



The evolution of access control

Relatore

Anders Eknert, Styra

#OSW2021



RETE ITALIANA
OPEN SOURCE



The evolution of access control



Identity and authorization in distributed systems

Anders Eknert

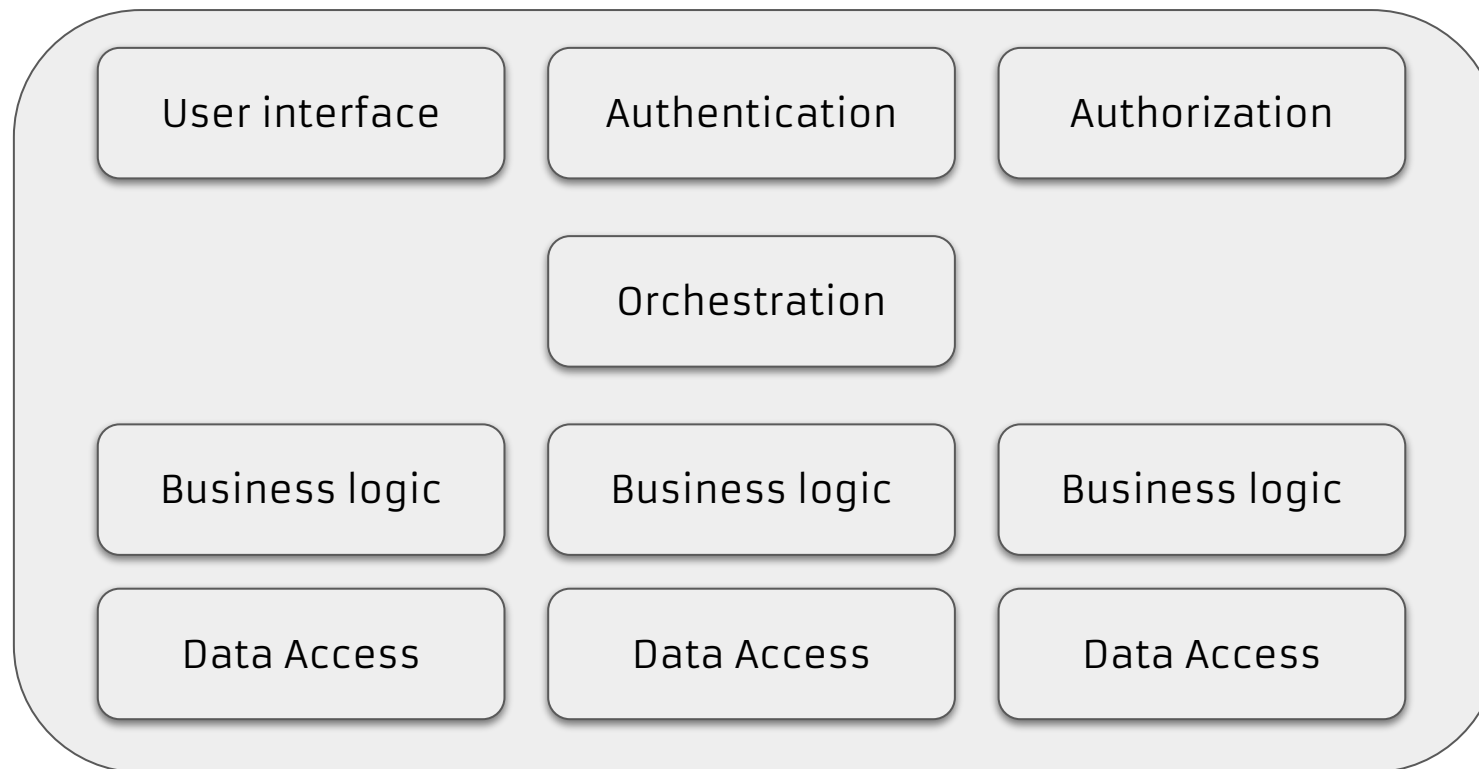


RETE ITALIANA
OPEN SOURCE

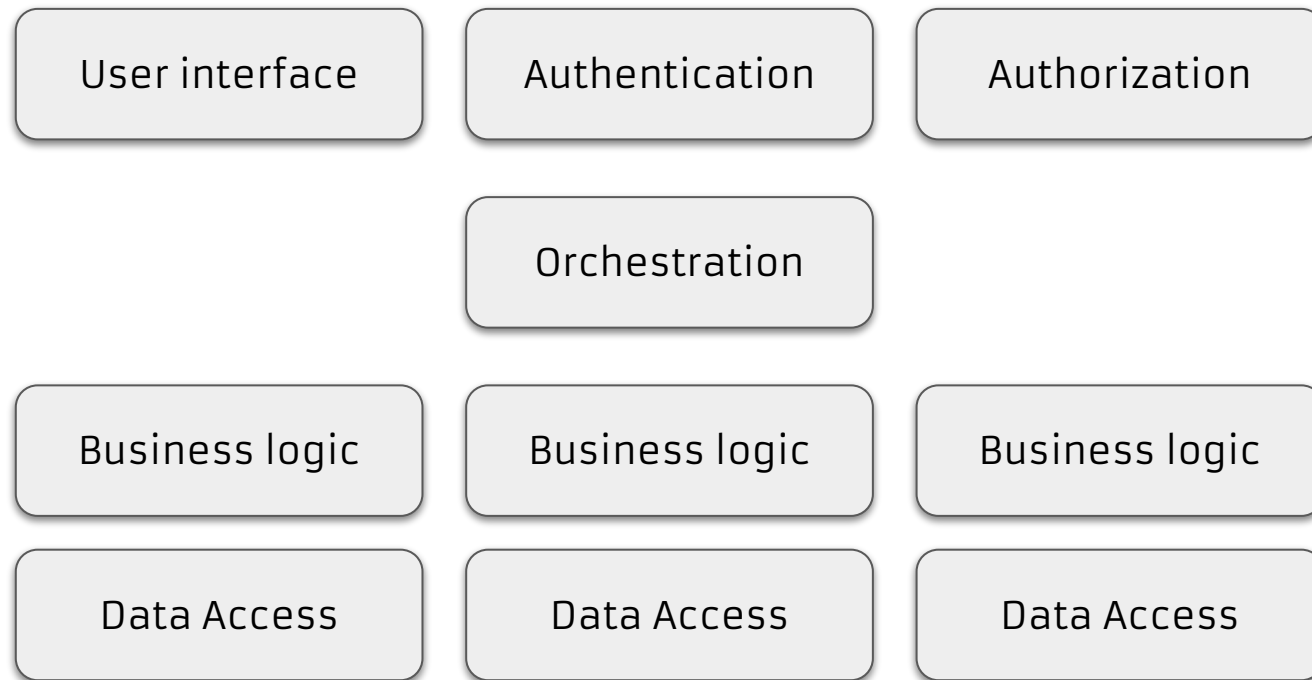


- Developer Advocate at  styra
- Software development
- Background in identity systems
- Three years into OPA
- Cooking and food
- Football 

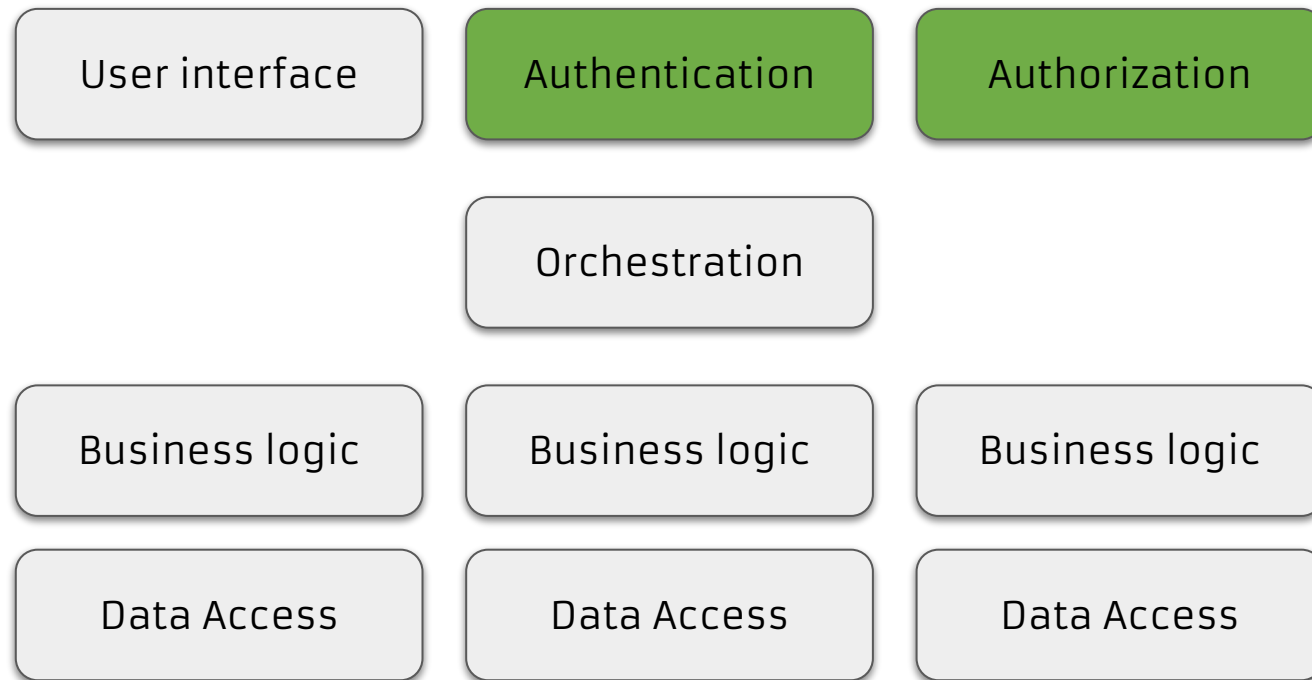
From monolith



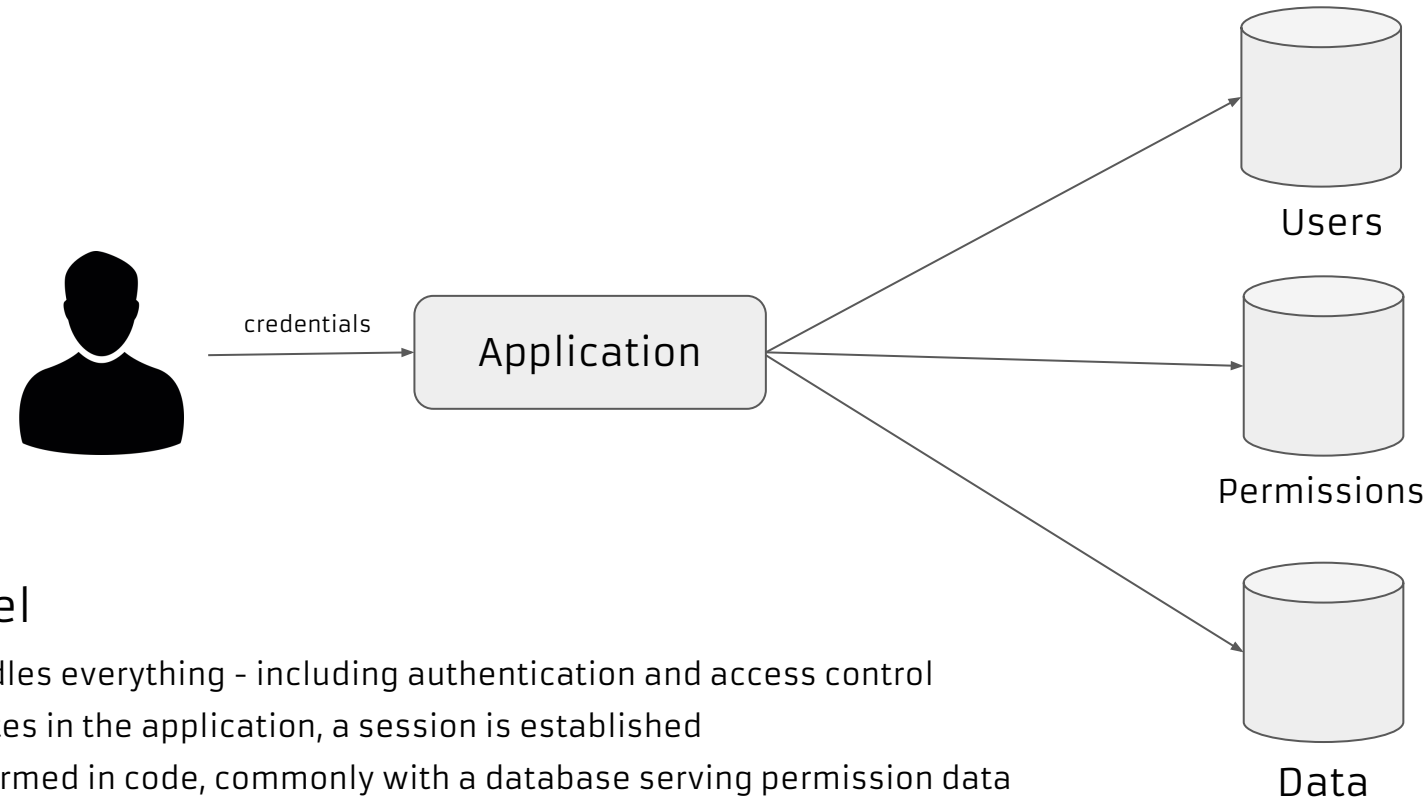
To microservices



To microservices



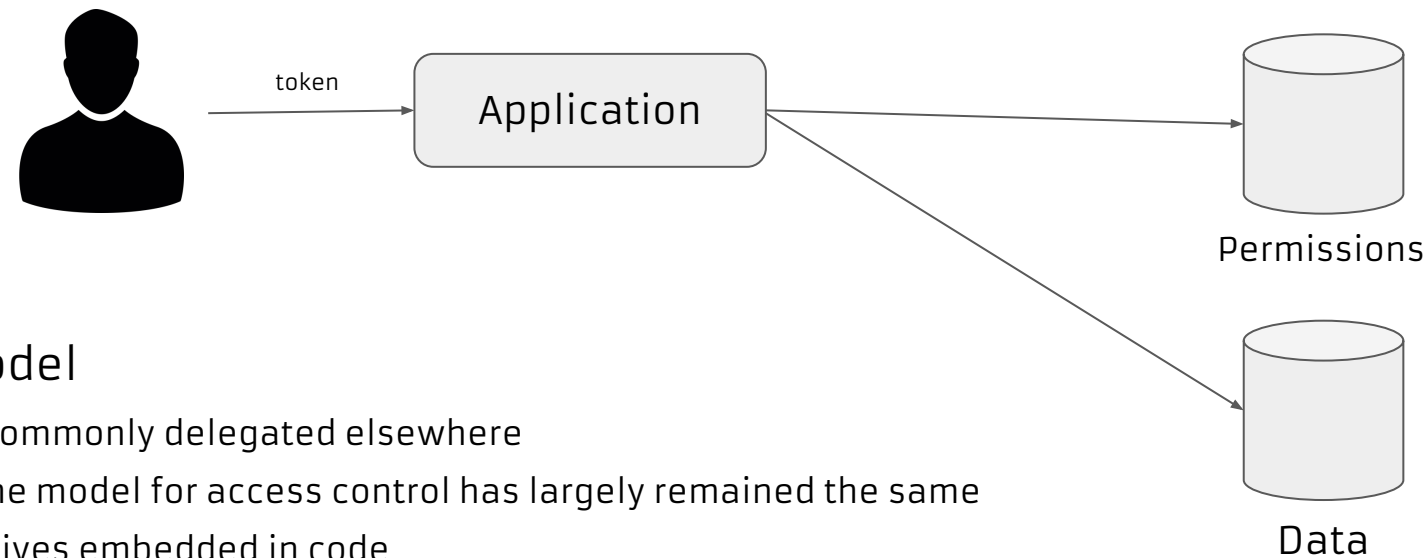
The evolution of identity



The monolith model

- The application handles everything - including authentication and access control
- The user authenticates in the application, a session is established
- Access control performed in code, commonly with a database serving permission data

The evolution of identity



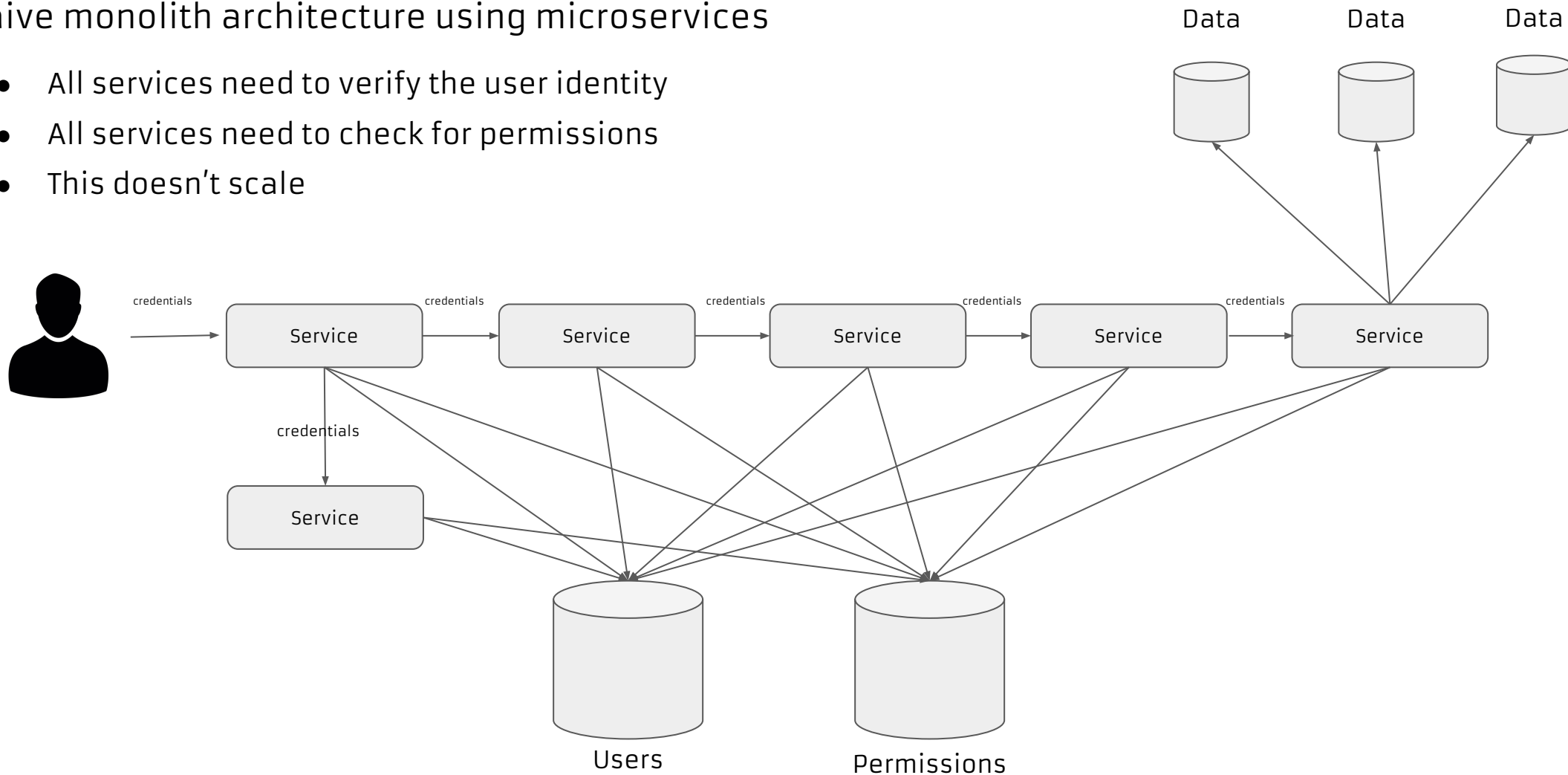
The distributed model

- Authentication is commonly delegated elsewhere
- But up until now, the model for access control has largely remained the same
- Authorization still lives embedded in code
- Permissions retrieved from external data sources
- This model scales poorly, and is difficult to manage

The evolution of identity

Naive monolith architecture using microservices

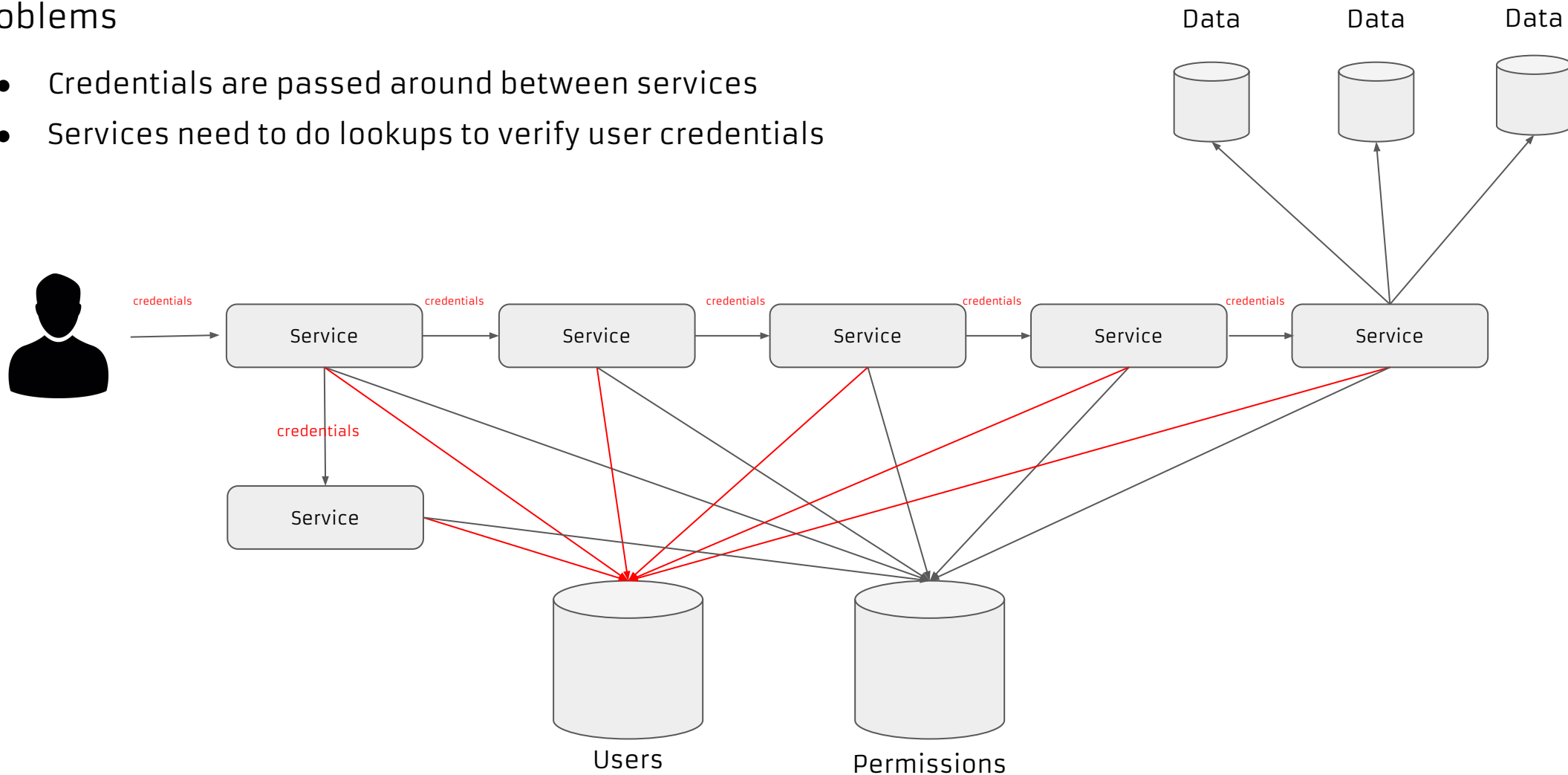
- All services need to verify the user identity
- All services need to check for permissions
- This doesn't scale



The evolution of identity

Problems

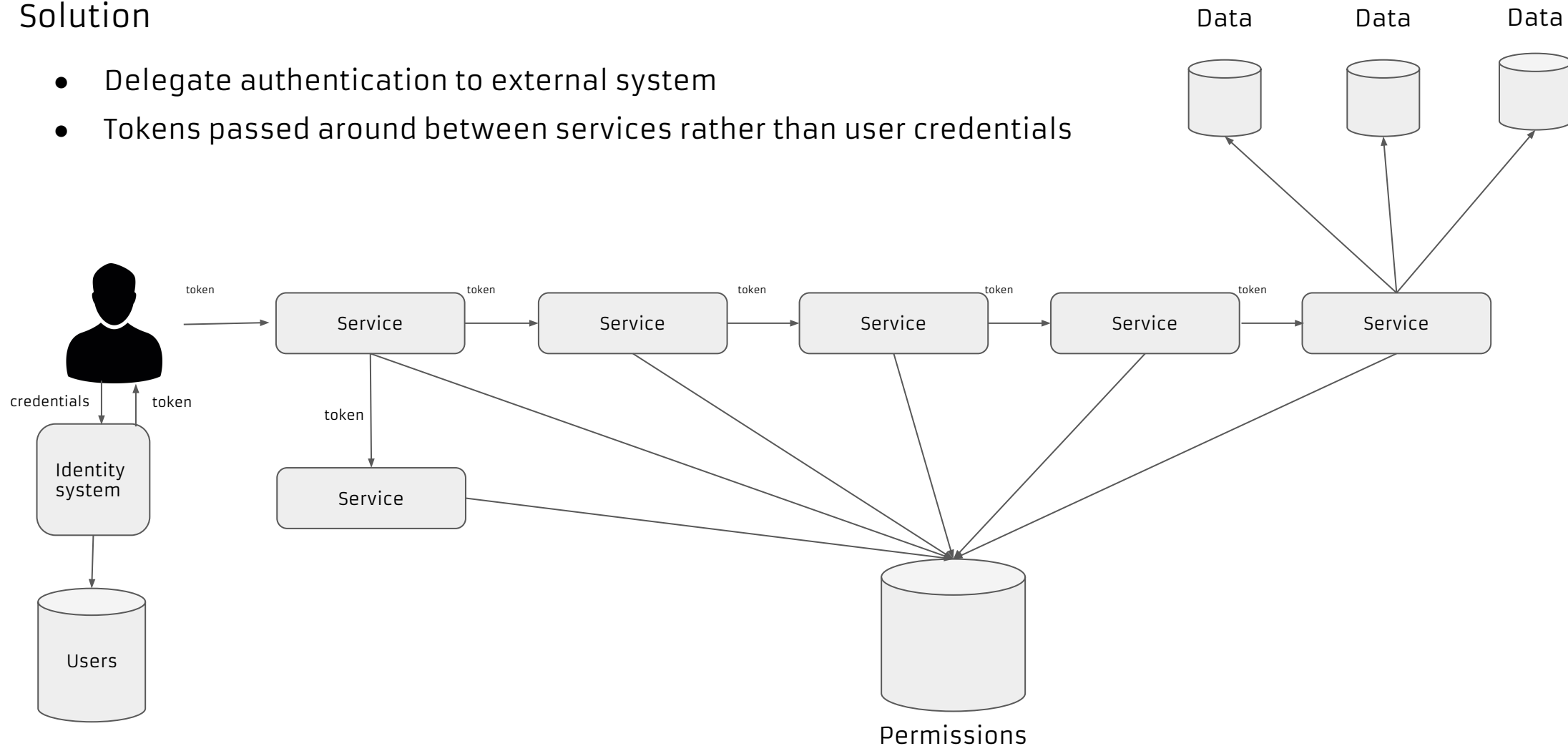
- Credentials are passed around between services
- Services need to do lookups to verify user credentials



The evolution of identity

Solution

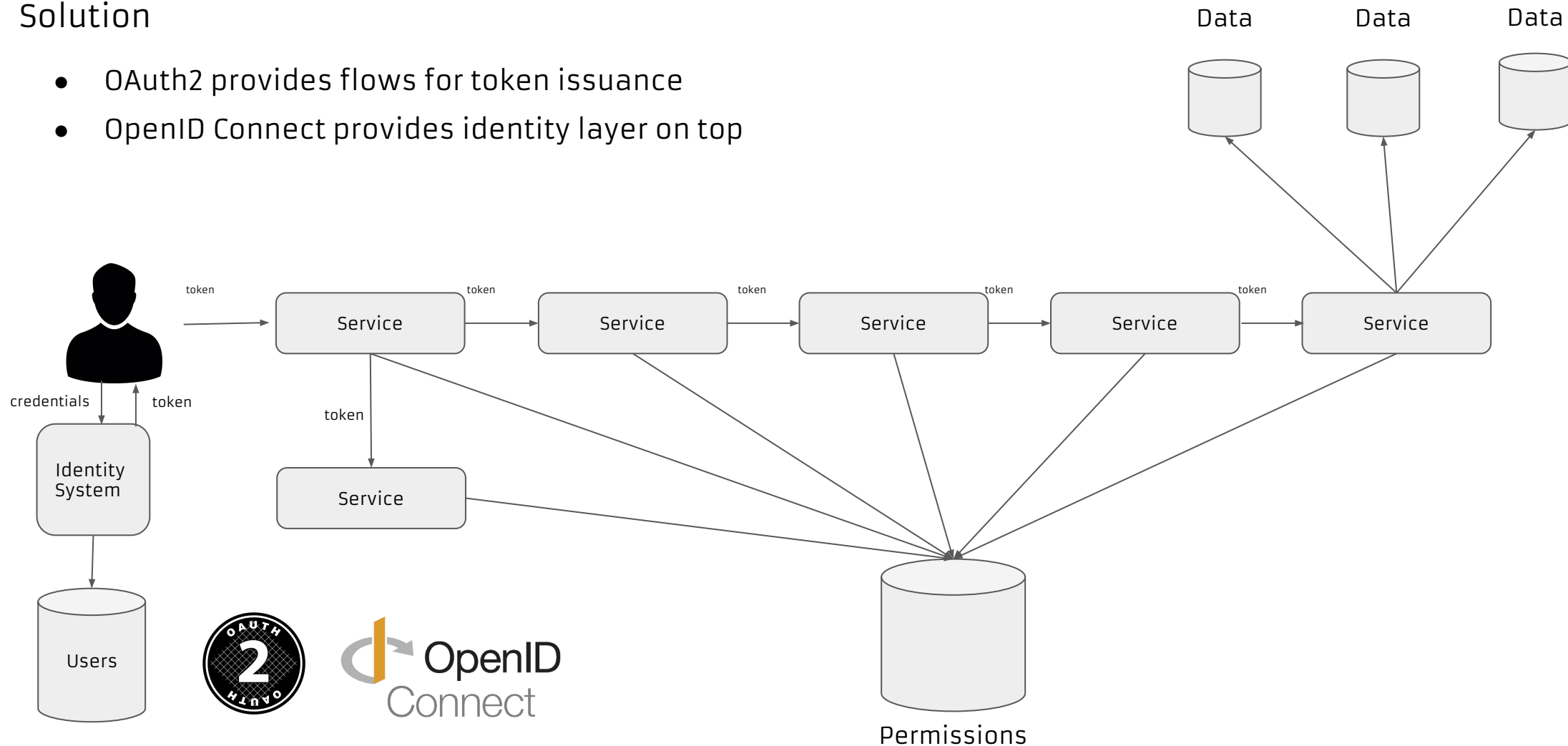
- Delegate authentication to external system
- Tokens passed around between services rather than user credentials



The evolution of identity

Solution

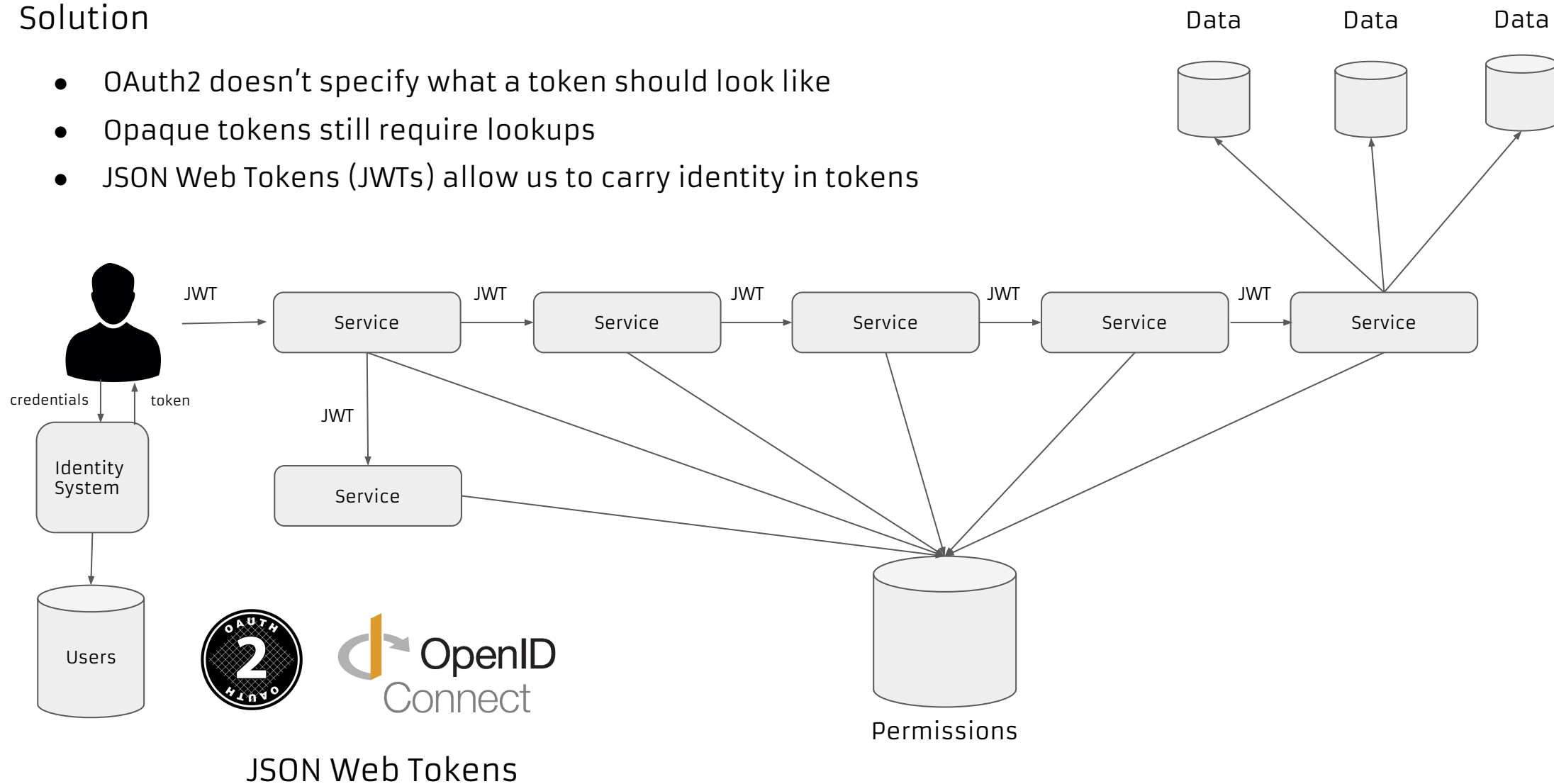
- OAuth2 provides flows for token issuance
- OpenID Connect provides identity layer on top



The evolution of identity

Solution

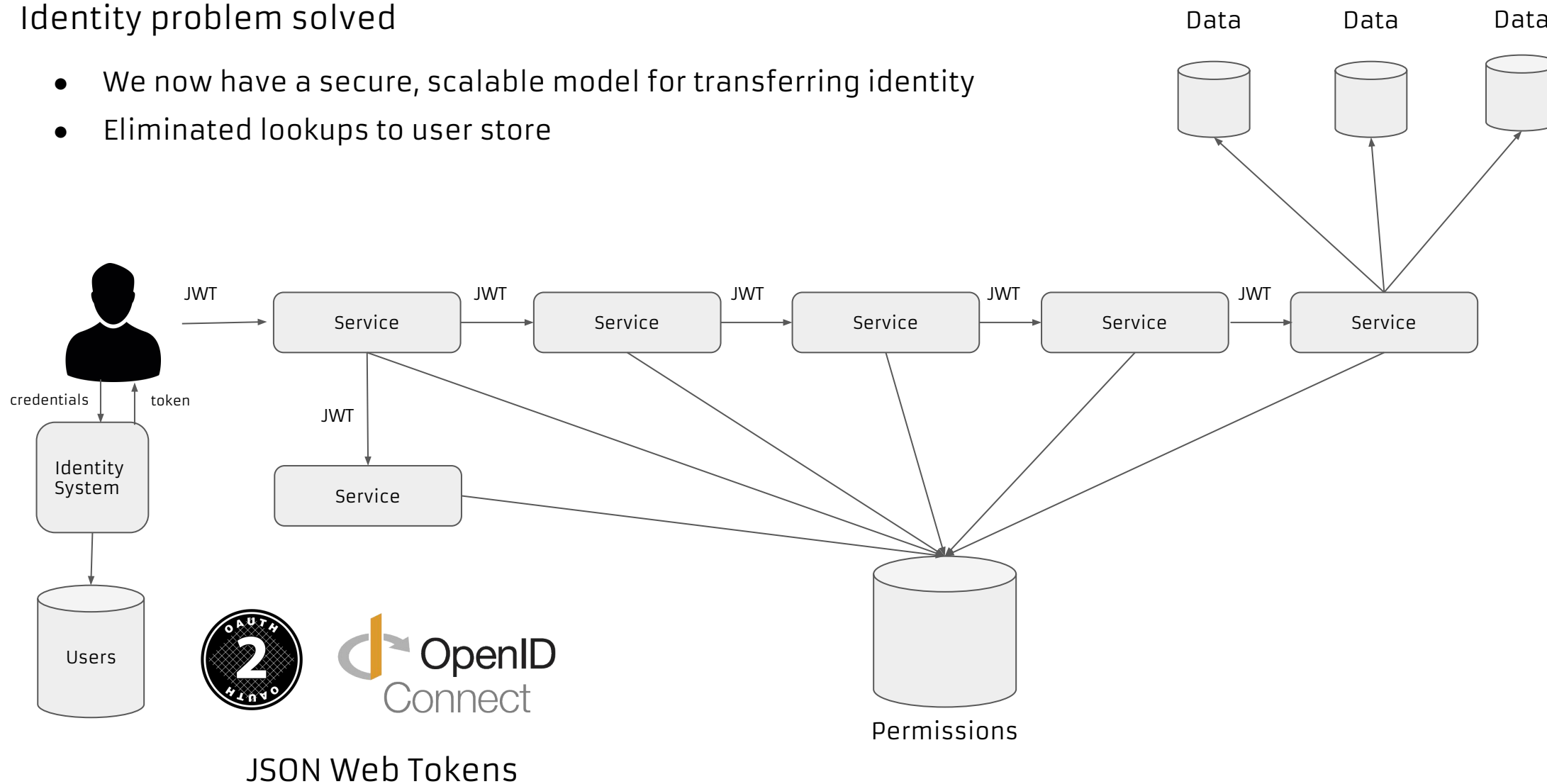
- OAuth2 doesn't specify what a token should look like
- Opaque tokens still require lookups
- JSON Web Tokens (JWTs) allow us to carry identity in tokens



The evolution of identity

Identity problem solved

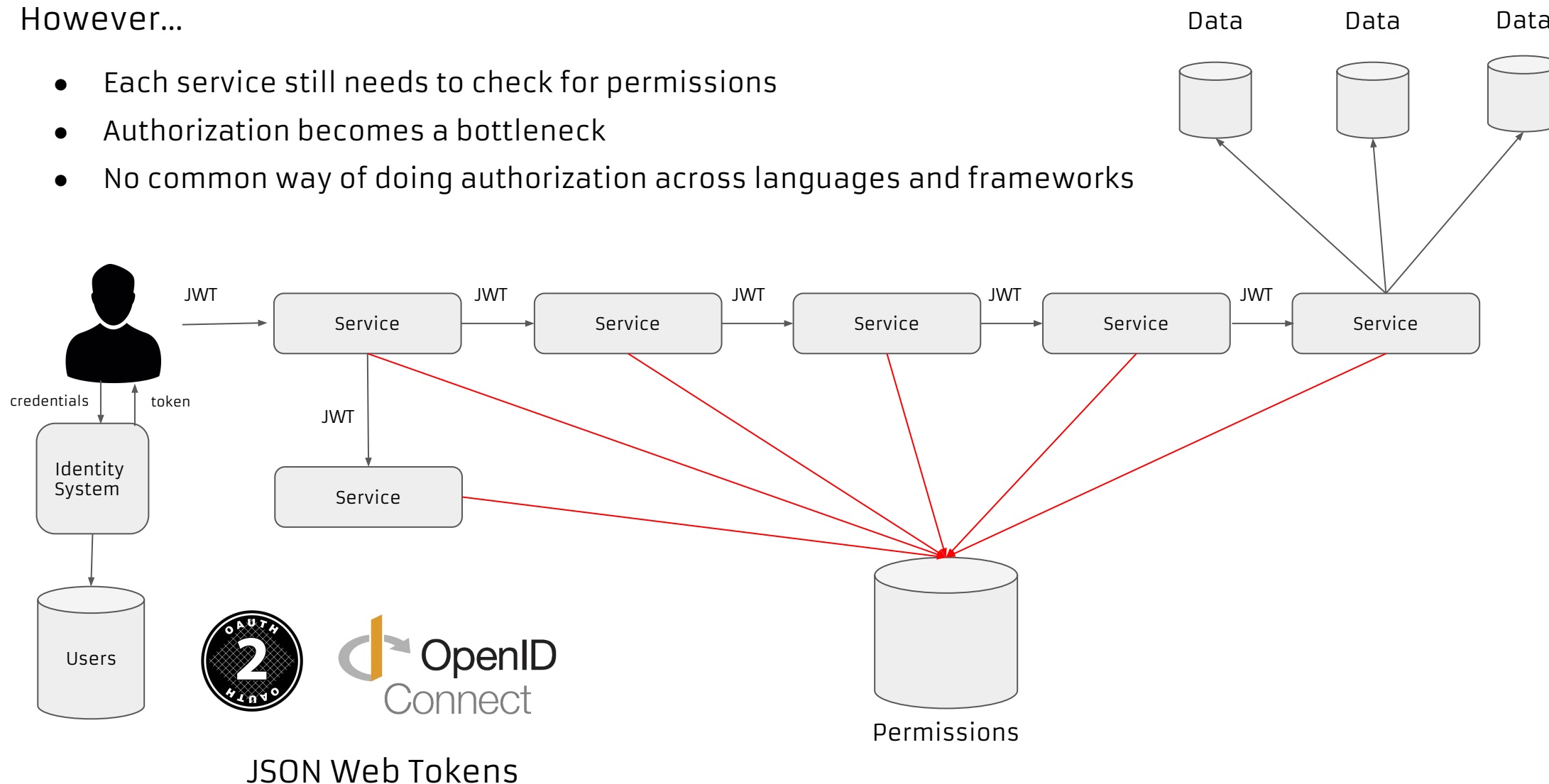
- We now have a secure, scalable model for transferring identity
- Eliminated lookups to user store



The evolution of identity

However...

- Each service still needs to check for permissions
- Authorization becomes a bottleneck
- No common way of doing authorization across languages and frameworks



What we want

- In order to **minimize latency**, authorization decisions should be made as **close to the application as possible**
- **Avoid** calling external permission database in each service - **expensive** and risks creating a **bottleneck**
- **Decouple** authorization code from application and business logic
 - Modern microservice architectures are **heterogeneous** - many languages and frameworks
 - **Updates** to authorization policies should be deployable independently of application lifecycle
 - Should be possible to **share** authorization policies across teams and services



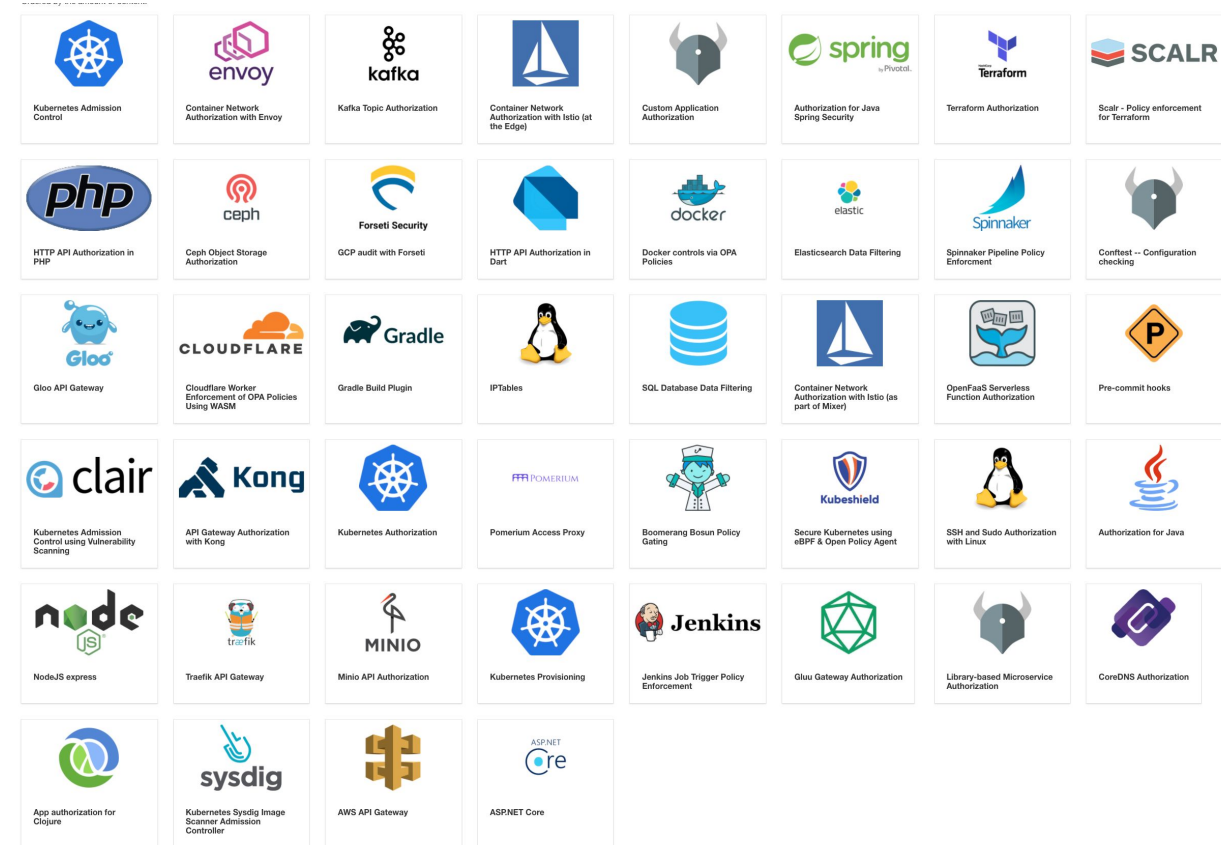
Open Policy Agent

- Open source general purpose **policy engine**
- **Decouples** policy from application / business logic
- Decouples policy decisions from actual **enforcement**
- **Unified** way of dealing with policies across the stack
- **Policies** written in declarative **Rego** language
- **Use cases** ranging from microservice authorization, kubernetes admission control, data source filtering, to CI/CD pipeline policies and much more



Vibrant Community

- 200+ Contributors
- 50+ Integrations
- 5800+ GitHub stars
- 5000+ Slack users
- 100+ million Docker image pulls
- Ecosystem including **Conftest**, **Gatekeeper**, **VS Code** and **IntelliJ IDEA** editor plugins.



Production Users

NETFLIX



Goldman
Sachs



ATLASSIAN



CLOUDFLARE®



intuit®



Kelsey Hightower 
@kelseyhightower

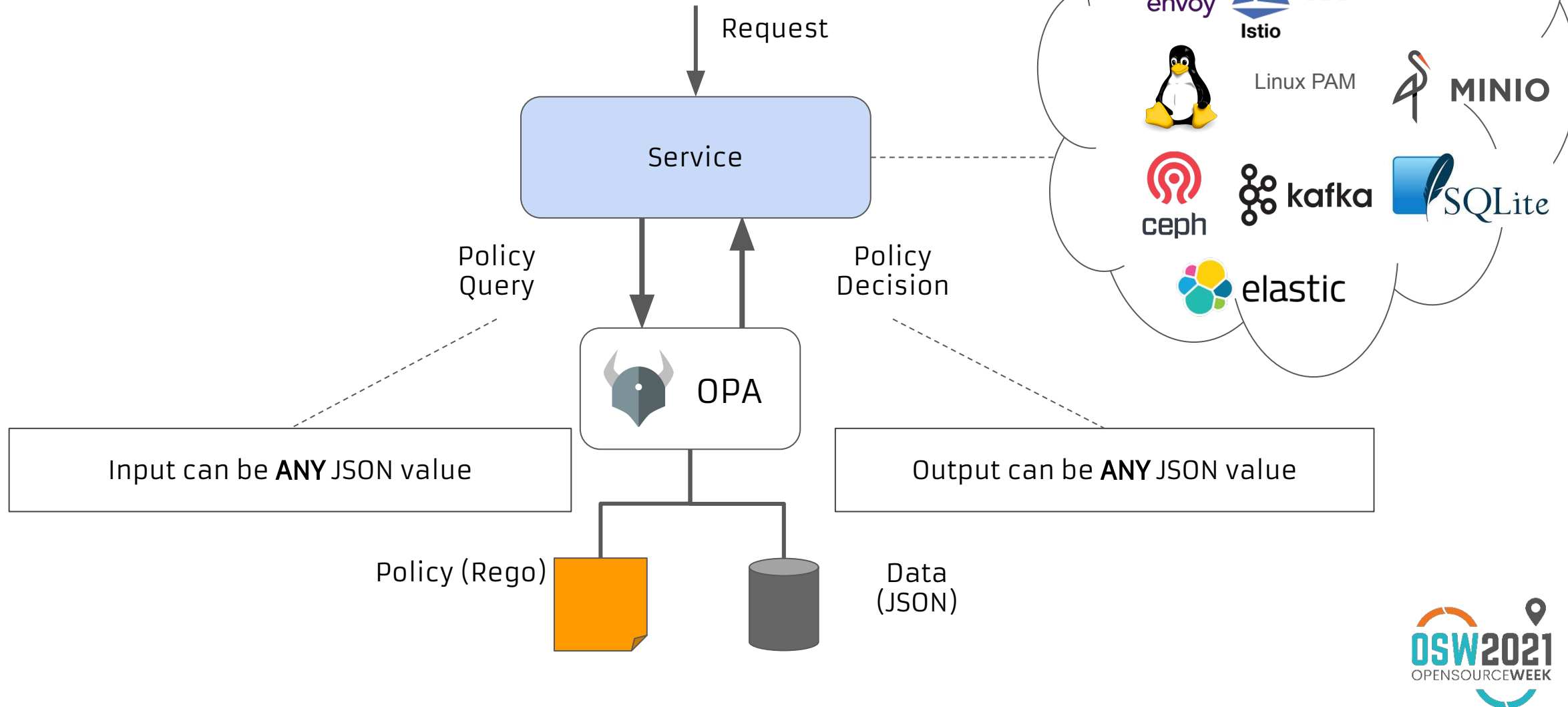


The Open Policy Agent project is super dope! I finally have a framework that helps me translate written security policies into executable code for every layer of the stack.



So, how does it work?

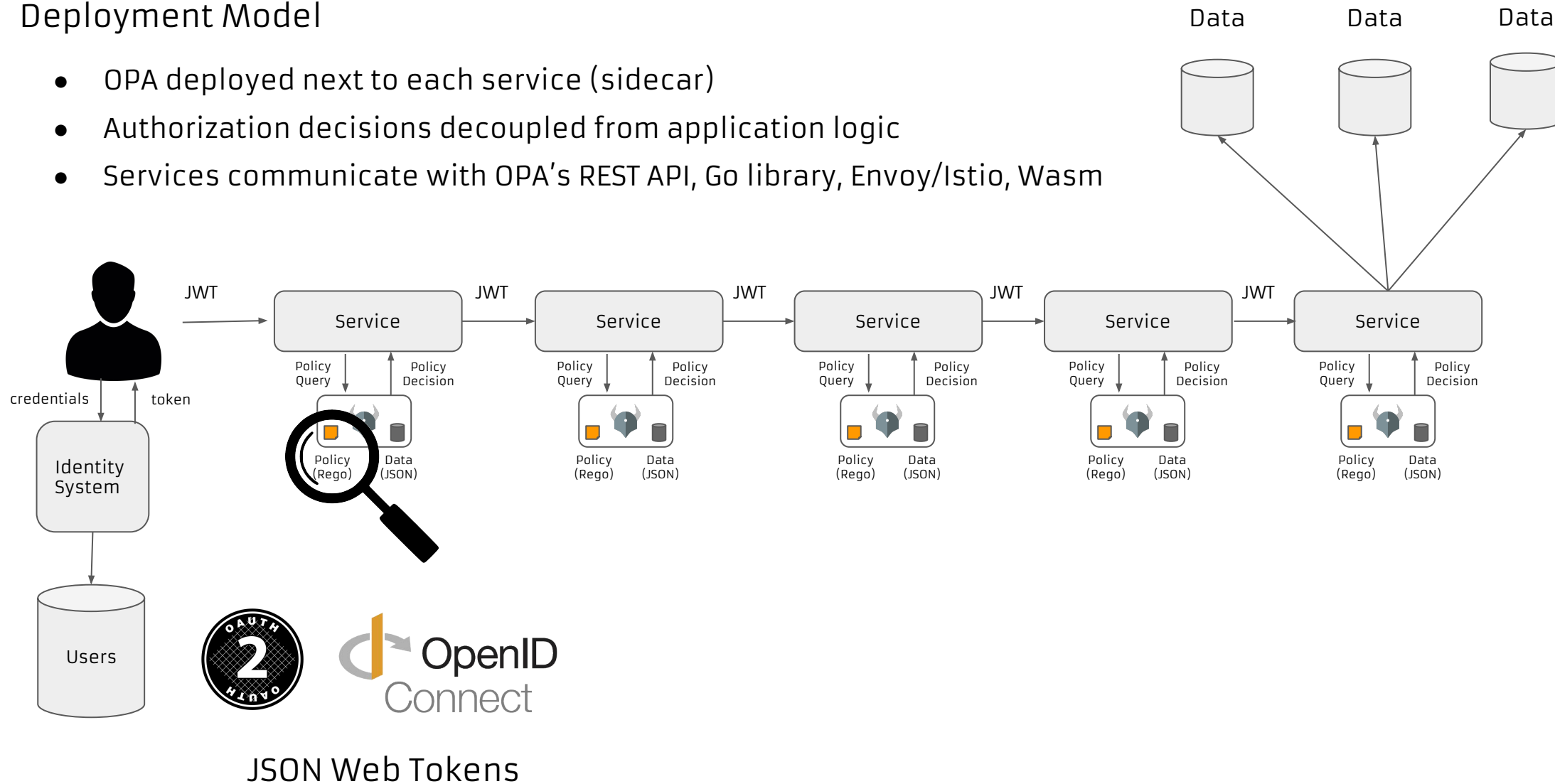
Policy Decision Model



Distributed Authorization With OPA

Deployment Model

- OPA deployed next to each service (sidecar)
- Authorization decisions decoupled from application logic
- Services communicate with OPA's REST API, Go library, Envoy/Istio, Wasm



- Declarative high-level policy language used by OPA
- Policy consists of any number of rules
- Rules commonly return true/false but may return any type available in JSON, like strings, lists and objects
- 150+ built-in functions to help with policy authoring
- Policy testing made easy with provided unit test framework
- Well documented
- [Rego Playground](#) - try it out!



```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2     "request": {
3         "method": "PUT",
4         "path": "/users/bob"
5     },
6     "user": {
7         "name": "alice",
8         "team": "infra1",
9         "roles": ["infra", "kube-admin", "postgres-dba"]
10    }
11 }
```

DATA

OUTPUT

```
1
2
3
```

```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2
3     "request": {
4         "method": "PUT",
5         "path": "/users/bob"
6     },
7     "user": {
8         "name": "alice",
9         "team": "infra1",
10        "roles": ["infra", "kube-admin", "postgres-dba"]
11    }
12 }
```

DATA

OUTPUT

```
1
2
3
```

```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10    input.request.method == "PUT"
11
12    path_split := split(input.request.path, "/")
13
14    main_path := path_split[1]
15    main_path == "users"
16
17    user_path := path_split[2]
18    user_path == input.user.name
19 }
```

INPUT

```
1 {
2   "request": {
3     "method": "PUT",
4     "path": "/users/bob"
5   },
6   "user": {
7     "name": "alice",
8     "team": "infra1",
9     "roles": ["infra", "kube-admin", "postgres-dba"]
10  }
11 }
```

DATA

OUTPUT

```
1
2
3
```

```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2     "request": {
3         "method": "PUT",
4         "path": "/users/bob"
5     },
6     "user": {
7         "name": "alice",
8         "team": "infra1",
9         "roles": ["infra", "kube-admin", "postgres-dba"]
10    }
11 }
```

DATA

OUTPUT

```
1
2
3
```



```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2
3     "request": {
4         "method": "PUT",
5         "path": "/users/bob"
6     },
7     "user": {
8         "name": "alice",
9         "team": "infra1",
10        "roles": ["infra", "kube-admin", "postgres-dba"]
11    }
12 }
```

DATA

OUTPUT

```
1
2
3
```

```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2
3     "request": {
4         "method": "PUT",
5         "path": "/users/bob"
6     },
7     "user": {
8         "name": "alice",
9         "team": "infra1",
10        "roles": ["infra", "kube-admin", "postgres-dba"]
11    }
12 }
```

DATA

OUTPUT

```
1
2
3
```

```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2     "request": {
3         "method": "PUT",
4         "path": "/users/bob"
5     },
6     "user": {
7         "name": "alice",
8         "team": "infra1",
9         "roles": ["infra", "kube-admin", "postgres-dba"]
10    }
11 }
```

DATA

OUTPUT

```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2     "request": {
3         "method": "PUT",
4         "path": "/users/bob"
5     },
6     "user": {
7         "name": "alice",
8         "team": "infra1",
9         "roles": ["infra", "kube-admin", "postgres-dba"]
10    }
11 }
```

DATA

OUTPUT

```
1
2
3
```

```
1 package play
2
3 default allow = false
4
5 allow {
6     input.user.roles[_] == "user-admin"
7 }
8
9 allow {
10     input.request.method == "PUT"
11
12     path_split := split(input.request.path, "/")
13
14     main_path := path_split[1]
15     main_path == "users"
16
17     user_path := path_split[2]
18     user_path == input.user.name
19 }
```

INPUT

```
1 {
2     "request": {
3         "method": "PUT",
4         "path": "/users/bob"
5     },
6     "user": {
7         "name": "alice",
8         "team": "infra1",
9         "roles": ["infra", "kube-admin", "postgres-dba"]
10    }
11 }
```

DATA

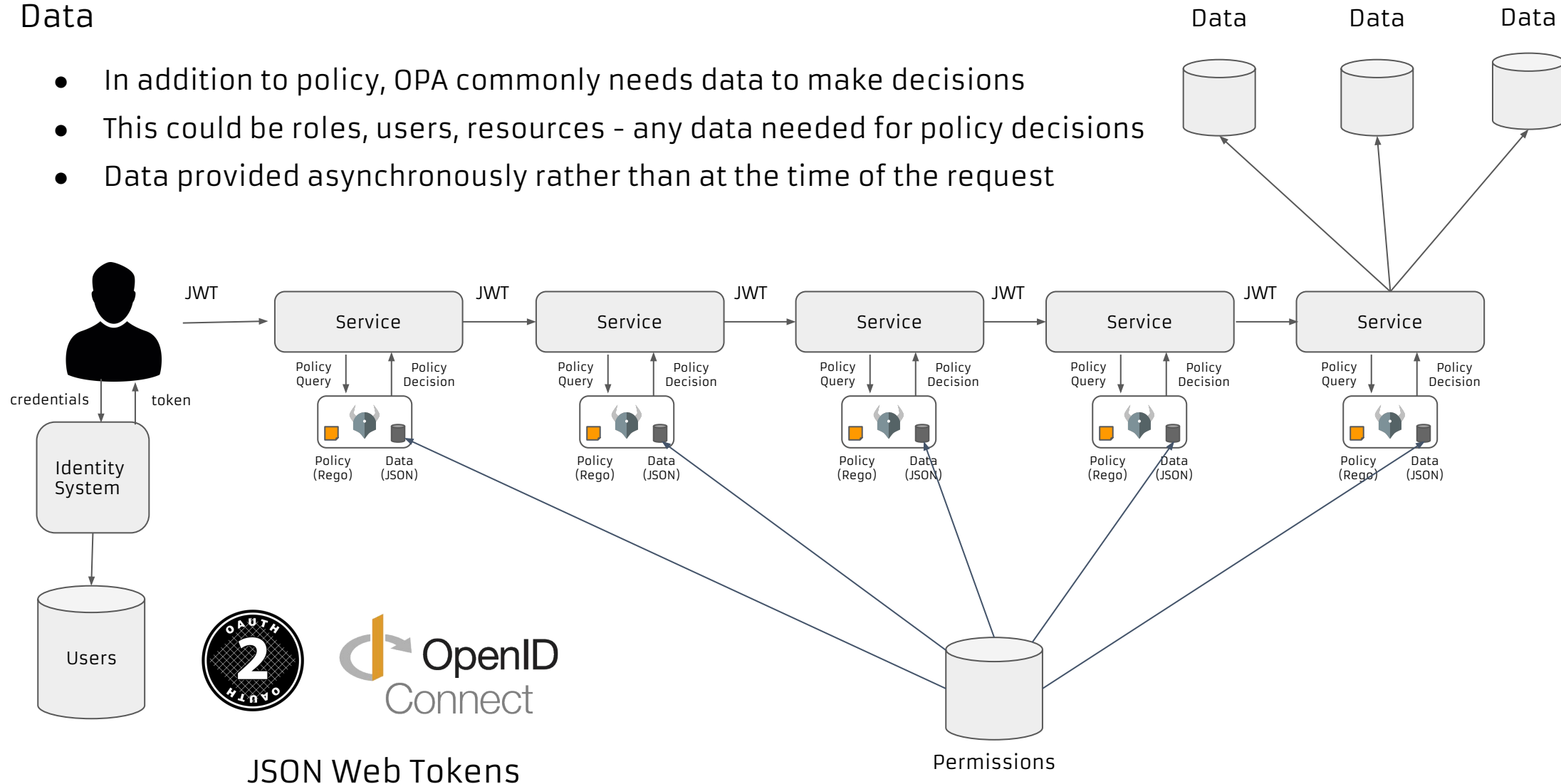
OUTPUT

```
Found 1 result in 115.465 µs.
1 {
2     "allow": false
3 }
```

Distributed Authorization With OPA

Data

- In addition to policy, OPA commonly needs data to make decisions
- This could be roles, users, resources - any data needed for policy decisions
- Data provided asynchronously rather than at the time of the request



Getting Started with OPA

1. **Start small** – write a few simple policies and tests
2. Browse the **OPA documentation**. Get a feel for the basics and the built-ins
3. Consider possible applications near to you - previous apps and libraries you've worked with. **Consider the informal policies it dealt with**
4. **Delegate policy responsibilities to OPA**. Again, start small! Perhaps a single endpoint to begin somewhere. Deploy and build experience
5. **Scale up** - management capabilities, logging, bundle server
6. [Styra Academy](#)
7. Join the [OPA Slack](#) community





Thank you!



#OSW2021



<http://www.reteitalianaopensource.net>



RETE ITALIANA
OPEN SOURCE