# “IAM very confused - A Friendly Guide to Cloud and Modern AuthZ”

- Bsides Basingstoke (2025-07-25) ~ Presented by Tom Cope

(warning: this presentation and content was entirely human generated)

# Hello World!

- Tom Cope (https://tomcope.com)
- Principal Consultant at Control Plane
  - Ex-CSO of Ava Security (now part of Motorola)
  - Security Engineer / SOC Analysis for UK leading asset manager
  - IAM SME at IBM
- Working in Cyber Security ~12 years
- Part time Security Researcher (CVE-2020-5014 + CVE-2021-29707)
  - CodeGate AI Write up coming soon!
- Blue team by day, red team by night
- Catch me at DEFCON later this year (Speaking @Malware Village)

| | | | | | |
|---|---|---|---|---|---|
| AWS Certified Solutions Architect – Associate<br>Amazon Web Services Training and Certification | CompTIA Security+ ce Certification<br>CompTIA | Certified Information Systems Security Professional (CISSP)<br>(ISC)² | Security and Privacy by Design Foundations<br>IBM | Think Like a Hacker<br>IBM | Government Insights and Solutions (Silver)<br>IBM |
| IBM Mentor<br>IBM | Docker Essentials: A Developer Introduction<br>IBM | IBM Recognized Teacher/ Educator<br>IBM | IBM Security Essentials for Architects<br>IBM | Get started with Kubernetes and IBM Cloud Container Service<br>IBM | Cloud Security Architect and Engineer Fundamentals<br>IBM |

🔥🔥 FREE DOWNLOAD: 🔥🔥
https://control-plane.io/hackingkubernetes

## ControlPlane - Who we are?

Cloud Native and Open Source security consultancy established in 2017. Our diverse culture empowers and develops individuals with talent and integrity.

Accustomed to work in highly-regulated environments and strategic programmes

### Industry Leading

Kubernetes (first threat model for Financial Services User Group (FSUG)

**Hacking Kubernetes O'Reilly, authors and Trainer**

Defending Cloud Native - SANS, SEC 584 authors and trainers

### Future Focussed

Linux Foundation's Security Technical Advisory Group (co-chair, CNCF)

OpenUK (CISO, pro bono)

Community organisers and collaborators (i.e. KubeCon, CloudNativeCon)

Our Cloud Native Security Consultancy Services

## *We help you Secure Kubernetes from supply chain and runtime attacks*

Threat Model Subscription

Assurance & Penetration Testing

DevSecOps Delivery

Security Roadmap

Threat Model

Discovery Workshop

**Security Audit and Architecture**

Deep security analysis of Cloud Native environments from Dev. to Prod.

**Cloud Native & Open Source Integration**

Develop & integrate for explosive platform growth & security resilience

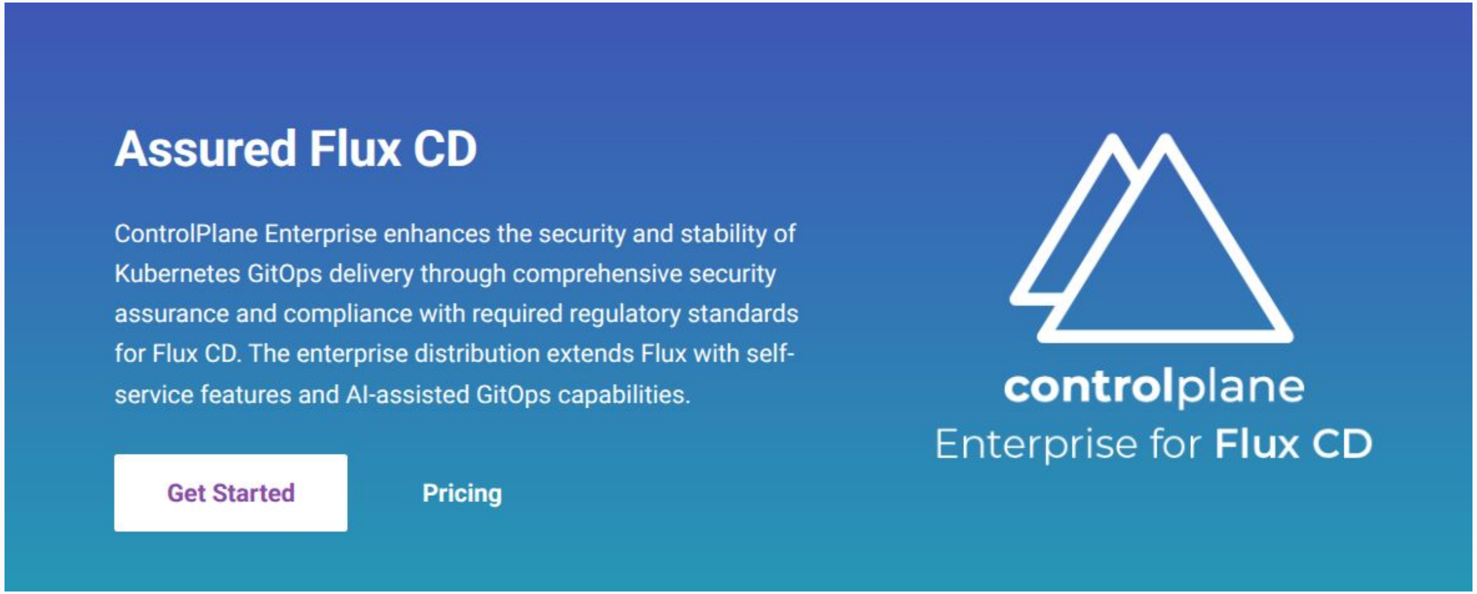
**Cloud Native Engineering**

Expert DevSecOps engineering for highly regulated environments

Training

Agile

**control**plane



## Assured Flux CD

ControlPlane Enterprise enhances the security and stability of Kubernetes GitOps delivery through comprehensive security assurance and compliance with required regulatory standards for Flux CD. The enterprise distribution extends Flux with self-service features and AI-assisted GitOps capabilities.

Come and see our booth!

https://github.com/fluxcd/flux2

https://control-plane.io/

# IAM - Identity Access Management

"authori<span>z</span>ation" = American English
"authorisation" = British English

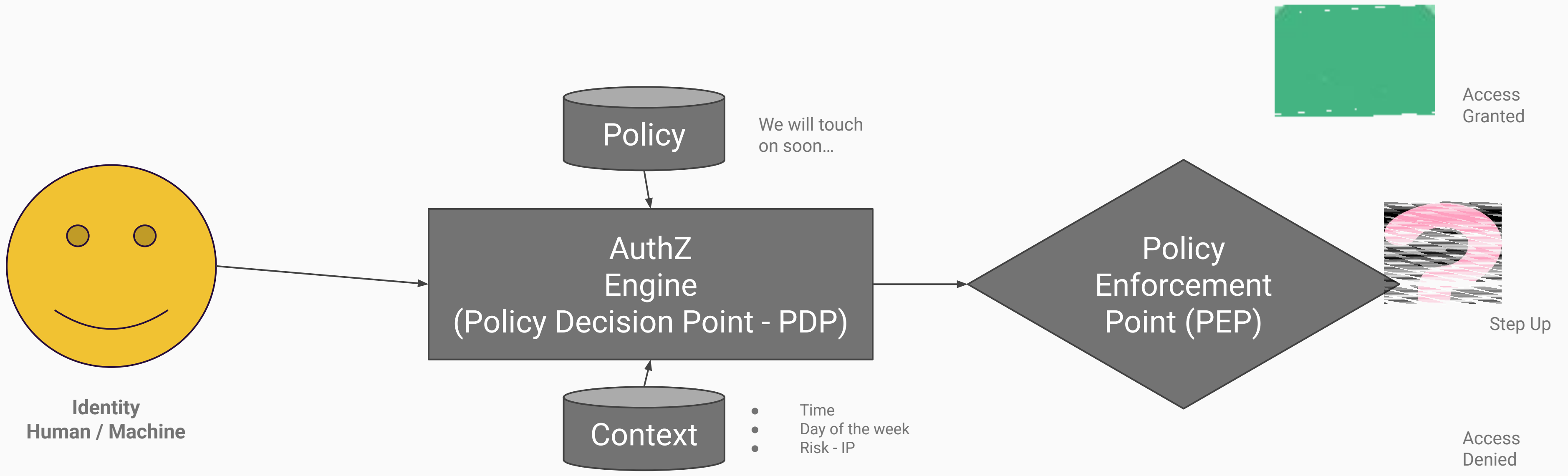
The three A's:

- A - Authentication (AuthN)
- A - Authorization (AuthZ)
- A - Audit

  - Who are you?
  - Are you allowed todo that?
  - Make a note of it

# Authorization - AuthZ

“Authorization is the process of determining what resources/actions a user or system is allowed to access/perform”



# Where do we use Authorization?

Business → **BOTH** ← Product

You are building your own custom Authorization application to solve a business need

Business

- Who can access your cloud?
- Google Workspace permissions
- Customer Database permissions
- Internal Apps

Product

- Building product X and you need a permission system to enforcee Y
- Roll your own! Or use a open source product / paid service

## The Great Compromise

- Easy of Use
- Maintainability
- Expressiveness of your policy language
- Compatibility
- Cost (both for services, transition and ongoing maintenance)

**"Perfect is the enemy of good"**

- Voltaire

## Two broad types of access control

Coarse-Grained (CGA)

“I can delete all blog posts”

Fine-Grained Access Control (FGA)

“I can delete only X blog post”

## Lets talks types of access control!

- Attribute-Based Access Control (ABAC)
- Policy-Based Access Control (PBAC)
- Role Based Access Control (RBAC)
- Relationship-Based Access Control (ReBAC)

And then talk about implementation!

# Attribute-Based Access Control (ABAC)

- **Authorization decisions are based on the presence or absence of tags**
- Subject / Resource (Humans, EC2 Instances, Applications etc) are associated with 0-n “Attributes” or tags:
  - project-y
  - infra-team
  - secret / restricted

Static or Dynamic:
- Auto Scaling groups
- Documents created with in a folder
- Terraform resources
- Users within a department

## Attribute-Based Access Control (ABAC)

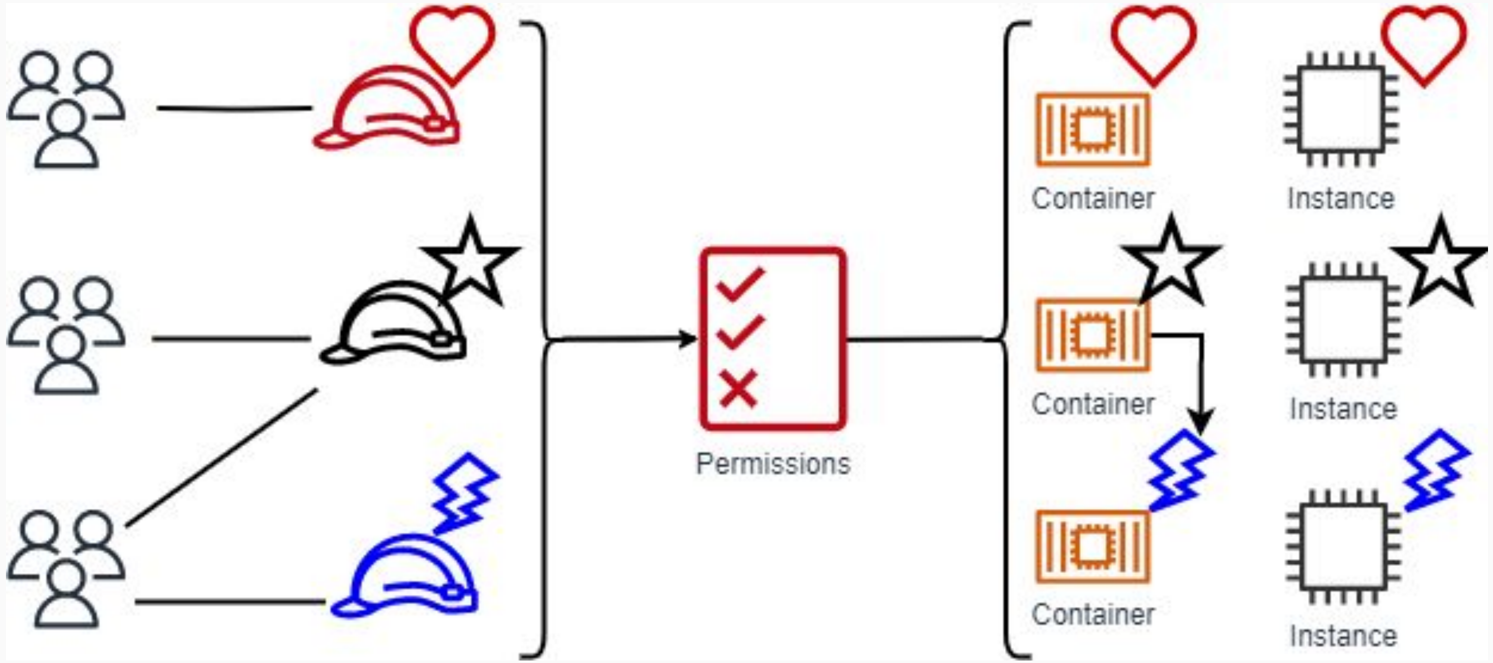
- Attributes can be environmental such as:
  - in-work-hours
  - tuesday
  - Remote-connection
- Attributes can be meta as well:
  - 2fa-provided
  - new-device
  - Phishing-resistant-2fa-used
- Attributes based on action:
  - create / delete
  - Transfer > £100

Could be used to trigger setup authentication

Commonly seen in Azure Conditional Access

Used to generate a risk score

# Attribute-Based Access Control (ABAC)



Ref: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction_attribute-based-access-control.html

# Attribute-Based Access Control (ABAC)

Example policies:

Policy: 🟩

infra-team -> project-y

Policy: 🟩

- Finance team
- Remote Working
- 2FA-Provided
- Known Device

->

Restricted documents

# Attribute-Based Access Control (ABAC)

- **Advantages**
  - Auto tagging of resources
  - Scales well with the business
  - Fewer Policies*
  - **Linked with sources of truth (eg corporate directory or cloud environment)**

- **Disadvantages**
  - Not as fine grained (per object basis)
  - How many attributes?
  - Getting all the tags to stick, support for tagging

## Policy-Based Access Control (PBAC)

- “Policy-Based Access Control is an overarching authorization model where access decisions are determined by evaluating a set of defined policies”
  - These policies can be anything implemented in a variety of ways
    - Leads to a overloaded term

## Policy-Based Access Control (PBAC)

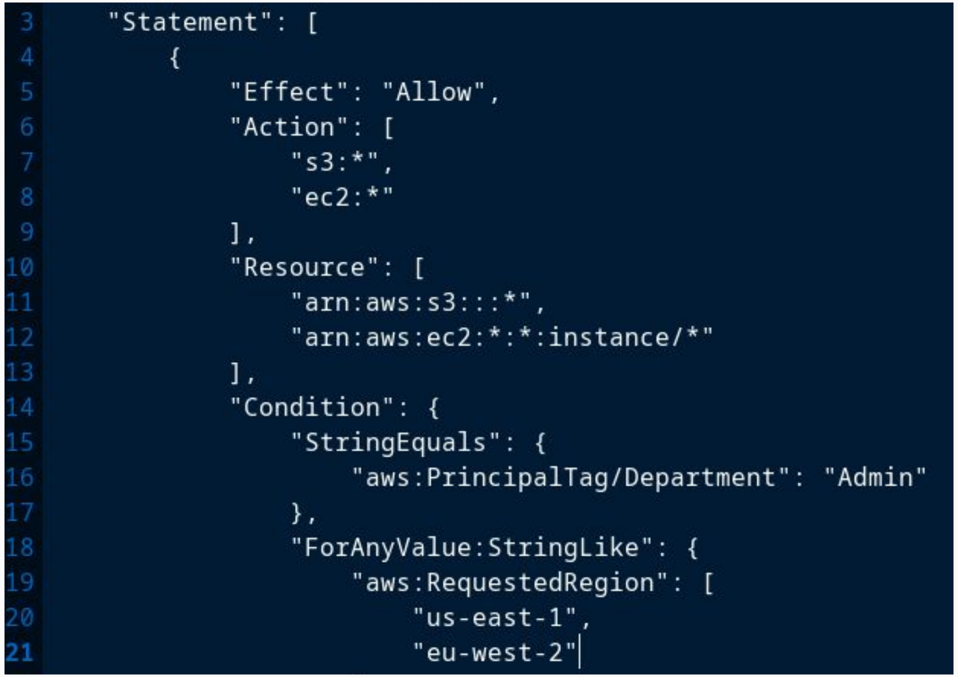
- As Code:

```python
def check_access(user_attributes: dict, http_path: str) -> bool:
    now = datetime.datetime.now()
    is_work_hour = 9 <= now.hour < 17 and now.weekday() < 5
    return user_attributes.get("admin", False) and "/admin" in http_path.lower() and is_work_hour
```

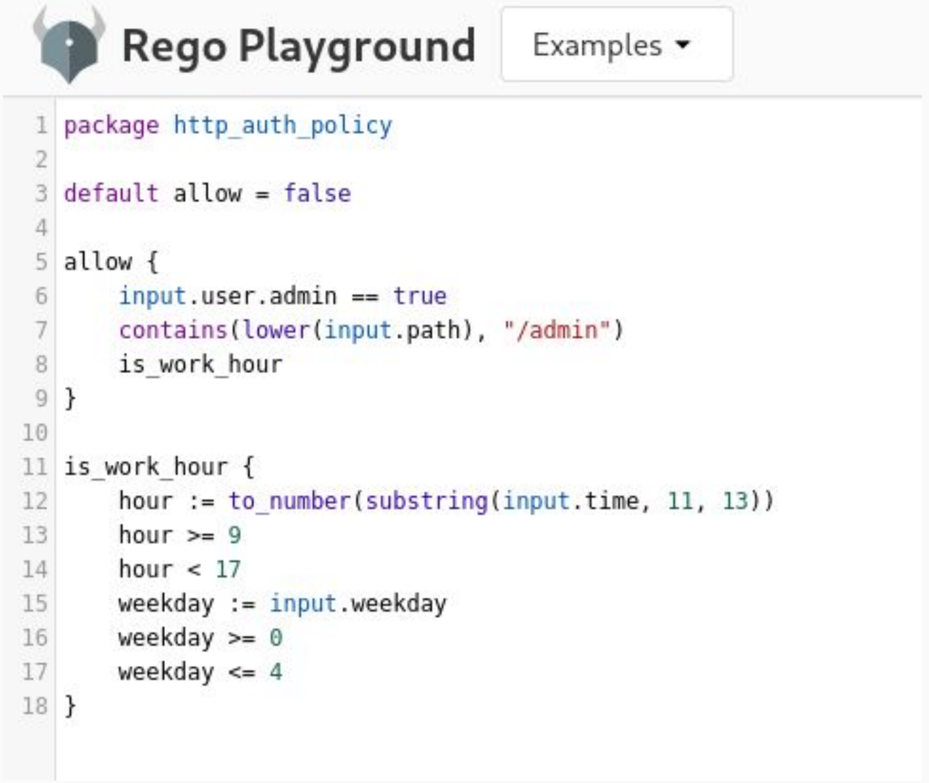
Only allow users with the “admin” attribute to access the “/admin” page during the working week

# Policy-Based Access Control (PBAC)

- In Code:



```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "ec2:*"
            ],
            "Resource": [
                "arn:aws:s3:::*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/Department": "Admin"
                },
                "ForAnyValue:StringLike": {
                    "aws:RequestedRegion": [
                        "us-east-1",
                        "eu-west-2"
```



```
package http_auth_policy

default allow = false

allow {
    input.user.admin == true
    contains(lower(input.path), "/admin")
    is_work_hour
}

is_work_hour {
    hour := to_number(substring(input.time, 11, 13))
    hour >= 9
    hour < 17
    weekday := input.weekday
    weekday >= 0
    weekday <= 4
}
```

*examples only: please don't use these policies in production

# Policy Based Access Control (PBAC)

- **Advantages**
  - Easy to review (with linters or human review)
  - Easy to audit (log inputs and policy decision)
  - Incredibly flexible
  - Can be very Fine-Grained

- **Disadvantages**
  - Can get complex quickly
  - Lots of policy re-writes as the business grows (new system = new policy)
  - Performance of evaluating the policy
  - Does each team manage their own policy or a central team?

# Role-Based Access Control (RBAC)

- Actor
  - User -> “tom”
  - Machine -> “database_ec2”
- Role
  - “admin”
- Permissions
  - “add_users”
  - “bake_cake”

- You’ve seen it everywhere:
- Every SAAS, IAAS
- AWS IAM Redefined Roles
- Google Workspace

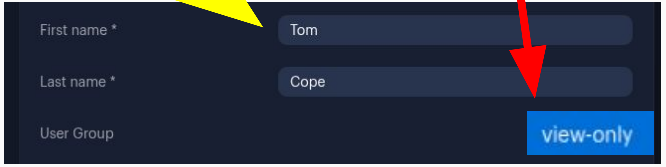
# Role-Based Access Control (RBAC)

CCTV System Example

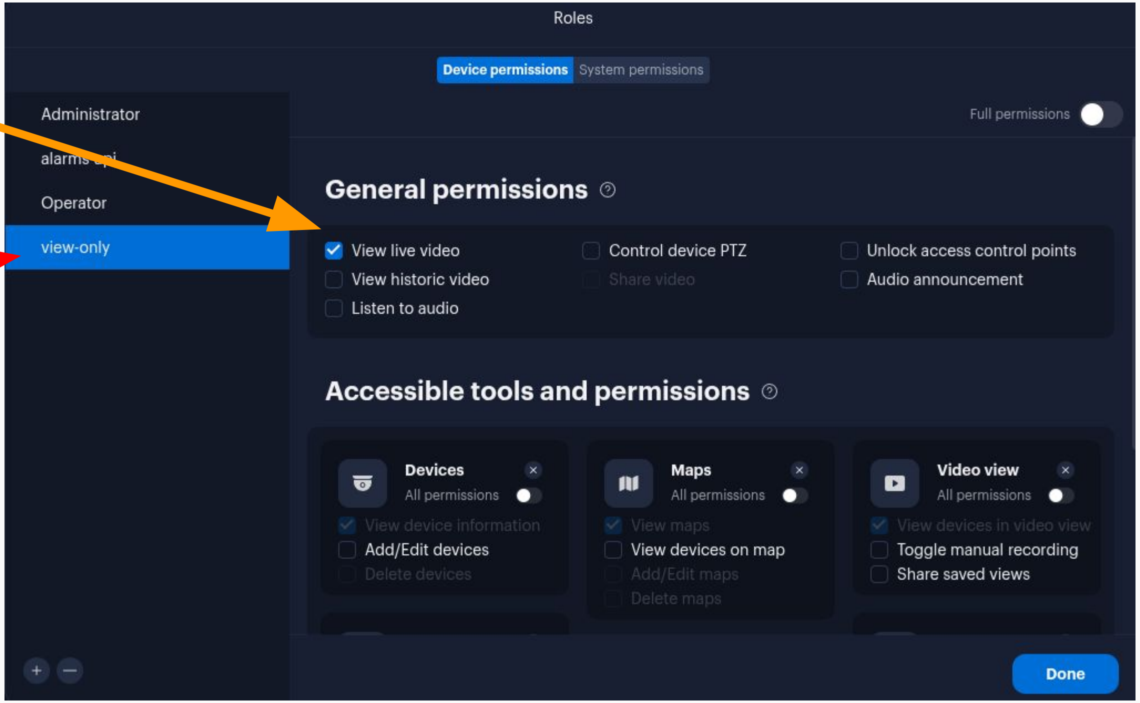
1. User

2. Role

3. Permissions



| | |
|---|---|
| First name * | Tom |
| Last name * | Cope |
| User Group | view-only |



Roles

Device permissions | System permissions

- Administrator
- alarms api
- Operator
- view-only

Full permissions

**General permissions**

- [x] View live video
- [ ] View historic video
- [ ] Listen to audio
- [ ] Control device PTZ
- [ ] Share video
- [ ] Unlock access control points
- [ ] Audio announcement

**Accessible tools and permissions**

Devices — All permissions
- [x] View device information
- [ ] Add/Edit devices
- [ ] Delete devices

Maps — All permissions
- [x] View maps
- [ ] View devices on map
- [ ] Add/Edit maps
- [ ] Delete maps

Video view — All permissions
- [x] View devices in video view
- [ ] Toggle manual recording
- [ ] Share saved views

## Role-Based Access Control (RBAC)

- **Advantages**
  - Easy to implement
  - Gets the job done for most use cases
  - Easy mental model -> “Finance” / “Infra”

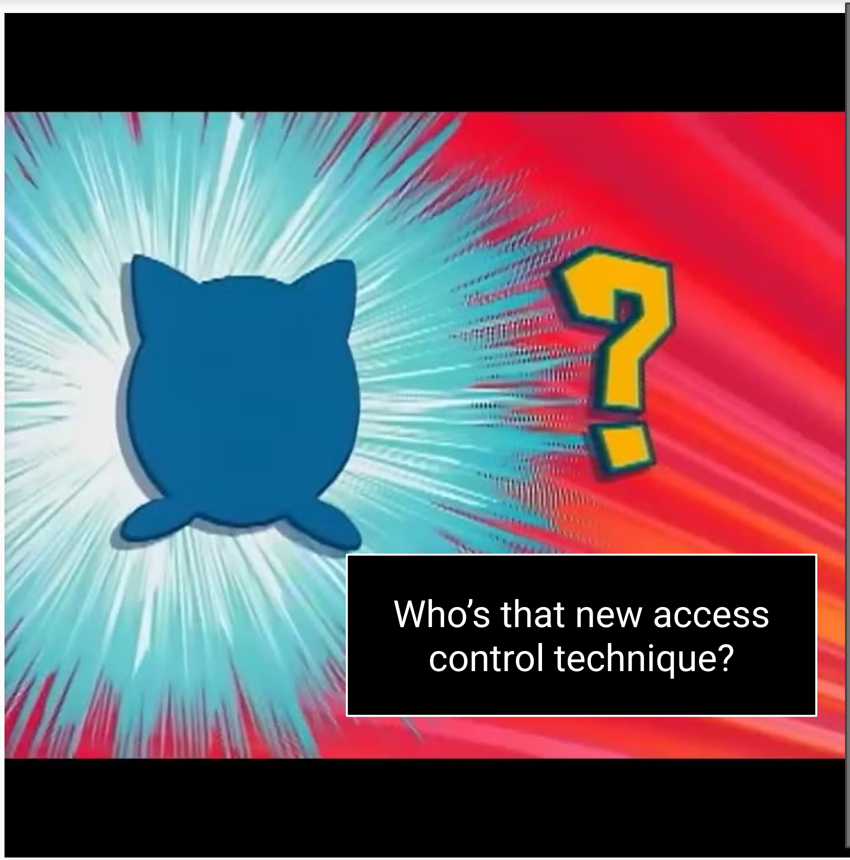
- **Disadvantages**
  - Does not scale well
  - Custom Roles = “Role Explosion”
  - Usually lacks context such as “time of day”
  - Lack fine grain permissions / not always in control of the permissions attached to a role (especially in SAAS)

## Semantics

- Rarely everything is so cut and dry

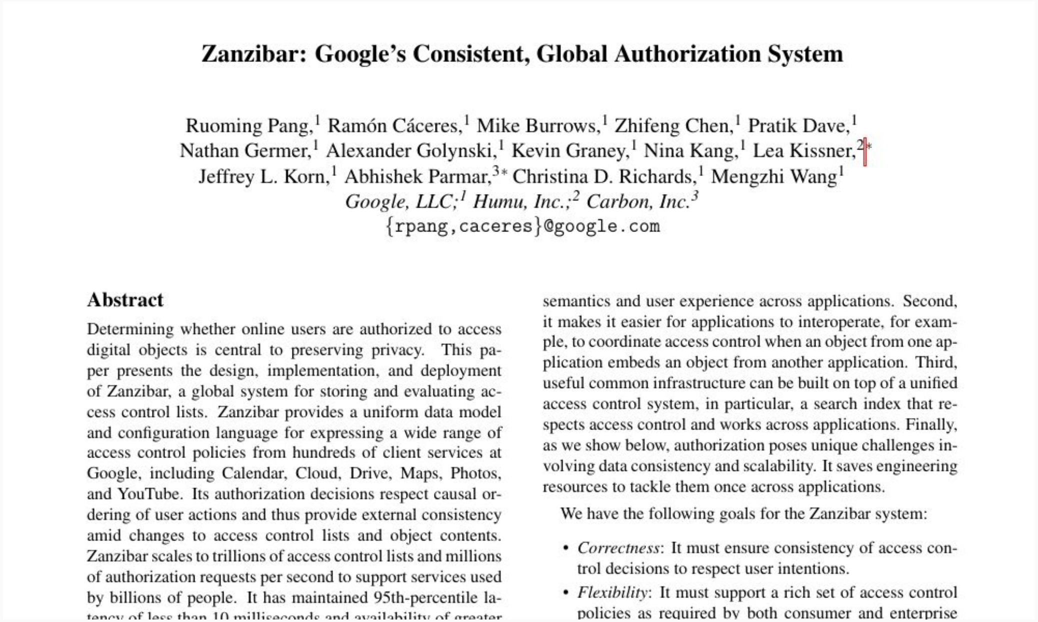
$$\begin{tikzcd}
\boxed{\text{Attribute-Based Access Control (ABAC)}} \arrow[d] & \text{Theory} \\
\boxed{\text{Policy-Based Access Control (PBAC)}} \arrow[d] & \text{You need a policy engine to implement ABAC} \\
\boxed{\text{Role-Based Access Control (RBAC)}} & \text{Style of Implementation}
\end{tikzcd}$$

## New Developments...



Who’s that new access control technique?

# Google Zanzibar and the introduction of Relationship-Based Access Control (ReBAC)



**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 Wang[1]
*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

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

Zanzibar: Google's Consistent, Global Authorization System

**2019** {USENIX} Annual Technical Conference ({USENIX} {ATC} '19)}

Ref: https://research.google/pubs/zanzibar-googles-consistent-global-authorization-system/

## Zanzibar

- Built by Google for Googles AuthZ Problems
- Designed to be extremely fast, globally consistent, highly scalable and fault tolerant to human and system error
- Built on top of Google Spanner Database
- Introduced the concept of:
  - Relationship-Based Access Control (ReBAC)

# Relationship-Based Access Control (ReBAC)

- **An authorization decision are based on direct or indirect relationship between entities.**

- Two separate components contained in a graph database like system:
  - Model
    - Defines “types” and “relations”
  - Tuples
    - Data within the database which provides the relationships

Ref: auth0 fga playground [https://play.fga.dev/sandbox/?store=slack](https://play.fga.dev/sandbox/?store=slack)

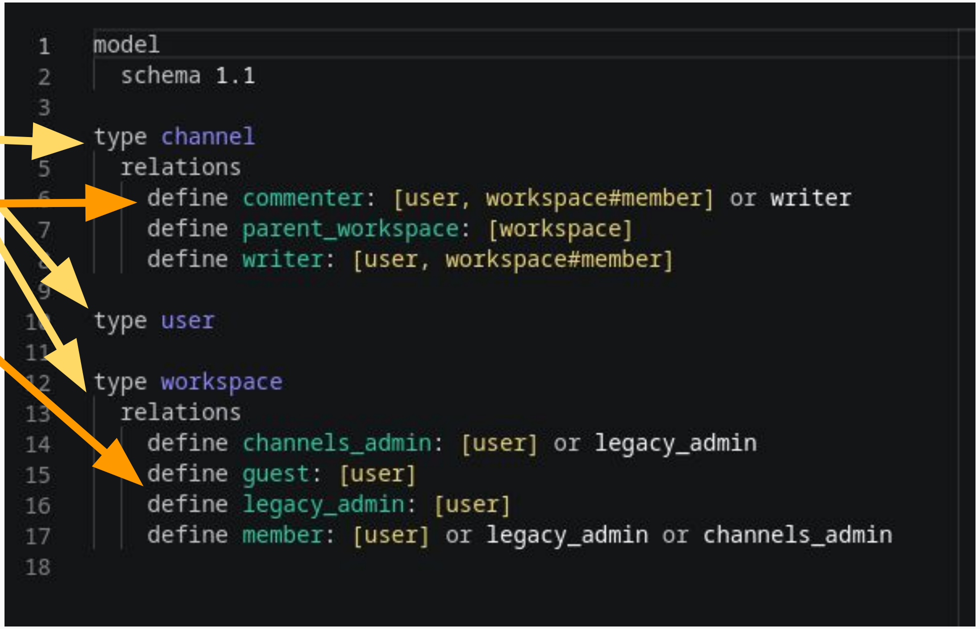
# Relationship-Based Access Control (ReBAC)

Model - Like a database scheme

- Different Types
- Relations - How the types can relate to each other

-----------------------------------------------

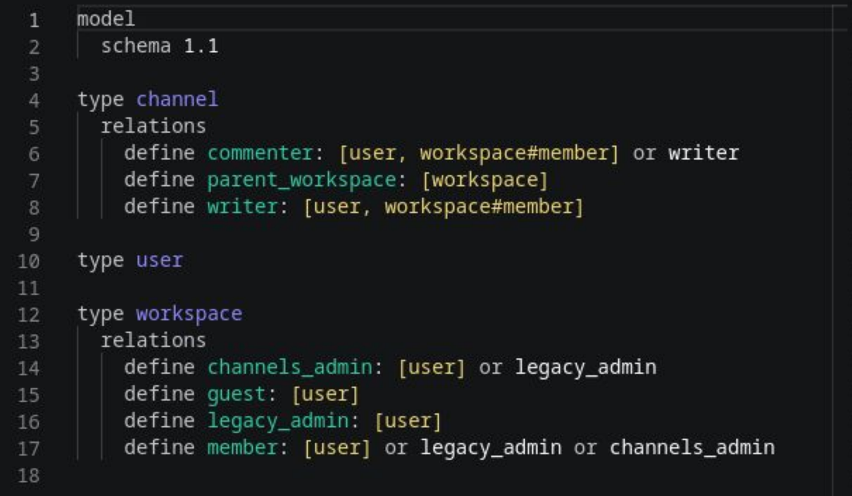
- Using slack permissions as a example
- “Channel” and “user” can be related by a “writer” relationship
- “Workspace” and “user” can be related by a “member” relationship



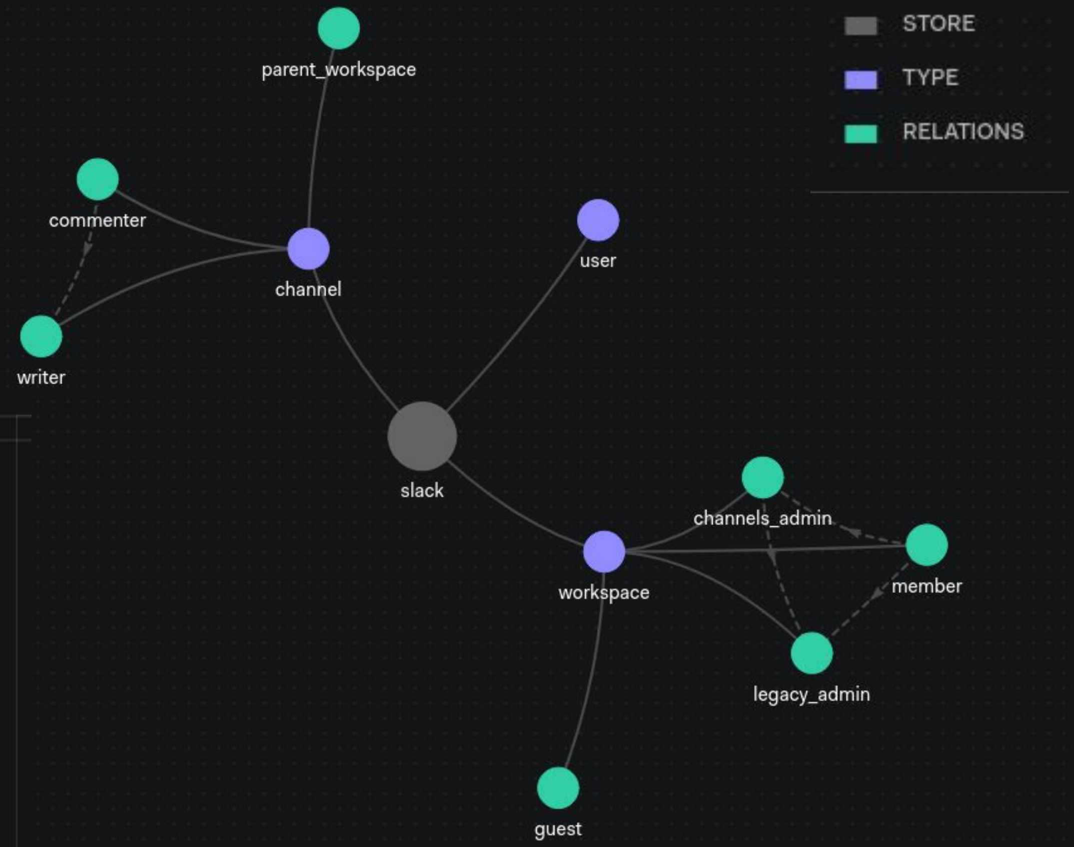
```
model
  schema 1.1

type channel
  relations
    define commenter: [user, workspace#member] or writer
    define parent_workspace: [workspace]
    define writer: [user, workspace#member]

type user

type workspace
  relations
    define channels_admin: [user] or legacy_admin
    define guest: [user]
    define legacy_admin: [user]
    define member: [user] or legacy_admin or channels_admin

```

Ref: auth0 fga playground [https://play.fga.dev/sandbox/?store=slack](https://play.fga.dev/sandbox/?store=slack)

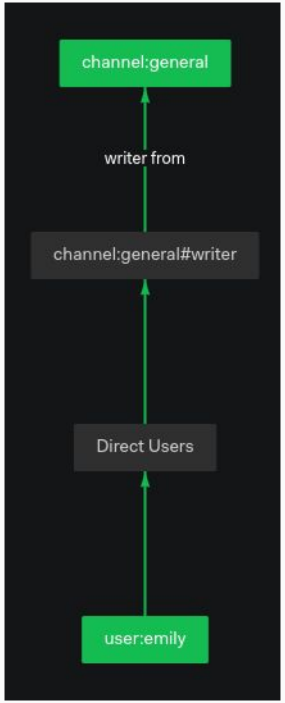
The model/scheme describes the relationships which **CAN** exist. The Tuples in the database explain which **DO** exist



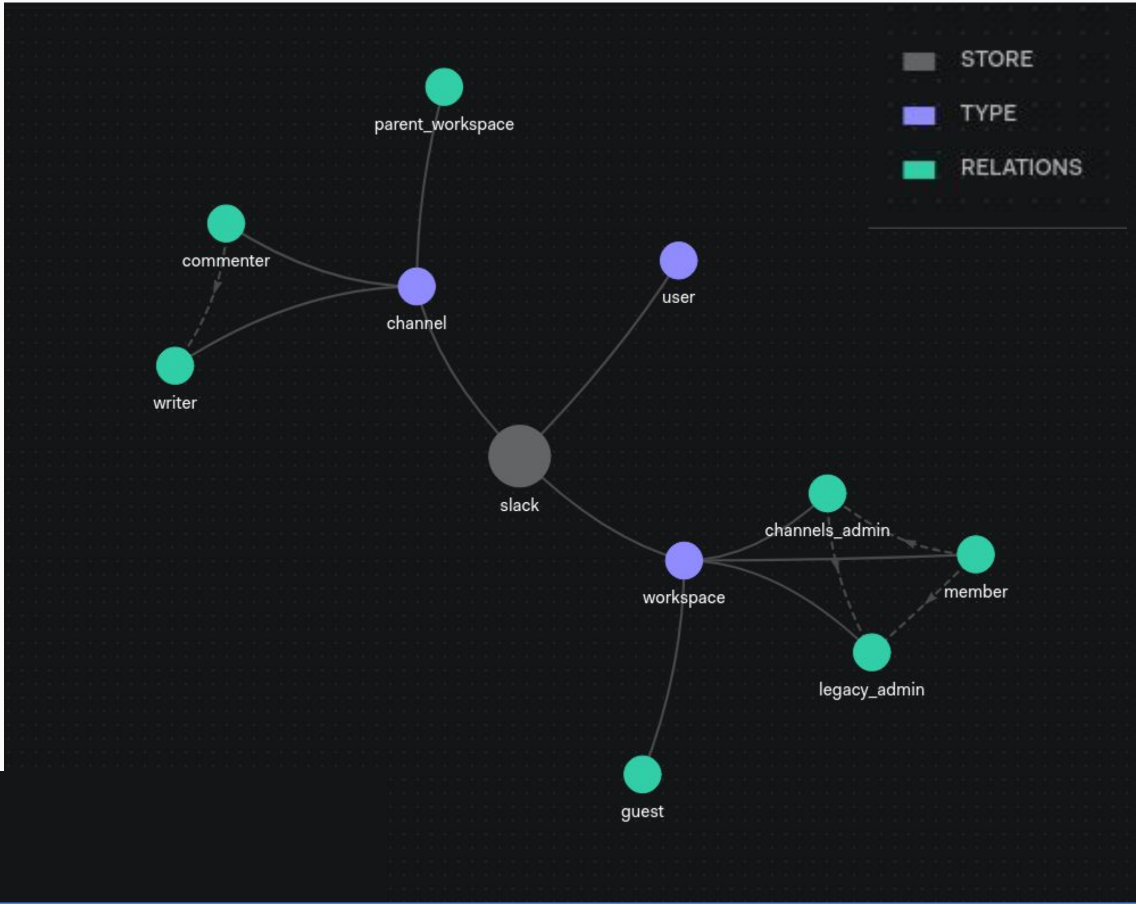
```
model
  schema 1.1

type channel
  relations
    define commenter: [user, workspace#member] or writer
    define parent_workspace: [workspace]
    define writer: [user, workspace#member]

type user

type workspace
  relations
    define channels_admin: [user] or legacy_admin
    define guest: [user]
    define legacy_admin: [user]
    define member: [user] or legacy_admin or channels_admin
```



Ref: auth0 fga playground https://play.fga.dev/sandbox/?store=slack



is user:emily related to channel:general as writer?

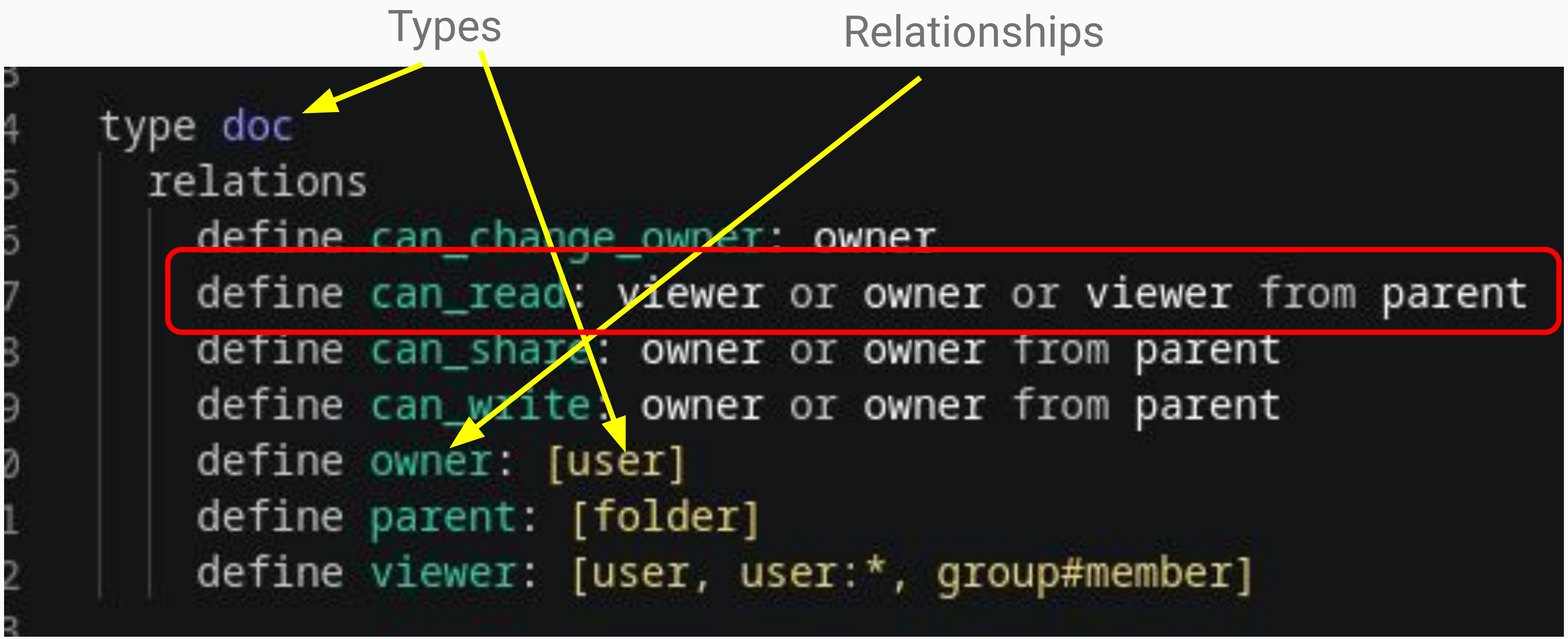


Tuple (stored in database)

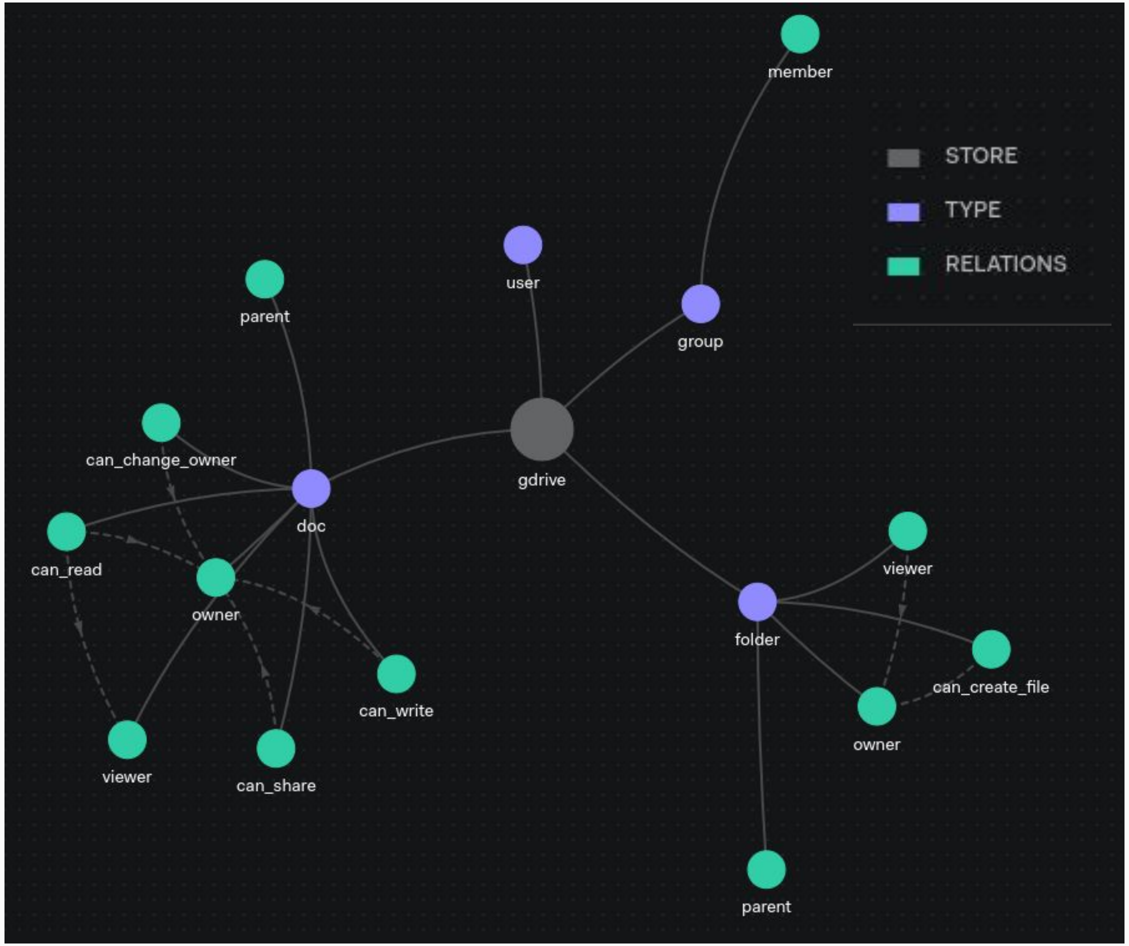
| | |
|---|---|
| USER | user:emily |
| RELATION | writer |
| OBJECT | channel:general |

Ref: auth0 fga playground https://play.fga.dev/sandbox/?store=slack

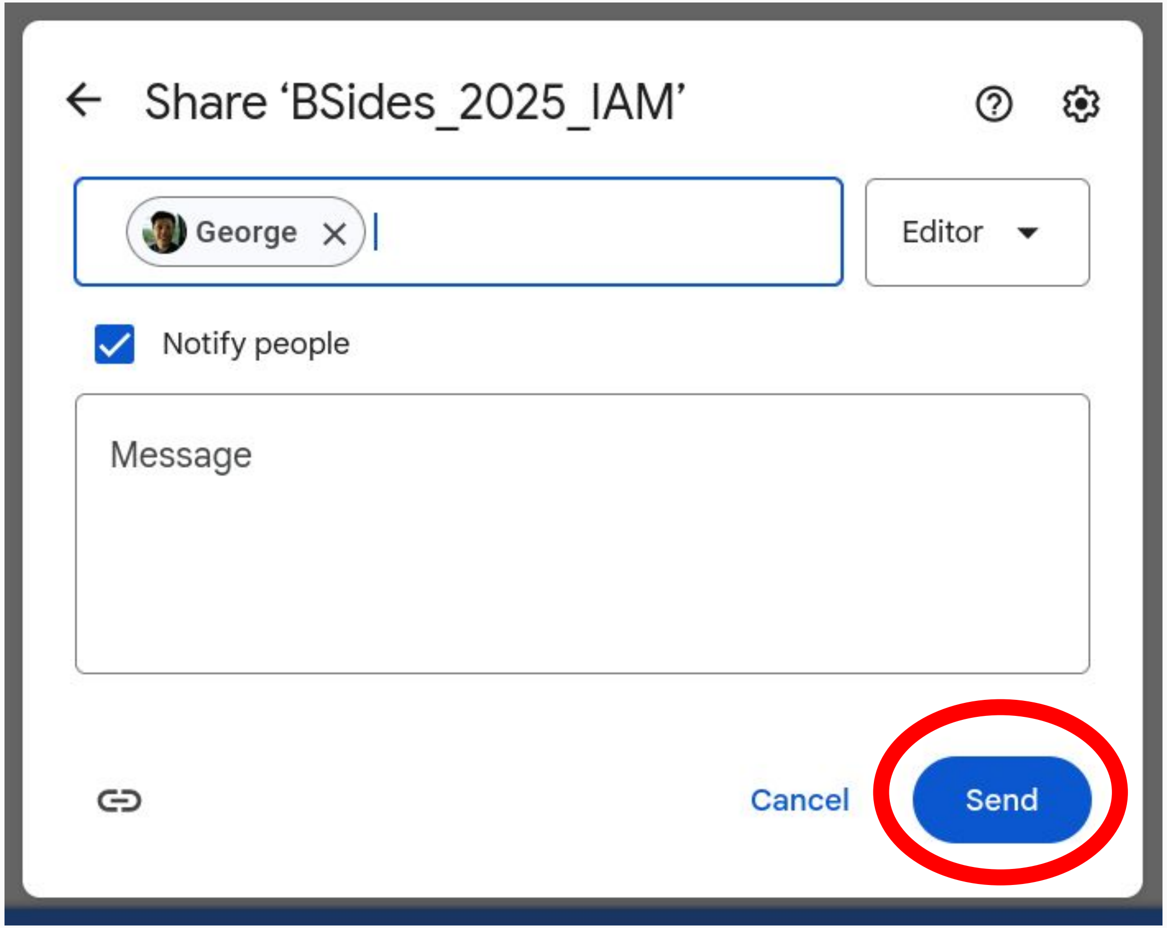
# Relationship-Based Access Control (ReBAC) Google Drive Example



```
type doc
  relations
    define can_change_owner: owner
    define can_read: viewer or owner or viewer from parent
    define can_share: owner or owner from parent
    define can_write: owner or owner from parent
    define owner: [user]
    define parent: [folder]
    define viewer: [user, user:*, group#member]
```



Ref: auth0 fga playground https://play.fga.dev/sandbox/?store=slack



Share 'BSides_2025_IAM'

George

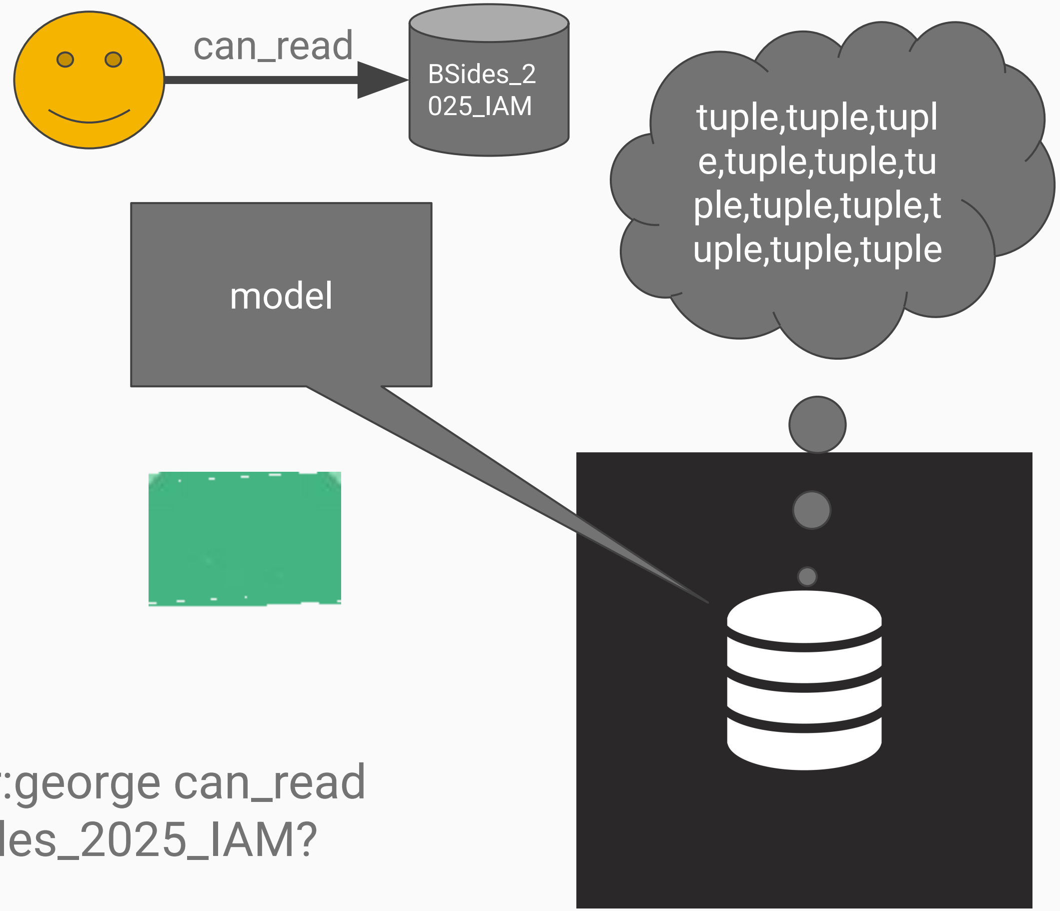
Editor

Notify people

Message

Tuple

- USER = user:george
- RELATION = editor
- OBJECT = doc:BSides_2025_IAM


can_read
BSides_2025_IAM
model
tuple,tuple,tuple,tuple,tuple,tuple,tuple,tuple,tuple,tuple,tuple

https://docs.google.com/presentation/d/1mZbSLRU21KjD5D8Ymc05ut-hX_rKt6fcUJVvl2FxgZQ/edit?usp=sharing

can user:george can_read doc:BSides_2025_IAM?

## Relationship-Based Access Control (ReBAC)

- **Advantages**
  - Expressiveness and Flexibility
  - Scale
  - Dynamic
  - Fast
  - Supports Reverse Indices: ReBAC can easily answer questions like "Who has access to this particular resource?"
  - Support for conditional relationships like “time of day” (coming soon*)

- **Disadvantages**
  - Big “startup” cost to implement
  - Duplication of business data -> tuples
  - Few solutions out there
  - Auditing - dynamic nature of ReBAC policies and relationships
  - Modeling relationships can be complicated

## That's all the theory!

Let's talk implementation!

For building authz into your own applications and integrating open source projects into your business

## Relationship-Based Access Control (ReBAC) Implementation

authzed

- https://authzed.com/
- Made by Ex-Googlers
- Inspired by Zanzibar

Open Source Project, self host
Or Cloud hosted paid product

spicedb

https://github.com/authzed/spicedb

# Relationship-Based Access Control (ReBAC) Implementation

OpenFGA™

- OpenFGA: Fine-Grained Authorization
- https://openfga.dev/
- https://github.com/openfga/openfga
- Initially developed by Auth0/Okta
- Inspired by Zanzibar
- Open Source Project

CNCF Sandbox Project

# Policy-Based Access Control (PBAC)

Open Policy Agent

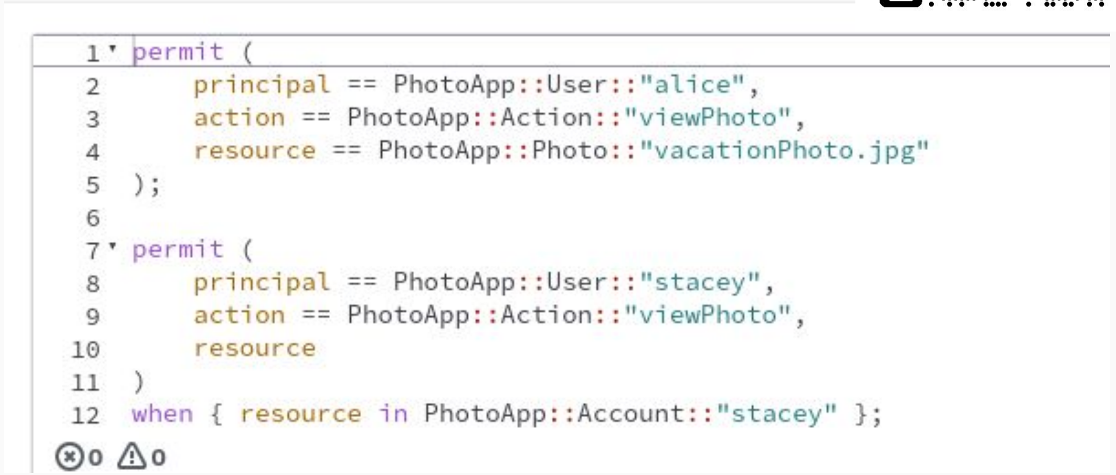
OPA is a policy engine that streamlines policy management across your stack for improved development, security and audit capability.

- https://www.openpolicyagent.org/
- **Policies are written in Rego**
- **Can read data from external sources to make policy decisions**
- Very widely supported
  - Gatekeeper for Policy Controller for Kubernetes
  - **Envoy proxy for easy access proxies, API authz or zero trust proxies**
- Great developer support and SDKs

envoy

# Policy-Based Access Control (PBAC)

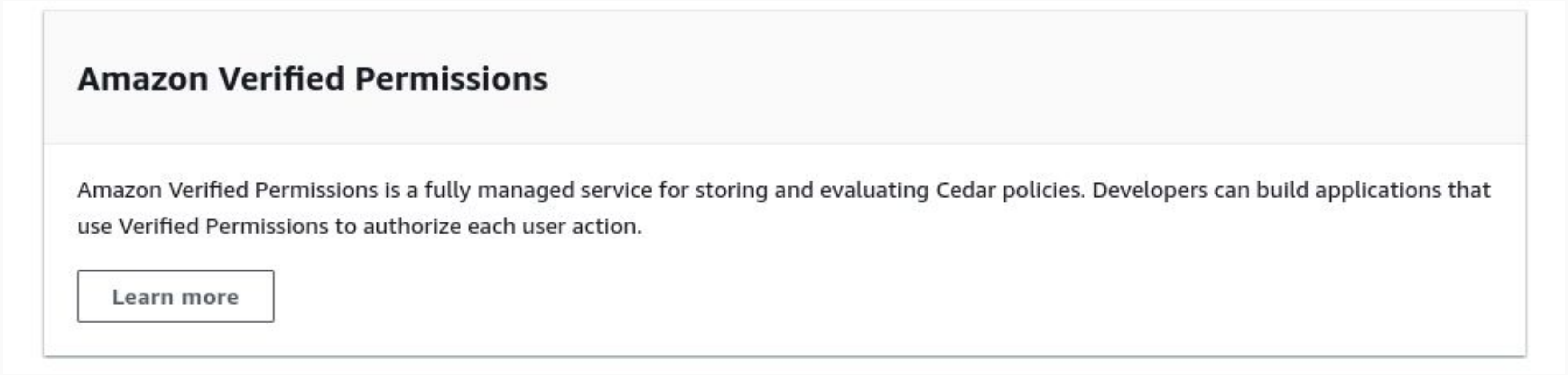
- https://www.cedarpolicy.com/
- Open Sourced by Amazon
- **Based on AWS IAM**



```
permit (
    principal == PhotoApp::User::"alice",
    action == PhotoApp::Action::"viewPhoto",
    resource == PhotoApp::Photo::"vacationPhoto.jpg"
);

permit (
    principal == PhotoApp::User::"stacey",
    action == PhotoApp::Action::"viewPhoto",
    resource
)
when { resource in PhotoApp::Account::"stacey" };
```

Ref: https://aws.amazon.com/about-aws/whats-new/2023/05/cedar-open-source-language-access-control/

# Policy-Based Access Control (PBAC)



**Amazon Verified Permissions**

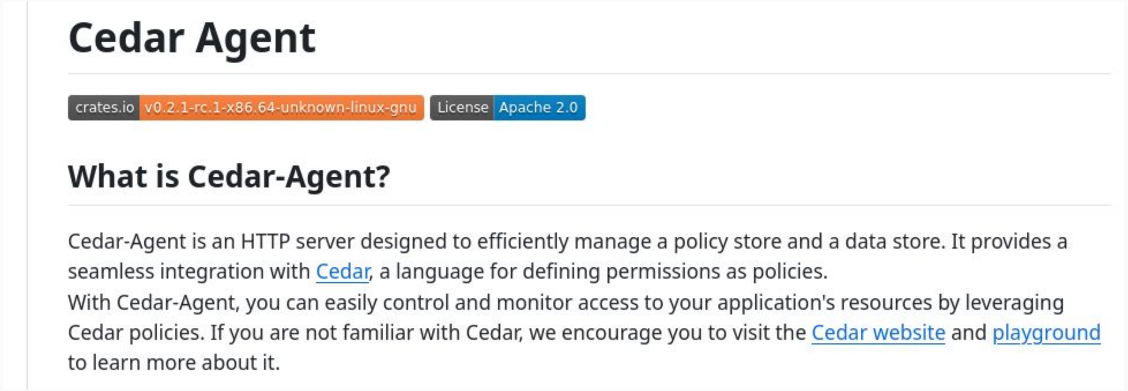
Amazon Verified Permissions is a fully managed service for storing and evaluating Cedar policies. Developers can build applications that use Verified Permissions to authorize each user action.

Good for if your building
your own product!

Ref: https://aws.amazon.com/about-aws/whats-new/2023/05/cedar-open-source-language-access-control/

# Policy-Based Access Control (PBAC)

- https://www.permit.io/
- Inspired by Cedar
- https://github.com/permitio/cedar-agent
- Open source Project
- Paid Cloud Service



## Cedar Agent

crates.io v0.2.1-rc.1-x86.64-unknown-linux-gnu License Apache 2.0

### What is Cedar-Agent?

Cedar-Agent is an HTTP server designed to efficiently manage a policy store and a data store. It provides a seamless integration with Cedar, a language for defining permissions as policies.
With Cedar-Agent, you can easily control and monitor access to your application's resources by leveraging Cedar policies. If you are not familiar with Cedar, we encourage you to visit the Cedar website and playground to learn more about it.

Ref: https://www.permit.io/blog/oss-aws-cedar-is-a-gamechanger-for-iam

# Policy-Based Access Control (PBAC)

Permit.io

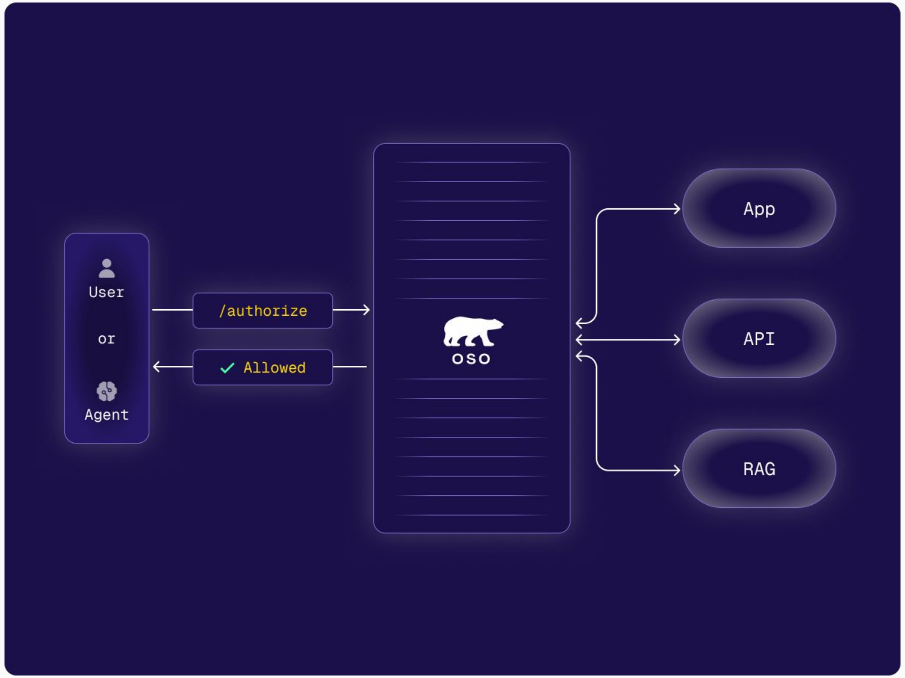
⚡OPAL⚡

Open Policy Administration Layer

- Sister Project
  - https://github.com/permitio/opal
  - OPAL is an administration layer for Policy Engines such as Open Policy Agent (OPA), and AWS' Cedar Agent detecting changes to both policy and policy data in realtime and pushing live updates to your agents. OPAL brings open-policy up to the speed needed by live applications.

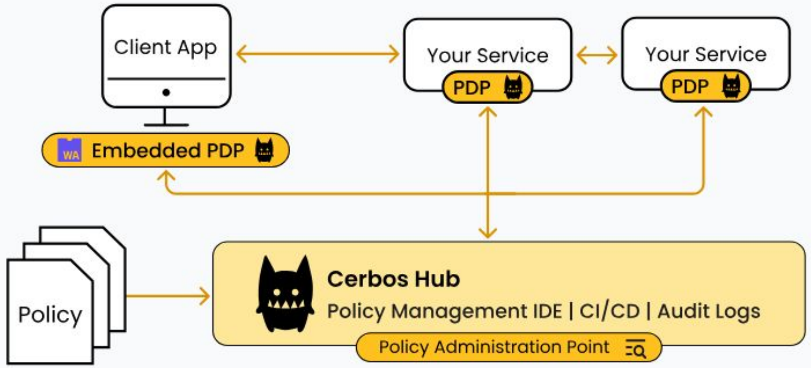
Ref: https://www.permit.io/blog/oss-aws-cedar-is-a-gamechanger-for-iam

# Policy-Based Access Control (PBAC)



- https://www.osohq.com/
- "Oso is authorization as a service"
- Paid Cloud Service
- Cloud Only (on prem soon ™ )
- Integrates well with existing data sources
  - Opinion: better data model than Zanzibar
- Supports:
  - RBAC
  - ReBAC
  - ABAC

# Policy-Based Access Control (PBAC)



- https://www.cerbos.dev/
- https://github.com/cerbos/cerbos
- Authorization engine is open source
- All on prem
- Supports:
  - RBAC
  - ABAC
- Cloud offering
  - “Cerbos Hub”
    - Policy Management IDE
    - Centralised Audit Logs

# TODO - https://casbin.org/

## The Great Compromise - Keep a decision log!

- Easy of Use
- Maintainability
- Expressiveness of your policy language
- Compatibility
- Cost (both for services, transition and ongoing maintenance)

**“Perfect is the enemy of good”**

- Voltaire

## Thank you!

Any questions?

My Website ->

<- Slides

Add me on Linkedin ->