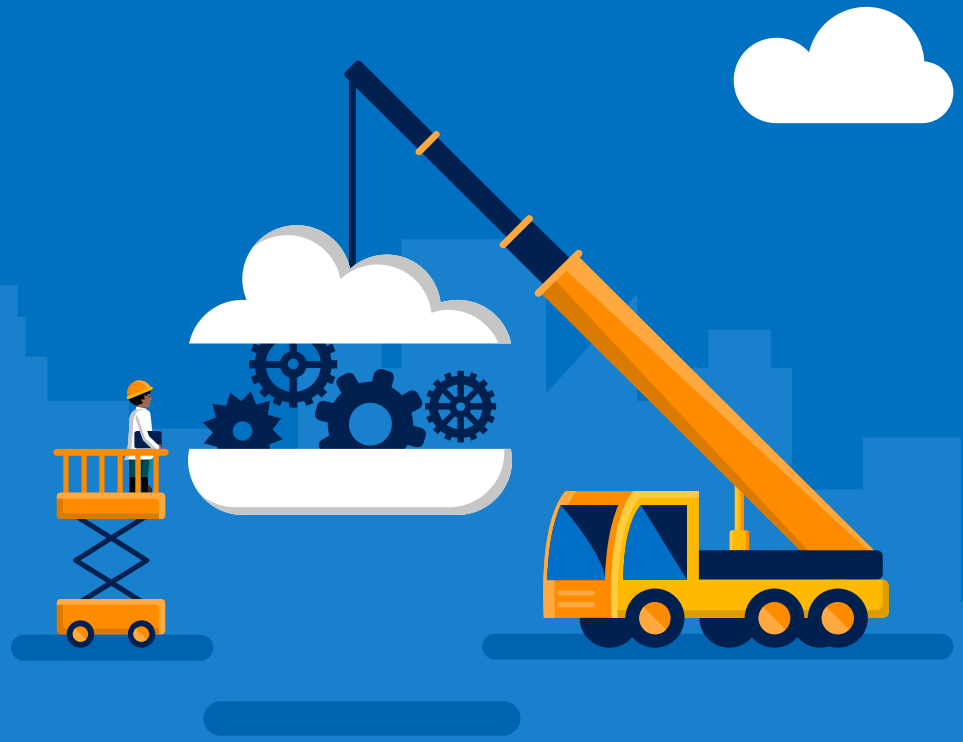
Microsoft

Email: CanadaOpenSource@microsoft.com

# Kubernetes Security with Calico and Open Policy Agent



Kevin Harris @kevbhar & Ray Kao @raykao
Open Source Solution Leads
Azure Global Black Belt
Microsoft Canada

# Network policy

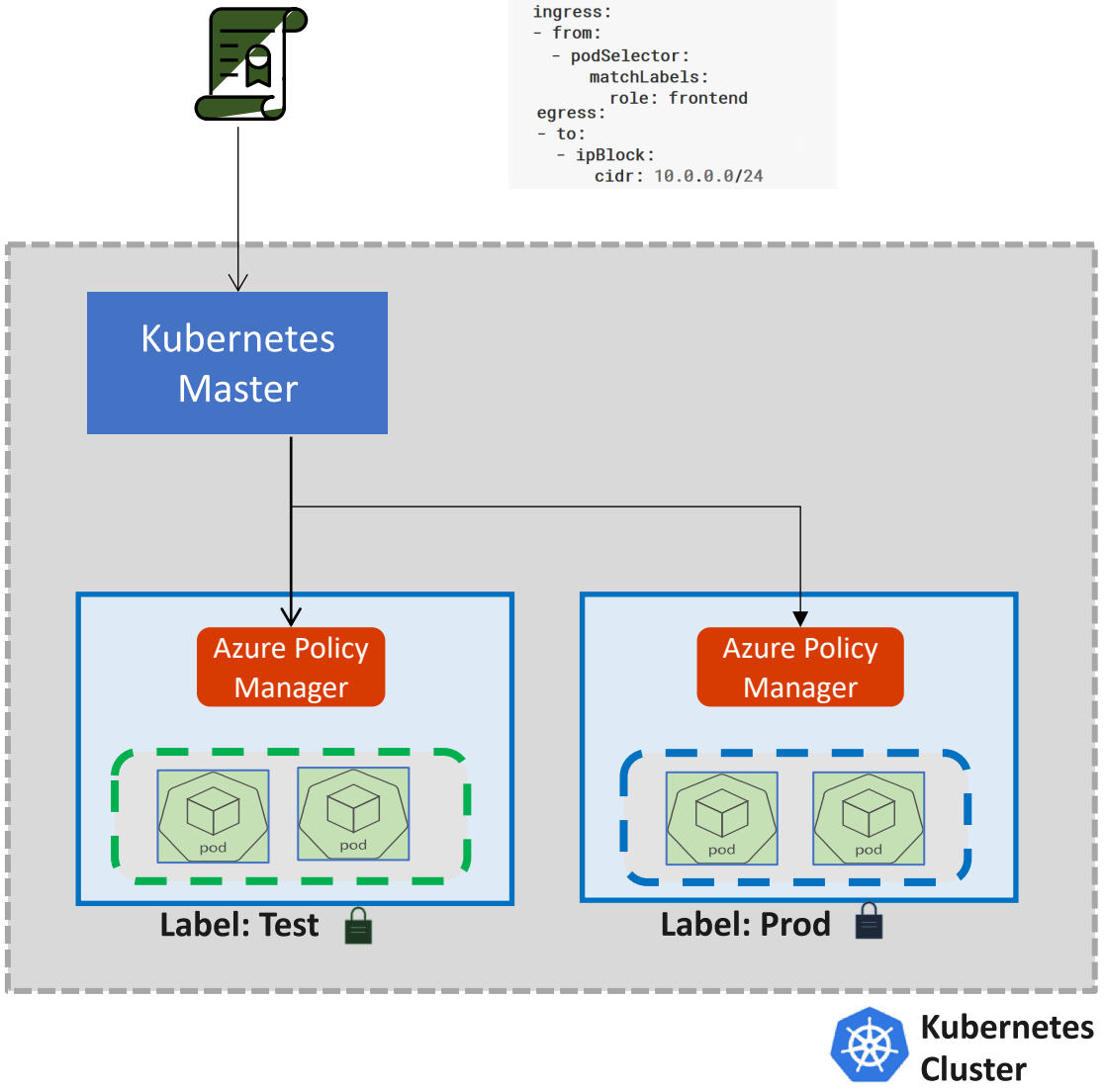
- All Pods are non-isolated by default
- Flat network. All pods can talk to other pods
- Accept traffic from anyone
- Multi stage/zone project this could expose security risks
  - 3 tier webapp.
  - Front end could technically talk directly to DB tier

# Azure Kubernetes Network Policies

- Provides micro-segmentation for containers – like NSGs for VMs
- Label-based selection of Pods
- Policy resource yaml file specifies Ingress and Egress policies
  - Policies defined for a label
- Works in conjunction with Azure CNI
- Supports Linux hosts
- Supported in *aks-engine*
  - Set `networkPolicy` setting to `azure` in cluster definition file
- Supported on AKS in **Preview**

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
```

```
kubectl apply -f
policy.yaml
```



# Network policy

- Pod Selector
- PolicyTypes
  - Ingress, egress

Ingress

- namespaceSelector
- podSelector
- ipBlock

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

# Network policy

- Recommend to create default policies that apply to all pods
  - Default deny all ingress traffic
  - Default deny all egress traffic.
  - Allow Specific Traffic (Ingress & Egress)

# Calico Global Network Policy

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: allow-tcp-6379
spec:
  selector: role == 'database'
  types:
  - Ingress
  - Egress
  ingress:
  - action: Allow
    protocol: TCP
    source:
      selector: role == 'frontend'
    destination:
      ports:
      - 6379
  egress:
  - action: Allow
```

# Calico Global Network Set

```
apiVersion:
projectcalico.org/v3
kind: GlobalNetworkSet
metadata:
  name: a-name-for-the-set
  labels:
    role: external-database
spec:
  nets:
  - 198.51.100.0/28
  - 203.0.113.0/24
```

```
...
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        user: alice
    podSelector:
      matchLabels:
        role: client
...
```

<- Single Rule where two conditions must be met

```
...
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        user: alice
  - podSelector:
      matchLabels:
        role: client
...
```

<- two conditions either/or must be met

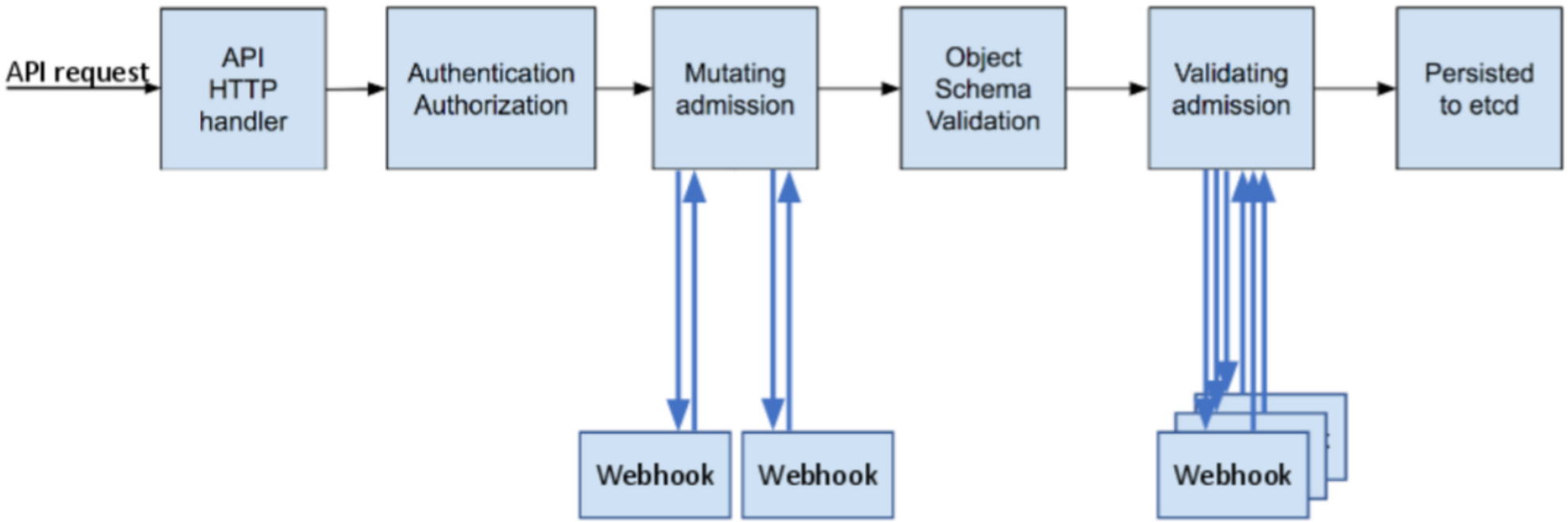
# When to use Azure NSG vs. Kubernetes Network policy

- Azure Network Security Groups
  - Use it to filter North-South traffic, that is, traffic entering and leaving your cluster subnet
- Kubernetes Network Policies
  - Use it to filter East-West traffic, that is, traffic between pods in your cluster

# Network Policy Demo

- Fruit Categorization App
- Web Frontend
- ML Backend for image recognition
- Signal R as a managed websocket service between FE/BE
- Default allow
- Create a policy (deny all by default)
- Open a policy to communicate between FE/BE services

# Open Policy Agent




Admission Controller Phases

# Dynamic Admission Control

- Validating Webhook
  - Allows you to intercept and validate requests
  - Can be run in parallel, as they don’t mutate objects
  - Example use case: restricting resource creation

- Mutating Webhook
  - Executes the mutation by sending requests to webhook server
  - Matching webhooks are called in serial
  - Example use case: injecting side cars

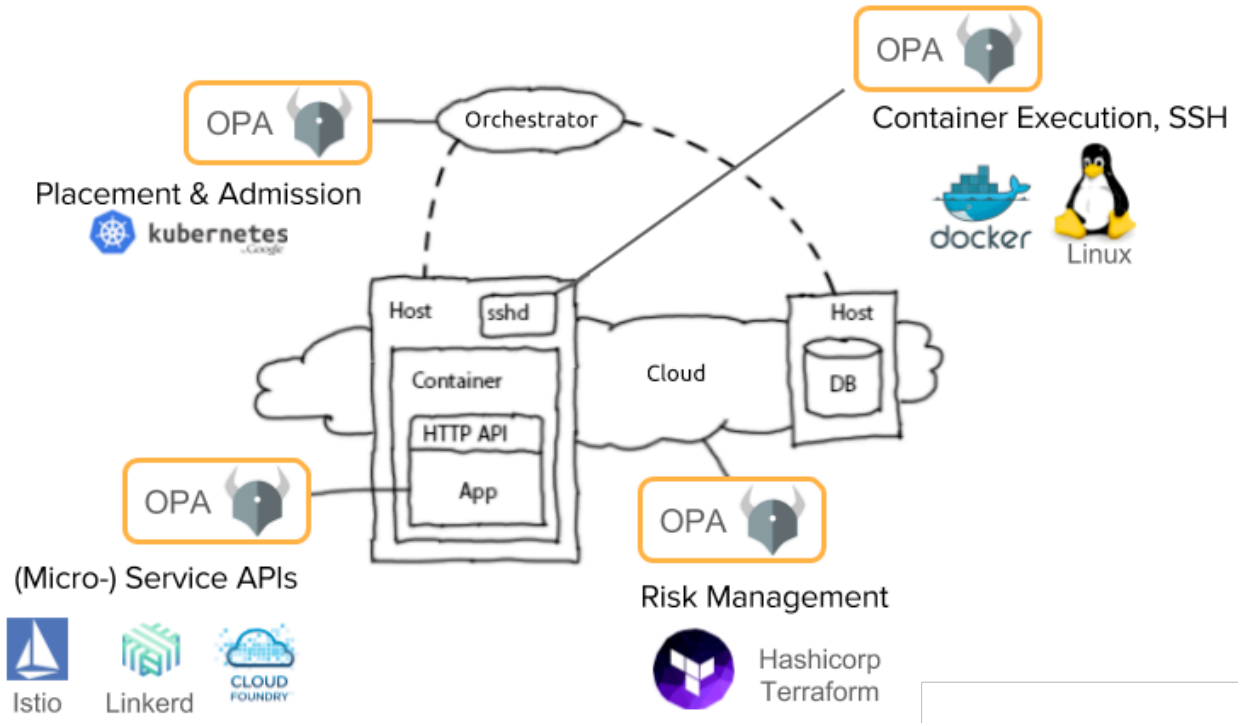
- Policy Enforcement
  - Admission Control is policy based on Kubernetes objects.
  - Network Policy and PodSecurity Policy focus on data plane policy
  - RBAC is policy enforced on the user

# That’s awesome! But…

# Sample Admission Webhook

```
181
182         http.HandleFunc("/services", serveServices)
183         http.HandleFunc("/mutating-services", serveMutateServices)
184         http.HandleFunc("/healthz", serveHealthz)
185         clientset := getClient()
186         server := &http.Server{
187                 Addr:      fmt.Sprintf(":%s", Options.PortNumber),
188                 TLSConfig: configTLS(clientset, &certKey),
189         }
190
191         glog.V(2).Infof("starting webserver on port %s", Options.PortNumber)
192         glog.V(2).Infof("service annotation to match/mutate: %s: %s", Options.ServiceAnnotationKey, Options.ServiceAnnotationV
193
194         if err := server.ListenAndServeTLS("", ""); err != nil {
195                 glog.Fatal(err)
196         }
197
198 }
```

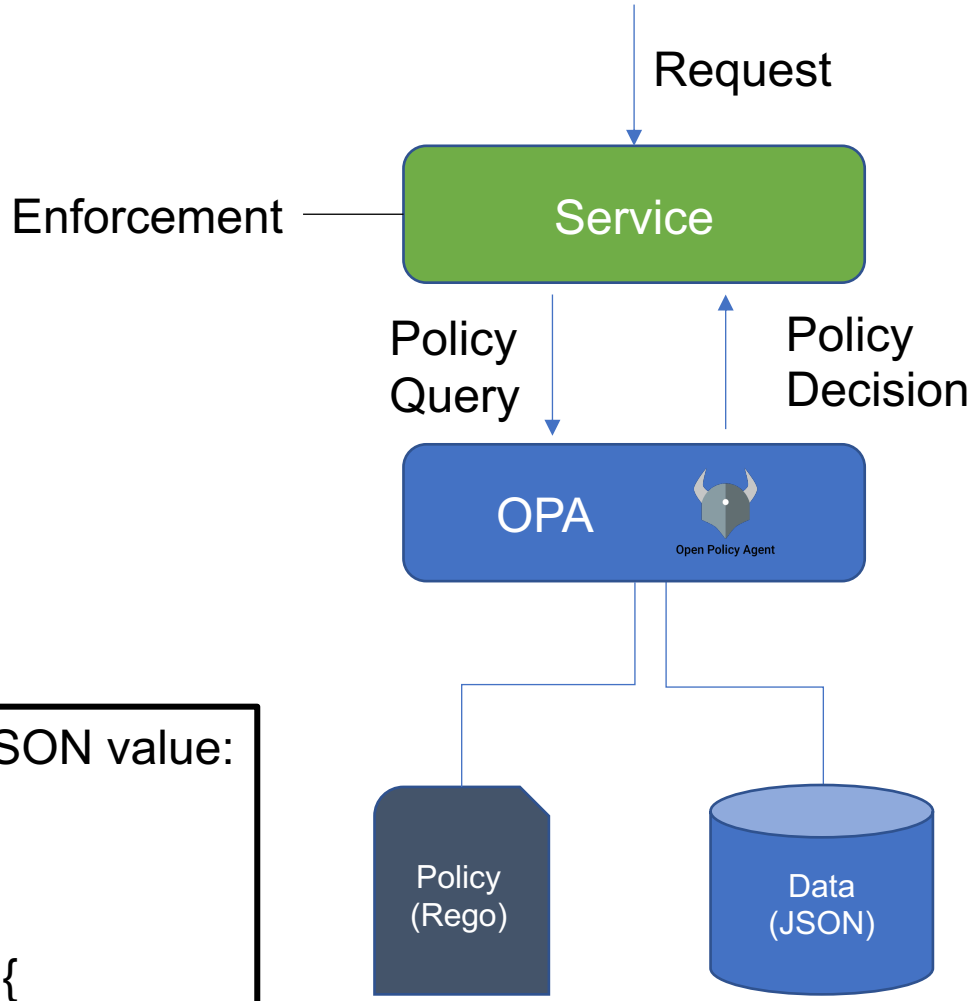
# Open Policy Agent




- CNCF Hosted Sandbox Project
- General purpose policy engine
- Can be used across the stack
- Declarative policy language (Rego)

Image: openpolicyagent.org

Open Policy Agent




**Service** refers to:

- Kubernetes API
- Custom API
- SSH Daemon
- Terraform
- Authorization APi

**Input** can be any JSON value:

```
"kind": "Service",
   "metadata": {
      "annotations": {
           department: dev
       }
```

**Output** can be any JSON value:

```
”true
   ”request annotated”
“   “annotations": {
          costCenter: 8000
    }
```

Diagram rewritten from:
www.openpolicyagent.org

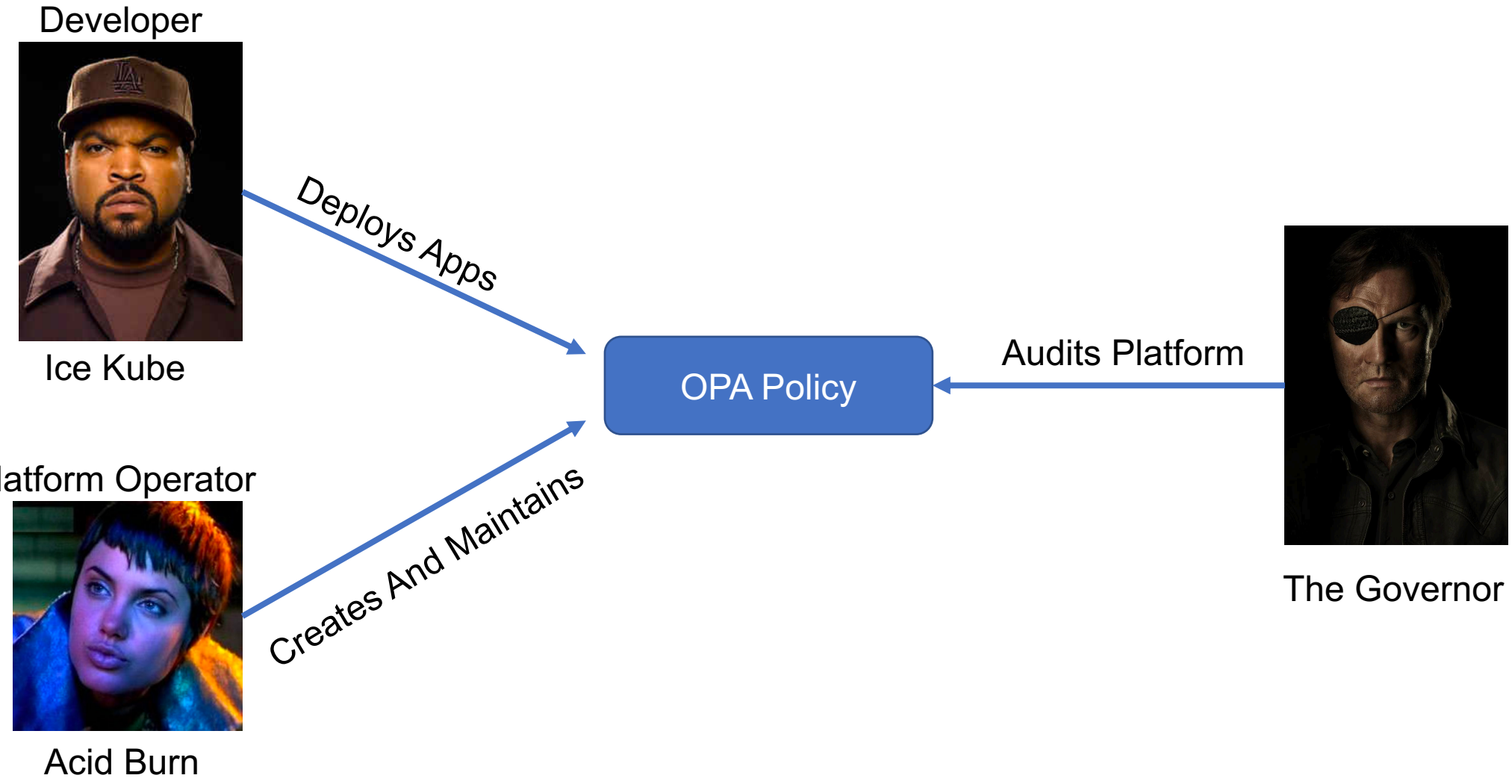
# Example Rego Policy

- Rego is a policy language and not a programing language, so don't think about sockets, methods, binary trees, etc.
- Think about two things: Logic and Data
- Rego logic is all queries. A query finds values for variables that make boolean conditions true.
- You write logic to search and combine JSON/YAML data from different sources.

```
deny[{
"id": "conditional-annotation",
"resource": {"kind": kind, "namespace": namespace, "name": name},
"resolution": {"patches": p, "message" : "conditional annotation"}, }] {

matches[[kind, namespace, name, matched_object]] matched_object.metadata.annotations[”Mr-T"]
p = [{"op": "add", "path": "/metadata/annotations/cost-center", "value": ”A-Team"}] }
```

# Who manages all this policy?



# Kubernetes Policy Controller

- Kubernetes Policy Controller
  - Moving to OPA org, as a standard Kubernetes Policy Controller
  - Authorization module makes it possible to implement a blacklist in front of RBAC
  - Provides auditing features
  - Deployment consist of three containers: OPA, kube-mgmt., and Controller
- Examples:
  - Whitelist / blacklist registries.
  - Not allow conflicting hosts for ingresses.
  - Label objects based on a user from a department.
  - Block kubectl exec <pod>

# The Good, The Bad, and Gotchas

- *Good*
  - OPA approach allows you to decouple policy from your applications
  - General purpose, so can be used outside of Kubernetes context.
- Bad
  - There can be a learning curve to Rego.
  - Can cause latency, but's negligible for most apps. (more of a consideration)
- Gotchas
  - Mutating objects need to be handled with care. They can cause unexpected behavior to what the end-user expects.

# Takeaways

- Focus on security is a ***must*** in any Kubernetes deployment.
- *Help* ***educate*** Security Teams on how to extend Kubernetes to integrate custom policies.
- Treat the Kubernetes cluster as ***immutable,*** just like you do with applications.
- Multiple ways to accomplish policy
  - **Build all your own logic** and utilize dynamic admission control
  - **Utilize Open Policy Agent to simplify** deployment and logic for rule sets.

# Example Policies

- https://github.com/open-policy-agent/contrib