



\$ whoami

- cofounder authzed, creators of SpiceDB
 - Previously Red Hat, CoreOS
 - OCI Maintainer, co-creator Operator Framework, etc...
- contact
 - email jimmy@authzed.com
 - github [@jzelinskie](https://github.com/jzelinskie)
 - x [@jimmyzelinskie](https://twitter.com/jimmyzelinskie)
 - bsky [@jimmy.zelinskie.com](https://bsky.app/profile/jimmy.zelinskie.com)
 - discord.gg/spicedb [@jzelinskie](https://discord.gg/spicedb)



— Jargon alert ⚠

AuthN vs AuthZ

~~AuthN vs AuthZ~~
identity vs access management

~~AuthN vs AuthZ~~
identity vs permissions

Broken Access Control
is **#1** on **OWASP's Top 10**
most critical security risks
to web apps

how did we get here?

Broken Access Control
is **#1** on OWASP's Top 10
most critical security risks
to web apps

... i'm not pointing any fingers, but ...
let's dive into the history of two groups:

- **ACADEMICS**
- **INDUSTRY PRACTITIONERS**

... i'm not pointing any fingers, but ...
let's dive into the history of two groups:

- **ACADEMICS**
- **INDUSTRY PRACTITIONERS**



ACADEMIA

- 1983 - DAC/MAC
- 1992 - RBAC
- 2015 - ABAC
- 2019 - ReBAC

- Discretionary Access Control
 - e.g. file systems/google docs
- Mandatory Access Control
 - e.g. SELinux
- As old as war itself
- TCSEC documents DAC/MAC

ACADEMIA

- 1983 - DAC/MAC
- 1992 - RBAC
- 2015 - ABAC
- 2019 - ReBAC

- Role-based access control
 - map users to groups that delegate access
- e.g. every enterprise app you've ever used
- It's never the same



ACADEMIA

- 1983 - DAC/MAC
- 1992 - RBAC
- 2015 - ABAC
- 2019 - ReBAC

- Attribute-based access control
- usually extends RBAC
 - "roles" become an attribute
- adds real-time context



ACADEMIA

- 1965 - Multics File system
- 1983 - DAC/MAC
- 1992 - RBAC
- 2015 - ABAC
- 2019 - ReBAC

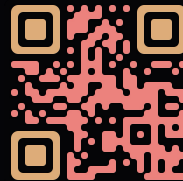
- Hierarchical file system tree
- Every branch had 5 attributes
 - read
 - write
 - exec
 - append
 - trap



ACADEMIA

- 1965 - Multics File system
- 1983 - DAC/MAC
- 1992 - RBAC
- 2015 - ABAC
- 2019 - ReBAC

- Relationship-based access control
- 2007 - Carrie Gates coins the term
- 2019 - Google's Zanzibar
- 2021 - SpiceDB is OSS



... i'm not pointing any fingers, but ...
let's dive into the history of two groups:

- ACADEMICS
- INDUSTRY PRACTITIONERS

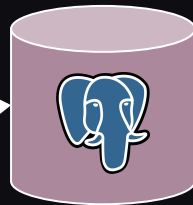
INDUSTRY

code embedded in applications



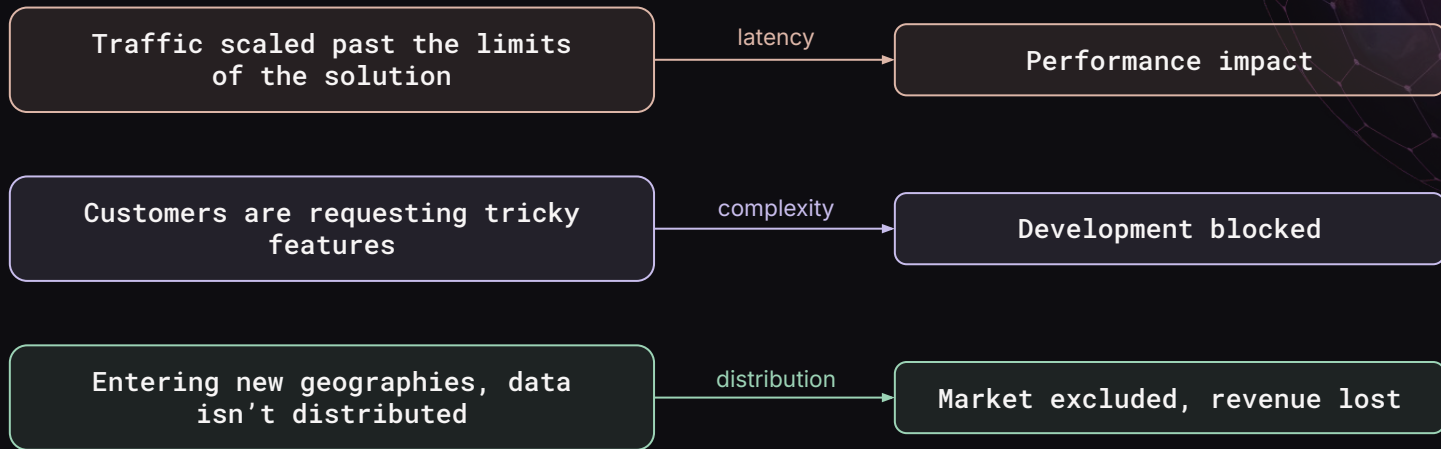
Monolithic Application

```
func authorized(u auth.U, allowed []auth.U) bool {  
    for _, allowedUser := range allowed {  
        if u == allowedUser {  
            return true  
        }  
    }  
    return false  
}  
  
func (s *Server) HandleObjectRequest(user auth.U, objID int) {  
    allowed := db.GetAuthorizedUsersForObject(objID)  
  
    if !authorized(user, allowed) {  
        s.Raise403()  
    }  
  
    s.SendResponse(db.LoadObject(objID))  
}
```



INDUSTRY

falls over at some point



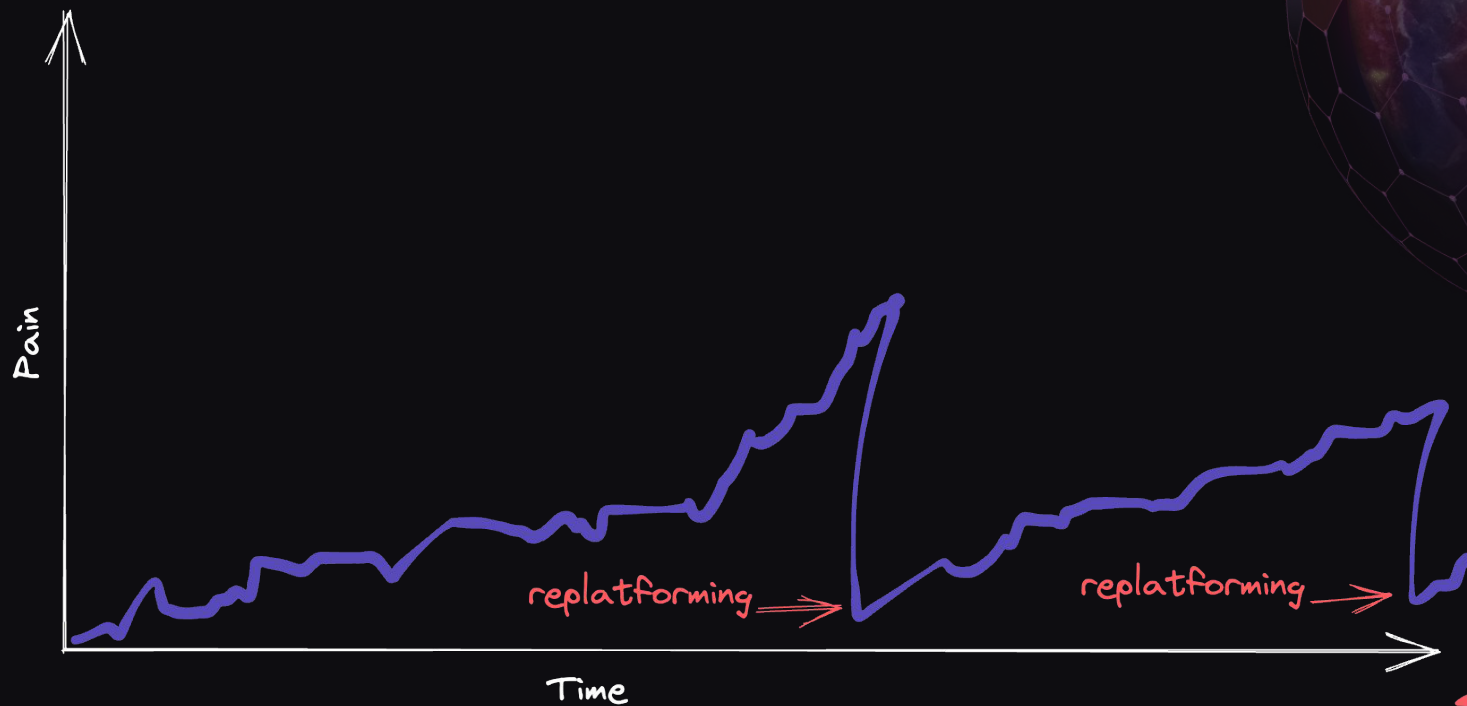
INDUSTRY

a redesign/fix is intricate

```
1 engineer * (1 month implementation +  
              1 month QA +  
              1 month deployment)  
= 1 new permission
```

INDUSTRY

rinse & repeat



how do we fix it?

Broken Access Control
is **#1** on OWASP's Top 10
most critical security risks
to web apps

how do we fix it?

"Although **RBAC** has a long history and remains popular among software developers today,

ABAC and **ReBAC**
should typically be preferred for application
development"

— OWASP Authorization Cheat Sheet

why?

- fine-grained, complex Boolean logic
- speed
- robustness
- multi-tenancy
- management ease

— OWASP Authorization Cheat Sheet



Google Zanzibar 2019

Zanzibar: Google's Consistent, Global Authorization System

Ruoming Pang,¹ Ramón Cáceres,¹ Mike Burrows,¹ Zhifeng Chen,¹ Pratik Dave,¹
Nathan Germer,¹ Alexander Golynski,¹ Kevin Graney,¹ Nina Kang,¹ Lea Kissner,^{2*}
Jeffrey L. Korn,¹ Abhishek Parmar,^{3*} Christina D. Richards,¹ Mengzhi Wang¹
Google, LLC;¹ Humu, Inc.;² Carbon, Inc.³
{rpang, caceres}@google.com

Abstract

Determining whether online users are authorized to access digital objects is central to preserving privacy. This paper presents the design, implementation, and deployment of Zanzibar, a global system for storing and evaluating access control lists. Zanzibar provides a uniform data model and configuration language for expressing a wide range of access control policies from hundreds of client services at Google, including Calendar, Cloud, Drive, Maps, Photos, and YouTube. Its authorization decisions respect causal ordering of user actions and thus provide external consistency amid changes to access control lists and object contents. Zanzibar scales to trillions of access control lists and millions of authorization requests per second to support services used by billions of people. It has maintained 95th-percentile latency of less than 10 milliseconds and availability of greater than 99.999% over 3 years of production use.

1 Introduction

Many online interactions require authorization checks to confirm that a user has permission to carry out an operation on a digital object. For example, web-based photo storage services typically allow photo owners to share some photos with friends while keeping other photos private. Such a service must check whether a photo has been shared with a user before allowing that user to view the photo. Robust authorization checks are central to preserving online privacy.

This paper presents Zanzibar, a system for storing permissions and performing authorization checks based on the stored permissions. It is used by a wide array of services offered by Google, including Calendar, Cloud, Drive, Maps, Photos, and YouTube. Several of these services manage billions of objects on behalf of more than a billion users.

A unified authorization system offers important advantages over maintaining separate access control mechanisms

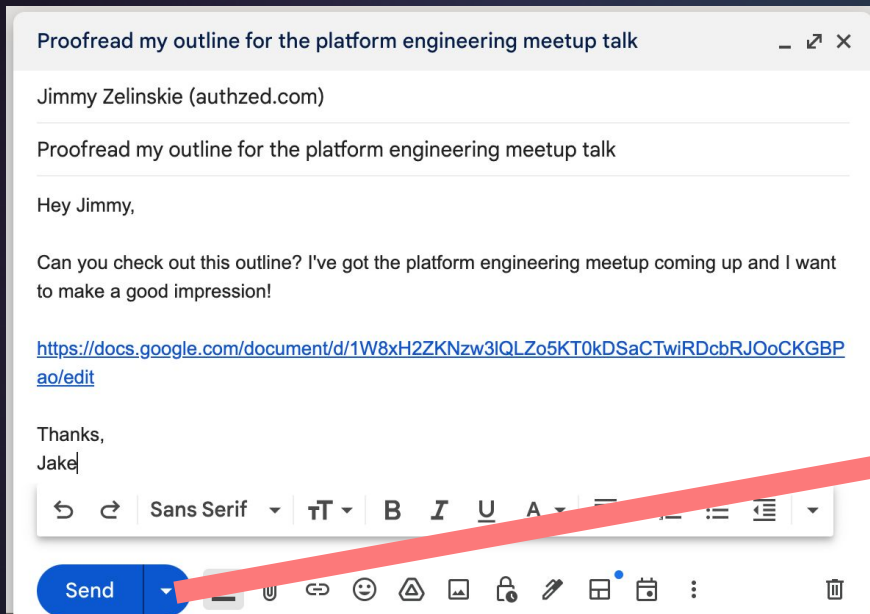
semantics and user experience across applications. Second, it makes it easier for applications to interoperate, for example, to coordinate access control when an object from one application embeds an object from another application. Third, useful common infrastructure can be built on top of a unified access control system, in particular, a search index that respects access control and works across applications. Finally, as we show below, authorization poses unique challenges involving data consistency and scalability. It saves engineering resources to tackle them once across applications.

We have the following goals for the Zanzibar system:

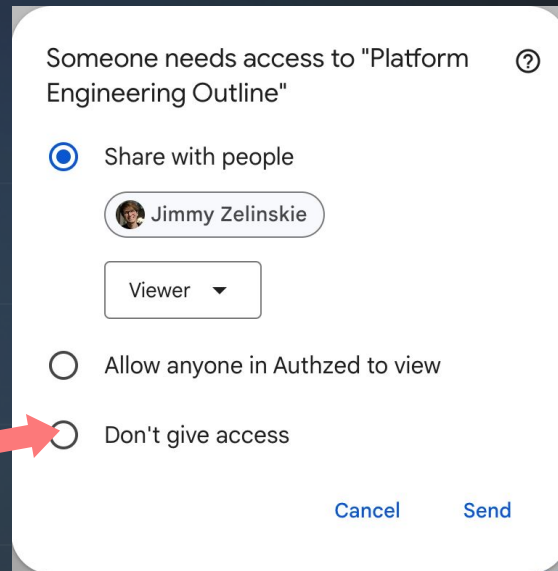
- *Correctness*: It must ensure consistency of access control decisions to respect user intentions.
- *Flexibility*: It must support a rich set of access control policies as required by both consumer and enterprise applications.
- *Low latency*: It must respond quickly because authorization checks are often in the critical path of user interactions. Low latency at the tail is particularly important for serving search results, which often require tens to hundreds of checks.
- *High availability*: It must reliably respond to requests because, in the absence of explicit authorizations, client services would be forced to deny their users access.
- *Large scale*: It needs to protect billions of objects shared by billions of users. It must be deployed around the globe to be near its clients and their end users.

Zanzibar achieves these goals through a combination of notable features. To provide flexibility, Zanzibar pairs a simple data model with a powerful configuration language. The language allows clients to define arbitrary relations between users and objects, such as `owner`, `editor`, `commenter`, and `viewer`. It includes set-algebraic operators such as intersection and union for specifying potentially complex access control policies in terms of these user-object relations. For

wait...what is this magic?



Gmail



Google Docs

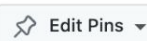
how do **i** zanzibar?

— you





Public



main



19 Branches



69 Tags



About



README



Code of conduct



Apache-2.0 license



SpiceDB

openssf best practices **passing**  container v1.29.0 docs authzed.com  discord 195 online  twitter @authzed

SpiceDB is an open source, [Google Zanzibar](#)-inspired database for creating and managing security-critical application permissions.

Developers create a [schema](#) and use [client libraries](#) to apply the schema to the database, insert [relationships](#) into the database, and query the database to efficiently check permissions in their applications.

Features that distinguish SpiceDB from other systems include:

- Expressive [gRPC](#) and [HTTP/JSON](#) APIs for checking permissions, listing access, and powering devtools
- A distributed, parallel graph-engine faithful to the architecture described in [Google's Zanzibar paper](#)
- A flexible consistency model configurable [per-request](#) that includes resistance to the [New Enemy Problem](#)
- An expressive [schema language](#) with a [playground](#) and CI/CD integrations for [validation](#) and [integration testing](#)
- A pluggable [storage system](#) supporting [in-memory](#), [Spanner](#), [CockroachDB](#), [PostgreSQL](#) and [MySQL](#)

Open Source, Google Zanzibar-inspired permissions database to enable fine-grained access control for customer applications

 [authzed.com/docs](#)

kubernetes security distributed-systems database scale latency production permissions acl grpc rbac cloud-native entitlements abac security-tools ciam fga fine-grained-access-control zanzibar



Readme



Apache-2.0 license



Code of conduct



Activity



Custom properties



4.2k stars

but WHAT IS SpiceDB

- highly parallel graph database optimized for authorization queries
- gRPC & HTTP API service written in Go
- additional servers to power devtools, testing services

```
definition user {}

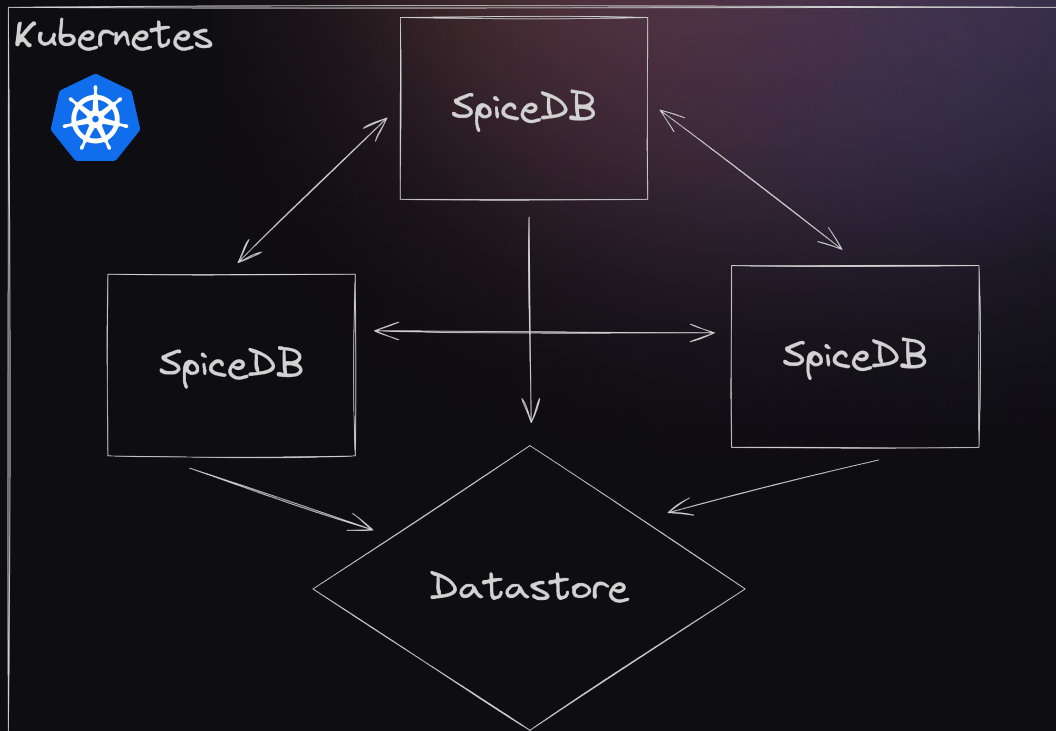
definition document {
  relation writer: user
  relation reader: user

  /**
   * edit determines whether a user can edit the document
   */
  permission edit = writer

  /**
   * view determines whether a user can view the document
   */
  permission view = reader + writer
}
```



\$ kubectl create SpiceDBCluster



\$ kubectl scale



\$ whatis zed

- CLI tool for SpiceDB
- manage credentials, backup/restore, import, validation
- commands for SpiceDB APIs + debugging

```
$ zed permission check --explain document:firstdoc view user:fred
```

```
true
✓ document:firstdoc view (66.333µs)
├─ ✗ document:firstdoc writer (12.375µs)
└─ ✓ document:firstdoc reader (20.667µs)
    └─ user:fred
```



<https://play.authzed.com>

- web IDE powered by WebAssembly

Resource			Subject		
Type	ID	Relation	Type	ID	Subject Relation (opt)
1	document	firstdoc	writer	user	tom
2	document	firstdoc	reader	user	fred
3	document	seconddoc	reader	user	tom
+ Add relationship					

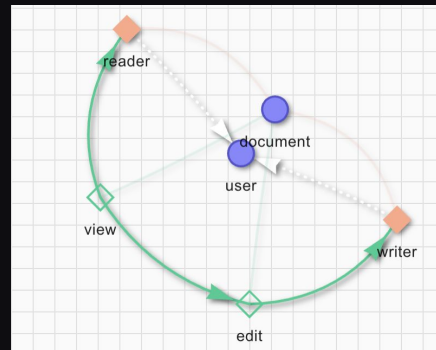
PROBLEMS

CHECK WATCHES

SYSTEM VISUALIZATION

LAST VALIDATION RUN

Resource	Permission	Subject
document:firstdoc	view	user:fred
document:firstdoc	view	
document:firstdoc	reader	
user:fred		



DEF SCHEMA TEST RELATIONSHIPS ASSERTIONS EXPECTED RELATIONS

Validated! RUN ACCEPT UPDATE REVERT UPDATE

```
1 document:firstdoc#view:
2 - "[user:tom] is <document:firstdoc#writer>"
3 - "[user:fred] is <document:firstdoc#reader>"
4 document:seconddoc#view:
5 - "[user:tom] is <document:seconddoc#reader>"
6
```

```
1 document:firstdoc#view:
2 - "[user:fred] is <document:firstdoc#reader>"
3+ - "[user:tom] is <document:firstdoc#writer>"
4 document:seconddoc#view:
5 - "[user:tom] is <document:seconddoc#reader>"
6
```



SpiceDB is Zanzibar +

SpiceDB

ABAC support with
SpiceDB "Caveats"

Ability to model more
complex user systems

Relations distinguished
from permissions

More granularly tunable
consistency

Improved devX: schema
language, playground

Reverse indexing: who
has access to what?

Zanzibar

Relationships as edges
in a graph (ReBAC)

Schema to flexibly interpret
those relationships

Scalable to >10M QPS at
99.999 availability

Built to support distributed
data stores

Solves the new enemy problem
with tokens "Zookies"

how do i spicedb?

— you

discord.gg/spicedb

thankz!

discord.gg/spicedb