Engineer

RBAC

Droits attribués à des rôles plutôt qu'à des personnes

Ce que c'est

Le RBAC décide de ce qu'une personne peut faire d'après les rôles qu'elle détient plutôt que d'après son identité. Ces lettres signifient role-based access control, ou contrôle d'accès fondé sur les rôles. Le NIST, où le modèle a été formalisé, parle aussi de « sécurité fondée sur les rôles ». Un rôle est un ensemble nommé de permissions, par exemple « employé comptable » ou « lecture seule en production ». Les utilisateurs reçoivent des rôles, et chaque permission qu'un utilisateur exerce lui vient de l'un d'eux.

David Ferraiolo et Rick Kuhn, du NIST, ont proposé le modèle général en 1992 comme alternative aux contrôles d'accès obligatoire et discrétionnaire de l'époque. En 1996, Sandhu, Coyne, Feinstein et Youman l'ont décomposé en couches : RBAC0 avec utilisateurs, rôles et permissions, RBAC1 ajoutant les hiérarchies de rôles, RBAC2 ajoutant des contraintes et RBAC3 réunissant les deux. Ces deux lignées ont été unifiées pour devenir la norme ANSI/INCITS 359-2004, révisée en 2012. Son modèle de référence comporte quatre composantes. Le RBAC de base, seule composante obligatoire dans tous les systèmes, contient les affectations des utilisateurs aux rôles et des permissions aux rôles, ainsi que la session, dans laquelle un utilisateur active un sous-ensemble des rôles qui lui sont attribués. Le RBAC hiérarchique permet à un rôle supérieur d'hériter d'un rôle inférieur. La séparation statique des tâches limite les rôles pouvant être attribués ensemble à un utilisateur, tandis que la séparation dynamique limite ceux pouvant être actifs simultanément dans une session. Une personne peut ainsi détenir à la fois le rôle qui crée un paiement et celui qui l'approuve sans utiliser les deux dans la même session.

La différence avec un groupe ordinaire est plus mince qu'il n'y paraît. La FAQ du NIST l'explique ainsi : un groupe est généralement un ensemble d'utilisateurs, et la plupart des mécanismes de groupe permettent aussi d'attacher une permission directement à un utilisateur. La FAQ qualifie cette possibilité de

« faille », car elle rend difficile le contrôle de la relation entre utilisateurs et permissions. Le RBAC comble cette faille en attachant les permissions uniquement aux rôles. Une autre phrase de la même FAQ en donne la raison : les rôles d'une organisation restent assez stables, alors que les utilisateurs et les permissions sont nombreux et changent vite. Faire passer tous les accès par les rôles simplifie donc leur gestion et leur examen.

Les bases de données implémentent souvent directement ce modèle. Depuis la version 8.1, PostgreSQL n'a qu'un seul type d'entité, le rôle, qui peut agir comme utilisateur, comme groupe ou comme les deux. Un rôle destiné à servir de groupe n'a généralement pas l'attribut LOGIN, et l'appartenance à ce rôle s'accorde comme tout autre privilège. Deux options d'appartenance règlent la manière dont un membre utilise les privilèges du groupe : INHERIT les lui donne automatiquement, tandis que SET lui permet de prendre ce rôle avec SET ROLE. Cela se rapproche de l'idée du RBAC consistant à n'activer un rôle que dans la session qui en a besoin.

```
-- the role carries the permissions; nobody logs in as it
CREATE ROLE reporting NOLOGIN;
GRANT USAGE ON SCHEMA sales TO reporting;
GRANT SELECT ON ALL TABLES IN SCHEMA sales TO reporting;

-- a person receives the role, never the table
CREATE ROLE alex LOGIN;
GRANT reporting TO alex;

-- a powerful role that has to be switched on deliberately
CREATE ROLE schema_admin NOLOGIN;
GRANT schema_admin TO alex WITH INHERIT FALSE;
-- later, in the one session that needs it: SET ROLE schema_admin;
```


Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que le RBAC ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des comptes, mots de passe et droits attribués selon les besoins exacts de chaque poste, puis retirés le jour du départ.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 1 000 règles RBAC pour sécuriser Linux — 1,000 règles d'accès écrites pour combler une faille de sécurité

1 000 règles RBAC pour sécuriser Linux Contactez-nous →

Pourquoi c’est important

Le contrôle d’accès doit répondre à deux questions qui tirent dans des directions différentes : cette requête est-elle autorisée maintenant, et qui peut faire quoi dans l’ensemble du système ? La plupart des mécanismes répondent bien à la première. Le RBAC est conçu pour la seconde, car sa réponse est une courte liste de rôles et de leurs membres, qu’un auditeur ou un responsable peut effectivement lire. Le NIST résume en une phrase l’autre approche, le contrôle d’accès fondé sur les attributs, dans lequel chaque décision évalue des règles portant sur les attributs de l’utilisateur, de la ressource et du contexte : il « peut être facile à mettre en place mais complexe à gérer ».

La décision pratique consiste à déterminer où les rôles ne suffisent plus. Le modèle de 1992 accorde l’accès « uniquement si » les exigences relatives au rôle sont satisfaites, formulation choisie pour laisser place à des vérifications supplémentaires au moment de l’accès. Un rôle de médecin permet de lire des dossiers médicaux, mais pas ceux de tous les patients : la relation entre ce médecin et ce patient est un fait portant sur les données, pas un intitulé de poste. C’est là qu’un second mécanisme trouve sa place, par exemple la sécurité au niveau des lignes dans la base de données ou une règle d’attributs dans l’application, superposé aux rôles sans les remplacer.

Nous recommandons de commencer par modéliser les accès sous forme de rôles nommés d’après les tâches réellement exercées, avec des permissions attachées uniquement aux rôles et jamais directement à un compte de connexion. Gardez une liste de rôles assez courte pour qu’une personne puisse la lire entièrement. N’ajoutez des règles au niveau des lignes ou des attributs que lorsqu’un rôle ne peut manifestement pas définir la limite. Laissez la base de données imposer tout ce qu’elle peut, car ses contrôles s’appliquent à chaque client, y compris au chemin de code oublié.

À quoi cela ressemble en pratique

Dans un poste d’ingénierie des bases de données occupé de 2022 à 2024, l’accès à chaque environnement de développement, instance applicative, base PostgreSQL ou MS SQL et serveur Linux a été encadré par 1 000 règles RBAC. Le modèle était la partie simple ; le volume constituait le défi. Les permissions ont été réparties en catégories réutilisables, comme « accès en lecture seule aux bases de production », puis associées à des rôles tels que développeurs, instances applicatives et administrateurs de bases de données. Les développeurs ne pouvaient atteindre que leurs serveurs désignés, et les instances applicatives

avaient des permissions limitées pour empêcher les déplacements latéraux. Ansible appliquait les règles de manière cohérente dans tous les environnements, et leur documentation permettait aux équipes de les auditer ou de les modifier. Le résultat livré était cet ensemble de règles par rôle et une méthode reproductible pour les appliquer et les examiner.

Les rôles agissent rarement seuls. Lorsqu'une application cartographique est passée de MSSQL à PostgreSQL avec PostGIS, le contrôle d'accès fondé sur les rôles a été mis en place avec la sécurité au niveau des lignes pour appliquer les politiques de protection des données à plusieurs groupes d'utilisateurs. C'est la répartition habituelle du travail : les rôles décident quelles tables un groupe peut utiliser, et les politiques de ligne décident quelles lignes à l'intérieur de ces tables.

La version la plus stricte met les tables entièrement hors de portée. Dans une couche de données PostgreSQL centrée sur les fonctions, réalisée dans le cadre d'une activité exercée depuis 2025, chaque opération de l'application est l'une des 1 275 fonctions réparties dans 34 schémas. Le rôle sous lequel l'API se connecte, mariner, possède EXECUTE sur les fonctions applicatives et USAGE sur les schémas, sans rien d'autre : environ 4 000 attributions explicites de privilèges plutôt qu'une attribution générale. Les fonctions s'exécutent en SECURITY DEFINER, sont détenues par un rôle distinct sans connexion et ont un search_path fixé. Le compte superutilisateur reste réservé aux migrations et à cron. Même compromise, l'API serait limitée à ce que permettent les fonctions. Le coût est lui aussi explicité : un nouveau champ nécessite une migration et une modification de fonction.

Où les problèmes surviennent

- **Les rôles se multiplient jusqu'à devenir illisibles.** Chaque exception devient un nouveau rôle, comme « développeur sauf facturation » ou « développeur plus un rapport », et il y a bientôt davantage de rôles que de personnes. Cette explosion détruit la raison d'être du RBAC : une liste assez courte pour être examinée. Attribuez deux ou trois rôles à une personne plutôt que d'en créer un par exception, et fusionnez ceux dont les permissions sont devenues identiques.

- **Une permission est accordée directement à un compte de connexion.** Un ticket urgent, un GRANT direct à un utilisateur, et celui-ci possède un accès qu'aucun examen des rôles ne montrera. C'est la faille décrite par le NIST pour les systèmes de groupes, recréée à la main. N'accordez les permissions qu'aux rôles et faites rechercher par la requête de contrôle les privilèges détenus par des rôles capables de se connecter.
- **Les valeurs par défaut donnent accès avant les rôles.** Dans PostgreSQL, PUBLIC désigne tous les rôles et reçoit par défaut EXECUTE sur les nouvelles fonctions ainsi que CONNECT et TEMPORARY sur les bases. Si une base a été mise à niveau depuis PostgreSQL 14 ou une version antérieure, PUBLIC peut aussi créer des objets dans le schéma nommé public. Révoquez délibérément ces privilèges par défaut, idéalement dans la transaction qui crée l'objet.
- **Les rôles ne font que s'accumuler.** Les personnes changent de poste et gardent l'ancien rôle à côté du nouveau. Un accès temporaire devient permanent parce que personne n'est chargé de le retirer. Chaque attribution doit avoir une raison et une date de réexamen, et le départ d'un poste devrait entraîner le retrait des rôles qui lui étaient liés.
- **Certains comptes ignorent les règles.** Les superutilisateurs et les rôles BYPASSRLS contournent la sécurité au niveau des lignes. Le propriétaire d'une table le fait normalement aussi, sauf si elle est configurée avec FORCE ROW LEVEL SECURITY. Une application qui se connecte comme propriétaire ou superutilisateur transforme chaque politique de ligne en décoration : connectez-la donc avec un rôle qui ne possède rien.

Pour aller plus loin

- Projet NIST sur le contrôle d'accès fondé sur les rôles — la page archivée du projet : histoire du modèle, norme de 2004 et révision de 2012.
- FAQ du NIST sur le RBAC — rôles et groupes, de RBAC0 à RBAC3, séparation des tâches et quatre composantes de la norme.
- PostgreSQL : appartenance aux rôles — rôles de groupe, attribution et révocation de l'appartenance, et options INHERIT et SET.
- PostgreSQL : politiques de sécurité des lignes — refus par défaut, rôles contournant les politiques de ligne et FORCE ROW LEVEL SECURITY.
- PostgreSQL : privilèges — privilèges de PUBLIC par défaut et manière de les révoquer.
- SQL Server : rôles au niveau de la base de données — rôles fixes et définis par l'utilisateur dans une base SQL Server.

Glossaire

Choisissez selon vos contraintes réelles

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

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

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

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

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

Imprimé depuis https://engineer.company/fr/glossary/rbac/