
KubeCon
CloudNativeCon
North America 2018
Securing Kubernetes With Admission Controllers

Who Am I?

Dave Strebel
Microsoft Global OSS Architect
Sig-Azure Co-Chair
Kubernetes Release Team
Twitter: @dave_Strebel

Azure

open source

@dave_strebel

So you’re going to deploy Kubernetes?

You're Going To Deploy What?

The Problem

- Dynamic nature of Cloud Native Patterns
- Tools **not** adopted for Cloud Native Patterns
- **Not** secure by default
- Clusters **not** immutable
- Policy becomes tribal knowledge and **not** defined in code

Approaches

- Manual Intervention
- Restrict users from creating objects


Then our architecture looks like this…
Our architecture looks like this…

Leads to frustration

Kubernetes Without Security Compliance!

Admission Controllers

Who's Using Admission Controllers?

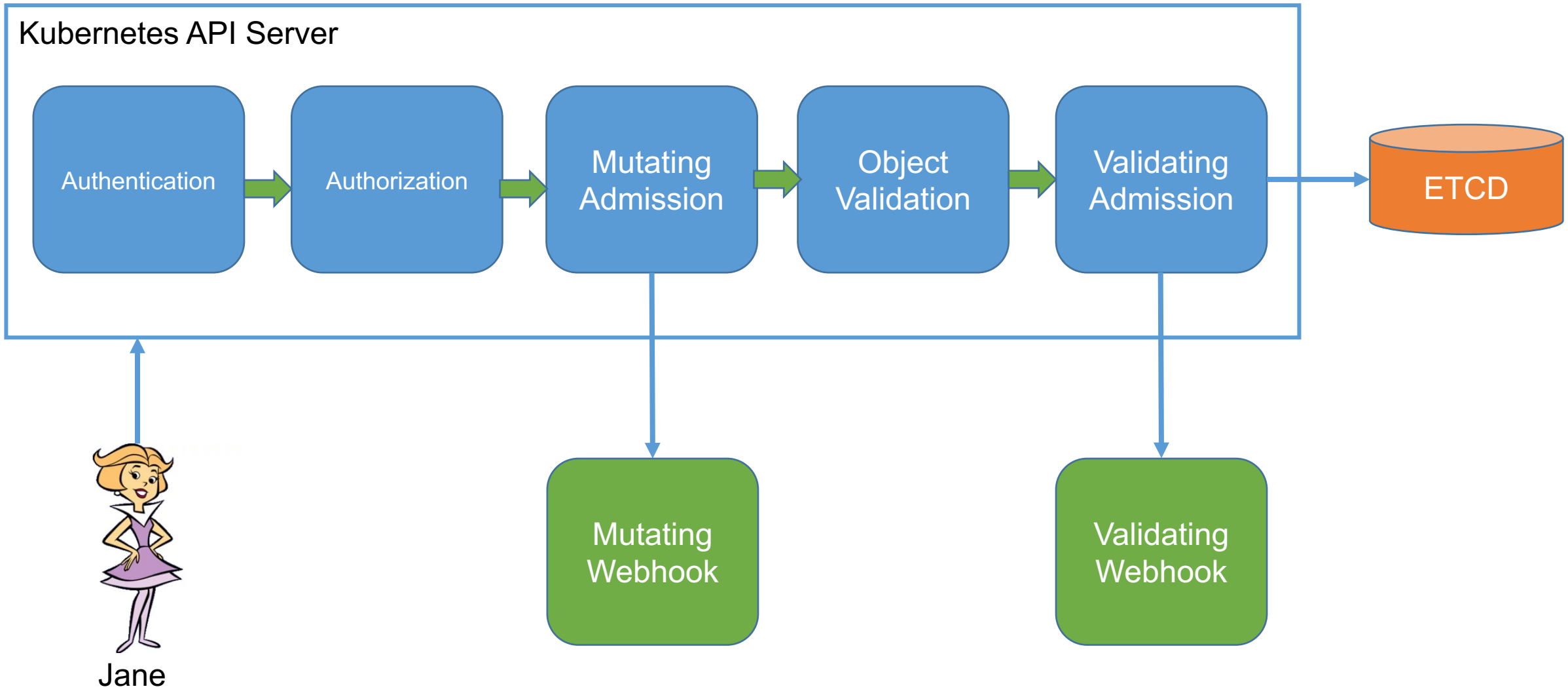
Default Admission Controllers

1. NamespaceLifecycle
2. LimitRanger
3. ServiceAccount
4. PersistentVolumeLabel
5. DefaultStorageClass
6. DefaultTolerationSeconds
7. ResourceQuota
8. Priority
9. MutatingAdmissionWebhook
10. ValidatingAdmissionWebhook

What Are Admission Controllers

An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized. - *Kubernetes.io*

How Admission Controllers Work



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…

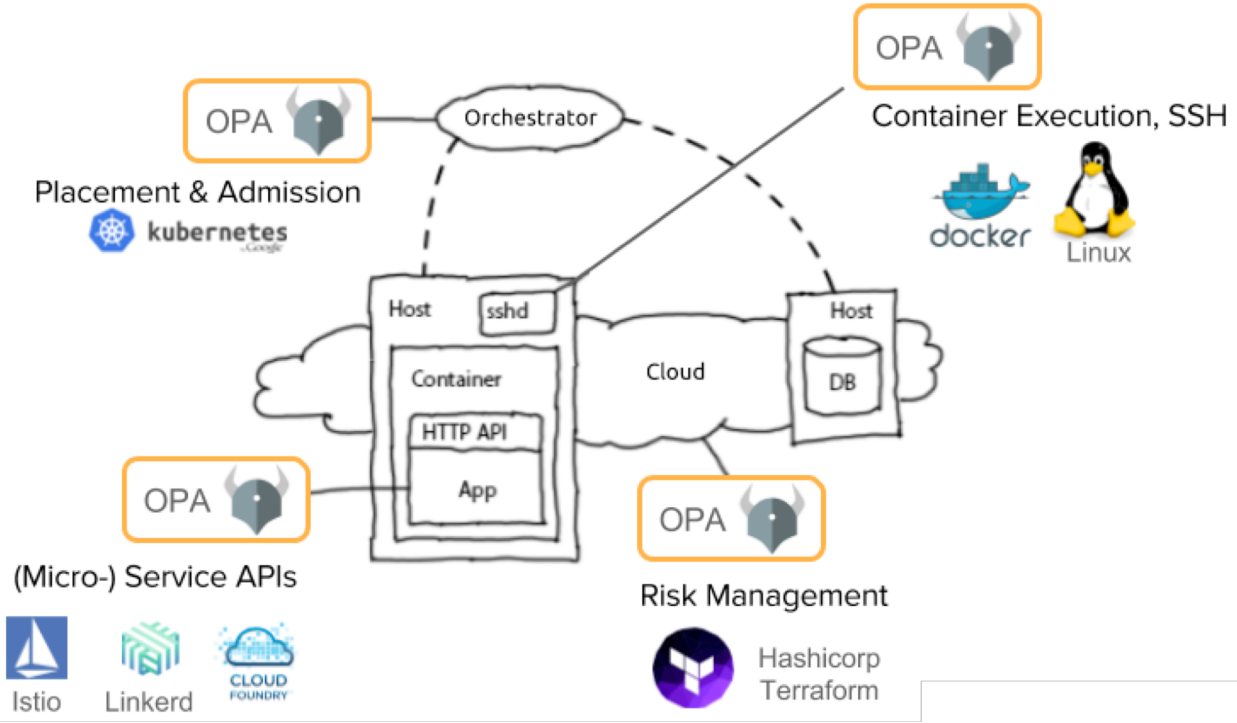
Sample Admission Webhook


```
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 }
```

How can you get policy enforcement without writing a bunch of custom logic?

You can use a general purpose policy engine

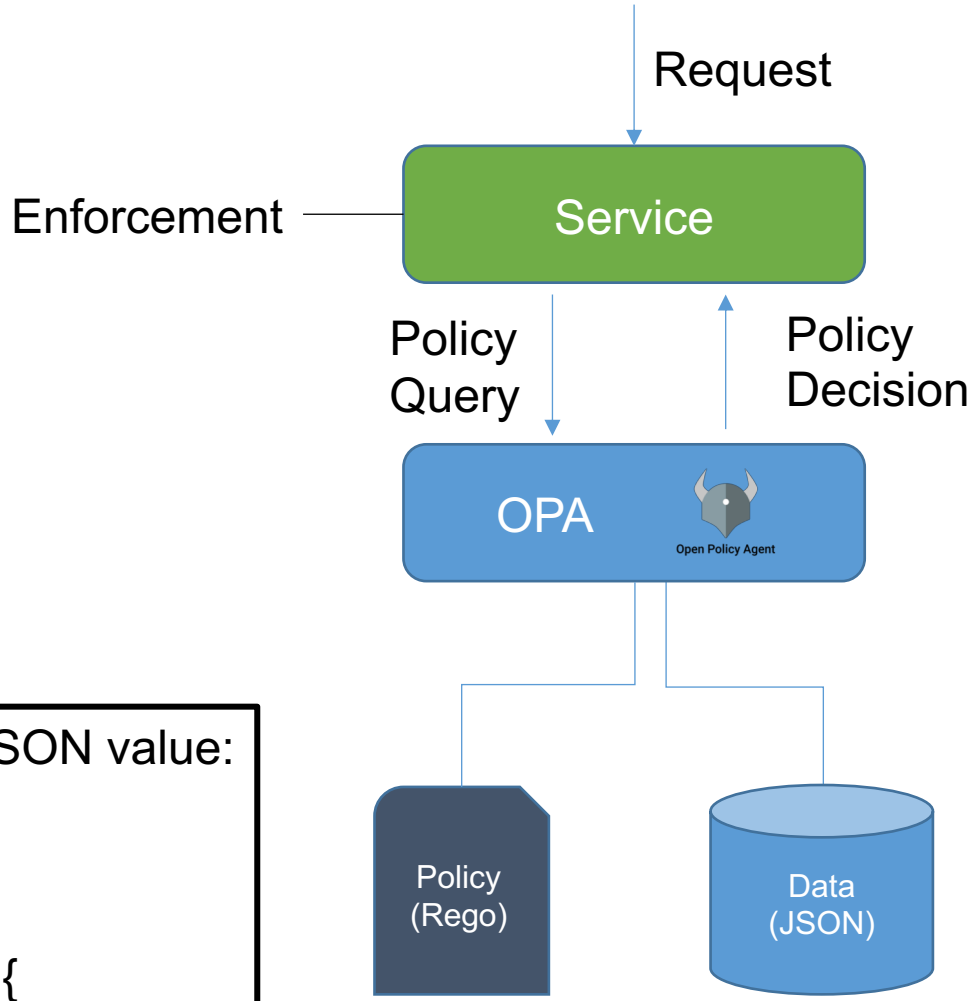
Open Policy Agent



- 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?

Developer

Ice Kube

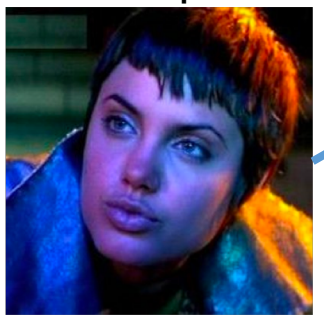
Deploys Apps

OPA Policy

Audits Platform

The Governor

Platform Operator

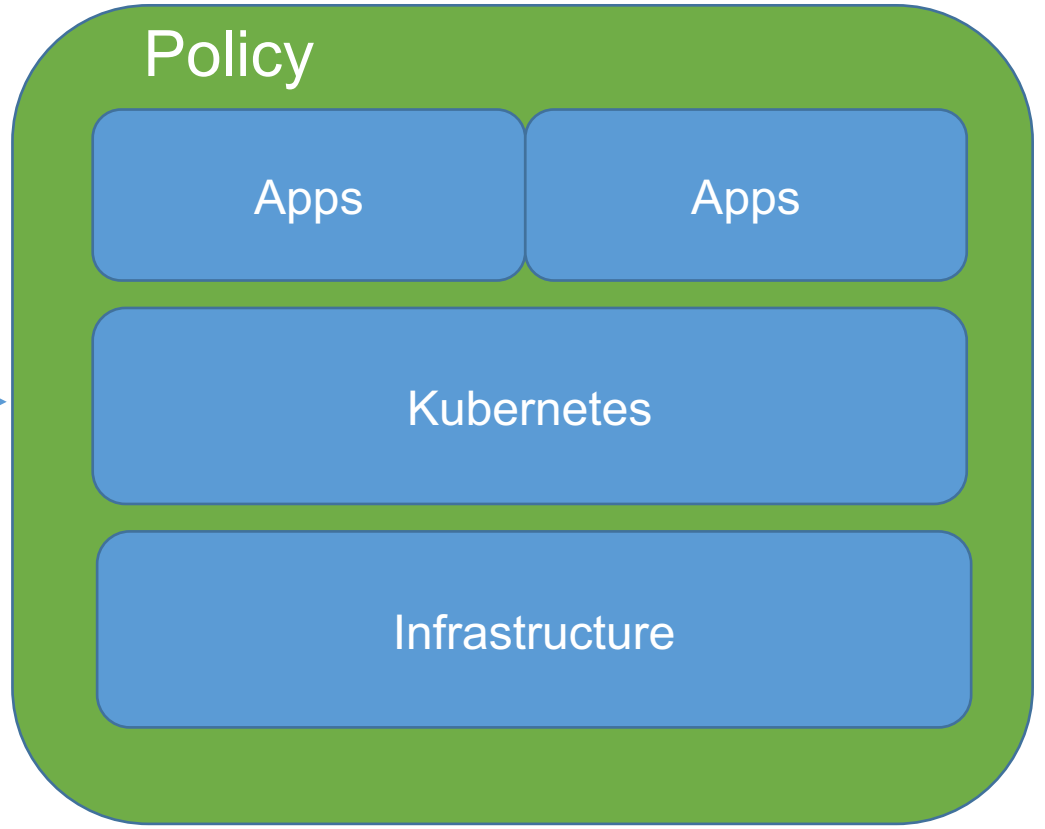


Acid Burn

Creates And Maintains

Immutable Platform

Deploy



Immutable Platform

But there’s more…

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>

Temporarily Home: https://github.com/Azure/kubernetes-policy-controller

Demos

request

```
apiVersion: v1
Kind: Service
Name: jetson-lb
Spec:
  type: LoadBalancer
….
```

evaluation

Mutate
Load Balancer

request

```
apiVersion: v1
Kind: Ingress
Name: elroy-ingress
Spec:
    host:
elroy.tugboatlabs
```

evaluation

Deny Conflicting
Ingress Host Names

request

kubectl exec api-server

evaluation

Restrict Access To
CRDs

Demo Time…..Excellent!

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.

Other Sessions

- Intro To Open Policy Agent – Case Study With Capital One and Intuit
- Deep Dive: Open Policy Agent


KubeCon
CloudNativeCon
North America 2018