Engineer

RBAC

Permisos concedidos a roles en vez de a personas

Qué es

RBAC decide qué puede hacer alguien según los roles que tiene, en vez de según quién es. Las siglas significan control de acceso basado en roles; NIST, donde se formalizó el modelo, también lo denomina «seguridad basada en roles». Un rol es un conjunto de permisos identificado por un nombre, como «auxiliar de contabilidad» o «acceso de solo lectura a producción». Se asignan roles a los usuarios, y cada permiso que ejerce un usuario le llega a través de uno de ellos.

David Ferraiolo y Rick Kuhn, de NIST, propusieron el modelo general en 1992 como alternativa al control de acceso obligatorio y al discrecional de entonces. En 1996, Sandhu, Coyne, Feinstein y Youman lo dividieron en niveles: RBAC0 con usuarios, roles y permisos; RBAC1 añade jerarquías de roles; RBAC2 añade restricciones; y RBAC3 incorpora ambas cosas. Las dos líneas se unificaron y dieron lugar a la norma ANSI/INCITS 359-2004, revisada en 2012. Su modelo de referencia tiene cuatro componentes. El RBAC básico, el único que debe tener todo sistema, contiene las asignaciones de usuarios a roles y de permisos a roles, además de la sesión, en la que un usuario activa un subconjunto de sus roles asignados. El RBAC jerárquico permite que un rol superior herede uno inferior. La separación estática de funciones restringe qué roles pueden asignarse juntos a un usuario, y la separación dinámica restringe cuáles pueden estar activos a la vez en una sesión. Así, alguien puede tener tanto el rol que ordena un pago como el que lo aprueba sin usar ambos en una misma sesión.

La diferencia respecto de un grupo normal es menor de lo que parece. Las preguntas frecuentes de NIST lo explican así: un grupo suele ser un conjunto de usuarios, y la mayoría de los mecanismos de grupos también permiten asignar un permiso directamente a un usuario. Las preguntas frecuentes lo llaman una «laguna» que dificulta controlar la relación entre usuarios y permisos. RBAC la cierra al asignar permisos únicamente a roles. Otro pasaje de las mismas preguntas frecuentes explica el motivo: los roles de una organización se

mantienen bastante estables, mientras que los usuarios y los permisos son numerosos y cambian deprisa. Canalizar todo acceso a través de roles simplifica, por tanto, su administración y revisión.

Las bases de datos suelen implementar el modelo directamente. Desde la versión 8.1, PostgreSQL tiene un único tipo de entidad, el rol, que puede actuar como usuario, grupo o ambos. Un rol destinado a ser un grupo normalmente no tiene LOGIN, y la pertenencia a él se concede como cualquier otro privilegio. Dos opciones de pertenencia controlan cómo utiliza un miembro los privilegios del grupo: INHERIT le da esos privilegios automáticamente, y SET le permite adoptar el rol con SET ROLE, algo próximo a la idea de RBAC de activar un rol solo en la sesión que lo necesita.

```
-- 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;
```

Utilice el concepto en una decisión

Aplique «¿Qué es RBAC?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cuentas, contraseñas y permisos organizados para que cada persona tenga exactamente el acceso que necesita y lo pierda el mismo día que deja la empresa.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 1.000 reglas RBAC para servidores Linux — 1,000 reglas de acceso escritas para cerrar una brecha de seguridad

1.000 reglas RBAC para servidores Linux Póngase en contacto →

Por qué importa

El control de acceso debe responder a dos preguntas que tiran en direcciones distintas: si esta solicitud está permitida ahora mismo y quién puede hacer qué en todo el sistema. La mayoría de los mecanismos responden bien a la primera. RBAC está diseñado para la segunda, porque su respuesta es una lista breve de

roles y de las personas que tienen cada uno, que un auditor o responsable realmente puede leer. El veredicto de NIST sobre la alternativa, el control de acceso basado en atributos, en el que cada decisión evalúa reglas sobre atributos del usuario, el recurso y el contexto, se resume en una frase: «puede ser fácil de configurar, pero complejo de administrar».

La decisión práctica es dónde dejan de bastar los roles. El modelo de 1992 concede acceso «solo si» se cumplen los requisitos del rol, una formulación elegida para dejar espacio a comprobaciones adicionales en el momento del acceso. Un rol de médico permite leer historiales clínicos, pero no el de cualquier paciente: la relación entre ese médico y ese paciente es un dato, no un cargo. Ahí corresponde un segundo mecanismo, como la seguridad por filas en la base de datos o una regla de atributos en la aplicación, superpuesto a los roles en vez de sustituirlos.

Se recomienda modelar primero el acceso mediante roles con nombres de tareas que las personas realmente desempeñan, con permisos asignados solo a roles y nunca directamente a una cuenta de inicio de sesión. Mantenga la lista de roles lo bastante breve como para que una persona pueda leerla entera. Añada reglas por filas o atributos solo donde se demuestre que un rol no basta para delimitar el acceso, y deje que la base de datos imponga todo lo que pueda: una comprobación hecha por la base de datos se aplica a todos los clientes, incluida la ruta de código que alguien olvidó.

Cómo se aplica en la práctica

En un puesto de ingeniería de bases de datos entre 2022 y 2024, el acceso a todos los entornos de desarrollo, instancias de aplicaciones, bases de datos PostgreSQL y MS SQL y servidores Linux quedó sujeto a 1.000 reglas RBAC. El modelo era la parte sencilla; el volumen era el reto. Los permisos se dividieron en categorías reutilizables, como «acceso de solo lectura a bases de datos de producción», y se asociaron a roles como desarrolladores, instancias de aplicaciones y administradores de bases de datos. Los desarrolladores solo accedían a sus servidores designados, y las instancias de aplicaciones tenían permisos limitados para impedir movimientos laterales. Ansible aplicaba las reglas de forma uniforme entre entornos, y estas se documentaron para que los equipos pudieran auditarlas o modificarlas. El resultado entregado fue el conjunto de reglas basadas en roles y un método repetible para aplicarlas y revisarlas.

Los roles rara vez actúan solos. Cuando una aplicación de mapas pasó de MSSQL a PostgreSQL con PostGIS, se incorporaron controles de acceso basados en roles junto con seguridad por filas para imponer políticas de protección de datos entre varios grupos de usuarios. Esa combinación es el reparto habitual de responsabilidades: los roles deciden a qué tablas puede acceder un grupo, y las políticas por filas deciden a qué filas dentro de ellas.

La versión más estricta deja las tablas totalmente fuera de alcance. En una capa de datos PostgreSQL basada en funciones, desarrollada en un puesto ocupado desde 2025, cada operación que realiza la aplicación es una de 1.275 funciones repartidas entre 34 esquemas. El rol con el que la API inicia sesión, mariner, tiene EXECUTE sobre las funciones de la aplicación y USAGE sobre los esquemas, y nada más: unas 4.000 concesiones explícitas en vez de una concesión general. Las funciones se ejecutan como SECURITY DEFINER, pertenecen a un rol separado sin inicio de sesión y tienen un search_path fijado; la cuenta de superusuario se reserva para migraciones y cron. Una API comprometida seguiría limitada a lo que permiten las funciones. El coste también queda escrito: un campo nuevo exige una migración y un cambio en una función.

Dónde surgen los problemas

- **Los roles se multiplican hasta que nadie puede leerlos todos.** Cada excepción se convierte en un rol nuevo, como «desarrollador excepto facturación» o «desarrollador más un informe», y pronto hay más roles que personas. Esa proliferación anula el motivo de RBAC: una lista lo bastante breve como para revisarla. Asigne a cada persona dos o tres roles en vez de crear uno por excepción, y fusione los roles cuyos conjuntos de permisos hayan pasado a ser idénticos.
- **Un permiso llega directamente a una cuenta de inicio de sesión.** Una incidencia urgente, un GRANT directo a un usuario, y este ya tiene un acceso que ninguna revisión de roles mostrará. Es la laguna que NIST describe en los sistemas de grupos, reconstruida a mano. Conceda permisos solo a roles y haga que la consulta de revisión busque privilegios en manos de roles que pueden iniciar sesión.
- **Los valores predeterminados dan acceso antes que cualquier rol.** En PostgreSQL, el nombre PUBLIC significa todos los roles y recibe de forma predeterminada EXECUTE sobre funciones nuevas y CONNECT y TEMPORARY sobre bases de datos. Si una base de datos se actualizó desde PostgreSQL 14 o una versión anterior, PUBLIC también puede crear objetos en el esquema llamado public. Revoque deliberadamente esos privilegios predeterminados, idealmente dentro de la transacción que crea el objeto.

- **Los roles solo crecen.** Las personas cambian de puesto y conservan el rol anterior junto al nuevo; el acceso temporal se vuelve permanente porque nadie tiene la tarea de retirarlo. Cada asignación necesita un motivo y una fecha de revisión, y abandonar un puesto debería retirar los roles asociados a él.
- **Algunas cuentas se saltan las reglas.** Los superusuarios y los roles con BYPASSRLS ignoran la seguridad por filas, y normalmente también lo hace el propietario de una tabla, salvo que esté configurada con FORCE ROW LEVEL SECURITY. Una aplicación que se conecta como propietario o superusuario convierte cada política por filas en decoración. Conéctela, por tanto, mediante un rol que no sea propietario de nada.

Lecturas adicionales

- Proyecto de control de acceso basado en roles de NIST — página archivada del proyecto: historia del modelo, la norma de 2004 y su revisión de 2012.
- Preguntas frecuentes de NIST sobre RBAC — roles frente a grupos, de RBAC0 a RBAC3, separación de funciones y los cuatro componentes de la norma.
- PostgreSQL: pertenencia a roles — roles de grupo, concesión y revocación de la pertenencia y las opciones INHERIT y SET.
- PostgreSQL: políticas de seguridad por filas — denegación predeterminada, roles que eluden las políticas por filas y FORCE ROW LEVEL SECURITY.
- PostgreSQL: privilegios — qué recibe PUBLIC de forma predeterminada y cómo revocarlo.
- SQL Server: roles de base de datos — roles fijos y definidos por el usuario dentro de una base de datos SQL Server.



Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es RBAC?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

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

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/rbac/