ORY / summit-22

Patrik Neu &
Henning Perl

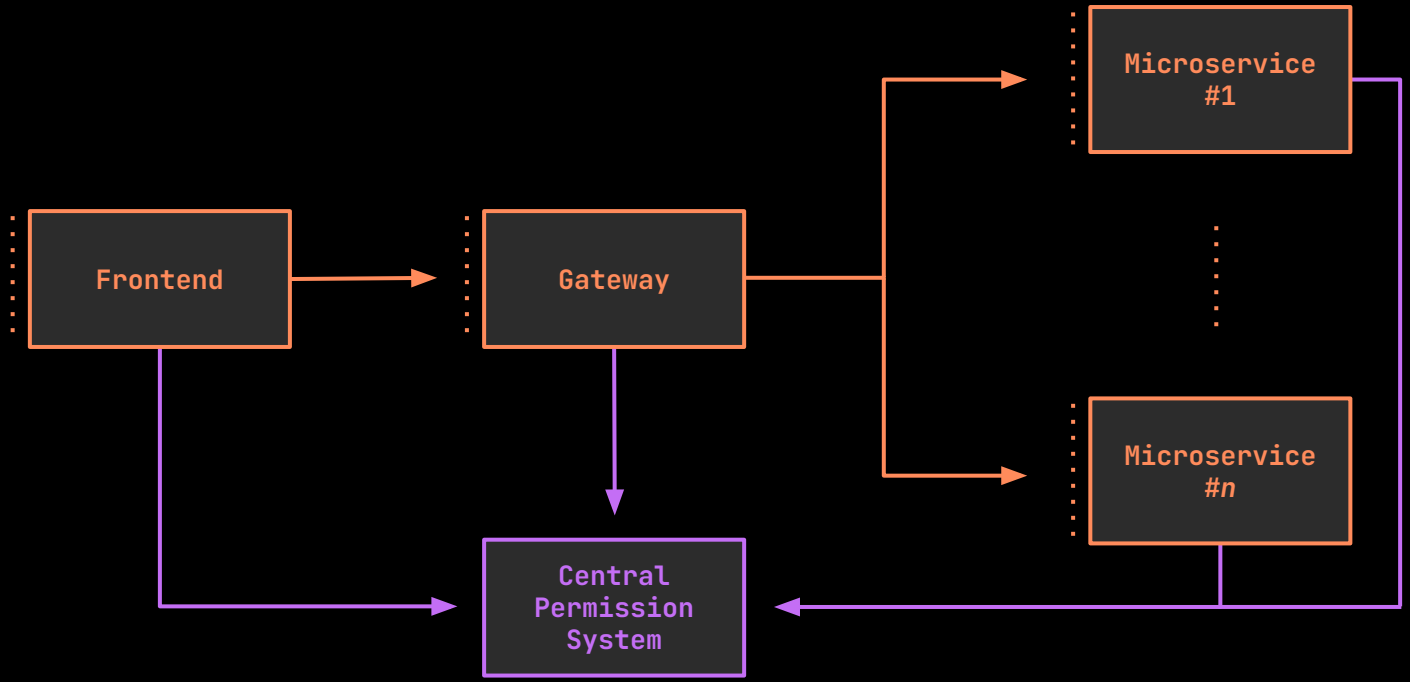
Engineers @ Ory

# Making of the Ory Permission Language

{patrik,henning}@ory.sh

October 20th 2022

# Why Use a Globally Distributed Permission System?




Single source of truth

# Powerful Permission Model

\# Go from explicit permissions
(*Bob can edit the document ‘secrets’*)
to relationships and derived permissions

| | | |
|---|---|---|
| | Relationship | Bob is owner of the document ‘secrets’ |
| + | Rule | Any owner of a document can edit it |
| ⟹ | Permission | Bob can edit ‘secrets’ |

⇝ Ory Permission Language expresses these rules

# Our Journey to the Ory Permission Language

#1 Requirements & Hypotheses

#2 User Interviews

#3 Final Design

#4 Implementation

#5 Launch ⬸ you are here :)

# Requirements & Hypotheses

No-one will configure permission systems full-time

- Self-explanatory, familiar
- Easy to modify with high confidence
- Good editor support
- Testing and static correctness checks

- *Not* Turing-complete
  - ⇝ safe to parse and execute without sandbox
  - ⇝ parsing in linear time

# User Interviews

## Setup

- ~1h guided interviews with community members
- Modelling permission schema in different languages

## Scenario

- *Documents* have *owners*, *editors*, or *viewers*
- An *owner* is always also an *editor*
- You can put documents in folders
- If you can view the folder, you can view the contained documents

# #1 Original Zanzibar

Takeaways:
- Verbose
- confusing

```
name: "document"
    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"
                }
            } }
        }
    }
}
```

# #2 Pythonesque

```
type document
    relations
        define parent as self
        define owner as self
        define editor as self or owner
        define viewer as self or editor or viewer from parent
```

Takeaways:

- “as self” confusing
- missing type safety
- concise is not better

# #3 Typed & Logical

```
type user {}
type document {
    relation owner, editor, viewer: user

    relation parent: document
}

for all document:d, user:u {
    u is owner of d => u is editor of d
    u is editor of d => u is viewer of d
}
for all document:d, document:p, user:u {
    p is parent of d & u is viewer of p
      => u is viewer of d
}
```

Takeaways:

- types are nice
- people either love or hate logic

# #4 Typed & Declarative

```
definition user {}
definition document {
    relation viewer: user
    relation editor: user
    relation owner: user

    relation parent: document

    permission view = owner or editor or
        viewer or parent->viewer
    permission edit = owner or editor
    permission own = owner
}
```

**Takeaways:**

- types are nice
- confusing syntax: parent→viewer

# Key Takeaways from the Interviews

- Types help guide the user
- Transitive rules were hard to grasp
- None of the languages felt familiar
- Hard to reason about the implications

Back to the drawing board: Use TypeScript!

- Self-explanatory, familiar ✓
- Easy to modify with high confidence ✓
- Good editor support ✓
- Unit test frameworks and linters ✓

# Example of the Ory Permission Language

```
// You define standard classes
class File implements Namespace {
  related: {
    parents: (File | Folder)[]
    viewers: (User | SubjectSet<Group, "members">)[]
    owners: (User | SubjectSet<Group, "members">)[]
  }

  permits = {
    view: (ctx: Context): boolean =>
      this.related.parents.traverse((p) => p.permits.view(ctx)) ||
      this.related.viewers.includes(ctx.subject) ||
      this.related.owners.includes(ctx.subject),

    edit: (ctx: Context) => this.related.owners.includes(ctx.subject),
  }
}
```

# Implementation

One last requirement:

\# *Not* Turing-complete

⇝ Custom parser in Go that accepts a subset of TypeScript

⇝ Outputs an abstract syntax tree similar to the original Zanzibar language

# Launch

The Ory Permission Language is now available in

- Ory Keto
- Ory Network through CLI & Console

Start building & share your experience with us

# What's next?

- Testing framework ⟶ change with confidence
- Playground to experiment with the OPL
- Visualization of rules and lookups
- Custom language server
- Large-scale migrations

⟶ Contributions are welcome!