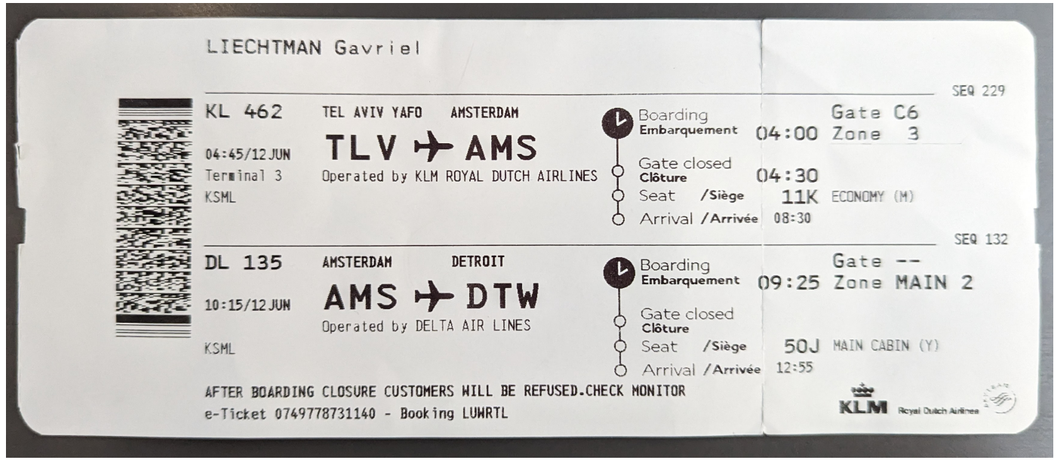
Building Authorization with Python: Dos and Don’ts

Gabriel L. Manor @ PyCon Israel 2023

Permit.io

@gemanor

Find the Difference



Permit.io

@gemanor

	Passport	Flight Ticket
Form of	Identity	Authority
Purpose	Verifies identity	Grants access to a flight
Scope	Global	Flight-specific
Issued by	Government	Airline desk, app, website, etc.
Information	Name, photo, birthdate, etc.	Name, flight number, seat, gate, etc.
Validity	Multiple years	One flight
Used	Once per flight	Multiple times per flight
Uniqueness	Unique to an individual	Unique to a flight and passenger
Changeable	No	Yes
Permissions	One (to travel)	Multiple (to board, to check bags, etc.)
Revocation granularity	All at once	One permission at a time
Transferable	No	Yes

	Authentication	Authorization
Purpose	Verifies user identity	Determines user permissions
Scope	Applies to all users	Specific to each user's role or status
Issued by	Identity provider	Any kind of data
Information	Username, social identity, biometrics, etc.	User roles, permissions, policy, external data, etc.
Validity	Until credentials change or are revoked	Per permissions check
Used	Once per session (typically)	Multiple times per session
Uniqueness	Unique to an individual user	Unique to a principal, action and resource
Changeable	Require session revocation	Yes (permissions can be updated, etc.)
Permissions	One (to access the system)	Multiple (to read, write, update, delete, etc.)
Revocation granularity	All at once (user is denied access)	One permission at a time
Transferable	No (credentials should not be shared)	Yes (policy can be applied to other users)

Authentication Advanced Features

Authentication Advanced Features

- Multi-factor authentication

Authentication Advanced Features

- Multi-factor authentication
- Single sign-on / Social login / Passwordless

Authentication Advanced Features

- Multi-factor authentication
- Single sign-on / Social login / Passwordless
- User / account management

Authentication Advanced Features

- Multi-factor authentication
- Single sign-on / Social login / Passwordless
- User / account management
- Session management

Authentication Advanced Features

- Multi-factor authentication
- Single sign-on / Social login / Passwordless
- User / account management
- Session management
- User registration / UI flows / customizations

Authentication Advanced Features

- Multi-factor authentication
- Single sign-on / Social login / Passwordless
- User / account management
- Session management
- User registration / UI flows / customizations
- Account verification / recovery

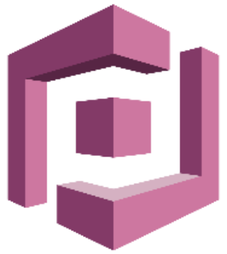
Authentication Advanced Features

- Multi-factor authentication
- Single sign-on / Social login / Passwordless
- User / account management
- Session management
- User registration / UI flows / customizations
- Account verification / recovery
- Audit / reporting / analytics / compliance

Authentication Advanced Features

- Multi-factor authentication
- Single sign-on / Social login / Passwordless
- User / account management
- Session management
- User registration / UI flows / customizations
- Account verification / recovery
- Audit / reporting / analytics / compliance
- Third party integrations

Auth0



clerk

STYTCH

frontegg

SuperTokens

Permit.io

@gemanor

Permit.io

@gemanor

Gabriel L. Manor

Director of DevRel @ Permit.io

Not an ethical hacker, zero awards winner, dark mode hater.

```
def delete_user(user_id):
    user = User.get(user_id)
    if user.role == 'admin':
        user.delete()
```

Permit.io

@gemanor

```python
# Middleware
def roles_required():
    ...
            if user.role == 'admin':
                return func(*args, **kwargs)
            else:
                raise Exception('User is not admin')
    ...

@roles_required('admin')
def delete_user(user_id):
    user = User.get(user_id)
    user.delete()
```

Permit.io

@gemanor

```python
# Middleware
def roles_required():
    ...
            if user.role == 'admin':
                return func(*args, **kwargs)
            else:
                raise Exception('User is not admin')
    ...
def permissions_required():
    ...
            if permissions == 'delete_user':
                return func(*args, **kwargs)
            else:
                raise Exception('User is not admin')
    ...
```

```python
# Business logic
@roles_required('admin')
@permissions_required('delete_user')
def delete_user(user_id):
    user = User.get(user_id)
    user.delete()
```

Permit.io

@gemanor

```python
# Middleware
def roles_required():
    ...
            if user.role == 'admin':
                return func(*args, **kwargs)
            else:
                raise Exception('User is not admin')
    ...

# Business logic
@roles_required('admin')
def enable_workflow(user_id):
    user = User.get(user_id)
    paid_tier = billing_service.get_user_tier(user_id) == 'paid'
    if not paid_tier:
        raise Exception('User is not on paid tier')
```

Permit.io

@gemanor

```
@roles_required('admin')
@permissions_required('enable_workflow')
def enable_workflow():
    user = User.get(user_id)
    step1 = workflow.run()
    step2 = workflow.run()
    if user.sms_enabled:
        send_sms_to_list(user.phone_number, 'Workflow is enabled')
```

Permit.io

@gemanor

Staging

```
@roles_required('admin')
@permissions_required('enable_workflow')
def enable_workflow():
    user = User.get(user_id)
    step1 = workflow.run()
```

Production

```
@roles_required('superadmin')
@permissions_required('enable_workflow')
def enable_workflow():
    user = User.get(user_id)
    step1 = workflow.run()
```

Permit.io

@gemanor

Django

```python
@permission_required('app_name.can_edit')
def my_view(request):
    # Your view logic here
    ...
```

Flask

```python
app = Flask(__name__)
login_manager = LoginManager(app)

class User(UserMixin):
    def __init__(self, id, role):
        self.id = id
        self.role = role

@login_manager.user_loader
def load_user(user_id):
    return User(user_id, 'admin')

@app.route('/admin')
@login_required
def admin():
    if current_user.role != 'admin':
        abort(403)
    ...
```

Permit.io

@gemanor

```python
if user.role == 'admin':
    if user.tier == 'paid':
        if user.sms_enabled:
            if user.phone_number:
                send_sms_to_list(user.phone_number, 'Workflow is enabled')
```

Authorization Best Practices

Declarative

Generic

Unified

Agnostic

Decoupled

Easy to audit

Permit.io

@gemanor

#1 Model

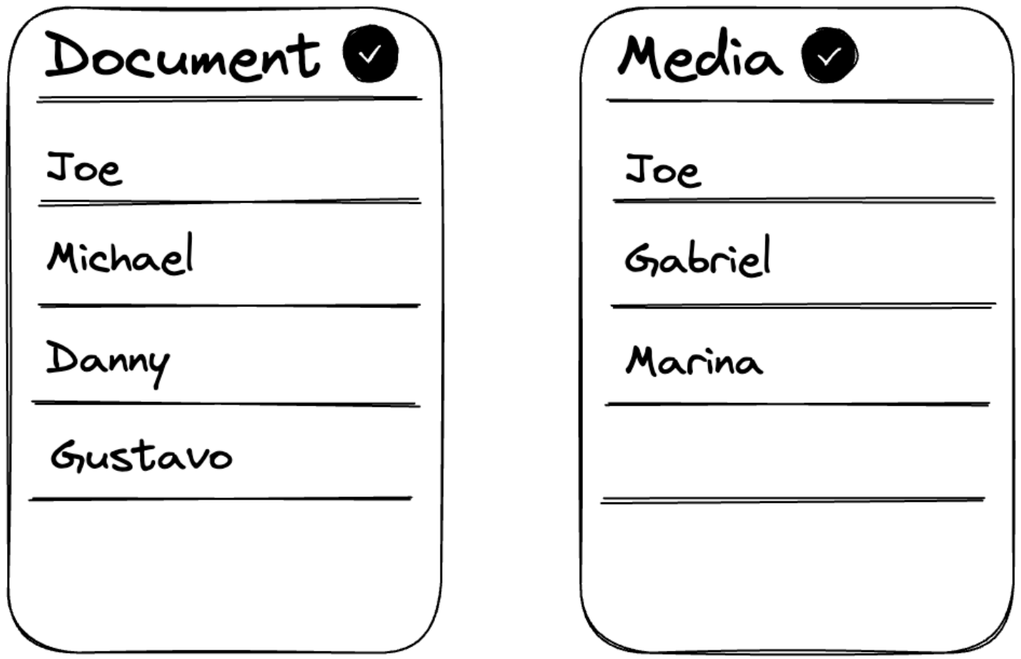
ACL - Access Control List

RBAC - Role-Based Access Control

ABAC - Attribute-Based Access Control

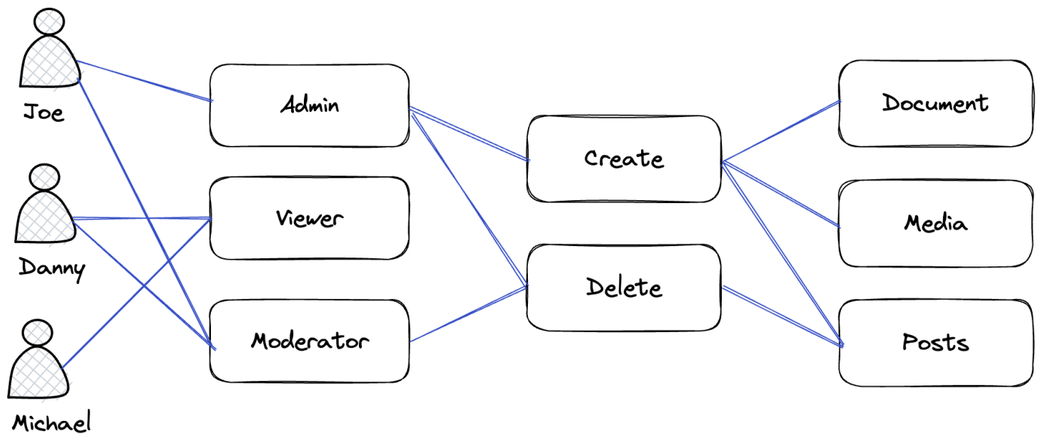
ReBAC - Relationship-Based Access Control

ACL - Access Control List



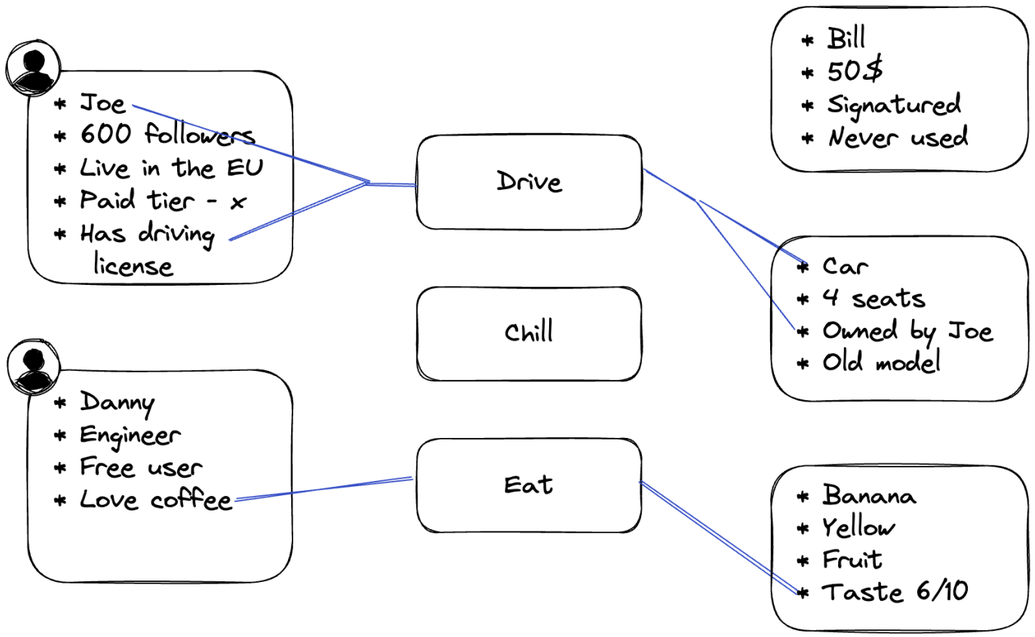
- EOL model
- Widely used in IT systems/ networks
- No segmentation/attribution support
- Hard to scale

RBAC - Role Based Access Control



- The widely-used model for app authorization
- 🤓 Easy to define, use and audit
- No resource inspection
- Limited scalablity

ABAC - Attribute Based Access Control

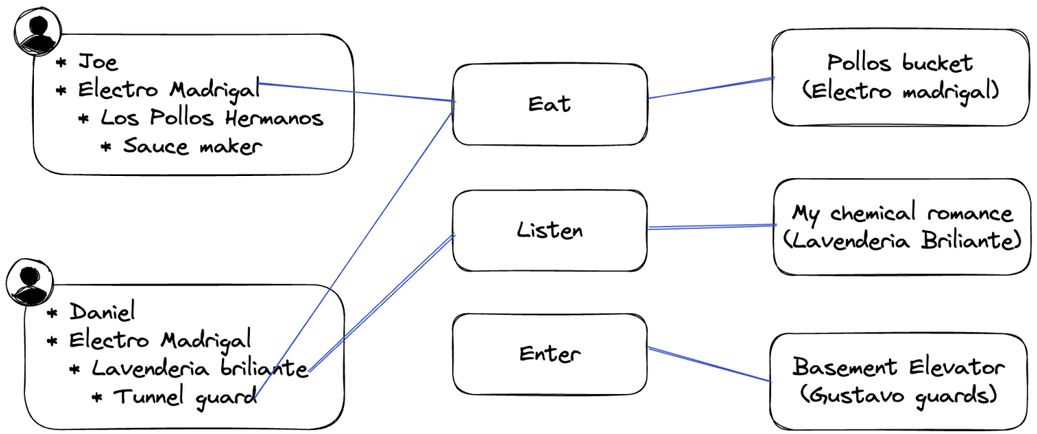


- The most robust model for inspection and desicion making
- Configuration could be hard
- Easy to handle multiple data sources
- 🚀 Highly scalable

Permit.io

@gemanor

ReBAC - Relationship Based Access Control



- Best fit for consumer-style applications
- Support in reverse indices and search for allowed data
- Easy to scale for users (>1b) hard in desicion's performance

#2 Author

Contracts Creates Better Relationships

👩🏽‍💻 *Especially in Human <> Machine Relationships* 🤖

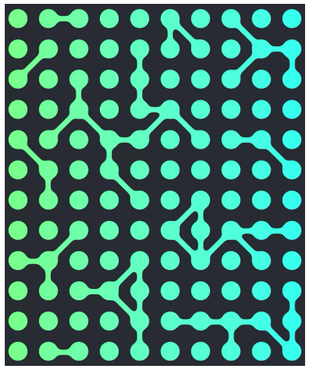

Permit.io

@gemanor

Open Policy Agent

AWS Cedar

Google Zanzibar - Open FGA



Is *[User]* Allowed to Perform *[Action]* on *[Resource]*

Is a Monkey Allowed to Eat a Banana

Permit.io

@gemanor


```
permit(
    principal in Role::"admin",
    action in [
        Action::"task:update",
        Action::"task:retrieve",
        Action::"task:list"
    ],
    resource in ResourceType::"task"
);
```

```
permit (
    principal,
    action in
        [Action::"UpdateList",
         Action::"CreateTask",
         Action::"UpdateTask",
         Action::"DeleteTask"],
    resource
)
when { principal in resource.editors };
```

```
permit (
    principal,
    action,
    resource
)
when {
    resource has owner &&
    resource.owner == principal
};
```


Permit.io

@gemanor

#3 Analyze

Cedar Agent

- Policy decision maker
- Decentralized continer, run as a sidecar to applications
- Monitored and audited
- Focused in getting very fast decisions >10ms

Permit.io

@gemanor

#4 Enforce

Enforcing Authorization Policies

```
# Call authorization service
# In the request body, we pass the relevant request information
allowed = requests.post('http://host.docker.internal:8180/v1/is_authorized', json={
    "principal": f"User::\"{user}\"",
    "action": f"Action::\"{method.lower()}\"",
    "resource": f"ResourceType::\"{original_url.split('/')[1]}\"",
    "context": request.json
}, headers={
    'Content-Type': 'application/json',
    'Accept': 'application/json'
})
```

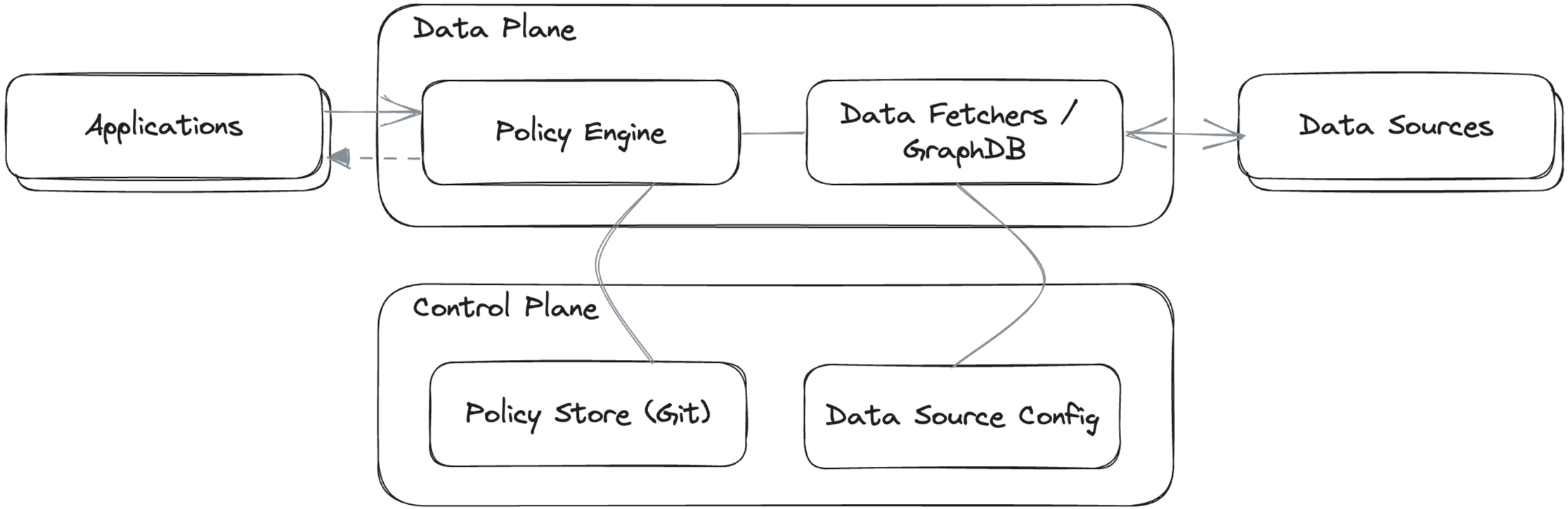
Permit.io

@gemanor

Permit.io

@gemanor

Authorization System Building Blocks



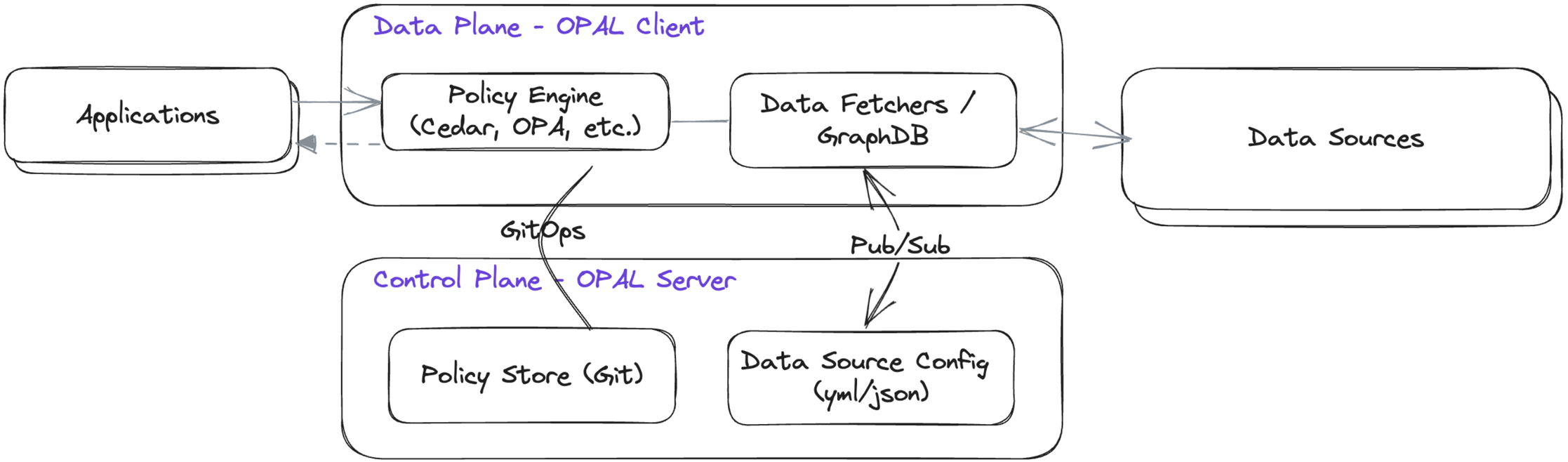
OPAL - Open Policy Administration Layer

- Open Source, Written in Python
- Sync decision points with data and policy stores
- Auto-scale for engines
- Centralized services such as Audit
- Unified APIs for the enforcement point
- Extensible for any kind of data source
- Supports OPA, Cedar (and soon to be announced more)
- Used by Tesla, Zapier, Cisco, Accenture, Walmart, NBA and thousands more

Permit.io

@gemanor

OPAL Based Authorization Architecture



Permit.io

@gemanor

It's Your Time to Shine 🌟

Contribute to OPAL

Permit.io

@gemanor

Thank You 🙏

Join our Next OPAL Office Hours

io.permit.io/opal_office_hours