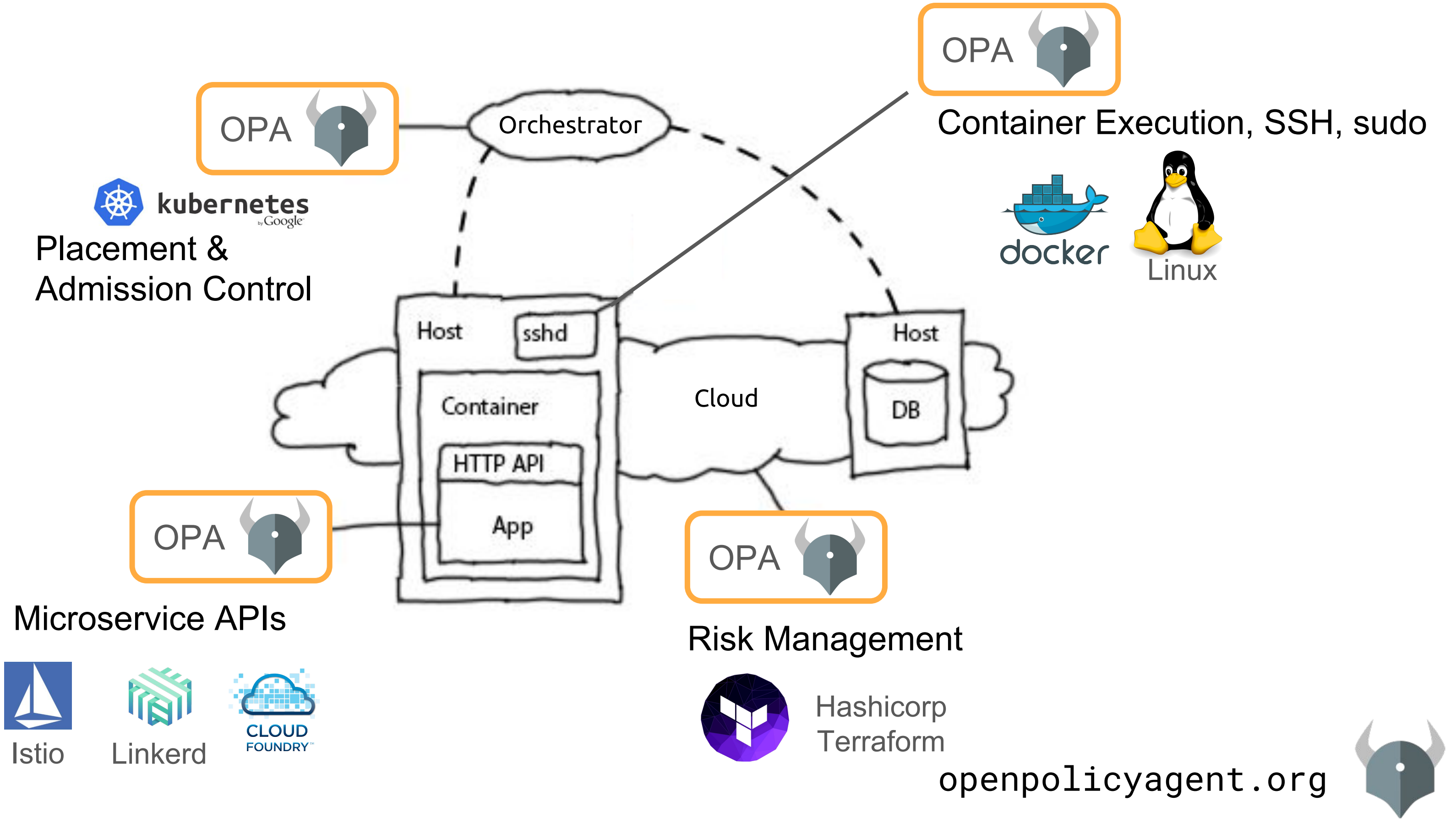
Open Policy Agent

Language Introduction

OPA: Add fine-grained policy to other projects



openpolicyagent.org

Use OPA to policy-enable your project

1 **Integrate**

Offload policy decisions from your project to OPA

2 **Author**

Write OPA policies that make decisions

3 **Manage**

Deploy OPA, retrieve policy, audit decisions, monitor health

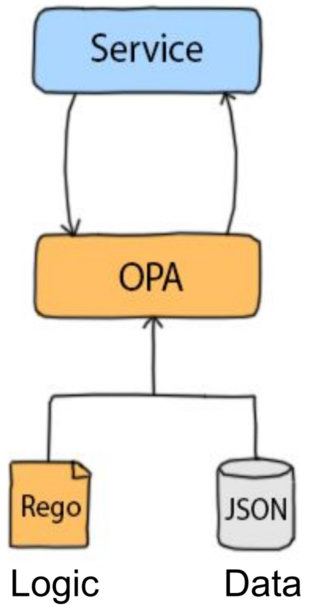
Agenda

- How Policies are Invoked
- Simple Policies
- Policies with Iteration
- Additional Topics
    - Modularity
    - Negation
    - Any/All
    - Non-boolean Decisions

How Policies are Invoked

- Overview
- Example:
  - HTTP API Authorization

How Policies are Invoked



How Policies are Invoked

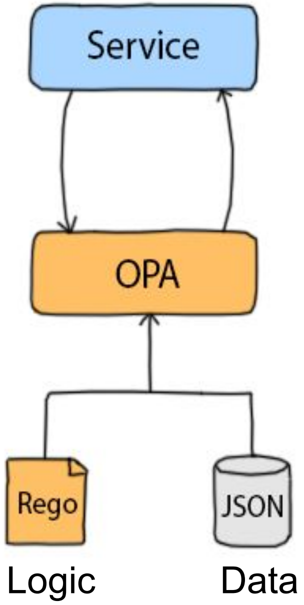
1. Decision Request

```
POST v1/data/<policy-name>

{“input”: <JSON>}
```

Any JSON value:

- `"alice"`
- `["api", "v1", "cars"]`
- `{"headers": {...}}`



How Policies are Invoked

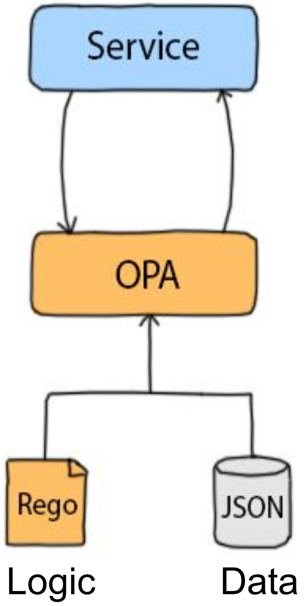
1. Decision Request

```
POST v1/data/<policy-name>

{“input”: <JSON>}
```

Any JSON value:

- `"alice"`
- `["api", "v1", "cars"]`
- `{"headers": {...}}`



2. Decision Response

```
200 OK

{“result”: <JSON>}
```

Any JSON value:

- `true, false`
- `"bob"`
- `{"servers": ["server-001", …]}`

openpolicyagent.org

How Policies are Invoked

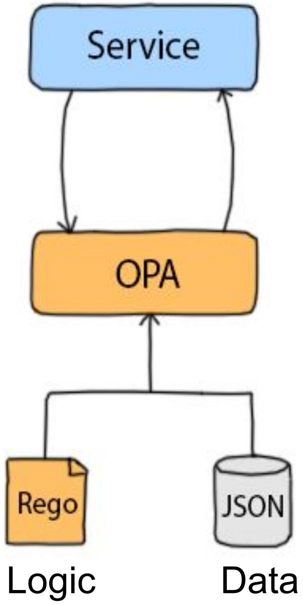
1. Decision Request

```
POST v1/data/<policy-name>

{“input”: <JSON>}
```

Any JSON value:

- `"alice"`
- `["api", "v1", "cars"]`
- `{"headers": {...}}`



2. Decision Response

```
200 OK

{“result”: <JSON>}
```

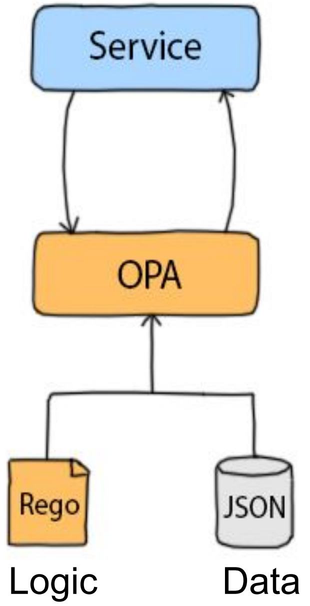
Any JSON value:

- `true, false`
- `"bob"`
- `{"servers": ["server-001", …]}`

Input is JSON. Policy decision is JSON.

openpolicyagent.org

Example: HTTP API Authorization

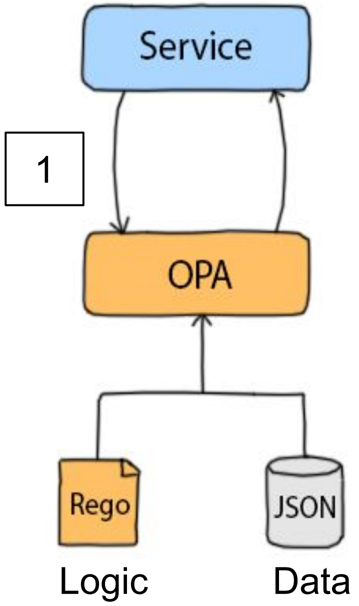


Example: HTTP API Authorization

1. Example Request to OPA

```
POST v1/data/http/authz/allow

{"input": {
    "method": "GET",
    "path":   ["finance", "salary", "alice"],
    "user":   "bob"}}
```



openpolicyagent.org

Example: HTTP API Authorization

1. Example Request to OPA

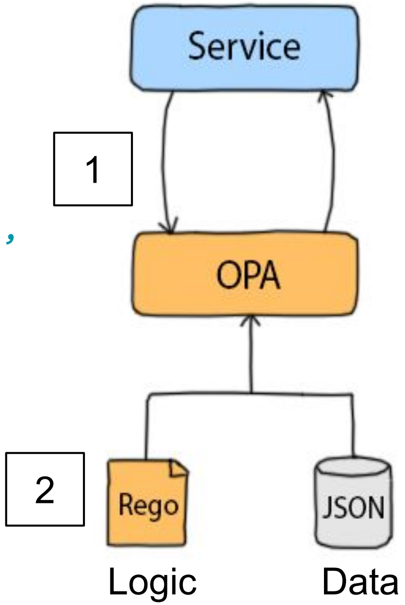
```
POST v1/data/http/authz/allow

{"input": {
    "method": "GET",
    "path":   ["finance", "salary", "alice"],
    "user":   "bob"}}
```

2. Example Policy in OPA

```
package http.authz

allow {
  input.user == “bob”
}
```



openpolicyagent.org

Example: HTTP API Authorization

1. Example Request to OPA

```
POST v1/data/http/authz/allow

{"input": {
    "method": "GET",
    "path":   ["finance", "salary", "alice"],
    "user":   "bob"}}
```

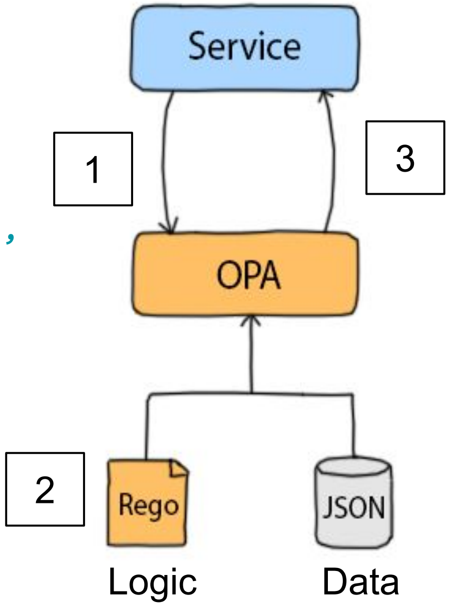
2. Example Policy in OPA

```
package http.authz

allow {
  input.user == “bob”
}
```

3. Example Response from OPA

```
{“result”: true}
```



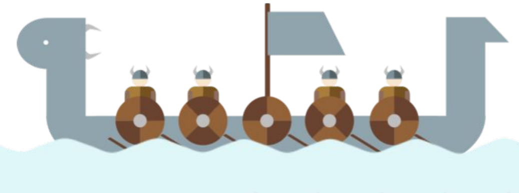
openpolicyagent.org

Agenda

- How Policies are Invoked
- **Simple Policies**
- Policies with Iteration
- Additional Topics
  - Modularity
  - Negation
  - Non-boolean Decisions

Simple Policies

- Lookup values
- Compare values
- Assign values
- Create rules
- Create functions
- Use context (data)



Lookup and Compare Values

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Lookup values.

```
input.method

input.path[0]
```

Lookup and Compare Values

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Lookup values. Compare values.

```
input.method == “GET”

input.path[0] == “finance”

input.user != input.method
```

Lookup and Compare Values

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Lookup values. Compare values.

```
input.method == “GET”

input.path[0] == “finance”

input.user != input.method

startswith(input.path[1], “sal”)

count(input.path) > 2
```

See 50+ operators documented at openpolicyagent.org/docs/language-reference.html

Assign Values to Variables

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Assign variables.

```
path := input.path
```

Use variables like input.

```
path[2] == "alice"
```

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules have a Head and a Body.

```
allow = true {
  input.method == "GET"
  input.user == "bob"
}
```

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules have a Head and a Body.

```
allow = true {
  input.method == "GET"
  input.user == "bob"
}
```

Rule Head

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules have a Head and a Body.

```
allow = true {
  input.method == "GET"
  input.user == "bob"
}
```

Rule Head

Name	`allow`
Value	`true`

openpolicyagent.org

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules have a Head and a Body.

```
allow {
  input.method == "GET"
  input.user == "bob"
}
```

Rule Head

Name	`allow`
Value	`true`

openpolicyagent.org

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules have a Head and a Body.

```
allow {
  input.method == "GET"
  input.user == "bob"
}
```

Rule Body

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules have a Head and a Body.

```
allow {
  input.method == "GET"
  input.user == "bob"
}
```

Rule Body

Multiple statements
in rule body
are ANDed together.

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules have a Head and a Body.

```
allow {
  input.method == "GET"
  input.user == "bob"
}
```

Rule Body

Multiple statements
in rule body
are ANDed together.

`allow` is true IF
`input.method` equals "`GET`" AND
`input.user` equals "`bob`"

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Multiple rules with same name.

```
allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

Create Rules

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Multiple rules with same name.

```
allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

Rule Head

Multiple statements
with same head
are ORed together.

openpolicyagent.org

Create Rules

Input

```
{
  "method": "POST",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Rules can be undefined.

```
allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

Create Rules

Input

```
{
  "method": "POST",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Different method.
"POST" instead of "GET"

Rules can be undefined.

```
allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

Create Rules

Input

```
{
  "method": "POST",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Different method.
"POST" instead of "GET"

Rules can be undefined.

```
allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

Neither rule matches.
allow is undefined (*not false!*)

openpolicyagent.org

Create Rules

Input

```
{
  "method": "POST",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Use default keyword.

```
default allow = false

allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

openpolicyagent.org

Create Rules

Input

```
{
  "method": "POST",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

default <name> = <value>
If no rules match
default value is returned.

Use default keyword.

```
default allow = false

allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

Create Rules

Input

```
{
  "method": "POST",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Use default keyword.

```
default allow = false

allow {
  input.method == "GET"
  input.user == "bob"
}

allow {
  input.method == "GET"
  input.user == input.path[2]
}
```

default <name> = <value>
If no rules match
default value is returned.

at most one default per rule set

openpolicyagent.org

Create Functions

Input

```
{
  "method": "GET",
  "path":   "/finance/salary/alice",
  "user":   "bob"
}
```

Path is a string now.

Create Functions

Input

```
{
  "method": "GET",
  "path":   "/finance/salary/alice",
  "user":   "bob"
}
```

Path is a string now.

Example rule

```
default allow = false

allow {
  trimmed := trim(input.path, "/")
  path := split(trimmed, "/")
  path = ["finance", "salary", user]
  input.user == user
}
```

Create Functions

Input

```
{
  "method": "GET",
  "path":   "/finance/salary/alice",
  "user":   "bob"
}
```

Path is a string now.

Example rule

```
default allow = false

allow {
  trimmed := trim(input.path, "/")
  path := split(trimmed, "/")
  path = ["finance", "salary", user]
  input.user == user
}
```

Avoid duplicating common logic like string manipulation

openpolicyagent.org

Create Functions

Input

```
{
  "method": "GET",
  "path":   "/finance/salary/alice",
  "user":   "bob"
}
```

Path is a string now.

Avoid duplicating common logic like string manipulation

Put common logic into functions

```
default allow = false

allow {
  path := split_path(input.path)
  path = ["finance", "salary", user]
  input.user == user
}

split_path(str) = parts {
  trimmed := trim(str, "/")
  parts := split(trimmed, "/")
}
```

Create Functions

Input

```
{
  "method": "GET",
  "path":   "/finance/salary/alice",
  "user":   "bob"
}
```

Functions are Rules with arguments.

```
read_method(str) = true {
  str == "GET"
}

read_method(str) = true {
  str == "HEAD"
}
```

Create Functions

Input

```
{
  "method": "GET",
  "path":   "/finance/salary/alice",
  "user":   "bob"
}
```

Functions are Rules with arguments.

```
read_method(str) = true {
  str == "GET"
}

read_method(str) = true {
  str == "HEAD"
}
```

"Function" Head

Multiple statements
with same head
are ORed together.

openpolicyagent.org

Create Functions

Input

```
{
  "method": "GET",
  "path":   "/finance/salary/alice",
  "user":   "bob"
}
```

Functions are Rules with arguments.

```
read_method(str) {
  str == "GET"
}

read_method(str) {
  str == "HEAD"
}
```

"Function" Head

Multiple statements
with same head
are ORed together.

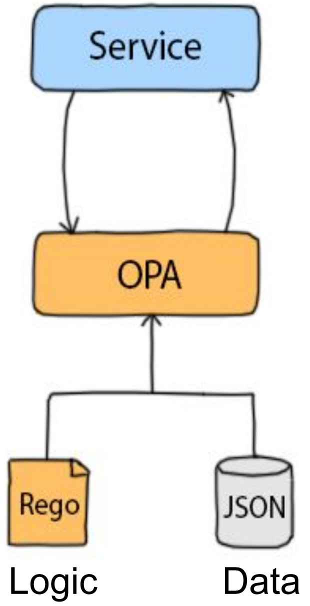
openpolicyagent.org

Policies can use Context from Outside World

Load Context/Data Into OPA

```
PUT v1/data/<path> HTTP/1.1
Content-Type: application/json

<JSON>
```



Policies Use Context

Input

```
{
  "method": "GET",
  "path":   ["finance", "salary", "alice"],
  "user":   "bob"
}
```

Data (context)

```
{
  "users": {
     "alice": {"department": "legal"},
     "bob":   {"department": "hr"},
     "janet": {"department": "r&d"}
  }
}
```

Policy

```
allow {
    # Users can access their own salary
    input.user == input.path[2]
}

allow {
    # HR can access any salary
    user := data.users[input.user]
    user.department == "hr"
}
```

openpolicyagent.org

Summary

Lookup values	input.path[1]
Compare values	"bob" == input.user
Assign values	user := input.user
Rules	<head> { <body> }
Rule Head	<name> = <value> { … } or <name> { … }
Rule Body	<statement-1>; <statement-2>; … (ANDed)
Multiple Rules *with same name*	<rule-1> OR <rule-2> OR ...
Default Rule Value	default <name> = <value>
Functions	Rules with arguments
Context	Reference with data. instead of input.

openpolicyagent.org

Agenda

- How Policies are Invoked
- Simple Policies
- **Policies with Iteration**
- Additional Topics
  - Modularity
  - Negation
  - Any/All
  - Non-boolean Decisions

Policies With Iteration

- Iteration
- Virtual documents
- Virtual documents vs Functions

What about Arrays?

Input

```
{
  "user":    "alice"
  "resource": "54cf10",
}
```

Data

```
{
  "resources": [
    {"id": "54cf10", "owner": "alice"},
    {"id": "3df429": "owner": "bob"}
    ...
  ],
  ...
}
```

Allow if user owns resource.
Not sure where resource is in array

```
# allow if resource is at element 0
allow {
    input.resource == data.resources[0].id
    input.user     == data.resources[0].owner
}
```

What about Arrays?

Input

```
{
  "user":    "alice"
  "resource": "54cf10",
}
```

Data

```
{
  "resources": [
    {"id": "54cf10", "owner": "alice"},
    {"id": "3df429": "owner": "bob"}
    ...
  ],
  ...
}
```

Allow if user owns resource.
Not sure where resource is in array

```
# allow if resource is at element 0
allow {
    input.resource == data.resources[0].id
    input.user     == data.resources[0].owner
}

# OR if resource is at element 1
allow {
    input.resource == data.resources[1].id
    input.user     == data.resources[1].owner
}
```

openpolicyagent.org

What about Arrays?

Input

```
{
  "user":    "alice"
  "resource": "54cf10",
}
```

Data

```
{
  "resources": [
    {"id": "54cf10", "owner": "alice"},
    {"id": "3df429": "owner": "bob"}
    ...
  ],
  ...
}
```

Problem: **Unknown number of elements. Cannot write allow for every index.**

Allow if user owns resource.
Not sure where resource is in array

```
# allow if resource is at element 0
allow {
    input.resource == data.resources[0].id
    input.user     == data.resources[0].owner
}

# OR if resource is at element 1
allow {
    input.resource == data.resources[1].id
    input.user     == data.resources[1].owner
}

...
```

Iterate over Arrays

Input

```
{
  "user":    "alice"
  "resource": "54cf10",
}
```

Data

```
{
  "resources": [
    {"id": "54cf10", "owner": "alice"},
    {"id": "3df429": "owner": "bob"}
    ...
  ],
  ...
}
```

Allow if user owns resource.
Not sure where resource is in array

```
# allow if resource is anywhere in array
allow {
    input.resource == data.resources[index].id
    input.user     == data.resources[index].owner
}
```

openpolicyagent.org

Iterate over Arrays

Input

```
{
  "user":    "alice"
  "resource": "54cf10",
}
```

Data

```
{
  "resources": [
    {"id": "54cf10", "owner": "alice"},
    {"id": "3df429": "owner": "bob"}
    ...
  ],
  ...
}
```

Allow if user owns resource.
Not sure where resource is in array

```
# allow if resource is anywhere in array
allow {
    input.resource == data.resources[index].id
    input.user     == data.resources[index].owner
}
```

Solution:

- **`allow`** is true if SOME value for **`index`** makes the rule body true.
- OPA automatically iterates over values for **`index`**.
- allow is true for **`index`** `= 0`

openpolicyagent.org

Iterate over Everything

Input

```
{
  "method": "GET",
  "path":   ["resources", "54cf10"],
  "user":   "bob"
}
```

Data

```
{
  "resources": [
    {"id": "54cf10", "owner": "alice"},
    {"id": "3df429": "owner": "bob"}
  ],
  "users": {
    "alice":   {"admin": false},
    "bob":     {"admin": true},
    "charlie": {"admin": true},
  }
}
```

Iterate over arrays/dictionaries (whether input or data)

```
# Iterate over array indexes/values
resource_obj := data.resources[index]

# Iterate over dictionary key/values
user_obj := data.users[name]

# Doesn’t matter whether input or data
value := input[key]

# Use _ to ignore variable name
# Iterate over just the array values
resource_obj := data.resources[_]
```

openpolicyagent.org

Duplicated Logic Happens with Iteration too

Data

```
{
  “users”: [
    {"name": "alice", "admin": false, “dept”: “eng”},
    {"name": "bob", "admin": true, “dept”: “hr”},
    {"name": "charlie", "admin": true, “dept”: “eng”},
  }
}
```

Duplicated logic with iteration

```
allow {
  user := data.users[_]
  user.admin == true
  user.name == input.user
  input.method == “GET”
}

allow {
  user := data.users[_]
  user.admin == true
  user.name == input.user
  input.method == “POST”
}
```

Duplicated Logic Happens with Iteration too

Data

```
{
  “users”: [
    {"name": "alice", "admin": false, “dept”: “eng”},
    {"name": "bob", "admin": true, “dept”: “hr”},
    {"name": "charlie", "admin": true, “dept”: “eng”},
  }
}
```

Duplicated logic with iteration

```
allow {
  user := data.users[_]
  user.admin == true
  user.name == input.user
  input.method == “GET”
}

allow {
  user := data.users[_]
  user.admin == true
  user.name == input.user
  input.method == “POST”
}
```

Avoid duplicating common logic like a search for admins

Create a Virtual Document

Data

```
{
  “users”: [
    {"name": "alice", "admin": false, “dept”: “eng”},
    {"name": "bob", "admin": true, “dept”: “hr”},
    {"name": "charlie", "admin": true, “dept”: “eng”},
  }
}
```

`admin` is a set that contains all of the admin names

Sets are an extension of JSON.

```
admin == { "bob", "charlie" }
```

Duplicated logic with iteration

```
allow {
  admin[input.user]
  input.method == “GET”
}

allow {
  admin[input.user]
  input.method == “POST”
}

admin[user.name] {
  user := data.users[_]
  user.admin == true
}
```

Different Syntaxes for Virtual Sets

Data

```
{
  “users”: [
    {"name": "alice", "admin": false, “dept”: “eng”},
    {"name": "bob", "admin": true, “dept”: “hr”},
    {"name": "charlie", "admin": true, “dept”: “eng”},
  }
}
```

Rule Syntax

```
admin[user.name] {
  user := data.users[_]
  user.admin == true
}
```

Set Comprehension Syntax

```
admin = {user.name |
  user := data.users[_]
  user.admin == true
}
```

Different Syntaxes for Virtual Sets

Data

```
{
  “users”: [
    {"name": "alice", "admin": false, “dept”: “eng”},
    {"name": "bob", "admin": true, “dept”: “hr”},
    {"name": "charlie", "admin": true, “dept”: “eng”},
  }
}
```

Rule Syntax

Supports OR with multiple rules.

```
admin[user.name] {
  user := data.users[_]
  user.admin == true
}
```

Set Comprehension Syntax

No support for OR.

```
admin = {user.name |
  user := data.users[_]
  user.admin == true
}
```

Create Virtual Dictionaries too

Data

```
{
  “users”: [
    {"name": "alice", "admin": false, “dept”: “eng”},
    {"name": "bob", "admin": true, “dept”: “hr”},
    {"name": "charlie", "admin": true, “dept”: “eng”},
  }
}
```

Rule Syntax

```
admin[user.name] = user.dept {
  user := data.users[_]
  user.admin == true
}
```

Dictionary Comprehension Syntax

```
admin = {user.name: user.dept |
  user := data.users[_]
  user.admin == true
}
```

openpolicyagent.org

Virtual Docs support iteration. Functions don’t.

Dictionary

```
admin[user_name] = user.dept {
  user := data.users[_]
  user.admin == true
  user.name == user_name
}
```

```
# lookup bob’s department
admin[“bob”]
# iterate over all user/dept pairs
admin[user] = department
# iterate over everyone in HR
admin[user] == “hr”
```

Function

```
admin(user_name) = user.dept {
  user := data.users[_]
  user.admin == true
  user.name == user_name
}
```

```
# lookup bob’s department
admin(“bob”)
# iterate over all user/dept pairs
Can’t.  Write different function.
# iterate over everyone in HR
Can’t.
```

openpolicyagent.org

Virtual Documents must be finite. Functions don’t.

Virtual Doc

Can’t express split_path.

Virtual docs must be “safe”.

Safety means the set of all input/output pairs is finite.

split_path takes any string as input. There are infinitely strings.

Function

```
split_path(str) = parts {
  trimmed := trim(str, "/")
  parts := split(trimmed, "/")
}
```

Agenda

- How Policies are Invoked
- Simple Policies
- Policies with Iteration
- **Additional Topics**
  - **Modularity**
  - **Negation**
  - **Any/All**
  - **Non-boolean Decisions**

People can Create Multiple Policies and Delegate

Entry point policy

```
package http.authz
import data.http.service_graph
import data.http.org_chart

allow {
   org_chart.allow
   service_graph.allow
}
```

Service graph policy

```
package http.service_graph
allow {
   input.source == “frontend”
   input.destination == “finance”
}
...
```

Organization chart policy

```
package http.org_chart
allow {
   admin[user.input]
}
...
```

openpolicyagent.org

Policies can use Negation

Entry point policy

```
package http.authz
import data.http.service_graph
import data.http.org_chart

allow {
   org_chart.allow
   not service_graph.deny
   not deny
}
deny { ... }
```

Service graph policy

```
package http.service_graph
deny {
   input.source == “frontend”
   input.destination == “finance”
}
...
```

Organization chart policy

```
package http.org_chart
allow {
   admin[user.input]
}
```

openpolicyagent.org

Any vs. All

Data

```
{
  “users”: {
    "alice":   {"admin": false, “org_code”: “11”},
    "bob":     {"admin": true,  “org_code”: “22”},
    "charlie": {"admin": true,  “org_code”: “33”}
  }
}
```

Check if all users are admins. Wrong ans:

```
all_admins = true {
  data.users[user_name].admin == true
}
```

Any vs. All

Data

```
{
  “users”: {
    "alice":   {"admin": false, “org_code”: “11”},
    "bob":     {"admin": true,  “org_code”: “22”},
    "charlie": {"admin": true,  “org_code”: “33”}
  }
}
```

Check if all users are admins. Wrong ans:

```
all_admins = true {
  data.users[user_name].admin == true
}
```

Problem: `all_admins` is `true` if ANY users are admins.

openpolicyagent.org

Any vs. All

Data

```
{
  “users”: {
    "alice":   {"admin": false, “org_code”: “11”},
    "bob":     {"admin": true,  “org_code”: “22”},
    "charlie": {"admin": true,  “org_code”: “33”}
  }
}
```

Check if all users are admins.

```
all_admins = true {
  not any_non_admins
}

any_non_admins = true {
  user := data.users[user_name]
  not user.admin
}
```

Solution:

1. Check if any users are NOT admins

2. Complement (1)

Any vs. All

Data

```
{
  “users”: {
    "alice":   {"admin": false, “org_code”: “11”},
    "bob":     {"admin": true,  “org_code”: “22”},
    "charlie": {"admin": true,  “org_code”: “33”}
  }
}
```

Solution:

1. Check if any users are NOT admins

2. Complement (1)

Check if all users are admins.

```
all_admins = true {
  not any_non_admins
}

any_non_admins = true {
  user := data.users[user_name]
  not user.admin
}
```

allow/deny are NOT special. Decisions are JSON

1. **Example Request**

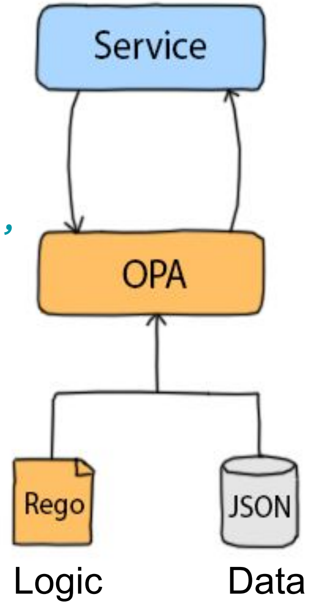
```
POST v1/data/http/authz/admin
{"input": {
    "method": "GET",
    "path":   ["finance", "salary", "alice"],
    "user":   "bob"}}
```

2. **Example Policy**

```
package http.authz
import data.http.service_graph
import data.http.org_chart

admin[x] {
   org_chart.admin[x]
}
admin[x] {
   service_graph.admin[x]
}
```

Sets defined with
multiple rules
are unioned together.



3. **Example Response**

```
{“result”: [“bob”, “charlie”]}
```

Policy decision can be any JSON data: boolean, number, string, null, array, or dictionary.

Sets are serialized to JSON arrays.

openpolicyagent.org

Thank You!

slack.openpolicyagent.org

github.com/open-policy-agent/opa

Policy Example with Join

Policies Iterate to Search for Data

Data

```
{
  “users”: {
    "alice":   {"admin": false, “org_code”: “11”},
    "bob":     {"admin": true,  “org_code”: “22”},
    "charlie": {"admin": true,  “org_code”: “33”}
  },
  “orgs”: {
    “00”: {“name”: “HR”},
    “11”: {“name”: “Legal”},
    “22”: {“name”: “Research”},
    “33”: {“name”: “IT”},
    “44”: {“name”: “Accounting”},
  }
}
```

Search for the data you need

```
# Find admin users and their organization
user_obj := data.users[user_name];
user_obj.admin == true;
org_name := data.orgs[user_obj.org_code].name
```

Variable assignments that satisfy search criteria

user_obj	user_name	org_name
{"admin": true, ...}	bob	Research
{"admin": true, ...}	charlie	IT

openpolicyagent.org

Policies Give Names to Search Results

Data

```
{
  “users”: {
    "alice":   {"admin": false, “org_code”: “11”},
    "bob":     {"admin": true,  “org_code”: “22”},
    "charlie": {"admin": true,  “org_code”: “33”}
  },
  “orgs”: {
    “00”: {“name”: “HR”},
    “11”: {“name”: “Legal”},
    “22”: {“name”: “Research”},
    “33”: {“name”: “IT”},
    “44”: {“name”: “Accounting”},
  }
}
```

Name the search results

```
admins[[org_name, user_name]] {
  user_obj := data.users[user_name]
  user_obj.admin == true
  org_name := data.orgs[user_obj.org_code].name
}
```

`admins` is a set that contains all of the **`[org_name, user_name]`** pairs that make the body true.

```
admins == {
  ["Research", "bob"],
  ["IT", "charlie"],
}
```

openpolicyagent.org

Policies Apply Search Results to Make Decisions

Input

```
{
  "method": "GET",
  "path":   ["resources", "54cf10"],
  "user":   "bob"
}
```

Data

```
{
  “users”: {
    "alice":   {"admin": false, “org_code”: “11”},
    "bob":     {"admin": true,  “org_code”: “22”},
    "charlie": {"admin": true,  “org_code”: “33”}
  },
  “orgs”: {
    “00”: {“name”: “HR”},
    “11”: {“name”: “Legal”},
    “22”: {“name”: “Research”},
  ...
```

Apply the search results

```
allow {
  # allow admins to do everything
  admins[[_, input.user]]
}

admins[[org_name, user_name]] {
  user_obj := data.users[user_name]
  user_obj.admin == true
  org_name := data.orgs[user_obj.org_code].name
}
```

Check if bob is an admin	`admins[[_, “bob”]]`
Lookup IT admins	`admins[[“IT”, name]]`
Iterate over all pairs	`admins[x]`

openpolicyagent.org