéresserons durant ce workshop à des outils graphiques spécifiques pour PostgreSQL.

OUTILS GRAPHIQUES

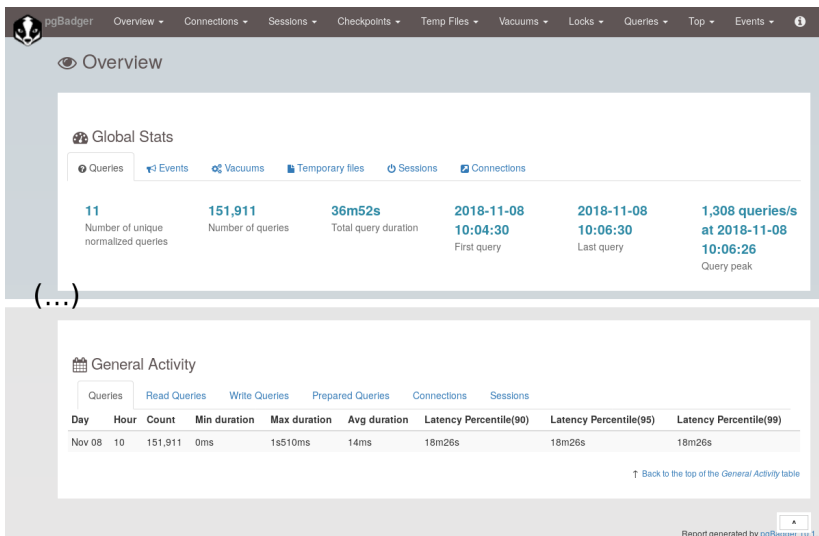
- Beaucoup d'outils existent
- Deux types :
 - en temps réel : PoWA et temboard
 - rétro-analyse : pgBadger

Il existe de nombreux programmes qui analysent les traces. On peut distinguer deux catégories :

- ceux qui le font en temps réel ;
- ceux qui le font après coup (de la rétro-analyse en fait).

L'analyse en temps réel d'une instance permet de réagir rapidement en cas de problème. Par exemple, il est important d'avoir une réaction rapide en cas de manque d'espace disque, ou bien de pouvoir comprendre les raisons de requêtes trop lentes. Dans cette catégorie, nous discuterons de deux outils : PoWA et temboard.

L'analyse après coup permet une analyse plus fine, se terminant généralement par un rapport, fréquemment en HTML, parfois avec des graphes. Cette analyse plus fine nécessite des outils spécialisés. Nous étudierons le logiciel pgBadger.



PGBADGER

- Script Perl
- site officiel : <https://pgbadger.darold.net/>
- Traite les journaux applicatifs de PostgreSQL
- Génère un rapport HTML très détaillé

pgBadger est un script Perl écrit par Gilles Darold. Il s'utilise en ligne de commande : il suffit de lui fournir le ou les fichiers de traces à analyser et il rend un rapport HTML sur

Supervision graphique en PostgreSQL

les requêtes exécutées, sur les connexions, sur les bases, etc. Le rapport est très complet, il peut contenir des graphes zoomables.

C'est certainement le meilleur outil actuel de rétro-analyse d'un fichier de traces PostgreSQL.

Le site web de pgBadger se trouve sur <https://pgbadger.darold.net/>

CONFIGURER POSTGRESQL POUR PGBADGER

- Utilisation des traces applicatives
- Où tracer ?
- Quel niveau de traces ?
- Tracer les requêtes
- Tracer certains comportements

Il est essentiel de bien configurer PostgreSQL pour que les traces ne soient pas en même temps trop nombreuses pour ne pas être submergées par les informations et trop peu pour ne pas savoir ce qu'il se passe. Un bon dosage du niveau des traces est important. Savoir où envoyer les traces est tout aussi important.

CONFIGURATION MINIMALE

- traces en anglais
 - `lc_messages='C'`
- ajouter le plus d'information possible
 - `log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '`

Les traces sont enregistrées dans la locale par défaut du serveur. Avoir des traces en français peut présenter certains intérêts pour les débutants mais cela présente plusieurs gros inconvénients. Chercher sur un moteur de recherche avec des traces en français donnera beaucoup moins de résultats qu'avec des traces en anglais.

De plus, les outils d'analyse automatique des traces se basent principalement sur des traces en anglais. Donc, il est vraiment préférable d'avoir les traces en anglais. Cela peut se faire ainsi :

```
lc_messages = 'C'
```

Lorsque la destination des traces est `syslog` ou `eventlog`, elles se voient automatiquement ajouter quelques informations dont un horodatage, essentiel. Lorsque la destination est `stderr`, ce n'est pas le cas. Par défaut, l'utilisateur se retrouve avec des traces sans horodatage, autrement dit des traces inutilisables. PostgreSQL propose donc le paramètre `log_line_prefix` qui permet d'ajouter un préfixe à une trace.

Ce préfixe peut contenir un grand nombre d'informations, comme un horodatage, le PID du processus serveur, le nom de l'application cliente, le nom de l'utilisateur, le nom de la base. Un paramétrage possible est le suivant :

```
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
```

TRACER CERTAINS COMPORTEMENTS

- `log_connections`, `log_disconnections`
- `log_autovacuum_min_duration`
- `log_checkpoints`
- `log_lock_waits`
- `log_temp_files`

En dehors des erreurs et des durées des requêtes, il est aussi possible de tracer certaines activités ou comportements.

Quand on souhaite avoir une trace de qui se connecte, il est intéressant de pouvoir tracer les connexions et, parfois aussi, les déconnexions. En activant les paramètres `log_connections` et `log_disconnections`, nous obtenons les heures de connexions, de déconnexions et la durée de la session.

`log_autovacuum_min_duration` correspond à `log_min_duration_statement`, mais pour l'autovacuum. Son but est de tracer l'activité de l'autovacuum si son exécution demande plus d'un certain temps.

`log_checkpoints` permet de tracer l'activité des checkpoints. Cela ajoute un message dans les traces pour indiquer qu'un checkpoint commence et une autre quand il termine. Cette deuxième trace est l'occasion d'ajouter des statistiques sur le travail du checkpoint :

```
2016-09-01 13:34:17 CEST LOG: checkpoint starting: xlog
2016-09-01 13:34:20 CEST LOG: checkpoint complete: wrote 13115 buffers (80.0%);
                                0 transaction log file(s) added, 0 removed,
                                0 recycled; write=3.007 s, sync=0.324 s,
                                total=3.400 s; sync files=16,
```

```
longest=0.285 s,  
average=0.020 s; distance=404207 kB,  
estimate=404207 kB
```

Le message indique donc en plus le nombre de blocs écrits sur disque, le nombre de journaux de transactions ajoutés, supprimés et recyclés. Il est rare que des journaux soient ajoutés, ils sont plutôt recyclés. Des journaux sont supprimés quand il y a eu une très grosse activité qui a généré plus de journaux que d'habitude. Les statistiques incluent aussi la durée des écritures, de la synchronisation sur disque, la durée totale, etc.

Le paramètre `log_lock_waits` permet de tracer une attente trop importante de verrous. En fait, quand un verrou est en attente, un chronomètre est déclenché. Lorsque l'attente dépasse la durée indiquée par le paramètre `deadlock_timeout`, un message est enregistré, comme dans cet exemple :

```
2016-09-01 13:38:40 CEST LOG:  process 15976 still waiting for  
                                AccessExclusiveLock on relation 26160 of  
                                database 16384 after 1000.123 ms  
2016-09-01 13:38:40 CEST STATEMENT:  DROP TABLE t1;
```

Plus ce type de message apparaît dans les traces, plus des contentions ont lieu sur le serveur, ce qui peut diminuer fortement les performances.

Le paramètre `log_temp_files` permet de tracer toute création de fichiers temporaires, comme ici :

```
2016-09-01 13:41:11 CEST LOG:  temporary file: path  
                                "base/pgsql_tmp/pgsql_tmp15617.1",  
                                size 59645952
```

Tout fichier temporaire demande des écritures disques.

Ces écritures peuvent poser problème pour les performances globales du système. Il est donc important de savoir si des fichiers temporaires sont créés ainsi que leur taille.

TRACER LES REQUÊTES

- `log_min_duration_statement`
- en production, trace les requêtes longues
 - 10000 pour les requêtes de plus de 10 secondes
- pour un audit, à 0 : trace toutes les requêtes

Le paramètre `log_min_duration_statement`, trace toute requête dont la durée d'exécution dépasse la valeur du paramètre (l'unité est la milliseconde). Il trace aussi la durée d'exécution des requêtes tracées. Par exemple, avec une valeur de 500, toute requête dont la durée d'exécution dépasse 500 ms sera tracée. À 0, toutes les requêtes se voient tracées. Pour désactiver la trace, il suffit de mettre la valeur -1 (qui est la valeur par défaut).

Suivant la charge que le système va subir à cause des traces, il est possible de configurer finement la durée à partir de laquelle une requête est tracée. Cependant, il faut bien comprendre que plus la durée est importante, plus la vision des performances est partielle. Il est parfois plus intéressant de mettre 0 ou une très petite valeur sur une petite durée, qu'une grosse valeur sur une grosse durée. Cela étant dit, laisser 0 en permanence n'est pas recommandé. Il est préférable de configurer ce paramètre à une valeur plus importante en temps normal pour détecter seulement les requêtes les plus longues et, lorsqu'un audit de la plateforme est nécessaire, passer temporairement ce paramètre à une valeur très basse (0 étant le mieux).

La trace fournie par `log_min_duration_statement` ressemble à ceci :

```
2018-09-01 17:34:03 CEST LOG:  duration: 136.811 ms  statement: insert into t1
                               values (2000000,'test');
```

CRÉATION D'UN RAPPORT PGBADGER

```
$ pgbadger postgresql-11-main.log
```

- Rapport dans le fichier `out.html`
- Très nombreuses options :
 - fichier de sortie : `--outfile`
 - filtrage par date : `--begin, --end`
 - autres filtres : `--dbname, --dbuser, --appname, ...`

La façon la plus simple pour créer un rapport pgBadger est de simplement indiquer au script le fichier de traces de PostgreSQL à analyser.

Supervision graphique en PostgreSQL

Il existe énormément d'options. L'aide fournie sur le [site web officiel](http://pgbadger.darold.net/documentation.html#SYNOPSIS)³ les cite intégralement. Il serait difficile de les citer ici, des options étant ajoutées très fréquemment.

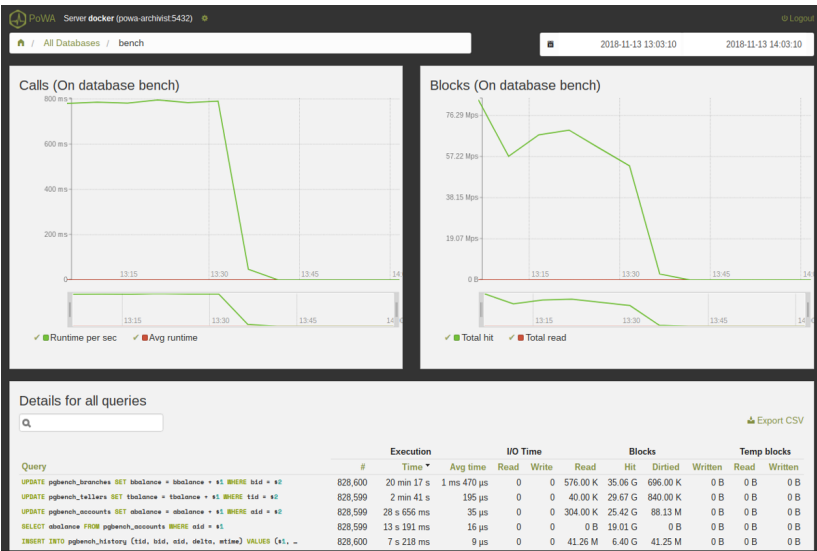
A noter un mode de fonctionnement incrémental. Combiné au format binaire, il permet de parser régulièrement les fichiers de traces applicatives de PostgreSQL. Puis, de générer des rapports HTML à la demande.

On peut ainsi créer un fichier chaque heure :

```
pgbadger --last-parsed .pgbadger_last_state -o YY_MM_DD_HH.bin postgresql.log
```

On pourra créer un rapport en précisant les fichiers binaires voulus :

```
pgbadger -o rapport_2018_11_05.html 2018_11_05_*.bin
```



³<http://pgbadger.darold.net/documentation.html#SYNOPSIS>

POWA

- *PostgreSQL Workload Analyzer*
- site officiel : <https://github.com/powa-team/powa>
- Capture régulière de métriques diverses
- Stockage efficace des données
- Interface graphique permettant d'exploiter ces informations

PoWA (*PostgreSQL Workload Analyzer*) est un outil communautaire, sous licence PostgreSQL.

L'outil récupère à intervalle régulier les statistiques collectées par diverses extensions, les stocke et les historise.

L'outil fournit également une interface graphique permettant d'exploiter ces données. On pourra observer en temps réel l'activité de l'instance. Cette activité est présentée sous forme de graphiques interactifs et de tableaux présentant les requêtes normalisées. Ces tableaux peuvent être triés selon divers critères sur un intervalle de temps sélectionné.

PoWA permet d'afficher de nombreuses informations qui manquent cruellement à l'écosystème PostgreSQL, par exemple :

- le taux de lecture dans le cache du système d'exploitation et les accès disques physiques
- l'utilisation processeur de chacune des requêtes

Il est aussi capable de suggérer la création d'index pertinents.

Et tout cela en temps réel.

EXTENSIONS DE COLLECTE

- *pg_stat_statements* : métriques côté PostgreSQL
- *pg_stat_kcache* : métriques côté système
- *pg_qualstats* : informations sur les prédicats
- extension en développement (ne pas utiliser en production) :
 - *pg_sortstats* : informations sur la mémoire pour les tris
 - *pg_wait_sampling* : statistiques sur les évènements en attente

La collecte des informations et métriques de PoWA sont fournis par des extensions de PostgreSQL. Leur mise en place nécessite le préchargement de bibliothèques dans la

Supervision graphique en PostgreSQL

mémoire partagée grâce au paramètre `shared_preload_libraries` dans le fichier `postgresql.conf`. Leur activation nécessite le redémarrage de l'instance.

`pg_stat_statements` est une extension officielle de PostgreSQL. Elle est disponible dans les modules *contrib*. Elle permet de récupérer de nombreuses métriques par requête normalisée, utilisateur et base de données. Ces données sont par exemple le nombre d'appels, le temps moyen, le nombre de blocs lus dans le cache de PostgreSQL pour chaque requête normalisée.

`pg_stat_kcache` est une extension développée pour PoWA. Elle fournit des indicateurs complémentaires à ceux de `pg_stat_statements`, mais côté système. On a donc à disposition par requête, utilisateur et base de données l'utilisation du processeur, ainsi que les accès physiques aux disques.

`pg_qualstats` est une extension développée pour PoWA. Elle fournit de nombreuses informations très pertinentes concernant les prédicats des requêtes exécutées sur une instance, comme la sélectivité d'un prédicat, les valeurs utilisées, etc.

`pg_wait_sampling` est une extension qui échantillonne à une fréquence élevée les `wait_event`. PoWA repose sur cette extension pour collecter les données et les historiser.

`pg_sortstats` est une extension développée pour PoWA. Elle récupère des statistiques sur les tris et permet d'estimer la quantité de mémoire `work_mem` nécessaire pour effectuer un tri en mémoire et non sur disque. Voir la [section consacrée à ce paramètre⁴](#) dans le chapitre sur l'optimisation de la formation [DBA4 - PostgreSQL Performances⁵](#) pour plus d'information.

Cette extension est en phase de développement et ne doit pas être utilisée en production. Vous êtes encouragés à la tester et à faire des retours aux développeurs du projet.

Plus d'information dans les documentations :

- [pg_stat_statements⁶](#)
- [pg_stat_kcache⁷](#)
- [pg_qualstats⁸](#)
- [pg_wait_sampling⁹](#)
- [pg_sortstats¹⁰](#)

⁴<https://cloud.dalibo.com/p/exports/formation/manuels/formations/dba4/dba4.handout.html#configuration---m%C3%A9moire>

⁵<https://www.dalibo.com/formation-postgresql-performance>

⁶<https://www.postgresql.org/docs/current/static/pgstatstatements.html>

⁷https://powa.readthedocs.io/en/latest/stats_extensions/pg_stat_kcache.html

⁸https://powa.readthedocs.io/en/latest/stats_extensions/pg_qualstats.html

⁹https://github.com/postgrespro/pg_wait_sampling

¹⁰https://github.com/powa-team/pg_sortstats

POWA ARCHIVIST


- Extension de PostgreSQL
- Capture régulière des statistiques
- Stockage efficace des données

PoWA archivist est l'extension centrale du logiciel PoWA. Son rôle est de capturer, échantillonner et stocker les informations fournies par les extensions de collectes.

Les actions de PoWA archivist sont gérées par un background worker, un processus dédié géré directement par PostgreSQL. Ce processus va capturer de façon régulière, suivant le paramétrage de l'extension, les métriques des collecteurs disponibles. Ces métriques sont ensuite échantillonnées grâce à des fonctions spécifiques. Les résultats sont stockés dans une base de données dédiée de l'instance. Les données stockées sont purgées pour ne garder que l'historique configuré.

Plus d'information sur les [clés de configuration](#)¹¹ dans la documentation.

HYPOPG

- 
- Extension de PostgreSQL
 - Créer des index hypothétiques
 - Permet la proposition de nouveaux index

HypoPG est une extension développée pour PoWA. Elle ne fournit pas de statistiques supplémentaires. Elle permet de créer des index hypothétiques qui n'existent pas sur disque. Leur création est donc instantanée et n'a aucun impact sur les disques ou la charge CPU. Couplée aux autres extensions de PoWA, HypoPG permet de tester si un nouvel index pourrait améliorer les performances d'une requête donnée.

Plus d'information dans la documentation du [projet HypoPG](#)¹².

¹¹<https://powa.readthedocs.io/en/latest/powa-archivist/configuration.html>

¹²https://powa.readthedocs.io/en/latest/stats_extensions/hypopg.html

POWA-WEB

- Interface graphique web
- Permet d'observer en temps réel l'activité des requêtes

PoWA fournit une interface web basée sur le framework [Tornado](https://www.tornadoweb.org/en/stable/)¹³ : PoWA-web. Cette interface permet d'exploiter les données stockées par PoWA archivist, et donc d'observer en temps réel l'activité de l'instance. Cette activité est présentée sous forme de graphiques interactifs et de tableaux permettant de trier selon divers critères les différentes requêtes normalisées sur l'intervalle de temps sélectionné.

Une démo de cette interface est disponible sur le site : <http://demo-powa.dalibo.com/>.

POINTS FAIBLES DE POWA

- Impact sur les performances
- Installation de plusieurs extensions sur les instances de production
- Pas d'informations sur les serveurs secondaires
 - développement en cours pour déporter PoWA sur sa propre instance

PoWA est efficace car il est au plus près de l'instance. Son utilisation a cependant un impact indéniable sur les performances. Cet impact est dû à l'activation de l'extension *pg_stat_statements* et à la collecte des données par PoWA. Pour plus d'information, voir [le comparatif](#)¹⁴ fait par notre équipe.

PoWA s'installe comme une extension de PostgreSQL et stocke ses informations dans une base de données de l'instance. Une défaillance sur un outil externe ne posera pas de problème sur l'instance. Un problème sur n'importe quelle extension de PoWA peut conduire à un crash de votre instance. Il est important de bien tester l'outil avant de l'installer et de l'utiliser en production.

Les informations étant fournies par des extensions requérant un accès direct au moteur de chaque instance, l'utilisation de PoWA nécessite le stockage d'information sur chacune des instances que l'on souhaite gérer. Un serveur secondaire est en lecture seule. On ne peut donc pas y utiliser PoWA.

¹³<https://www.tornadoweb.org/en/stable/>

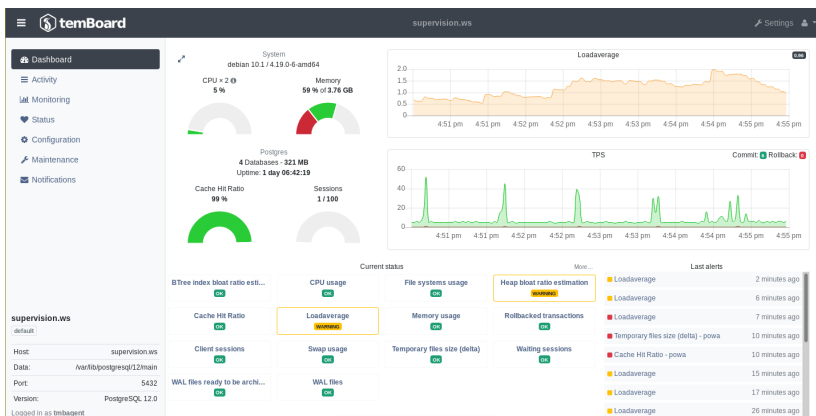
¹⁴<https://github.com/powa-team/powa/wiki/POWA-vs-pgBadger>

POINTS FORTS DE POWA

- Information en temps réel
 - et dans le passé
- Granularité des informations jusqu'à la requête
 - voire jusqu'au prédicat

En cas d'incident sur une production, PoWA permet de détecter immédiatement la ou les requêtes posant problème. Il peut proposer des optimisations pour améliorer la situation. Si la conservation d'un historique suffisant a été configuré, il permet de comparer les performances entre deux périodes données.

Les modules de collectes regroupent leurs métriques par requête normalisée, utilisateur et base de données. Toutes les informations récoltées par PoWA sont donc disponibles avec cette granularité.



TEMBOARD

- Adresse: <https://github.com/dalibo/temboard>
- Console centrale d'administration et de supervision
- Architecture serveur (interface) / agent
- Historisation des données et temps réel
- Extensible

temBoard est une console d'administration et de supervision d'instances PostgreSQL. Il offre une centralisation des interactions et accès aux données collectées, s'inscrivant ainsi dans une politique de gestion de parc.

L'outil est constitué de deux composants :

- un serveur, proposant une interface web au travers de laquelle les DBAs vont pouvoir interagir avec les instances Postgres;
- un agent, devant être déployé sur chaque hôte hébergeant une instance Postgres à surveiller.

Le serveur nécessite l'usage de sa propre base de données dans le but d'historiser les différentes données remontées par les agents.

Chaque fonctionnalité est implémentée sous forme de *plugin*, et peut être activée ou désactivée par instance.

TEMBOARD - SERVEUR

- Interface Web
 - Python 2.7 / Tornado / SQLAlchemy
- Base de données historique et metadonnées
 - PostgreSQL 9.4+
- Authentification
- Disponible sur PyPI et packagé pour :
 - CentOS / RHEL 7
 - debian *jessie* et *stretch*

L'interface utilisateur de temboard est développée en python 2.7 et repose sur le framework web Tornado.

Une base de données, appelée *repository* est nécessaire à son fonctionnement, en effet, celle-ci va permettre de stocker :

- la liste des comptes utilisateurs habilités à se connecter à l'interface;
- la liste des instances Postgres à gérer;
- l'historique des données collectées.

L'accès à cette interface est protégée par une authentification utilisateur.

Il est possible d'installer temBoard en utilisant PyPI, ou bien

L'installation de temBoard est possible soit le gestionnaire de paquets *pip* via [PyPI¹⁵](#) . Des paquets sont également disponible sur les dépôt [yum¹⁶](#) et [apt¹⁷](#) de dalibo pour les systèmes d'exploitation CentOS/RHEL en version 7 et debian en version *jessie* et *stretch*.

TEMBOARD - AGENT

- Mono-instance
- Pas de dépendances
- API REST
- Authentification
- Disponible sur PyPI et packagé pour :
 - centos / RHEL (6 et 7)
 - debian (7 à 10)

L'agent temboard doit quant à lui être déployé sur chaque hôte qui héberge une instance PostgreSQL. Celui-ci ne peut gérer qu'une seule instance PostgreSQL.

Il est développé en python et est compatible avec des versions 2.6+ et 3.5+.

Il est interrogeable et contrôlable au travers d'une *API REST (HTTPS)* et peut facilement entrer en interaction avec des outils tiers.

Documentation de l'API : <https://temboard-agent.readthedocs.io/en/latest/api.html>

L'agent embarque son propre système d'authentification, qui est indépendant de celui de l'interface utilisateur. La sécurité des échanges est garantie par le protocole sécurisé *HTTPS*.

¹⁵<https://pypi.org/search/?q=temboard>

¹⁶<https://yum.dalibo.org/labs/>

¹⁷<https://apt.dalibo.org/labs/>

TEMBOARD - FONCTIONNALITÉS

- Tableau de bord
- Activité
- Supervision / Alerting
- Statut
- Configuration
- Maintenance
- Notifications

De nombreuses fonctionnalités sont disponibles.

PLUGIN DASHBOARD : TABLEAU DE BORD

- Vision en temps réel de l'état du système
- Présentation de quelques métriques de l'instance

Donne une vision en temps réel (rafraîchissement toutes les 2 secondes) de l'état du système et de l'instance Postgres en mettant en évidence certaines données :

- Métriques système : usage CPU, mémoire, *loadaverage*.
 - Métriques Postgres : Cache Hit Ratio, Sessions, TPS.
 - Statut de chaque métrique calculé selon des seuils (*alerting*).
-

PLUGIN ACTIVITY : ACTIVITÉ

- liste des requêtes SQL en cours d'exécution
- permet de terminer un processus serveur

Ce plugin permet de consulter en temps réel la liste des requêtes SQL en cours d'exécution, les requêtes bloquées ou les requêtes bloquantes. La liste affichée des *backends* contient les informations suivantes :

- PID du processus,
- nom de la base,
- utilisateur,
- usage CPU,
- usage mémoire,

- I/O,
- si le *backend* est en attente de verrou,
- durée d'exécution,
- requête SQL.

Il permet également de mettre en pause le rafraichissement automatique et de terminer un backend.

PLUGIN MONITORING : SUPERVISION

- Affichage des données de supervision
- au niveau du système :
 - CPU, mémoire, occupation disque...
- au niveau PostgreSQL :
 - TPS, tailles, cache hit ratio, verrous...

L'agent collecte périodiquement des données de métrologie que ce soit au niveau du système (CPU, mémoire, occupation disque, charge) ou au niveau de l'instance Postgres (TPS, tailles, cache hit ratio, verrous, taux d'écriture des WAL, etc). Ces données sont ensuite envoyées à l'interface puis historisées dans le *repository*.

L'interface offre au DBA une consultation de ces données sous forme de graphiques, navigables dans le temps. Ces données sont également comparées lors de la réception à des seuils (configurables) afin de déclencher une alerte si la valeur excède le seuil défini. Ces alertes sont visibles soit sur le *dashboard*, soit sur une page dédiée appelée *Status*. L'historique des alertes est également navigable dans le temps.

PLUGIN MONITORING : ALERTING

- Envoi d'alerte en cas de dépassement de seuils
- Par mail ou par SMS
- Configuration possible par utilisateur

Le plugin *Monitoring* permet également d'envoyer des alertes en cas de dépassement de seuils des sondes. Ces alertes peuvent être faites par mail ou par SMS. Les notification par SMS sont effectuées grâce au service [Twilio](https://www.twilio.com/)¹⁸.

¹⁸<https://www.twilio.com/>

On peut activer / désactiver les alertes pour chaque utilisateur.

PLUGIN STATUS

- Vision synthétique de l'état des sondes
- Accès au graphique détaillé

La vue *Status* permet d'afficher l'état des différentes sondes de l'instance sur une seule page. Pour chaque sonde, les niveaux d'alertes *Warning* et *Critical* sont précisés.

On cliquant sur la sonde, on accède au graphique historique. On peut à partir de cette vue :

- changer son nom,
 - activer / désactiver la sonde,
 - modifier les niveau d'alertes *Warning* et *Critical*.
-

PLUGIN CONFIGURATION

- Visualisation des paramètres de l'instance
- Modification par l'ordre SQL **ALTER SYSTEM**
- Recharge de la configuration possible

Permet un accès simplifié en lecture et écriture aux paramètres de configuration de l'instance Postgres. La modification des paramètres s'effectue avec l'ordre SQL **ALTER SYSTEM**.

Dans le cas où une modification de paramètre nécessite le redémarrage de l'instance, une alerte s'affiche en haut de cette page.

PLUGIN MAINTENANCE

- Volume disque par :
 - base de données / schéma / table
- des tables, index, toast et fragmentation
- Lancement d'opérations de **ANALYSE**, **VACUUM** ou **REINDEX**

Ce module permet de visualiser pour chaque objet de l'instance le volume des tables et des index. Pour chacun de ces types d'objets, il permet de voir le volume de fragmentation présent.

On peut naviguer dans les histogrammes des volumes de plus en plus finement :

- par base de données,
- par schéma d'une base de données,
- par table et index d'un schéma.

On peut choisir de trier les objets selon différents critères : nom, taille totale, taille des tables, taille des index, volume de fragmentation...

En sélectionnant une table, on peut y lancer des opérations de maintenance :

- **ANALYSE**
- **VACUUM**
- **REINDEX** de la table ou d'un index

Des messages d'information alertent le DBA du besoin d'exécution de certaines opérations.

PLUGIN NOTIFICATIONS

- Visualisation des opérations effectuées depuis temBoard

La page **Notifications** permette de réaliser un audit des connections et actions effectuées depuis l'interface de temBoard.

POINTS FAIBLES DE TEMBOARD

- Pas de gestion des sauvegardes
- Pas cloud public ready
- Pas d'accès en observateur au niveau de l'agent
 - accès administrateur uniquement

temBoard est encore en phase de développement, certaines fonctionnalités intéressantes ne sont pas encore implémentées mais devraient l'être dans un avenir proche.

Par exemple, temBoard ne gère pas les sauvegardes physiques ou logiques. Il ne sait pas non plus alerter en cas d'erreur sur des sauvegardes.

L'agent doit être colocalisé avec l'instance. Ce n'est pas possible dans le cloud public.

Une autre problématique est que tout accès à l'agent se fait avec des droits d'administration. Il n'est pas possible de donner uniquement des droits d'observation à un utilisateur authentifié auprès d'un agent.

POINTS FORTS DE TEMBOARD

- Outil multi-fonctionnalités
- Accès centralisés
- Orienté PostgreSQL *uniquement*

L'objectif de temBoard est de proposer un seul et unique outil permettant l'administration courante et la supervision d'un parc d'instance Postgres. Il reste encore du chemin pour atteindre ce but, mais le développement est actif.

CONCLUSION

- Un système est pérenne s'il est bien supervisé
- Supervision automatique importante
- Utiliser le bon outil suivant la problématique

Une bonne politique de supervision est la clef de voûte d'un système pérenne. Pour cela, il faut tout d'abord s'assurer que les traces et les statistiques sont bien configurées. Ensuite,

l'installation d'un outil d'historisation, de création de graphes et de génération d'alertes, est obligatoire pour pouvoir tirer profit des informations fournies par PostgreSQL.

Les trois outils graphiques présentés demandent une configuration préalable. S'ils ne sont pas utiles pour alerter en cas de problèmes, leur utilisation permet de découvrir rapidement les origines du problème et de le corriger :

- pgBadger analyse les logs de l'instance et permet de pointer les erreurs et axes d'améliorations des applications.
- PoWA analyse en temps réel de l'intérieur de l'instance, les requêtes s'y exécutant. Il permet de pointer les requêtes lentes et peut même proposer la création d'index.
- temBoard analyse lui aussi en temps réel l'activité de l'instance, mais aussi du système l'hébergeant. Il permet également d'agir sur l'instance en modifiant ses paramètres ou en stoppant des requêtes.

TRAVAUX PRATIQUES

PGBADGER

Installation

Suivant les outils à disposition sur votre ordinateur, vous avez 2 moyens à disposition pour installer le blaireau : utiliser le paquet sur le dépôt communautaire ou récupérer le projet github.

Installer le paquet sur debian sera aussi simple qu'un :

```
$ apt install --yes pgbadger
$ pgbadger --version
pgBadger version 11.1
```

Si Perl est disponible sur votre ordinateur, vous pouvez opter pour la récupération du projet sur github et le lancement du script en direct :

```
$ git clone https://github.com/darold/pgbadger.git
$ export PATH="$PATH:$(pwd)/pgbadger/pgbadger"
$ pgbadger --version
pgBadger version 11.1
```

Les logs PostgreSQL utilisés pour le TP sont stockés dans le répertoire `~dalibo/Documents`.

Dans tous les cas, vous pouvez également récupérer ces logs en suivant le lien suivant : https://public.dalibo.com/workshop/workshop_supervision/logs_postgresql.tgz.

L'archive contient 9 fichiers de traces de 135 Mo chacun :

<https://dalibo.com/formations>

Supervision graphique en PostgreSQL

```
$ tar xzf logs_postgresql.tgz
$ cd logs_postgresql
$ du -sh *
135M    postgresql-11-main.1.log
135M    postgresql-11-main.2.log
135M    postgresql-11-main.3.log
135M    postgresql-11-main.4.log
135M    postgresql-11-main.5.log
135M    postgresql-11-main.6.log
135M    postgresql-11-main.7.log
135M    postgresql-11-main.8.log
135M    postgresql-11-main.9.log
```

Premier rapport

Nous allons commencer par créer un premier rapport à partir du premier fichier de logs. L'option `-j` est à fixer à votre nombre de processeurs :

Si vous utiliser le script Perl :

```
$ pgbadger -j 4 postgresql-11-main.1.log
```

Le fichier de rapport *out.html* est créé dans le répertoire courant. Avant de l'ouvrir dans le navigateur, lançons la création du rapport complet :

```
$ pgbadger -j 4 --outfile rapport_complet.html postgresql-11-main.*.log
```

Pendant la création du rapport complet, ouvrez-le fichier *out.html* dans votre navigateur. Parcourir les différents onglets et graphiques.

Une fois le rapport complet créé, ouvrez le dans votre navigateur.

On peut observer dans les sections *Connections* et *Sessions* un nombre de sessions et de connexions proches. Chaque session doit ouvrir une nouvelle connexion. Ceci est assez coûteux, un processus et de la mémoire devant être alloués.

La section *Checkpoints* montre une activité d'écriture normale.

La section *Temp Files* permet, grâce au graphique temporel, de vérifier si un ralentissement de l'instance est corrélé à un volume important d'écriture de fichiers temporaires. Le rapport permet également de lister les requêtes ayant généré des fichiers temporaires. Suivant les cas, on pourra tenter une optimisation de la requête ou bien un ajustement de la mémoire de travail, `work_mem`.

La section *Vacuums* liste les différentes tables ayant fait l'objet d'un `VACUUM`.

Le section *Locks* permet d'obtenir les requêtes normalisées ayant le plus fait l'objet d'attente sur verrou. Le rapport pgBadger ne permet pas toujours de connaître la raison

de ces attentes.

La section *Queries* fournit une connaissance du type d'activité sur chaque base de données : *application web*, *OLTP*, *data warehouse*. Elle permet également, si le paramètre `log_line_prefix` le précise bien, de connaître la répartition des requêtes selon la base de données, l'utilisateur, l'hôte ou l'application.

La section *Top* est très intéressante. Elle permet de lister les requêtes normalisées ayant pris le plus de temps. Fixer le paramètre `log_min_duration_statement` à 0 permet de lister toutes les requêtes exécutées. Une requête peut ne mettre que quelques dizaines de millisecondes à s'exécuter. Mais si elle est lancée des millions de fois par heure, une optimisation de quelques pourcent peut avoir un impact non négligeable.

Dans la section *Top*, normalisée les plus lentes et le nombre de fois où elles ont été exécutées.

Format des logs

Dans certains cas, pgBadger peut ne pas réussir à détecter automatiquement le format du fichier ou le préfixe des lignes. Pour le format du fichier, on pourra utiliser l'option `--format`. On pourra par exemple préciser que le fichier est au format csv ou bien qu'il s'agit d'un log de pgbouncer.

Pour le préfixe des lignes, on pourra utiliser l'option `--prefix`. En cas de doute sur la configuration et si l'accès à l'instance est possible, on pourra récupérer l'information via la requête :

```
$ psql -At -c 'show log_line_prefix'
%t [%p]: user=%u,db=%d,app=%a,client=%h
```

Rapport précis

La vue des verrous nous informe d'un problème sur la base de données *bank* vers 16h50. Nous allons réaliser un rapport spécifique sur cette base de données et cette période :

```
$ pgbadger -j 4 --outfile rapport_bank.html --dbname bank \
  --begin "2018-11-12 16:45:00 CET" --end "2018-11-12 16:55:00 CET" \
  postgresql-11-main.*.log
```

Nous voulons connaître plus précisément quelles requêtes venant de l'IP 192.168.0.84 et avoir une vue plus fine des graphiques. Nous allons créer un rapport en filtrant par client et en calculant les moyennes par minute :

```
$ pgbadger -j 4 --outfile rapport_host_89.html --dbclient 192.168.0.89 \
  --average 1 postgresql-11-main.*.log
```

Il est également possible de filtrer par application avec l'option `--appname`.

Mode incrémental

Les fichiers de logs sont volumineux. On ne peut pas toujours conserver un historique assez important. pgBadger peut parser les fichiers de log et stocker les informations dans des fichiers binaires. Un rapport peut-être construit à tout moment en précisant les fichiers binaires à utiliser.

```
$ mkdir /tmp/incr_report
$ pgbadger -j 4 -I --noreport -O /tmp/incr_report/ postgresql-11-main.1.log
$ tree /tmp/incr_report
/tmp/incr_report
├── 2018
│   ├── 11
│   │   └── 12
│   │       ├── 2018-11-12-25869.bin
│   │       ├── 2018-11-12-25871.bin
│   │       ├── 2018-11-12-25872.bin
│   │       └── 2018-11-12-25873.bin
│   └── LAST_PARSED
└── 3 directories, 5 files
$ cat /tmp/incr_report/LAST_PARSED
2018-11-12 16:36:39 141351476 2018-11-12 16:36:39 CET [17303]: user=banquier,
db=bank,app=gestion,client=192.168.0.84 LOG: duration: 0.2
```

Le fichier *LAST_PARSE* stocke la dernière ligne analysée. Dans le cas d'un fichier de log en cours d'écriture, pgBadger commencera son analyse au prochain lancement de ,

Le fichier *postgresql-11-main.1.log* occupe 135 Mo. On peut le compresser pour le réduire à 7 Mo. Voyons l'espace occupé par les fichiers incrémentaux de pgBadger :

```
$ mkdir /tmp/incr_report
$ pgbadger -j 4 -I --noreport -O /tmp/incr_report/ postgresql-11-main.1.log
$ du -sh /tmp/incr_report/
340K /tmp/incr_report/
```

On pourra reconstruire à tout moment les rapports avec la commande :

```
$ pgbadger -I -O /tmp/incr_report/ --rebuild
```

Ce mode permet de plus construire des rapports réguliers journalier et hebdomadaire. Vous pouvez vous référer à la [documentation](http://pgbadger.darold.net/documentation.html#INCREMENTAL-REPORTS)¹⁹ pour en savoir plus sur ce mode incrémental.

¹⁹<http://pgbadger.darold.net/documentation.html#INCREMENTAL-REPORTS>

POWA

Afin de créer de l'activité SQL sur notre environnement PoWA, nous allons générer du trafic SQL via l'outil `pgbench`. Pour cela, nous allons lancer les commandes suivantes dans un terminal :

```
$ sudo -iu postgres
postgres$ pgbench -c 4 -T 1000 bench
```

Installation

L'installation d'un environnement fonctionnel PoWA en version 3.2 a été effectué sur la machine virtuelle.

En attendant que l'activité soit visible, étudions comment nous avons mis en place PoWA sur la machine virtuelle. Nous avons tout d'abord installé `powa-archivist` sur le serveur hébergeant l'instance :

```
# installation de powa et des différents modules disponible
apt install postgresql-12-powa postgresql-12-pg-qualstats \
    postgresql-12-pg-stat-kcache postgresql-12-hypopg
```

La configuration de l'instance a été mise à jour pour charger les modules au démarrage, récupérer les temps d'accès des entrées / sorties et récupérer des métriques dans PoWA toutes les 15 secondes :

```
shared_preload_libraries = 'pg_stat_statements,pg_stat_kcache,pg_qualstats,powa'
track_io_timing = on
powa.frequency = '15s'
```

L'instance a ensuite été redémarrée :

```
# systemctl restart postgresql@12-main.service
```

Nous avons ensuite procédé, via l'utilisateur `postgres` à l'installation de la base de données PoWA :

```
postgres$ psql -c 'CREATE DATABASE powa'
postgres$ psql -d powa -c 'CREATE EXTENSION btree_gist'
postgres$ psql -d powa -c 'CREATE EXTENSION pg_stat_statements'
postgres$ psql -d powa -c 'CREATE EXTENSION pg_qualstats'
postgres$ psql -d powa -c 'CREATE EXTENSION pg_stat_kcache'
postgres$ psql -d powa -c 'CREATE EXTENSION powa'
postgres$ psql -d powa -c 'CREATE EXTENSION hypopg'
```

Puis créé l'utilisateur `powa_user` :

```
postgres$ psql -c "CREATE ROLE powa_user LOGIN SUPERUSER PASSWORD 'powa'"
```

Supervision graphique en PostgreSQL

Nous avons ensuite mis en place la base de données *bench* sur laquelle nous sommes en train de générer de l'activité :

```
postgres$ psql -c "CREATE DATABASE bench;"
postgres$ pgbench -i bench
```

Pour montrer l'intérêt de PoWA pour la suggestion d'index, nous avons supprimé une contrainte :

```
postgres$ psql -d bench \
    -c "ALTER TABLE pgbench_accounts DROP CONSTRAINT pgbench_accounts_pkey"
```

Nous avons ensuite procédé à l'installation de *powa-web* à partir du dépôt git :

```
dalibo:~/git$ git clone https://github.com/powa-team/powa-web.git
dalibo:~/git$ cd powa-web
dalibo:~/git/powa-web$ git checkout 682d0a78311e9ef44f8f355d0903657b189cac4a
dalibo:~/git/powa-web$ cp powa-web.conf-dist powa-web.conf
```

Nous avons installé les paquets python nécessaire à son fonctionnement :

```
# apt install python-tornado python-sqlalchemy python-psycopg2
```

Visualisation

Nous devons lancer le logiciel *powa-web* :

```
dalibo:~/$ cd /git/powa-web
dalibo:~/git/powa-web$ ./powa-web
[I 191107 15:45:46 powa-web:12] Starting powa-web on http://0.0.0.0:8888
```

Ouvrir votre navigateur à l'adresse <http://127.0.0.1:8888>

Pour l'authentification, le nom d'utilisateur est « *powa_user* », le mot de passe « *powa* ».

La page principale vous pouvez maintenant permet de visualiser les différentes métriques par base de données.

En sélectionnant une base de données, on accède aux métriques par requêtes.

La sélection d'une requête permet d'accéder à des informations spécifique pour cette requête. Cette vue permet de voir si une requête change de comportement au cours du temps.

Choisir la base de données *bench*. Cliquer sur le bouton *Optimize Database*. Que constate-t-on ?

Choisir une requête qui procède à des mises à jour de la table *pgbench_accounts*. Naviguer dans l'onglet *Predicates*. Quel serait le gain si l'index suggéré était utilisé ?

TEMBOARD

L'installation d'un environnement fonctionnel temBoard en version 4.1 a été effectué sur la machine virtuelle.

Pour créer de l'activité SQL pour notre environnement temBoard, nous allons de nouveau générer du trafic SQL via l'outil **pgbench**. Pour cela, lancer les commandes suivantes dans un nouveau terminal :

```
$ sudo -iu postgres
postgres$ psql -p 5433 -c "CREATE DATABASE bench;"
postgres$ pgbench -p 5433 -i bench
postgres$ pgbench -p 5433 -c 2 -T 500 bench
```

Stopper le trafic lancée lors du TP PoWA sur l'instance en version 12 et le relancer avec plus de sessions :

```
postgres$ pgbench -c 8 -T 200 bench
```

Installation

En attendant que l'activité soit visible, étudions comment nous avons mis en place temBoard sur la machine virtuelle. L'installation d'un environnement fonctionnel temBoard a été effectuée grâce à **pip** :

```
apt install python-pip
pip install psycopg2-binary temboard temboard-agent
```

Pour bien fonctionner, temBoard a besoin d'un FQDN bien fonctionnel. On l'a défini dans le fichier **/etc/hosts** :

```
# hostname --fqdn
supervision
# sed -i 's/127.0.1.1\tsupervision/127.0.1.1\tsupervision.ws\tsupervision/' /etc/hosts
# hostname --fqdn
supervision.ws
```

Nous avons ensuite initié temBoard. Le script temBoard **auto_configure.sh** a créé la base de données et la configuration du logiciel :

```
# /usr/local/share/temboard/auto_configure.sh
Creating Postgres user, database and schema.
Creating system user temBoard.
Configuring temboard in /etc/temboard.
Using snake-oil SSL certificate.
```

Success. You can now start temboard using:

Supervision graphique en PostgreSQL

```
systemctl start temboard
```

Remember to replace default admin user!!!

PoWA utilise le port 8888. Nous allons faire tourner temBoard sur le port 9999. Nous ajoutons la clé de configuration du port dans la section `[temboard]` du fichier `/etc/temboard/temboard.conf` :

```
port = 9999
```

Nous avons ensuite configuré le service pour qu'il démarre de façon automatique :

```
# systemctl enable temboard
# systemctl start temboard
```

Dans l'interface de temBoard, nous avons déclaré un nouvel utilisateur administrateur pour l'accès aux agents, login : `tmbagent`, mot de passe : `temboard`.

Une fois l'interface utilisateur installée, il faut installer un agent par instance PostgreSQL à superviser. Nous avons installé un agent pour l'instance PostgreSQL 12-main en utilisant l'utilisateur `tmbagent` créé ci-dessus :

```
# export PGPORT=5432
# /usr/local/share/temboard-agent/auto_configure.sh https://supervision.ws:9999
# sudo -u postgres temboard-agent-adduser -c /etc/temboard-agent/12/main/temboard-agent.conf
# systemctl enable temboard-agent@12-main.service
# systemctl start temboard-agent@12-main.service
# sudo -u postgres temboard-agent-register \
    -c /etc/temboard-agent/12/main/temboard-agent.conf --host $(hostname --fqdn) \
    --port 3345 --groups default https://supervision.ws:9999
```

Nous avons procédé de même pour l'agent pour de l'instance PostgreSQL 11-main :

```
# export PGPORT=5433
# /usr/local/share/temboard-agent/auto_configure.sh https://supervision.ws:9999
# sudo -u postgres temboard-agent-adduser -c /etc/temboard-agent/11/main/temboard-agent.conf
# systemctl enable temboard-agent@11-main.service
# systemctl start temboard-agent@11-main.service
# sudo -u postgres temboard-agent-register \
    -c /etc/temboard-agent/11/main/temboard-agent.conf --host $(hostname --fqdn) \
    --port 3345 --groups default https://supervision.ws:9999
```

Visualisation

Ouvrir votre navigateur à l'adresse <https://127.0.0.1:9999>

Pour l'authentification, le nom d'utilisateur est **admin**, mot de passe **admin**.

Les instances sont nommées par leurs noms d'hôte. Les 2 instances sont donc nommées *supervision.ws*. Ici, le numéro de port et la version de PostgreSQL permettent de différencier entre les instances supervisées.

Cliquez sur **supervision.ws** de la version 12, et étudiez les différents modules.

Nous allons à présent verrouiller de manière exclusive un table de la base **bench** dans le but de bloquer l'activité. Pour cela, dans un autre onglet du terminal :

```
$ sudo -iu postgres psql bench
```

```
bench=# BEGIN;
```

```
bench=# LOCK TABLE pgbench_tellers IN EXCLUSIVE MODE;
```

Revenir sur le **Dashboard** temboard et attendre quelques instants, que constate-t-on ?

Cliquer sur **Status**, puis sur **Waiting sessions**.

Aller sur la vue **Activity** et naviguer entre les onglets **Running**, **Waiting**, **Blocking**.

Depuis l'onglet **Blocking**, cocher la ligne de la requête bloquante, puis cliquer sur **Terminate**, enfin confirmer.

Revenir sur le **Dashboard**, attendre quelques instants. Que constate-t-on ?

Nous allons maintenant ajouter un nouvel utilisateur, **alice**, qui aura accès uniquement à l'instance PostgreSQL en version 10 sur le port 5434.

Commençons par installer l'agent pour l'instance en version 10 en suivant la procédure utilisée pour les 2 autres agents. Utilisez l'utilisateur *tmbagent* pour connecter l'agent à l'UI.

Une fois l'agent installé et l'instance visible dans l'UI temBoard, nous allons ajouter une nouvelle utilisatrice et un nouveau groupe. Pour se faire, cliquer sur le bouton « Settings » en haut à droite de l'interface.

Nous allons créer :

- un nouveau groupe d'utilisatrices « user_pg10 »,
- une nouvelle utilisatrice « alice » dans le groupe « user_pg10 »,
- un nouveau groupe d'instances « instance_pg10 » accessibles par le groupe « user_pg10 »,
- enfin, ajouter l'instance en version 10 dans le groups d'instances « instance_pg10 ».

Supervision graphique en PostgreSQL

Déconnectons nous et vérifions ce que l'utilisatrice alicia peut visualiser.

Nous constatons que l'utilisatrice ne peut pas modifier le paramétrage de l'instance avec ses identifiants de connexion.

Si nous voulons que alicia puisse agir sur l'instance, nous devons la déclarer au niveau de l'agent. Nous pouvons le faire via le script `temboard-agent-adduser` :

```
# export PGPORT=5434
# sudo -u postgres temboard-agent-adduser -c /etc/temboard-agent/10/main/temboard-agent.conf
```

Vérifions ensuite que alicia peut agir sur les paramètres de l'instance.