Fine-grained authorization for applications with Amazon Verified Permissions

Julian Lovelock (He/Him)
Product Manager
AWS

Abhishek Panday (He/Him)
Product Manager
AWS

Agenda

What is it and why did we build it

What our customers told us

How can you get started

Permissions

every application needs them

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

Why are permissions hard?

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

    return {
        'id': book.id,
        'title': book.title,
        'rating': book.rating,
        'numReviews': book.numReviews,
    }
```

Why are permissions hard?

```
def get_book(request):
    if db.query(request.bookId).owner != request.user:
        return 'AccessDenied'

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

    return {
        'id': book.id,
        'title': book.title,
        'rating': book.rating,
        'numReviews': book.numReviews,
    }
```

Why are permissions hard?

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

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

    return {
        'id': book.id,
        'title': book.title,
        'rating': book.rating,
        'numReviews': book.numReviews,
    }
```

Why are permissions hard?

```
def get_book(request):
    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 {
        'id': book.id,
        'title': book.title,
        ...
```

Why are permissions 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 {
        'id': book.id,
        ...
```

How does Amazon Verified Permissions help?

```python
def get_book(request):
    if not client.is_authorized(...):
        return 'AccessDenied'

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

    return {
        'id': book.id,
        'title': book.title,
        'rating': book.rating,
        'numReviews': book.numReviews,
    }
```

```
permit (principal,
        action == Action::"GetBook",
        resource)
when { resource.owner == principal };
```

```
permit (principal,
        action == Action::"GetBook",
        resource)
when { principal.admin };
```

```
forbid (principal,
        action == Action::"GetBook",
        resource)
unless { principal.multiFactorAuth };
```

Agenda

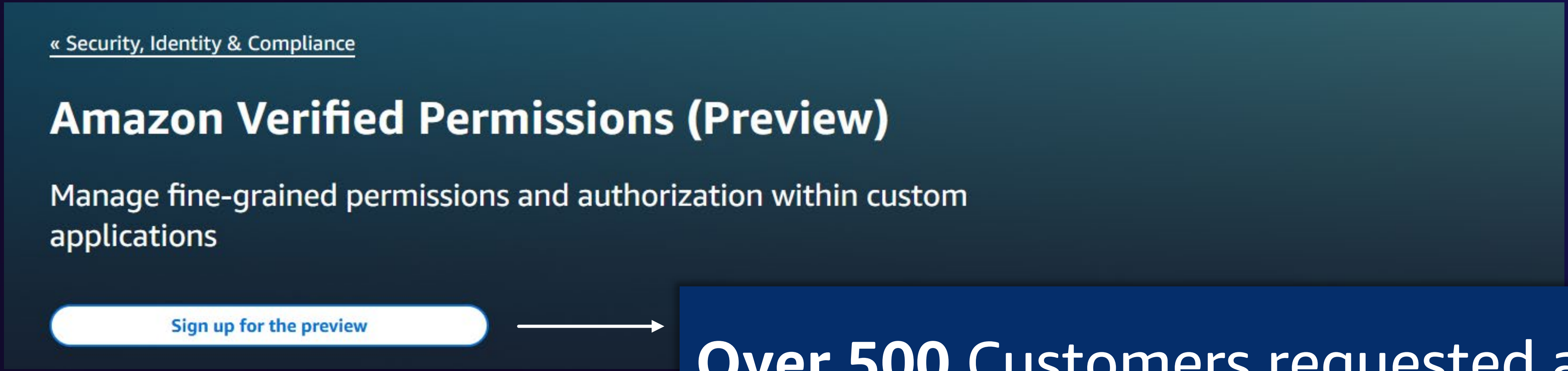
What is it and why did we build it

What our customers told us

How can you get started

Gated Preview

WHAT DID WE LEARN


« Security, Identity & Compliance

Amazon Verified Permissions (Preview)

Manage fine-grained permissions and authorization within custom applications

Over 500 Customers requested access
On-boarded **90** customers
You told us **4** things

#1 – One size fits all

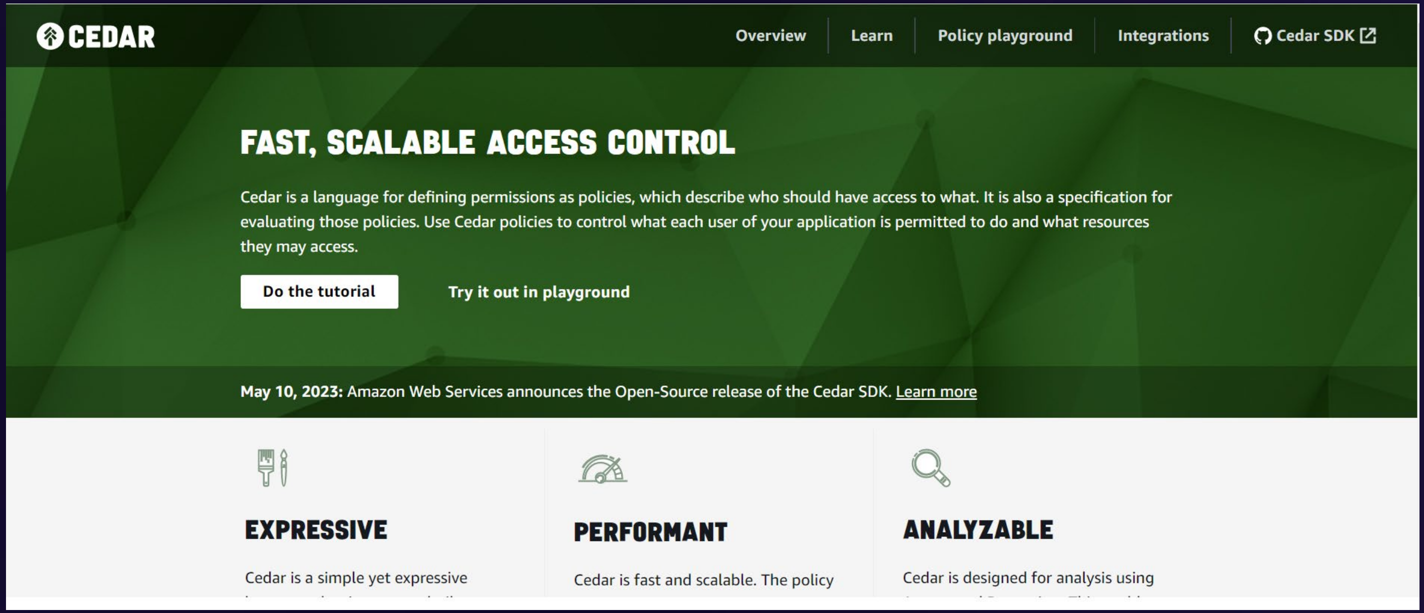
KIND OF, SORT OF

Wide range of

- applications
- use cases
- architectures

Cedar policy language provided the flexibility for these different <u>authorization models</u>

Permissions are expressed in an open source policy language called Cedar



FAST, SCALABLE ACCESS CONTROL

Cedar is a language for defining permissions as policies, which describe who should have access to what. It is also a specification for evaluating those policies. Use Cedar policies to control what each user of your application is permitted to do and what resources they may access.

May 10, 2023: Amazon Web Services announces the Open-Source release of the Cedar SDK. Learn more

EXPRESSIVE

Cedar is a simple yet expressive

PERFORMANT

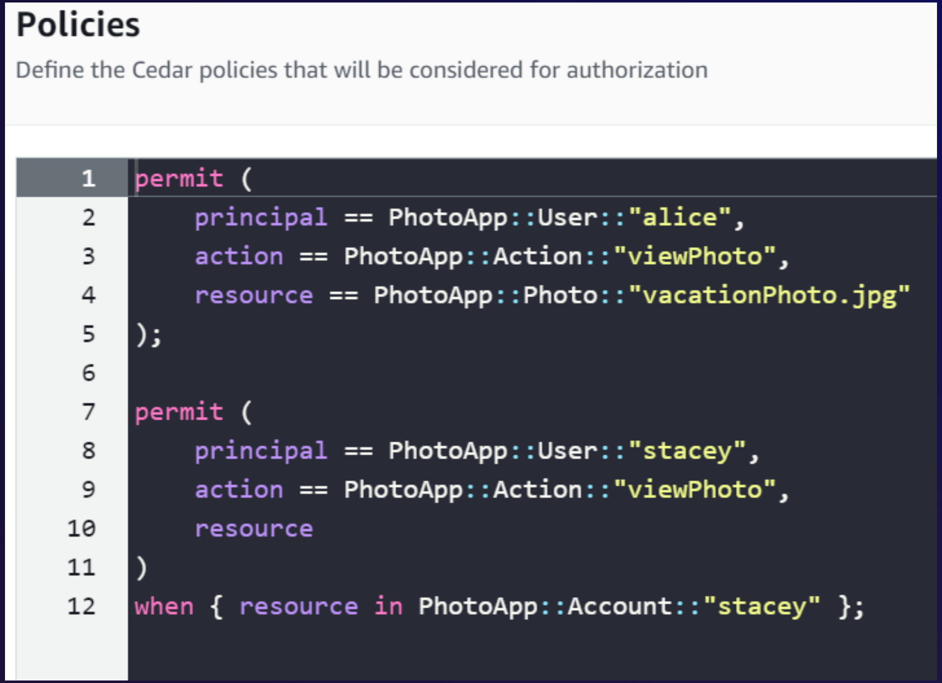
Cedar is fast and scalable. The policy

ANALYZABLE

Cedar is designed for analysis using

Cedar was developed by AWS and open sourced on May 10th, 2023

[Cedar Language (cedarpolicy.com)](https://www.cedarpolicy.com)



Policies

Define the Cedar policies that will be considered for authorization

```
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" };
```

Internal service access

In-house build applications for employees accessing corporate data

Characteristics

- Org hierarchy / User roles
- Many application dev teams
- Zero Trust driver

Challenges

- Role explosion

Customer Facing

Applications to sell, deliver, support a company’s products and services

Characteristics

- Relationship based access
- UX is everything

Challenges

- End-users manage permissions

Software Vendor SaaS

The software is the product

Characteristics

- Workforce oriented
- Multi-tenant
- Many Identity Providers

Challenges

- Custom roles

#2 – Personas

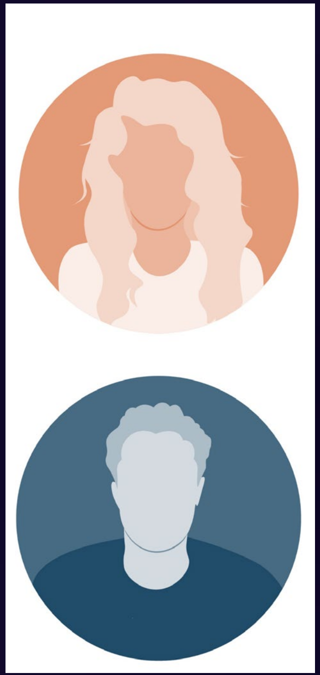
WHO CARES ABOUT WHAT

Application Developer

Access administrator

Compliance manager

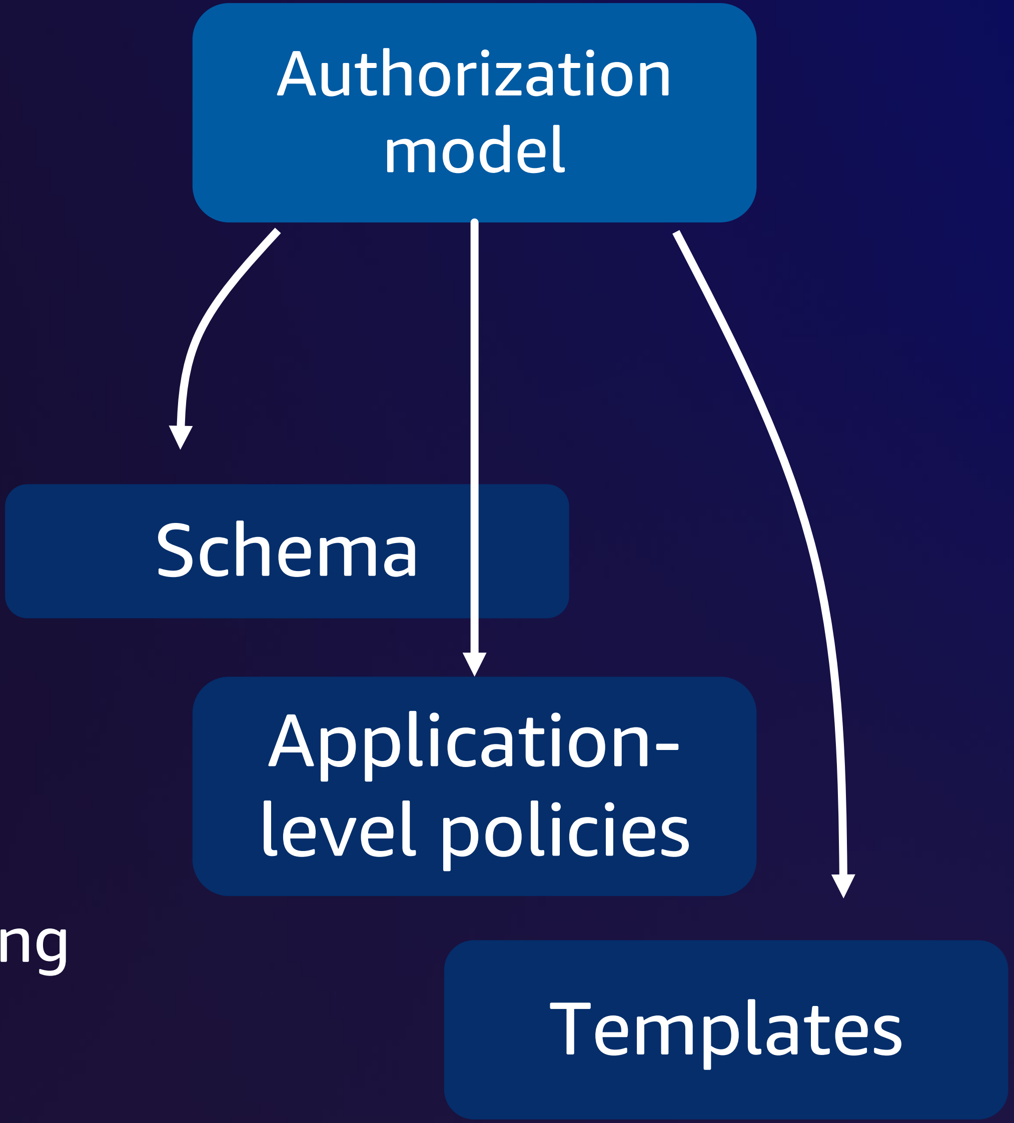
Application developer



Cares about

1. Building faster
2. Features
3. AppSec review

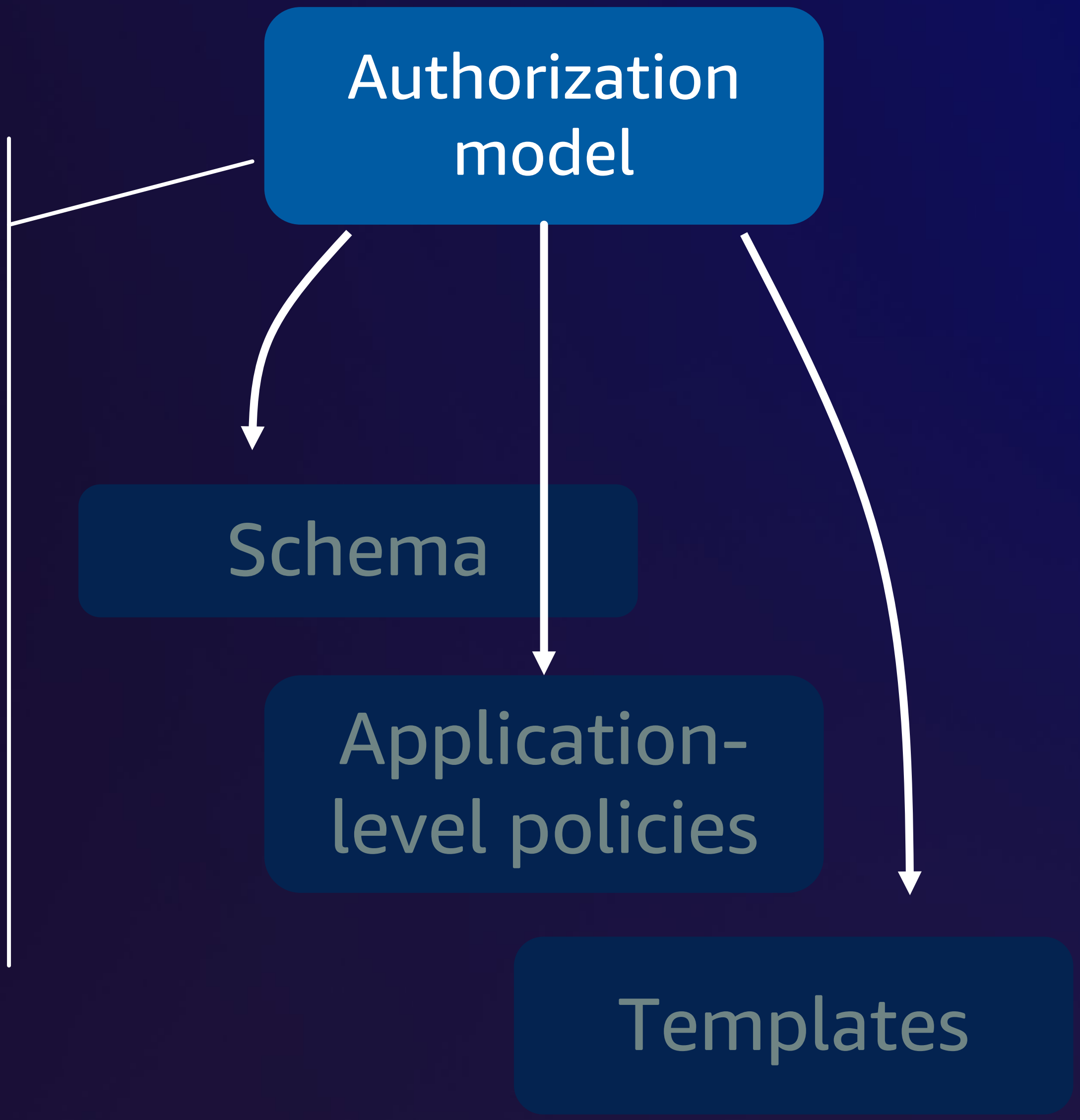
"We've identified opportunities to accelerate application development by 20%, by externalizing authorization “.



Pet Store - Authorization Model

NARRATIVE THAT DESCRIBES HOW PERMISSIONS WORK

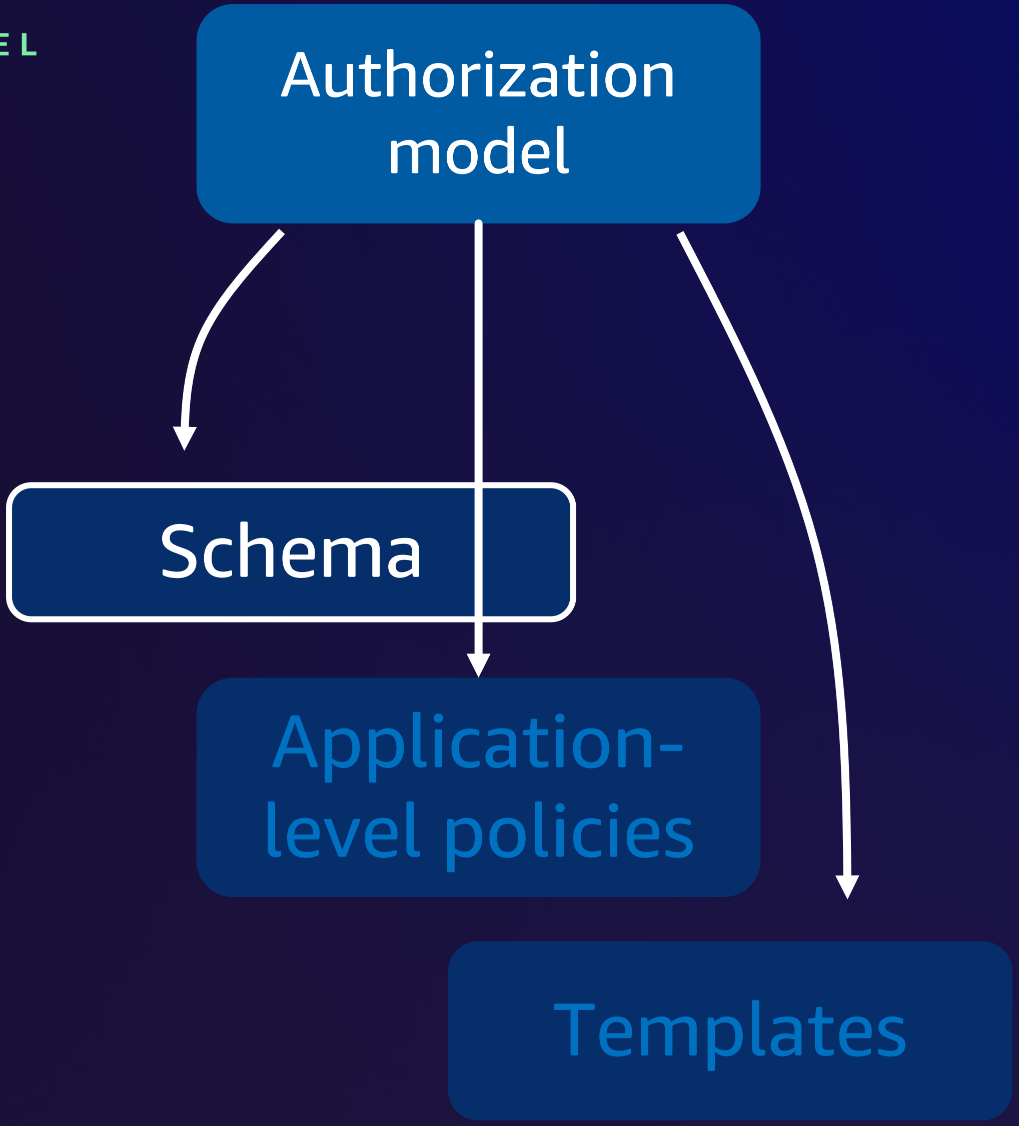
Anyone can register on the site, thereby becoming a Customer. Customers are allowed to add pets for sale, search pets, …



Pet Store - Schema

DEFINES THE TYPES OF ENTITIES DESCRIBED IN YOUR MODEL

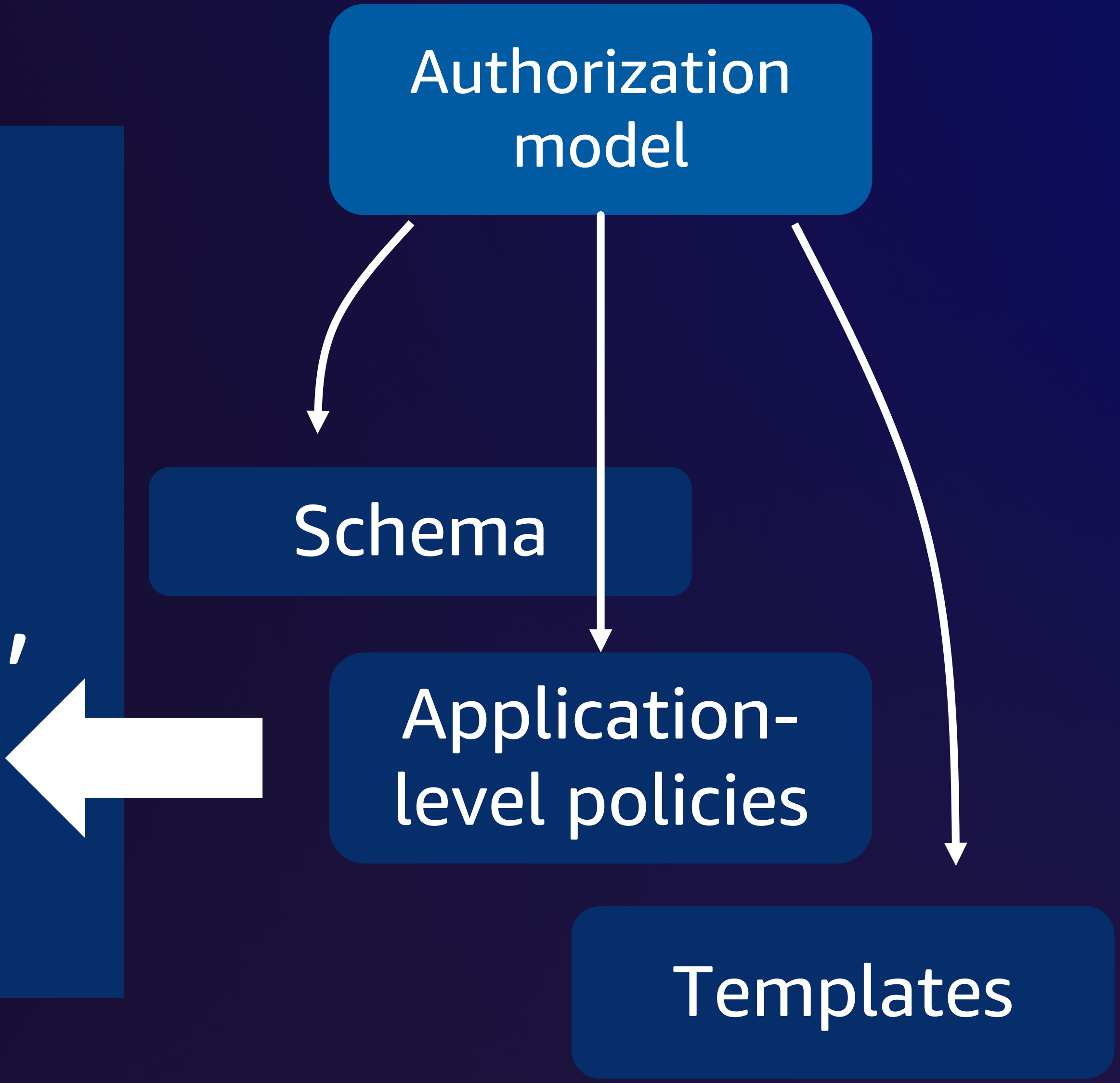
```
"PetStore": {
"actions": {
"SearchPets": {
"appliesTo": {
"resourceTypes": [
"Pets" …
```



Application level policies

BASE LEVEL POLICIES FOR YOUR APPLICATION

```
permit ( principal in
    Users::"Customers",
  action ==
    Action::"SearchPets",
  resource in
    Pets::"All"
```



#3 – The importance of fitting in

ESTABLISHED IDENTITY ECOSYSTEM

- Identity Providers (IdP)
- Identity Governance Administration (IGA)
- Orchestration
- Privileged Access Managers (PAM)
- API Gateways

Launch Partners

AMAZON VERIFIED PERMISSIONS LAUNCH PARTNERS

CYBERARK®
The Identity Security Company

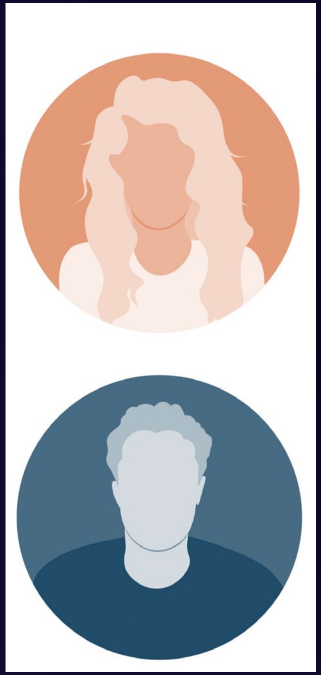
Ping Identity®

STRATA
Identity Orchestration

transmit
security

#4 – Give me a working application

NO ONE WANT TO READ THE MANUAL



Demo – Sample Pet Store

How can you get started

How you can get started

```
def get_book(request):
    if not client.is_authorized(...):
        return 'AccessDenied'

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

    return {
        'id': book.id,
        'title': book.title,
        'rating': book.rating,
        'numReviews': book.numReviews,
    }
```

```
permit (principal,
        action == Action::"GetBook",
        resource)
when { resource.owner == principal };
```

```
permit (principal,
        action == Action::"GetBook",
        resource)
when { principal.admin };
```

```
forbid (principal,
        action == Action::"GetBook",
        resource)
unless { principal.multiFactorAuth };
```

How you can get started – Step 1

```
def get_book(request):
    client.is_authorized(...)

    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)
    ...
```

CloudTrail log of
who is accessing *what*

How you can get started – Step 2

```
def get_book(request):
    shadow_result = client.is_authorized(...)
    current_result = current_access_check(...)

    if current_result != shadow_result:
        log('Access control mismatch: [...]')

    if current_result == 'AccessDenied':
        return current_result

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

    return {
        ...
```

How you can get started – Step 3

```
def get_book(request):
    if not client.is_authorized(...):
        return 'AccessDenied'

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

    return {
        'id': book.id,
        'title': book.title,
        'rating': book.rating,
        'numReviews': book.numReviews,
    }
```

How you can get started

1. Call Verified Permissions and discard the result
2. Call Verified Permissions and compare the result
3. Call Verified Permissions and enforce the result

Thank you

Follow this link to download our sample petstore app from a public Github rep