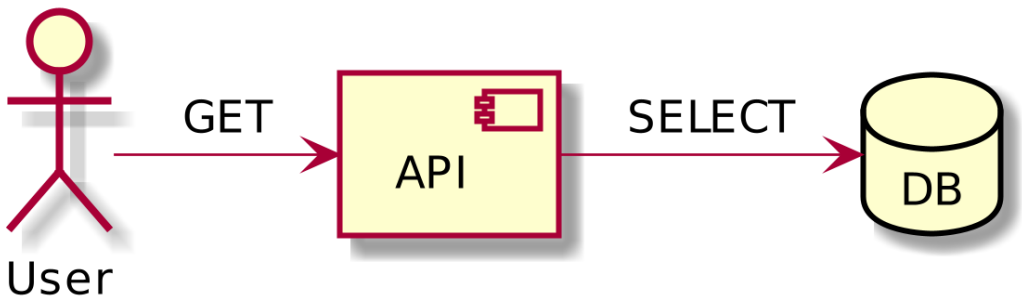
Open Policy Agent

Michiel Kalkman - DigIO

Let's build an API

Requirement 1 : Orders are accessible via an API

Design time!



Controller v1 : Get the orders

```
getOrders(request, response, ctxt) => {
  let orders = ctx.database.findAll('order');
  response.status = 200;
  response.body = orders;
  return [request, response];
}
```

Requirement 16 : A valid session is required to access the API

Controller v2 : Check that the session is valid

```
getOrders(request, response, ctxt) => {
  if (!request.tokenisValid) {
    response.status = 401;
    response.body = 'Unauthenticated';
  } else {
    let orders = ctx.database.findAll('order');
    response.status = 200;
    response.body = orders;
  }
  return [request, response];
}
```

Requirement 31 : Only premium users can access Orders

Controller v3 : You need the premium role

```
getOrders(request, response, ctxt) => {
  if (!request.token.isValid) {
    response.status = 401;
    response.body = 'Unauthenticated';
  } else if (request.roles.includes('premium')) {
    response.status = 403;
    response.body = 'Unauthorized';
  } else {
    let orders = ctx.database.findAll('order');
    response.status = 200;
    response.body = orders;
  }
  return [request, response];
}
```

Requirements analysis

- How is *premium* defined? Is it a role?
  - Is that role required to interact with orders?
- Does creating an order make the creator the owner?
  - Is there a difference between owner and creator?
- Can users see orders that they have created themselves?
- Can users see orders that they are the owner of?
- Can orders be changed?
  - Are there restrictions? (E.g. changing a fulfilled order)
- Can users create orders for other users?
  - If so, are there restrictions? (E.g. within groups or hierarchies)

Controller v4 : Users can only see their own orders

```
getOrders(request, response, ctxt) => {
  // auth session + roles
  let orders = ctx.database.findAll('order', {
    filter : [
      { creatorId: request.session.userId },
      { ownerId: request.session.userId },
    ] // implicit OR?
  });
  response.status = 200;
  response.body = orders;
  return [request, response];
}
```

Controller v5 : Admins see everything

```
getOrders(request, response, ctxt) => {
  // auth session + roles
  if (!request.roles.includes('admin')) {
    let orders = ctx.database.findAll('order', {
      filter : [
        { creatorId: request.session.userId },
        { ownerId: request.session.userId },
      ] // implicit OR?
    });
  } else {
    let orders = ctx.database.findAll('order', { });
  }
  // response
}
```

Controller v6 : Add audit trail

```
getOrders(request, response, ctxt) => {
  // auth session + roles
  const userId = request.session.userId;
  if (!request.roles.includes('admin')) {
    logger.info(`getOrders: user:${userId}`);
    let orders = ctx.database.findAll('order', {
      filter : [
        { creatorId: userId }, { ownerId: userId },
      ] // implicit OR?
    });
  } else {
    // TOFIX: Admins need the premium role to get here
    logger.info(`getOrders: user:${userId} as admin`);
    let orders = ctx.database.findAll('order', { });
  }
  // response
}
```

Oops

Figure 1: How did we get here?

Breakdown

Function	**Concern**
Check user is authenticated	Authentication
Check user is authorized	Authorization
Check user if user is an admin	Authorization
Check user for premium role	Authorization
Generate audit logs	Auditing
Create user filter restrictions	Confidentiality
Execute query	API
Return results	API

Casual observations

In this simplified example,

- 75% of code is unrelated to the actual API function
- This will be similar for other functions (verbs)
- 26 LoC per function * 4 verbs = 104 Loc per Resource
- Say 10 Resources for the project = 1040 lines of Resource code
- Industry average 15 to 50 errors per 1000 **delivered** LoC
  - *From Code Complete, Microsoft Press*

Why is everything on fire?

75% of code is unrelated

- Wasted effort
- Lost opportunity
- Testing overhead
  - Complexity doesn't increase linearly
- Subject to change
  - Any change is more complex than necessary
- Still has to be maintained
  - Any maintenance is more complex than necessary
- 300% more bugs than necessary
  - Fixes, maintenance, testing costs, etc
- Technical debt with no up side
  - Compound interest on that debt

Definition

policy *(noun) a set of ideas or* ***a plan of what to do in particular situations that has been agreed to officially by a group of people, a business organization****, a government, or a political party*

Information security policy decomposition

CIA Triad

- **Confidentiality** the protection of data from unauthorized disclosure
- **Integrity** the protection of data from unauthorized alteration
- **Availability** the protection from disruption of authorized access

The Triple A's

- **Authentication** the identification of a user
- **Authorization** controlling the subject-object-activity model
- **Auditing** ensure controls are operational and effective

Where are we now?

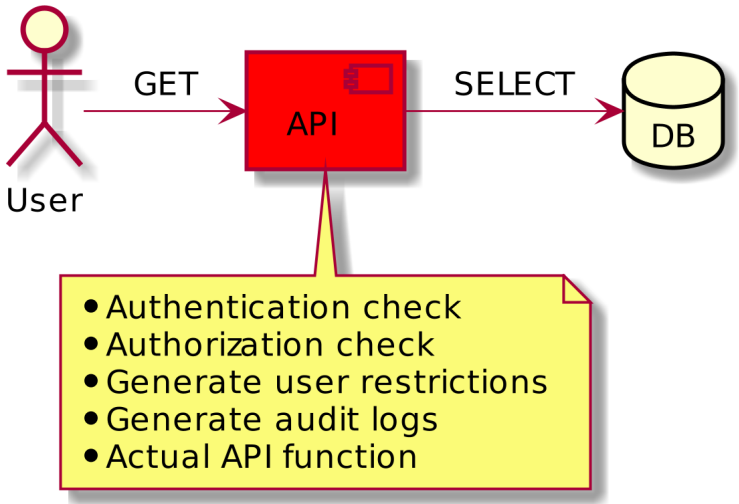


Figure 2: Not so good

Refactor time!

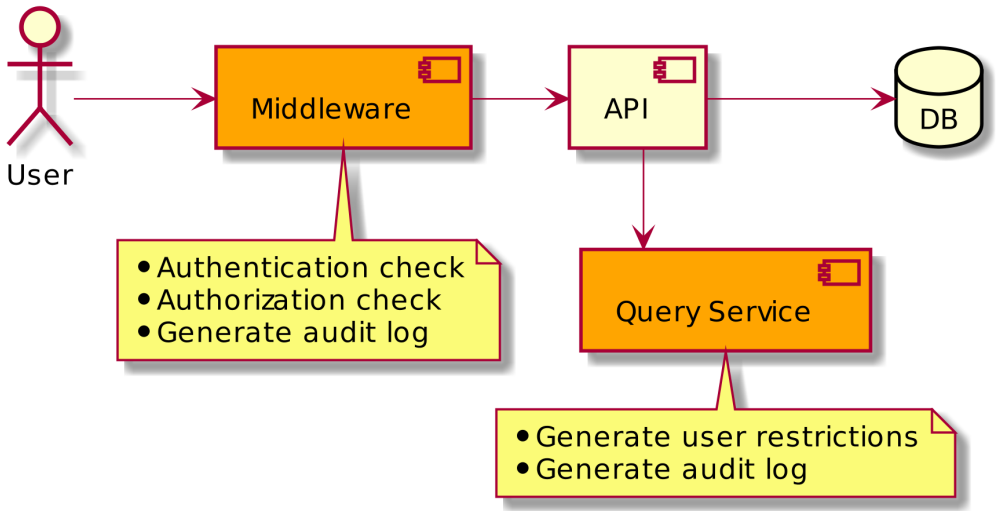


Figure 3: Better

Further refactoring time!

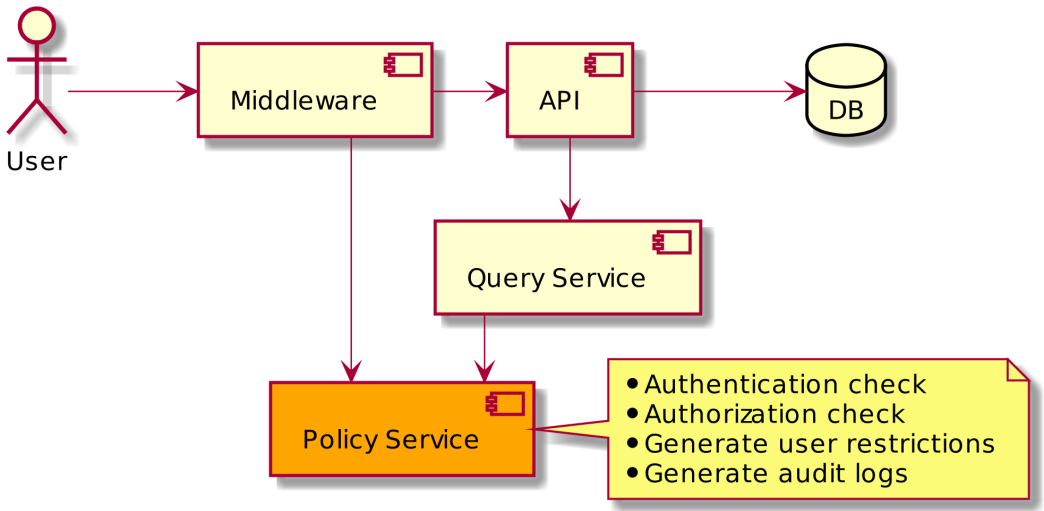


Figure 4: Even better

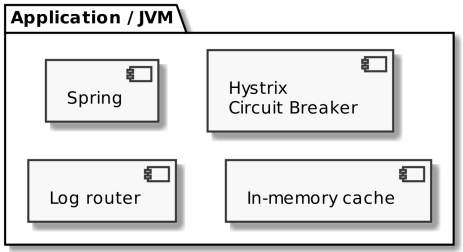
Controller v7 : Restore sanity

```
getOrders(request, response, ctxt) => {
  let orders = ctx.database.findAll('order',
      ctxt.queryService.filtersFor(
        request
      )
  );
  response.status = 200;
  response.body = orders;
  return [request, response];
}
```

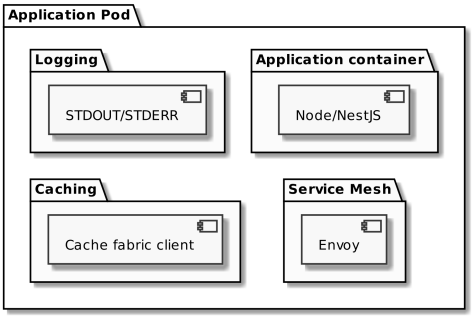
Deployment

Trends

Then



Now



Manage it separately

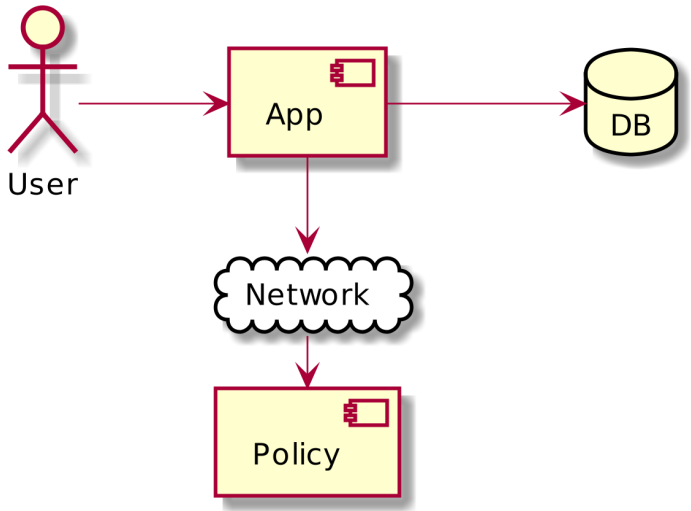
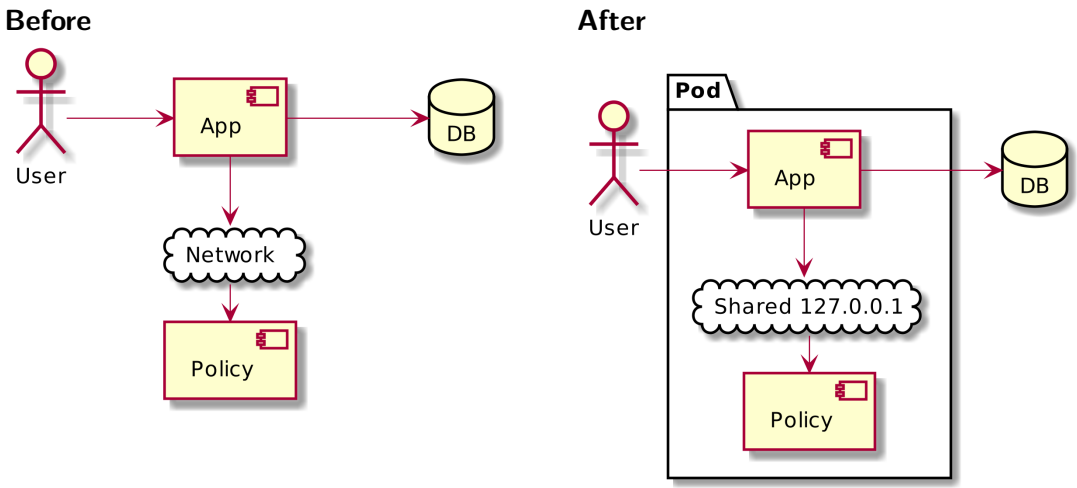


Figure 5: Separate process

.. separately, as a sidecar



.. as a shared resource

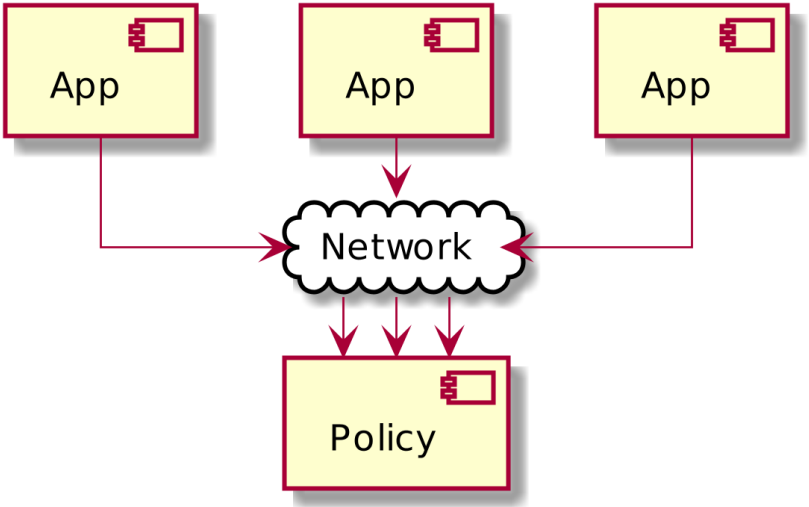
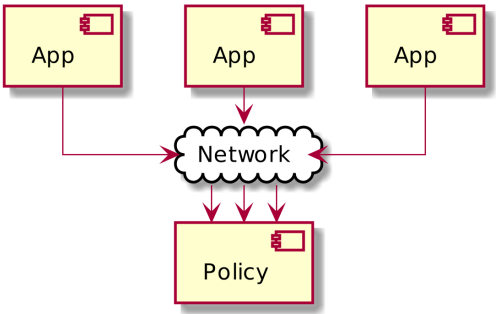


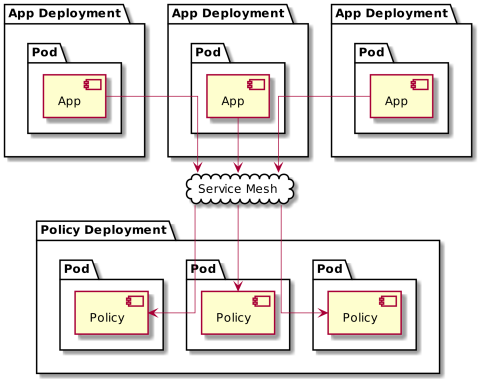
Figure 6: Shared resource

.. shared resource, as a service

Before



After



.. as a shared central resource

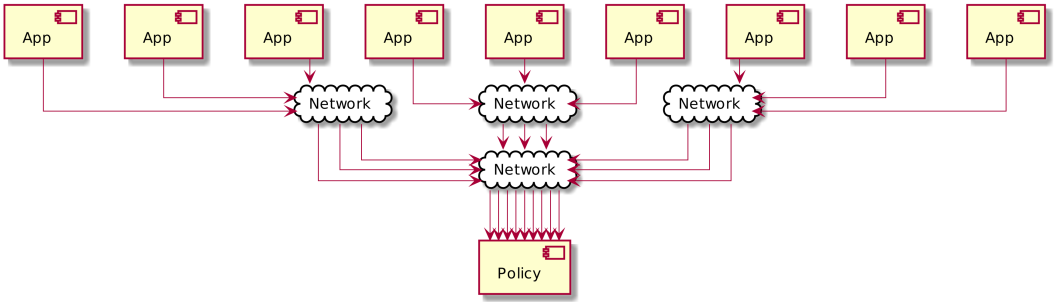


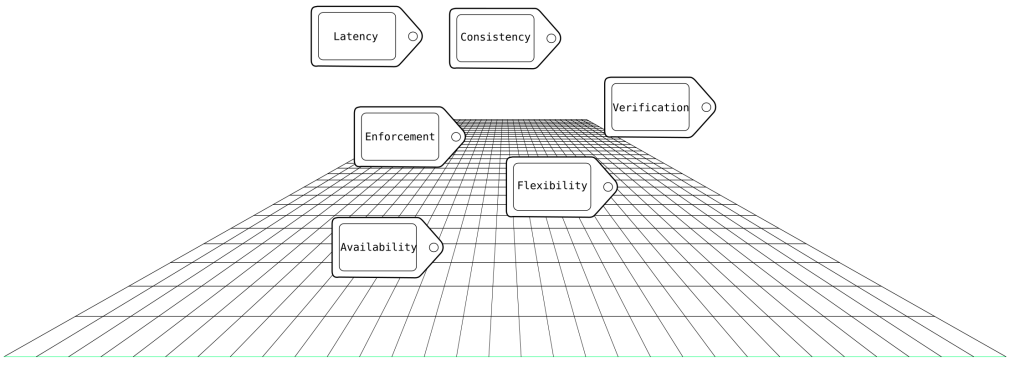
Figure 7: Centralized shared resource

Oooops

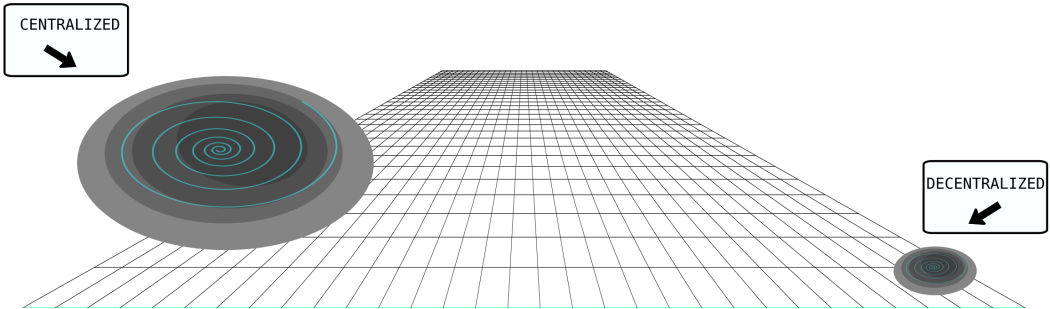
Figure 8: This is not fine

Policy Management

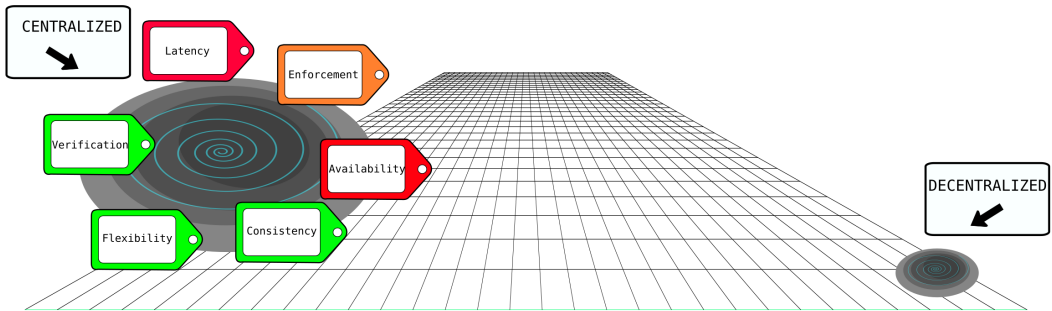
Policy control



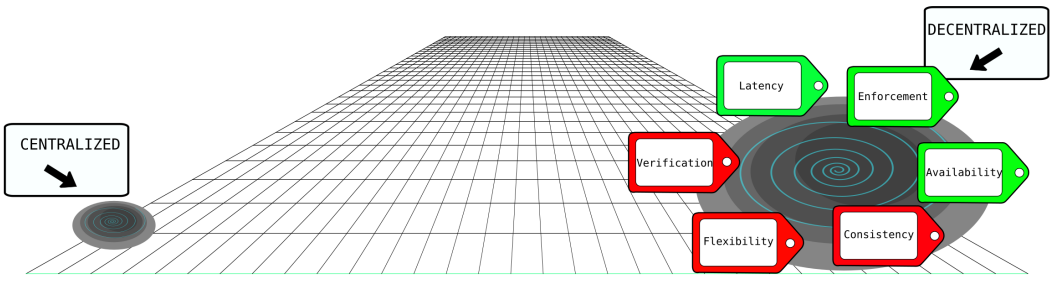
Policy control



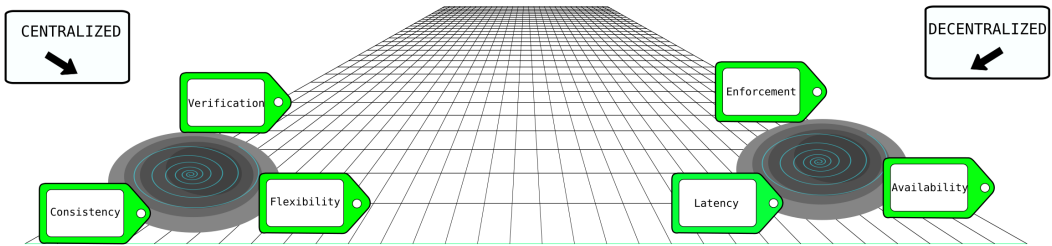
Policy control



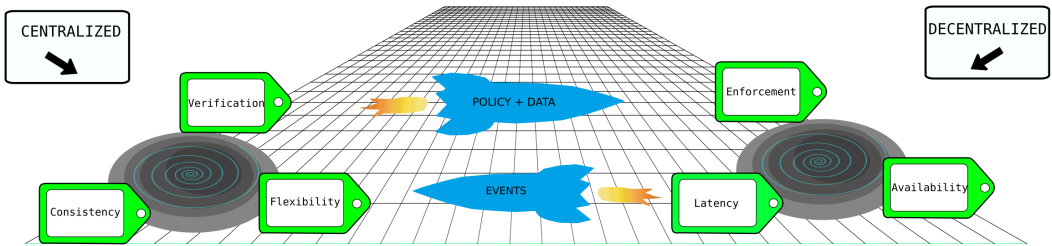
Policy control



Policy control



Policy control



Open Policy Agent

Figure 9: Viking helm OPA logo

Example AD/JWT flow

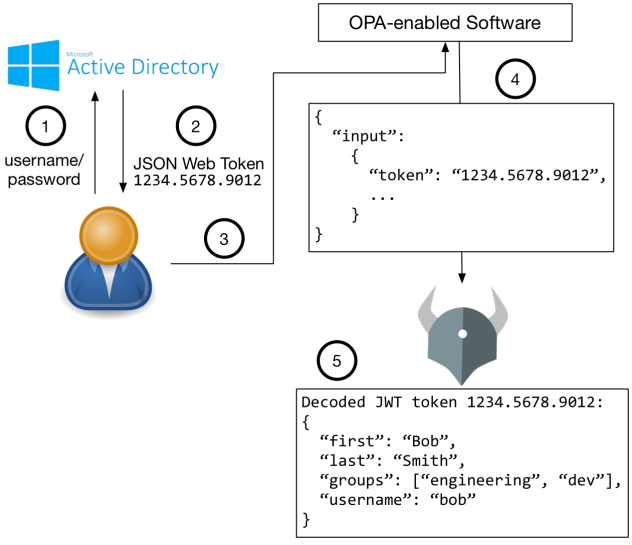


Figure 10: Best practice, Identity and JWT

Data flow

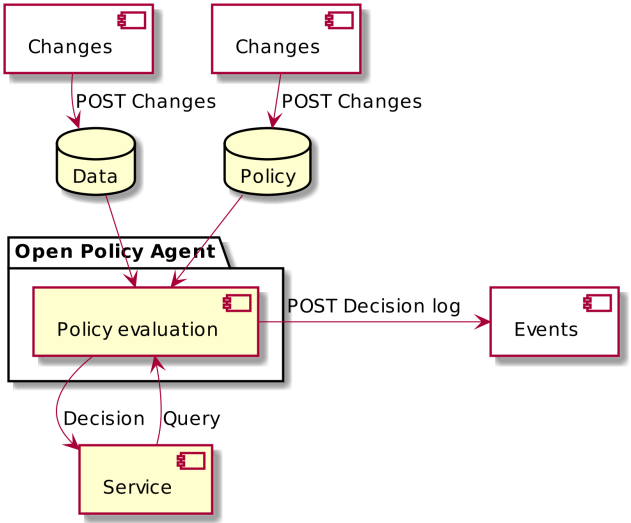


Figure 11: Components

Dynamic policy deployment

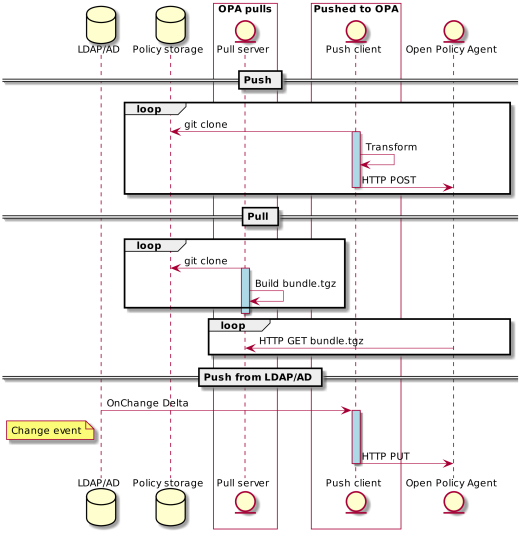


Figure 12: Centralized management, distributed enforcement

Rego policies

Rego language

Rego was inspired by Datalog, which is a well understood, decades old query language. Rego extends Datalog to support structured document models such as JSON.

Rego queries are assertions on data stored in OPA. These queries can be used to define policies that enumerate instances of data that violate the expected state of the system.

Rego Variable Keys

References can include variables as keys. References written this way are used to select a value from every element in a collection.

rego

```
sites := [
    {"name": "prod"},
    {"name": "smoke1"},
    {"name": "dev"}
]

q[v1] { v1 := sites[v2].name }

# q == ["prod", "smoke1", "dev"]
```

javascript

```
const sites = [
    {"name": "prod"},
    {"name": "smoke1"},
    {"name": "dev"},
];

const q = sites.map(s=> s["name"]);

// q == ["prod", "smoke1", "dev"]
```

Rego Variable Keys

rego REPL

```
> sites[i].servers[j].host
+---+---+------------------------+
| i | j | sites[i].servers[j].host|
+---+---+------------------------+
| 0 | 0 | "hydrogen"             |
| 0 | 1 | "helium"               |
| 0 | 2 | "lithium"              |
| 1 | 0 | "beryllium"            |
| 1 | 1 | "boron"                |
| 1 | 2 | "carbon"               |
| 2 | 0 | "nitrogen"             |
| 2 | 1 | "oxygen"               |
+---+---+------------------------+
```

python

```python
def hosts(sites):
  result = []
  for site in sites:
    for server in site.servers:
      result.append(server.host)
  return result
```

Rego Packages

```
package my.scratchpad
servers = [
  {"name": "smoke1", "protocol": "http"},
  {"name": "smoke2", "protocol": "gopher"}
]
```

```
$ curl http://localhost/v1/data/my/scratchpad/servers
```

```
package my.testpad
import data.my.scratchpad.servers
http_servers[server] {
    server := servers[_]
    server.protocols[_] == "http"
}
# http_servers == [
#  {"name": "smoke1", "protocol": "http"}
# ]
```

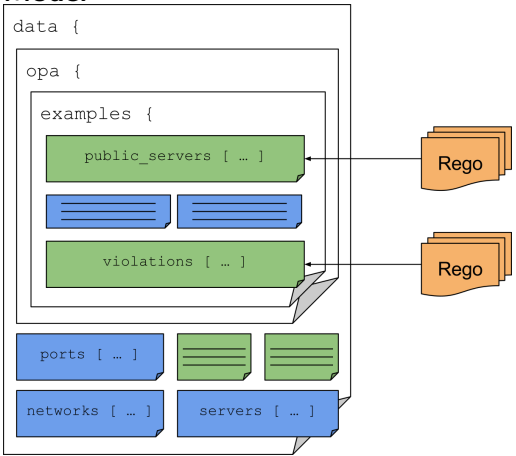
Logical data model

Policy (rego)

```
package opa.examples
import data.servers
import data.networks
import data.ports

violations[server] {
  server := servers[_]
  server.protocols[_] == "http"
  public_servers[server]
}
public_servers[server] {
  server := servers[_]
  server.ports[_] == ports[i].id
  networks[j].public == true
}
```

Model



Example HTTP API policy

```
package acmecorp.authz
default allow = false

# Allow people to read their own salaries.
allow {
    input.method = "GET"
    input.path = ["salaries", employee_id]
    input.user = employee_id
}

# Also allow managers to read the salaries of people they manage.
allow {
    input.method = "GET"
    input.path = ["salaries", employee_id]
    input.user = data.manager_of[employee_id]
}
```

SSH access

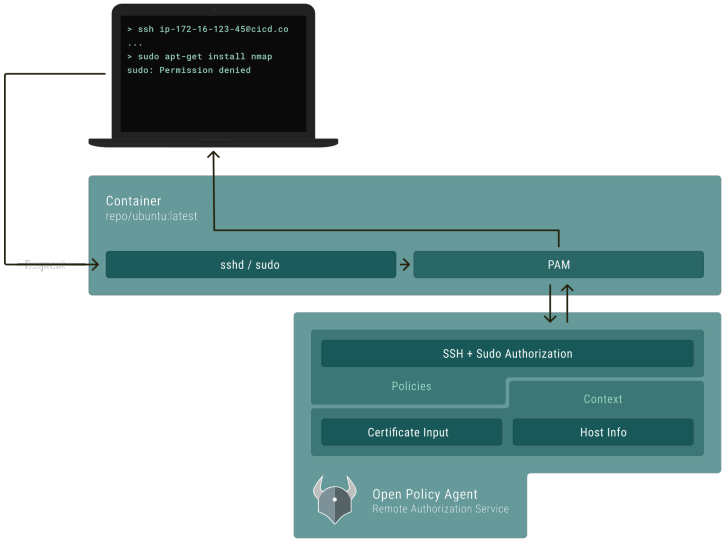


Figure 13: SSH

Example sshd/sudo policy

```
package ssh.fine_grained

# Allow "dev" users access if they possess a certificate proving
# they are assigned to an application running on the host.
allow {
  certs := crypto.x509.parse_certificates(input.certificates)

  certs[i].Subject.Organization[j] == data.host_info.apps[_]
  certs[i].Subject.OrganizationalUnit[j] == "dev"

  time.now_ns() >= certs[i].NotBefore
  time.now_ns() <= certs[i].NotAfter
}
```

Policy testing

Policy

```
package authz

allow {
 input.path == ["users"]
 input.method == "POST"
}

allow {
 input.path = ["users", user_id]
 input.method == "GET"
 user_id == input.user_id
}
```

Tests

```
package authz
test_post_allowed {
 allow with input as {
  "path": ["users"],
  "method": "POST"
 }}

test_get_anonymous_denied {
 not allow with input as {
  "path": ["users"],
  "method": "GET"
 }}
```

Testing - Data mocking

Policy

```
package authz

allow {
 x := data.policies[_]
 x.name == "test"
 matches_role(input.role)
}

matches_role(role) {
 data.roles[role][_] == input.user
}
```

Tests

```
package authz

policies = [{"name": "test"}]
roles = {"admin": ["alice"]}

test_allow_with_data {
 allow with input as {
  "user": "alice",
  "role": "admin"
 }
  with data.policies as policies
  with data.roles as roles
}
```

Testing

```
$ opa test -v example.rego example_test.rego
data.authz.test_post_allowed: PASS (1.85µs)
data.authz.test_get_anonymous_denied: PASS (929ns)
--------------------------------------------------
PASS: 2/2
$ echo $?
0
```

Open Policy Agent - Query API

Policy checks for a GET-Object request

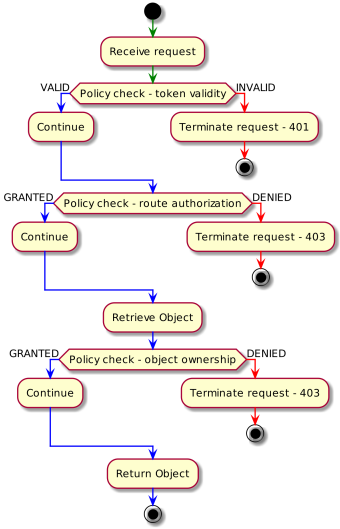


Figure 14: Request flow

API Policy - GET /orders - Route authorization

Policy (rego)

```
default deny
allow {
 input.method = "GET"
 input.path = ["orders"]
}
```

Request

```
{
 "method": "GET",
 "path": ["orders"]
}
```

Response

```
{
 "result" : {
  "allowed": true
 }
}
```

Open Policy Agent - Compile API

Partial Evaluation (SQL)

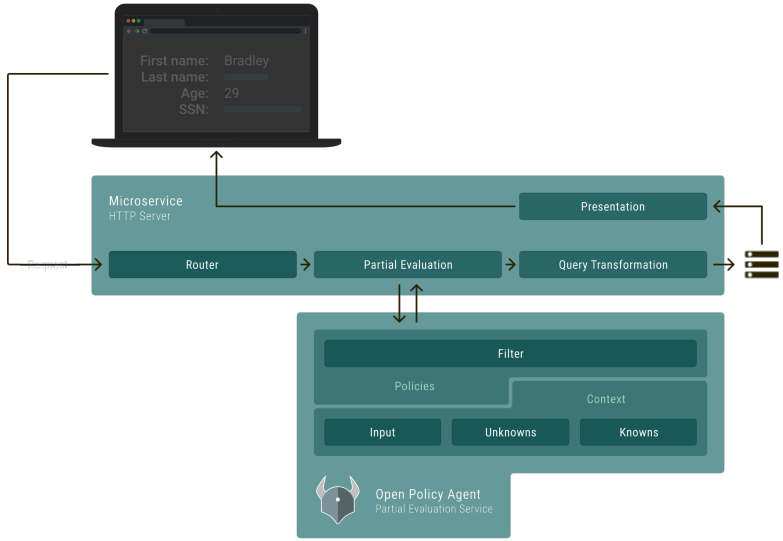


Figure 15: Add SQL WHERE clauses

Partial evaluation

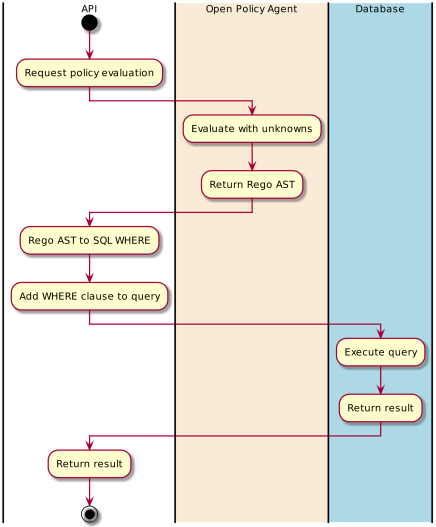


Figure 16: Partial evaluation to generate WHERE clauses

Example of data filtering

```
package example

allow {
  input.subject.clearance_level >= data.reports[_].clearance_level
}

allow {
  data.break_glass = true
}
```

Data filtering / compile query

POST /v1/compile

```
{
  "query": "data.example.allow == true",
  "input": {
    "subject": {
      "clearance_level": 4
    }
  },
  "unknowns": [
    "data.reports"
  ]
}
```

Scenario : Users and Admins accessing Orders

Scenario Users and Admins

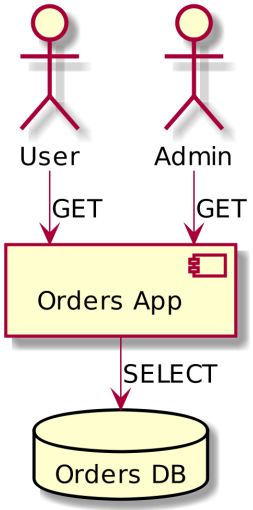


Figure 17: Everyone on the same interface

Scenario Users and Admins

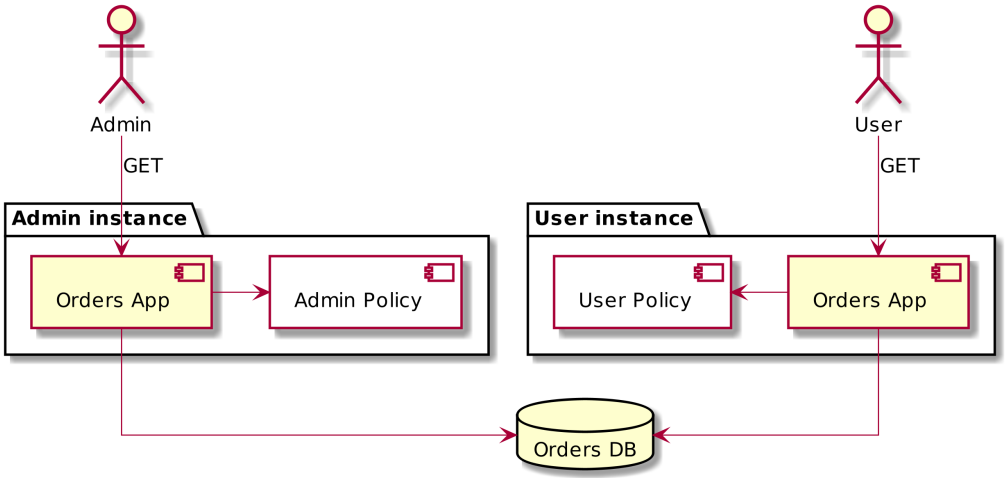


Figure 18: Multiple instances, different policies

Scenario Users and Admins

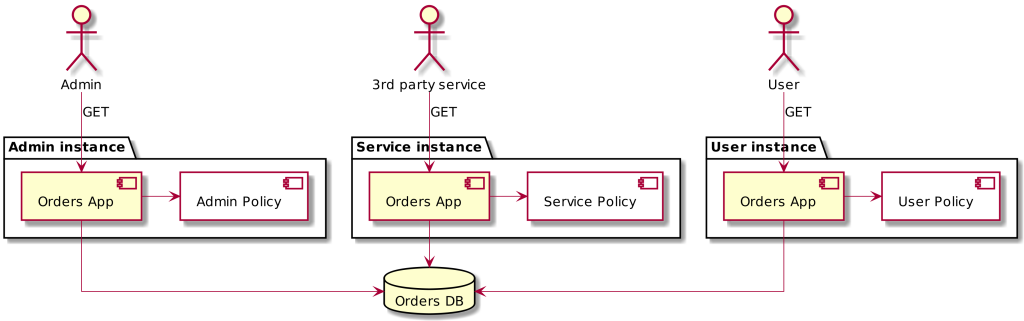


Figure 19: Scaling up instances, policies

Decision Log


```
[ {
  "labels": {
    "app": "finance-app",
    "id": "1780d507-aea2-45cc-ae50-fa153c8e4a5a"
  },
  "decision_id": "4ca636c1-55e4-417a-b1d8-4aceb67960d1",
  "revision": "W3sibCI6InN5cy9jYXRhbG9nIiwicyI6NDA3MX1d",
  "path": "http/example/authz/allow",
  "input": {
    "method": "GET",
    "path": "/salary/bob"
  },
  "result": "true",
  "requested_by": "[::1]:59943",
  "timestamp": "2018-01-01T00:00:00.000000Z"
} ]
```

Further deployment options

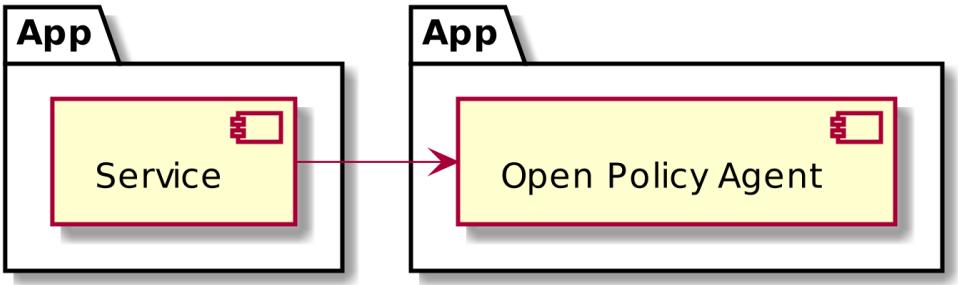


Figure 20: What we have now

Further deployment options

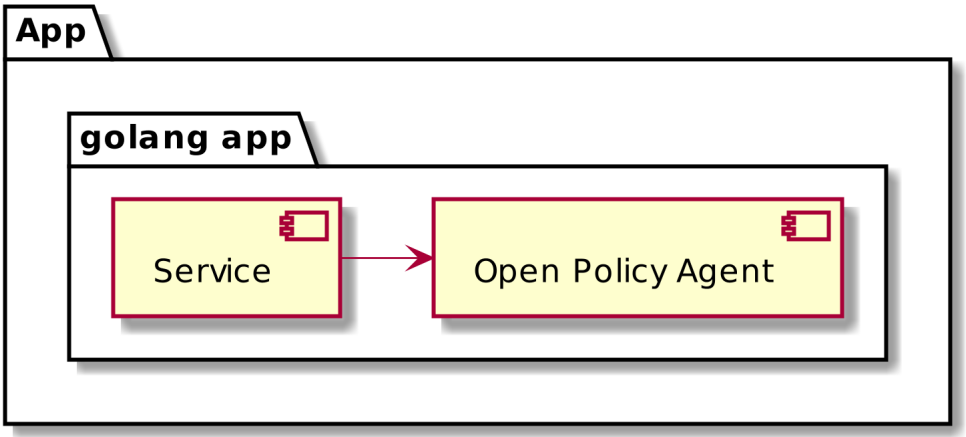


Figure 21: Embedded golang library

Further deployment options

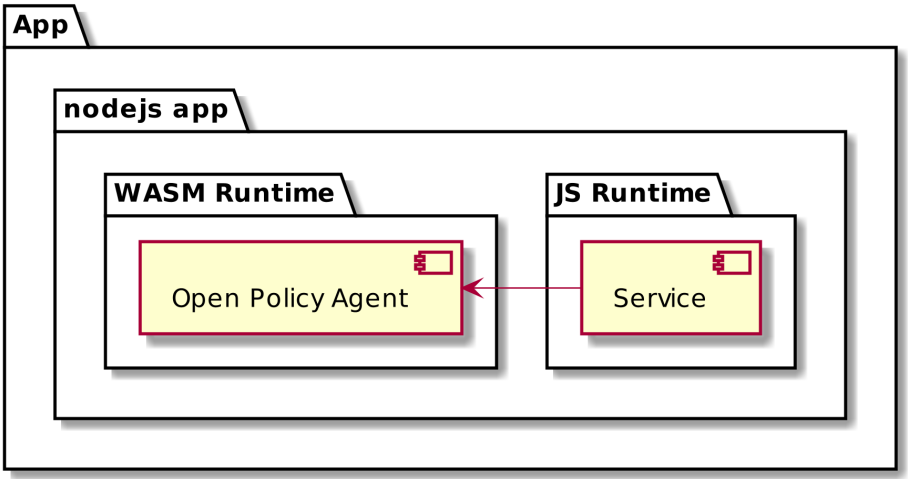


Figure 22: WASM + node

Further deployment options

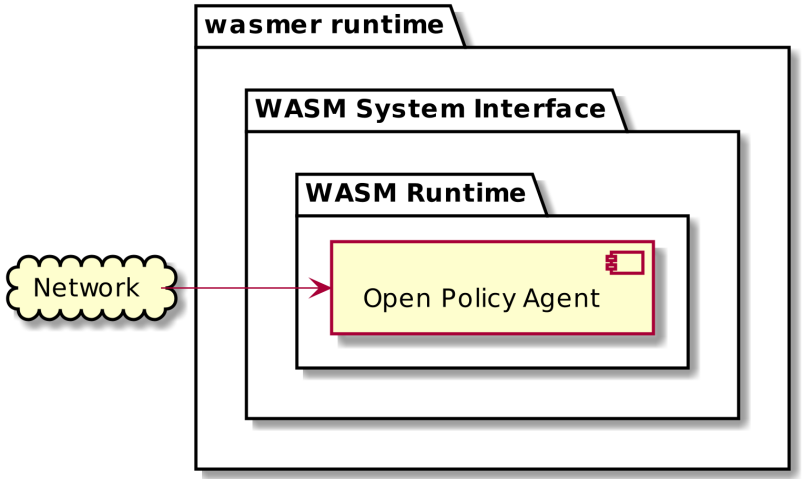


Figure 23: WASM + WASI

Summary

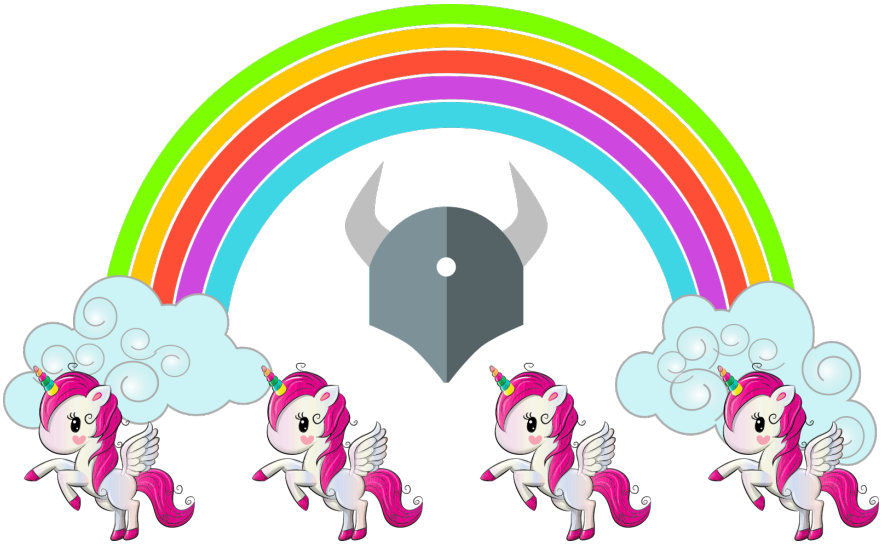


Figure 24: OPA makes the sun shine, the birds sing and the grass green

Links

- openpolicyagent.org
- actix-web OPA Middleware