

The Case for Centralized Authorization

Three services. One rule. Three answers.

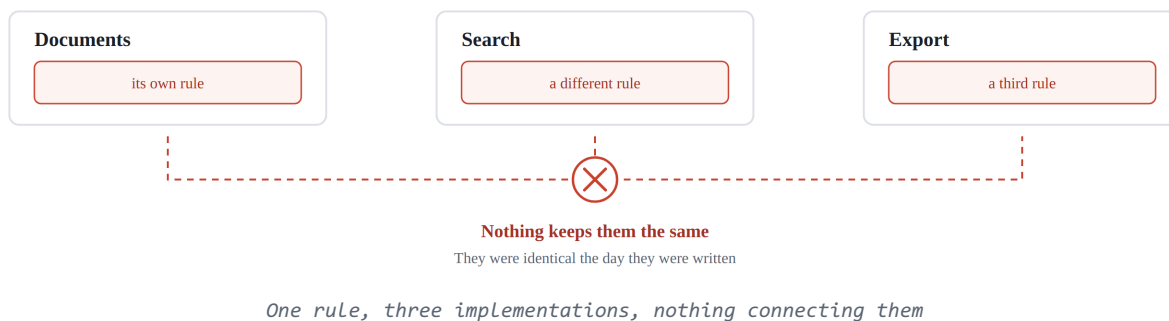
ONE • THE PROBLEM

How it happens

Authorization is the last thing most teams centralize, and the first thing that hurts when they don't — not because anyone builds it badly, but because each service builds it separately.

One service needs a permission check, so it gets one. It is correct. A year later a second service needs the same check and writes its own, because importing the first would mean a dependency nobody wants. A third team arrives and writes a third.

No step in that sequence is a mistake. Every one is the reasonable local decision. But the company now has three definitions of one rule, and the only thing holding them together is that the people who wrote them remember roughly what the others do.

**THE MOMENT IT SURFACES**

Usually not a breach. Usually a support ticket — **a manager who cannot see a report they should, or a contractor who still can.** And then the harder question arrives: *who else is affected?* Which nobody can answer, because the answer was computed three different ways and thrown away each time.

— What this document is

Five pages on what changes when the decision moves to one place, what that costs, and when not to do it. It is a summary of the argument rather than a proposal — the numbers that matter are yours, and we would rather look at them with you.

What it actually costs you

— The implementations drift

Identical the day they were written. Then one team fixed a bug, another widened a rule after a complaint, a third never added the check that arrived later. The three now disagree.

The direction matters. Over-granting shows up in an audit, if at all. Under-granting shows up as a support ticket — **so you find the harmless failures and not the expensive ones.**

— Every change costs three changes

A one-sentence requirement becomes three code changes, in three languages, across three test suites and three release schedules. Plus three fresh chances to implement it differently.

— Some questions become unanswerable

“What can this person see, across everything?” has **no code path** when the logic lives inside each application. Most teams write a report instead, and quietly accept it may be wrong.

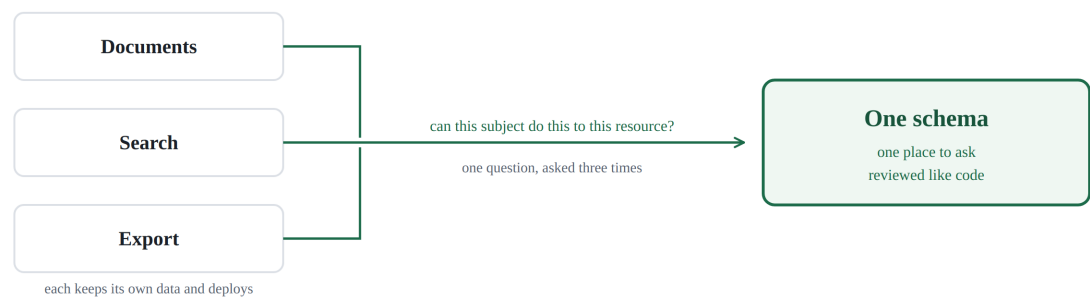
— It blocks what you want to build next

Permission-filtered search, list endpoints without a check per row, AI retrieval that reads only what its user may see. All three need the **permission set**, from one place.

What centralizing actually moves

The decision moves. Nothing else does.

Each application keeps its own database, its own interface and its own deploys. Each still enforces the answer it gets back. What they stop keeping is a private copy of the rule.



The decision moves. The applications do not.

THE DISTINCTION WORTH MAKING ON THE CALL

This is not a shared database, and not a service that holds your business logic. One question moves: **can this subject do this to this resource**. Everything else stays exactly where it is.

— The rule becomes reviewable

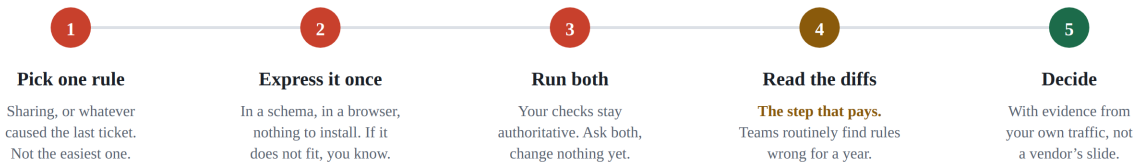
Written once, in a schema someone who did not write it can read in a pull request. A new requirement is one line changed — and **every application gets it without any of them deploying**.

— And the hard questions get answers

QUESTION	HOW IT IS ANSWERED
What can this person reach?	One call, returning the set — not a check per row
Who can reach this object?	One call, which is what an access review actually asks for
Who could reach it last quarter?	Relationship history, rather than reconstructing it from code
Did that revocation take effect?	A consistency token, so the check cannot be answered from a stale replica

Don't migrate everything

The projects that work start with one rule, not a platform decision.



Nothing is cut over until step five — and steps one to four are reversible.

— The test for whether this applies to you

How many places implement your most complex permission rule today, and how many of the people who approve changes to it understand all of them?

If both are one, you do not have this problem. If the first is three and the second is one, you already have it — you may simply not have found the disagreement yet.

WHAT WE WOULD DO NEXT

Bring your hardest permission rule to a working session and we will model it with you, live, from your real resources and roles. No slides. If the model does not fit your rule, that is a useful answer too, and it costs an hour.

SpiceDB is an open-source authorization database inspired by Google's Zanzibar paper. github.com/authzed/spicedb · authzed.com