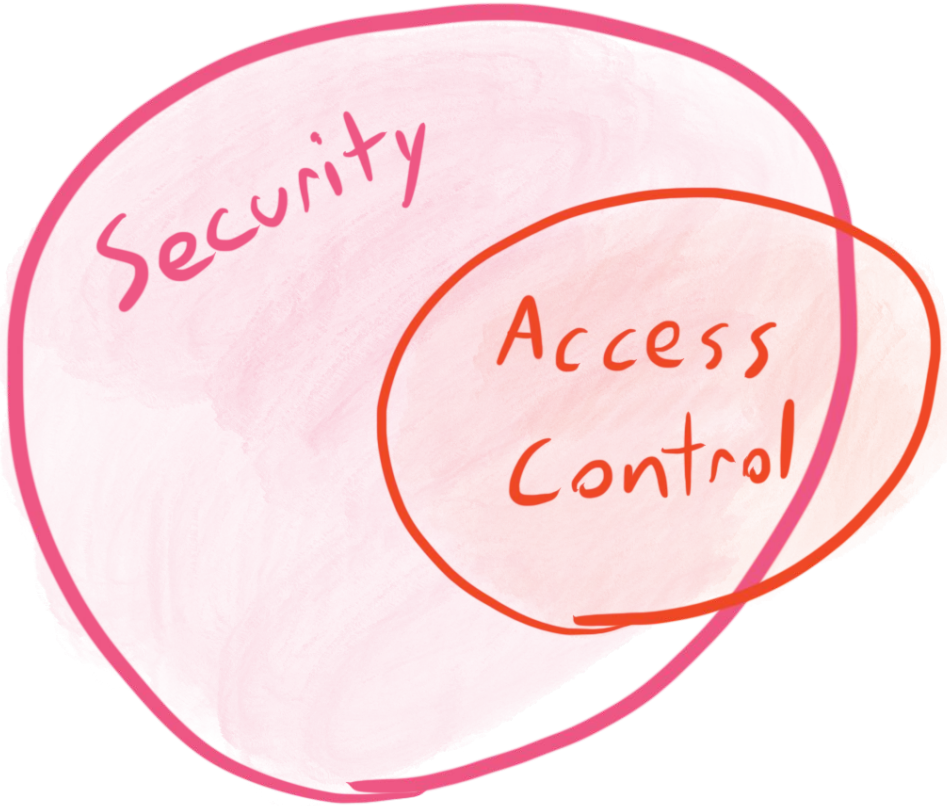

# Access Control In Kubernetes

## What's Missing, And How To Fix That

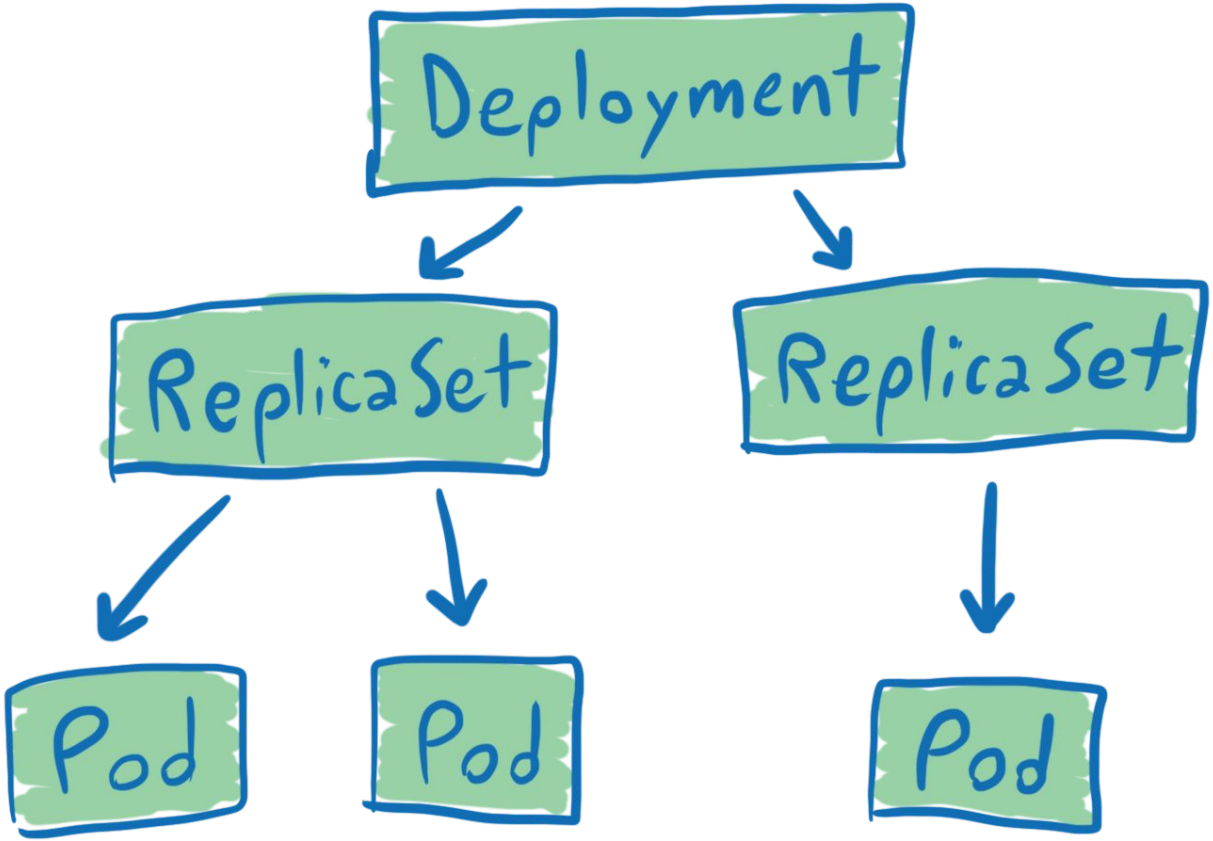
@vllry @SethMcCombs


Security
Access
Control

# Disclaimer



- This is not about Lyft systems
- This is not about Triller systems
- We might be wrong


Deployment
ReplicaSet
ReplicaSet
Pod
Pod
Pod

# Broad Categories



- Runtime characteristics
  - What can run, and in what way?
- Network access
  - What can access what?
- Meta: Kubernetes access
  - What can the app do to the Kubernetes control & data planes?

# What Can We Do About All This?

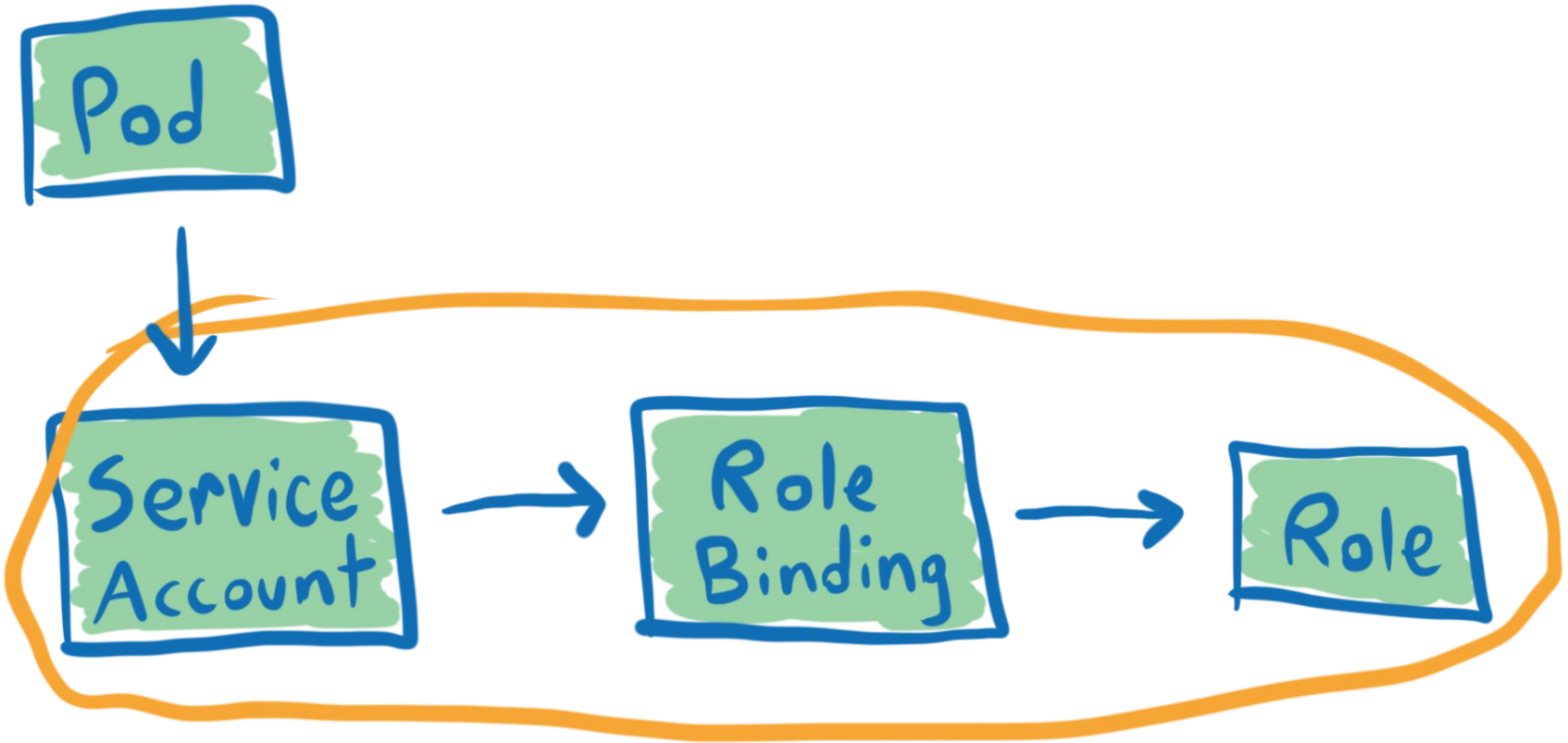


- RBAC
- Network Policies (native, Calico, Istio, etc)
- Custom API gateways
- Validating & Mutating admission webhooks (native)
- Open Policy Agent (addon)

# Essentials: RBAC

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: vallery
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```


Pod
Service Account
Role Binding
Role

```
$ kubectl getrole deployments

DEPLOYMENTS  ROLES     PERMISSIONS
my-ingress   ingress   watch endpoints, list ingress
```

# Other Forms of K8s API Access Control



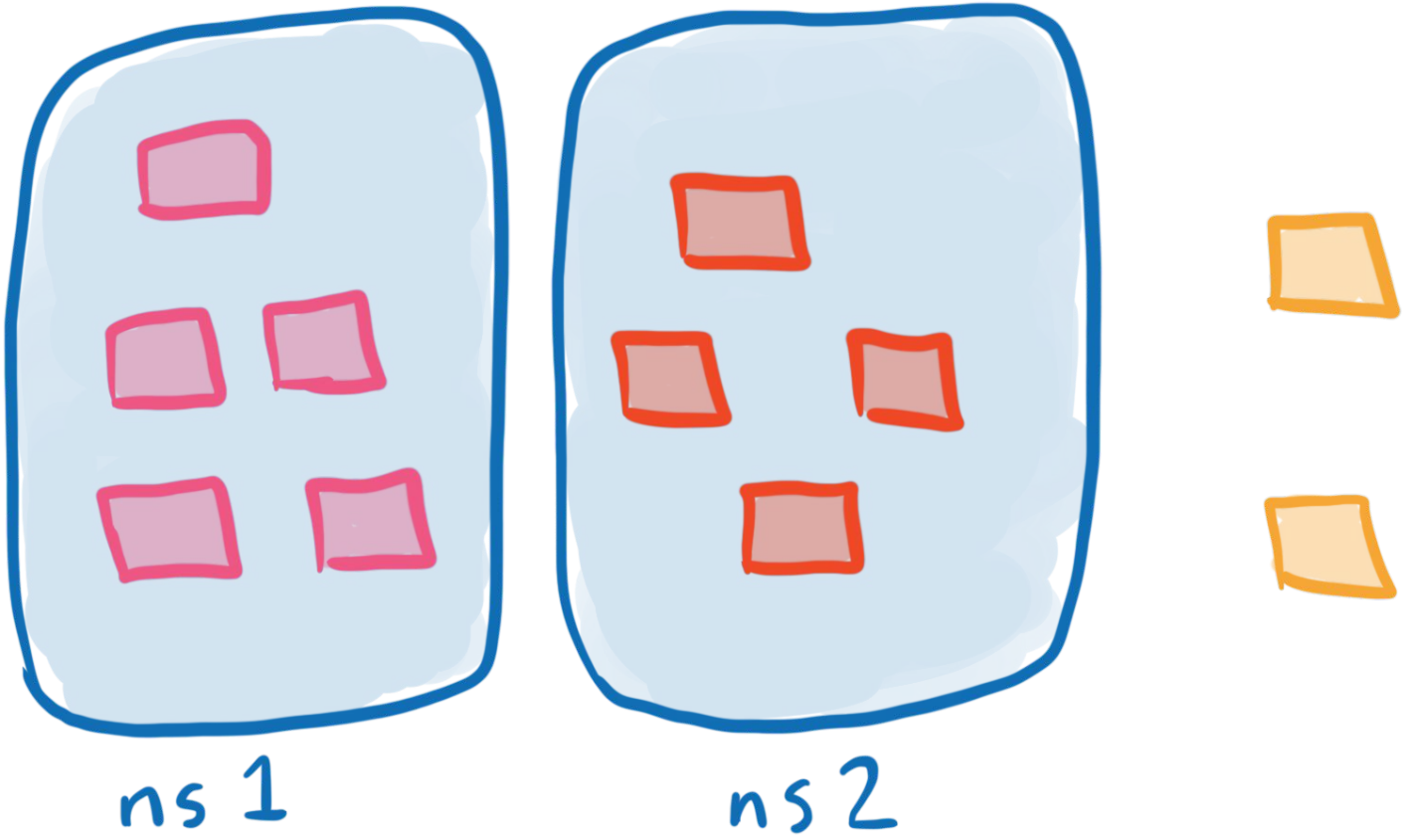
- Node
  - For kubelets only. Kubelet is granted permissions based on the pods it runs.
- ABAC (old)
  - Granular user-based permissions.
- Webhook
  - Post to URL, use that response.

# RBAC Limitations

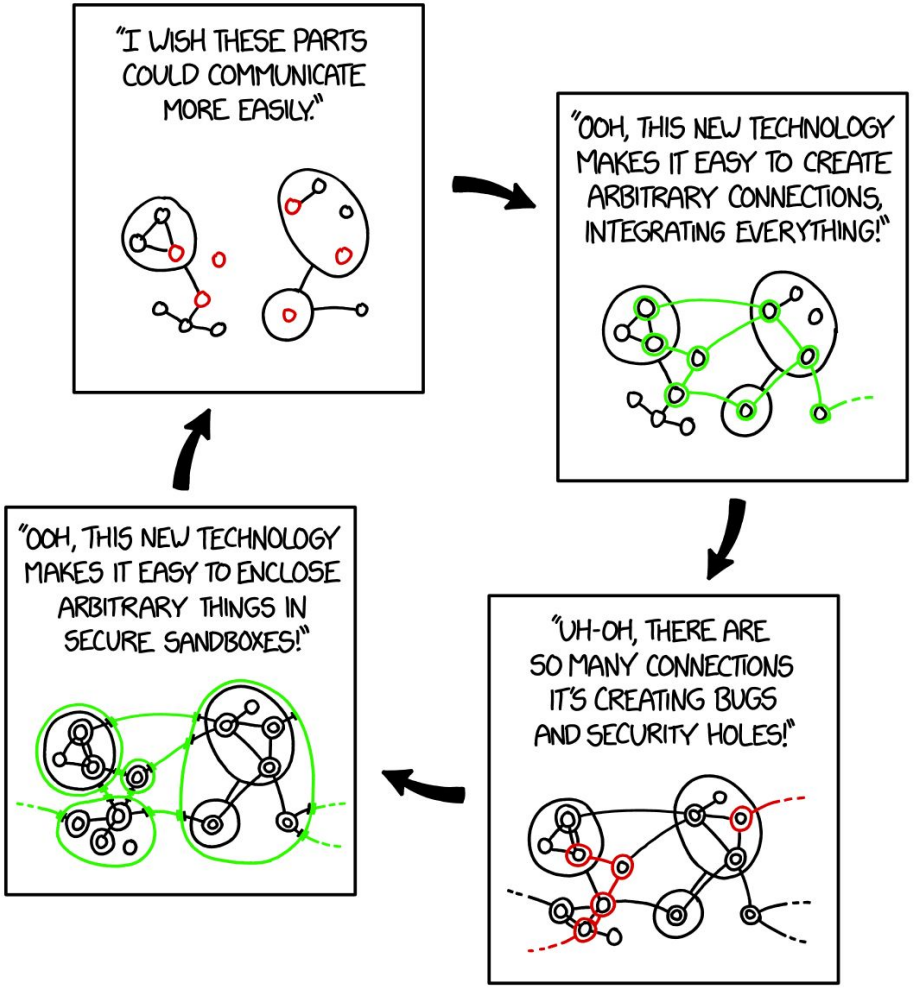


- Universal permission by resource type.
  - EG: “pods” means **all pods** in the namespace, from **any deployment**.
- No field-level access control.
  - EG: to edit the labels of an object, a user needs full write permission to the object.

# Essentials: Namespaces


ns 1
ns 2

“I wish these parts could communicate more easily.”



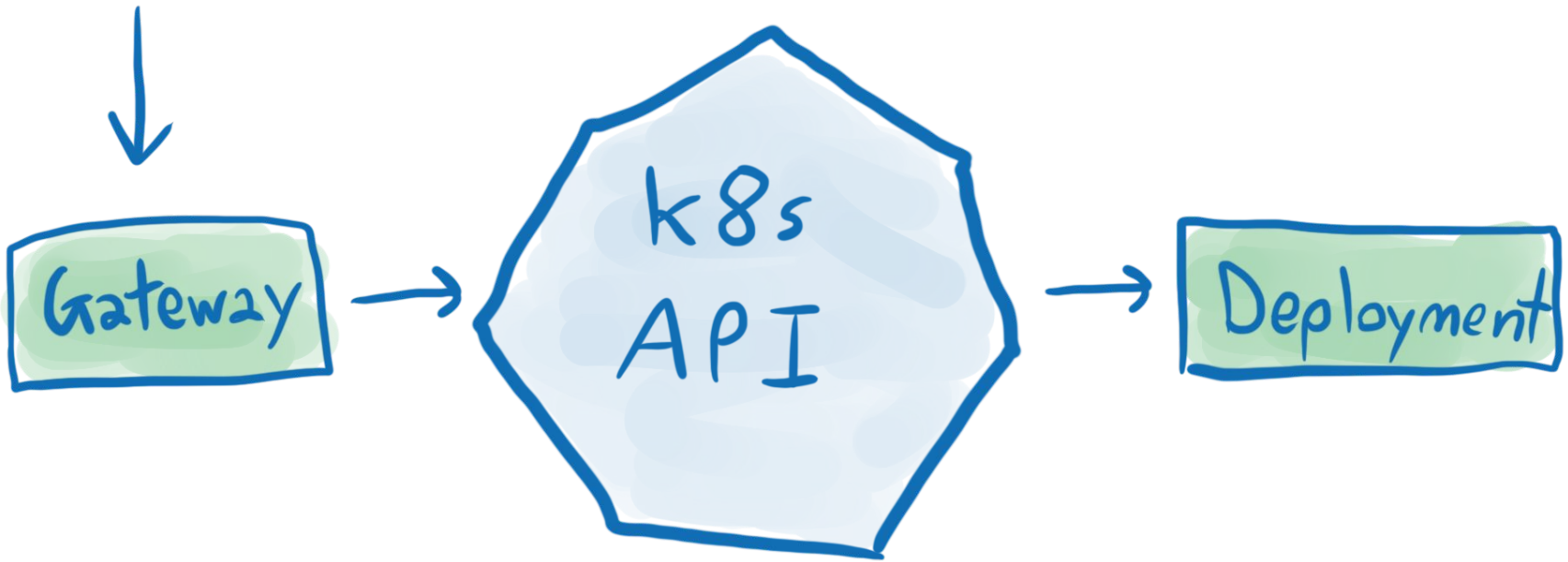

https://xkcd.com/2044/


Custom API Gateways

# What Is An API Gateway?



- A deputy/gateway is designed to perform specific actions, which require elevated permissions.
- The deputy exposes an API to trigger these actions.
- Acts as a *logical gate* to the underlying system.


Gateway
k8s
API
Deployment

# API Deputy Drawbacks



- You still have a service with elevated permissions.
- Many actions require a very *simple* logical gate.
  - EG “only allow updates to this field”.

# Admission Webhooks

# Admission Webhooks



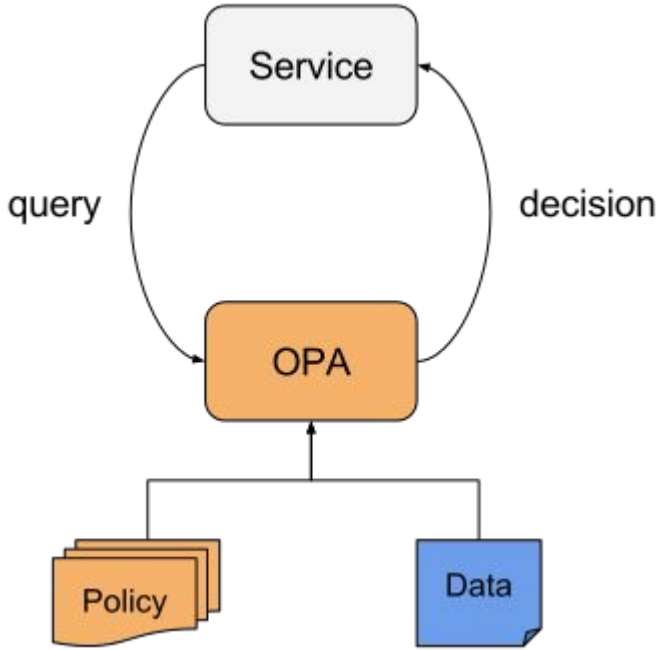
- Validating & Mutating WebHooks
- Custom hooks can be written to ensure resources not meeting cluster criteria are not created - Validating
- Change objects missing certain requirements - Adding labels/annotations, resource limits, etc - Mutating
- Hooks can be used to restrict the creation of objects in a Namespace based on requirements


Open Policy Agent (OPA)

# Open Policy Agent



- OPA offloads Policy Decisions from a service (across the stack)
- Queries - can an action be taken?
  - answer is provided back to the service with an allow or deny.
- Allows context specific decisions
- Can understand the status or specs of other services



# Using OPA for Resource Access Control



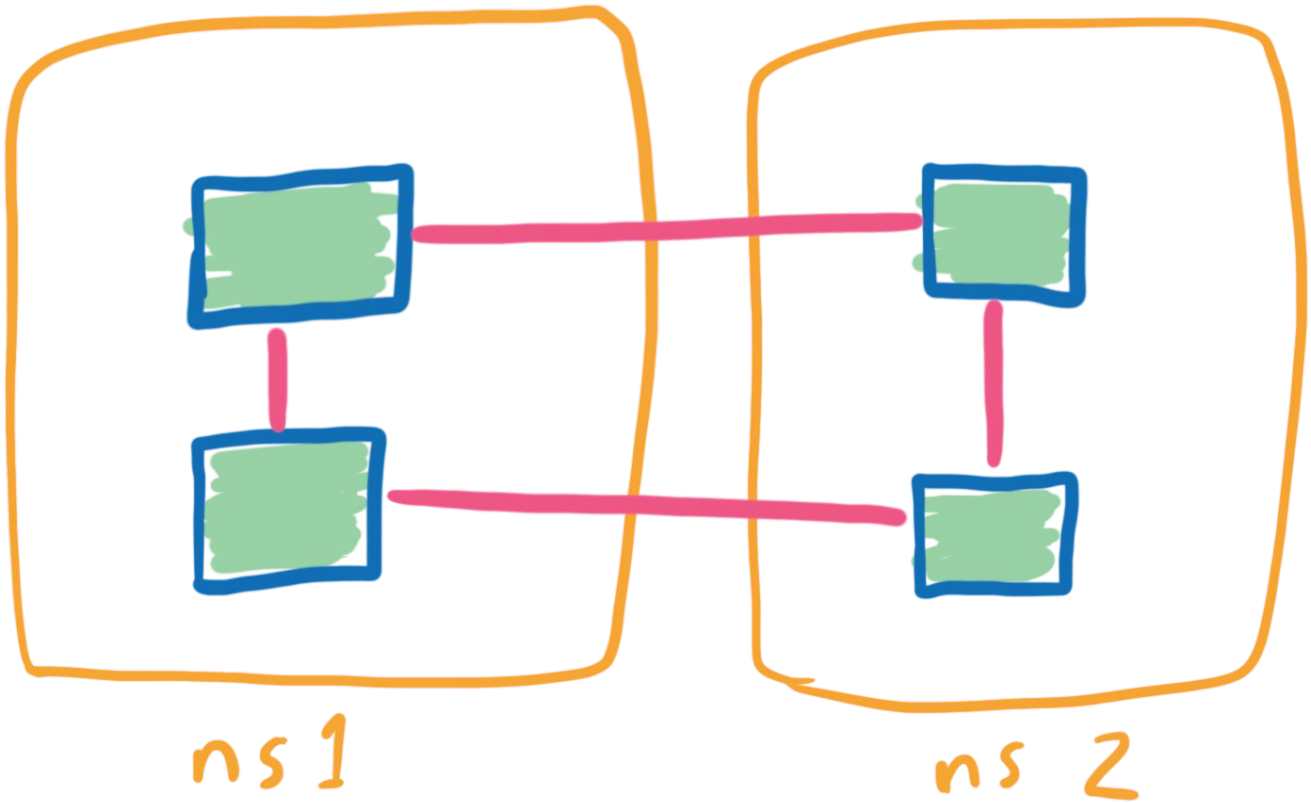
- API calls are sent to OPA with the JSON Object
- OPA compares to its data, and returns an `Allow` or `Deny`
- Universal language makes it easier to report *why* an action was denied
- Denied requests could return a patch that should be applied to the object
  - Could be used to enable a Mutating AdmissionWebhook, and apply a patch directly

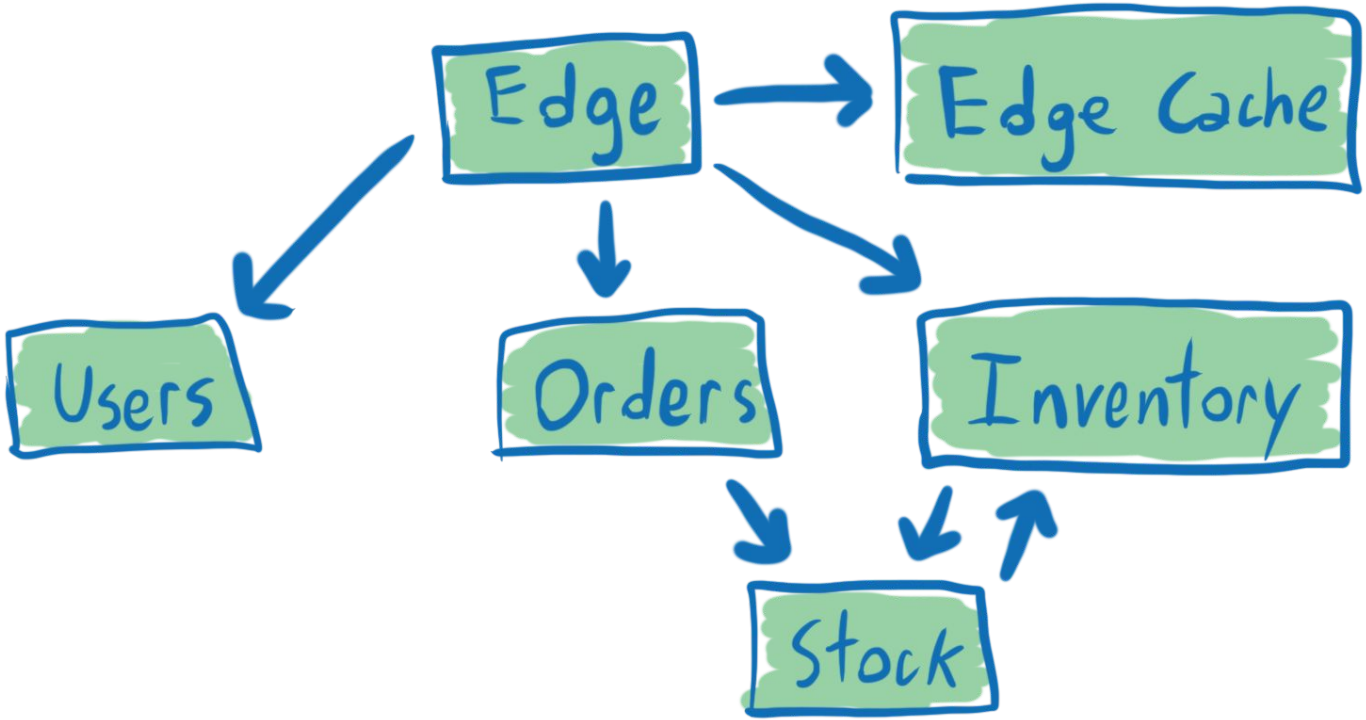
# Using OPA for NetworkPolicy

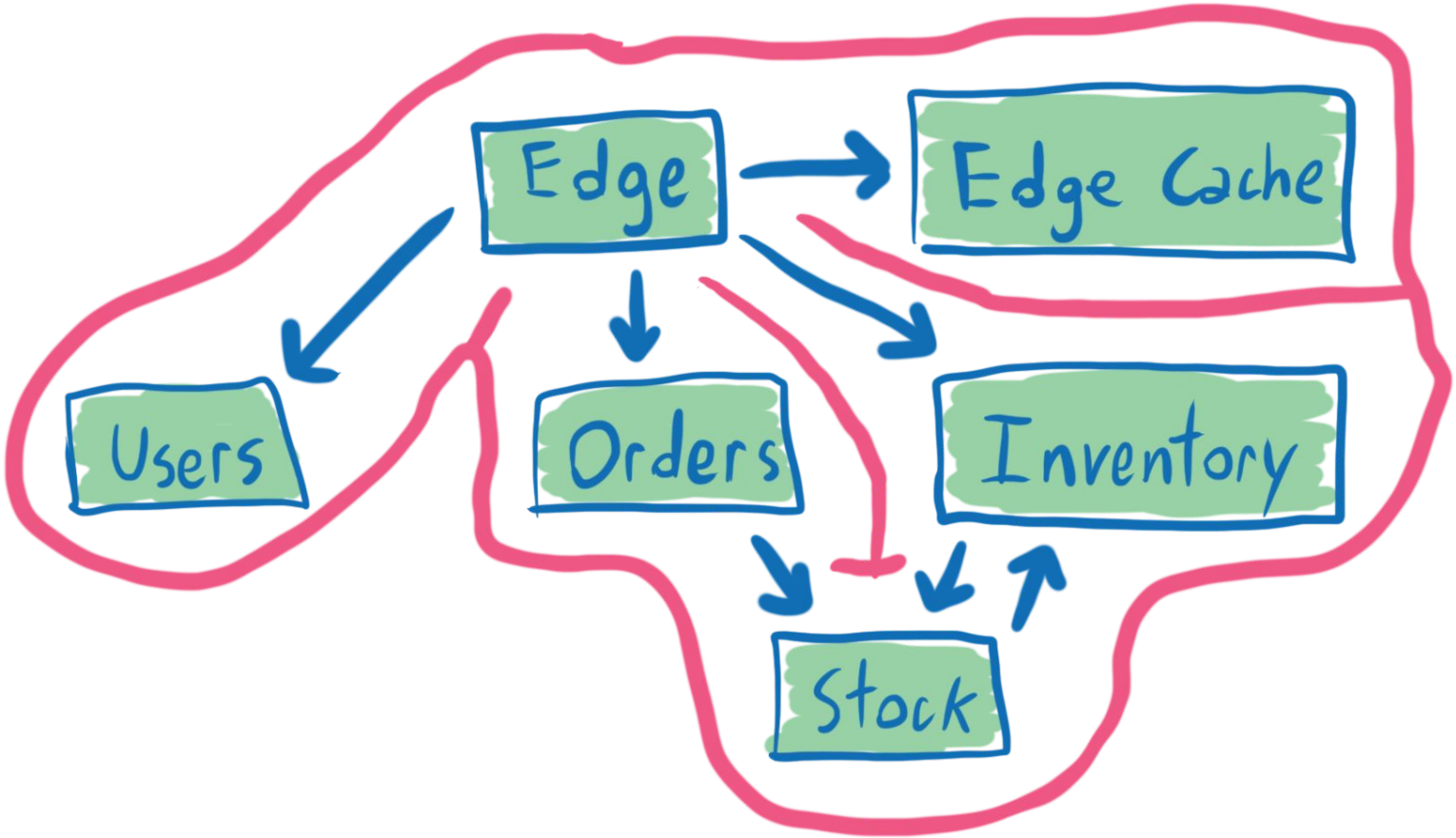


- OPA could enhance NetworkPolicy
- Layer 7 restrictions (`post` on `/widgets` but not `get` on `/widgets`)
- Labels/Annotations could be used to manage intra-Namespace communication between objects
- Powerful combo of OPA for NetworkPolicy & AdmissionControllers


Network Access Policy


ns 1
ns 2


Edge
Edge Cache
Users
Orders
Inventory
Stock


Edge
Edge Cache
Users
Orders
Inventory
Stock

# Kubernetes NetworkPolicy



- Kubernetes has a native NetworkPolicy
  - Selectors pods with a selector label, those pods can receive no traffic unless allowed by the policy.
- Requires a 3rd-party controller - policies do nothing without one

Examples: https://github.com/ahmetb/kubernetes-network-policy-recipes

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-nginx
  namespace: demo
spec:
  podSelector:
    matchLabels:
      $KEY: $VALUE
  ingress:
    - from:
      - podSelector:
          matchLabels:
       $KEY: $VALUE
```

# Implementing Network Policies



- Still an evolving addon space in Kubernetes.
  - Cilium policies
  - Istio policies
  - Calico
- Some service mesh-type tools offer similar functionality
  - Istio
  - Cilium
  - etc


Time For The Opinions!


Sub-namespace
Permissions?

```
SpaceshipGrey:~ vallery$ ls -l
total 32
drwx------@   5 vallery  staff    ... Applications
drwx------+  17 vallery  staff    ... Desktop
```

```
apiVersion: ...
kind: ...
metadata:
    ...
spec:
    ...
status:
    ...
```

```
apiVersion: ...
kind: ...
metadata:
    owningRoles:
        - jenkins
        - ops
    ...
spec:
    ...
status:
    ...
```


Smaller Namespaces?

# What Are Our Limiting Factors?



- Objects that rely on one another need to be in the same namespace.
  - EG Ingress / Service / Deployment / Pods
- (Namespace level) service accounts can’t be used in multiple namespaces.
  - Users / bots will need more accounts.

# Cookbook For Success



- Keep ~1 distinct (lowercase s) service per namespace.
- Restrict pod traffic between namespaces with a NetworkPolicy, to only allow required cross-namespace traffic.
- Use multiple controllers/SAs to operate on objects spread between namespaces.
  - Use a cluster-level controller/SA if the scope/access is appropriate.


Too Long, Didn’t Listen

0. Kubernetes' design priority is extensibility, not out-of-the-box usability. This includes security considerations.

1. Put individual sets of resources, like a service/project in their own namespaces.

2. Use RBAC to restrict access to **reasonable** as-needed access. Prefer too-broad over painfully granular.

3. Use policy tools like webhooks or OPA to provide narrow restrictions to resources, beyond RBAC granularity.

# Vallery Lancey

- Infrastructure at Lyft
- @vllry on Twitter and Github

Kubernetes contributor (mostly SIG-network). Dabbles in many aspects of distributed systems and reliability engineering.

# Seth McCombs

- SRE at Triller
- Tweets at @SethMcCombs

More ops than dev, fiddles with containers and Kubernetes. Avid guitarist and collector of fountain pens.