Yelp Security

Fine-Grained User Authorization for Kubernetes with OPA And LDAP

Charlie Cetin - ccetin@yelp.com
Quentin Long -qlo@yelp.com

September 2021

Speakers

- Charlie Cetin
    - Tech Lead for Identity and Access Management @ Yelp

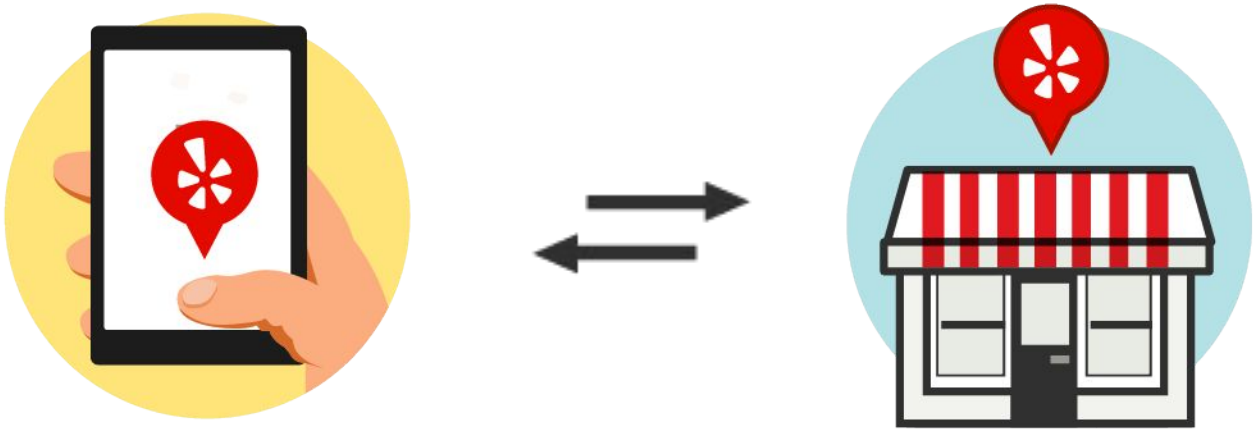
- Quentin Long
    - Tech Lead for Data Security @ Yelp

Outline

➢ **Motivation**

- Authentication Architecture
- Authorization Architecture
- End to End Example
- Rollout Strategy and System Reliability
- Conclusions

Connecting people with great local businesses.



Yelp Activity

- **224M Cumulative Reviews** (as of Dec. 31, 2020)

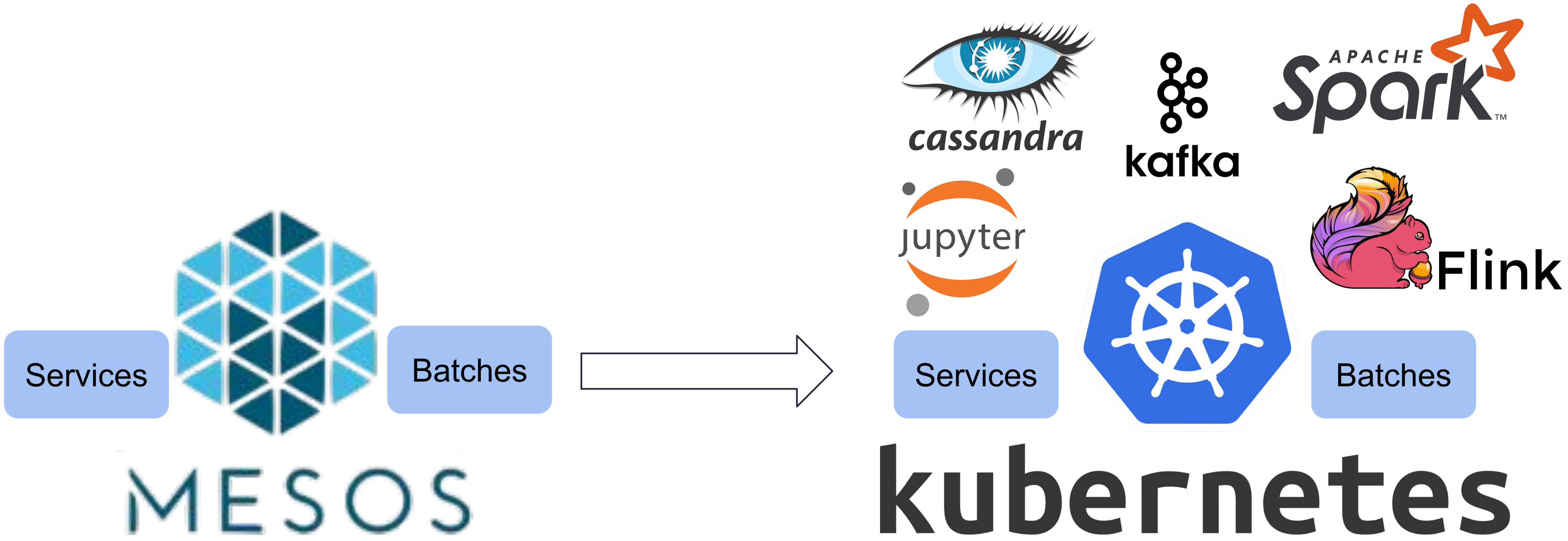
- **31M App Unique Devices** (monthly average for 2020)

- **528K Paying Advertising Locations** (monthly average for Q2 2021)

Yelp Kubernetes Infrastructure

- **1000+** software engineers
- **100+** teams
- **600+** microservices
- **3000+** K8s resources
- **20+** K8s Clusters on ec2

Mesos Migration to Kubernetes



Motivation: Initial K8s access-controls

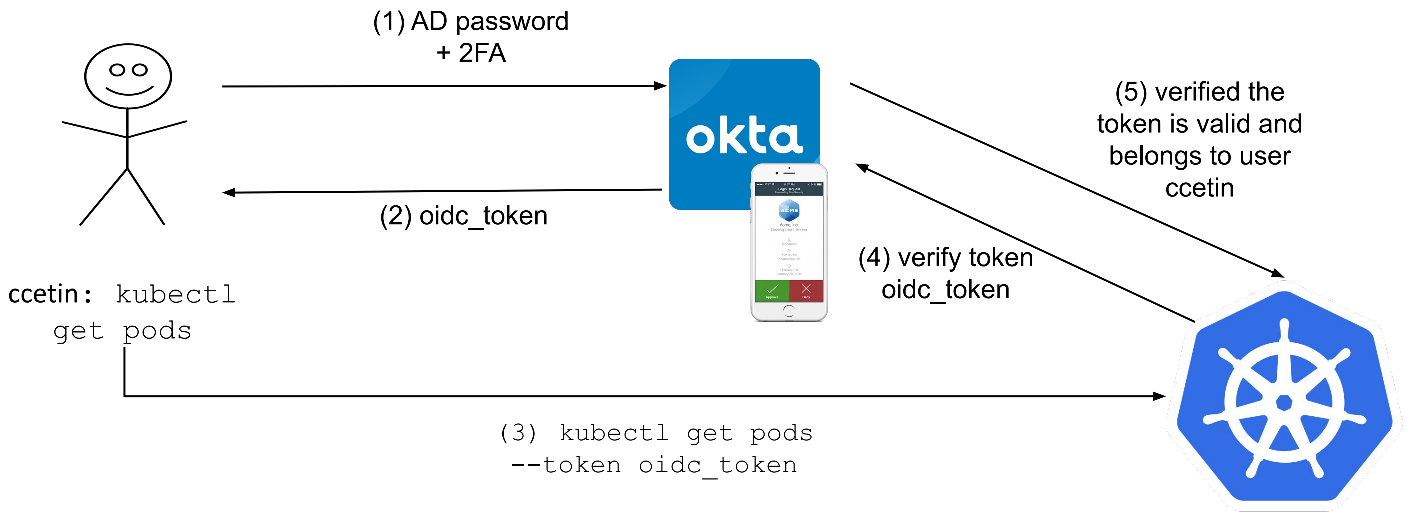
- **Administrative** access to K8s clusters for all users
- **sudo** required to run any kubectl command
- Disadvantages
    - Accountability
    - Reliability
    - Compliance

Kubernetes AuthNZ Requirements

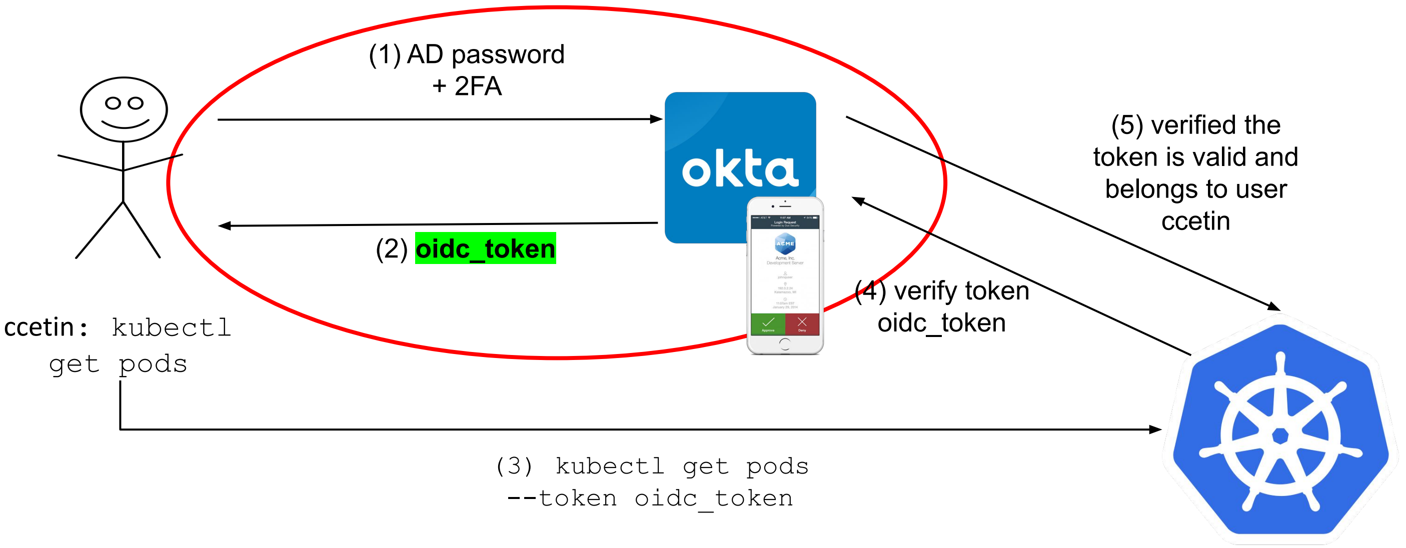
- Users must be authenticated individually
- Enforce **least-privilege** authorization rules
  - Service ownership
  - Resource sensitivity levels
  - Action sensitivity
  - Arbitrary labels and annotations
- Formal paper trail for access control changes
- Support for human users

Outline

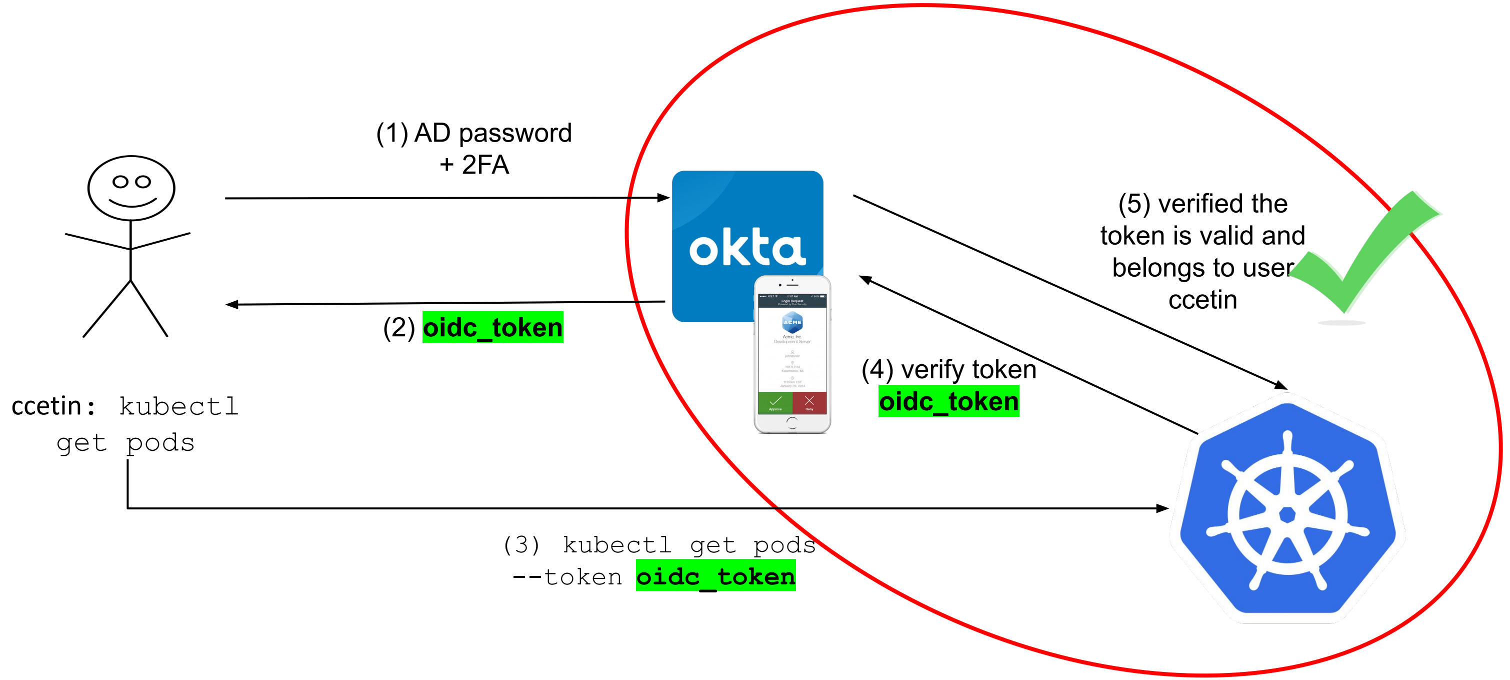
Authenticating Users with Okta



Authenticating Users with Okta



Authenticating Users with Okta


(1) AD password + 2FA
(2) oidc_token
ccetin: kubectl get pods
(3) kubectl get pods --token oidc_token
(4) verify token oidc_token
(5) verified the token is valid and belongs to user ccetin
okta

Authenticating Users with Okta: Benefits

- sudo no longer needed to interact with Kubernetes
- Each action attributed to a specific user
- Temporary credentials

Outline

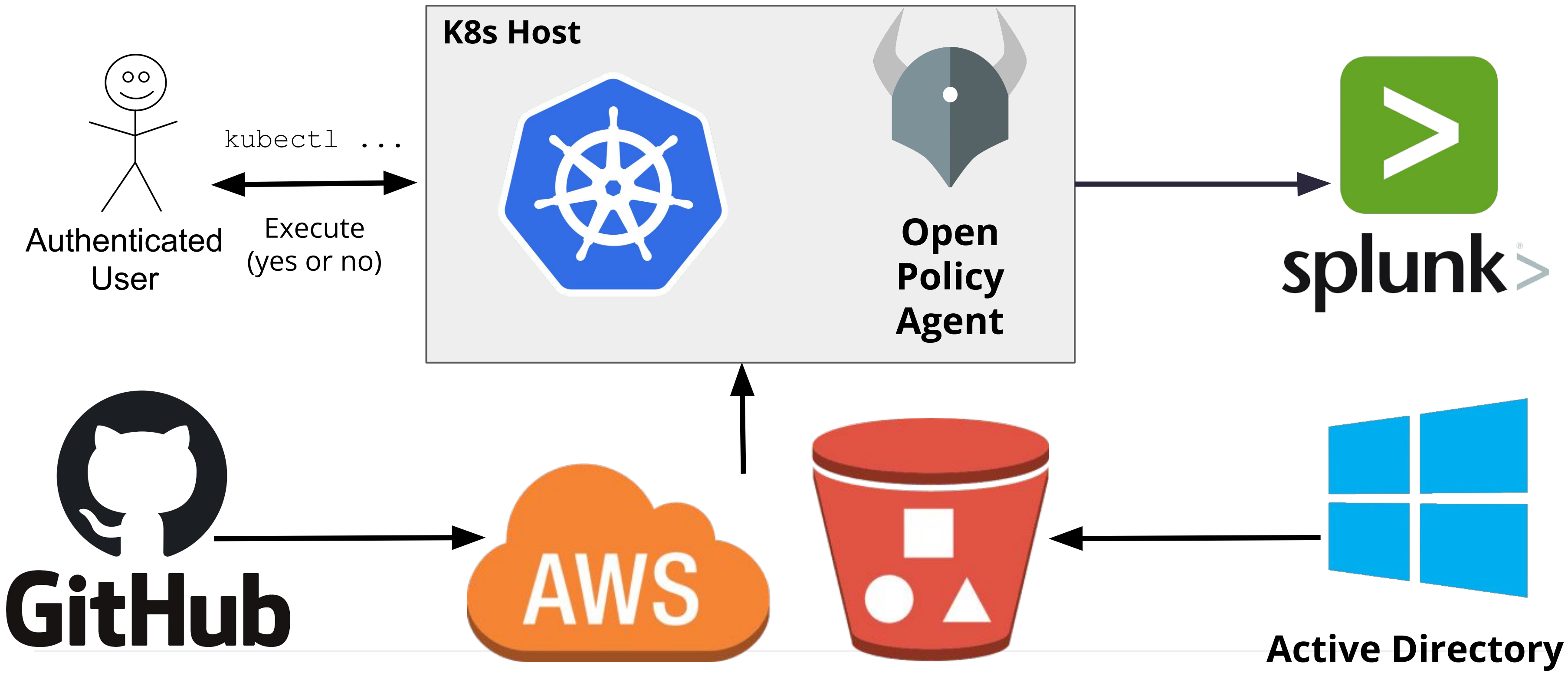
Kubernetes Authorization Options

- ~~RBAC~~
  - ~~Single namespace hosting hundreds of teams workloads~~
  - ~~Yet Another Yaml File for permissions and group memberships~~

- Authorization Webhook
  - Delegate authorization decisions to external service
  - Open Policy Agent
  - Active directory for source of truth

Outline

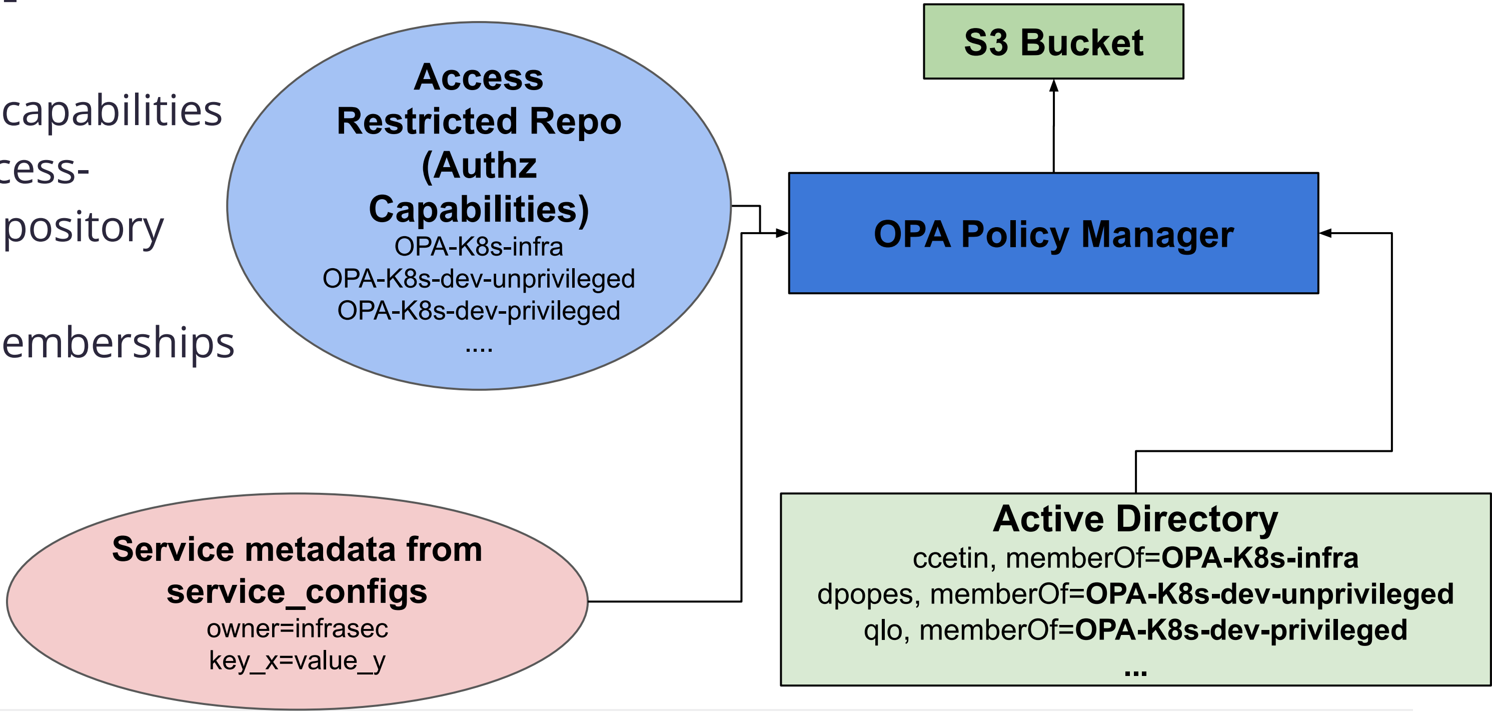
Authorization Architecture Overview



Authorization Component: OPA Capabilities, User Groups, Service Metadata

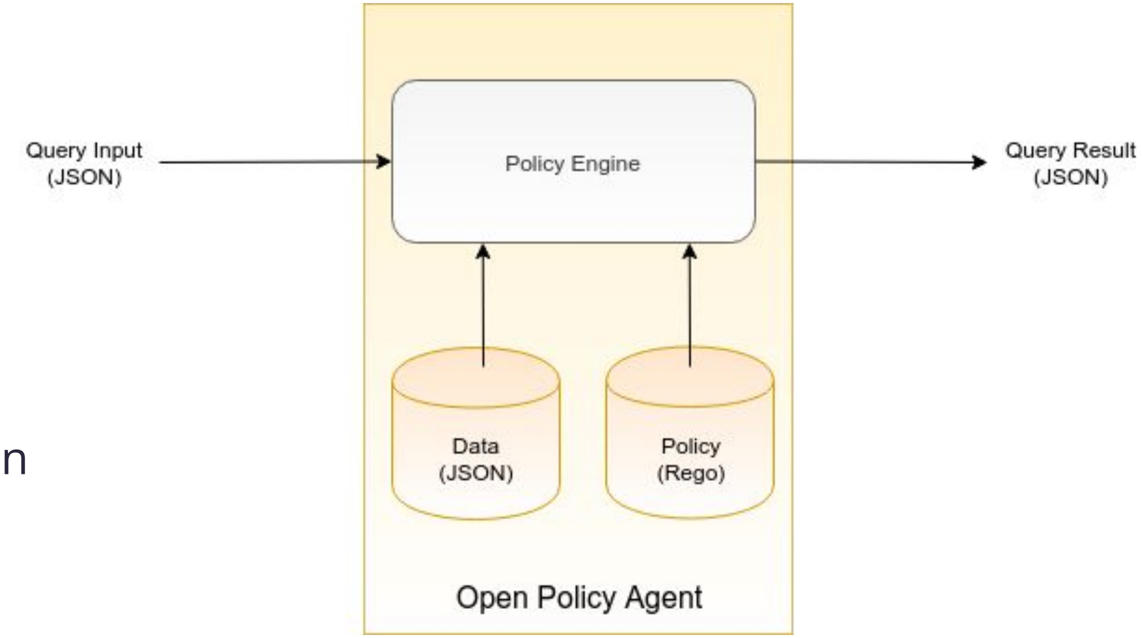
- Access Control capabilities stored in an access-restricted git repository

- Users' group memberships stored in AD


S3 Bucket
Access Restricted Repo (Authz Capabilities)
OPA-K8s-infra
OPA-K8s-dev-unprivileged
OPA-K8s-dev-privileged
....
OPA Policy Manager
Service metadata from service_configs
owner=infrasec
key_x=value_y
Active Directory
ccetin, memberOf=OPA-K8s-infra
dpopes, memberOf=OPA-K8s-dev-unprivileged
qlo, memberOf=OPA-K8s-dev-privileged
...

Authorization Component: OPA Policies

- OPA policy: Written in Rego
- Our policy uses:
  - Service metadata
  - Capabilities
  - User Groups
  - K8s webhook input
- Policy enforces authorization



Authorization Component: OPA Capabilities

- Each capability defines permissions based on
  - Clusters
  - Namespaces
  - Resources
  - Subresources
  - Resource names
  - Verbs
  - Pod metadata
  - Service metadata
- Each capability can have sub capabilities
- Empty list? **Allow all**

```
OPA-K8s-admin:
    admin:
        clusters:[]
        namespaces: []
        verbs: []
        resources: []
        subresources: []
        resourcenames: []
        pod_metadata: {}
        service_metadata: {}
```

Capability Example: Dev-Unprivileged

```
OPA-K8s-dev-unprivileged:
    dev_admin:
        clusters:
        - playground
        - test
        namespaces: []
        verbs: []
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata: {}
    read_only:
        clusters: []
        namespaces: []
        verbs:
        - list
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata: {}
```

Capability Example: Dev-Unprivileged

Members of **OPA-K8s-dev-unprivileged**:

- Any commands in the test clusters

```
OPA-K8s-dev-unprivileged:
  dev_admin:
      clusters:
      - playground
      - test
      namespaces: []
      verbs: []
      resources: []
      resourcenames: []
      subresources: []
      pod_metadata: {}
      service_metadata: {}
  read_only:
      clusters: []
      namespaces: []
      verbs:
      - list
      resources: []
      resourcenames: []
      subresources: []
      pod_metadata: {}
      service_metadata: {}
```

Capability Example: Dev-Unprivileged

Members of **OPA-K8s-dev-unprivileged**:

- Any commands in the test clusters
- Also, **list** commands anywhere, such as
  - `kubectl get pods`
  - `kubectl get namespaces`

```
OPA-K8s-dev-unprivileged:
    dev_admin:
        clusters:
        - playground
        - test
        namespaces: []
        verbs: []
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata: {}
    read_only:
        clusters: []
        namespaces: []
        verbs:
        - list
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata: {}
```

Capability Example: service-access

Members of **OPA-K8s-security-sensitive**:

- Any command if its resource interacts with **infrasec_service1** or **infrasec_service2**
  - `kubectl get pods -n infrasec infrasec-service-pod`

```
OPA-K8s-security-sensitive:
    sec_sensitive:
        clusters: []
        namespaces: []
        verbs: []
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata:
            yelp.com/service_name:
             - infrasec_service_1
             - infrasec_service_2
        service_metadata: {}
```

Capability Example: team-based-access

Members of **OPA-K8s-infrasec-team**:

- Any command if resource owned by **infrasec**

```
OPA-K8s-infrasec-team:
    sec:
        clusters: []
        namespaces: []
        verbs: []
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata:
            team:
            - infrasec
```

Capability Example: team-based-access (cont'd)

Members of **OPA-K8s-my-team-unprivileged**:

- Basic access to services **owned** by a user's team

```
OPA-K8s-my-team-unprivileged:
    basic:
        clusters: []
        namespaces: []
        verbs:
        - get
        - list
        - watch
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata:
            team:
            - '#myteam'
```

Capability Example: team-based-access (cont'd)

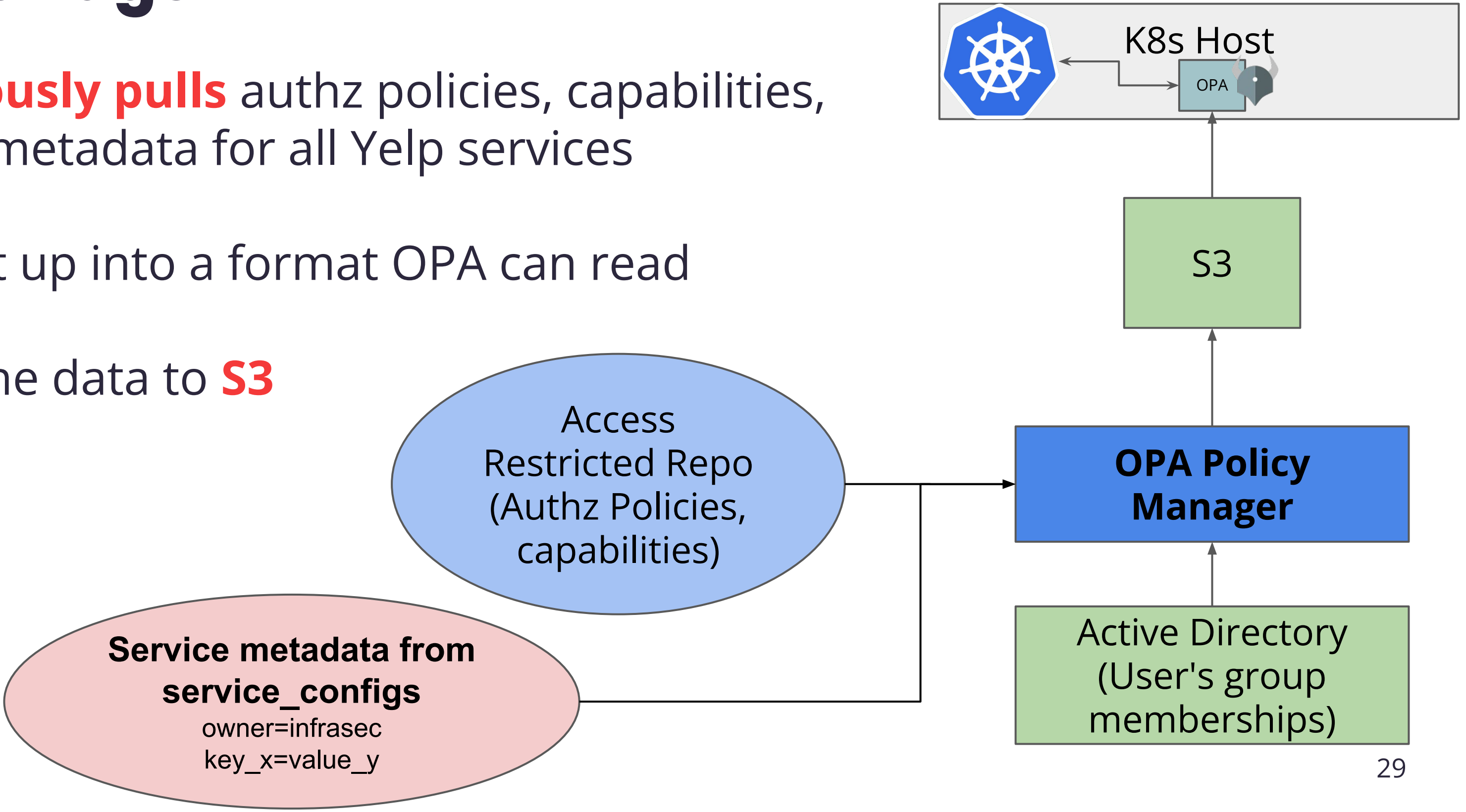
Members of **OPA-K8s-my-team-privileged** can:

- Administrative access to services **owned** by a user's team

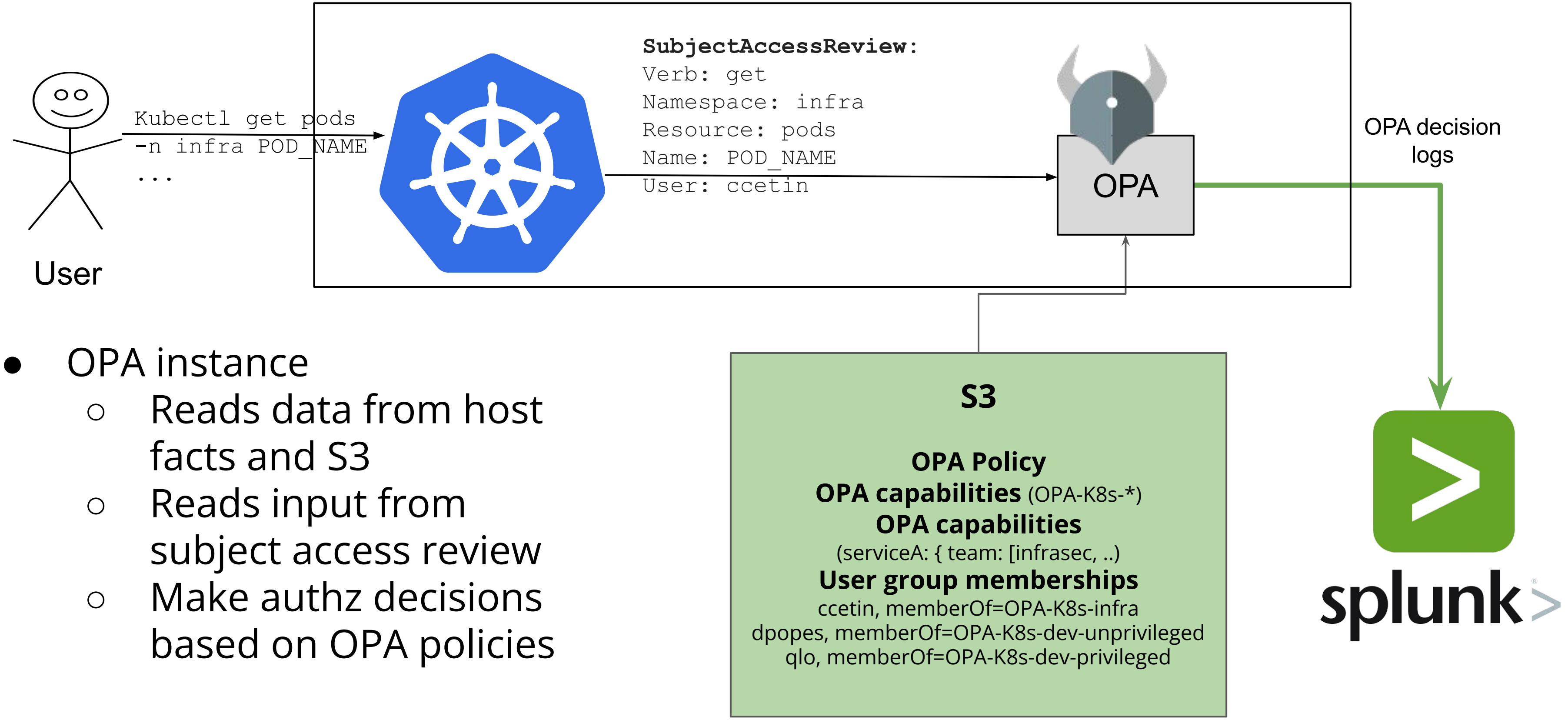
```
OPA-K8s-security-team-privileged:
    team-admin:
        clusters: []
        namespaces: []
        verbs: []
        resources: []
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata:
            team:
            - '#myteam'
```

Authorization Component: The Policy Manager

- **Continuously pulls** authz policies, capabilities, AD data, metadata for all Yelp services
- Bundles it up into a format OPA can read
- **Pushes** the data to **S3**


K8s Host
OPA
S3
Access Restricted Repo (Authz Policies, capabilities)
OPA Policy Manager
Service metadata from service_configs
owner=infrasec
key_x=value_y
Active Directory (User's group memberships)

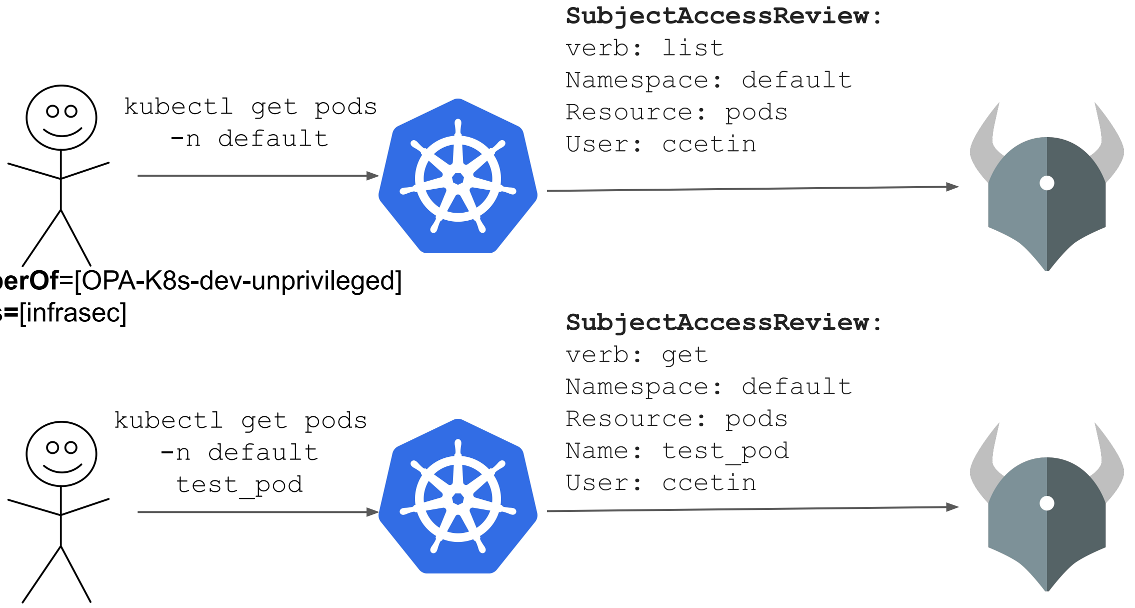
Authorization Component: Client side enforcement



- OPA instance
  - Reads data from host facts and S3
  - Reads input from subject access review
  - Make authz decisions based on OPA policies

Outline

Example run: Basic



```
kubectl get pods
   -n default
```

```
SubjectAccessReview:
verb: list
Namespace: default
Resource: pods
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-dev-unprivileged]
teams=[infrasec]

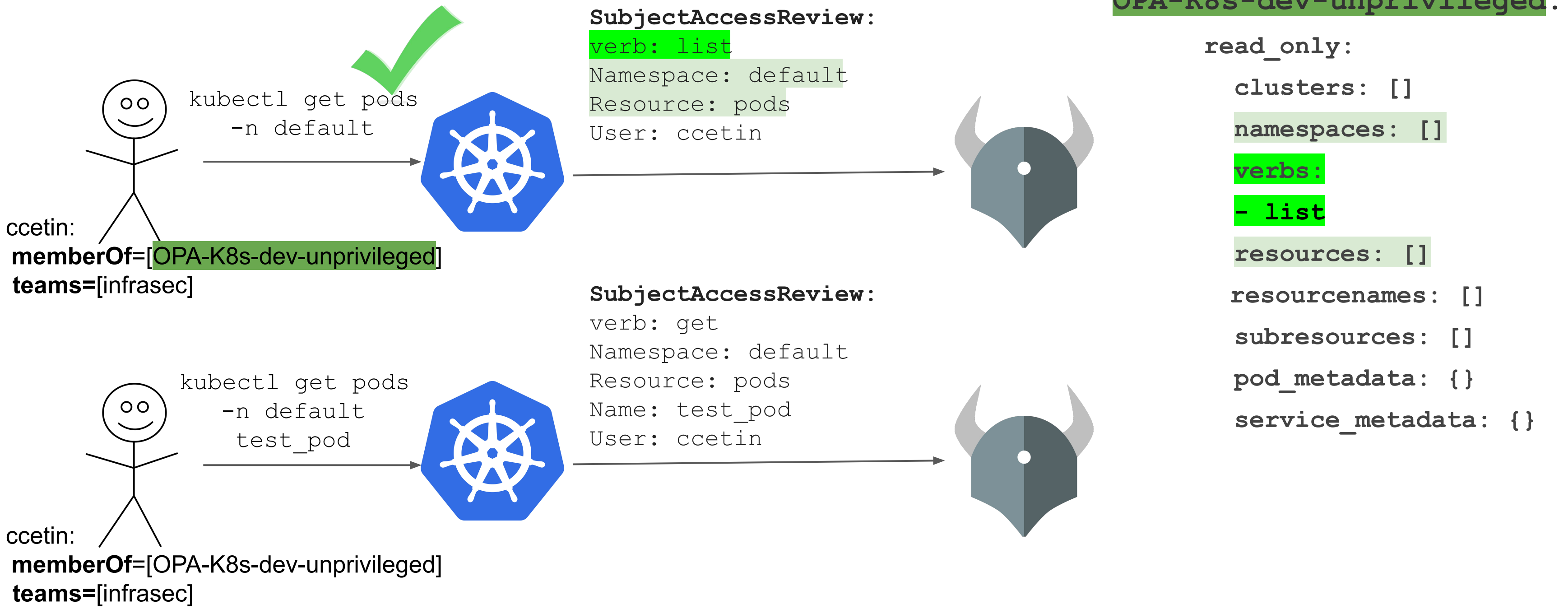
```
kubectl get pods
   -n default
    test_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: test_pod
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-dev-unprivileged]
teams=[infrasec]

```
OPA-K8s-dev-unprivileged:
    read_only:
      clusters: []
      namespaces: []
      verbs:
      - list
      resources: []
      resourcenames: []
      subresources: []
      pod_metadata: {}
      service_metadata: {}
```

Example run: Basic



ccetin:
memberOf=[OPA-K8s-dev-unprivileged]
teams=[infrasec]

```
kubectl get pods
   -n default
```

```
SubjectAccessReview:
verb: list
Namespace: default
Resource: pods
User: ccetin
```

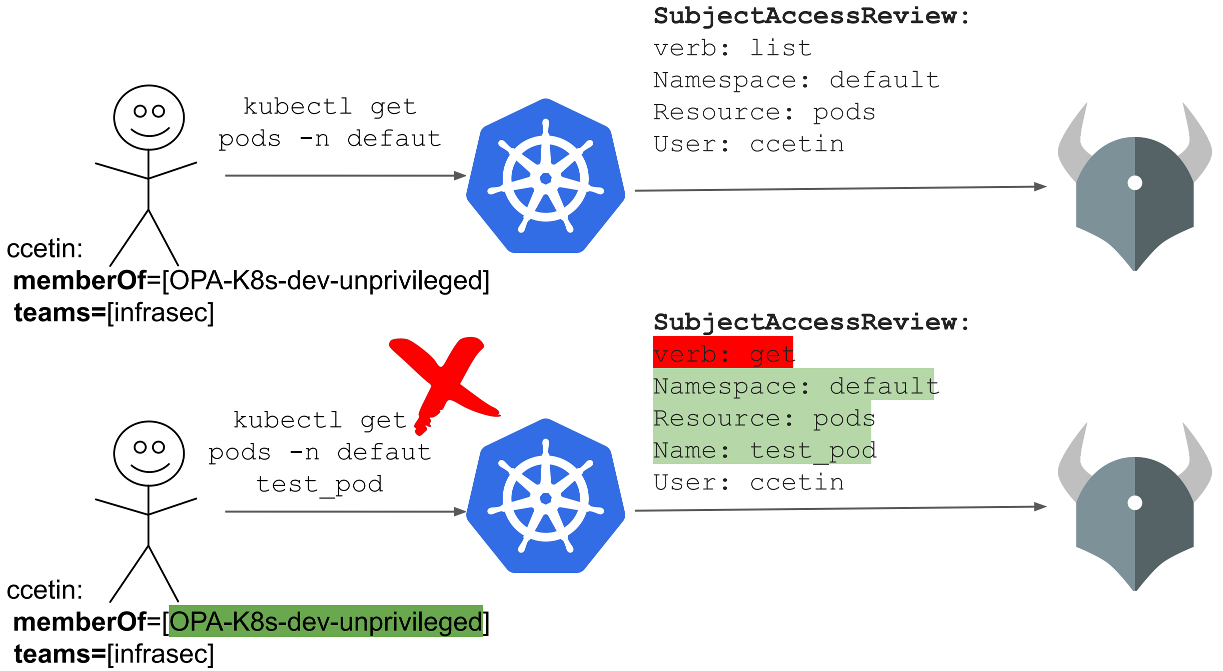
ccetin:
memberOf=[OPA-K8s-dev-unprivileged]
teams=[infrasec]

```
kubectl get pods
   -n default
    test_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: test_pod
User: ccetin
```

```
OPA-K8s-dev-unprivileged:
     read_only:
       clusters: []
       namespaces: []
       verbs:
       - list
       resources: []
       resourcenames: []
       subresources: []
       pod_metadata: {}
       service_metadata: {}
```

Example run: Basic

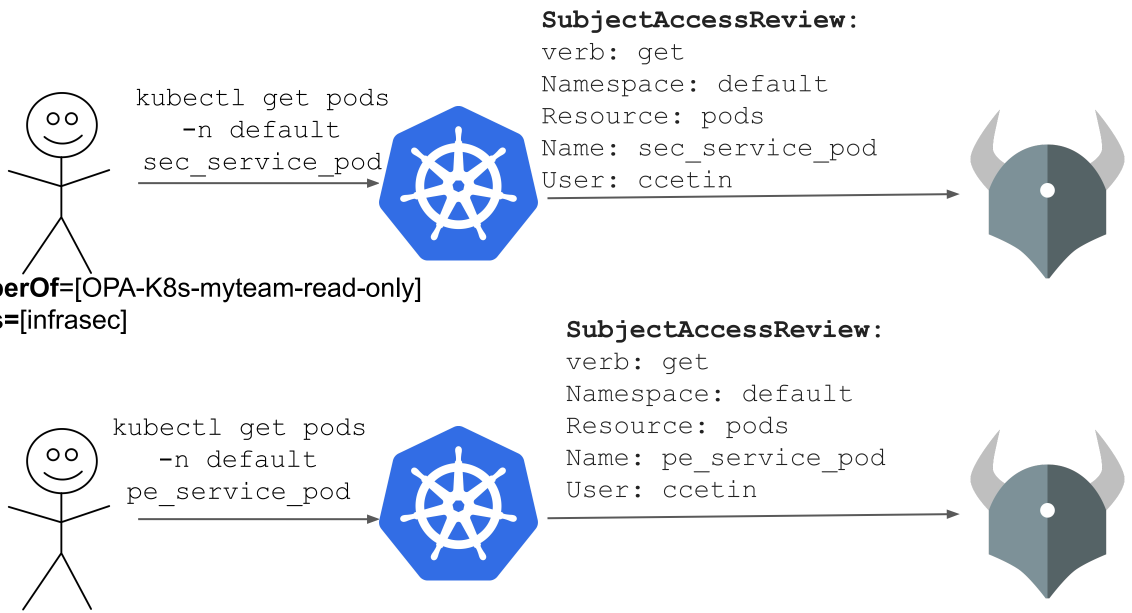

kubectl get pods -n defaut
ccetin:
memberOf=[OPA-K8s-dev-unprivileged]
teams=[infrasec]
kubectl get pods -n defaut test_pod
ccetin:
memberOf=[OPA-K8s-dev-unprivileged]
teams=[infrasec]

```
SubjectAccessReview:
verb: list
Namespace: default
Resource: pods
User: ccetin
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: test_pod
User: ccetin
```

```
OPA-K8s-dev-unprivileged:
    read_only:
      clusters: []
      namespaces: []
      verbs:
      - list
      resources: []
      resourcenames: []
      subresources: []
      pod_metadata: {}
      service_metadata: {}
```

Example run: team-based



```
kubectl get pods
  -n default
 sec_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: sec_service_pod
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

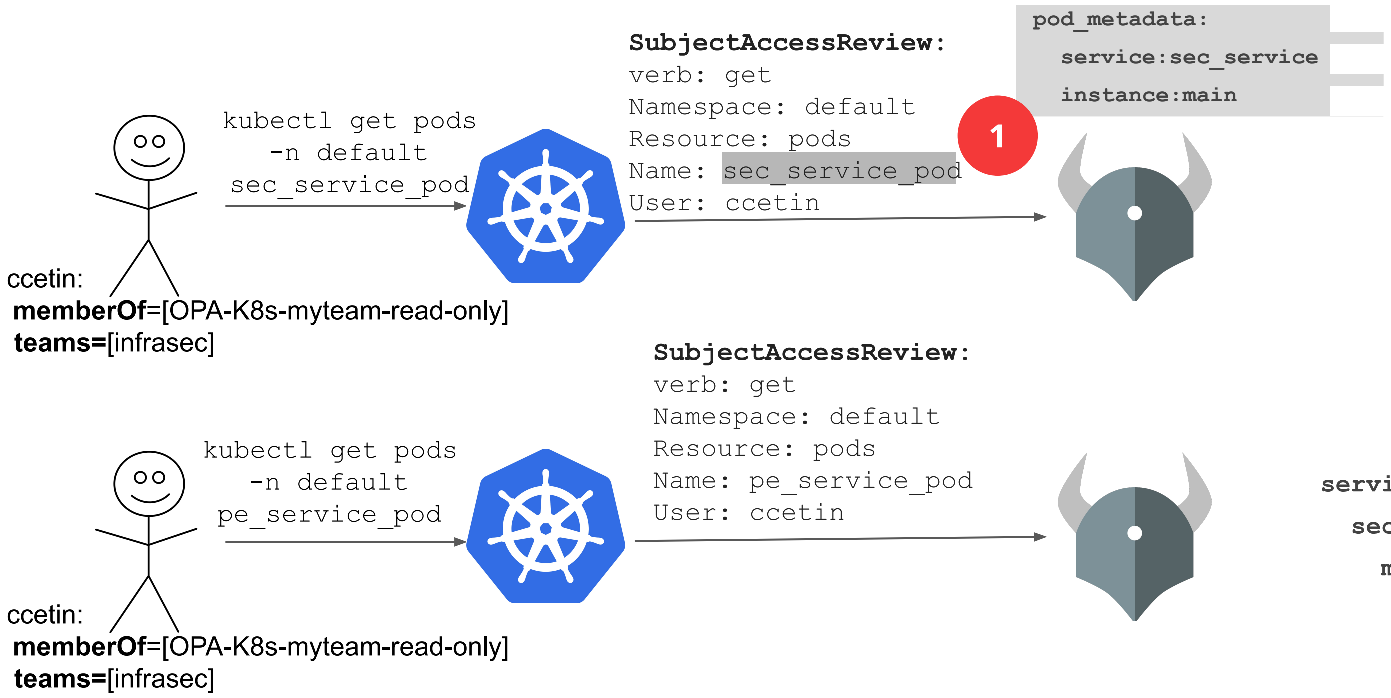
```
kubectl get pods
  -n default
 pe_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: pe_service_pod
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

```
OPA-K8s-myteam-read-only:
    team_read_only:
      clusters: []
      namespaces: []
      verbs:
      - get
      - list
      - watch
      resources:[]
       resourcenames: []
      subresources: []
      pod_metadata: {}
      service_metadata:
           teams:
           - '#myteam'
service_metadata:
  sec_service:
    main:
      owners:
      - infrasec
      - appsec
```

Example run: team-based



```
kubectl get pods
  -n default
 sec_service_pod
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: sec_service_pod
User: ccetin
```

1

```
pod_metadata:
   service:sec_service
   instance:main
```

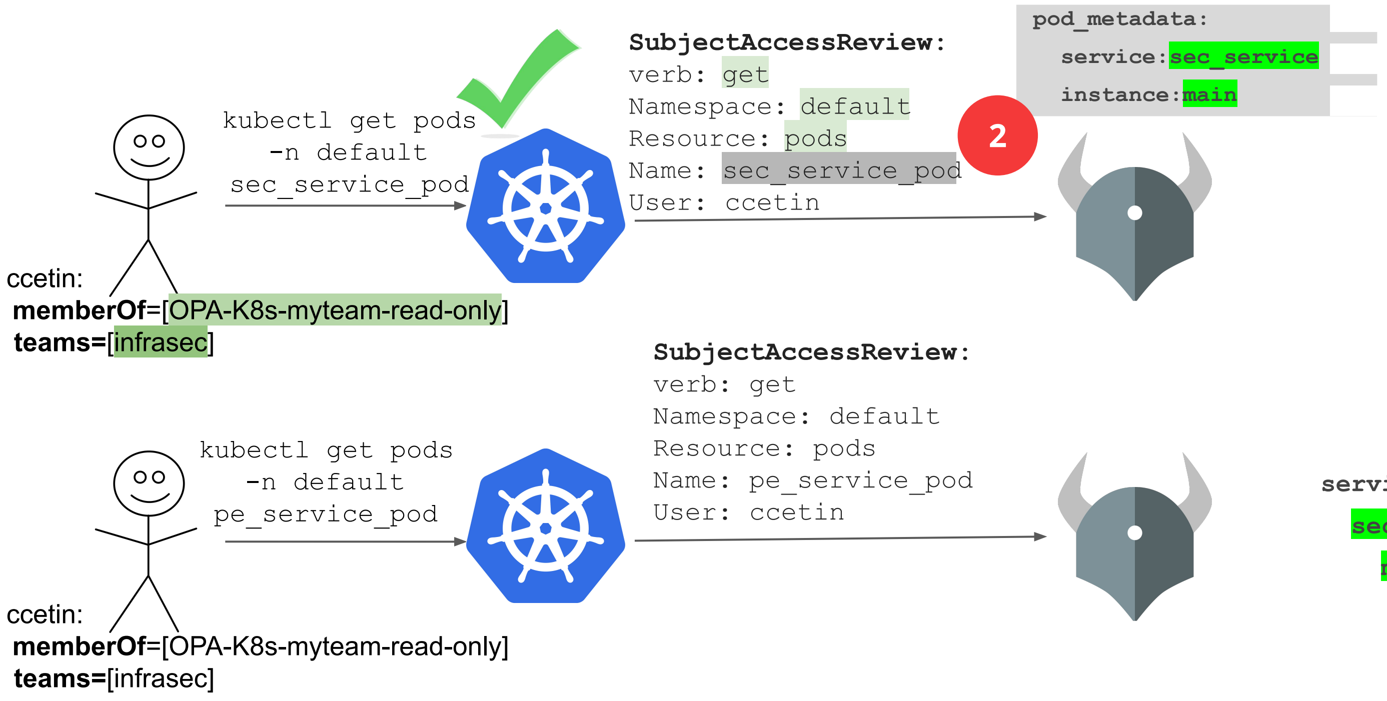
```
kubectl get pods
  -n default
 pe_service_pod
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: pe_service_pod
User: ccetin
```

```
OPA-K8s-myteam-read-only:
      team_read_only:
        clusters: []
        namespaces: []
        verbs:
        - get
        - list
        - watch
        resources:[]
         resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata:
             teams:
             - '#myteam'
service_metadata:
   sec_service:
     main:
       owners:
       - infrasec
       - appsec
```

Example run: team-based



```
kubectl get pods
  -n default
 sec_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: sec_service_pod
User: ccetin
```

```
pod_metadata:
  service:sec_service
  instance:main
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

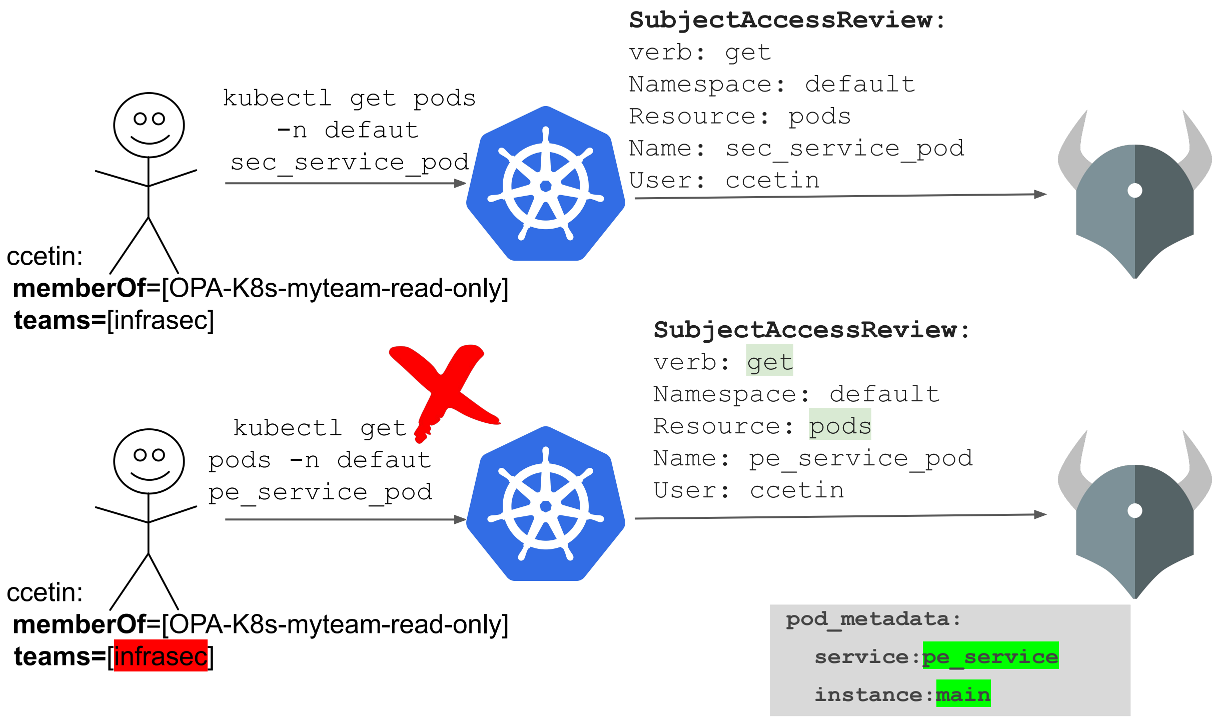
```
kubectl get pods
  -n default
 pe_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: pe_service_pod
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

```
OPA-K8s-myteam-read-only:
  team_read_only:
    clusters: []
    namespaces: []
    verbs:
    - get
    - list
    - watch
    resources:[]
    resourcenames: []
    subresources: []
    pod_metadata: {}
    service_metadata:
       teams:
       - '#myteam'
service_metadata:
  sec_service:
   main:
     owners:
     - infrasec
     - appsec
```

Example run: team-based



ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

```
kubectl get pods
   -n defaut
 sec_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: sec_service_pod
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

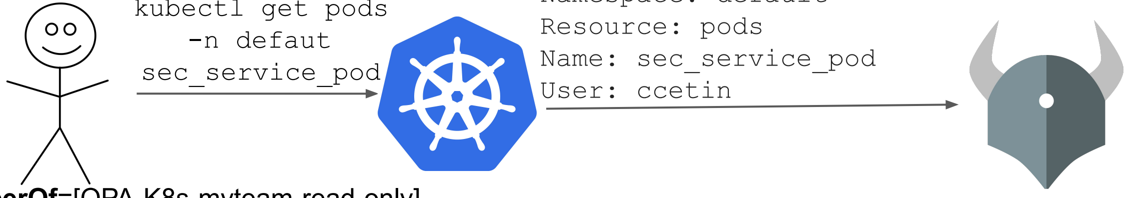
```
 kubectl get
pods -n defaut
pe_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: pe_service_pod
User: ccetin
```

```
pod_metadata:
   service:pe_service
   instance:main
```

```
OPA-K8s-myteam-read-only:
   team_read_only:
      clusters: []
      namespaces: []
      verbs:
      - get
      - list
      - watch
      resources:[]
       resourcenames: []
      subresources: []
      pod_metadata: {}
      service_metadata:
             teams:
             - '#myteam'
service_metadata:
   pe_service:
     main:
        owners:
        - pe
```

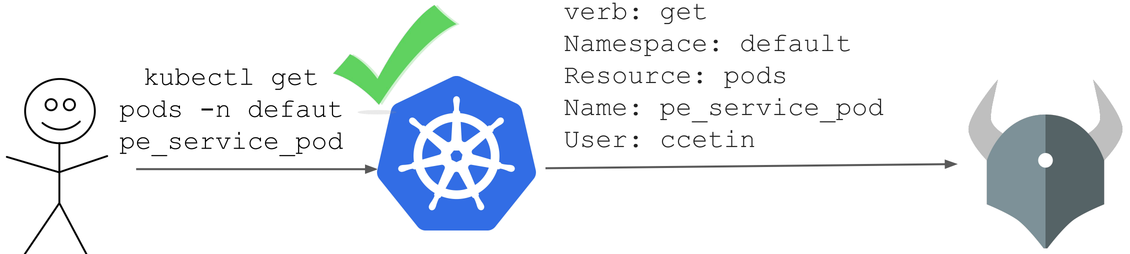
Example run: team-based



```
kubectl get pods
    -n defaut
 sec_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: sec_service_pod
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]



```
 kubectl get
pods -n defaut
pe_service_pod
```

```
SubjectAccessReview:
verb: get
Namespace: default
Resource: pods
Name: pe_service_pod
User: ccetin
```

ccetin:
memberOf=[OPA-K8s-myteam-read-only]
teams=[infrasec]

```
pod_metadata:
  service:pe_service
  instance:main
```

```
OPA-K8s-myteam-read-only:
    team_read_only:
      clusters: []
      namespaces: []
      verbs:
      - get
      - list
      - watch
      resources:[]
      resourcenames: []
      subresources: []
      pod_metadata: {}
      service_metadata:
          teams:
          - '#myteam'
          - pe
service_metadata:
  pe_service:
    main:
      owners:
      - pe
```

Decision Log

- Records all authorization requests
- Shows which groups you have versus which would give access

```
{
    "input": {
        "kind": "SubjectAccessReview",
        "spec": {
            "resourceAttributes": {...}
    },
    "result": {
        "allowed": False,
        "allowed_groups": ["OPA-K8s-Admin"],
        "user_groups": ["OPA-K8s-unprivileged"],
    …
}
```

Outline

- ✓ Motivation
- ✓ Authorization Architecture
- ✓ Authorization Architecture
- ✓ End to End Example
- ➢ **Rollout Strategy and System Reliability**
- ● Conclusions

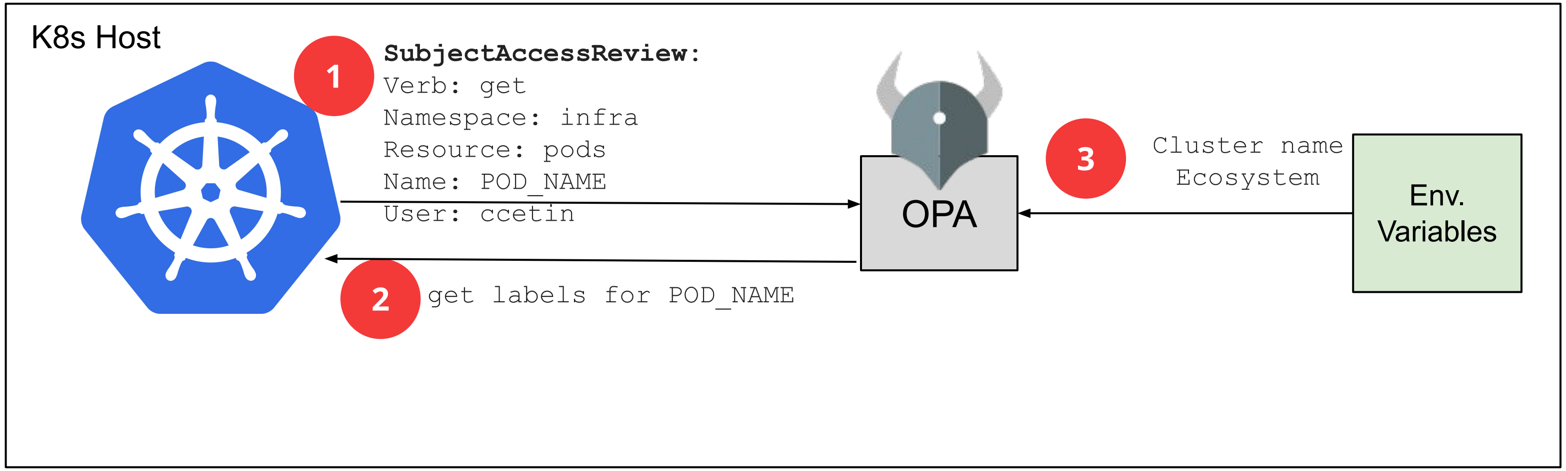
Rollout Strategy

- Ensure that changes are rollback safe
- Configure infra to support a dry-run mode
- Roll out dry-run mode incrementally
- Begin to log usage patterns
- Provision authz capabilities based on usage
- Roll out enforcement mode incrementally
- Over communicate!

Challenges and Special Cases

Problem: K8s authz webhook does not provide needed information:

- Resource labels, service names
- Kubernetes cluster, Yelp ecosystem

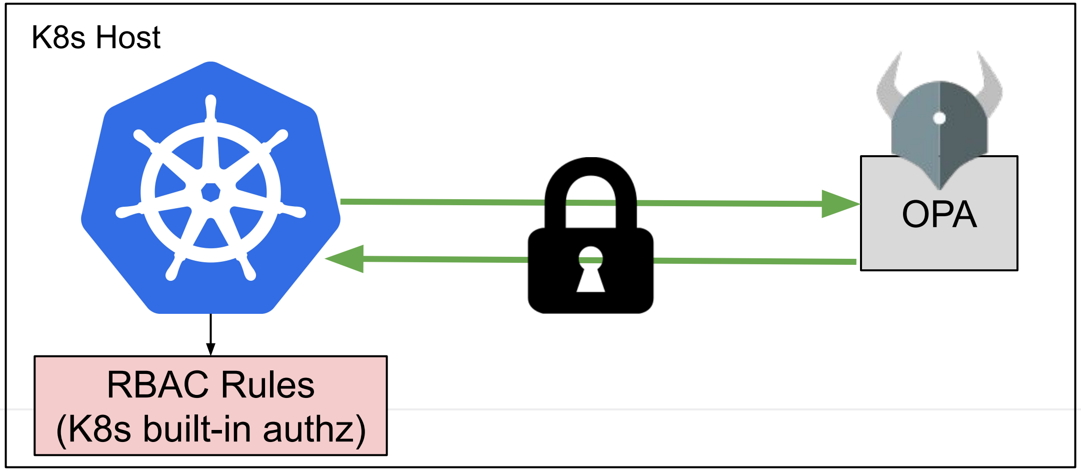


Challenges and Special Cases

Problem:

- Engineers may have network access to OPA instance
- OPA can be configured by API

- **mTLS** between:
  - K8s and OPA
- RBAC policy for OPA to only run **get** command in K8s



Challenges and Special Cases

Problem: Multiple teams own services in a single namespace:

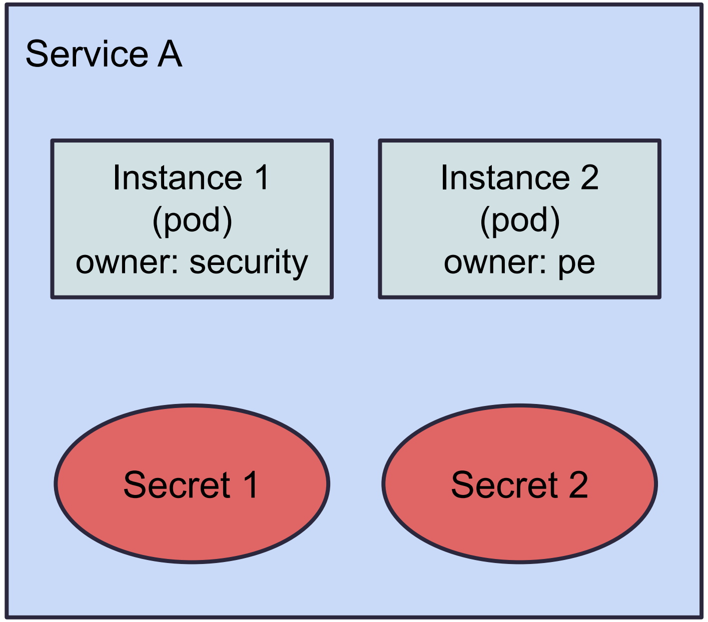
- Difficult to assess all use cases within a single namespace
- Create one **general** team-based policy
    - No bespoke policy for most teams!
- Use a special variable to represent user's team

```
OPA-K8s-myteam-read-only:
      team_read_only:
        clusters: []
        namespaces: []
        verbs:
        - get
        - list
        - watch
        resources:[]
        resourcenames: []
        subresources: []
        pod_metadata: {}
        service_metadata:
             teams:
             - '#myteam'
```

Challenges and Special Cases

Problem: How do we associate the team with non-pod resources with metadata?

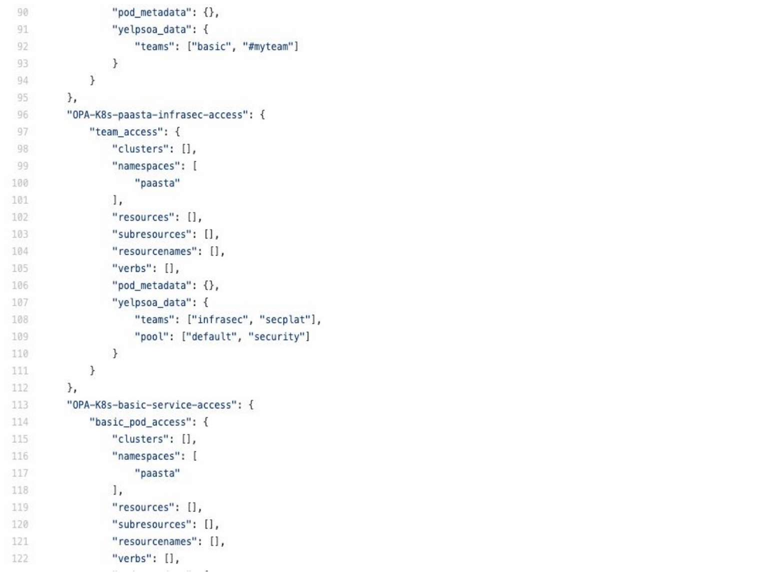
- Normal Case
  - Pod name -> Service and Instance Name -> Owner team
- Special Case, secrets
  - Kubernetes secret -> service name -> **all instances'** owners
- New code path in the rego which knows how to map between secret names and service metadata



Challenges and Special Cases

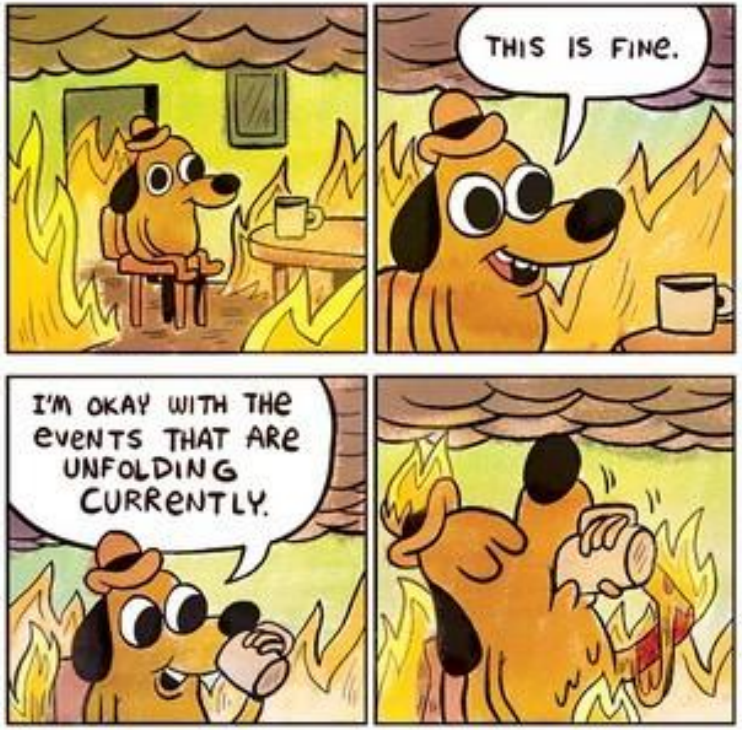
Problem: Rego policy becomes overly complex. Hard to modify or read!

- Come up with extensive test cases



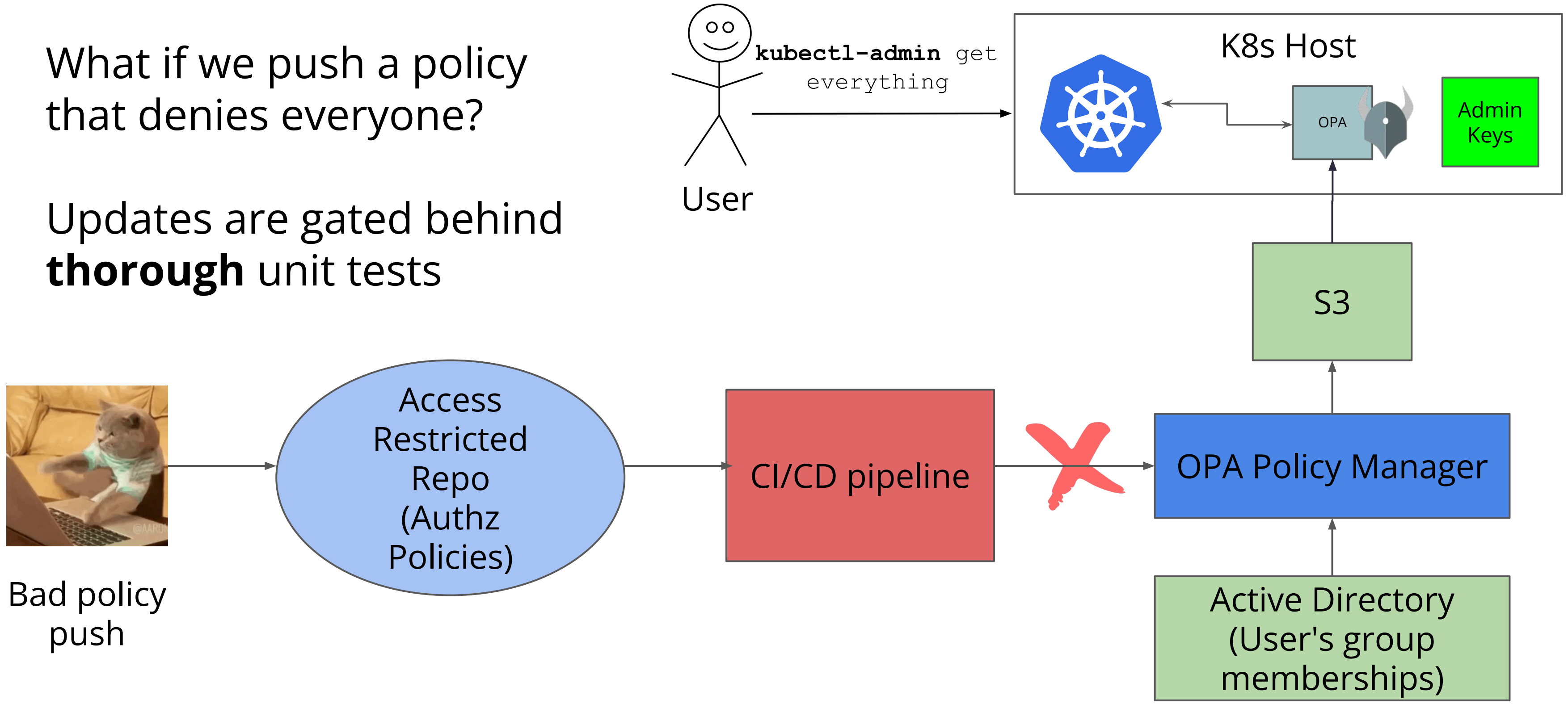
```
                "pod_metadata": {},
                "yelpsoa_data": {
                    "teams": ["basic", "#myteam"]
                }
            }
        },
        "OPA-K8s-paasta-infrasec-access": {
            "team_access": {
                "clusters": [],
                "namespaces": [
                    "paasta"
                ],
                "resources": [],
                "subresources": [],
                "resourcenames": [],
                "verbs": [],
                "pod_metadata": {},
                "yelpsoa_data": {
                    "teams": ["infrasec", "secplat"],
                    "pool": ["default", "security"]
                }
            }
        },
        "OPA-K8s-basic-service-access": {
            "basic_pod_access": {
                "clusters": [],
                "namespaces": [
                    "paasta"
                ],
                "resources": [],
                "subresources": [],
                "resourcenames": [],
                "verbs": [],
```

System Reliability



System Reliability

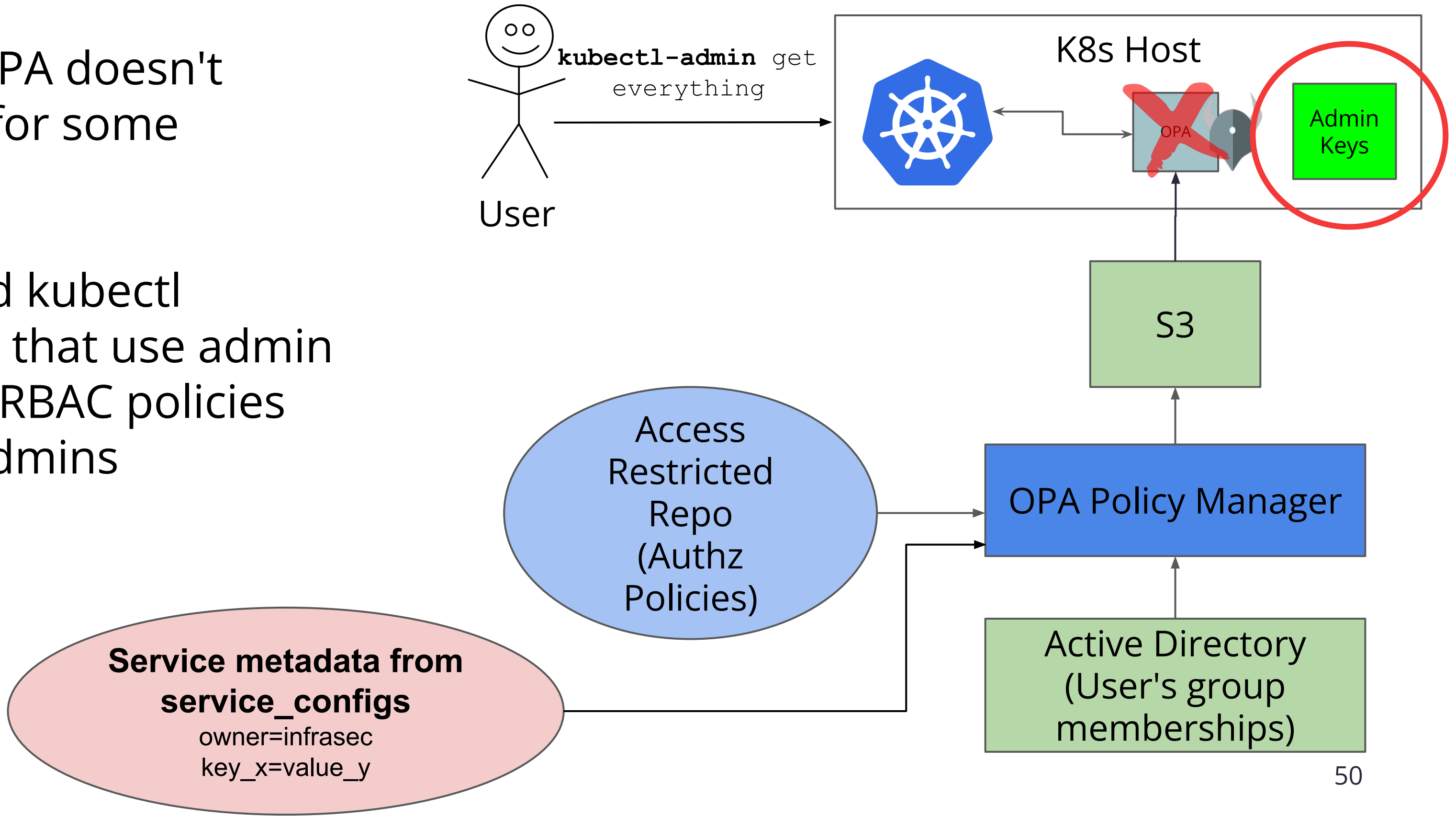
- What if we push a policy that denies everyone?
- Updates are gated behind **thorough** unit tests


User
kubectl-admin get everything
K8s Host
OPA
Admin Keys
S3
Bad policy push
Access Restricted Repo (Authz Policies)
CI/CD pipeline
OPA Policy Manager
Active Directory (User's group memberships)

System Reliability

- What if OPA doesn't respond for some reason?

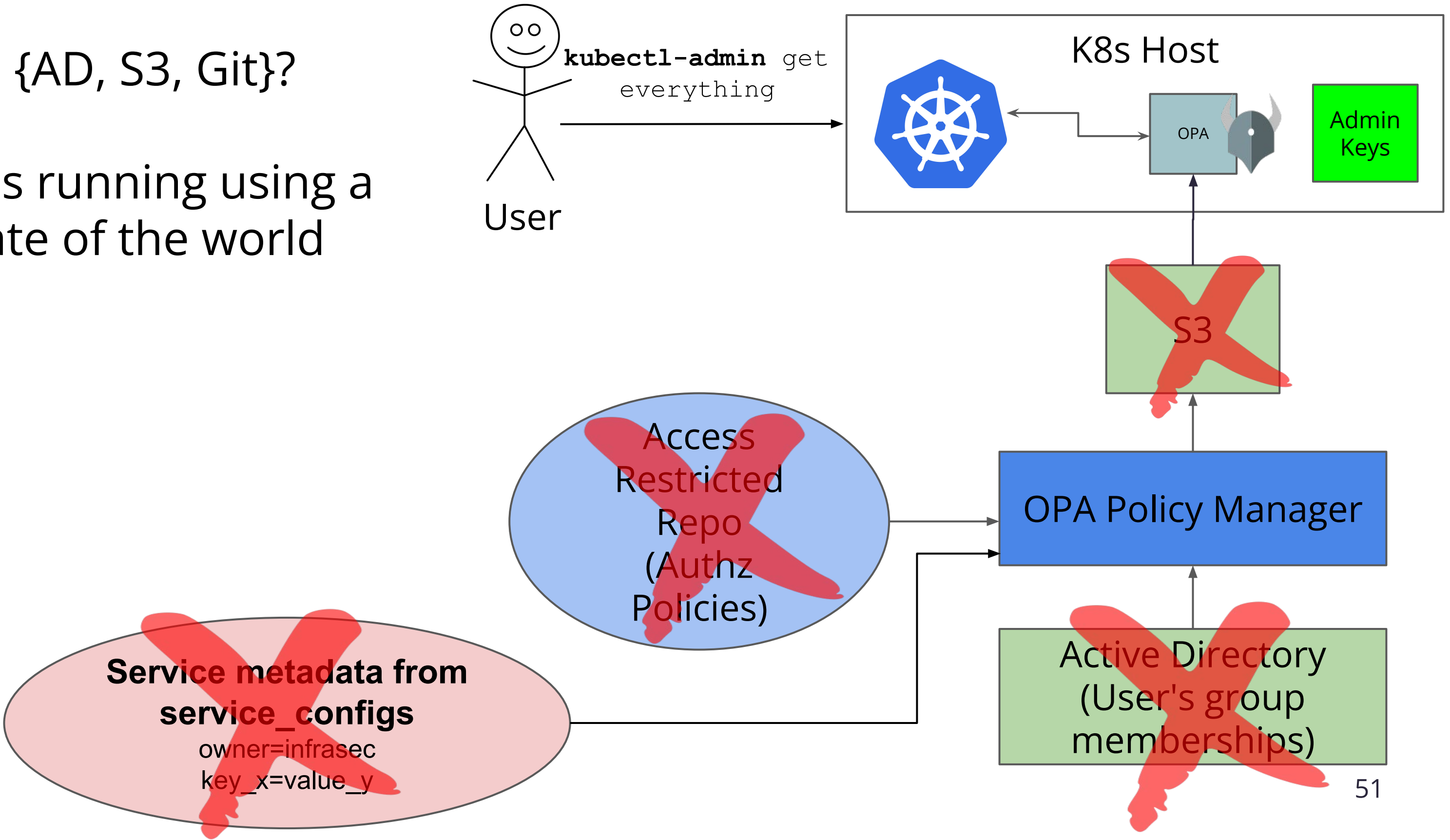
- Dedicated kubectl wrappers that use admin keys and RBAC policies just for admins



User
kubectl-admin get everything
K8s Host
OPA
Admin Keys
S3
Access Restricted Repo (Authz Policies)
OPA Policy Manager
Active Directory (User's group memberships)
Service metadata from service_configs
owner=infrasec
key_x=value_y

System Reliability

- Outage in {AD, S3, Git}?
- OPA keeps running using a frozen state of the world



User
kubectl-admin get everything
K8s Host
OPA
Admin Keys
S3
Access Restricted Repo (Authz Policies)
OPA Policy Manager
Service metadata from service_configs
owner=infrasec
key_x=value_y
Active Directory (User's group memberships)

Outline

Shortcomings and Future Improvements

- Not every resource has meaningful metadata labels (e.g., pv)
  - Admission controller: Make sure each created resource has metadata
- RBAC for service users **and** OPA for human users
- Okta authentication has a 1 hour TTL
- No time-limited access control.
  - Once a user gets the permissions, they keep it
- No service to sweep unused permissions

Conclusions

- Don't just blindly carry over your old security model
    - It's important to re-evaluate as your paradigm shifts
- Reliable system design makes a smooth review process for SRE teams
- Build the authorization system first
    - The actual least-privilege process is a separate, later project
- Least-privilege == less risk == better security

Questions