Securing Kubernetes with Open Policy Agent (OPA) and Gatekeeper

Anton Sankov, 03.10.2022

Anton Sankov

- Senior Software Engineer @ VMware Carbon Black
- Passionate about Kubernetes Security

a_sankov

Anton Sankov

https://asankov.dev

Agenda

Why Kubernetes Security

Different aspects of Kubernetes Security

Implementing Security

Open Policy Agent (OPA) and Gatekeeper

Demo

Why Kubernetes Security?

Kubernetes is more than orchestrator, It is a platform

Kubernetes as a Platform

- Orchestrate containers and ensure availability
- Orchestrate storage and other resources (Ingress, certificates, etc.)
- Secrets and configuration management

- Kubernetes for CI/CD pipelines
- Kubernetes for managing Kubernetes

A system is only as secure as its weakest link

Kubernetes Exploits

An attacker that got access to your cluster can:

- Deploy malicious workloads/crypto miners
- Steal/Tamper with your data/secrets
- Do a container escape and get access to your whole infrastructure

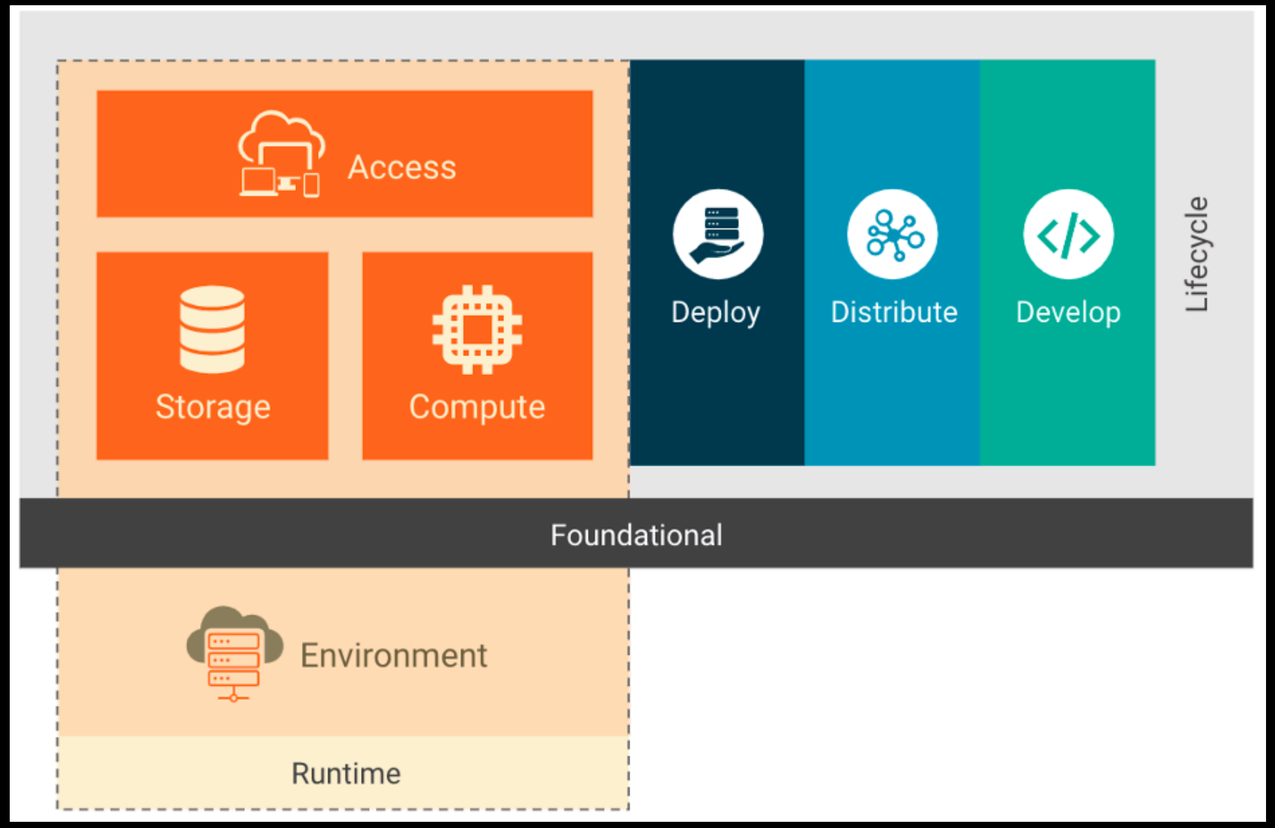
a_sankov

How to do Kubernetes Security?

Aspects of Kubernetes Security

According to the Cloud Native Security Whitepaper

- Develop
- Distribute
- Deploy
- Runtime



https://github.com/cncf/tag-security/tree/main/security-whitepaper/

a_sankov

Implementing (Some) Security

Deploy

Pre Flight Checks

Validate Image Integrity and Signature

Apply Image Runtime Policies

Apply Runtime Container Policies

Runtime Policies

Apply Runtime Security Controls

- Standards based: NIST*, CIS*

Host Security

- Vulnerabilities
- Compliance Controls
- Micro-segmentation

Container Security

1. Workload Isolation
2. Network Policy
3. File Integrity
4. Process Integrity
5. Syscalls

a_sankov

Goals

- Enforce rules for our Kubernetes Deployments
- Rules are checked when Deployment object is applied to the API server
- Deployment object should not be created if it violates the rule

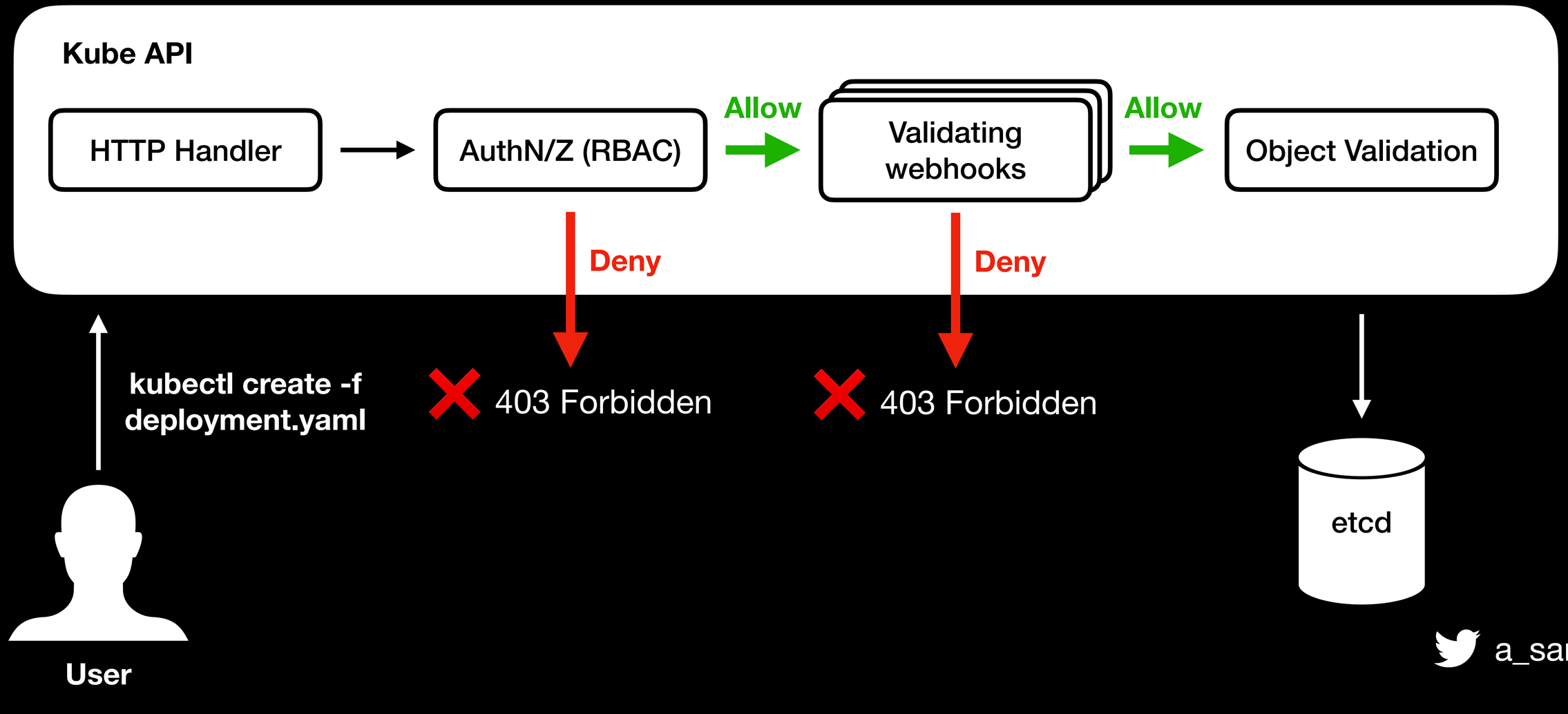
Solution: Validating Webhooks

- Pluggable mechanism for adding additional verification to Kubernetes resource being created/updated
- Can have many of them, Kubernetes calls all in order
- If a validating webhook denies the request, Kubernetes aborts the operation

- Anyone can write and plug-in their own

https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/

Kubernetes Validating Webhooks



a_sankov

Solution: Validating Webhooks

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: “admission.asankov.dev”
webhooks:
- name: “admission.asankov.dev”
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  [“CREATE”, “UPDATE”]
    resources:   ["Deployments"]
  clientConfig:
    url: "https://admission.asankov.dev/admission"
  admissionReviewVersions: ["v1", "v1beta1"]
  sideEffects: None
  timeoutSeconds: 5
```

Solution: Validating Webhooks

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: “admission.asankov.dev"
webhooks:
- name: “admission.asankov.dev”
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  [“CREATE”, “UPDATE”]
    resources:   ["Deployments"]
  clientConfig:
    url: "https://admission.asankov.dev/admission"
  admissionReviewVersions: ["v1", "v1beta1"]
  sideEffects: None
  timeoutSeconds: 5
```

Solution: Validating Webhooks

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: “admission.asankov.dev"
webhooks:
- name: “admission.asankov.dev”
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  [“CREATE”, “UPDATE”]
    resources:   ["Deployments"]
  clientConfig:
    url: "https://admission.asankov.dev/admission"
  admissionReviewVersions: ["v1", "v1beta1"]
  sideEffects: None
  timeoutSeconds: 5
```

Solution: Validating Webhooks

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: “admission.asankov.dev"
webhooks:
- name: “admission.asankov.dev”
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  [“CREATE”, “UPDATE”]
    resources:   ["Deployments"]
  clientConfig:
    url: "https://admission.asankov.dev/admission"
  admissionReviewVersions: ["v1", "v1beta1"]
  sideEffects: None
  timeoutSeconds: 5
```

Kubernetes will call **this URL** when **Deployments** are being **created or updated**.

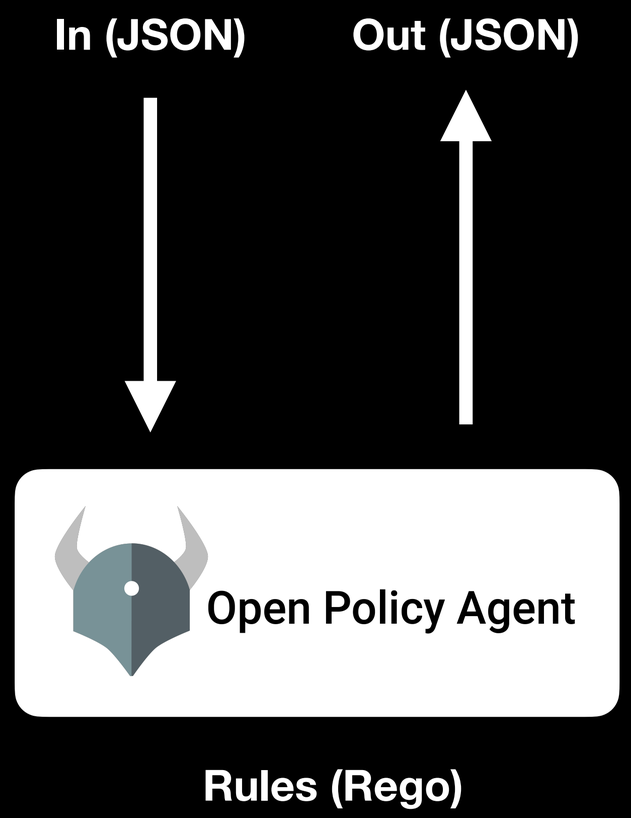
So… should I write my own Validating Webhook?

Not necessarily.

Open Policy Agent (OPA)

- Open-source General-purpose policy agent
- Write policies in **Rego** language
- In: JSON input
- Out: JSON output

- Does not have anything to do with Kubernetes



https://www.openpolicyagent.org/

a_sankov

A (really) simple Rego policy

Input (JSON):

```
{
  “conference”: {
    “name”: “KubeHuddle”
  }
}
```

Output (JSON):

```
{
  “allow”: true
}
```

Policy (Rego):

```
package policies

default allow = false

allow = true {
      input.conference.name = “KubeHuddle”
}
```

https://www.openpolicyagent.org/docs/latest/policy-language/

https://play.openpolicyagent.org/

a_sankov

A (really) simple Rego policy

Input (JSON):

```
{
  “conference”: {
    “name”: “SomeotherConf”
  }
}
```

Output (JSON):

```
{
  “allow”: false
}
```

Policy (Rego):

```
package policies

default allow = false

allow = true {
      input.conference.name = “KubeHuddle”
}
```

https://www.openpolicyagent.org/docs/latest/policy-language/

https://play.openpolicyagent.org/

a_sankov

A (less) simple Rego policy

Input (JSON):

```
{
  “conference”: {
    “name”: “SomeotherConf”,
    “venue”: “SomeotherVenue”
  }
}
```

Output (JSON):

```
{
  “violation”: [
    {"msg": "name and venue are wrong, - [SomeotherConf, SomeotherVenue]”}
  ]
}
```

Policy (Rego):

```
package policies

violations[{"msg": msg}] {
    input.conference.name != "KubeHuddle"
    input.conference.venue != “Edinburgh”
    msg = sprintf("name and venue are wrong - [%s, %s]", [input.conference.name, input.conference.venue])
}
```

https://www.openpolicyagent.org/docs/latest/policy-language/

https://play.openpolicyagent.org/

a_sankov

Rego rules are just chained AND conditions

```
package policies

violations[{"msg": msg}] {
    input.conference.name != "KubeHuddle"
    input.conference.venue != “Edinburgh”
    msg = sprintf("name and venue are wrong - [%s, %s]", [input.conference.name, input.conference.venue])
}
```

Translates to

```
if input.conference.name != “KubeHuddle" AND input.conference.venue != “Edinburgh” {
    msg = sprintf("name and venue are wrong - [%s, %s]", [input.conference.name, input.conference.venue])
    violations = append(violations, {“msg”: msg})
}
```

Which means that no message will be produced if conference.name is equal to “KubeHuddle” but the venue is different

A (less) simple Rego policy

Input (JSON):

```
{
  “conference”: {
    “name”: “SomeotherConf”,
    “venue”: “SomeotherVenue”
  }
}
```

Output (JSON):

```
{
  “violation”: [
    {“msg”: “name is wrong”},
    {“msg”: ”venue is wrong”}
  ]
}
```

Policy (Rego):

```
package policies

violations[{"msg": msg}] {
    input.conference.name != "KubeHuddle"
    msg := "name is wrong"
}

violations[{"msg": msg}] {
    input.conference.venue != “Edinburgh"
    msg := "venue is wrong"
}
```

A (less) simple Rego policy

Input (JSON):

```
{
  “conference”: {
    “name”: “SomeotherConf”,
    “venue”: “Edinburgh”
  }
}
```

Output (JSON):

```
{
  “violation”: [
    {“msg”: “name is wrong”}
  ]
}
```

Policy (Rego):

```
package policies

violations[{"msg": msg}] {
    input.conference.name != "KubeHuddle"
    msg := "name is wrong"
}

violations[{"msg": msg}] {
    input.conference.venue != “Edinburgh"
    msg := "venue is wrong"
}
```

Let’s add Kubernetes to the mix

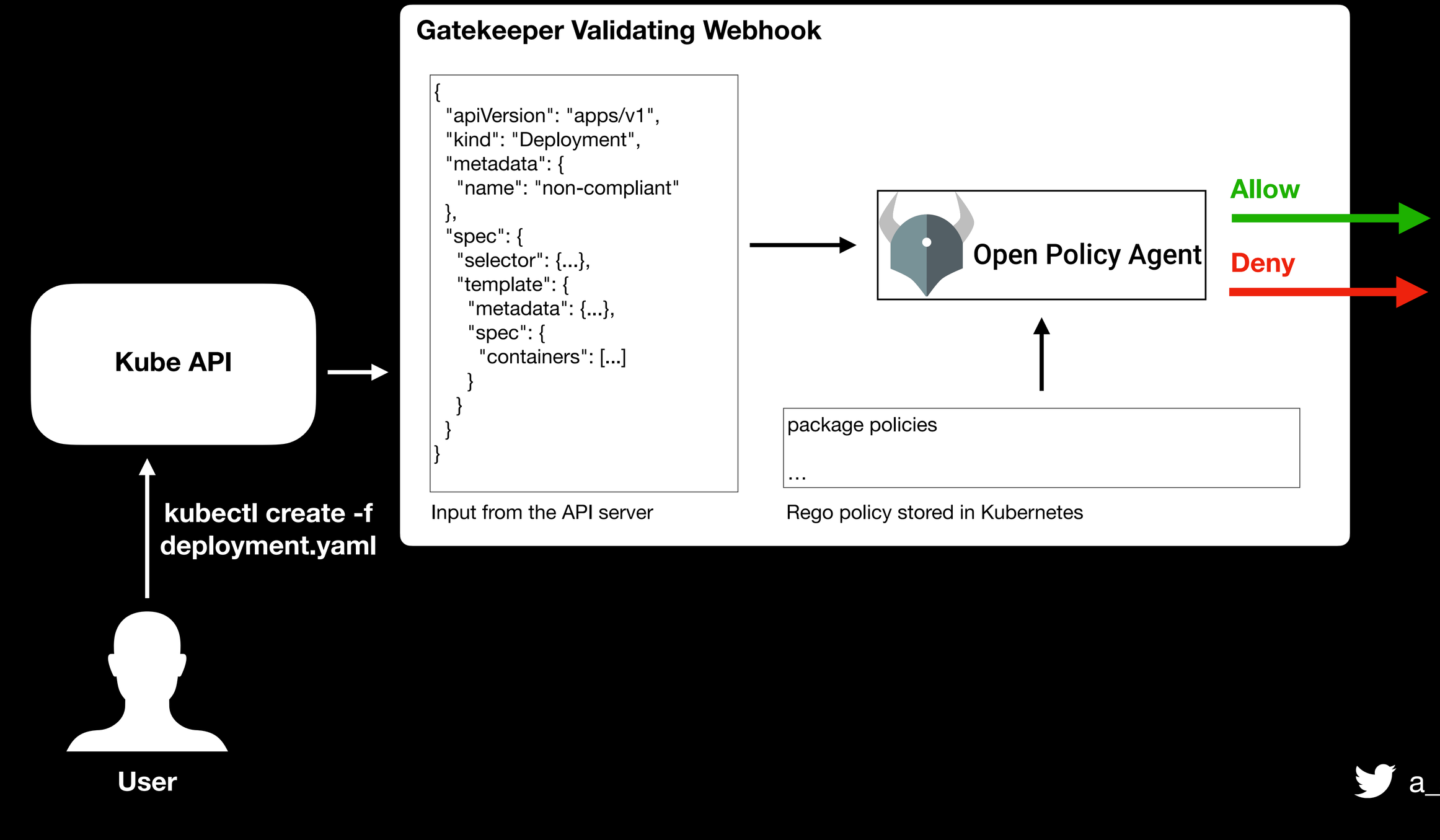
OPA Gatekeeper

- First-class integration between OPA and Kubernetes
- Implements a validating webhook

- Calls OPA with the Kubernetes object as JSON input
- Returns a response that says whether the action can be completed based on the existing policies
- Policies are stored as Kubernetes objects (CRDs)

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

a_sankov


Gatekeeper Validating Webhook
{
"apiVersion": "apps/v1",
"kind": "Deployment",
"metadata": {
"name": "non-compliant"
},
"spec": {
"selector": {...},
"template": {
"metadata": {...},
"spec": {
"containers": [...]
}
}
}
}
Input from the API server
Open Policy Agent
Allow
Deny
package policies
…
Rego policy stored in Kubernetes
Kube API
kubectl create -f deployment.yaml
User

a_sankov

Writing Gatekeeper policies

- Gatekeeper allows you to register policies as CRDs
- **ConstraintTemplate** - wraps the Rego policy
- **Constraint** - shows when to invoke the **ConstraintTemplate**

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

Writing Gatekeeper policies

In programming terms

- **ConstraintTemplate** - a function that describes the policy
- **Constraint** - shows when to invoke the function

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

a_sankov

Let’s write some Gatekeeper Policies

Goals

Reject Deployments that run as root

- A Deployment runs as Root if it has containers that:
  - Does not have a **securityContext**
  - OR does not have **securityContext.runAsUser** set
  - OR has **securityContext.runAsUser** set to **root uid (0)**

Demo

Alternatives

- Write your own Validation Webhook
- Kyverno
- Use ~~PodSecurityPolicies~~/ PodSecurityStandards
- Use a proprietary solution

Next steps

- Check out the links on the slides
- Other interesting talks about OPA:
  - https://youtu.be/Vdy26oA3py8
  - https://youtu.be/ejH4EzmL7e0
  - https://youtu.be/RDWndems-sk
- Write some policies

https://asankov.dev/k8s-sec-opa/

a_sankov

Summary

- Kubernetes security is important
- Validating Webhooks are a pluggable mechanism for enforcing more granular rules on our Kubernetes objects
- OPA is a general-purpose policy agent
- Gatekeeper is Kubernetes-native OPA adapter
- Write rules and policies as code and interact with them the same you interact with other Kubernetes resources

a_sankov

Questions?

Thank you!

 a_sankov

 Anton Sankov

 asankov.dev