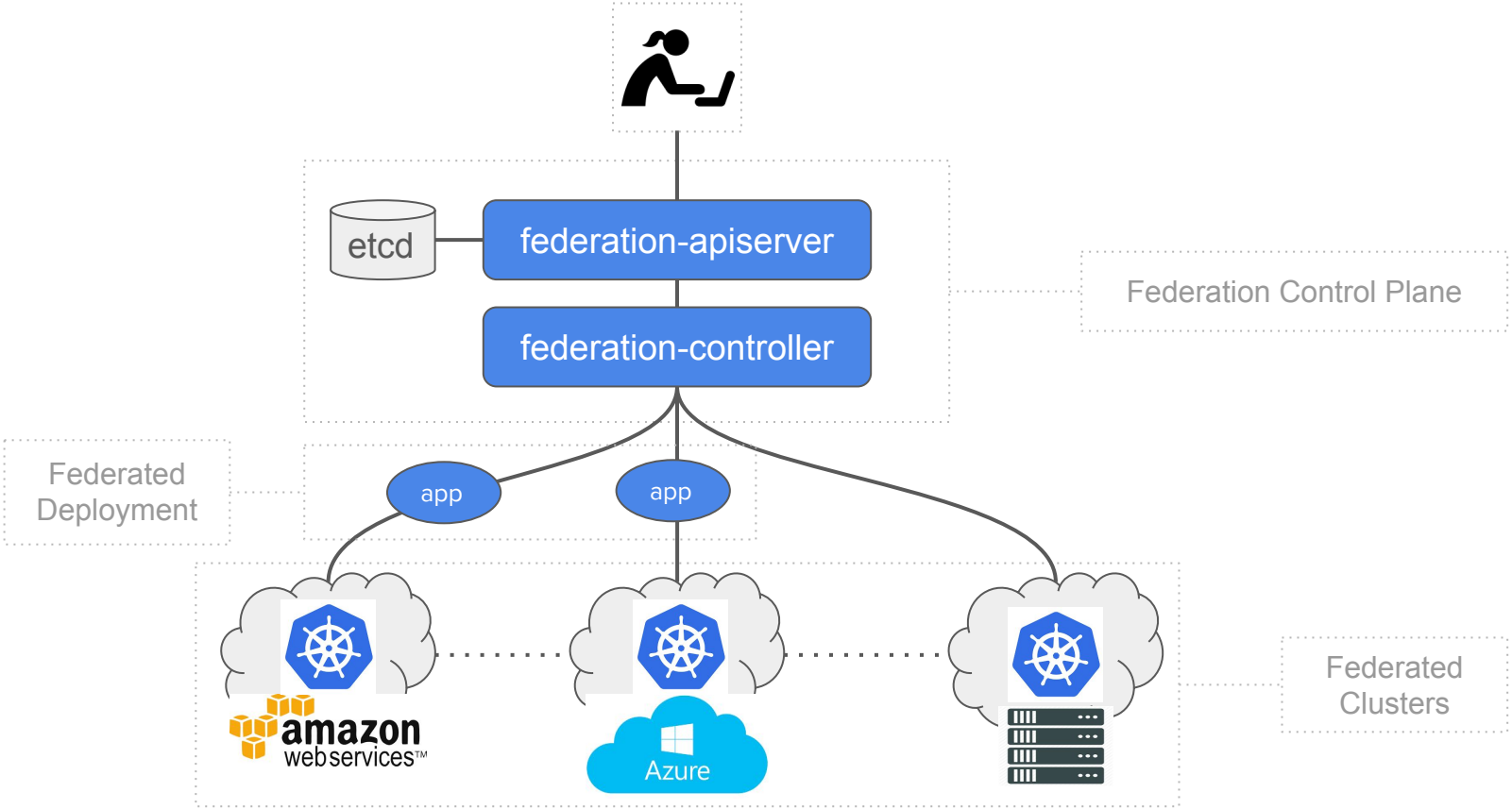
Policy-based Resource Placement

...across Hybrid-Cloud Federations of Kubernetes Clusters

Irfan UR Rehman (Huawei)
@irfanurrehman

Torin Sandall (Styra)
@sometorin

Federation: Overview

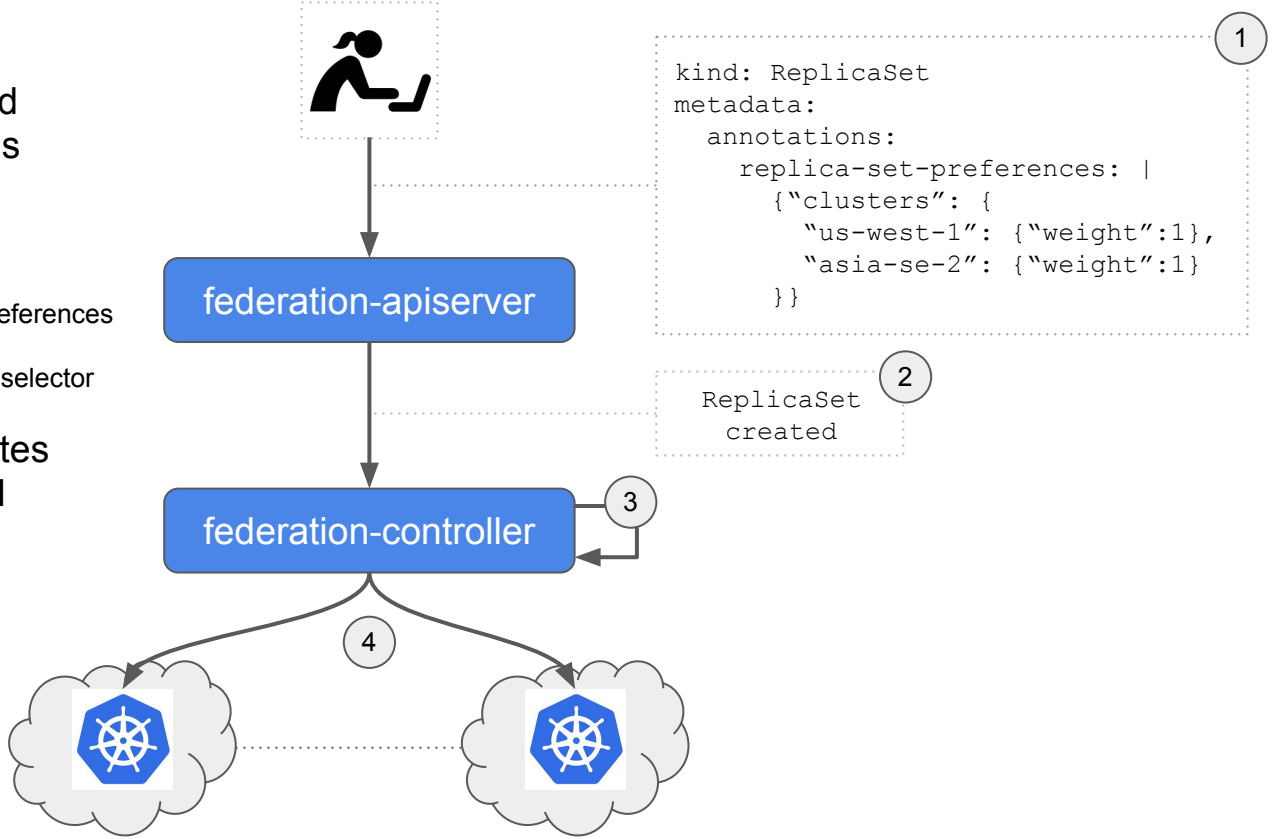


Federation: Placement

- Placement can be controlled per-resource via annotations
- 2 annotations supported:

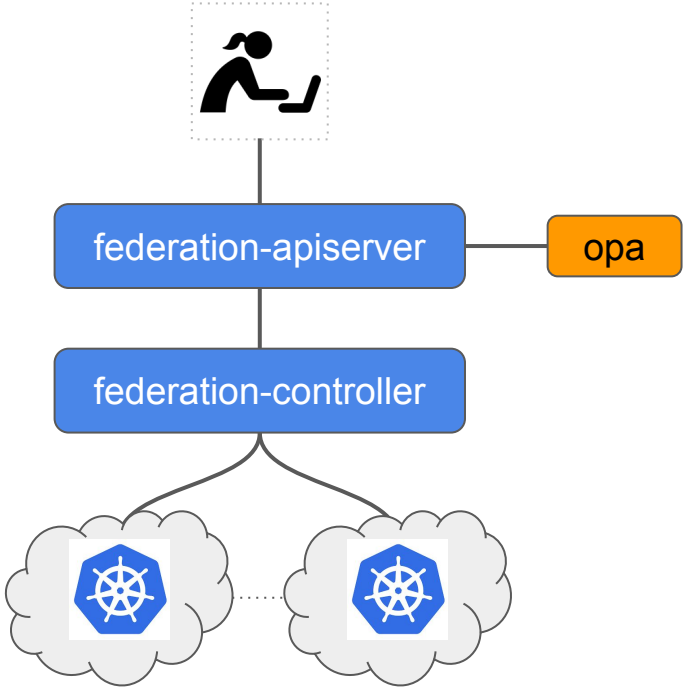
  federation.kubernetes.io/replica-set-preferences

  federation.alpha.kubernetes.io/cluster-selector
- federation-controller evaluates annotations to produce final placement



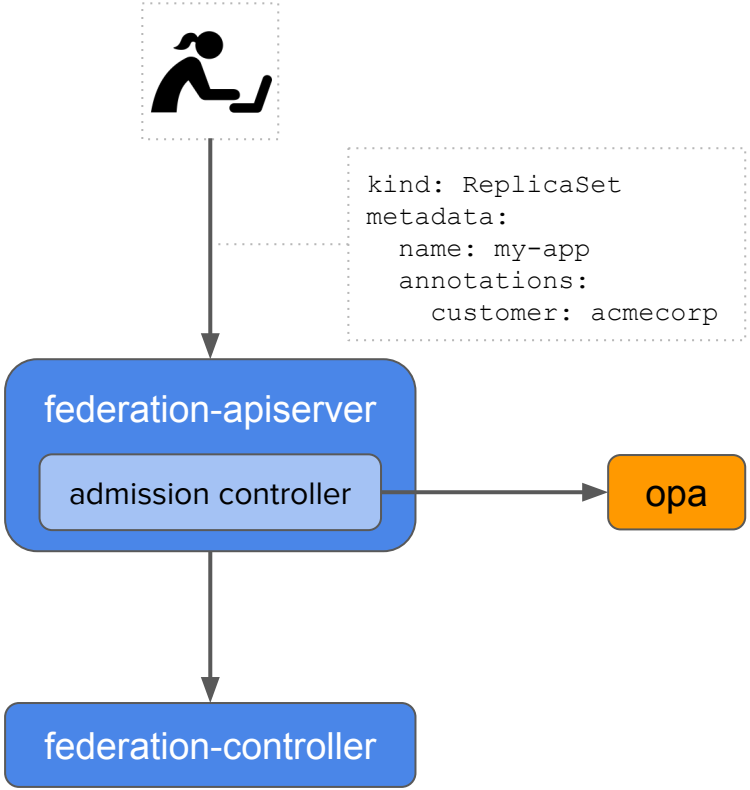
Policy-based Placement

- Resource placement is a “policy-rich” problem space
  - Legal regulation, cost, technical constraints, internal conventions, etc.
- Goal: give admins greater control and flexibility
  - Automated & programmable
  - Expressiveness
  - Leverage context
  - Ease management
- Decouple developer intent from admin policy
  - Avoid duplication
  - Prevent (and detect) violations
  - Abstract policy implementation
- Policy Engine decides which clusters app runs on
  - Pluggable
  - Simple interface



Architecture

- Admission Controller inside federation-apiserver queries Policy Engine when resources are created or updated
- Admission Controller implements “fail-closed” model in case query fails.

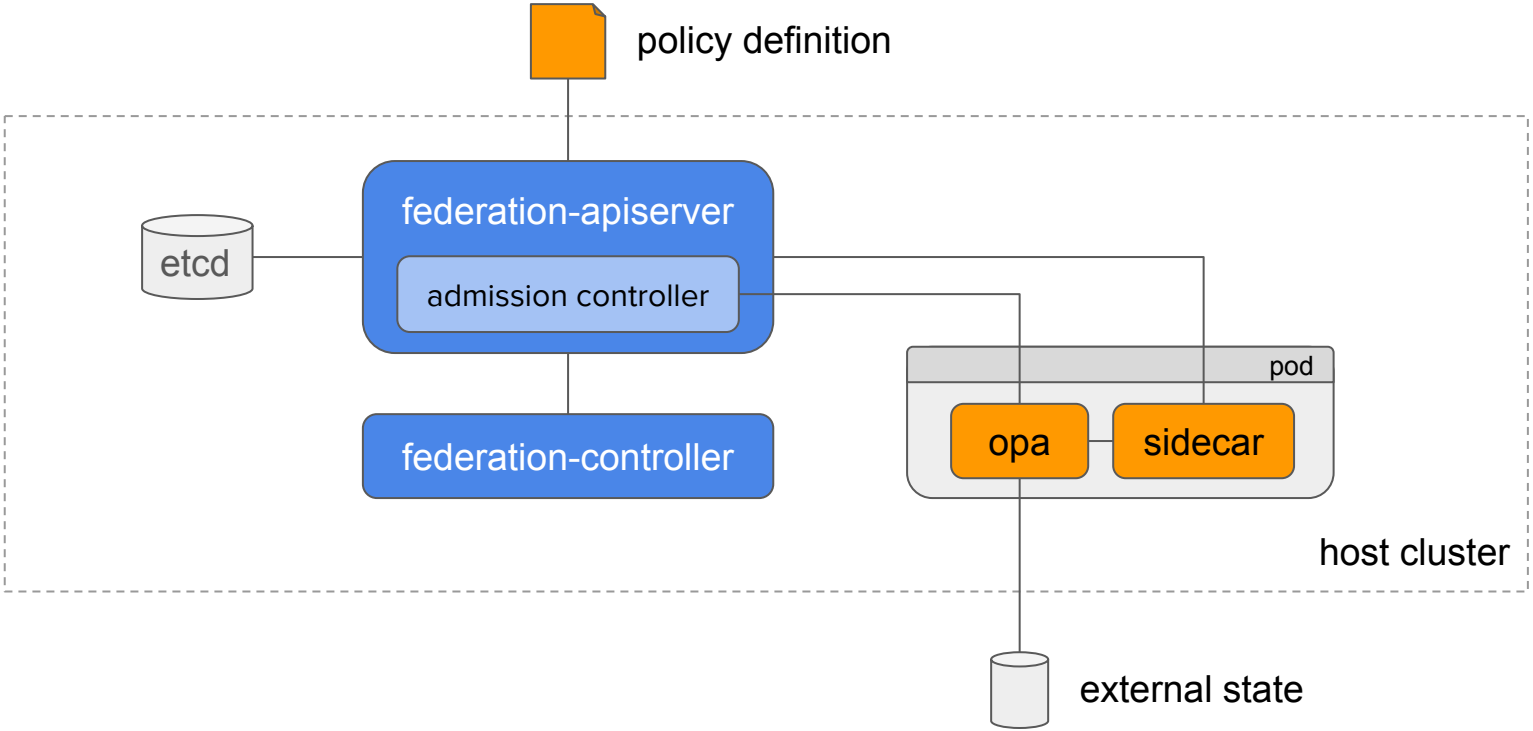


Example Query

```
POST /v1/data/k8s/placement

input:
  resource:
    metadata:
      name: my-app
      annotations:
        customer: acmecorp
    ...
```

```
HTTP/1.1 200 OK

result:
  annotations:
    replica-set-preferences:
      clusters:
        us-west-1: {weight: 1}
        asia-se-3: {weight: .5}
```

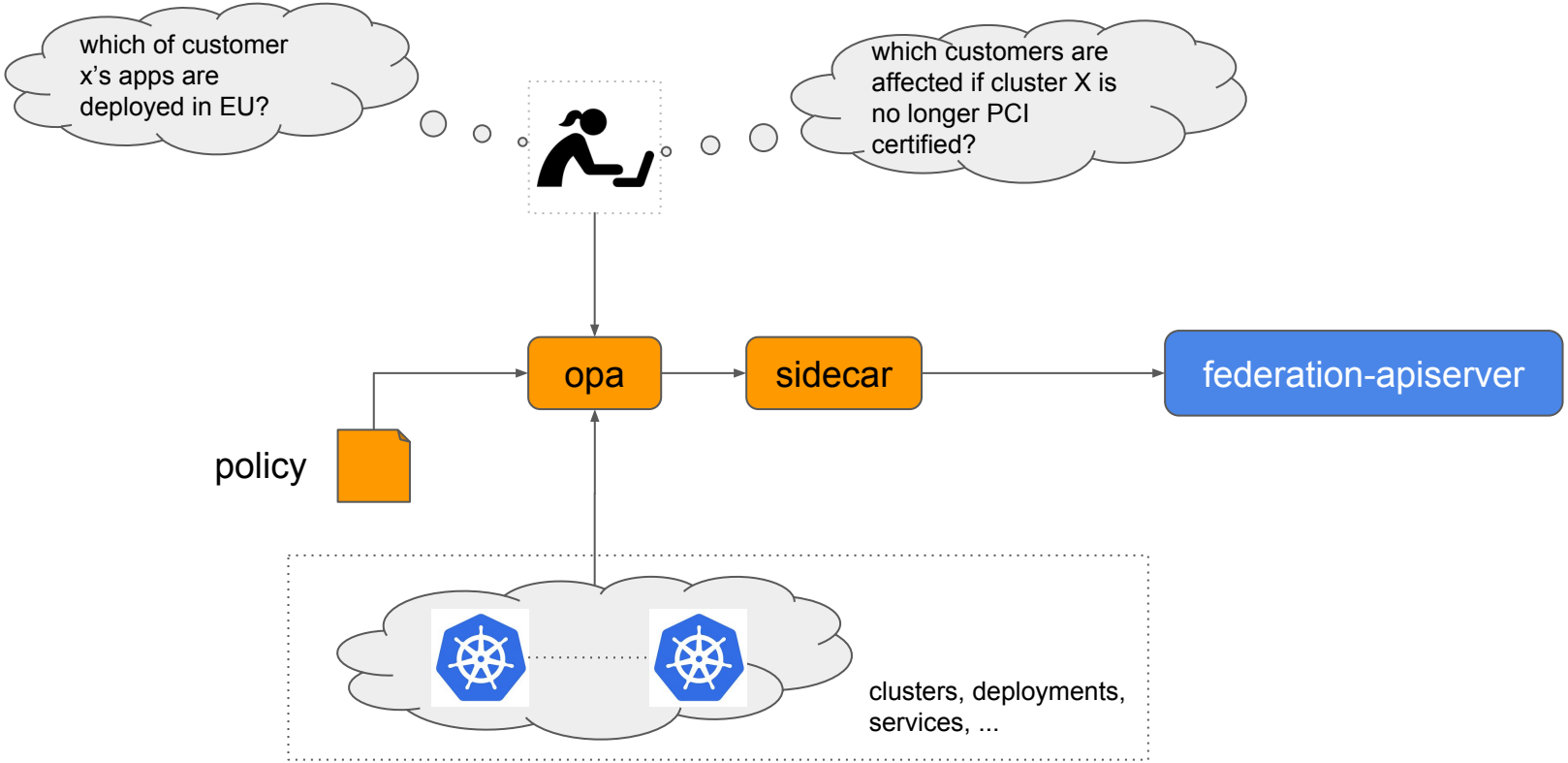
Architecture



Example

*Apps labelled with **customer name**. Customers are associated with a **jurisdiction**. Furthermore, apps may be labelled with **criticality**. If low then **public cloud** clusters may be used, otherwise, **on-prem** clusters must be used.*

Visibility & Remediation



Conflicts

```yaml
kind: ReplicaSet
metadata:
  annotations:
    customer: acmetel-US
    criticality: low
    replica-set-preferences:
      clusters:
        - us-west-2
        - eu-central-1
        ...
```

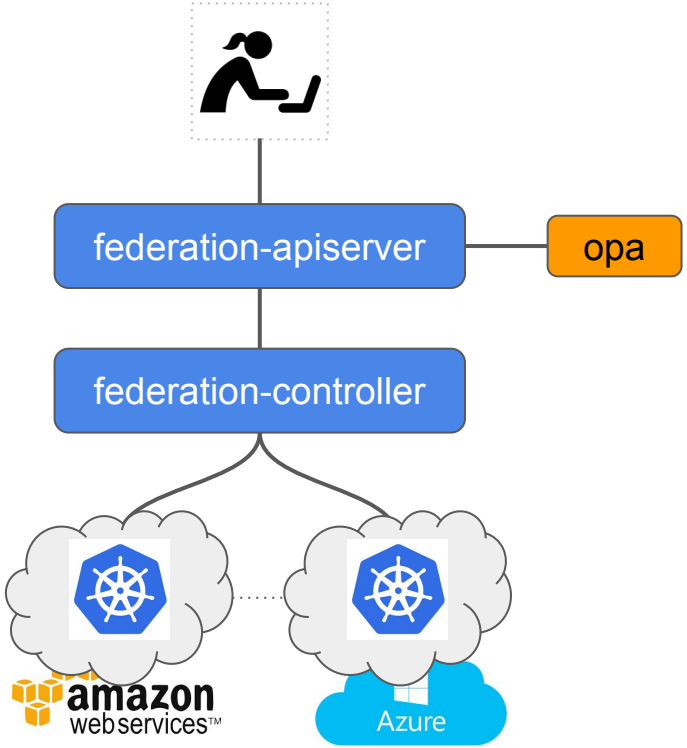
```
not_allowed[cluster] {
  requested_clusters[cluster]
  not allowed_clusters[cluster]
}

errors["invalid cluster(s)"] { not_allowed != set() }
```

- Developers could accidentally specify conflicting intent (result: empty set)
- Developers could explicitly request invalid clusters (result: error)
- Resolve conflicts within policy engine whenever possible
  - Policy is the only place where all intent is known

Future Work

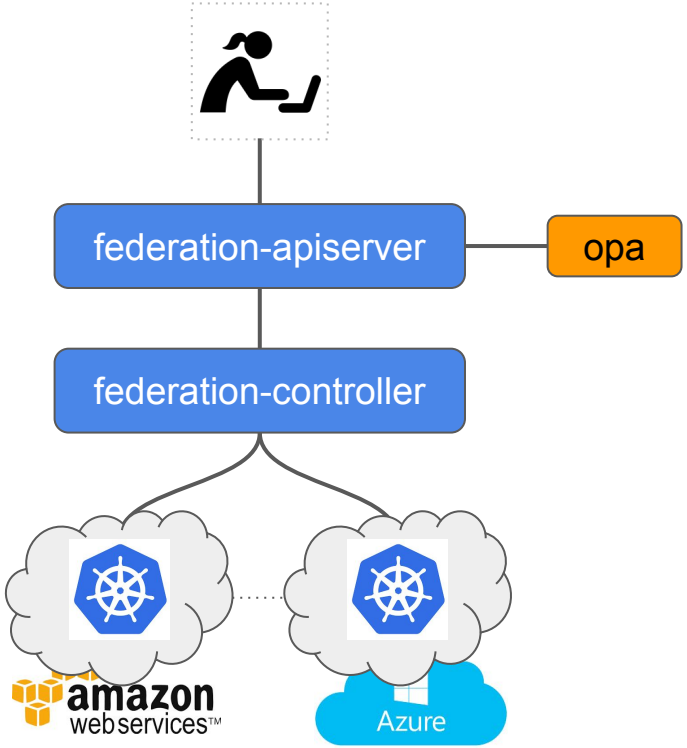
- Improve policy management
  - Current: policies stored as ConfigMaps in the federation-apiserver
  - Future: policies represented as first-class API objects
  - Cleaner mechanism for reporting policy enforcement status
    - Installed, errors, etc.

- Demonstrate new use cases
  - Cost-based policies
    - Replicate external data representing resource pricing (e.g., cpu, memory, etc.)
    - Pick clusters based on pricing data
    - Cluster inter-connect may be expensive



Conclusion

- Kubernetes Federation enables hybrid-cloud deployments for a variety of use cases
- Resource placement is a policy-rich problem that must address important business requirements
- Policy solution should empower admins with greater control and flexibility



Thank You!

SIG-Federation

Open Policy Agent (OPA)

github.com/open-policy-agent/opa