

ENGINEERING CAPACITY

Build great features. Not authorization.

Nobody sets out to build an authorization engine. It starts as a role column and a few conditionals, and it becomes a system with its own latency budget, its own test suite and its own on-call rotation. The question is not whether your engineers could build it. It is whether owning the engine is the best use of them.

How it happens

Early on, authorization looks simple: an administrator manages settings, an editor changes a record, a viewer reads it. Then customers arrive with nested teams, custom roles, delegated administration, inherited permissions, external sharing, and an AI assistant that must retrieve only what its user may see.

Teams answer with role flags, join tables, middleware and conditionals — usually in more than one service. The same rule gets implemented twice, the two drift, and changing a permission becomes something nobody wants to be responsible for.

What this paper argues

That authorization deserves the same build-versus-buy discipline you already apply to payments, messaging and object storage — **and that the honest answer is sometimes to build.**

It sets out where the cost actually sits, what centralizing does and does not change, a cost model you can run with your own numbers, and what stays your team's responsibility either way. No savings percentage is claimed, because none would survive contact with a real estate.

You probably would not build your own Stripe, Slack or S3 because your product needs payments, messaging or storage. Those are essential and they are rarely why customers choose you. **Authorization is usually in the same category — and almost never gets the same scrutiny.**

CONTENTS

1. The hidden cost of building it
2. What centralizing actually changes
3. Why relationships, not just roles
4. A three-year cost model you can run
5. What remains your team's job
6. A practical next step

AUDIENCE

Engineering leaders and architects weighing build versus buy for authorization

POSITION

Written by AuthZed, who maintain SpiceDB. Caveats included rather than omitted

READING TIME

About ten minutes

1 · THE HIDDEN COST

It starts as application code. It becomes infrastructure.

The build is the part teams estimate. It is also the smaller part. What follows is five separate responsibilities, each of which keeps costing after launch.

RESPONSIBILITY	WHAT THE TEAM OWNS, PERMANENTLY
Permission model	Roles, relationships, inheritance, sharing and the exceptions that arrive one enterprise deal at a time
Application integration	Consistent enforcement across every API, service and background job — including the ones written next year
Security and testing	Preventing unauthorized access, and catching permission regressions that are silent by nature
Performance and consistency	Meeting a latency budget on the request path, and deciding what happens to a cached allow when access is revoked
Operations and debugging	Monitoring, incident response, upgrades, and answering “why could they see that?” at 2am

2 · WHAT CENTRALIZING CHANGES

From scattered rules to one place to ask

A central authorization service separates *evaluating* a permission from each application's business logic. Services ask one question — can this subject do this thing to this resource — and each application still enforces the answer and protects its own data.

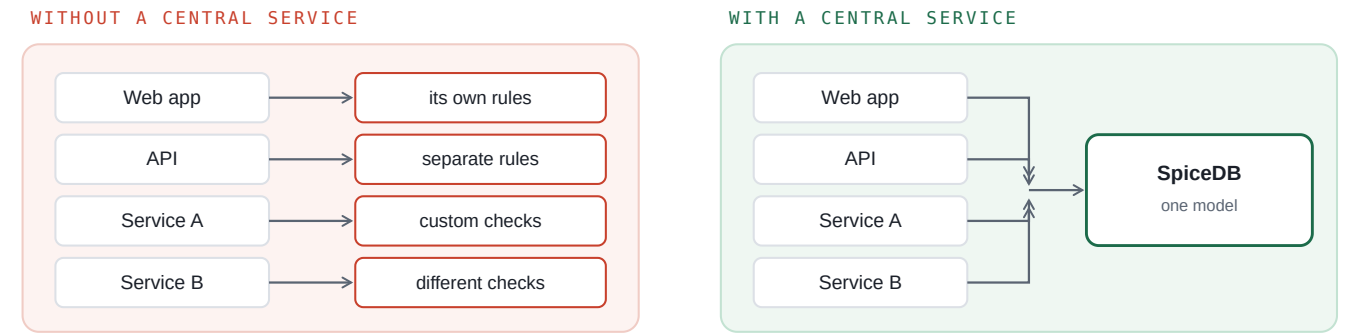


Figure 1 – Centralizing means one place to *evaluate* a permission. Each application still enforces the decision and protects its own data. This is not one service holding all your business logic.

WHAT THIS IS NOT

It is not putting your business logic in one service. Applications, APIs and data systems stay modular. What moves is the responsibility for **evaluating** a permission — and nothing else.

THE TEST WORTH APPLYING

If the same permission rule is implemented in more than one place today, the two have already drifted or they will. That drift is the cost centralizing removes — not the code you write once.

3 · WHY RELATIONSHIPS

Roles stop where the object starts

RBAC works well when access maps cleanly onto roles. It struggles the moment access depends on **which object** is being asked about: a folder shared with one partner, a project inherited from an organization, a document an agent may retrieve on behalf of one user.

Relationship-based access control models those connections and derives permissions from them. **RBAC, ABAC and ReBAC complement each other** — the right model depends on the case, and a purpose-built service gives you one place to express whichever you need.

WHAT CENTRALIZING CAN ENABLE	WHY
Faster delivery	Reuse one model instead of duplicating logic per service
Less duplicated effort	One foundation rather than several overlapping ones
More consistent behaviour	Define and test permissions centrally
New capabilities	Granular sharing, delegated admin, permission-aware retrieval

4 · THE COST MODEL

Run it with your numbers, not ours

Do not assume buying is cheaper. The figures below are **illustrative placeholders** chosen to show the shape of the calculation — they are not a benchmark, not a typical company, and not a claim about yours. Replace every one.

ILLUSTRATIVE ASSUMPTION	EXAMPLE
Fully loaded engineering cost	\$200,000 / engineer-year
Initial internal build	3 engineers × 6 months = \$300,000
Ongoing maintenance	1 engineer × 25% × 3 years = \$150,000
Infrastructure and operations	\$30,000 / year × 3 = \$90,000
Illustrative three-year total	\$540,000

WHAT THIS FIGURE EXCLUDES

The cost of product work delayed while the engine is built. Any growth in scope over three years — and scope always grows. For a buy comparison, add integration and migration effort plus the actual vendor price, and keep whatever infrastructure and operational cost remains. Compare like for like.

The useful output is **not** a savings percentage. It is a defensible answer to one question: **does owning and operating the authorization engine create enough strategic value to justify its full three-year cost?**

For some companies it does. If authorization *is* your product, or a genuine differentiator, building it is the right call and this paper is not for you.

Centralizing does not eliminate authorization work. It changes **which** work your engineers own — moving it from the engine to the model and the integration, where the product knowledge actually lives.

5 · SPICEDB, AND THE DIVISION OF LABOUR

You keep the model. You do not rebuild the engine.

SpiceDB is an open-source, Google Zanzibar–inspired database for fine-grained permissions. You write a schema describing your resource types, relationships and permissions; write relationship data as things happen in your product; and ask it one question per request. What matters most for this decision is what each side is responsible for.

STILL YOURS, ENTIRELY

Deciding what access means. No tool answers that for you.

Keeping relationship data accurate as your product writes it.

Integrating checks into every path that needs one.

Testing that the model does what you believe.

Enforcing the decision and protecting your own data.

WHAT YOU STOP BUILDING

The evaluation engine and its traversal.

The consistency model for revocations and caches.

The schema language and its tooling.

The storage layer behind permission data.

The scale work that arrives at your next enterprise deal.

SpiceDB is open source, so operating it yourself is a genuine option and nothing here has to be taken on trust — the schema language, the consistency model and the APIs are all readable in the project. Teams that would rather not run a database on the request path can evaluate **AuthZed Cloud**, a hosted service, or **AuthZed Dedicated**, a private isolated deployment operated by AuthZed. Confirm current product details and commitments with AuthZed directly.

6 · A PRACTICAL NEXT STEP

Map one workflow before you decide anything

Pick the single most complex permission workflow in your product — usually sharing, or external collaboration. Then answer four questions about it. The answers are typically more persuasive than any vendor material, including this document.

ASK ABOUT THAT ONE WORKFLOW	WHAT THE ANSWER TELLS YOU
Where are its rules implemented today?	Count the places. If it is more than one, they have already drifted or they will.
How many services enforce it?	Each one is a place a future change has to land correctly.
What happens when someone leaves a team?	If the answer involves a script or a nightly job, you have found the cost.
Could you prove who had access last quarter?	If not, that is a gap no amount of engine work closes on its own.

Where to look

- SpiceDB: authzed.com/spicedb
- Documentation: authzed.com/docs
- Zanzibar concepts: authzed.com/docs/spicedb/concepts/zanzibar
- Source: github.com/authzed/spicedb

SAID PLAINLY

This paper is published by AuthZed, who maintain SpiceDB, so read it with that in mind. The financial figures are illustrative placeholders rather than benchmarks. The Stripe, Slack and S3 comparisons are analogies for a build-versus-buy conversation, not claims that those products are equivalent or that buying is always right. No customer results, performance figures or savings percentages are claimed anywhere in this document, because none would be defensible without your numbers.