RFC

Fine-Grained Authorization in the Grafbase Enterprise Platform

Contents

Introduction

There is demand for finer grained authorization in the Grafbase Enterprise Platform. This RFC is an attempt at a principled, future-proof solution that will give us configurable, robust and performant access control that is maximally convenient for our users and does not require any additional infrastructure or setup as part of the Enterprise Platform.

The scope of this RFC is authorization in the Enterprise Platform:

- for human users logging in with the Dashboard or CLI,
- for non-human agents interacting with the Grafbase API, object storage and the telemetry sink. For example CI jobs, Grafbase Gateway, external integrations or AI agents.

This RFC will often reference OpenID Connect (OIDC) and OAuth 2.1 concepts, basic familiarity with them is assumed. Our current authentication solution is based on OIDC.

Requirements

Infrastructure simplicity

We want to avoid introducing additional infrastructure or setup as part of the Enterprise Platform. Fewer, simpler services mean a faster path to adoption, easier security reviews and generally a platform that is easier to operate.

Performance

Access control should not introduce excessive latency or network roundtrips.

Familiarity and convenience

As much as possible, the authorization model should be familiar to enterprises looking to adopt Grafbase. It should be documentable, and the path to customization should be clear and accessible.

Expressive policies

We want to enable fine-grained authorization decisions based on ownership of resources, team memberships, roles and other attributes. If the design does not accommodate this requirement, we will need to tack on more configurability to the solution later as a workaround — we should have enterprise use cases in mind from day one.

Customizability

The authorization rules are specific to organizations. We should have defaults, but everything must be tweakable to the needs of enterprise organizations adopting Grafbase.

Suitability to both the hosted and self hosted Enterprise Platform

Self-hosting is top of mind, but the new solution should also work for the multi-tenant deployment of the Grafbase Enterprise Platform at grafbase.com.

Integration

Enterprises have centralized authorization services — we will dive into that in more detail in this document. Ideally, we should support delegating authorization decisions to these services.

Landscape

Terminology

- In an access control decision, a **principal** is the subject of the decision. The principal is who or what wants to perform an action on a resource.

  > A security principal is any entity that can be authenticated by the operating system, such as a user account, a computer account, or the security groups for these accounts.
  >
  > — Microsoft Learn

- The **action** is the operation that the principal wants to perform on the resource. There are standard actions such as “read”, “write”, and “delete”, but many relevant access control actions do not fit in these standard verbs.
- A **resource** is either a category of objects (“organization settings”, “graphs”) or an object (“user”, “team”, “graph”) that is the object of an access control decision.

Models of authorization

- **Role-based access control (RBAC)** is the traditional model for authorization. Access is determined by roles. You assign permissions to a role (like editor or viewer), and then you assign users to those roles. A user gets all the permissions associated with their assigned role(s). In pure RBAC, categories of users (roles) have access to specific actions on categories of resources: “admins can edit teams”, “users with the engineer role can publish subgraphs”. There is no notion of ownership (“I am allowed to see my patients’ medical records”), conditional access (“employees can edit this data only during business hours”) or sharing (“I can view the documents shared with me by their owners”) in pure RBAC.
  - RBAC usually supports a hierarchical structure, where roles can inherit permissions from other roles. This allows for more flexibility and scalability.
  - RBAC is susceptible to role explosion, as soon as you have too many combinations of permissions to express in a small number of roles.
  - In real world scenarios, RBAC is often extended with some notion of ownership of resources, but it is often done in ad-hoc, application specific ways.
- **Attribute-based access control (ABAC)** is more dynamic and granular. Access decisions are made by evaluating policies based on attributes (characteristics) of the user, the resource they’re trying to access, and the environment (like time of day or location). For example, a policy might state: “Allow a manager to access only their reports’ performance reviews, and only during business hours.”. Attributes can include ownership information, such as the user’s department or the resource’s creator.
  - In ABAC, the role can be expressed as an attribute on the principal.

- **Relationship-based access control (ReBAC)** grants access based on a user's relationship to a resource. It answers the question, "Can this user perform this action on this resource because of how they are connected?" For example, you can edit a Google Doc because you are its owner, or you can see a photo on Instagram because you are a follower of the person who posted it. Relationships are rule based and transitive. This is a very powerful model for applications like Google Drive, where ownership can be shared ("I allowed this external user to view, but not edit, that document"), inherited ("ownership of a directory grants you ownership of its subdirectories"), or at the intersection of multiple rules, for example when a user belongs to a company-wide group that has `Viewer` access to a folder, but they have also been given a direct link that grants `Commenter` access to a specific document inside it. The system must resolve both the inherited 'viewer' rule from the folder and the direct 'commenter' rule from the link to determine the user's final permission on that document.

Modern enterprise authorization implementations use elements of all these models, where roles are still a relevant concept, layered in with attribute-based access control and fine-grained permissions for each instance of a resource. Even ostensibly ReBAC focused systems like SpiceDB and OpenFGA bake in a context of attributes (ABAC) when evaluating access decisions. These models complement each other. They can coexist in the same system.

Another term that has become common is **Policy-based access control (PBAC)**. In PBAC, a policy decision point evaluates policies to manage what an individual or system can do. These policies are essentially a set of rules and conditions that are centrally managed and can be as simple or as complex as needed. Instead of permissions being hard-coded into applications, PBAC externalizes these decisions, allowing for greater flexibility and easier management of access rights. That most often goes hand in hand with policy as code (see below). While RBAC, ABAC, and ReBAC represent different methodologies for determining access, PBAC can be seen as an overarching approach that often incorporates elements of these other models to enforce its policies.

Modern enterprise authorization

Policy as code

Authorization as Code (AaC) is the practice of managing and defining access control policies using code, rather than through manual configurations in a database or an admin UI. These policy files are treated just like application code: they are stored in version control (like Git), tested automatically, and deployed through a CI/CD pipeline. It is a subset of Policy as Code.

The auditability, traceability and composability benefits known from the usage of code and VCSs as sources of truth apply here in the same way as in other models like Infrastructure as Code (IaC).

In this model, authorization logic is also centralized and decoupled from application code: rules are managed in one central place instead of being scattered across one or more application's codebase.

Policies can be automatically tested for errors and vulnerabilities before they go live, dramatically reducing the risk of misconfigurations that could lead to data breaches. This makes the system easier to understand, manage, and update. Collaboration is also simplified:writing policies as readable code allows different stakeholders—from developers to security and product managers—to understand and collaborate on the rules that govern their application.

Examples:

- IAM configuration with Cloudformation or Terraform
- Rego policies in Open Policy Agent (OPA)
- SpiceDB schemas

Integration with OAuth 2.0 and OpenID Connect

Emerging standard: Authzen
Authzen is a standardization effort started in 2023 and hosted by the OpenID foundation. Authzen wants to be to authentication what OpenID Connect is to user authentication.

The actors and their roles are described by the **P*P model**:

- **Policy Enforcement Point (PEP)**. The protected resource server. That is to say, the service that sits in the critical path of application requests. In our case, the Grafbase API. This role is often played by API Gateways. That issue is orthogonal, but we are exploring fleshing out support for acting as an Authzen PEP in Grafbase Gateway.
- **Policy Decision Point (PDP)**. The authorization service. When a PEP receives a request, it sends a query to the PDP. The PDP then makes the access decision by evaluating the request against the policies it receives from the PAP and the data it gathers from the PIP. The PDP can also act as both PAP and PIP, depending on the implementation.
- **Policy Information Point (PIP)**. The PIP provides the PDP with the necessary context and attributes to make a decision. This can include information about the user (like their session validity), the resource, the environment, and other business-related attributes, in addition to the context provided by the PEP.
- **Policy Administration Point (PAP)**. The PAP is responsible for creating, managing, and updating the access control policies that the PDP will use. This allows for centralized control over authorization rules.

In this RFC, we will focus on the PEP and PDP. The PEP will be the Grafbase API. The PDP will be either embedded in the Grafbase API, or an external service, depending on the configuration of a specific Enterprise Platform deployment.

https://openid.net/how-authzen-and-shared-signals-caep-complement-each-other/

https://medium.com/identity-beyond-borders/the-four-horsemen-of-authorization-the-p-ps-pep-pdp-pap-and-pip-42717e445ce7

...particularly for access tokens.

Proposed solution

The general idea is authzen-based pluggable access control, with a default baked-in but customizable implementation.

As a principle, we should push as much of the ownership and… to the users' IdP.

But there is a tension. On the one hand, users of the hosted offering, or those with a more basic IdP and authorization setup, will want built-in roles and permission management, with the state managed inside the Grafbase platform. On the other hand, enterprises will want to rely on their existing IdP for fine grained scopes and Rich Authorization Requests (RAR), and on their enterprise policy decision points (PDP) for access control.

To reconcile these differing needs, the rest of the RFC will expose a solution that can be summarized as:

- on the **entitlements** side, *both* built-in roles, teams and ownership data, and custom scopes / RARs from the IdP
- on the **access control** side, *either* a built-in Cedar Policy based decision point (PDP) or pluggable

The core idea to enable both convenient built-in authorization as a reasonable default and delegating as much to external services for those that need it is the decoupling of the policy decision part of the architecture: the PDP can decide to take into account or ignore as much of the built-in attributes (for example, what team owns which subgraph in the Grafbase API database) as it wants.

The two extremes are on one end, the default built-in setup, and on the other end, a Grafbase Enterprise Platform deployment with no teams and no ownership data in Postgres, with all the entitlements and access control managed between the users' IdP and their PDP.

The interface between the application layer and the authorization layer in the Enterprise Platform will be Authzen, which will enable swapping out the built-in PDP with an external one through simple configuration.

The interface: Authzen, schema, entities

Built-ins

Built-in entitlements

Built-in PDP with Cedar

Access tokens

The Grafbase API as Authorization Server

Scoping with OAuth 2.0 Rich Authorization Requests

https://oauth.net/2/rich-authorization-requests/

Also see

https://spiffe.io/docs/latest/spiffe-about/overview/