

Projet Bases de données réparties

Stockage à travers 2 régions distinctes sur un SGBD relationnel réparti

Cas métier: Contrôle d'accès physique à des locaux d'entreprise

Lien du dépôt Git : <https://github.com/JiveOff/efrei-bdr-access-control>

Introduction	2
Analyse des besoins	3
Description du cas métier	3
Besoins spécifiques	3
Analyse	4
Volumétrie des données	4
Fréquence d'accès	4
Répartition géographique	4
Contraintes réglementaires	4
Conception de la base de données	6
Architecture	6
Modèle de données	7
Répartition des données	8
Notions	8
Régionalisation	8
Fragmentation des données sur le cluster Citus	8
Exploitation de la base de données	9
Fonctionnalités implémentées	10
Exemple de vue	10
Exemple de trigger	11
Fonctions métier	12
distributed.check_access	12
distributed.enter_workspace	12
Test du cas métier	12
Optimisations et tests	14
Scénario nominal	14
Scénarios avec séparation géographique	15
Scénario sans FDW	16
Scénario de simulation avec FDW	16
Pistes d'amélioration	17
CI/CD et environnements	17
RGPD	17
Nettoyage des ressources	17

Introduction

Dans le cadre du projet "Conception et déploiement d'un système de bases de données réparties en production" pour le module "ALTN83 - Bases de données réparties", nous avons choisi de mettre en place une solution de bases de données réparties utilisant Citus. Ce projet vise à répondre aux besoins d'une entreprise mondiale comptant des dizaine voire des centaines de milliers de salariés à travers le monde, qui souhaite sécuriser ses lieux de travail en contrôlant les accès aux différents espaces de travail et en journalisant ces accès pour répondre aux besoins de gestion des ressources humaines et aux potentiels audits.

Pour cela, nous avons analysé les besoins spécifiques de l'entreprise et les contraintes réglementaires qui s'appliquent à la gestion des données d'accès aux espaces d'entreprise, notamment en ce qui concerne la durée de stockage de ces informations et l'accès et la modification des données personnelles. Nous avons également estimé la volumétrie des données nécessaires et la fréquence d'accès à ces données pour dimensionner notre solution.

Nous avons ensuite conçu l'architecture de la base de données en utilisant le modèle de données adéquat et en répartissant les données sur plusieurs clusters Citus pour assurer une scalabilité horizontale et une haute disponibilité.

Enfin, nous avons implémenté et configuré notre solution en utilisant les outils et les technologies adaptés pour répondre aux besoins de l'entreprise et aux contraintes réglementaires. Dans les sections suivantes, nous détaillerons chacune de ces étapes pour présenter notre solution de bases de données réparties utilisant Citus.

Analyse des besoins

Description du cas métier

Une entreprise mondiale comptant environ 100000 salariés à travers le monde souhaite sécuriser ses lieux de travail. Pour cela, elle doit contrôler les accès aux différents espaces de travail pour répondre au besoin de sécurité des espaces et de confidentialité, journaliser les accès aux espaces pour répondre au besoin de gestion des ressources humaines et répondre aux potentiels audits et enfin journaliser les échecs d'accès à un espace afin de gérer les potentielles anomalies.

Pour répondre à ces besoins, nous pouvons donc collecter et traiter des données sur :

- Les personnes physiques : il s'agit des salariés de l'entreprise, qui seront identifiés par un numéro de matricule unique. Nous n'avons pas besoin de données supplémentaires, la correspondance entre le numéro de matricule et l'identité des salariés est une donnée qui sera stockée extérieurement à notre solution afin de "pseudonymiser" nos données.
- Les espaces de travail : nous devons répertorier tous les espaces de travail de l'entreprise. Ces espaces peuvent être de nature complexe (imbriqués par exemple), et chaque espace est délimité par un ensemble de portiques.
- Les droits d'accès : nous devons définir les règles d'accès aux différents espaces de travail en fonction des salariés. Ces règles seront formalisées sous la forme de droits d'accès, qui seront associés à chaque salarié et à chaque espace de travail. Les droits d'accès seront soumis à expiration et devront être renouvelés.
- Les tentatives et accès effectifs : nous devons journaliser toutes les tentatives d'accès aux espaces de travail, qu'elles soient couronnées de succès ou non. Ces journaux d'accès contiendront des données telles que la date et l'heure de la tentative, le numéro de matricule du salarié, l'espace de travail, le résultat de la tentative (accès autorisé ou refusé), etc. Ces données seront utiles pour la gestion des ressources humaines, mais aussi pour détecter les éventuelles anomalies et les tentatives d'intrusion.

En collectant et en traitant ces données, nous pourrions donc mettre en place un système de contrôle d'accès efficace et sécurisé, qui répondra aux besoins de l'entreprise en matière de sécurité des espaces de travail et de gestion des ressources humaines.

Besoins spécifiques

Pour répondre aux besoins de l'entreprise en matière de sécurité des lieux de travail, plusieurs aspects doivent être pris en compte. Voici quelques-uns des besoins clés qui devraient être considérés :

- **Scalabilité** : l'entreprise compte environ 100 000 salariés à travers le monde, ce qui signifie que la solution doit être capable de gérer un grand nombre d'utilisateurs et de points d'accès. La solution doit donc être conçue pour être évolutive et capable de s'adapter à la croissance de l'entreprise.
- **Disponibilité** : la sécurité des lieux de travail est une préoccupation constante, et la solution de sécurité doit donc être disponible 24 heures sur 24, 7 jours sur 7. Cela signifie qu'elle doit être conçue avec des fonctionnalités de redondance pour minimiser les temps d'arrêt et garantir la continuité de l'activité.
- **Performance** : la solution de sécurité doit être capable de traiter rapidement et efficacement un grand nombre de demandes d'accès. Cela signifie qu'elle doit être conçue pour minimiser les temps de latence et optimiser les performances, afin de garantir que les salariés puissent accéder aux lieux de travail dont ils ont besoin rapidement et facilement.
- **Conformité réglementaire** : l'entreprise doit se conformer à diverses réglementations en matière de sécurité et de confidentialité des données, telles que le RGPD. La solution de sécurité doit donc être conçue pour répondre à ces exigences réglementaires, en protégeant les données personnelles des salariés et en garantissant la conformité aux normes de sécurité.

C'est cet ensemble de besoins qui justifient l'utilisation d'une base de données répartie.

Analyse

Volumétrie des données

Pour fournir une estimation du volume de données nécessaire, il faut poser un certain nombre d'hypothèses. Supposons que:

- L'effectif concerné pendant les cinq dernières années environne les 100 000 employés.
- L'entreprise est implantée dans plusieurs régions du monde, et possède plusieurs milliers d'espaces de travail distincts.
- Un espace de travail est délimité par un nombre variable de portiques, en général de 2 à 15 (en moyenne 5).
- Les employés et invités sont amenés à franchir ces portiques plusieurs fois par jour, disons 8.
- Les employés travaillent en moyenne 250 jours.

On a alors:

- Environ 100 000 personnes
- Environ 5 000 espaces de travail
- Environ 25 000 portiques
- Environ $250/365 * 8 * 100\,000 \approx 550\,000$ franchissements de portiques journaliers, soit 135 000 000 en 3 mois
- Environ 125 000 000 d'enregistrements sur le temps de travail en 5 ans.

Nous pouvons donc estimer qu'au bout de 5 ans, en régime permanent, le volume de données total sera tout au plus de quelques dizaines de Gigaoctets. Ces volumes restent faibles dans le contexte d'une base de données et ne justifient pas à eux seuls l'utilisation d'une base de données répartie.

Fréquence d'accès

- Des centaines de personnes peuvent être créées ou supprimées chaque jour
- Des milliers d'informations personnelles peuvent être mises à jour chaque jour
- Des centaines de milliers de temps de travail sont créés / modifiés / supprimés par jour
- Quelques portiques ou espaces de travail peuvent être créés, supprimés ou mis à jour quotidiennement.
- Presque 1 million de franchissements de portique chaque jour donc le même nombre de lectures d'informations sur les portiques, les espaces de travail et les droits d'accès. C'est aussi la fréquence de journalisation de ces accès.
- Quelques centaines ou milliers de consultations de journaux d'accès dans le cadre d'audit ou de résolution de problèmes d'accès.

Répartition géographique

Le groupe possède des locaux à travers le monde, sur plusieurs continents. Certaines zones géographiques (par exemple l'Europe, les côtes Est et Ouest des États Unis) sont plus densément fournies en locaux que d'autres.

L'Europe se distingue au niveau réglementaire car possède les politiques les plus restrictives en termes de confidentialité et de sécurité des données.

Contraintes réglementaires

Un certain nombre de contraintes réglementaires s'appliquent à la gestion des données d'accès aux espaces d'entreprise. Cette réglementation est plus restrictive au sein de l'Union européenne qu'ailleurs, entre autres grâce à la RGPD.

Est contrainte notamment la durée de stockage de ces informations:

- Les données d'accès doivent être supprimées après 3 mois au maximum,
- Les données relatives au temps de travail des employés doivent être sauvegardées pendant 5 ans au minimum.

Ces données comprenant un certain nombre d'informations personnelles, l'accès et la modification des données doit être contrôlé:

- Seules les personnes habilitées peuvent accéder à ces données à des fins de gestion du personnel, de la paie ou de la sécurité.
- Toute opération sur ces données doit être journalisée.

De plus, les individus concernés disposent de droits à propos de leurs données, comprenant:

- La consultation,
- La rectification,
- Le recours.

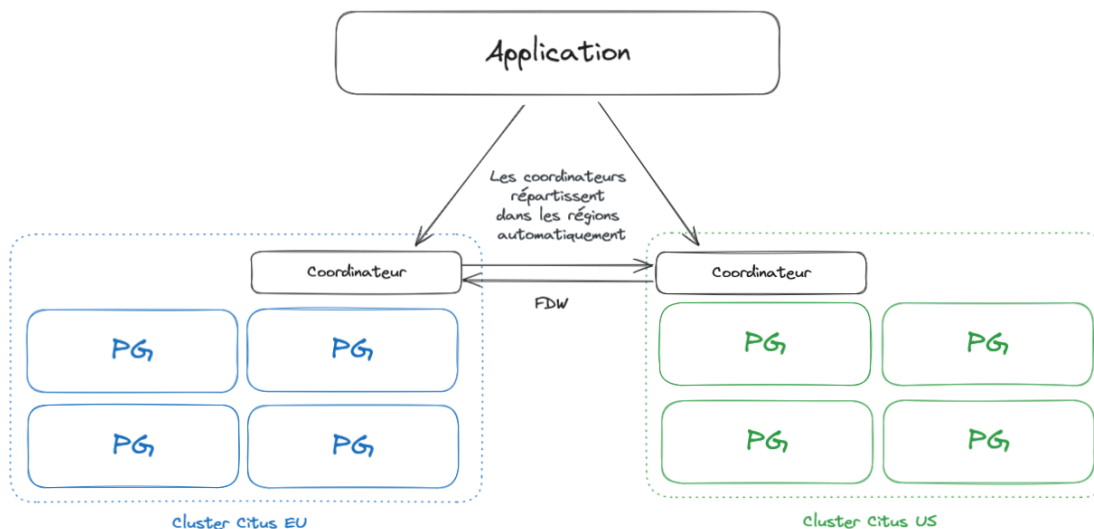
Nous devons donc rendre possible la modification et la lecture des données. La gestion des utilisateurs et des droits d'accès ainsi que la gestion du consentement sont réalisées au niveau applicatif et sont par conséquent en dehors du périmètre du projet.

Afin de faciliter la gestion des données à caractère personnel et de rester conforme à la réglementation, nous chercherons à limiter au maximum le volume de données personnelles stockées.

(Source: [CNIL](#))

Conception de la base de données

Architecture



Pour répondre au mieux aux besoins de disponibilité et aux contraintes réglementaires, nous avons opté pour une solution répartie géographiquement. Notre première action de répartition se fait au niveau de la virtualisation: un cluster se matérialise sous forme d’une machine virtuelle (lxc) sur l’hyperviseur Proxmox. Dans notre cas métier et pour réduire la latence dans les continents, nous mettons en œuvre deux clusters **EU (Europe) & US (Amérique)**.

	901 (bdr-eu-1)	28.6 %	30.9 %	3.1% of 4 ...	6 days 20:29...	1.0% of 12...	2.0 %	dmz	efrei
	902 (bdr-us-1)	38.8 %	29.6 %	2.7% of 4 ...	6 days 20:29...	0.9% of 12...	1.9 %	dmz	efrei

Les deux clusters sont identifiés avec leur nom d’hôte (et pas directement leur IP) et sont utilisés pour la communication inter-cluster nécessaire pour la mise en œuvre des *Foreign Data Wrappers*.

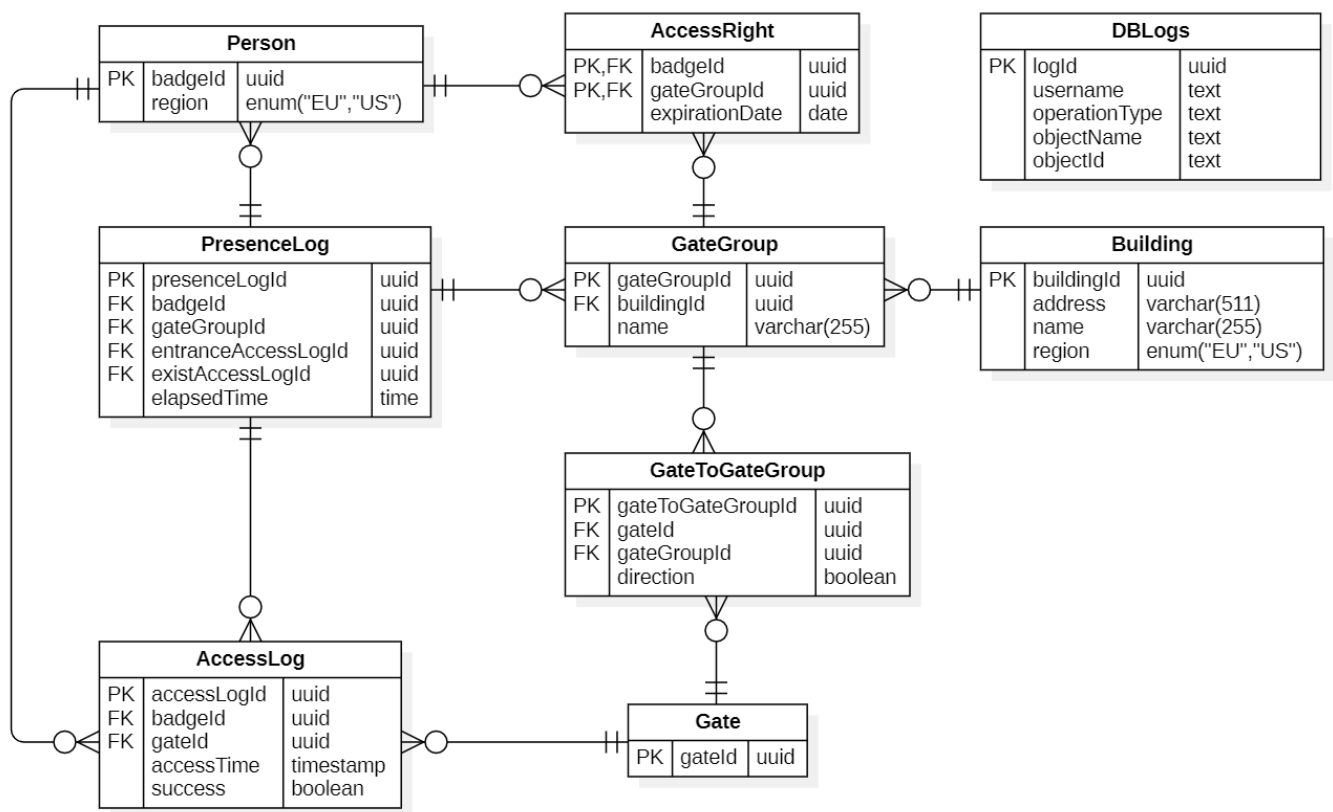
Chaque machine virtuelle possède un contexte Docker permettant de démarrer des services décrits par un fichier docker-compose.yml. Pour un cluster, ces services vont démarrer des conteneurs Docker et dans notre contexte du projet, nous démarrons une infrastructure type pour Citus avec un conteneur dédié au coordinateur et N conteneurs pour les workers, encadrés par le *membership manager* qui s’occupe d’ajouter les workers au coordinateur. Nous avons également ajouté un conteneur “doku” permettant de monitorer l’usage des ressources de stockage.

Quant au réseau, nous avons pris la précaution de n’exposer qu’un seul conteneur pour toute la machine: le coordinateur. Dans l’architecture type de Citus, le coordinateur est une base de données Postgres ne contenant que les métadonnées des workers ainsi que les tables. Citus s’occupe de la répartition des fragments de façon intelligente et automatique à travers les workers. Le coordinateur ne maintient aucune donnée des tables et s’occupe de l’interrogation auprès des workers.

Containers	
Name ↓↑	State ↓↑ Filter ▼
bdr-eu-1-doku-1	running
bdr-eu-1-worker-1-1	healthy
bdr-eu-1-worker-2-1	healthy
bdr-eu-1-worker-3-1	healthy
bdr-eu-1-worker-4-1	healthy
bdr-eu-1_manager	healthy
bdr-eu-1_master	healthy

- us_remote
 - foreign tables 9
 - accesslog
 - accessright
 - building
 - dblogs
 - gate
 - gategroup
 - gatetogategroup
 - person
 - presencelog
 - Database Objects

Modèle de données



- Person : Contient les données d'une personne : son matricule et sa région.
- Building : Représente un site de l'entreprise. Cette donnée est stockée afin d'offrir une classification claire des espaces de travail. Un site peut contenir plusieurs espaces de travail.
- GateGroup : Représente un espace de travail, délimité par un groupe de portiques.
- Gate : Représente une badgeuse, permettant d'ouvrir un portique. Ainsi, un portique unidirectionnel correspond à une Gate, mais un portique bi-directionnel correspond à deux Gates. Cette distinction est nécessaire afin de savoir si un employé rentre ou sort d'un espace de travail. Cette table contient un seul attribut, elle a une fonction référentielle.
- GateToGateGroup : Permet de créer une relation plusieurs à plusieurs entre Gate et GateGroup, et ajoute une information de direction. Cette information est nécessaire, car il est possible qu'en entrant

dans un espace de travail, on sorte d'un autre (la Gate est le même, mais la direction est différente pour chaque GateGroup).

- PresenceLog : Permet de journaliser les temps de présence, à des fins de gestion des ressources humaines.
- AccessLog : Permet de journaliser les tentatives d'accès, à des fins de sécurité et de journalisation.
- DBLogs : Permet de journaliser les opérations de mutation des données (Création, mise à jour, suppression).

Répartition des données

Notions

D'entrée de jeu, notre modèle de données a été pensé pour être performant et scalable sur des systèmes de bases de données distribuées mais pour y arriver, nous avons dû faire des prévisions qui s'accorderaient sur notre cas métier (voir la partie Volumétrie ci-dessus). Ces prévisions, une fois translatées vers notre modèle de données, se traduisent en une fragmentation principalement horizontale de nos données. La volumétrie n'étant pas située sur l'axe des colonnes, la fragmentation verticale n'a pas été considérée et n'affiche pas un support satisfaisant avec l'extension Postgres étudiée en cours. Il est tout de même à noter que Citus va au-delà de ces principes pour proposer, grâce à sa fragmentation, plusieurs stratégies de distribution, notamment pour optimiser les relations de clés étrangères, tout en appliquant des règles strictes pour rester dans le cadre de l'aspect distribué des données (notamment dans les facteur de réplication supérieurs à 1, où les relations sont assez compliquées à mettre en oeuvre étant donné que nous n'avons plus la garantie d'être en colocation avec nos relations).

Régionalisation

Notre système repose sur une répartition en deux temps, dans un premier temps, un des coordinateurs reçoit une requête sur une des vues distribuées, cette vue distribuée peut également avoir des triggers pour réaliser les opérations de INSERT, UPDATE & DELETE. Dans les cas de manipulation des données, il faut savoir où mettre la donnée, et nous réalisons cela avec deux champs région situés dans notre modèle de données sur

Person et *Building*. Notre modèle de données étant fortement relié avec des relations, nous pouvons remonter facilement aux tables contenant ce champ de région. Une fois la région déterminée, nous pouvons alors effectuer les opérations sur les *foreign tables* de la région correspondante. Il est à noter que pour des raisons de maintenance du code, nous partageons les mêmes scripts SQL pour tous les coordinateurs, cela nous permet de plus ou moins avoir les mêmes environnements peu importe la région. Il est néanmoins possible d'optimiser encore plus le code pour arriver à de bien meilleures performances.

```
CREATE TYPE Region AS ENUM ('EU', 'US');

CREATE TABLE IF NOT EXISTS Person (
    badgeId UUID PRIMARY KEY,
    region Region NOT NULL
);
```

Dans un deuxième temps, une fois que l'opération sur la *foreign table* est transmise aux tables du coordinateur, Citus prend le relais et s'occupe de la fragmentation sur ces workers sous-jacents.

Fragmentation des données sur le cluster Citus

Pour nos tables, nous avons décidé de distribuer:

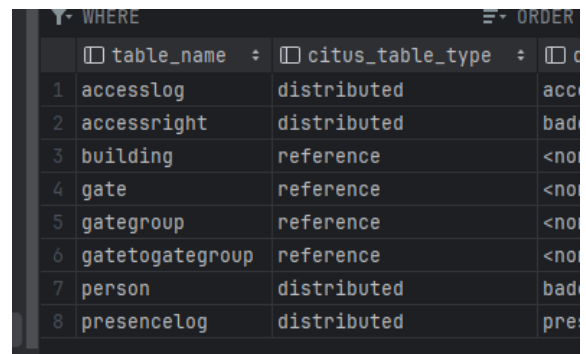
- *Person*, étant donné qu'il peut y en avoir 100 000 environ et que le modèle de données pourrait évoluer et avoir plus d'informations sur les personnes.
- *AccessRight* en colocation* avec *Person*, une personne peut avoir plusieurs accès, ce qui fait beaucoup d'enregistrements également.

- *AccessLog*, la journalisation étant un point vital de l'application et qu'elle sera substantielle niveau stockage.
- *PresenceLog*, pour la même raison que *AccessLog*.

* La colocation est une fonctionnalité offerte par Citus pour faire vivre les enregistrements liés sur le même nœud afin de pouvoir utiliser les contraintes, par exemple les clés étrangères. Dû à notre changement du replication factor ci-dessus, nous ne pouvons pas mettre de clé étrangère car nous ne sommes pas assurés que toutes les répliques aient un shard contenant la personne visée par la clé étrangère.

Pour le reste des tables, nous avons choisi de les distribuer en intégralité sous forme de tables de référence sur Citus. Cela veut dire que le contenu sera synchronisé entre tous les nœuds de travail. Nous pouvons nous permettre cela étant donné que le volume de données pour ces tables ne sera pas substantiel. Les tables de référence permettent également de faciliter les clés étrangères vers celles-ci étant donné que tout le contenu est présent sur tous les nœuds, ce qui optimise également les requêtes.

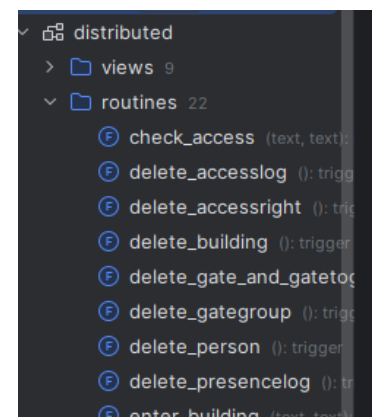
Afin de permettre une façon de récupérer les données si un worker d'une région tombe, nous avons décidé de mettre un [replication factor](#) de 2 sur le coordinateur. Cela permet d'avoir des répliques des shards Postgres selon ce même facteur de réplication. Celui-ci nous permet donc d'avoir plus de résilience en cas de dysfonctionnement d'un worker.



	table_name	citus_table_type	
1	accesslog	distributed	acc
2	accessright	distributed	bad
3	building	reference	<no
4	gate	reference	<no
5	gategroup	reference	<no
6	gatetogategroup	reference	<no
7	person	distributed	bad
8	presencelog	distributed	pre

Exploitation de la base de données

L'exploitation de la base de données entre les régions utilisera un modèle de gestion analogue au cours, celui de l'utilisation des FDW pour la communication inter-coordonateurs des deux régions. L'idée étant de centraliser la manipulation des données avec des vues pour y appliquer des règles avec des triggers gérant ainsi une première répartition des données sur les régions, nous avons mis en œuvre une panoplie de fonctions permettant de répondre à notre besoin métier.



distributed
views 9
routines 22
check_access (text, text)
delete_accesslog (): trigger
delete_accessright (): trigger
delete_building (): trigger
delete_gate_and_gatetog
delete_gategroup (): trigger
delete_person (): trigger
delete_presencelog (): trigger
enter_building (text, text)

Fonctionnalités implémentées

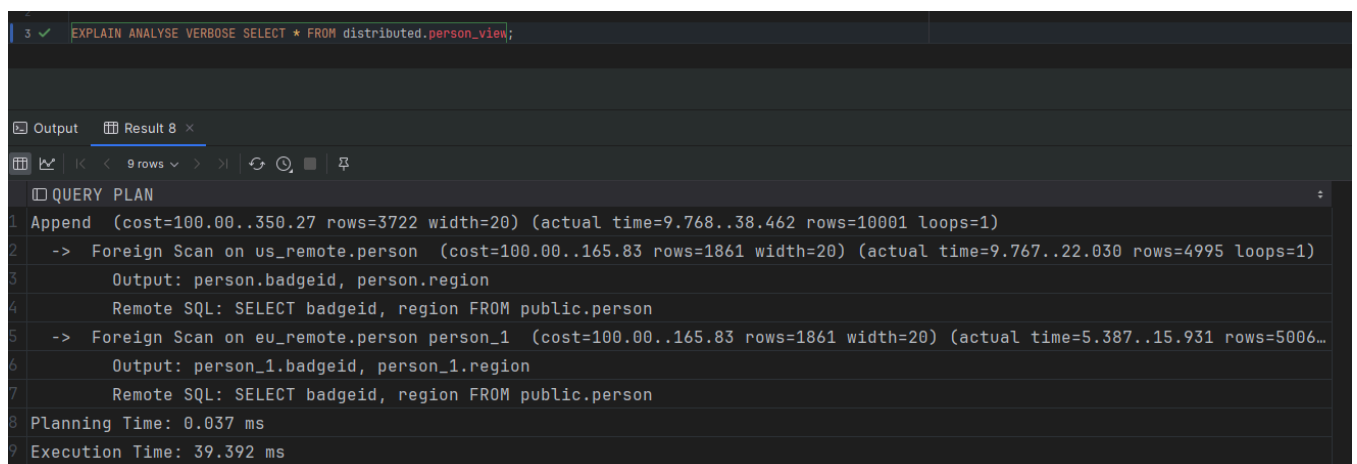
Dans les quelques exemples suivants, nous allons nous concentrer sur la table Person.

Exemple de vue

Afin de pouvoir utiliser notre base de données de façon totalement transparente, peu importe la région d'accès et où et comment sont stockées les données, nous avons créé un ensemble de vue, pour chaque table, dans le script [3 - views.sql](#).

```
-- Person
CREATE OR REPLACE VIEW distributed.person_view AS
    SELECT badgeId, region
    FROM us_remote.person
UNION ALL
    SELECT badgeId, region
    FROM eu_remote.person;
```

Ces vues sont simples : elles récupèrent toutes les lignes d'une table dans chacune des régions, puis font une jointure. En diagnostiquant la requête, on observe que cela fonctionne comme prévu :



The screenshot shows a database interface with a query plan for the statement: `EXPLAIN ANALYZE VERBOSE SELECT * FROM distributed.person_view;`. The plan is displayed in a table format with 9 rows. The first row is the 'Append' operation, which is the final step of the query. It has a cost of 100.00..350.27, 3722 rows, and a width of 20. The actual time taken was 9.768..38.462 ms, with 10001 rows and 1 loop. The subsequent rows show the 'Foreign Scan' operations for the two regions: 'us_remote.person' and 'eu_remote.person person_1'. Each scan has a cost of 100.00..165.83, 1861 rows, and a width of 20. The actual times were 9.767..22.030 ms and 5.387..15.931 ms respectively, with 4995 and 5006 rows and 1 loop each. The plan also shows the 'Output' and 'Remote SQL' for each scan. The 'Output' for the us_remote scan is 'person.badgeid, person.region' and for the eu_remote scan is 'person_1.badgeid, person_1.region'. The 'Remote SQL' for both is 'SELECT badgeid, region FROM public.person'. The final row of the plan shows the 'Planning Time: 0.037 ms' and 'Execution Time: 39.392 ms'.

	QUERY PLAN
1	Append (cost=100.00..350.27 rows=3722 width=20) (actual time=9.768..38.462 rows=10001 loops=1)
2	-> Foreign Scan on us_remote.person (cost=100.00..165.83 rows=1861 width=20) (actual time=9.767..22.030 rows=4995 loops=1)
3	Output: person.badgeid, person.region
4	Remote SQL: SELECT badgeid, region FROM public.person
5	-> Foreign Scan on eu_remote.person person_1 (cost=100.00..165.83 rows=1861 width=20) (actual time=5.387..15.931 rows=5006...
6	Output: person_1.badgeid, person_1.region
7	Remote SQL: SELECT badgeid, region FROM public.person
8	Planning Time: 0.037 ms
9	Execution Time: 39.392 ms

Exemple de trigger

```
CREATE OR REPLACE FUNCTION distributed.insert_person() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.region = 'US'::region THEN
        INSERT INTO us_remote.person (badgeId, region) VALUES (NEW.badgeId, NEW.region::region);
    ELSIF NEW.region = 'EU'::region THEN
        INSERT INTO eu_remote.person (badgeId, region) VALUES (NEW.badgeId, NEW.region::region);
    ELSE
        RAISE EXCEPTION 'Invalid region: %', NEW.region;
    END IF;

    PERFORM distributed.log(p_region: NEW.region, p_operationtype: 'INSERT', p_objectname: 'person', p_objectid: NEW.badgeId::text);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION distributed.delete_person() RETURNS TRIGGER AS $$
BEGIN IF ... END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER insert_person_trigger
INSTEAD OF INSERT ON distributed.person_view
FOR EACH ROW EXECUTE FUNCTION distributed.insert_person();

CREATE OR REPLACE TRIGGER delete_person_trigger
INSTEAD OF DELETE ON distributed.person_view
FOR EACH ROW EXECUTE FUNCTION distributed.delete_person();
```

Les triggers sont dans le script [4-triggers.sql](#).

Pour les personnes, nous avons choisi d'implémenter les méthodes d'insertion et de suppression, étant donné que modifier la région d'un utilisateur impliquerait une migration totale des données de celui-ci, nous ne traiterons pas de cela.

Dans un premier temps, nous vérifions les données de l'enregistrement, si nous mettons la région US, nous devons insérer dans les foreign tables de la région US par exemple. Et dans un second temps, nous insérons un log pour la traçabilité.

Ces fonctions sont alors utilisées par nos triggers pour l'exécution de la logique.

```
INSERT INTO distributed.person_view (badgeid, region) VALUES (gen_random_uuid(), 'EU'::region);

SELECT * FROM distributed.person_view;

```

	badgeid	region
1	b8c15cf8-7d87-4673-999d-0bd022142046	EU

```
SELECT * FROM eu_remote.person;

```

	badgeid	region
1	b8c15cf8-7d87-4673-999d-0bd022142046	EU

```
SELECT * FROM us_remote.person;

```

	badgeid	region
--	---------	--------

Fonctions métier

Nous fournissons également, dans le script [6-business.sql](#), deux fonctions utilisables dans notre cas métier.

`distributed.check_access`

Cette fonction permet de vérifier si une personne (désignée par “p_badgeld”) peut entrer dans un espace de travail, c’est-à-dire qu’elle possède un droit d’accès pour chaque espace de travail lié au portique.

En effet, un portique peut permettre d’entrer à la fois d’un espace, et de sortir d’un autre, ou bien peut permettre d’entrer dans deux espaces de travail en même temps (accès direct à un espace imbriqué dans l’autre). Un exemple concret de cette dernière situation est l’accès direct depuis l’extérieur à une salle serveur (accès à la DSI uniquement), mais qui fait bien partie des locaux de l’entreprise. En effet, le temps de travail dans la salle serveur doit aussi être comptabilisé comme temps de travail dans les locaux de l’entreprise. Pour accéder à cette salle, il faut donc deux droits : le droit d’accès à la salle serveur et le droit d’accès aux locaux de l’entreprise.

`distributed.enter_workspace`

Cette fonction permet de simuler l’entrée d’une personne dans un espace de travail. Voici la logique suivie:

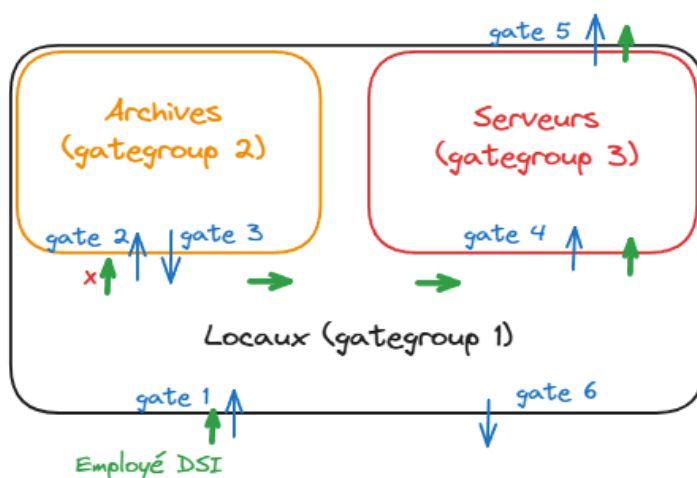
- Vérification du droit d’accès,
- Génération d’un journal de tentative d’accès avec le résultat de l’accès,
- Pour chaque espace de travail dans lequel on entre, on génère un journal de présence.
- Pour chaque espace de travail duquel on sort, on met à jour le dernier journal de présence créé, pour lequel il n’y a pas de tentative de sortie renseignée, en renseignant cette tentative et en remplissant la durée écoulée entre l’entrée et la sortie.

Test du cas métier

Pour tester nos fonctions métiers, nous avons réalisé un court script : [business-testing.sql](#).

Nous sommes dans une entreprise disposant de locaux, dans lesquels il y a une salle serveurs et une salle d’archives. La salle serveur possède une sortie directe vers l’extérieur.

Nous simulons le trajet d’un employé de la DSI, qui n’est autorisé à entrer que dans les locaux et la salle serveurs :



- 1 : L’employé entre dans les locaux (a le droit)
- 2 : L’employé tente d’entrer dans la salle des archives (refusé)
- 3 : L’employé entre dans la salle des serveurs
- 4 : L’employé sort des locaux de l’entreprise par la salle serveur.

Nous devrions donc avoir quatre logs d’accès et deux logs de présence (pour locaux et serveurs).

<pre>SELECT distributed.enter_workspace(p_badgeid: '00000000-0000-0000-0000-000000000001', p_gateid: '00000000-0000-0000-0000-000000000001'); SELECT distributed.enter_workspace(p_badgeid: '00000000-0000-0000-0000-000000000001', p_gateid: '00000000-0000-0000-0000-000000000002'); SELECT distributed.enter_workspace(p_badgeid: '00000000-0000-0000-0000-000000000001', p_gateid: '00000000-0000-0000-0000-000000000004'); SELECT distributed.enter_workspace(p_badgeid: '00000000-0000-0000-0000-000000000001', p_gateid: '00000000-0000-0000-0000-000000000005');</pre>																							
<table><tr><td><input type="checkbox"/> enter_workspace</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr></table>				<input type="checkbox"/> enter_workspace				1															
<input type="checkbox"/> enter_workspace																							
1																							
<pre>SELECT accesslogid, gateid, accesstime, success FROM distributed.accesslog_view WHERE badgeid = '00000000-0000-0000-0000-000000000001';</pre>																							
<table><tr><td><input type="checkbox"/> accesslogid</td><td><input type="checkbox"/> gateid</td><td><input type="checkbox"/> accesstime</td><td><input type="checkbox"/> success</td></tr><tr><td>1 5c5651c8-76f9-46c0-859c-d346aafd0430</td><td>00000000-0000-0000-0000-000000000001</td><td>2024-05-24 21:11:41.728028</td><td>• true</td></tr><tr><td>2 05f71186-dfde-4a0c-b57c-ad51fb10eff8</td><td>00000000-0000-0000-0000-000000000002</td><td>2024-05-24 21:11:43.097985</td><td>• false</td></tr><tr><td>3 56b885b6-38af-4f2d-993d-60494ac11ec7</td><td>00000000-0000-0000-0000-000000000005</td><td>2024-05-24 21:11:46.370668</td><td>• true</td></tr><tr><td>4 54881048-a907-470a-a1a1-94650a3303f2</td><td>00000000-0000-0000-0000-000000000004</td><td>2024-05-24 21:11:45.208229</td><td>• true</td></tr></table>				<input type="checkbox"/> accesslogid	<input type="checkbox"/> gateid	<input type="checkbox"/> accesstime	<input type="checkbox"/> success	1 5c5651c8-76f9-46c0-859c-d346aafd0430	00000000-0000-0000-0000-000000000001	2024-05-24 21:11:41.728028	• true	2 05f71186-dfde-4a0c-b57c-ad51fb10eff8	00000000-0000-0000-0000-000000000002	2024-05-24 21:11:43.097985	• false	3 56b885b6-38af-4f2d-993d-60494ac11ec7	00000000-0000-0000-0000-000000000005	2024-05-24 21:11:46.370668	• true	4 54881048-a907-470a-a1a1-94650a3303f2	00000000-0000-0000-0000-000000000004	2024-05-24 21:11:45.208229	• true
<input type="checkbox"/> accesslogid	<input type="checkbox"/> gateid	<input type="checkbox"/> accesstime	<input type="checkbox"/> success																				
1 5c5651c8-76f9-46c0-859c-d346aafd0430	00000000-0000-0000-0000-000000000001	2024-05-24 21:11:41.728028	• true																				
2 05f71186-dfde-4a0c-b57c-ad51fb10eff8	00000000-0000-0000-0000-000000000002	2024-05-24 21:11:43.097985	• false																				
3 56b885b6-38af-4f2d-993d-60494ac11ec7	00000000-0000-0000-0000-000000000005	2024-05-24 21:11:46.370668	• true																				
4 54881048-a907-470a-a1a1-94650a3303f2	00000000-0000-0000-0000-000000000004	2024-05-24 21:11:45.208229	• true																				
<pre>SELECT presencelogid, gategroupid, elapsedtime FROM distributed.presencelog_view WHERE badgeid = '00000000-0000-0000-0000-000000000001';</pre>																							
<table><tr><td><input type="checkbox"/> presencelogid</td><td><input type="checkbox"/> gategroupid</td><td><input type="checkbox"/> elapsedtime</td><td></td></tr><tr><td>1 b7c63bd2-6837-4ead-b3db-95b72346d91c</td><td>00000000-0000-0000-0000-000000000003</td><td>00:00:01</td><td></td></tr><tr><td>2 c5dfc372-3f08-42d8-93ae-95d2192a046a</td><td>00000000-0000-0000-0000-000000000001</td><td>00:00:04</td><td></td></tr></table>				<input type="checkbox"/> presencelogid	<input type="checkbox"/> gategroupid	<input type="checkbox"/> elapsedtime		1 b7c63bd2-6837-4ead-b3db-95b72346d91c	00000000-0000-0000-0000-000000000003	00:00:01		2 c5dfc372-3f08-42d8-93ae-95d2192a046a	00000000-0000-0000-0000-000000000001	00:00:04									
<input type="checkbox"/> presencelogid	<input type="checkbox"/> gategroupid	<input type="checkbox"/> elapsedtime																					
1 b7c63bd2-6837-4ead-b3db-95b72346d91c	00000000-0000-0000-0000-000000000003	00:00:01																					
2 c5dfc372-3f08-42d8-93ae-95d2192a046a	00000000-0000-0000-0000-000000000001	00:00:04																					

→ Nous constatons que l'accès à la salle des archives a bien été refusé. Les logs de présence correspondent également aux locaux, ici nous avons été 4 secondes dans les locaux et 1 seconde dans la salle serveur (ce qui correspond bien au temps d'exécution des requêtes, manuellement et une par une), ce qui est cohérent.

Optimisations et tests

Comme dit plus haut, nous avons réalisé un [script](#) permettant d'effectuer une insertion de masse d'entrées/sorties (appelées aussi *simulation*). Ces premiers tests en termes de performance et de latence ont été très concluants mais étant donné la configuration réseau assez avantageuse* et peu réaliste, ces tests doivent être pris avec un grain de sel. Nous avons tout de même effectué certaines requêtes de diagnostic afin de reporter les différentes valeurs en vue d'amélioration. Nous avons également eu l'opportunité d'avoir une séparation géographique.

* ping entre les deux clusters:

```
2024-05-24 20:22:25 UTC 0 0 2024-05-24 19:49:15 (0:00:29) - WARNING
root@bdr-eu-1:/srv/efrei-bdr-access-control/docker# ping bdr-us-1.epsilon
PING bdr-us-1.epsilon (10.21.1.13) 56(84) bytes of data.
64 bytes from bdr-us-1.epsilon (10.21.1.13): icmp_seq=1 ttl=64 time=0.066 ms
64 bytes from bdr-us-1.epsilon (10.21.1.13): icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from bdr-us-1.epsilon (10.21.1.13): icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from bdr-us-1.epsilon (10.21.1.13): icmp_seq=4 ttl=64 time=0.034 ms
64 bytes from bdr-us-1.epsilon (10.21.1.13): icmp_seq=5 ttl=64 time=0.034 ms
^C
--- bdr-us-1.epsilon ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 1007ms
```

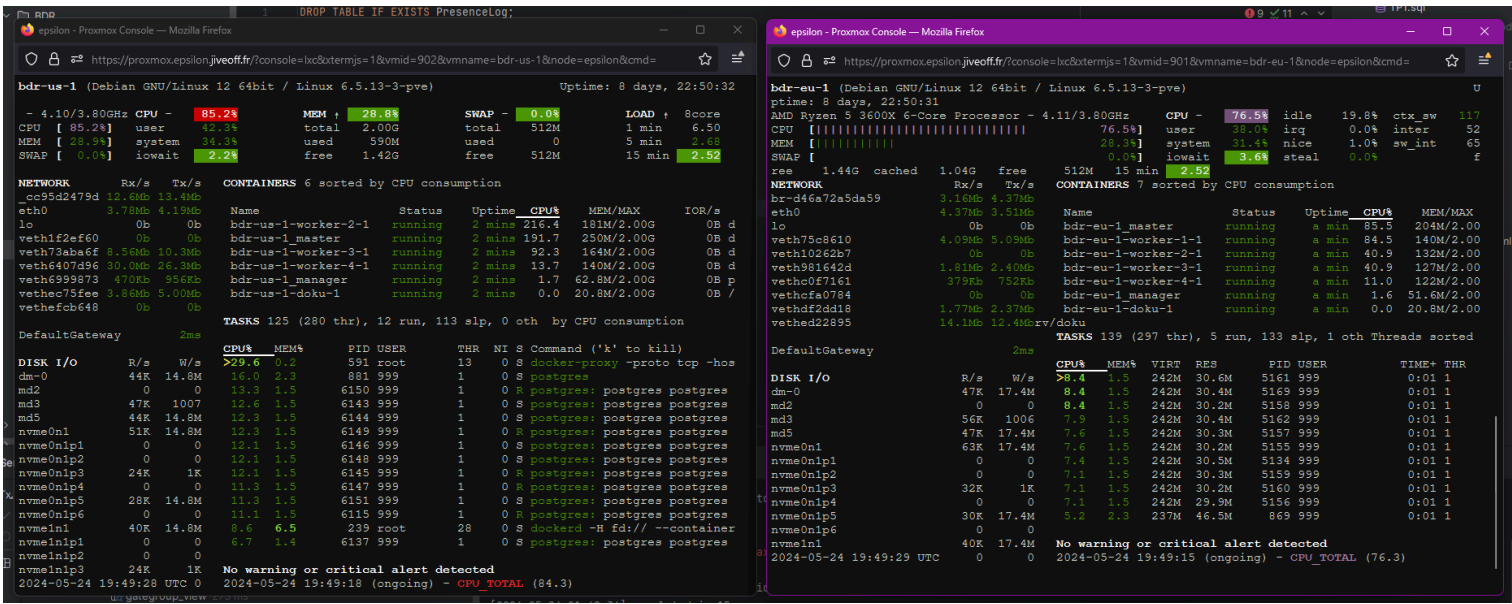
Scénario nominal

Environnement assez proche de la production: utilisation des FDW sur les coordinateurs avec 4 workers Citus en dessous des coordinateurs de chaque cluster (schéma *distributed*)

```
jiveoff's load-testing/ on main = 1 via 18.14.1
> bun run index.ts -- -s true
Fetched 12001 persons
Fetched 3000 gategroupgates
Created 24002 simulations
2024-05-24T19:41:02.456Z - Progress: 42/24002 (cur: 42/s - avg: 42/s)
2024-05-24T19:41:03.460Z - Progress: 93/24002 (cur: 51/s - avg: 47/s)
2024-05-24T19:41:04.464Z - Progress: 139/24002 (cur: 46/s - avg: 46/s)
2024-05-24T19:41:05.478Z - Progress: 187/24002 (cur: 48/s - avg: 47/s)
2024-05-24T19:41:06.488Z - Progress: 235/24002 (cur: 48/s - avg: 47/s)
2024-05-24T19:41:07.499Z - Progress: 288/24002 (cur: 53/s - avg: 48/s)
2024-05-24T19:41:08.507Z - Progress: 339/24002 (cur: 51/s - avg: 48/s)
2024-05-24T19:41:09.511Z - Progress: 387/24002 (cur: 48/s - avg: 48/s)
2024-05-24T19:41:10.522Z - Progress: 441/24002 (cur: 54/s - avg: 49/s)
2024-05-24T19:41:11.526Z - Progress: 492/24002 (cur: 51/s - avg: 49/s)
2024-05-24T19:41:12.539Z - Progress: 543/24002 (cur: 51/s - avg: 49/s)
2024-05-24T19:41:13.542Z - Progress: 592/24002 (cur: 49/s - avg: 49/s)
2024-05-24T19:41:14.546Z - Progress: 642/24002 (cur: 50/s - avg: 49/s)
2024-05-24T19:41:15.549Z - Progress: 693/24002 (cur: 51/s - avg: 50/s)
2024-05-24T19:41:16.558Z - Progress: 744/24002 (cur: 51/s - avg: 50/s)
```

Le script de simulation (en mode transactionnel) mesure environ 50 entrées/sorties par seconde.

Par contre, si nous désactivons le mode transactionnel, les chiffres sont beaucoup plus élevés au prix d'une consommation assez substantielle des ressources:



Par contre, si nous désactivons le mode transactionnel, les chiffres sont beaucoup plus élevés au prix d'une consommation assez substantielle des ressources:

Nous constatons que la VM s'occupant de l'ingestion des données monte à constamment 90% du CPU, le cluster EU étant à 70%. Il est à noter que les VM des clusters EU & US sont dotées de 2GB de RAM avec un CPU virtualisé 8 coeurs cadencés à 4.2GHz en moyenne.

Les chiffres de notre simulation indiquent que le système actuel peut donc supporter environ 200 entrées/sorties par seconde, ce qui s'avère très correct mais soulignant également les diverses améliorations que nous pourrions mener.

Scénarios avec séparation géographique

Pour les scénarios ci-dessous, nous introduisons un nouveau cluster: eu-2 se situant assez loin de us-1 (eu-2 venant en remplacement de eu-1). La différence avec us-1/eu-1, c'est que celui-ci possède plus de puissance de calcul au niveau du CPU (10 coeurs, i9 12k)

```
root@bdr-eu-2:/srv/efrei-bdr-access-control/docker# ping bdr-us-1.epsilon
PING bdr-us-1.epsilon (10.21.1.13) 56(84) bytes of data:
64 bytes from 10.21.1.13 (10.21.1.13): icmp_seq=1 ttl=62 time=9.23 ms
64 bytes from 10.21.1.13 (10.21.1.13): icmp_seq=2 ttl=62 time=10.1 ms
64 bytes from 10.21.1.13 (10.21.1.13): icmp_seq=3 ttl=62 time=9.37 ms
64 bytes from 10.21.1.13 (10.21.1.13): icmp_seq=4 ttl=62 time=11.0 ms
```

Il est à noter la grande différence de ping (et implicitement la bande passante), celle-ci joue énormément dans les requêtes par seconde.

Scénario sans FDW

Nous testons maintenant les performances liées à Citus sur le cluster eu-2, où nous faisons des requêtes directement aux tables réparties (schéma *public*).

Nous enlevons les communications via FDW:

```

jiveoff's load-testing/ on ymain = ~2 via js 18.14.1
> bun run .\index.ts -- -p 10000
Created 10000 persons
person - 2024-05-24T20:33:49.482Z | Progress: 1360/10000 (cur: 1360/s - avg: 1360/s - peak: 1360/s)
person - 2024-05-24T20:33:50.482Z | Progress: 2858/10000 (cur: 1498/s - avg: 1429/s - peak: 1498/s)
person - 2024-05-24T20:33:51.483Z | Progress: 4415/10000 (cur: 1557/s - avg: 1472/s - peak: 1557/s)
person - 2024-05-24T20:33:52.483Z | Progress: 5832/10000 (cur: 1417/s - avg: 1458/s - peak: 1557/s)
person - 2024-05-24T20:33:53.483Z | Progress: 7348/10000 (cur: 1516/s - avg: 1470/s - peak: 1557/s)
person - 2024-05-24T20:33:54.483Z | Progress: 8806/10000 (cur: 1458/s - avg: 1468/s - peak: 1557/s)
person - 2024-05-24T20:33:55.486Z | Progress: 9786/10000 (cur: 980/s - avg: 1398/s - peak: 1557/s)
person - Elapsed: 7835ms
Inserted persons

```

On peut constater un pic à 1500 personnes par seconde.

On pourrait donc faire l'hypothèse que le traitement que nous faisons sur le trigger *INSTEAD OF* pour la création des personnes est un *bottleneck*, or, nous ne faisons qu'un insert sur la foreign table des person & dbLogs sur le cluster correspondant au *NEW.region* mentionné sur la requête.

On peut alors songer à avoir un autre système, autre que les FDW, pour assurer la répartition géographique, mais cela sort du cadre de ce projet.

Scénario de simulation avec FDW

```

2024-05-24T20:56:52.655Z - Progress: 3924/20000 (cur: 39/s - avg: 40/s)
2024-05-24T20:56:53.669Z - Progress: 3949/20000 (cur: 25/s - avg: 39/s)
2024-05-24T20:56:54.678Z - Progress: 3977/20000 (cur: 28/s - avg: 39/s)
2024-05-24T20:56:55.690Z - Progress: 4007/20000 (cur: 30/s - avg: 39/s)
2024-05-24T20:56:56.701Z - Progress: 4038/20000 (cur: 31/s - avg: 39/s)

```

Le scénario de simulation entre les deux clusters fait des pics à environ 50 entrées/sorties par seconde.

Ces chiffres font sens étant donné le ping entre les deux clusters faisant un x100 voire plus environ par rapport à si les deux clusters étaient sur le même hyperviseur.

Pistes d'amélioration

CI/CD et environnements

Nous avons l'habitude de créer plusieurs environnements sur nos projets et de le tenir *iso-prod* en employant une méthodologie de CI/CD, mais nous avons décidé pour ce projet de nous en tenir à des déploiements manuels pour pouvoir appréhender au mieux le sujet même des bases de données réparties. Nous considérons alors que la mise en place d'une méthodologie CI/CD, dans le cadre de ce projet, serait un axe d'amélioration.

Nous avons pu mettre en œuvre ces méthodologies notamment avec les outils Terraform & Ansible, ce qui nous aurait été grandement utile pour le déploiement et le provisionnement des clusters à travers nos hyperviseurs. Néanmoins, la mise en place de cette méthodologie à travers les pipelines pourrait complexifier le processus et ne toucherait que à la partie infrastructure et non la partie données pour des raisons d'intégrité. Nous pourrions tout de même faire en sorte de déployer l'infrastructure et exécuter des scripts de reprise si besoin.

Ce projet aurait pu également être un bon starting point pour apprendre Kubernetes mais dans le cas précis de Citus, il aurait fallu créer un opérateur dédié à la gestion des workers et de leur enregistrement auprès du coordinateur (là où le membership-manager fourni par Citus le fait automatiquement au sein du docker-compose.yml fourni).

RGPD

Le RGPD a été longuement considéré sur ce projet et est la racine de notre architecture où l'Europe serait isolée des autres régions, mais cela a été plus compliqué que prévu. Par ailleurs, nous avons, lors du projet, retiré le champ *name* des personnes pour répondre au besoin de pseudonymisation imposé par ce règlement. Il peut alors être convenu qu'il y ai une liste qui n'appartiendrait pas à cette application répartie qui permettrait de faire le lien entre la personne et son identifiant de badge, cela permettrait également de centraliser le consentement de la personne, mais cela sort du cadre du projet.

Une deuxième solution serait de mettre en œuvre une solution de chiffrement globalisée sur tous les clusters. On pourrait alors faire l'hypothèse qu'un premier chiffrement se ferait au niveau du stockage en blocs sur les VM que nous utilisons. Deuxièmement, on pourrait également appliquer un autre chiffrement symétrique intervenant à partir de la couche applicative, où cette couche posséderait une clé permettant le chiffrement et déchiffrement de certaines données jugées sensibles, comme le nom ou autre données à caractère personnel.

Nettoyage des ressources

Nous voulions initialement, afin de respecter la réglementation mais aussi de ne pas stocker de données (logs) inutiles, réaliser un nettoyage periodique des données :

- Tous les logs d'accès de plus de 3 mois sont supprimés
- Tous les logs d'accès de plus de 5 ans sont supprimés
- Tous les logs d'opération de plus de 1 semaine sont supprimés.

La solution initialement envisagée était l'utilisation de l'extension `pg_cron`. Nous arrivions à faire fonctionner cette extension localement sur un coordinateur, mais à cause de problèmes de configuration que nous n'avons pu résoudre, le membership manager ne fonctionnait plus.

L'alternative envisagée était de garder les fonctions créées dans [5-jobs.sql](#), mais de les planifier en lançant un script au déploiement de la base de données. Ce programme aurait également pu exécuter tous les scripts sql de notre dossier [scripts](#) (create_tables, setup, views, triggers, jobs, business puis planification des jobs...).

Malheureusement, nous n'avons pas eu le temps de mettre en place cette solution.