Open Policy Agent

CLOUD NATIVE
SANDBOX

@sometorin

@OpenPolicyAgent

nternal-s02.acmecorp.com/pages?pageid=settingermsinprod

Setting permissions in Production [edit]

Important notice: *If you are deploying to east_4A or icebreaker2, you MUST fill out the change request form and submit it through the config-deployment portal before continuing.*

Update: 2016-09-16: Jeff is working on automating this process.

NOTE: September, 2017: remember to add the following permissions to your production service.

Group	API Permission	Version
ops-auth	all	*
ops-admin	all	*
net-dev	net/iam	v2.0 and newer

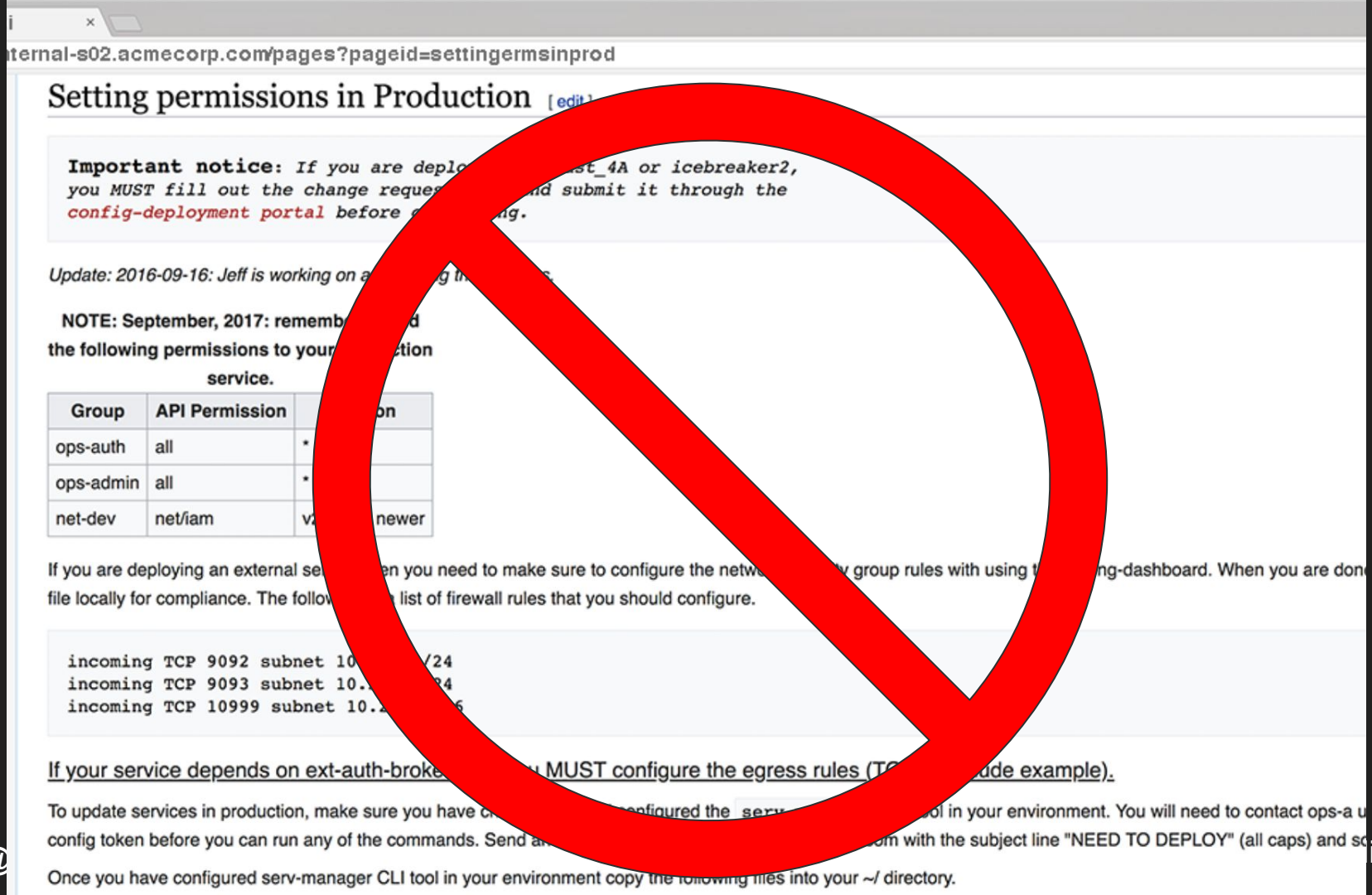
If you are deploying an external service then you need to make sure to configure the network security group rules with using the neteng-dashboard. When you are don file locally for compliance. The following is a list of firewall rules that you should configure.

```
incoming TCP 9092 subnet 10.2.2.0/24
incoming TCP 9093 subnet 10.2.2.0/24
incoming TCP 10999 subnet 10.2.0.0/16
```

If your service depends on ext-auth-broker then you MUST configure the egress rules (TODO: include example).

To update services in production, make sure you have checked out and configured the `serv-manager` CLI tool in your environment. You will need to contact ops-a u config token before you can run any of the commands. Send an e-mail to ops-auth@internal.acmecorp.com with the subject line "NEED TO DEPLOY" (all caps) and s

Once you have configured serv-manager CLI tool in your environment copy the following files into your ~/ directory.


nternal-s02.acmecorp.com/pages?pageid=settingermsinprod
Setting permissions in Production [edit]
Important notice: If you are deplo st_4A or icebreaker2,
you MUST fill out the change reque nd submit it through the
config-deployment portal before g.
Update: 2016-09-16: Jeff is working on a g th
NOTE: September, 2017: rememb d
the following permissions to your tion
service.
Group
API Permission
on
ops-auth
all
ops-admin
all
net-dev
net/iam
v2
newer
If you are deploying an external se en you need to make sure to configure the netw y group rules with using t ng-dashboard. When you are don
file locally for compliance. The follow list of firewall rules that you should configure.
incoming TCP 9092 subnet 10 /24
incoming TCP 9093 subnet 10. 24
incoming TCP 10999 subnet 10. 6
If your service depends on ext-auth-broke MUST configure the egress rules (TC ude example).
To update services in production, make sure you have c nfigured the serv ol in your environment. You will need to contact ops-a u
config token before you can run any of the commands. Send a om with the subject line "NEED TO DEPLOY" (all caps) and s
Once you have configured serv-manager CLI tool in your environment copy the following files into your ~/ directory.

Policy decisions should be decoupled from policy enforcement.

Treat policy as a separate concern.

...just like DB, messaging, monitoring, logging, orchestration, CI/CD...

Gain better control and visibility over policy throughout your system.

Everyone is affected by policy...

"Analysts can read client data but PII must be redacted."

"QA must sign-off on images deployed to the production namespace."

"Give developers SSH access to machines listed in JIRA tickets assigned to them."

"Restrict ELB changes to senior SREs that are on-call."

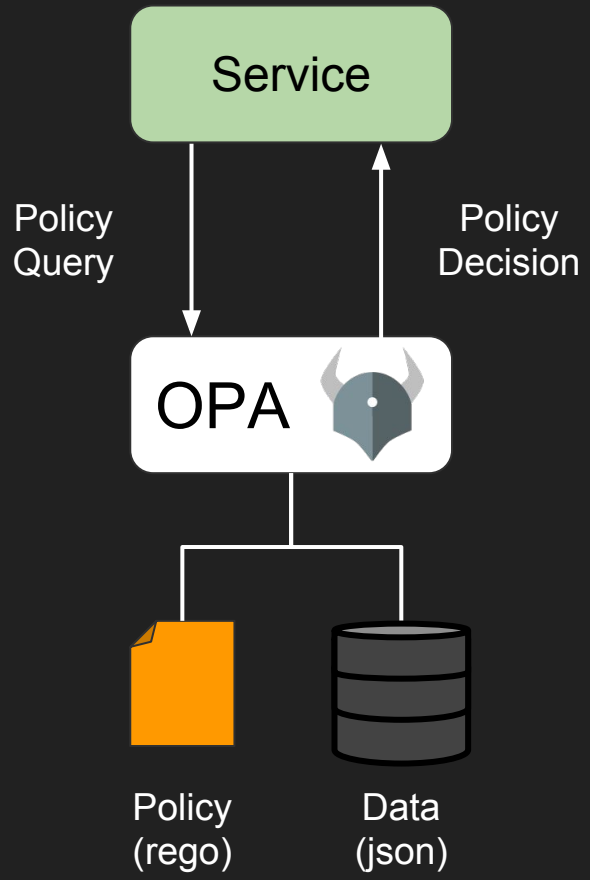
Policy enforcement is a fundamental problem for your organization.

Tribal knowledge provides NO guarantee that policies are being enforced.

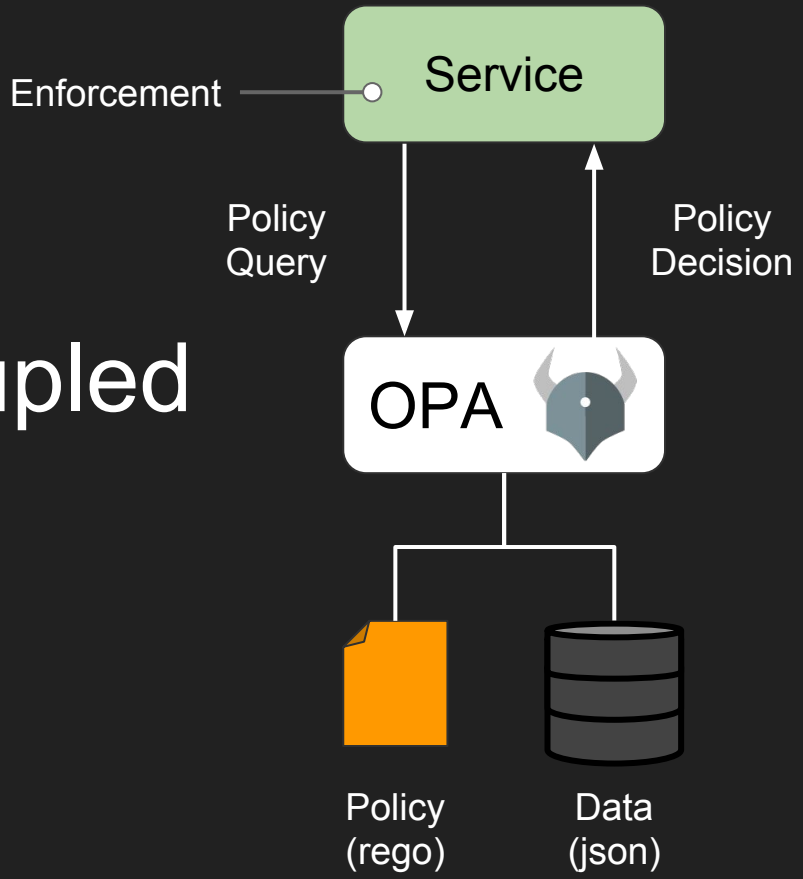
"Tribal knowledge" is the know-how or collective wisdom of the organization.

It is expensive and painful to maintain policy decisions that are hardcoded into the app.

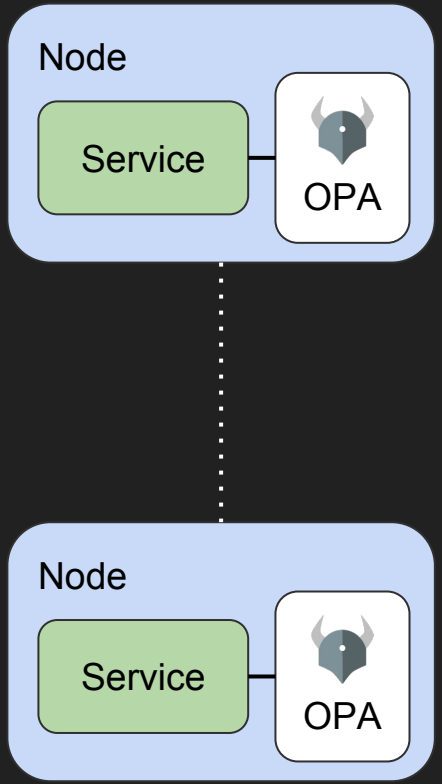
OPA is an open source, general-purpose policy engine.

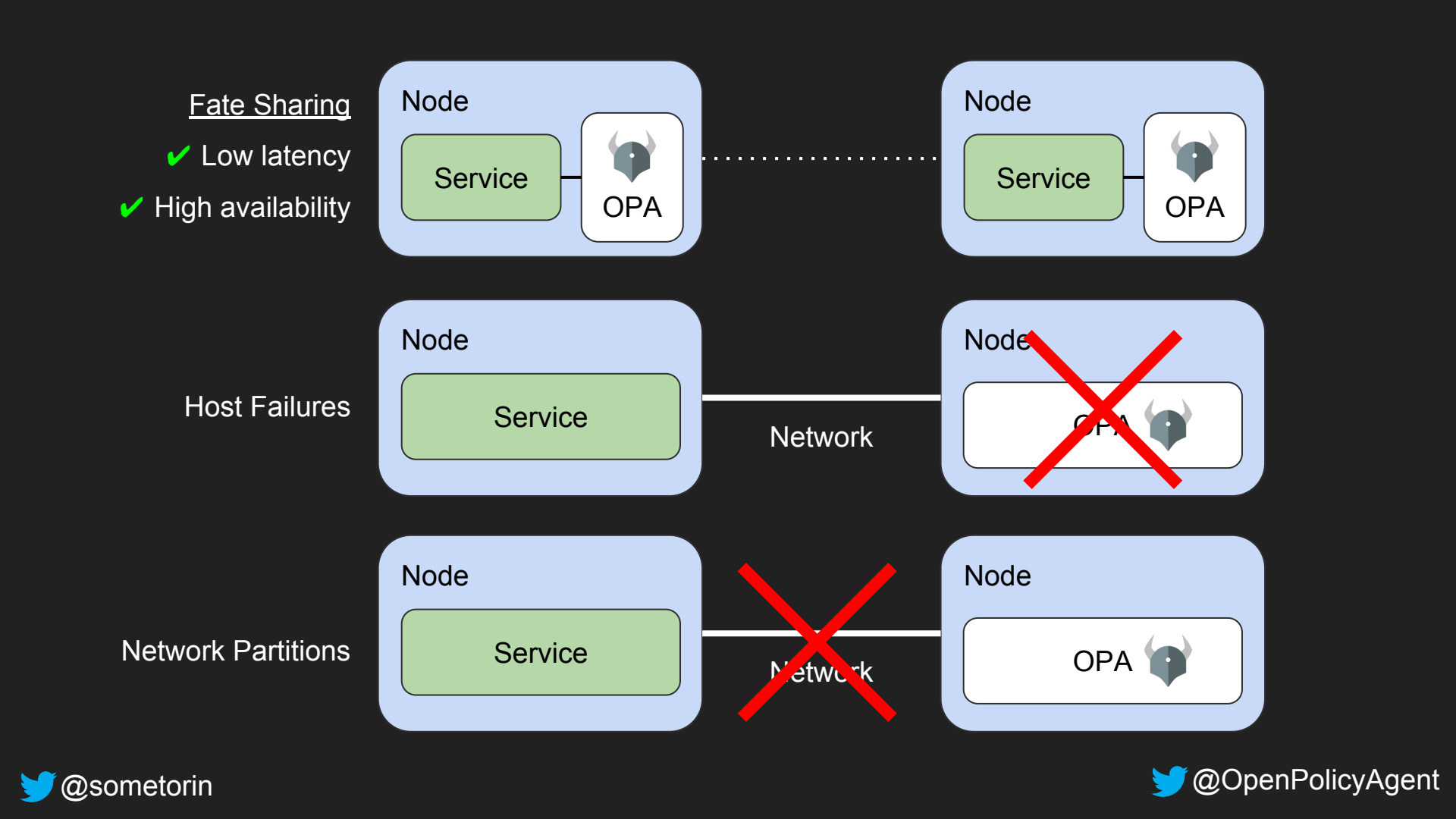


Decisions are decoupled from enforcement.



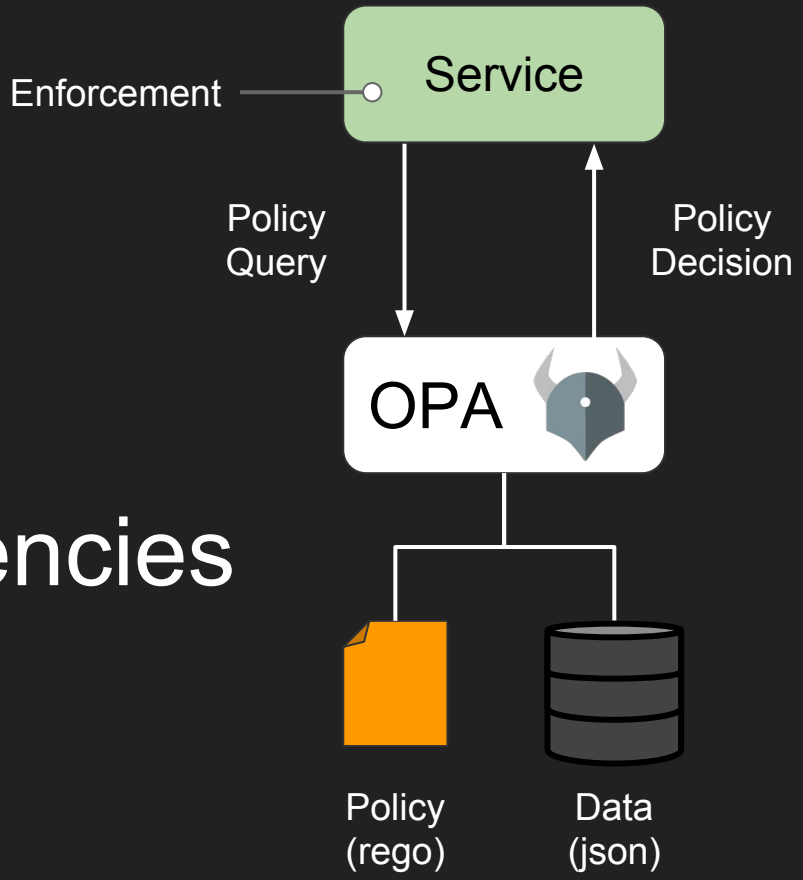
OPA is a host-local cache for policy decisions.




Fate Sharing
✔ Low latency
✔ High availability
Node
Service
OPA
Node
Service
OPA
Host Failures
Node
Service
Network
Node
OPA
Network Partitions
Node
Service
Node
OPA
@sometorin
@OpenPolicyAgent

Policy and data are stored in-memory.

No runtime dependencies during enforcement.



localhost/bob

bob (sign out)

Bob

"There's nothing wrong with having a tree as a friend."

Employee Details

Name: Bob Ross

Birth Date: October 29, 1942

Position: Cloud Engineer

T-Shirt Size: Medium

Manager: Janet

SSN: 1234567890

Performance Reviews

Bob doesn't make mistakes. Only happy accidents.

— Alice

★☆☆☆☆

Bob's great at building happy little clouds.

— Janet

★★★★★

localhost/bob

landing page service

bob (sign out)

Bob

"There's nothing wrong with having a tree as a friend."

details service

Employee Details

Name: Bob Ross

Birth Date: October 29, 1942

Position: Cloud Engineer

T-Shirt Size: Medium

Manager: Janet

SSN: 1234567890

reviews service

Performance Reviews

Bob doesn't make mistakes. Only happy accidents.

— Alice

★☆☆☆☆

ratings service

Bob's great at building happy little clouds.

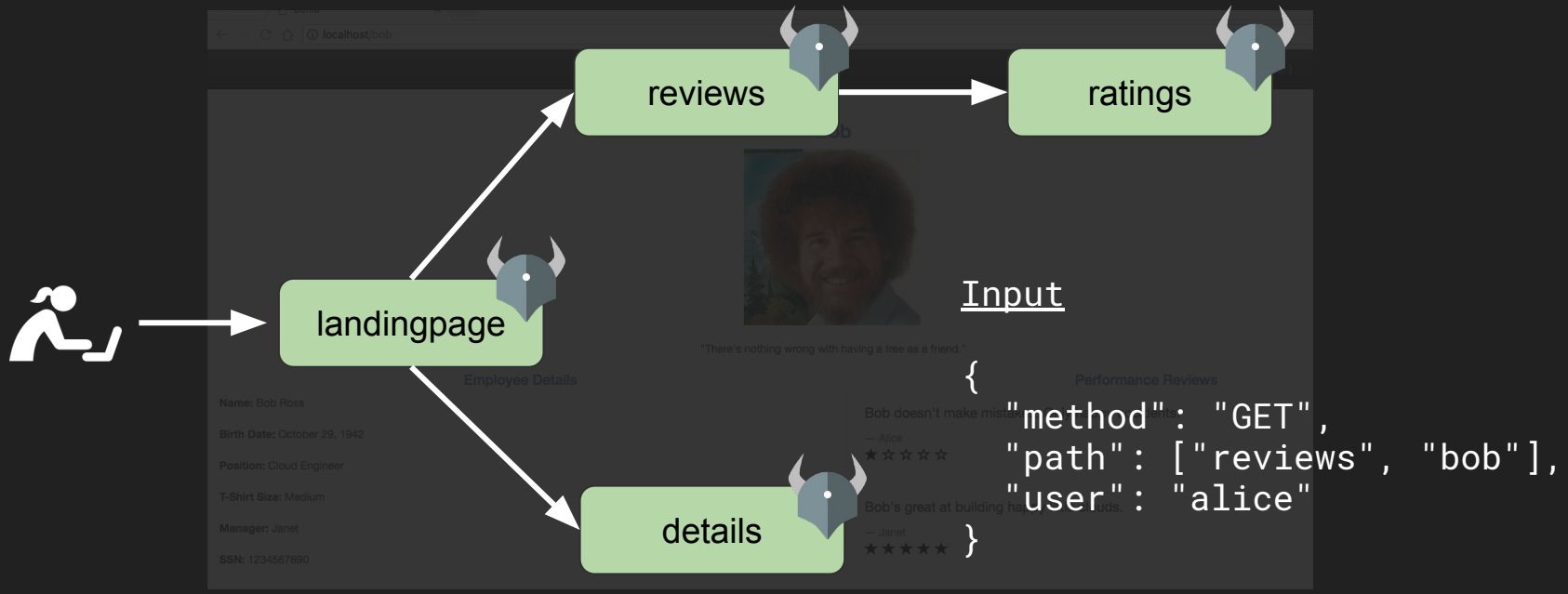
— Janet

★★★★★

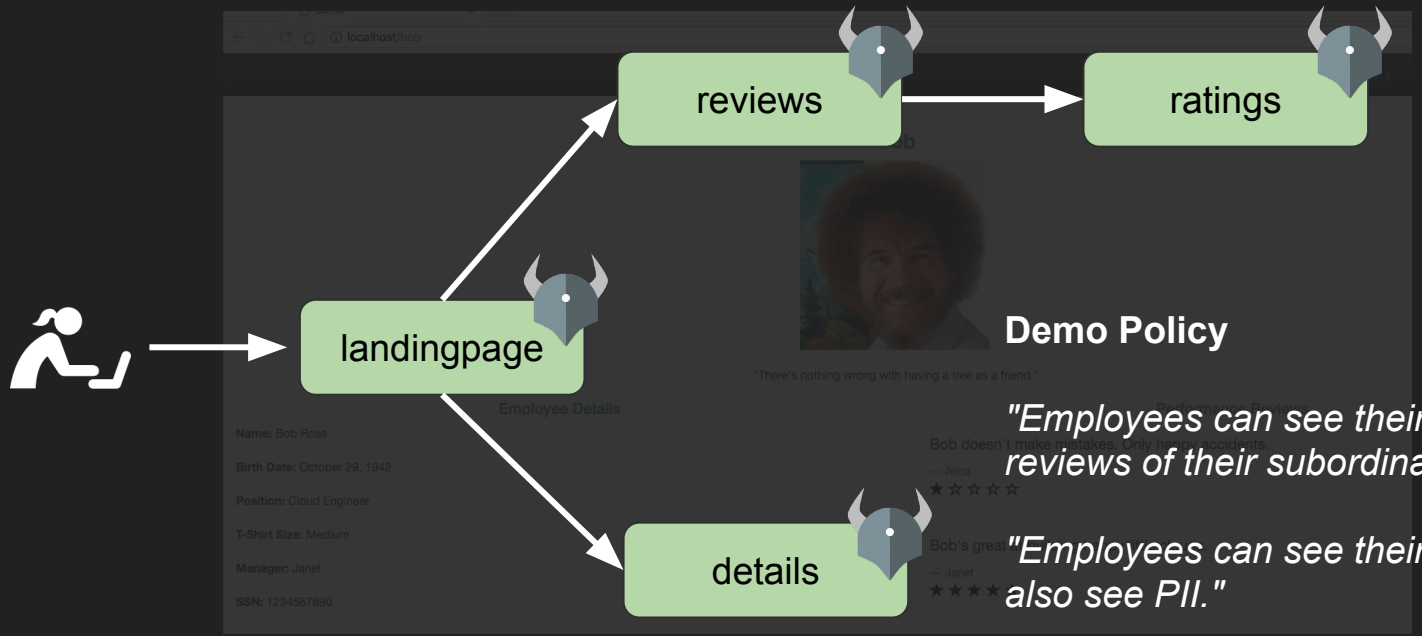
@sometorin

@OpenPolicyAgent

Demo: Authorization



Demo: Authorization



Demo Policy

"Employees can see their own reviews and the reviews of their subordinates."

"Employees can see their own PII. HR can also see PII."

Declarative Language (Rego)

- Is user X allowed to call operation Y on resource Z?
- Which annotations must be added to new Deployments?
- Which users can SSH into production machines?

"Employees may read their own reviews and the reviews of their subordinates."

"Employees may read their own reviews [...]"

"Employees may read their own reviews [...]"

Input

```
{"method": "GET",
 "path": ["reviews", "bob"],
 "user": "bob"}
```

"Employees may read their own reviews [...]"

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = employee_id
}
```

Input

```
{"method": "GET",
 "path": ["reviews", "bob"],
 "user": "bob"}
```

"Employees may read their own reviews [...]"

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", "bob"]
  input.user = "bob"
}
```

Input

```
{"method": "GET",
 "path": ["reviews", "bob"],
 "user": "bob"}
```

"Employees may read their own reviews [...]"

```
allow = true {
  input.method = "GET"                # OK
  input.path = ["reviews", "bob"]     # OK
  input.user = "bob"                  # OK
}
```

Input

```
{"method": "GET",
 "path": ["reviews", "bob"],
 "user": "bob"}
```

"Employees may read their own reviews [...]"

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = employee_id
}
```

Input

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

"alice" instead of "bob"

"Employees may read their own reviews [...]"

```
allow = true {
  input.method = "GET"                # OK
  input.path = ["reviews", "bob"]     # OK
  "alice" = "bob"                     # FAIL
}
```

Input

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

"alice" instead of "bob"

"Employees may read [...] the reviews of their subordinates."

```
allow = true {
  input.method = "GET"                # OK
  input.path = ["reviews", "bob"]     # OK
  "alice" = "bob"                     # FAIL
}
```

Input

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

"alice" instead of "bob"

"Employees may read [...] the reviews of their subordinates."

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = employee_id
}
```

Input

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

Data (in-memory)

```
{"manager_of": {
  "bob": "alice",
  "alice": "janet"}}
```

"Employees may read [...] the reviews of their subordinates."

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = employee_id
}

allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = data.manager_of[employee_id]
}
```

Input

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

Data (in-memory)

```
{"manager_of": {
  "bob": "alice",
  "alice": "janet"}}
```

"Employees may read [...] the reviews of their subordinates."

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = employee_id
}

allow = true {
  input.method = "GET"
  input.path = ["reviews", "bob"]
  input.user = data.manager_of["bob"]
}
```

Input

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

Data (in-memory)

```
{"manager_of": {
  "bob": "alice",
  "alice": "janet"}}
```

"Employees may read [...] the reviews of their subordinates."

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = employee_id
}

allow = true {
  input.method = "GET"
  input.path = ["reviews", "bob"]
  input.user = "alice"
}
```

Input

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

Data (in-memory)

```
{"manager_of": {
  "bob": "alice",
  "alice": "janet"}}
```

"Employees may read [...] the reviews of their subordinates."

```
allow = true {
  input.method = "GET"
  input.path = ["reviews", employee_id]
  input.user = employee_id
}

allow = true {
  input.method = "GET"                # OK
  input.path = ["reviews", "bob"]     # OK
  input.user = "alice"                # OK
}
```

Input

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

Data (in-memory)

```
{"manager_of": {
  "bob": "alice",
  "alice": "janet"}}
```

What about RBAC?

RBAC solves XX% of the problem.

"Allow all HTTP requests from 10.1.2.0/24."

"Restrict employees from accessing the service outside of work hours."

"QA must sign-off on images deployed to the production namespace."

"Restrict ELB changes to senior SREs that are on-call."

"Analysts can read client data but PII must be redacted."

RBAC is not enough.

"Give developers SSH access to machines listed in JIRA tickets assigned to them."

"Prevent developers from running containers with privileged security contexts in the production namespace."

"Workloads for euro-bank must be deployed on PCI-certified clusters in the EU."

...but everyone knows RBAC.

Implement RBAC with OPA.

Data (in-memory)

```
bindings:
  - user: inspector-alice
    role: widget-reader
  - user: maker-bob
    role: widget-writer
roles:
  - operation: read
    resource: widgets
    name: widget-reader
  - operation: write
    resource: widgets
    name: widget-writer
```

Implement RBAC with OPA.

```
allow = true {
  # Find binding(s) for user.
  binding := data.bindings[_]
  input.user = binding.user
```

Data (in-memory)

```
bindings:
  - user: inspector-alice
    role: widget-reader
  - user: maker-bob
    role: widget-writer
roles:
  - operation: read
    resource: widgets
    name: widget-reader
  - operation: write
    resource: widgets
    name: widget-writer
```

Implement RBAC with OPA.

```
allow = true {
  # Find binding(s) for user.
  binding := data.bindings[_]
  input.user = binding.user

  # Find role(s) with permission.
  role := data.roles[_]
  input.resource = role.resource
  input.operation = role.operation
```

Data (in-memory)

```
bindings:
  - user: inspector-alice
    role: widget-reader
  - user: maker-bob
    role: widget-writer
roles:
  - operation: read
    resource: widgets
    name: widget-reader
  - operation: write
    resource: widgets
    name: widget-writer
```

Implement RBAC with OPA.

```
allow = true {
  # Find binding(s) for user.
  binding := data.bindings[_]
  input.user = binding.user

  # Find role(s) with permission.
  role := data.roles[_]
  input.resource = role.resource
  input.operation = role.operation

  # Check if binding matches role.
  role.name = binding.role
}
```

Data (in-memory)

```
bindings:
  - user: inspector-alice
    role: widget-reader
  - user: maker-bob
    role: widget-writer
roles:
  - operation: read
    resource: widgets
    name: widget-reader
  - operation: write
    resource: widgets
    name: widget-writer
```

This rule *searches* over the RBAC data.

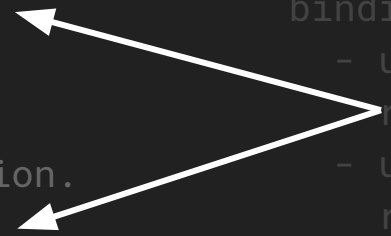
```
allow = true {
  # Find binding(s) for user.
  binding := data.bindings[_]
  input.user = binding.user

  # Find role(s) with permission.
  role := data.roles[_]
  input.resource = role.resource
  input.operation = role.operation

  # Check if binding matches role.
  role.name = binding.role
}
```

```
Data (in-memory)

bindings:
  - user: inspector-alice
    role: widget-reader
  - user: maker-bob
    role: widget-writer
roles:
  - operation: read
    resource: widgets
    name: widget-reader
  - operation: write
    resource: widgets
    name: widget-writer
```



Find bindings and roles that match input.

Partial Evaluation: rules + data ⇒ simplified rules

```
allow = true {
  # Find binding(s) for user.
  binding := data.bindings[_]
  input.user = binding.user

  # Find role(s) with permission.
  role := data.roles[_]
  input.resource = role.resource
  input.operation = role.operation

  # Check if binding matches role.
  role.name = binding.role
}
```

Data (in-memory)

```
bindings:
  - user: inspector-alice
    role: widget-reader
  - user: maker-bob
    role: widget-writer
roles:
  - operation: read
    resource: widgets
    name: widget-reader
  - operation: write
    resource: widgets
    name: widget-writer
```


```
allow = true {
  input.user = "bob"
  input.resource = "/widgets"
  input.operation = "write"
}

allow = true {
  input.user = "alice"
  input.resource = "/widgets"
  input.operation = "read"
}
```

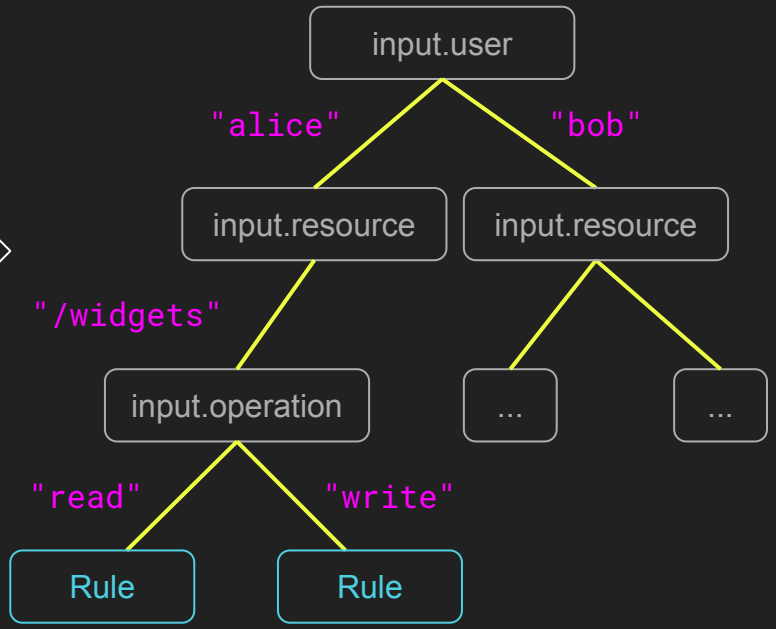
OPA builds an index from simplified rules.

```
allow = true { ... }
allow = true { ... }
allow = true { ... }
allow = true { ... }
allow = true { ... }

# Many rules (100s, 1000s)

allow = true {
  input.user = "alice"
  input.resource = "/widgets"
  input.operation = "read"
}
```

Rule Indexing



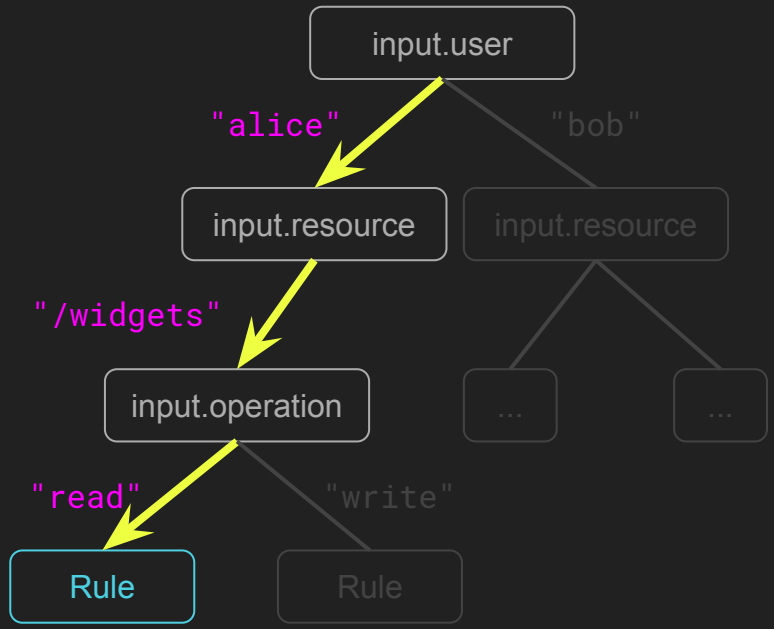
OPA uses the index to quickly find applicable rules.

Query

```
allow
```

Input

```
{
  "user": "alice",
  "resource": "/widgets",
  "operation": "read"
}
```



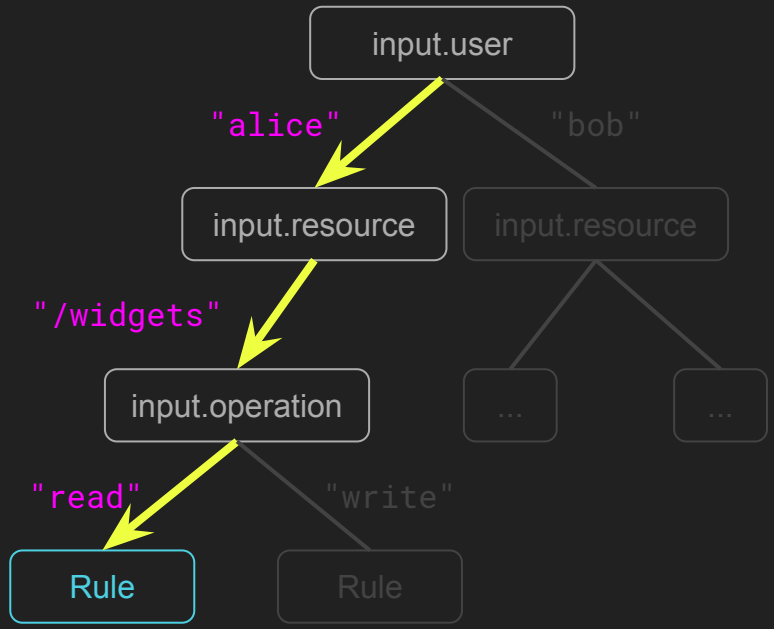
OPA only evaluates applicable rules.

```
allow = true { ... }
allow = true { ... }
allow = true { ... }
allow = true { ... }
allow = true { ... }

# Many rules (100s, 1000s)

allow = true {
  input.user = "alice"
  input.resource = "/widgets"
  input.operation = "read"
}
```

OPA ignores these.



# Roles	# Bindings	Normal Eval (ms)	With Partial Eval (ms)
250	250	5.50	0.0468
500	500	11.87	0.0591
1,000	1,000	21.64	0.0543
2,000	2,000	45.49	0.0624

blog.openpolicyagent.org

Partial Evaluation https://goo.gl/X6Qu6u
Rule Indexing https://goo.gl/uoSw3U

"Analysts can read client data but PII must be redacted."

"QA must sign-off on images deployed to the production namespace."

"Give developers SSH access to machines listed in JIRA tickets assigned to them."

"Restrict ELB changes to senior SREs that are on-call."

Use OPA to enforce policy across the stack.

It's all just data.

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

```
method: GET
path: /salary/bob
service.source:
  namespace: production
  service: landing_page
service.target:
  namespace: production
  service: details
user: alice
```

```
deny {
  not metadata.labels["qa-signoff"]
  metadata.namespace == "prod"
  spec.containers[_].privileged
}
```

```
metadata:
  name: nginx-149353-bvl8q
  namespace: production
spec:
  containers:
  - image: nginx
    name: nginx
    securityContext:
      privileged: true
  nodeName: minikube
```

```
deny {
  is_read_operation
  is_pii_topic
  not in_pii_consumer_whitelist
}
```

```
operation: Read
resource:
  name: credit-scores
  resourceType: Topic
session:
  principal:
    principalType: User
    name: CN=anon_producer,O=OPA
  clientAddress: 172.21.0.5
```


```
allow {
  score = risk_budget
  count(plan_names["aws_iam"]) == 0
  blast_radius < 500
}
```

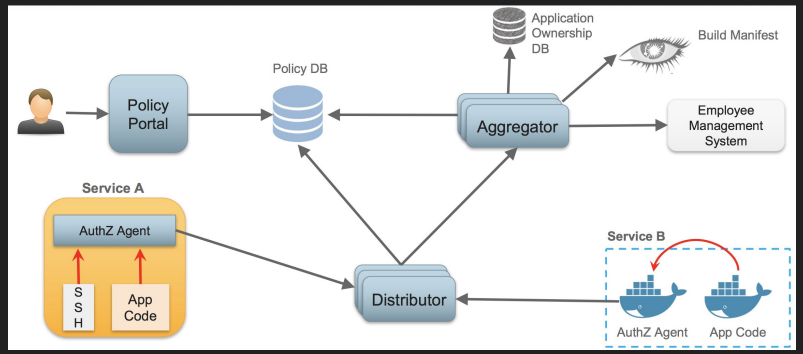
```
aws_autoscaling_group.lamb:
  availability_zones#: '1'
  availability_zones.3205: us-west-1a
  desired_capacity: '4'
  launch_configuration: kitten
  wait_for_capacity_timeout: 10m
aws_instance.puppy:
  ami: ami-09b4b74c
  instance_type: t2.micro
```

@sometorin

@OpenPolicyAgent

User Study: Netflix

- Complex environment
  - >1,000 services
  - Many resource and identity types
  - Many protocols, languages, etc.
- Key requirements
  - Low latency
  - Flexible policies
  - Ability to capture intent
- Using OPA across the stack
  - HTTP and gRPC APIs
  - Kafka producers
  - SSH (coming soon)

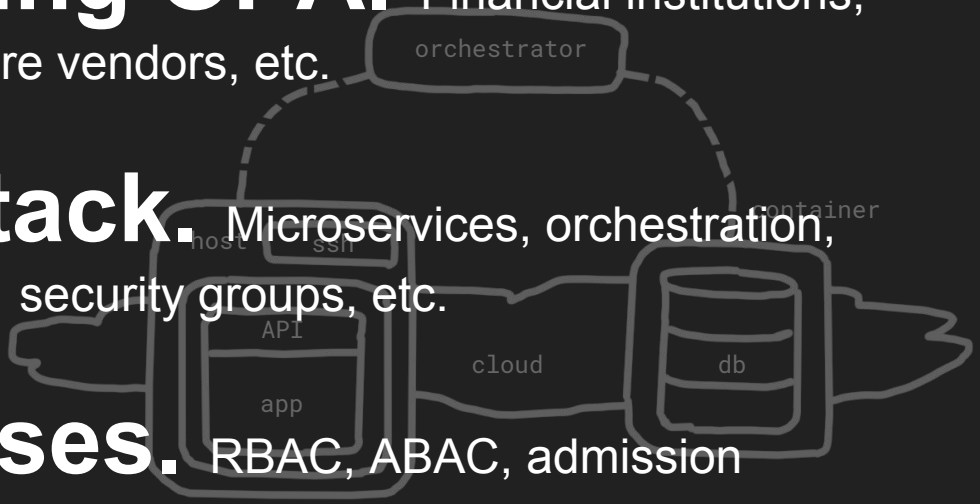


How Netflix is Solving Authorization Across Their Cloud (KubeCon US 2017)

20+ companies using OPA. Financial institutions, service providers, IT companies, software vendors, etc.

Used across the stack. Microservices, orchestration, provisioning, host daemons, data layer, security groups, etc.

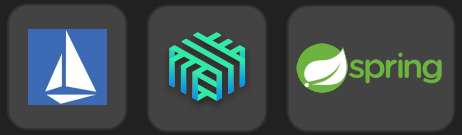
Bring more use cases. RBAC, ABAC, admission control, data protection, risk management, rate liming, auditing, etc.



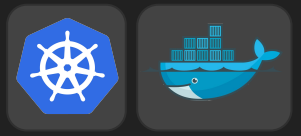
Demo

Policy decisions should be decoupled from policy enforcement.

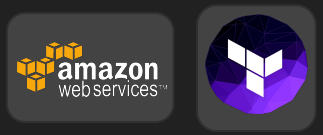
HTTP API Authorization



Admission Control



Risk Management



Try tutorials at openpolicyagent.org

Data Protection

SSH and sudo

@sometorin

@OpenPolicyAgent

Leverage OPA to solve fundamental policy and security problems.

Thank You!

open-policy-agent/opa

Star us on GitHub.

CLOUD NATIVE SANDBOX

@sometorin

@OpenPolicyAgent