



Application Code of Conduct – Full Stack Policy as Code

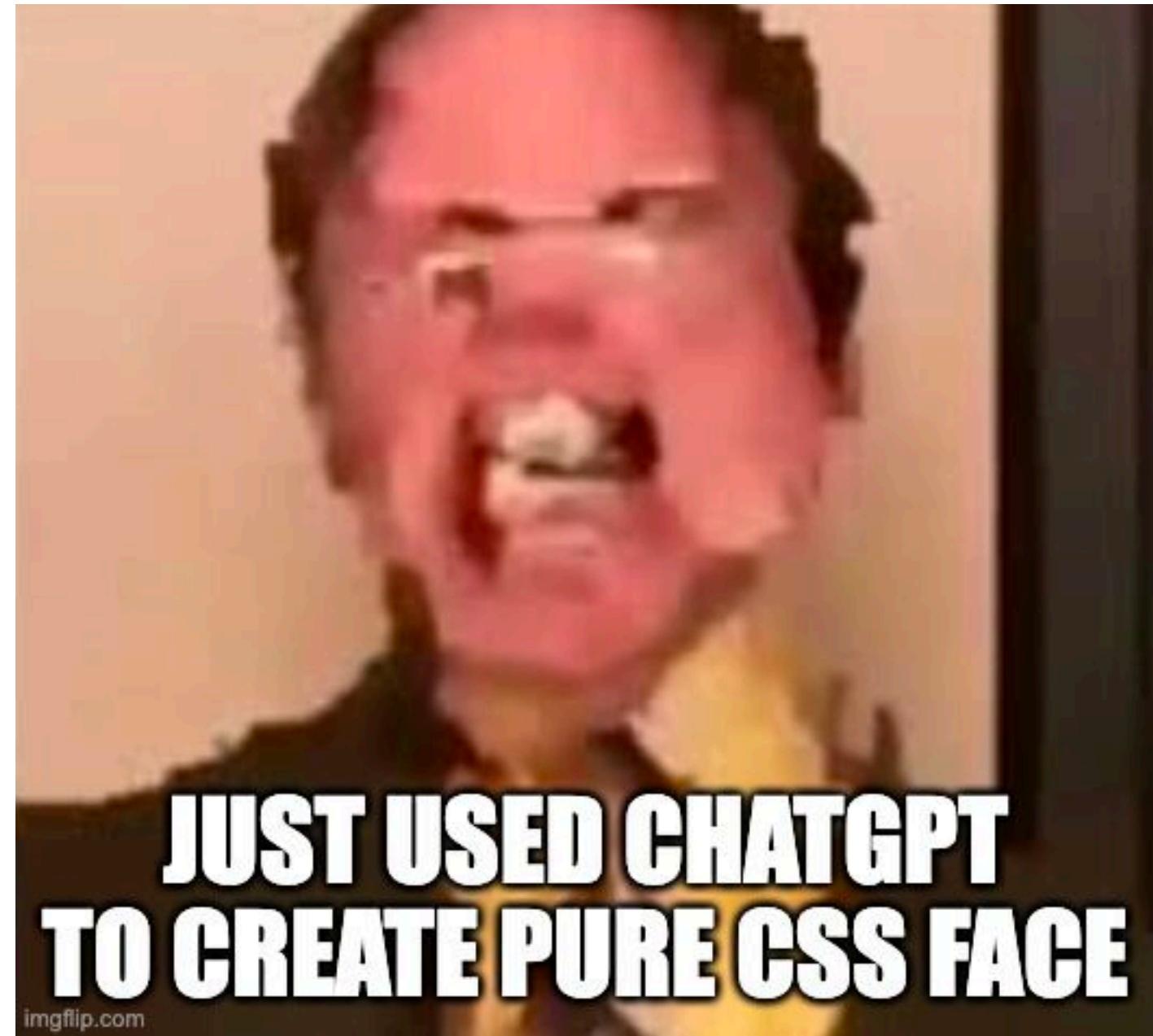
Gabriel L. Manor



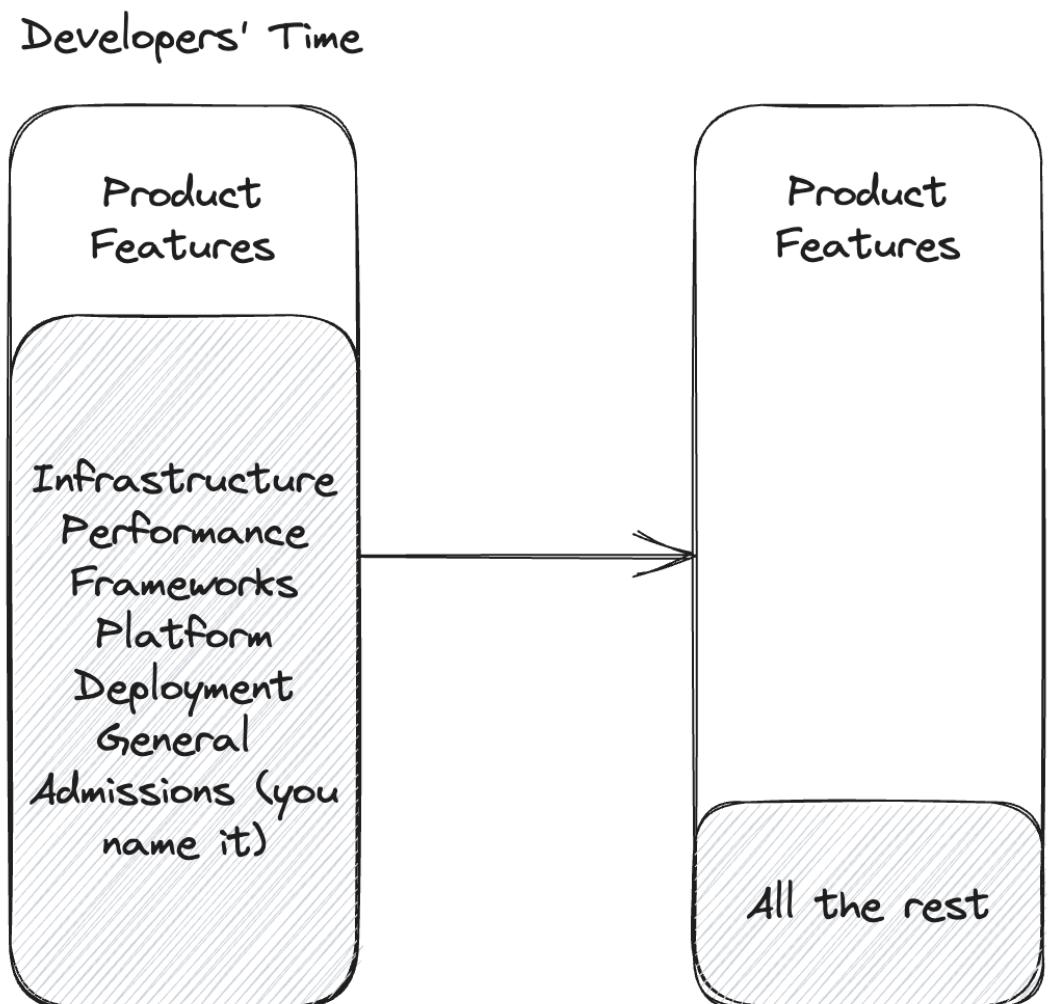
@gemanor

AI Is Going to Replace Us

– *Every Developer in 2023*



Agile New Mission



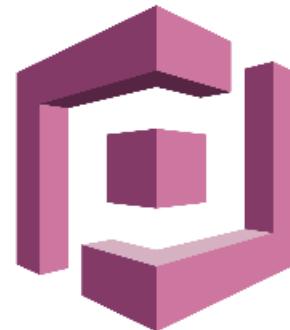
Case Study

Application-Level Access Control

	Authentication	Authorization
Definition	Verifies the identity of a user or system.	Determines what a user or system can do.
Purpose	Grants access to the system.	Controls access to resources.
Focus	"Who you are."	"What you can do."
Process	Verifies credentials (e.g., username/password).	Checks permissions and privileges.
Examples	Logging in with a username and password.	Granting read or write access to a file.
Goal	Establishes trust and identity.	Ensures appropriate access and usage.
Applied to	Users, systems, or devices.	Resources, data, or functionalities.
Scope	Typically performed once per session.	Performed for every access attempt.



Auth0



c clerk

STYTCH

frontegg

SuperTokens



Permit.io

@gemanor

The Auth Comfort Zone

Authentication 😊

Verify who the user is

```
if (!user) {  
    return;  
}
```

Authorization 😔

Check what a user can do

```
if (!allowed(  
    user,  
    action,  
    resource  
)  
) {  
    return;  
}
```



Gabriel L. Manor

Director of DevRel @ Permit.io

🥇 Struggling with authorization for the last 8y

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

```
// Middleware
if (req.user.roles.indexOf(role) === -1) {
  return res.send(403);
}

// Endpoint
@authz('admin')
const Document = () => {
  ...
}
```

```
// Middleware
const hasRole = roles.filter(r => (
  req.user.roles.indexOf(role) > -1
))
if (!hasRole) {
  return res.send(403);
}
```

```
// Endpoint
@authz('admin')
@authz('paid')
const Document = () => {
  ...
}
```

```
// Middleware
if (args.length === 2 && req.user[args[0]] !== args[1]) {
  return res.send(403);
}
if (req.user.roles.indexOf(role) > -1) {
  return res.send(403);
}

// Endpoint
@authz('admin')
@authz('location', 'eu')
const Document = () => {
  ...
}
```



Permit.io

@gemanor

```
// Middleware
if (typeof args[0] === 'object') {
  return AuthzDesicion(args[0])
}
if (args.length === 2 && req.user[args[0]] !== args[1]) {
  return res.send(403);
}
if (req.user.roles.indexOf(role) > -1) {
  return res.send(403);
}

// Endpoint
@authz('admin')
@authz('location', 'eu')
@authz({
  ...
})
const Document = () => {
  ...
}
```



@gemanor

Implement RBAC to an Application

~~Implement RBAC to Application~~

Enforce Permissions in Applications

User | Action | Resource

Does [User] Allowed to Perform [Action] on [Resource]
Does a Monkey Allowed to Eat a Banana

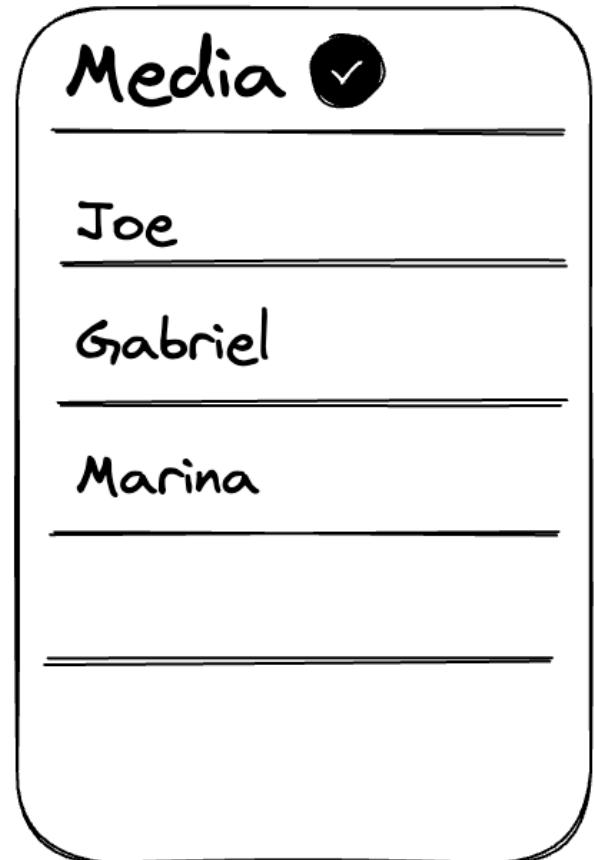
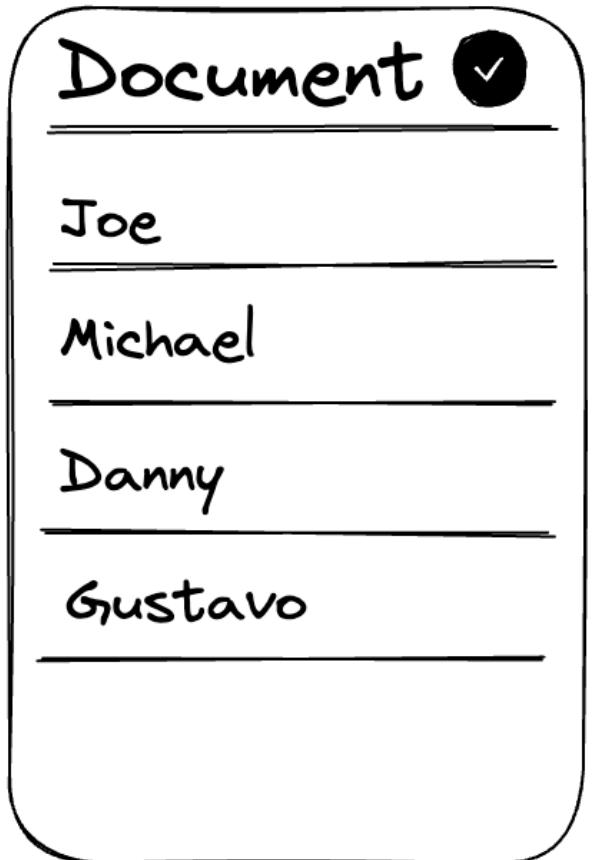
ACL - Access Control List

RBAC - Role-Based Access Control

ABAC - Attribute-Based Access Control

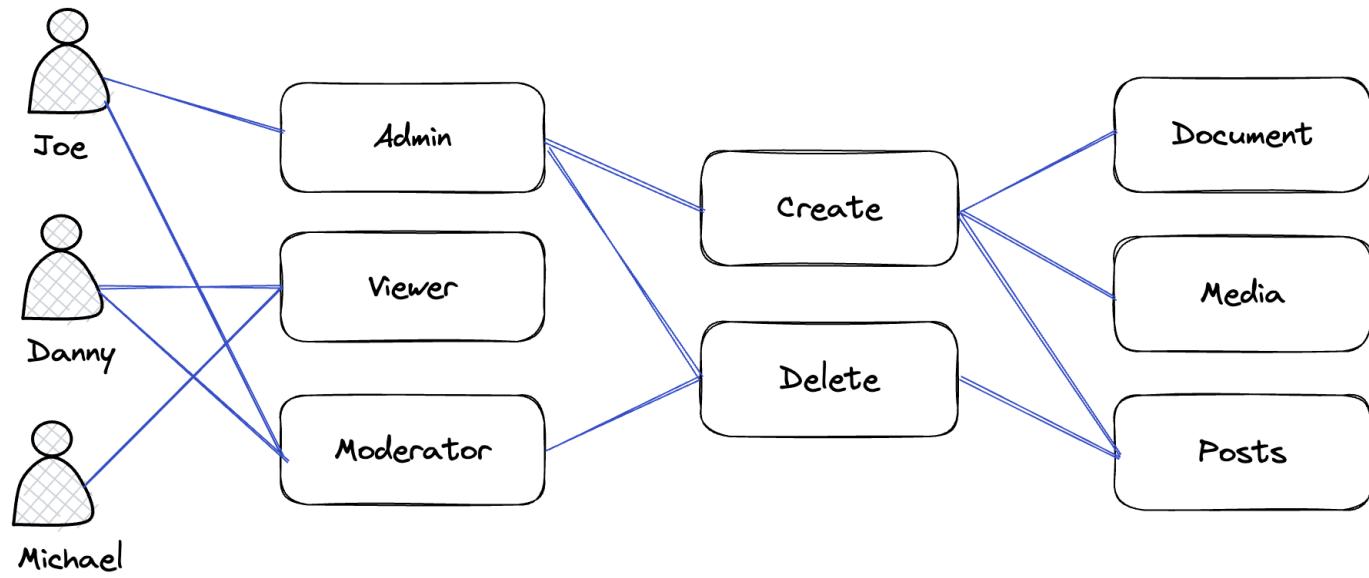
ReBAC - Relationship-Based Access Control

ACL - Access Control List



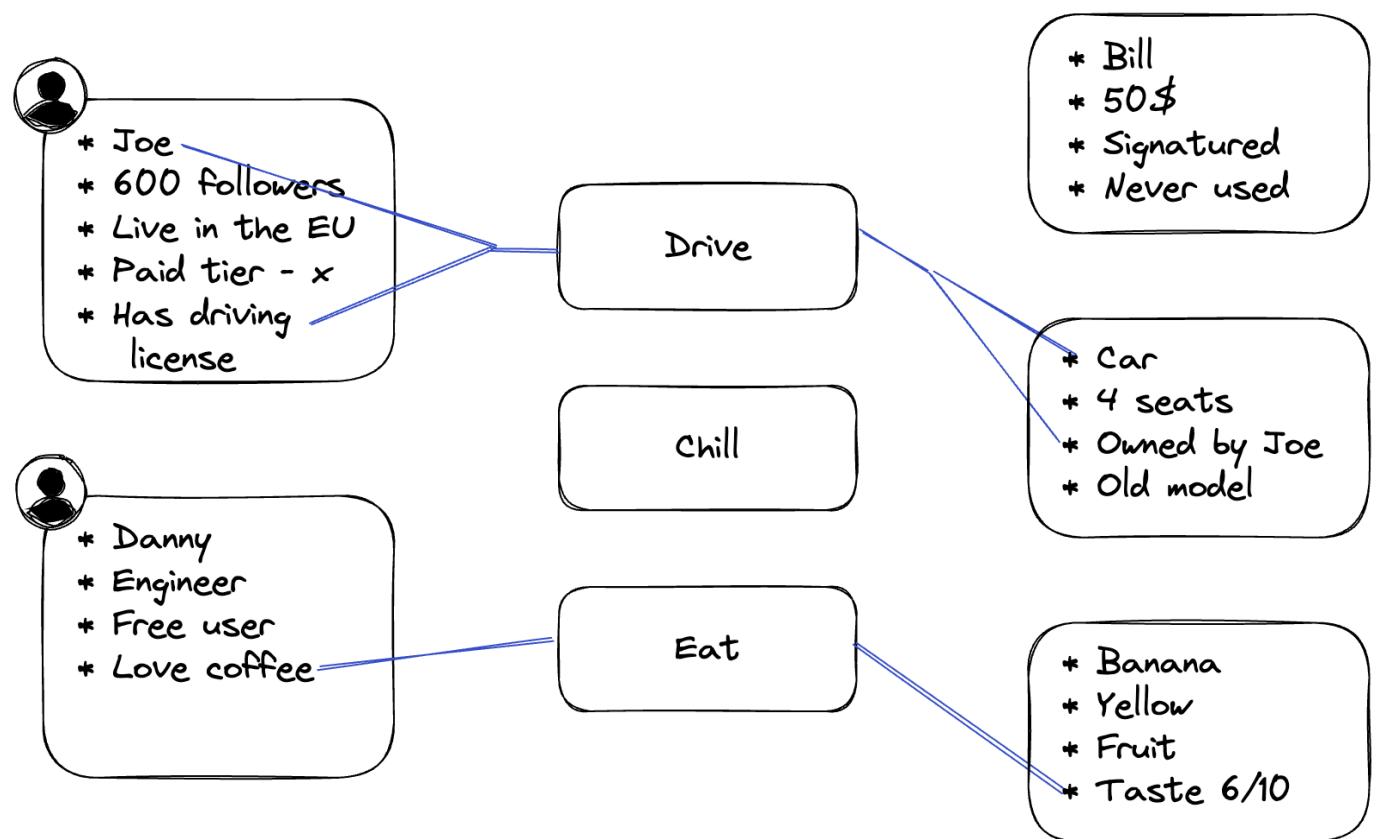
- EOL model
- Widely used in IT systems/networks
- No segmentation/attribution support
- Hard to scale

RBAC - Role Based Access Control



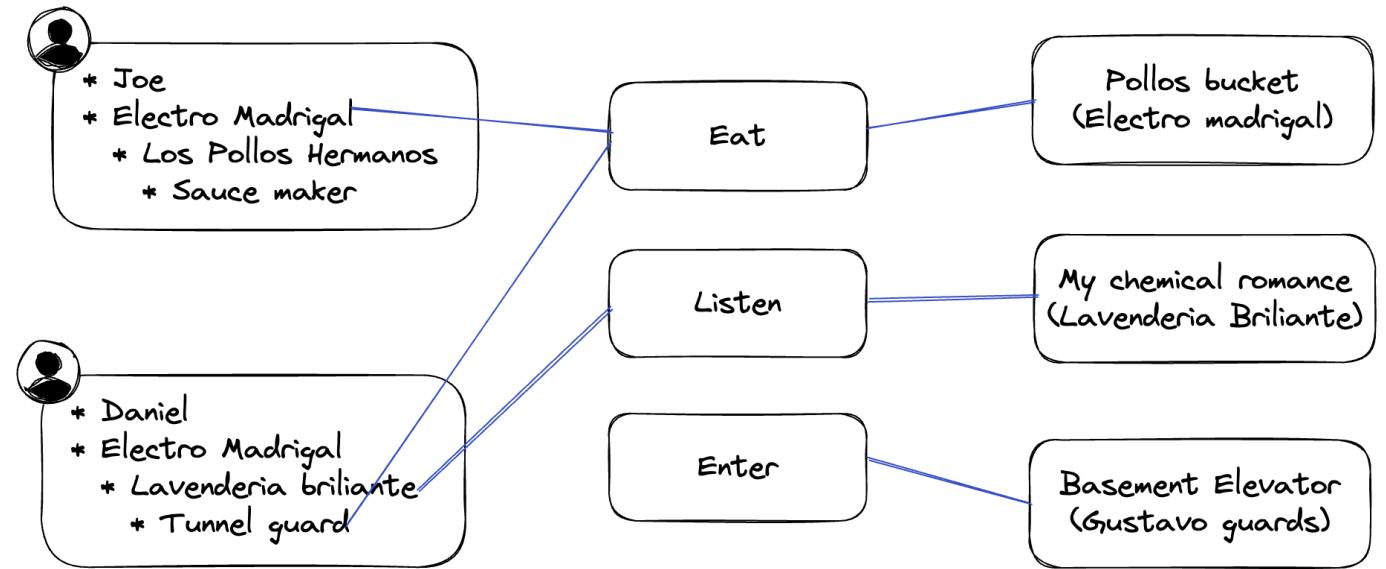
- The widely-used model for app authorization
- 😎 Easy to define, use and audit
- No resource inspection
- Limited scalability

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

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

```
// Middleware
if (typeof args[0] === 'object') {
  return AuthzDesicion(args[0])
}
if (args.length === 2 && req.user[args[0]] !== args[1]) {
  return res.send(403);
}
if (req.user.roles.indexOf(role) > -1) {
  return res.send(403);
}

// Endpoint
@authz('admin')
@authz('location', 'eu')
@authz({
  ...
})
const Document = () => {
  ...
}
```



@gemanor



```
// Middleware
if (typeof args[0] === 'object') {
  return AuthzDesicion(args[0])
}
if (args.length === 2 && req.user[args[0]] !== args[1]) {
  return res.send(403);
}
if (req.user.roles.indexOf(role) > -1) {
  return res.send(403);
}

// Endpoint
@authz('admin')
const Document = () => {
  const enrichedUser = getSocialData(req.user);
  if (enrichedUser.lastGeolocation !== 'eu') {
    return;
  }
  ...
}
```



@gemanor

```
// Middleware
if (typeof args[0] === 'object') {
  return AuthzDesicion(args[0])
}
if (args.length === 2 && req.user[args[0]] !== args[1]) {
  return res.send(403);
}
if (req.user.roles.indexOf(role) > -1) {
  return res.send(403);
}

// Endpoint
@authz('admin')
const Document = () => {
  const enrichedUser = getSocialData(req.user);
  if (enrichedUser.lastGeolocation !== 'eu') {
    return;
  }
  ...
}
```



@gemanor

Authorization Top 5 Best-Practices

Authorization Top 5 Best-Practices

- Agnostic to the permissions model

Authorization Top 5 Best-Practices

- Agnostic to the permissions model
- Separate policy from code

Authorization Top 5 Best-Practices

- Agnostic to the permissions model
- Separate policy from code
- Support the whole stack

Authorization Top 5 Best-Practices

- Agnostic to the permissions model
- Separate policy from code
- Support the whole stack
- Support in the whole authorization lifecycle (author, analyze, audit)

Authorization Top 5 Best-Practices

- Agnostic to the permissions model
- Separate policy from code
- Support the whole stack
- Support in the whole authorization lifecycle (author, analyze, audit)
- Unified policy management

Contracts Creates Better Relationships



Especially in Human <> Machine Relationships



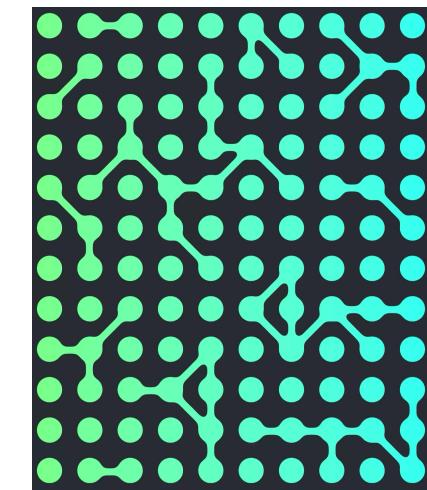
Open Policy
Agent



AWS
Cedar



Google
Zanzibar



```
package app.rbac

import future.keywords.contains
import future.keywords.if
import future.keywords.in

default allow := false

allow if user_is_admin

allow if {
    some grant in user_is_granted
    input.action == grant.action
    input.type == grant.type
}

user_is_admin if "admin" in data.user_roles[input.user]

user_is_granted contains grant if {
    some role in data.user_roles[input.user]
    some grant in data.role_grants[role]
}
```

```
package app.abac

import future.keywords.if

default allow := false

allow if user_is_owner

allow if {
    user_is_employee
    action_is_read
}

allow if {
    user_is_employee
    user_is_senior
    action_is_update
}

allow if {
    user_is_customer
    action_is_read
    not pet_is_adopted
}

user_is_owner if data.user_attributes[input.user].title == "owner"

user_is_employee if data.user_attributes[input.user].title == "employee"

user_is_customer if data.user_attributes[input.user].title == "customer"

user_is_senior if data.user_attributes[input.user].tenure > 8

action_is_read if input.action == "read"

action_is_update if input.action == "update"

pet_is_adopted if data.pet_attributes[input.resource].adopted == true
```

Open Policy Agent

-  Wide ecosystem
-  Rego is comprehensive and robust
- Rego is complex 
- Data cache replica
- Enforcement plugins



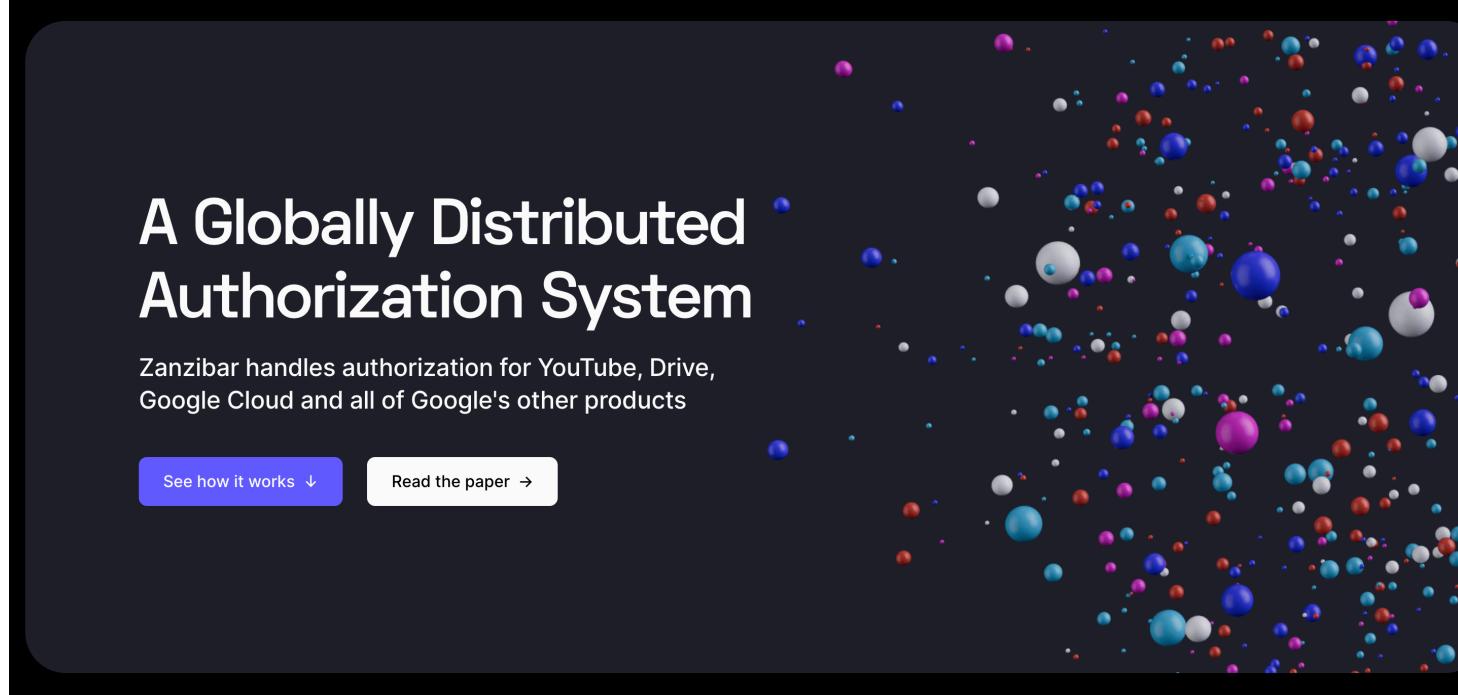
```
permit(  
    principal in Role::"admin",  
    action in [  
        Action::"task:update",  
        Action::"task:retrieve",  
        Action::"task:list"  
    ],  
    resource in ResourceType::"task"  
);
```

```
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  
};
```

AWS Cedar

-  First designed as application authz language
-  Backed by AWS
- Just released 
- ReBAC via ABAC
- Benchmark winner



```
name: "doc"

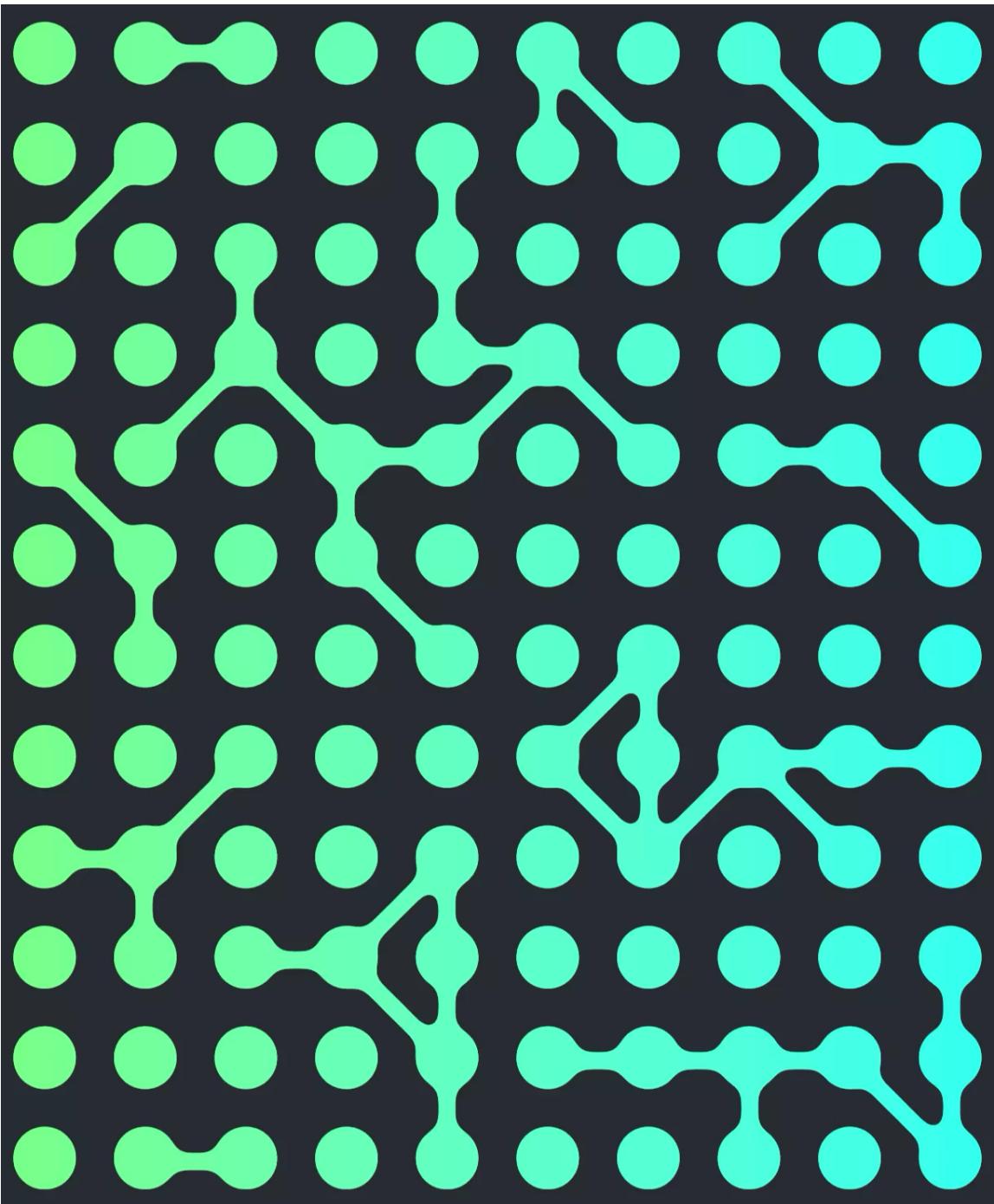
relation { name: "owner" }

relation {
  name: "editor"
  userset_rewrite {
    union {
      child { _this {} }
      child { computed_userset { relation: "owner" } }
    } } }

relation {
  name: "viewer"
  userset_rewrite {
    union {
      child { _this {} }
      child { computed_userset { relation: "editor" } }
      child { tuple_to_userset {
        tupleset { relation: "parent" }
        computed_userset {
          object: $TUPLE_USERSET_OBJECT # parent folder
          relation: "viewer"
        } } }
    } } }
```

OpenFGA Implementation

```
{  
  "schema_version": "1.1",  
  "type_definitions": [{}  
    {"type": "user"  
    }, {  
      "type": "document",  
      "relations": {  
        "reader": {"this": {}},  
        "writer": {"this": {}},  
        "owner": {"this": {}}  
      },  
      "metadata": {  
        "relations": {  
          "reader": {  
            "directly_related_user_types": [{"type": "user"}]  
          },  
          "writer": {  
            "directly_related_user_types": [{"type": "user"}]  
          },  
          "owner": {  
            "directly_related_user_types": [{"type": "user"}]  
          }  
        }  
      }  
    }  
  ]  
}
```

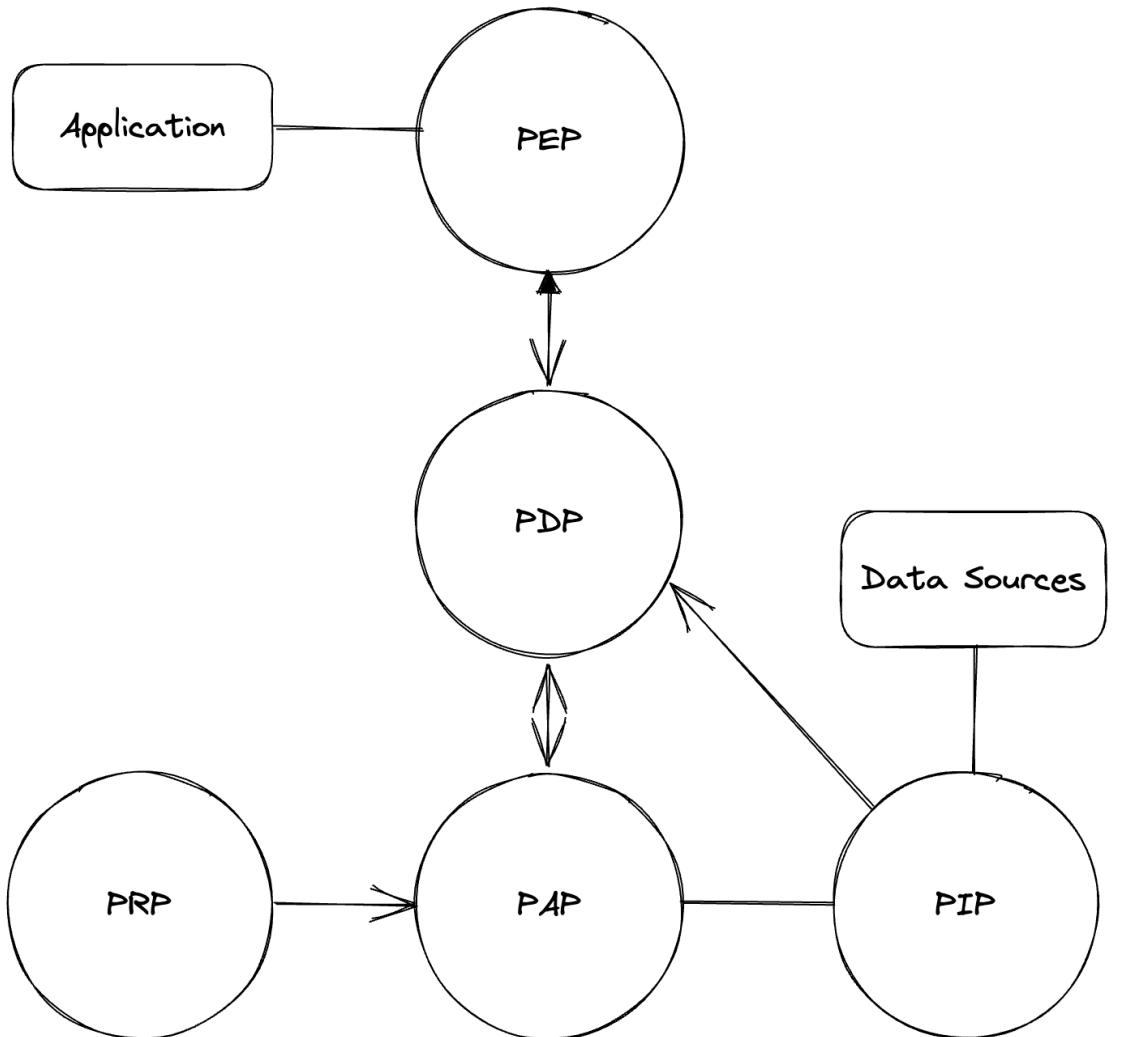


Google Zanzibar - OpenFGA

- Simple ReBAC
- Complex RBAC/ABAC
- Reverse indices support
- Backed by OKTA
- Stateful

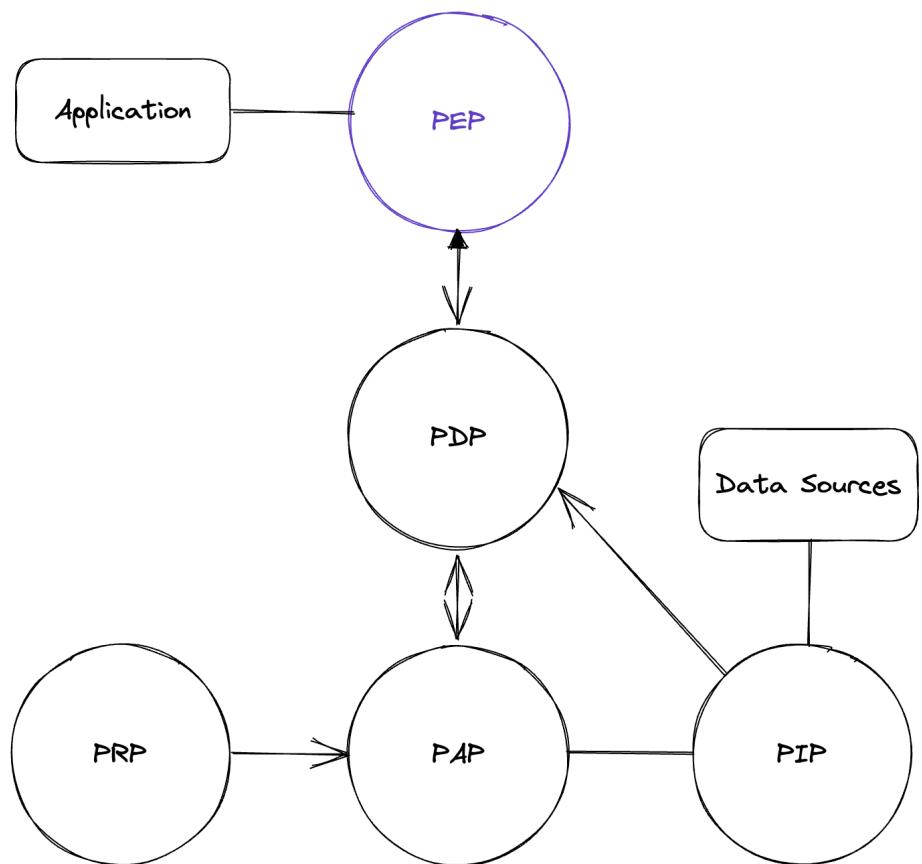
Next Station: Architecture

The Policy System Building Blocks



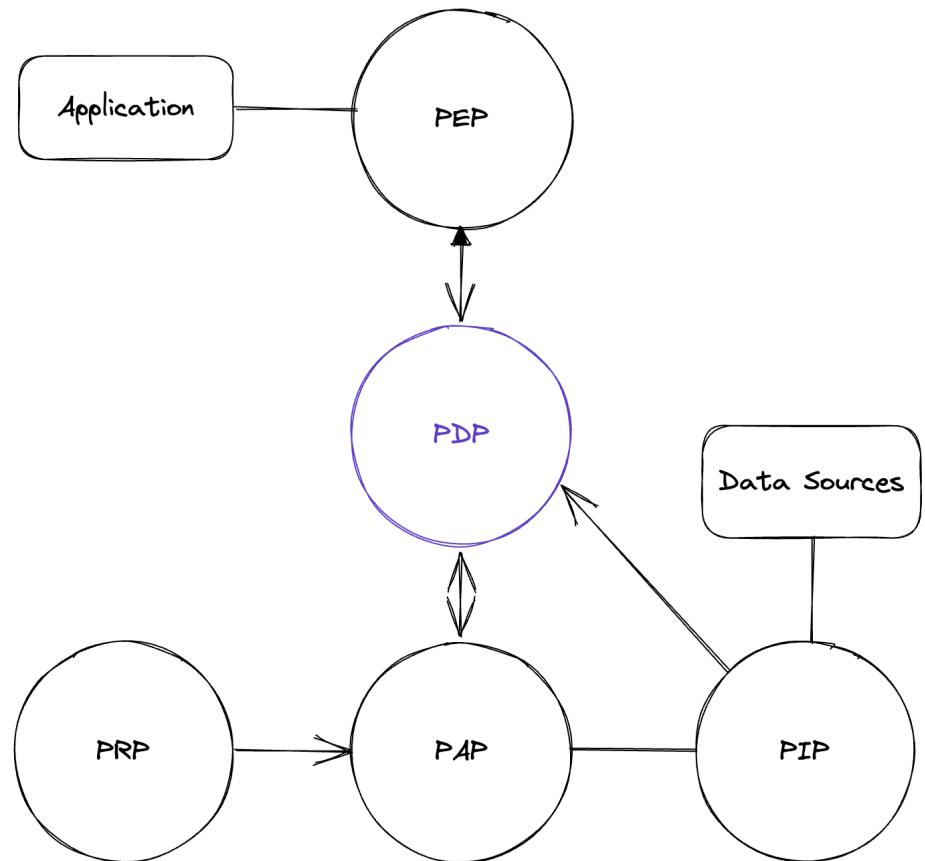
- Policy Enforcement Point
- Policy Decision Point
- Policy Retrieval Point
- Policy Information Point
- Policy Administration Point

PEP - Policy Enforcement Point



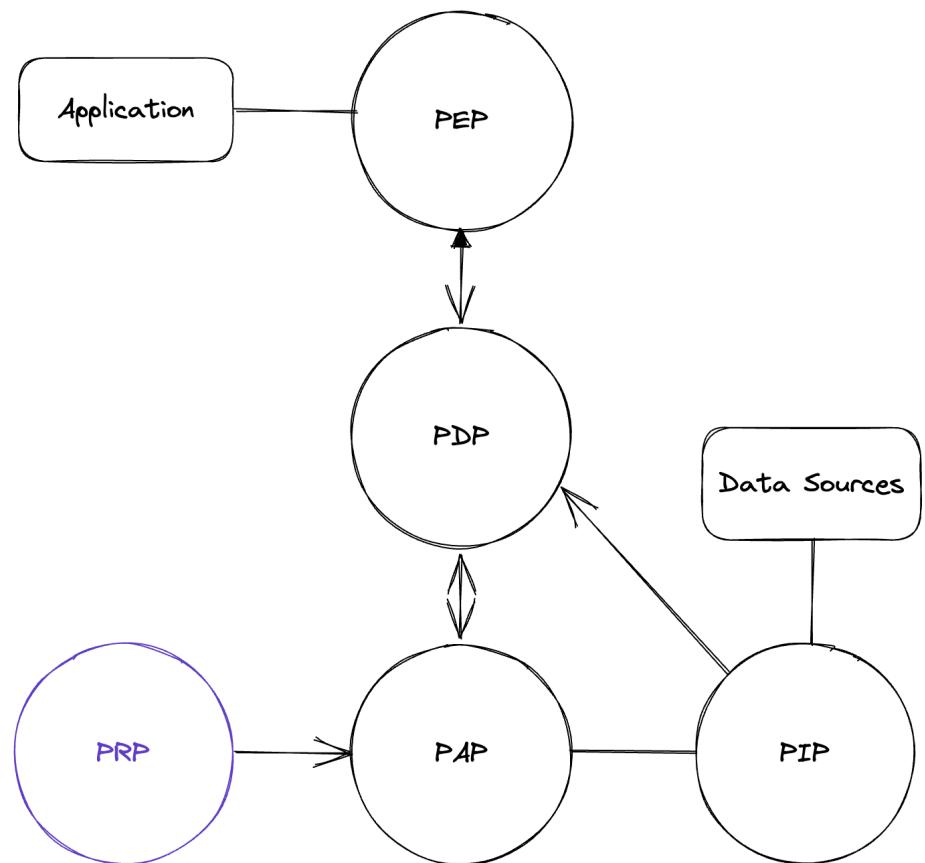
- Enforce policy decisions in the application
- An abstract representation of check function
check(user, action, resource)
- Policy contract agnostic
- Could be in application code or as a plugin
- In ReBAC should include search/list allowed object endpoint

PDP - Policy Desicion Point



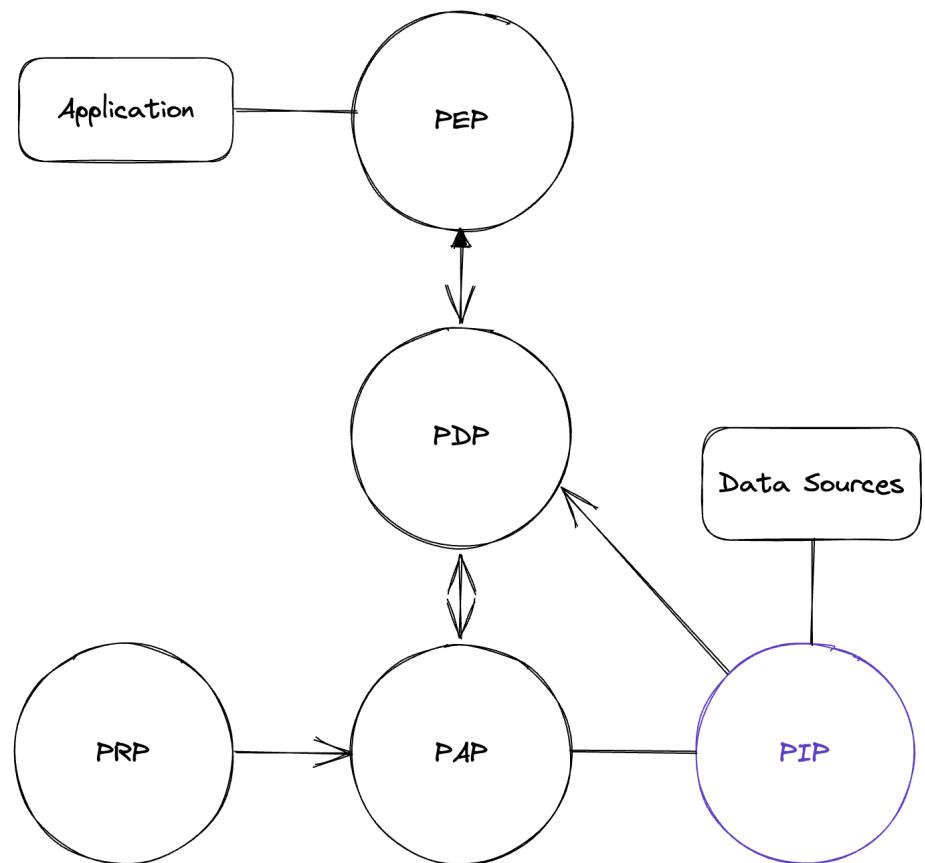
- Decision engine
- Stateless and cache enabled
- Serve request in <10ms
- Should run as a sidecar to the application

PRP - Policy Retrieval Point



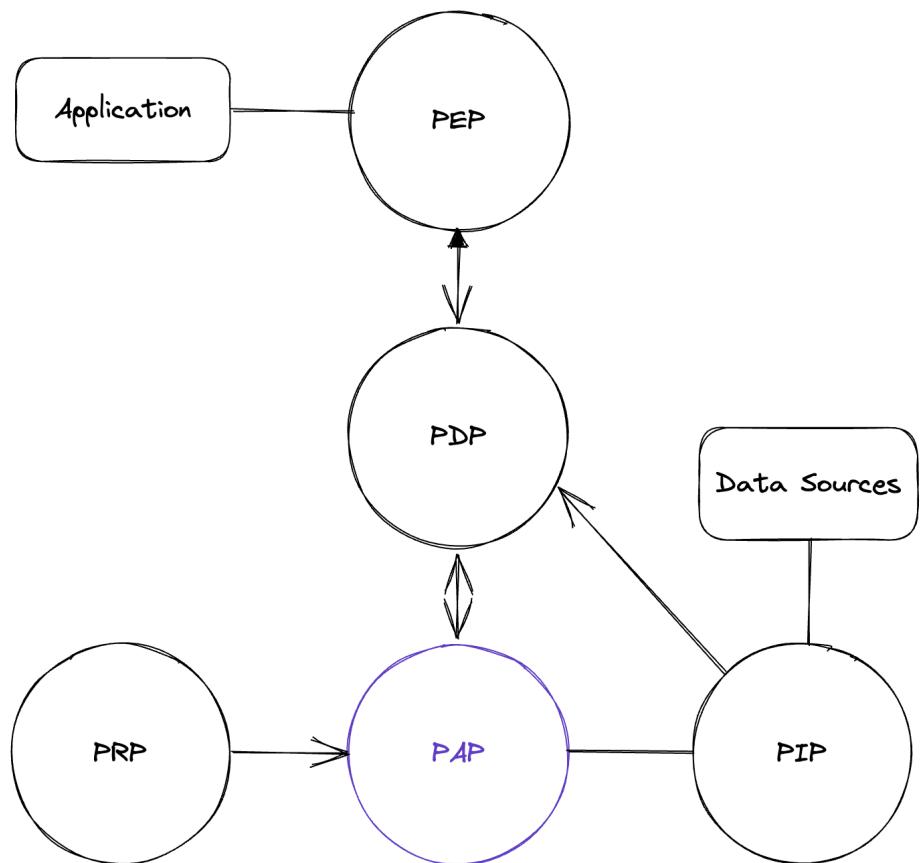
- Policy config storage
- GitOps enabled git repository
- Use IaC principles
- Branches represents environments

PIP - Policy Information Point

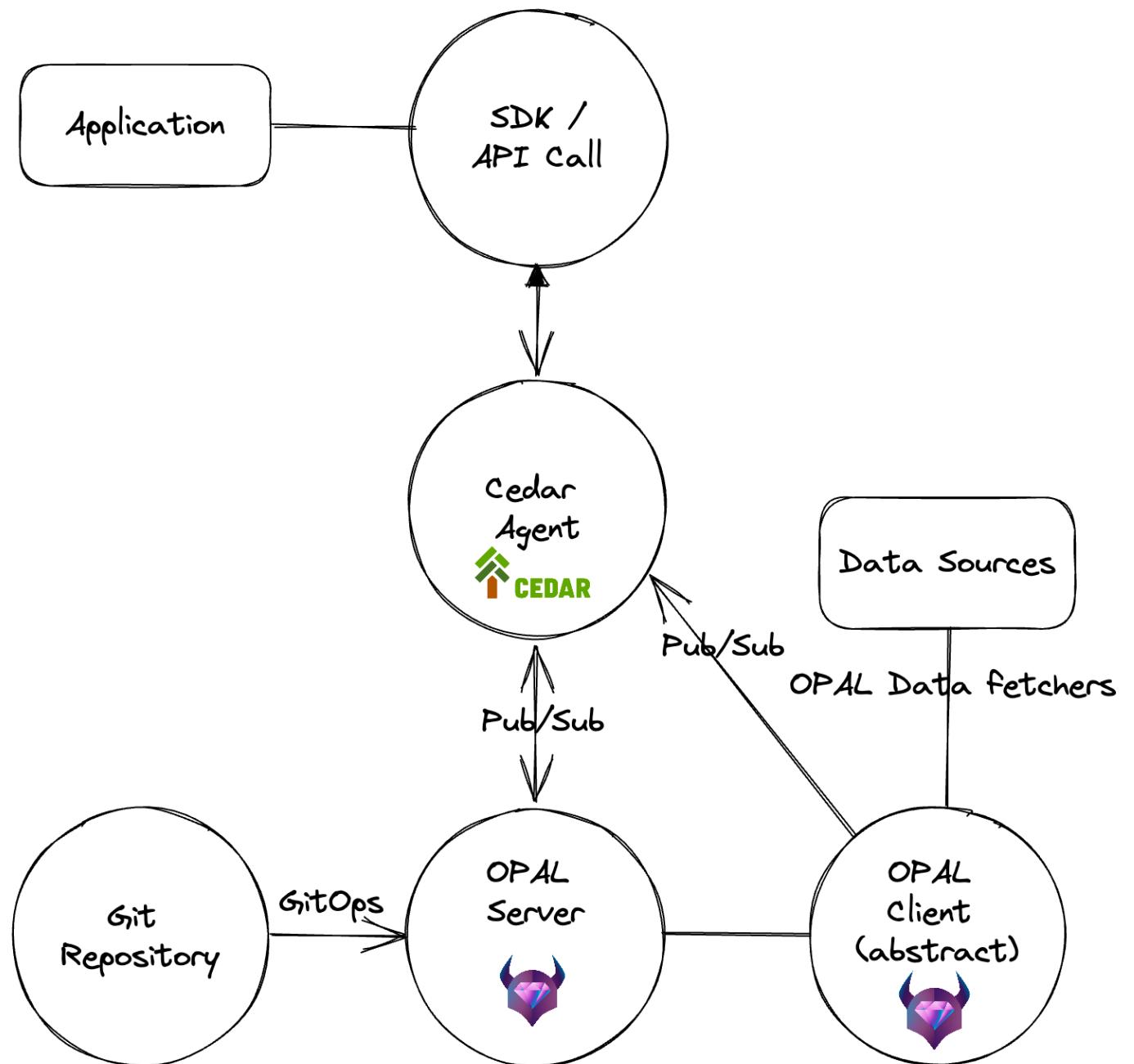


- Every kind of data to enrich the policy decision making
- Traditionally - IDM
- Currently - data fetchers
- Pub/Sub with PDP

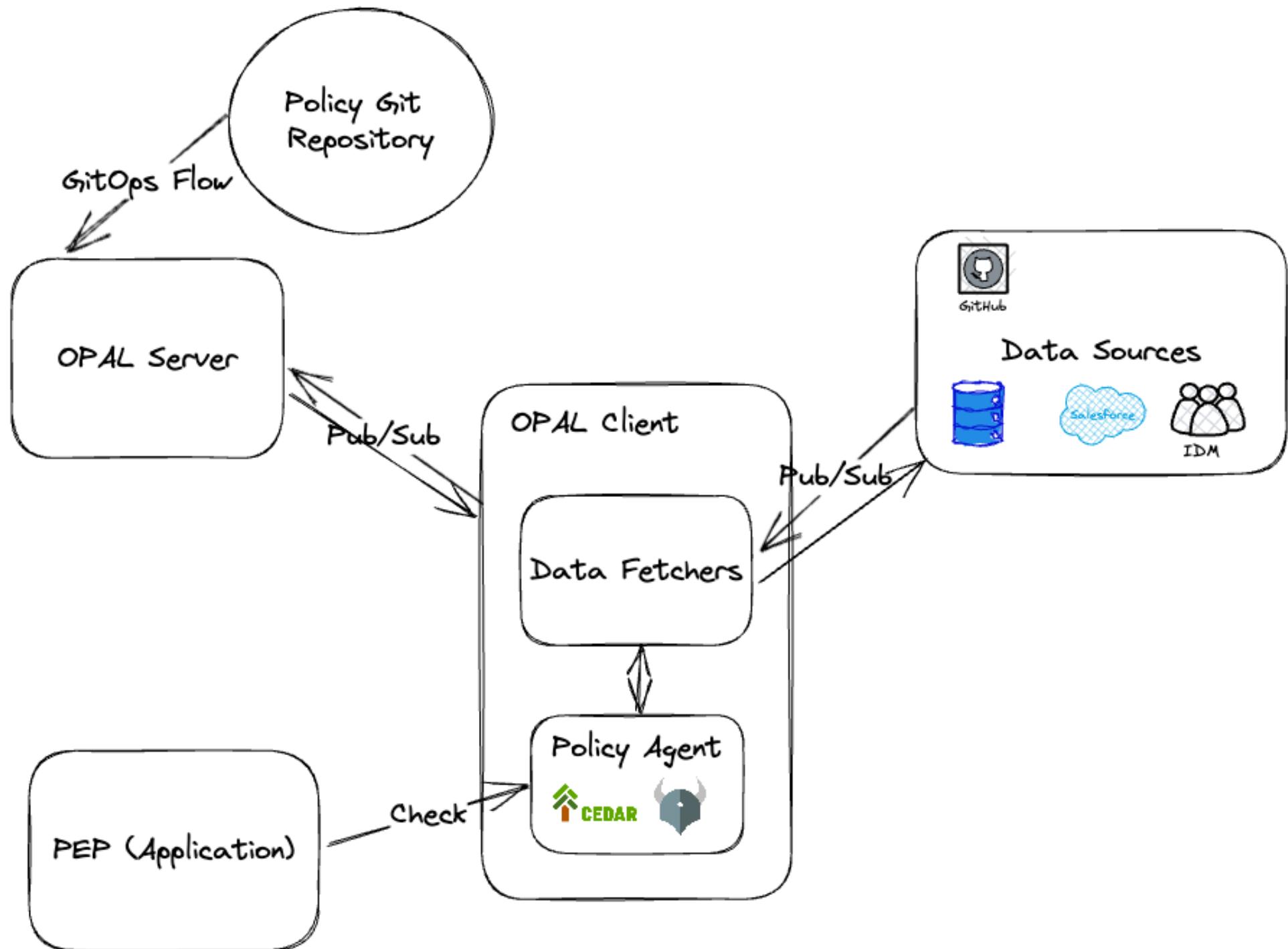
PAP - Policy Administration Point



- Keep the PDP/PIP/PRP synced
- The PDP "backend"
- Make sure all the PDP data is up to date
- System configuration one-stop-shop
- Scale PDPs



- PEP - Call the PDP APIs (HTTP/REST)
- PDP - Cedar Agent
<https://github.com/permit-policy>
<https://github.com/permitio/cedar-agent>
- PAP - OPAL server
<https://github.com/permitio/opal>
- PRP - Git Repository
- PIP - OPAL Client data fetchers



Permit.io

@gemanor

PEP Deployment Strategies

- Application level
 - HTTP API calls | Middlewares | SDKs | API Gateway plugins
- Persistence Layer
 - DB access plugins
- Adminissions
 - K8s controllers | TF plugins

CASL - The Frontend PEP SDK

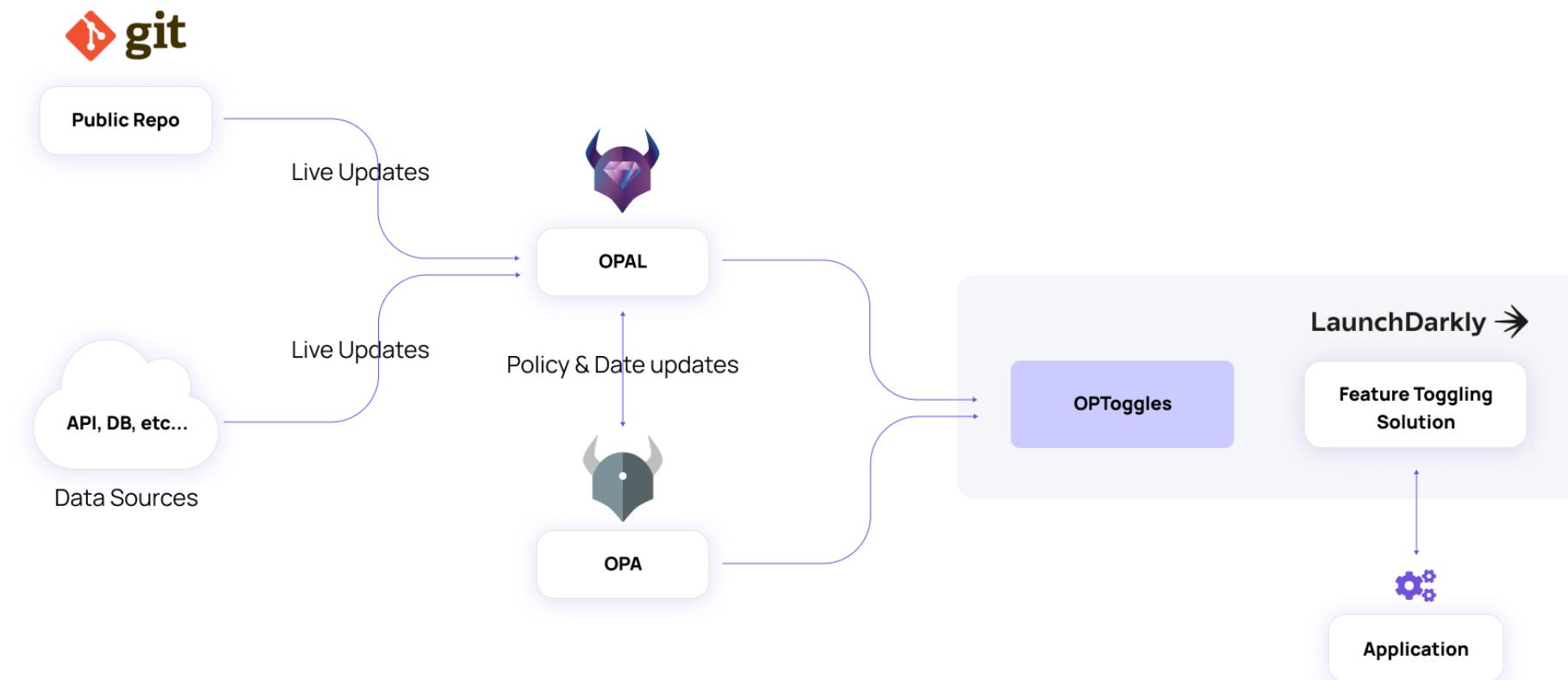
```
import { createMongoAbility, AbilityBuilder } from '@casl/ability';

// define abilities
const { can, cannot, build } = new AbilityBuilder(createMongoAbility);

can('read', ['Post', 'Comment']);
can('manage', 'Post', { author: 'me' });
can('create', 'Comment');

// check abilities
const ability = build();

ability.can('read', 'Post') // true
```

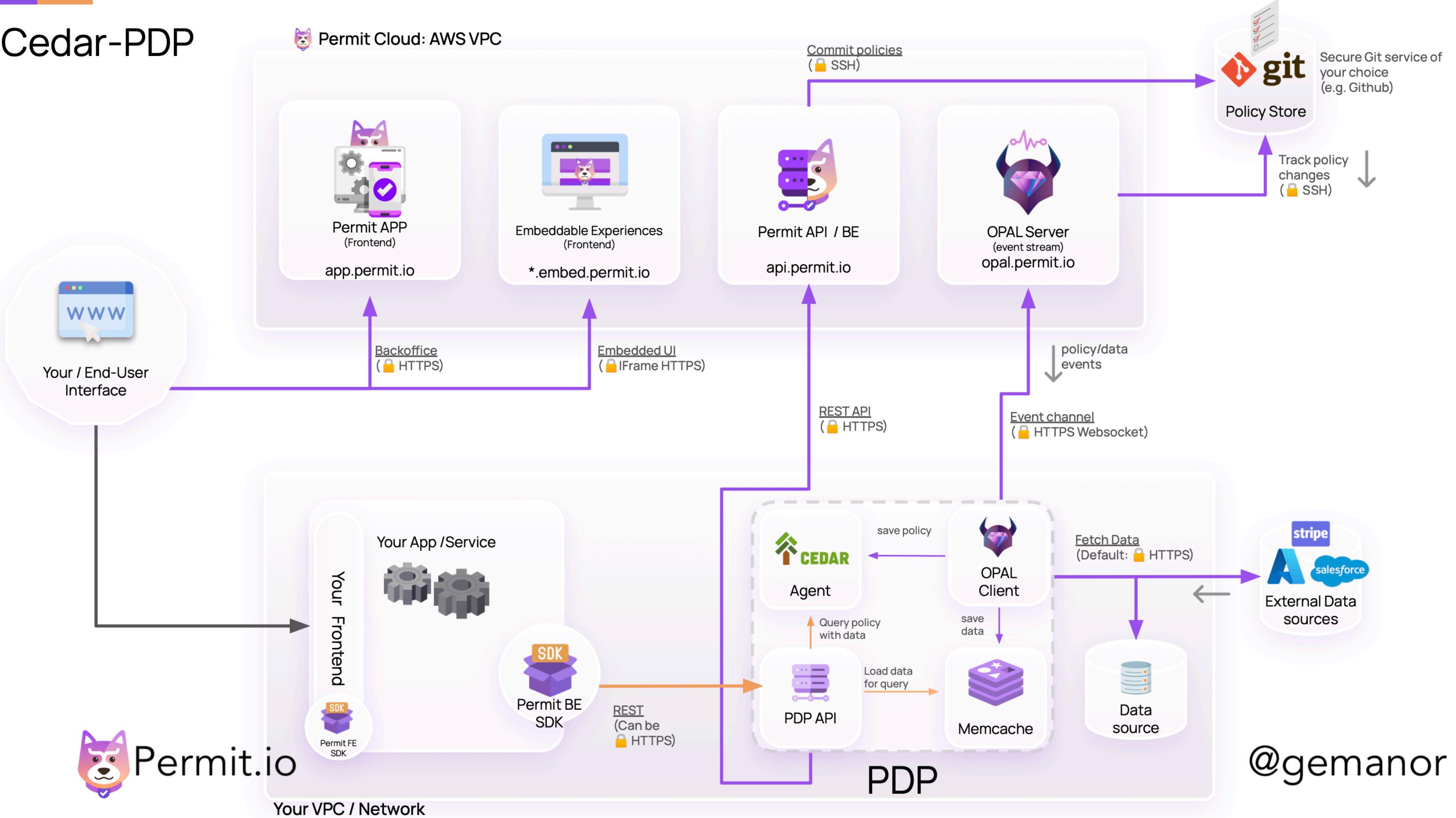


<https://github.com/permitio/optoggles>

Demo time



Cedar-PDP





Thank You 🙏

Let's keep in touch at our
Authorization Community

io.permit.io/chat_with_gabriel