Build verifiable and effective application authorization in 40 minutes

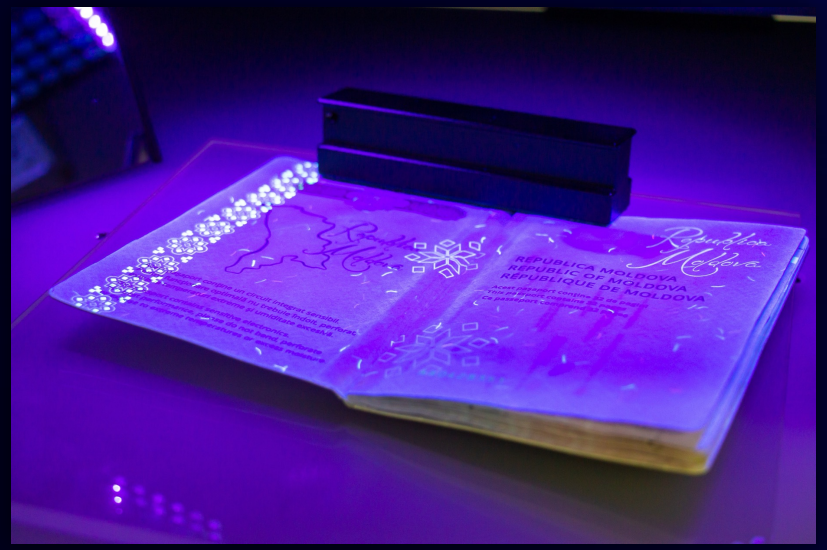
Wojciech Gawronski

he/him
Senior Developer Advocate (CEE)
Amazon Web Services

However, we need to start with something *obvious* . . .

Authentication (*auth-n*) vs. Authorization (*auth-z*)

Authentication is about
verifying who a user is

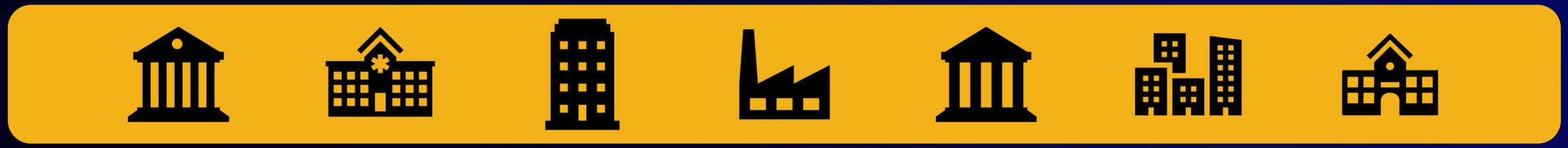


Authorization is about
verifying what an authenticated user is allowed to do

Permissions

Every application needs them

Permissions are sets of rules that describe what each user of the application is permitted to do



Permissions: Why are they so hard?

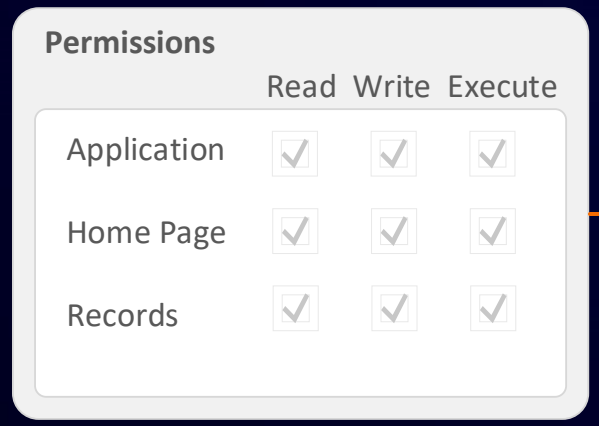
```
def get_book(request):
    log("Handling book request " + request.id)
    book = db.query(request.bookId)

    return {
        ‘product_id': book.id,
        'title': book.title,
        ...
    }
```

Permissions: Why are they so hard?

```
def get_book(request):
    if not db.query(request.bookId).isPublic:
        if not db.query(request.user).admin:
            if db.query(request.bookId).owner != request.user:
                return 'AccessDenied'

        if not request.multiFactorAuth:
            return 'AccessDenied'

    log("Handling book request " + request.id)
    book = db.query(request.bookId)

    return {
        ‘product_id': book.id,
        ...
```

Usual evolution of a self-built permissions system

Custom management UI



Custom database schema

Principal	Group	Resource	Role
user1	group1	MyRes	null
user2	group1	2ndRes	admin
user3	null	MyRes	audit
user4	groupX	*	user

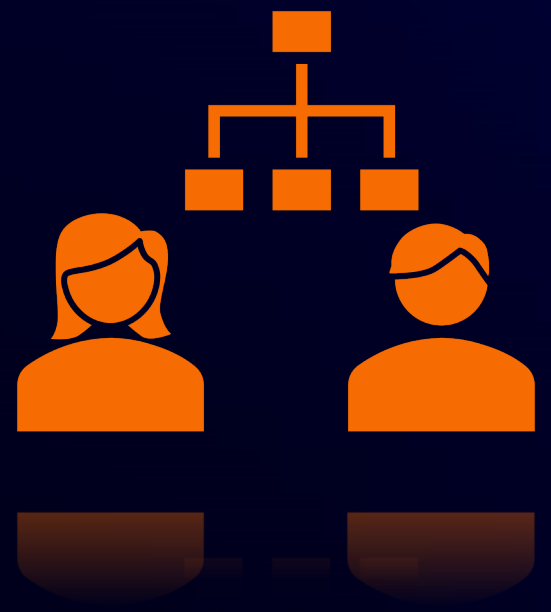
Before you know it, you've **built an entire application** just to support permissions

This is:

- Time consuming
- Difficult to do right
- Might not scale
- Lacks governance

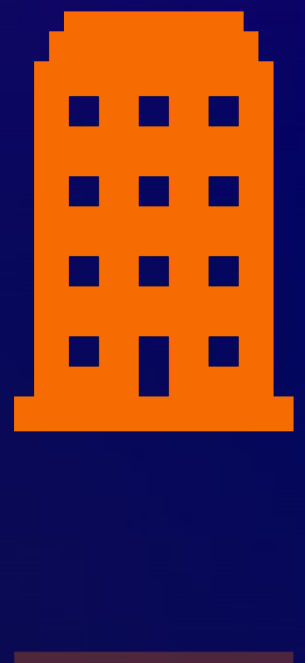
Authorization models

Role-based access control (**RBAC**)

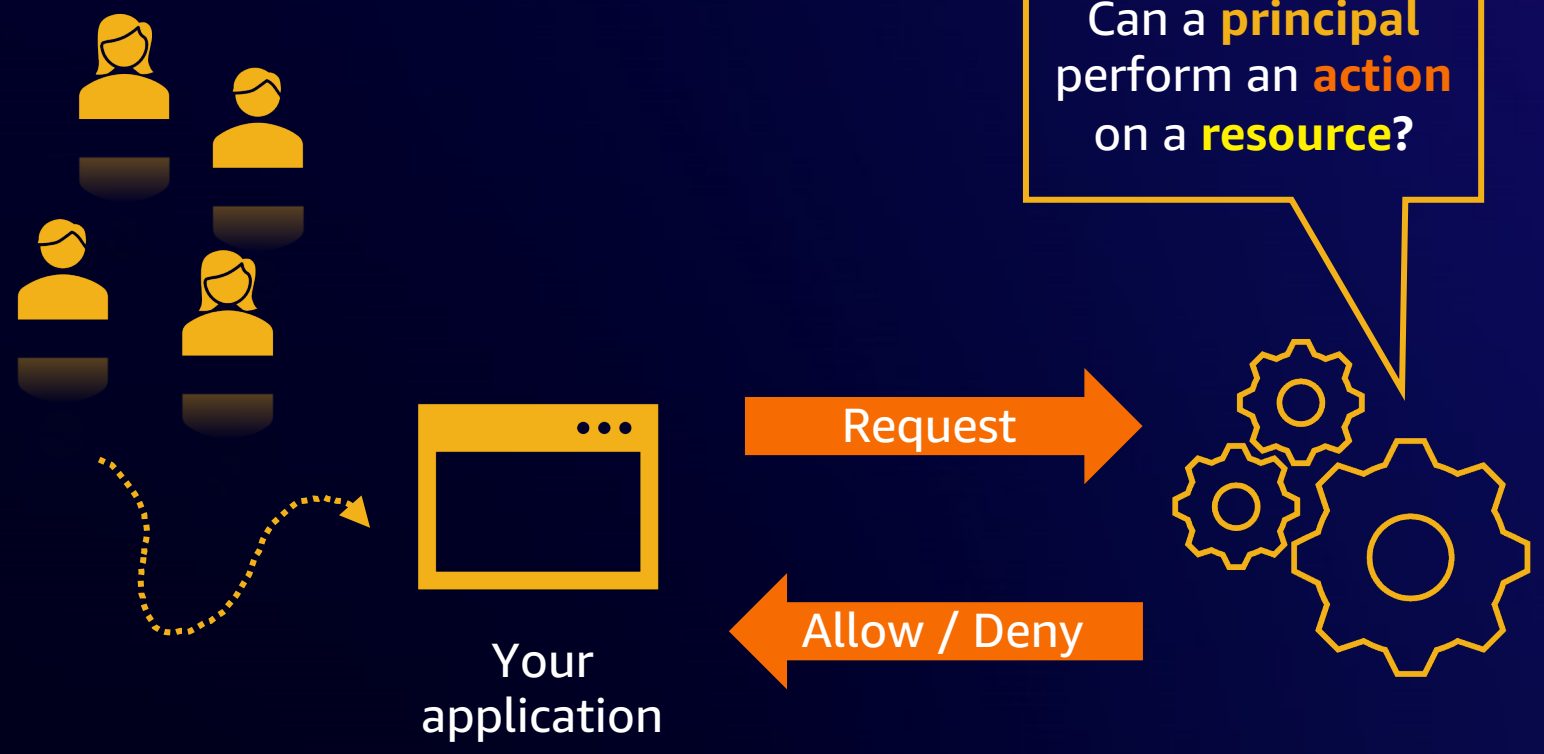


Attribute-based access control (**ABAC**)

Other (simpler, mix of both, etc.)



What is policy-based authorization?



Policies

```
permit (
  principal,
  action == Action::"Check Out",
  resource == Location::"Hotel California"
);

forbid (
  principal,
  action == Action::"Leave",
  resource == Location::"Hotel California"
);
```

Authorization as a service

Protect your applications

Access enforcement through Amazon API Gateway and AWS AppSync

Enforce

Policy enforcement point

Verified permissions

Centrally manage application policies and provide access decisions in milliseconds

Define

Policy administration point

Decide

Policy decision point

Amazon **Verified Permissions**

Scalable permissions management and fine-grained authorization for applications you build

GA June 13, 2023

Accelerate app development

Protect app data and resources

Simplify compliance audits at scale

Build apps with continuous authorization

Introducing Amazon Verified Permissions

BUILD FASTER, GO FURTHER

Plug fine-grained authorization into your application

Features of Amazon Verified Permissions

Policy administration

Runtime authorization

Audit fine-grained authorization

Application-managed access in custom apps

Define permissions as easy-to-understand policies

Fine-grained

Access for individual resources and users

Performant

Low-latency access decisions

Scalable

Policy structure is amenable to machine parsing

Expressive yet readable

User permissions expressed as easy-to-understand policies

Policy description

Describe the purpose of this policy and the permissions it grants.

Enable owners and managers to maintain customer account data

Maximum length 150 bytes.

Policy body

```
permit (
    principal in Usergroup::"SalesTeam",
    action in Action::"Maintain",
    resource in AccountData::"Customers")
when {
    principal == resource.Owner ||
    principal.Role.contains("Manager")
};
```

Role based

Attribute based

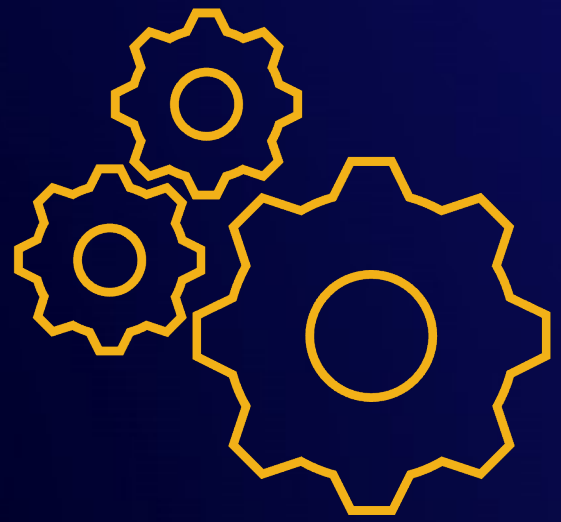
Introducing Cedar

CEDAR

Cedar policy language

```
permit (
  principal,
  action == Action::"Check Out",
  resource == Location::"Hotel California"
);

forbid (
  principal,
  action == Action::"Leave",
  resource == Location::"Hotel California"
);
```

Policy evaluation and authorization engine



Easy to analyze

Source: cedarpolicy.com

Cedar: Policy syntax and semantics

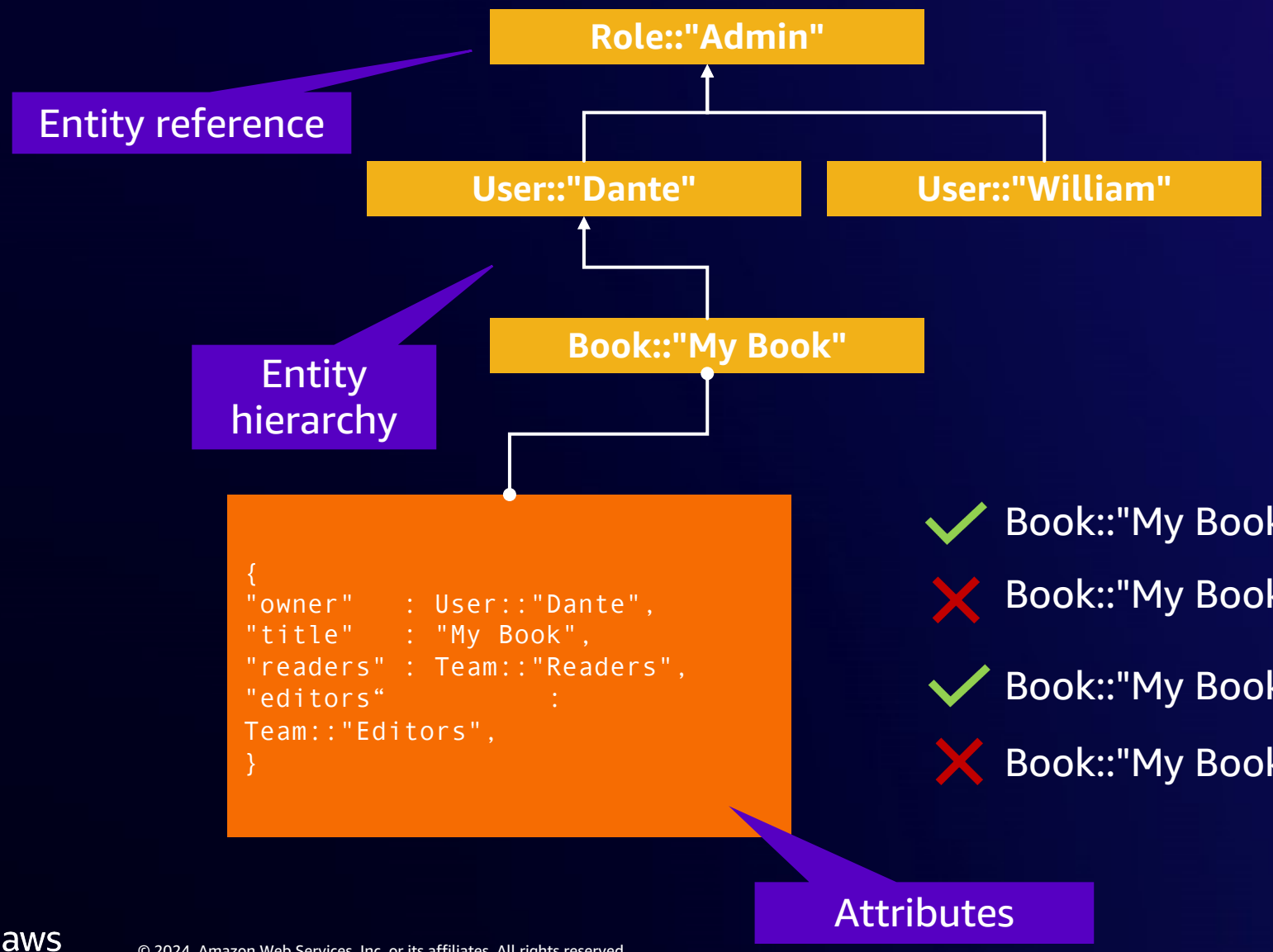
```
permit (
  principal in Bookstore::Role::"Admin",
  action in [ Bookstore::Action::"View" ],
  resource == Bookstore::Book::"ID"
)
```

head

```
when {
  context.region == "US"
};
```

condition

Data model and semantics of Cedar



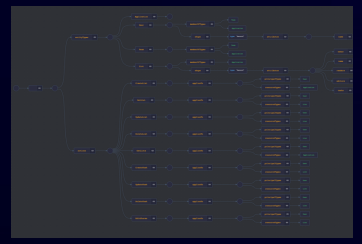
- ✓ Book::"My Book" **in** User::"Dante"
- ✗ Book::"My Book" **in** User::"William"
- ✓ Book::"My Book" **has** year
- ✗ Book::"My Book" **has** chapters

Authorization engine

Context

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Age: 521648
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Fri, 06 Mar 2020 17:36:11 GMT
Etag: "3147526947+gzip"
Expires: Fri, 13 Mar 2020 17:36:11 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (dcb/7EC9)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 648
```

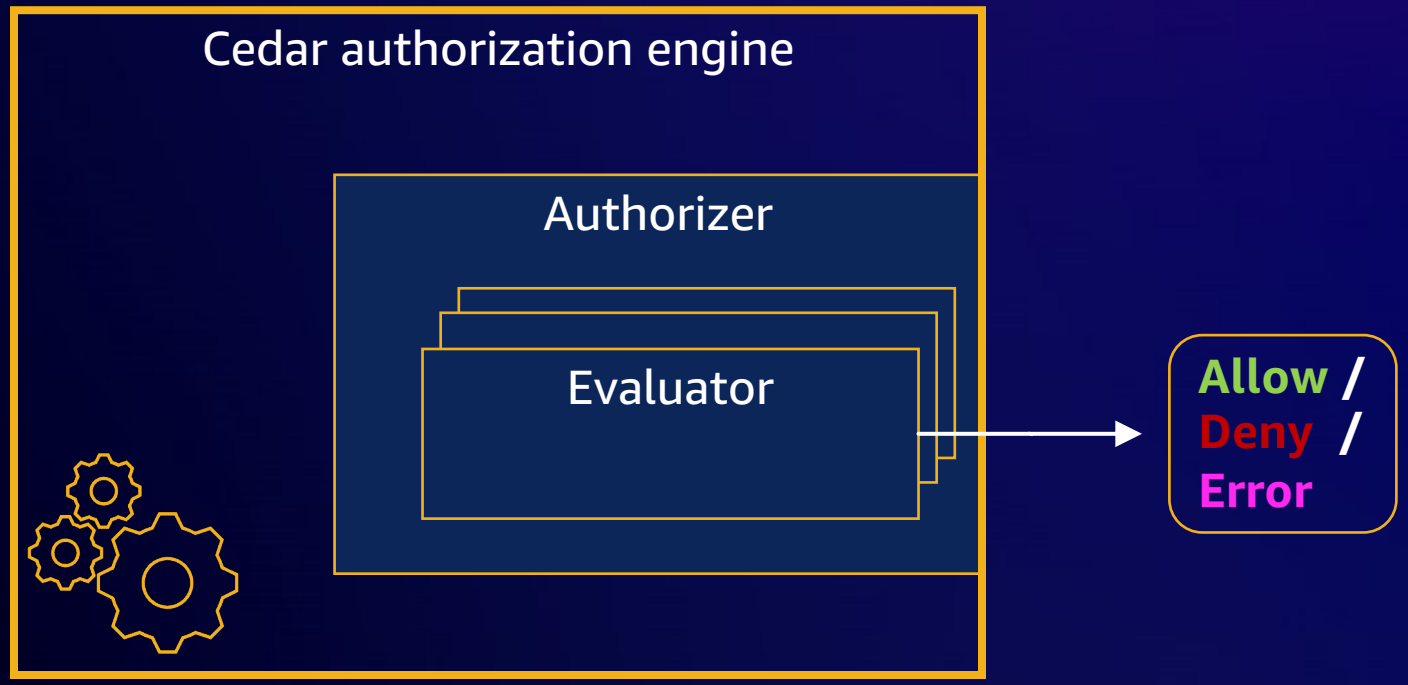
Application entity data



Cedar policy

```
permit (
  principal,
  action == Action::"Check Out",
  resource == …
);

forbid (
  principal,
  action == Action::"Leave",
  resource == …
);
```

Input: entity data, Cedar policy, context



Policy administration

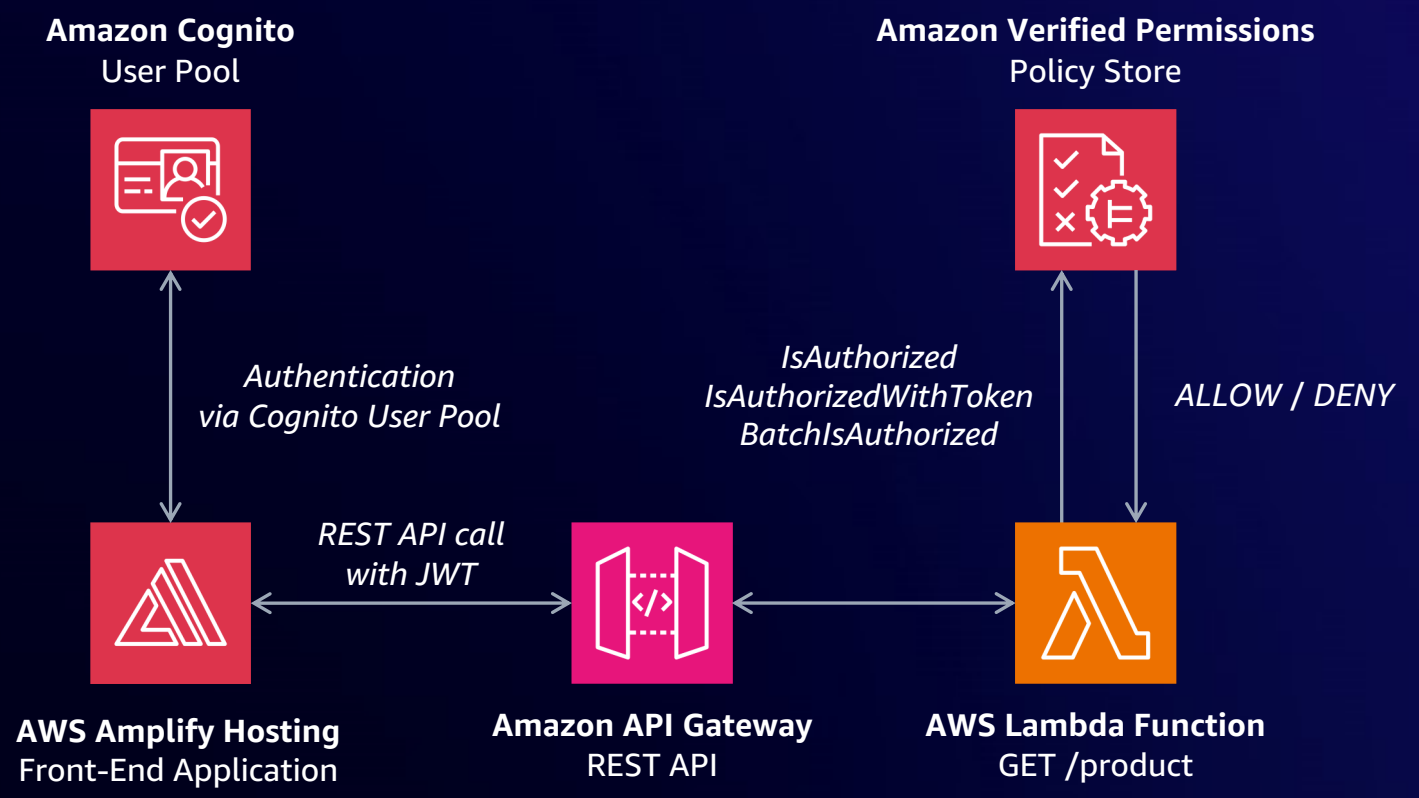
Generate an **application schema** based on the live application

Import **user schema** from the identity provider

Delegated administration enables service scale

Integrated **policy simulator** to test policy logic

Let’s introduce our use case: Bookstore App



github.com/build-on-aws/bookstore-demo-app-with-authz

What’s next?
Summary and recap

Best practices and recommendations

CEDAR

SCHEMA

Schema format

Schema grammar

BEST PRACTICES

Naming conventions

Normalize data input

Using the context

Populate the policy scope

Meta-permissions

Avoid mutable identifiers

Using role-based access control

Security requirement: Normalize input data prior to invoking the authorization APIs

The Cedar policy language omits some well-known operators, including those used to format data and to manipulate and transform strings and lists. This omission is intentional. One reason why is that these operators disrupt the ability to apply automated reasoning techniques to Cedar policy statements. Another reason that Cedar does not provide operators is that Cedar is designed to support situations where policy authors can reside outside the service team, or even be external customers. To provide a safe, intuitive policy authoring experience for these audiences, each individual policy author should not be required to discover and apply appropriate formatting rules.

As a result, application owners should format data prior to passing it into the authorization APIs. For example, instead of passing the following data in the context record:

```
{
  "url": "https://example.com/path/to/page?name=alice&amp;color=red"
}
```

docs.cedarpolicy.com

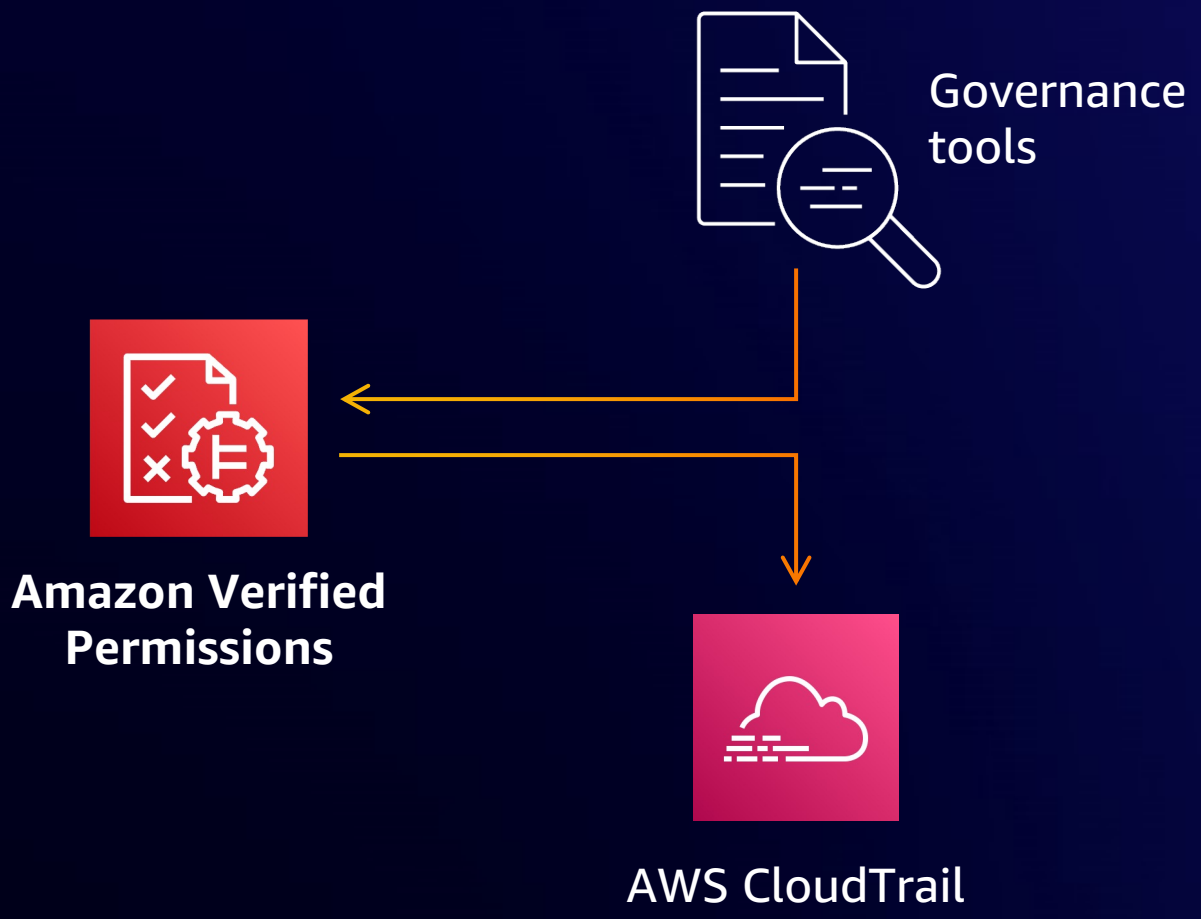
Considerations

Lock-in

Auditability

Pricing

Auditability



Audit data **streamed** to AWS CloudTrail

Provide **auditing tools** with APIs to analyze and maintain permissions

External auditors **evaluate policies** for completeness and correctness

Set **alarms and notifications** based on unusual activity

Pricing

Flexible pricing model, where you **pay only for what you are using**

Tiered pricing for **authorization requests per month** and **policy management requests**

It is also possible to use **Cedar** in a standalone mode as it is open sourced, *however . . .*

. . . when comparing **Amazon Verified Permissions** with custom-built alternative on top of open-sourced version, it is important to compare the **total cost of ownership (TCO)**

Pricing: Fully Managed vs. Self-Hosted

Fully Managed Service

Amazon Verified Permissions

Self-Hosted

Cedar (Authorization Engine)

Amazon Verified Permissions for Zero Trust

01
Centrally create and maintain policies

02
Manage fine-grained permissions across applications

03
Authorize end-user actions based on roles/attributes

04
Audit permissions at scale

Repository: Bookstore App

github.com/build-on-aws/bookstore-demo-app-with-authz

Learning tool: avp-cli

- Tool for easing out the start with **Amazon Verified Permissions** (and **Cedar**)
- Supports all actions available in **Amazon Verified Permissions**
- Predefined blueprints with tests for different authorization scenarios that can be deployed to **AWS**

github.com/pigius/avp-cli

Workshop: Verified Permissions in Action

Self-guided and free workshop that presents the whole lifecycle of adding **Amazon Verified Permissions** to an existing application (**TinyTodo**)

Verified Permissions in Action (*Workshop*)

Learn more about Cedar and AVP!

Series of blog posts
from Daniel on dev.to

How we built Cedar with automated reasoning and differential testing (*Amazon Science*)

GitHub organization
for Cedar policy languge

Other AWS re:Invent 2023 sessions
about Amazon Verified Permissions

Stay connected with AWS!

Community.AWS

AWS Skill Builder

@awsdevelopers

@aws-developers

@awsdevelopers

collectives/aws

Thank you!

Wojciech Gawroński

wojgaw@amazon.com

in/afronski (LinkedIn)

@afronski (Twitter)

@afronsky (Instagram)

afronski (GitHub)

awsmaniac.com