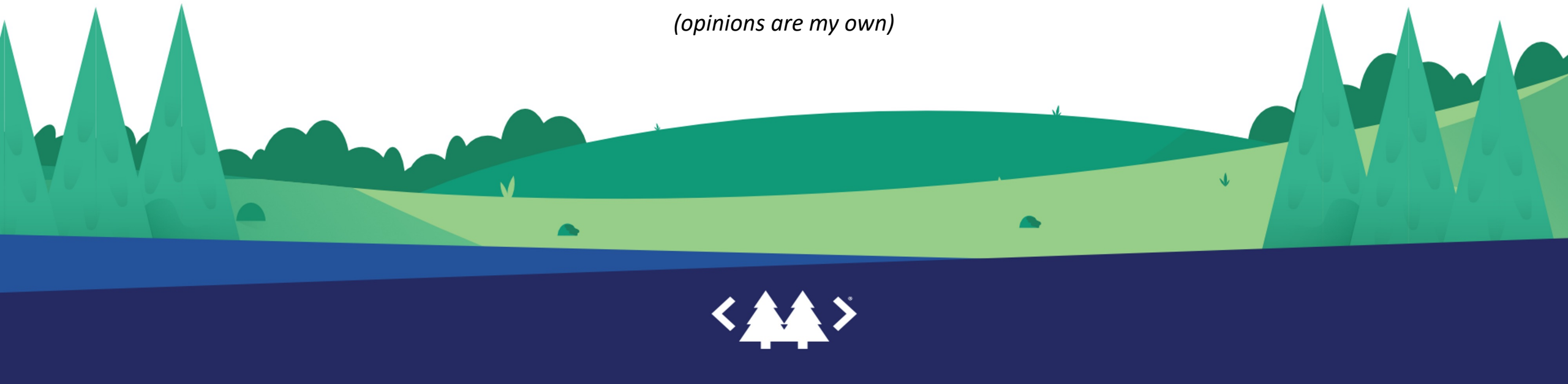
Fine-Grained Authorization in Modern Software Applications

Kevin Hakanson (he/him)
Sr. Solutions Architect, AWS

(opinions are my own)



SPECIAL THANKS TO ALL OUR AWESOME CAMP SPONSORS!

Unspecified

HEARTLAND BUSINESS SYSTEMS

CLOUDFLARE

DATASTAX

Temporal

PROPELAUTH

PIECES FOR DEVELOPERS

redgate

MESCIUS FORMERLY GRAPECITY, INC.

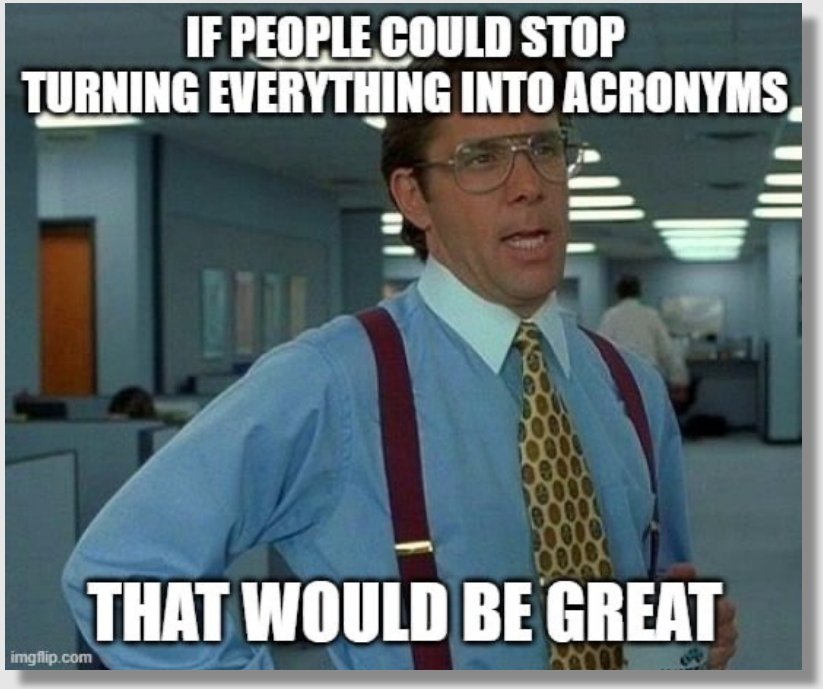
AWS Amplify

Acronyms, Buzzwords, blah, blah, blah

Authentication (AuthN) and Authorization (AuthZ) are critical for most software applications. The increased adoption of standardized frameworks for AuthN has improved overall security posture. “Broken Authentication” was #2 risk on the OWASP Top 10:2017 list but slid in 2021 to be part of a rescoped #7. AuthZ is trending the wrong direction with “Broken Access Control” the #1 security risk on 2021 list. This session discusses how open-source policy languages and evaluation engines can improve access control in applications.

The key acronyms are reviewed for background: JWT concepts (claims, scopes); access control models (RBAC, ABAC, ReBAC), data-flow model of XACML (PAP, PDP, PEP, PIP). Examples of applications requiring fine-grained authorization are modeled using different open-source solutions (Cedar, OpenFGA, OPA) focusing on their policy language and evaluation engine integration. This session spans high-level architecture to low-level code, and sprinkles humor (and acronyms) throughout.

Poll: Who said it better?



Authentication vs Authorization

"In some systems, complete access is granted after successful authentication of the user, but most systems require more sophisticated and complex control. In addition to the *authentication* mechanism (such as a password), access control is concerned with how *authorizations* are structured."

- NISTIR 7316 Assessment of Access Control Systems

Authentication

Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in a system.

Authorization

The granting or denying of access rights to a user, program, or process.

Source: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7316.pdf


AUTHENTICATION
AUTHORIZATION
imgflip.com


You can't spend rest of your life alone, dressed in black, listening to angry music, staying up all night.
Yes, I can, 'cause I'm Batman.
ifunny.co

OWASP Top 10:2021

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

- **A01:2021-Broken Access Control** moves up from the fifth position; 94% of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category.

- **A07:2021-Identification and Authentication Failures** was previously Broken Authentication and is sliding down from the second position, and now includes CWEs that are more related to identification failures. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks seems to be helping.

Source: https://owasp.org/www-project-top-ten/

History of “Broken Authentication”

Application functions related to authentication and session management are often implemented incorrectly…

- #7 in 2021 - Identification and Authentication Failures
- #2 in 2017 - Broken Authentication
- #2 in 2013 - Broken Authentication and Session Management
- #3 in 2010 - Broken Authentication and Session Management
- #7 in 2007 - Broken Authentication and Session Management

OpenID Connect (OIDC)

OpenID Connect (February 2014) implements authentication as an extension to the OAuth 2.0 (October 2012) authorization process.

The primary extension is the **ID Token** data structure represented as a JSON Web Token (JWT). It has Claims expressing such information as the Issuer, the Subject Identifier, when the authentication was performed, etc.

Source: https://openid.net/specs/openid-connect-core-1_0.html

Common Claim Names in JWTs

The "**sub**" (subject) claim identifies the principal that is the subject of the JWT. The claims in a JWT are normally statements about the subject. The subject value MUST either be scoped to be locally unique in the context of the issuer or be globally unique.

The "**iss**" (issuer) claim identifies the principal that issued the JWT.

The "**aud**" (audience) claim identifies the recipients that the JWT is intended for.

Source: https://datatracker.ietf.org/doc/html/rfc7519#page-9

History of “Broken Access Control”

Restrictions on what authenticated users are allowed to do are often not properly enforced…

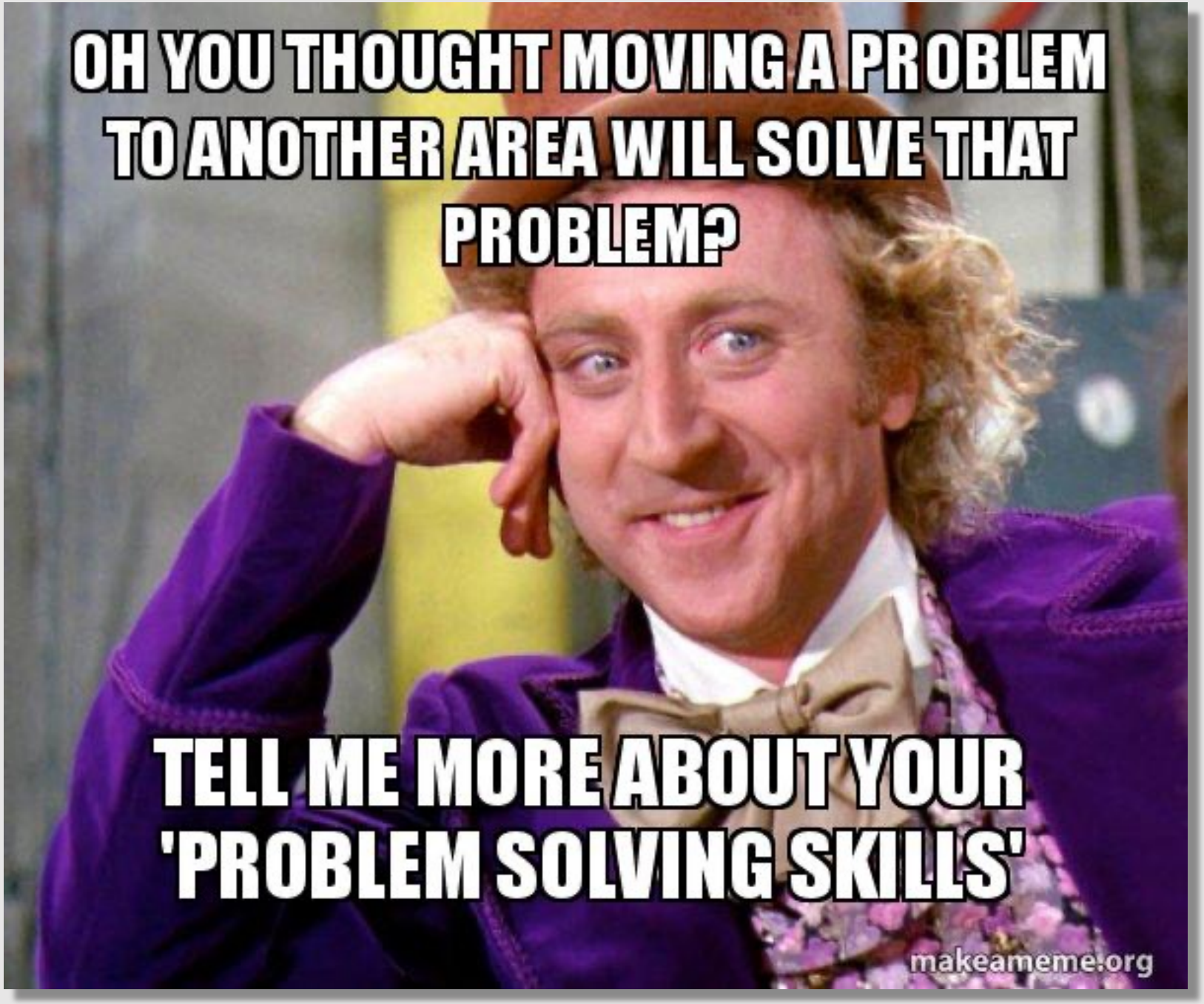
- Broken Access Control (#1 in 2021, #5 in 2017)
- Insecure Direct Object References (#4 in 2013, 2010, 2007)
- Missing Function Level Access Control (#7 in 2013)
- Failure to Restrict URL Access (#8 in 2010, #10 in 2007)

OWASP Top 10 API Security Risks – 2023

More specific to APIs (mobile, desktop, web) vs other generic application security risks you might find in APIs.

- API1:2023 - Broken Object Level Authorization
- API2:2023 - Broken Authentication
- API3:2023 - Broken Object Property Level Authorization
- API5:2023 - Broken Function Level Authorization

Source: https://owasp.org/API-Security/editions/2023/en/0x11-t10/


OH YOU THOUGHT MOVING A PROBLEM TO ANOTHER AREA WILL SOLVE THAT PROBLEM?
TELL ME MORE ABOUT YOUR 'PROBLEM SOLVING SKILLS'
makeameme.org

Access Token

Access tokens are credentials used to access protected resources.

The token may denote an identifier used to retrieve the authorization information or may self-contain the authorization information in a verifiable manner (i.e., a token string consisting of some data and a signature).

Source: https://www.rfc-editor.org/rfc/rfc6749#section-1.4

JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens [RFC9068]

Provide a standardized and interoperable profile as an alternative to the proprietary JWT access token layouts going forward.

An authorization server wanting to include [authorization] attributes in a JWT access token SHOULD use the "**groups**", "**roles**", and "**entitlements**" attributes of the "User" resource schema defined by Section 4.1.2 of [RFC7643]) as claim types.

Source: https://www.rfc-editor.org/rfc/rfc9068

System for Cross-domain Identity Management (SCIM): Core Schema [RFC7643]

groups

A list of groups to which the user belongs. The values are meant to enable expression of common group-based or role-based access control models, although no explicit authorization model is defined.

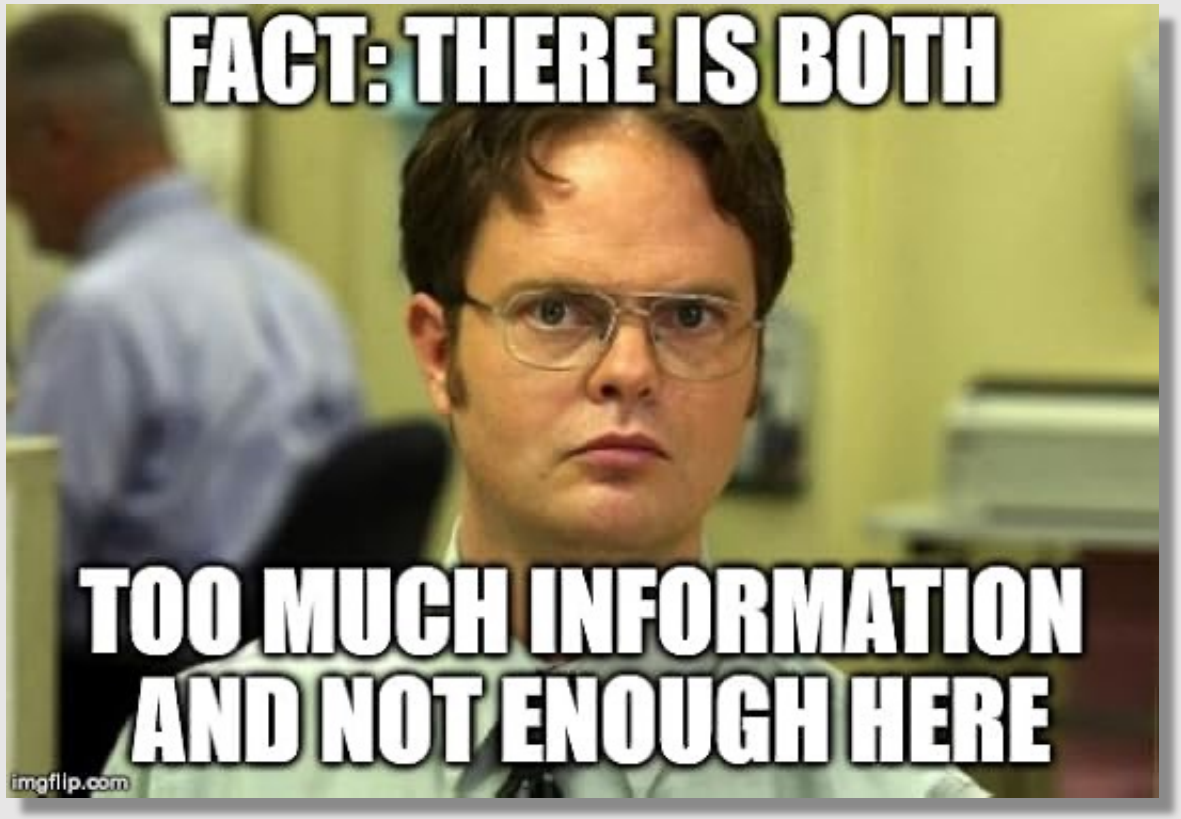
entitlements

A list of entitlements for the user that represent a thing the user has. An entitlement may be an additional right to a thing, object, or service. No vocabulary or syntax is specified.

roles

A list of roles for the user that collectively represent who the user is, e.g., "Student", "Faculty". No vocabulary or syntax is specified, although it is expected that a role value is a String or label representing a collection of entitlements.

Source: https://www.rfc-editor.org/rfc/rfc7643.html#section-4.1.2


FACT: THERE IS BOTH
TOO MUCH INFORMATION
AND NOT ENOUGH HERE
imgflip.com


STOP! JUST STOP!
THERE IS TOO MUCH
INFORMATION!

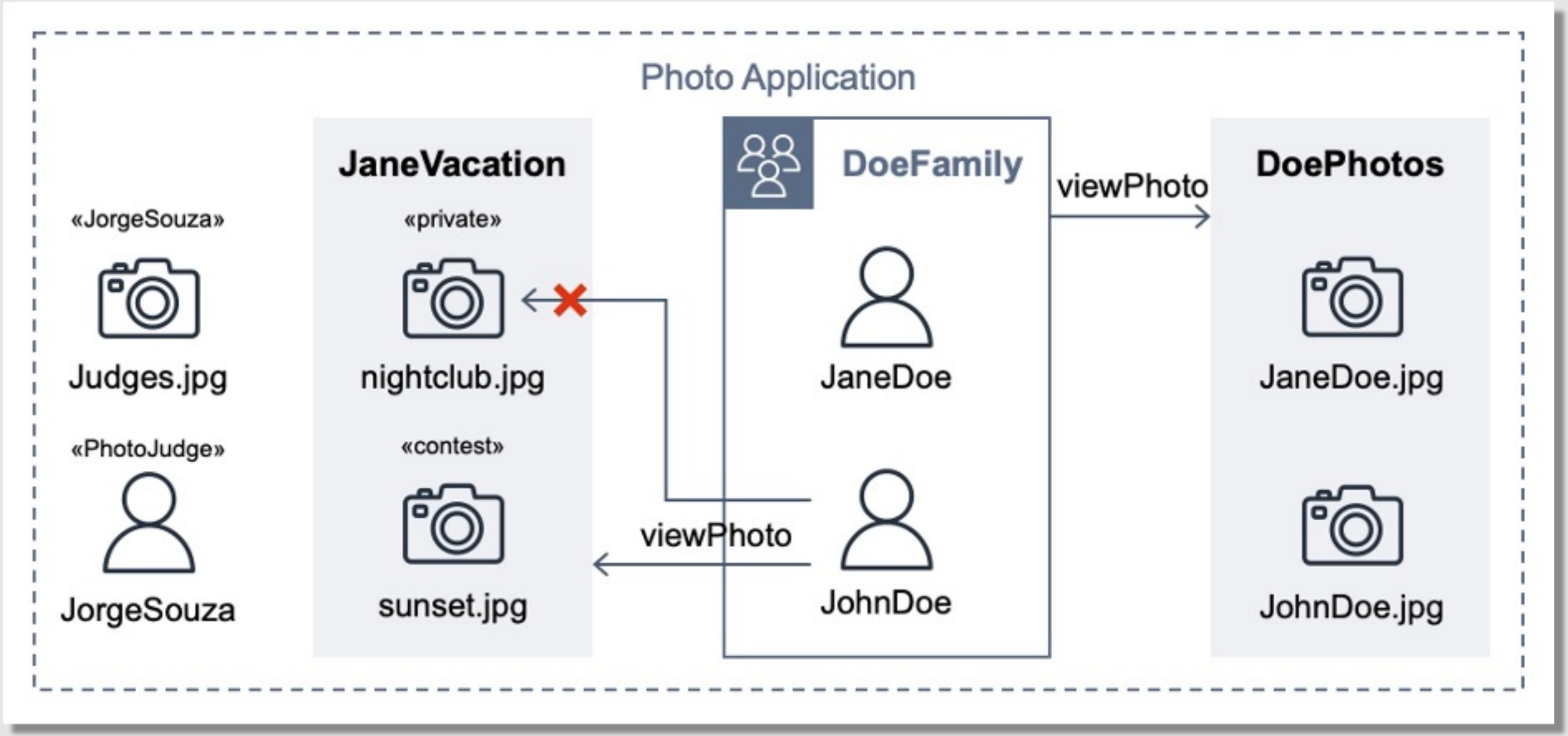


Photo Application
JaneVacation
DoeFamily
DoePhotos
viewPhoto
«JorgeSouza»
«private»
Judges.jpg
nightclub.jpg
JaneDoe
JaneDoe.jpg
«PhotoJudge»
«contest»
viewPhoto
JorgeSouza
sunset.jpg
JohnDoe
JohnDoe.jpg

Authorization Stories

- UserGroup DoeFamily has view access to Album DoePhotos
- Everyone has view, edit, and delete access to any Photo where they are the owner
- User JohnDoe has view access to Album JaneVacation, but no one but the owner has access to Photos whose labels include the string "private"
- Role PhotoJudge has view access to Photos whose labels include the string "contest" when in a judgingSession context
- A User has view access to Photos whose optional subjects include their User entity

Source: https://github.com/aws-samples/cedar-policy-validation-pipeline/tree/main/policystore/tests

Access Control Policies

Discretionary access control (DAC)

leaves a certain amount of access control to the discretion of the object's owner, or anyone else who is authorized to control the object's access. The owner can determine who should have access rights to an object and what those rights should be.

Mandatory access control (MAC)

means that access control policy decisions are made by a central authority, not by the individual owner of an object. User cannot change access rights.

Source: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-192.pdf

ABAC and RBAC

Attribute Based Access Control (ABAC): An access control approach in which access is mediated based on attributes associated with subjects (requesters) and the objects to be accessed.

Access to an object is authorized or denied depending upon whether the required (e.g., policy-defined) correlation can be made between the attributes of that object and of the requesting subject.

Role Based Access Control (RBAC): A model for controlling access to resources where permitted actions on resources are identified with roles rather than with individual subject identities.

Source: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf

Theoretical (partial) JWT Access Tokens

Questions about groups, roles, and entitlements:

- How are they populated?
  - Custom authorization servers / enriching Access Token
- How are they updated?
  - Use Refresh Token to obtain new Access Token

```
{
  "sub": "JaneDoe",
  "iss": "idp.example.com",
  "aud": "photoapp.example.com",
  "groups": ["DoeFamily"],
  "roles": [],
  "entitlements": ["PhotoApp"],
}
```

```
{
  "sub": "JorgeSouza",
  "iss": "idp.example.com",
  "aud": "photoapp.example.com",
  "groups": [],
  "roles": ["PhotoJudge"],
  "entitlements": ["PhotoApp"],
}
```

What is Policy-as-Code?

Policy-as-Code refers to the practice of managing and implementing policy decisions through code, making them enforceable and verifiable within IT environments.

This approach enables automated enforcement, validation, and auditing of policies, typically used in areas like security, compliance, and infrastructure configuration.

Source: https://www.strongdm.com/what-is/policy-as-code

eXtensible Access Control Markup Language (XACML)

Policy administration point (PAP)

The system entity that creates a ***policy*** or ***policy set***

Policy decision point (PDP)

The system entity that evaluates ***applicable policy*** and renders an ***authorization decision***.

Policy enforcement point (PEP)

The system entity that performs ***access control***, by making ***decision requests*** and enforcing ***authorization decisions***.

Policy information point (PIP)

The system entity that acts as a source of ***attribute*** values

Source: https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html


CEDAR

Cedar

- Open source policy language and evaluation engine.
- Enables developers to express fine-grained permissions as easy-to-understand policies enforced in their applications, and decouple access control from application logic.
- Supports common authorization models such as role-based access control and attribute-based access control.
- Built from the ground up to be verified formally by using automated reasoning, and tested rigorously using differential random testing.

Source: https://github.com/cedar-policy

Cedar key concepts

Authorization **policies** describe who (the *principal*) is allowed to perform which *actions*, on which *resources*, and in what *context*.

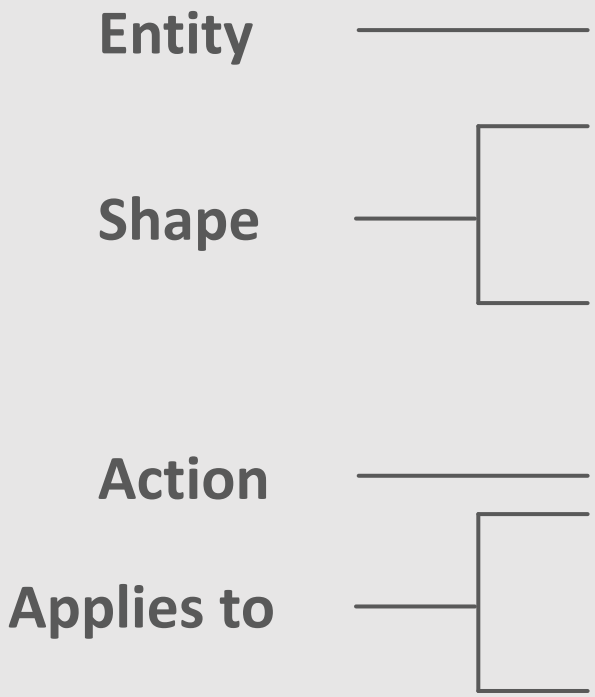
Entities are application data that Cedar needs to see to make authorization decisions.

The **context** is a part of a request that consists of transient data.

A **schema** defines the types of entities recognized by the application, and which are also the principals, resources, and actions referenced in policies.

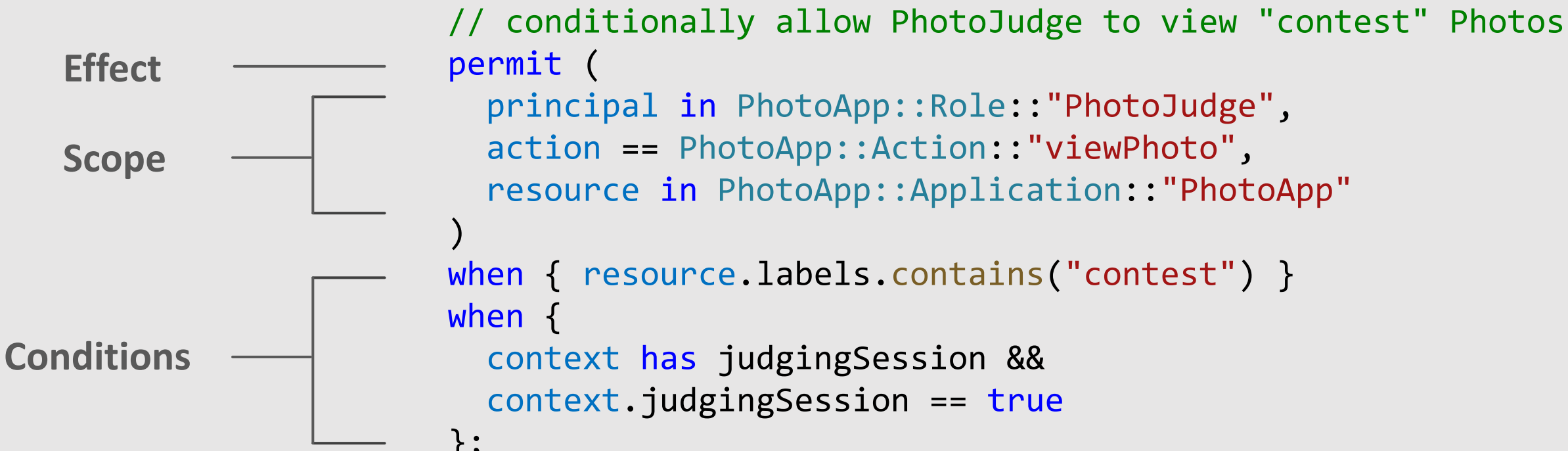
Source: https://docs.cedarpolicy.com/

Cedar schema



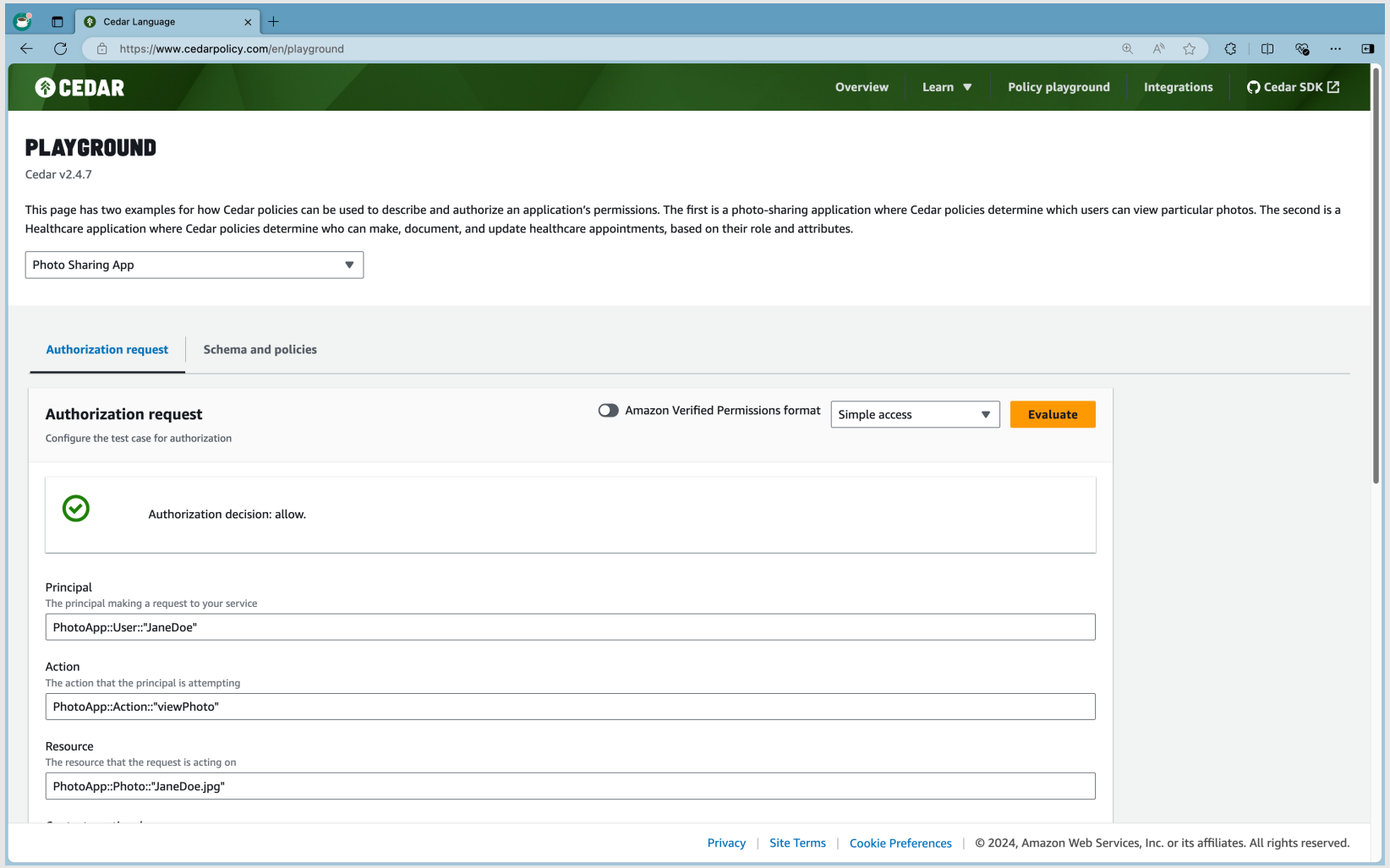

```
namespace PhotoApp {
  entity User in [UserGroup, Role];
  entity Photo in [Album, Application] = {
    "labels": Set<String>,
    "owner": User,
    "subjects"?: Set<User>
  };
  entity UserGroup, Role, Album, Application;
  action "viewPhoto" appliesTo {
    principal: [User],
    resource: [Photo],
    context: {"judgingSession"?: Bool}
  };
}
```

Cedar policy



Cedar entities

www.cedarpolicy.com/en/playground



Cedar CLI

```
{
  "principal": "PhotoApp::User::\"JorgeSouza\"",
  "action": "PhotoApp::Action::\"viewPhoto\"",
  "resource": "PhotoApp::Photo::\"sunset.jpg\"",
  "context": { "judgingSession": true }
}
```

```
cedar authorize -v -f plain \
  --policies testdata/temppolicies.cedar \
  --entities cedarentities.json \
  --request-json "testdata/Role PhotoJudge/ALLOW/JorgeSouza-view-sunset-session.cedarauth.json"

ALLOW

note: this decision was due to the following policies:
PhotoJudge
```

@cedar-policy/cedar-wasm

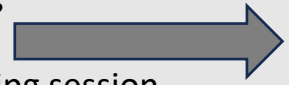
```
const cedar = require('@cedar-policy/cedar-wasm/nodejs’);
const policies: PolicySet = readFileSync('./testdata/temppolicies.cedar', 'utf-8');
const entitiesJson: string = readFileSync('./cedarentities.json', 'utf-8');
const entities: Array<EntityJson> = JSON.parse(entitiesJson);

function viewPhoto(decision: Decision, principal: TypeAndId,
                   resource: TypeAndId, context: Context)
{
  const answer = cedar.isAuthorized({
    principal: principal,
    action: { type: 'PhotoApp::Action', id: 'viewPhoto' },
    resource: resource,
    context: context,
    slice: { policies: policies, entities: entities,},
  });

  expect(answer.type).toBe('success’);
  expect(answer.response.decision).toBe(decision);
}
```

```
✓ cedar.test.ts (13)
  ✓ Cedar version (1)
    ✓ Cedar version: 3.2.1
  ✓ Album DoePhotos (3)
    ✓ JaneDoe view JohnDoe.jpg
    ✓ JohnDoe view JaneDoe.jpg
    ✓ JorgeSouza view JaneDoe.jpg
  ✓ Album JaneVacation (4)
    ✓ JaneDoe view nightclub.jpg
    ✓ JohnDoe view nightclub.jpg
    ✓ JaneDoe view sunset.jpg
    ✓ JohnDoe view sunset.jpg
  ✓ Role PhotoJudge (2)
    ✓ JorgeSouza view sunset.jpg session
    ✓ JorgeSouza view sunset.jpg nosession
  ✓ User JorgeSouza (3)
    ✓ JorgeSouza view sunset.jpg
    ✓ JorgeSouza view nightclub.jpg
    ✓ JorgeSouza view sunset.jpg

Test Files  1 passed (1)
     Tests  13 passed (13)
```



```ts
const JORGESOUZA = { type: 'PhotoApp::User', id: 'JorgeSouza' };
const SUNSET_JPG = { type: 'PhotoApp::Photo', id: 'sunset.jpg' };

suite('Role PhotoJudge', async () => {
  test('JorgeSouza view sunset.jpg session', () => {
    viewPhoto('Allow', JORGESOUZA, SUNSET_JPG, { judgingSession: true });
  });

  test('JorgeSouza view sunset.jpg nosession', () => {
    viewPhoto('Deny', JORGESOUZA, SUNSET_JPG, { judgingSession: false });
  });
})
```

Cedar performance

Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization (Extended Version)

Comparing Cedar to two open-source languages, OpenFGA and Rego, we find (subjectively) that Cedar has equally or more readable policies, but (objectively) performs far better.

Source: https://arxiv.org/pdf/2403.04651

Open Policy Agent

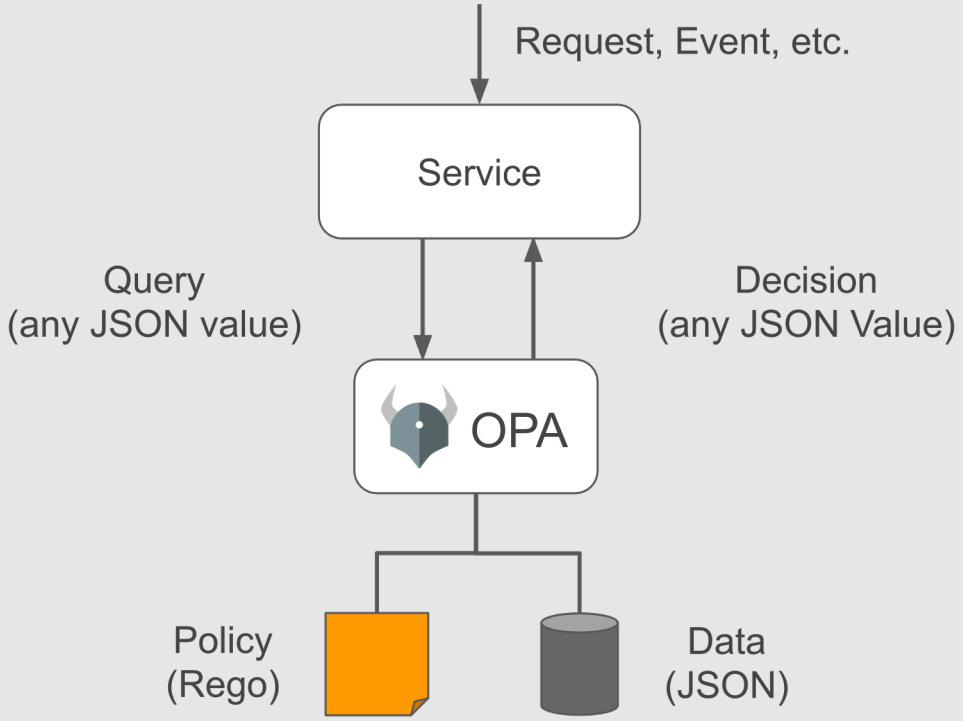
Open Policy Agent (OPA)

Declarative

- Express policy in a high-level, declarative language that promotes safe, performant, fine-grained controls.
- Use a purpose-built, declarative policy language (Rego) for a world where JSON is pervasive.
- Iterate, traverse hierarchies, and apply 150+ built-ins like string manipulation and JWT decoding to declare the policies you want enforced.

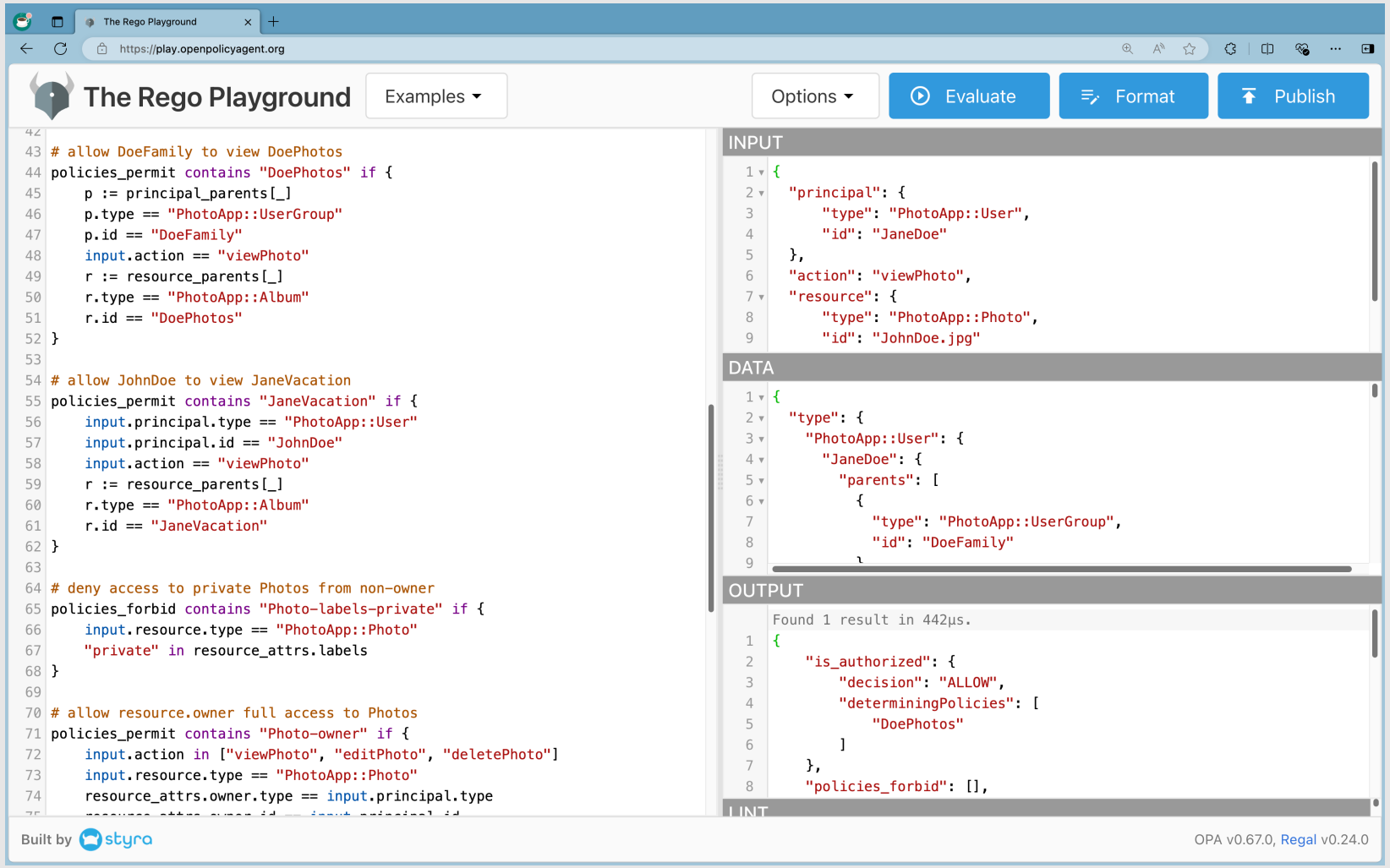
Source: https://www.openpolicyagent.org/

- When your software needs to make policy decisions it queries OPA and supplies arbitrary structured data (e.g., JSON) as input.
- OPA generates policy decisions by evaluating the query input against policies and data.
- Policy decisions are not limited to simple allow/deny answers. Like query inputs, your policies can generate arbitrary structured data as output.



Source: https://www.openpolicyagent.org/docs/latest/

play.openpolicyagent.org



```
{
  "type": {
    "PhotoApp::User": {
      "JaneDoe": {
        "parents": [
           { "type": "PhotoApp::UserGroup", "id": "DoeFamily" }
         ]
    },
    "PhotoApp::Photo": {
      "JohnDoe.jpg": {
        "parents": [
          { "type": "PhotoApp::Album", "id": "DoePhotos" }
        ],
        "attrs": {
          "labels": [],
          "owner": {"type": "PhotoApp::User", "id": "JohnDoe" }
      }
    }
  }
}
```

```
# default DENY
default is_authorized := {
  "decision": "DENY", # ALLOW | DENY
  "determiningPolicies": [],
}

# forbid policies always DENY authorization
is_authorized := {
  "decision": "DENY",
  "determiningPolicies": policies_forbid,
} if {
  count(policies_forbid) > 0
}
```

```
# forbid overrides permit
is_authorized := {
  "decision": "ALLOW",
  "determiningPolicies": policies_permit,
} if {
  count(policies_forbid) == 0
  count(policies_permit) > 0
}
```

```
## helper rules

principal_parents := data.type[input.principal.type][input.principal.id].parents

resource_parents := data.type[input.resource.type][input.resource.id].parents

resource_attrs := data.type[input.resource.type][input.resource.id].attrs
```

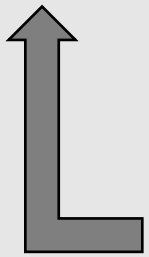
```
{
  "principal": {
    "type": "PhotoApp::User",
    "id": "JaneDoe"
  },
  "action": "viewPhoto",
  "resource": {
    "type": "PhotoApp::Photo",
    "id": "JohnDoe.jpg"
  },
  "context": {}
}
```

```
# allow DoeFamily to view DoePhotos
policies_permit contains "DoePhotos" if {
  some p in principal_parents
  p.type == "PhotoApp::UserGroup"
  p.id == "DoeFamily"
  input.action == "viewPhoto"
  some r in resource_parents
  r.type == "PhotoApp::Album"
  r.id == "DoePhotos"
}
```

```
opa eval "data.photoapp.is_authorized" \
  --schema schemas/ \
  --format pretty \
  --data photoapp.rego \
  --data data.json \
  --input input/JaneDoe-view-JohnDoe.regoinput.json
```

```
{
  "decision": "ALLOW",
  "determiningPolicies": [
    "DoePhotos"
  ]
}
```

```
opa test data.json photoapp.rego photoapp_test.rego -v
photoapp_test.rego:
data.photoapptest.Album_DoePhotos.test_JaneDoe_viewPhoto_JohnDoe: PASS (1.393502ms)
--------------------------------------------------------------------------------
PASS: 1/1
```



```
package photoapptest

Album_DoePhotos.test_JaneDoe_viewPhoto_JohnDoe if {
  result := data.photoapp.is_authorized with input as {
    "principal": {
      "type": "PhotoApp::User",
      "id": "JaneDoe"
    },
    "action": "viewPhoto",
    "resource": {
      "type": "PhotoApp::Photo",
      "id": "JohnDoe.jpg"
    },
    "context": {}
  }
  result.decision == "ALLOW"
  result.determiningPolicies["DoePhotos"]
}
```

To help you verify the correctness of your policies, OPA also gives you a framework that you can use to write tests for your policies.

OPA WebAssembly (Wasm)

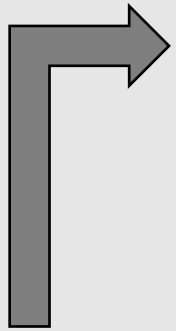
OPA is able to compile Rego policies into executable Wasm modules that can be evaluated with different inputs and external data. This is not running the OPA server in Wasm, nor is this just cross-compiled Golang code.

```
import { loadPolicy } from '@open-policy-agent/opa-wasm';
const policyWasm = readFileSync('policy.wasm');
policy = await loadPolicy(policyWasm);
const data = readFileSync('data.json');
policy.setData(JSON.parse(data.toString()));
```

Source: https://www.openpolicyagent.org/docs/latest/wasm/

```
function isAuthorized(regoinput, decision, policyId) {
  const input = readFileSync(`input/${regoinput}.regoinput.json`);
  const resultSet = policy.evaluate(JSON.parse(input.toString()));
  expect(resultSet).not.toBeNull();
  expect(resultSet.length).toBe(1);
  expect(resultSet[0].result.decision).toEqual(decision);
  expect(resultSet[0].result.determiningPolicies).toContain(policyId);
}

test.each([
  ['JaneDoe-view-JohnDoe', 'ALLOW', 'DoePhotos'],
  ['JorgeSouza-view-Judges', 'ALLOW', 'PhotoJudge'],
])('%s', async (regoinput, decision, policy) => {
  isAuthorized(regoinput, decision, policy);
});
```



```
✓ opa.test.js (2)
  ✓ photoapp test suite (2)
    ✓ JaneDoe-view-JohnDoe
    ✓ JorgeSouza-view-Judges

Test Files  1 passed (1)
     Tests  2 passed (2)
```

OpenFGA™

OpenFGA

- Takes the best ideas from Google's Zanzibar paper for Relationship-Based Access Control (ReBAC), and also solves problems for Role-based Access Control and Attribute-Based Access Control use cases.

- Originally developed by Auth0/Okta, and donated to the Cloud Native Computing Foundation on September 14, 2022, and is currently at the Sandbox level of graduation.

Source: https://openfga.dev/

Zanzibar: Google's Consistent, Global Authorization System

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.

- Allows clients to define arbitrary relations between users and objects
- ACLs are collections of object-user or object-object relations represented as *relation tuples*
- Groups are simply ACLs with membership semantics
- Can implement RBAC policies on top of namespace configuration language

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

What Is A Relationship Tuple?

A relationship tuple consists of:

- a **user** (entity in the system)
- a **relation** (string defined in the type definition)
- an **object** (entity in the system)
- a **condition** (optional truthy outcome)

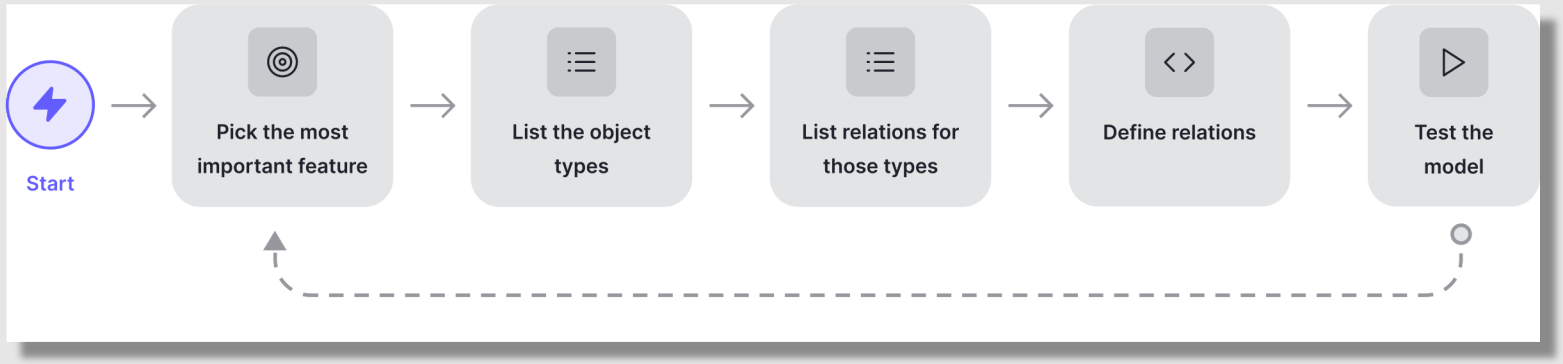
An **authorization model**, together with relationship tuples, determinate whether a relationship exists between a user and an object.

Source: https://openfga.dev/docs/concepts

Define a ReBAC model in OpenFGA

Your feature description should include the objects, users and groups of users participating in the system. Sentences should look like this:

> A user {user} can perform action {action} to/on/in {object types} ... IF {conditions}



Source: https://openfga.dev/docs/modeling/getting-started

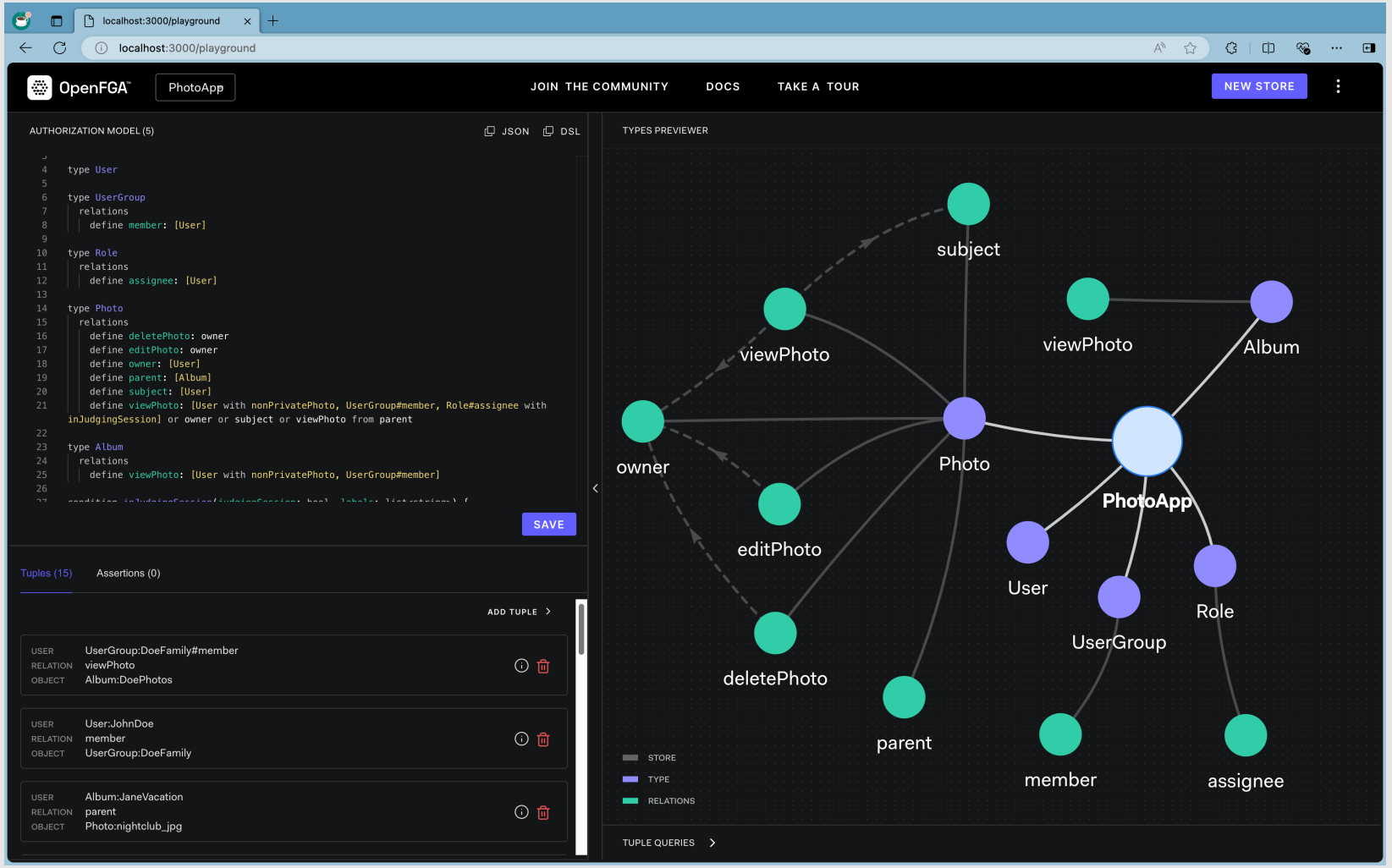
The Configuration Language describes the relations possible for an object of a given type and lists the conditions under which one is related to that object.

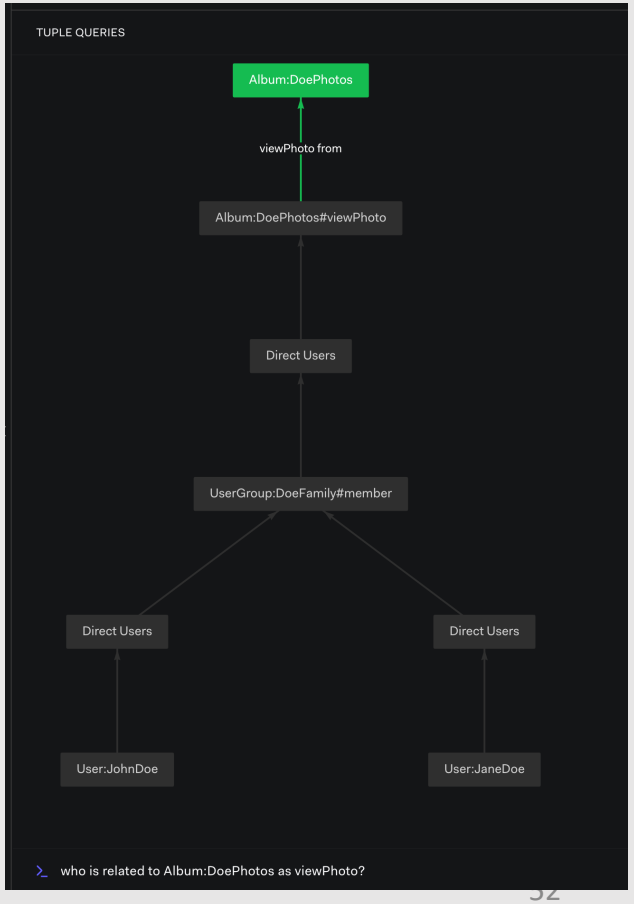
```
model
schema 1.1

type User
type UserGroup
  relations
    define member: [User]
type Role
  relations
    define assignee: [User]
type Photo
  relations
    define parent: [Album]
    define viewPhoto: [User with nonPrivatePhoto, UserGroup#member, Role#assignee with inJudgingSession]
                       or owner or subject or viewPhoto from parent
    define editPhoto: owner
    define deletePhoto: owner
    define owner: [User]
    define subject: [User]
type Album
  relations
    define viewPhoto: [User with nonPrivatePhoto, UserGroup#member]

condition inJudgingSession(judgingSession: bool, labels: list<string>) {
  judgingSession == true && "contest" in labels
}
condition nonPrivatePhoto(labels: list<string>) {
  !("private" in labels)
}
```

localhost:3000/playground





OpenFGA CLI

- A cross-platform CLI to interact with an OpenFGA server
- If a model is provided, the test will run in a built-in OpenFGA instance (you do not need a separate server).

```
fga model validate --file model.fga
{
  "is_valid":true
}
```

```
fga model test --tests store.fga.yaml
# Test Summary #
Tests 12/12 passing
Checks 25/25 passing
```

Source: https://github.com/openfga/cli

A user {UserGroup DoeFamily} can perform action {viewPhoto} to/on/in {Album DoePhotos}

Object-to-Object relationships can be used to indicate that a user's relation with one object can influence their relationship with another object.

```
type Photo
  relations
    define parent: [Album]
    define viewPhoto: [User with nonPrivatePhoto, UserGroup#member] or owner or viewPhoto from parent
    define owner: [User]
type Album
  relations
    define viewPhoto: [User with nonPrivatePhoto, UserGroup#member]
```

```
- user: User:JaneDoe
  relation: member
  object: UserGroup:DoeFamily
- user: Album:DoePhotos
  relation: parent
  object: Photo:JohnDoe_jpg
- user: UserGroup:DoeFamily#member
  relation: viewPhoto
  object: Album:DoePhotos
```

```
- name: JaneDoe view JohnDoe_jpg
  check:
    - user: User:JaneDoe
      object: Photo:JohnDoe_jpg
      assertions:
        viewPhoto: true
        owner: false
```

A user {User} can perform action {viewPhoto} to/on/in {Photo}
... IF {subject include User}

```
type Photo
  relations
    define parent: [Album]
    define viewPhoto: [User with nonPrivatePhoto, UserGroup#member] or owner or subject
    define owner: [User]
    define subject: [User]
```

```
- user: User:JohnDoe
  relation: owner
  object: Photo:Judges_jpg
- user: User:JorgeSouza
  relation: subject
  object: Photo:Judges_jpg
```

```
- name: JorgeSouza view Judges_jpg
  check:
  - user: User:JorgeSouza
    object: Photo:Judges_jpg
    assertions:
      viewPhoto: true
      subject: true
```

A user {User JohnDoe} can perform action {viewPhoto} to/on/in {Album JaneVacation}
... IF {labels doesn't include "private"}

Conditions allow you to model more complex authorization modeling scenarios involving attributes and can be used to represent some Attribute-based Access Control (ABAC) policies.

```
condition nonPrivatePhoto(labels: list<string>)
{
  !("private" in labels)
}
```

```
- user: User:JohnDoe
  relation: viewPhoto
  object: Album:JaneVacation
  condition:
    name: nonPrivatePhoto
```

```
- name: JohnDoe view sunset_jpg
  check:
    - user: User:JohnDoe
      object: Photo:sunset_jpg
      context:
        labels: ["contest"]
      assertions:
        viewPhoto: true
        owner: false
- name: JohnDoe view nightclub_jpg
  check:
    - user: User:JohnDoe
      object: Photo:nightclub_jpg
      context:
        labels: ["private"]
      assertions:
        viewPhoto: false
        owner: false
```

Call to Action

- Bookmark / Star the GitHub repo
  - https://github.com/hakanson/that2024
- Take an authorization problem and model using one of these open-source projects

Thank You

Kevin Hakanson (kevinhakanson.com)