Securing Kubernetes with Open Policy Agent (OPA)

Anton Sankov, 19.06.2022

Anton Sankov

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

a_sankov

Anton Sankov

https://asankov.dev

Agenda

Kubernetes

Kubernetes Security

Open Policy Agent (OPA) and Gatekeeper

Demo

Every project mentioned in this session is open-source

Kubernetes

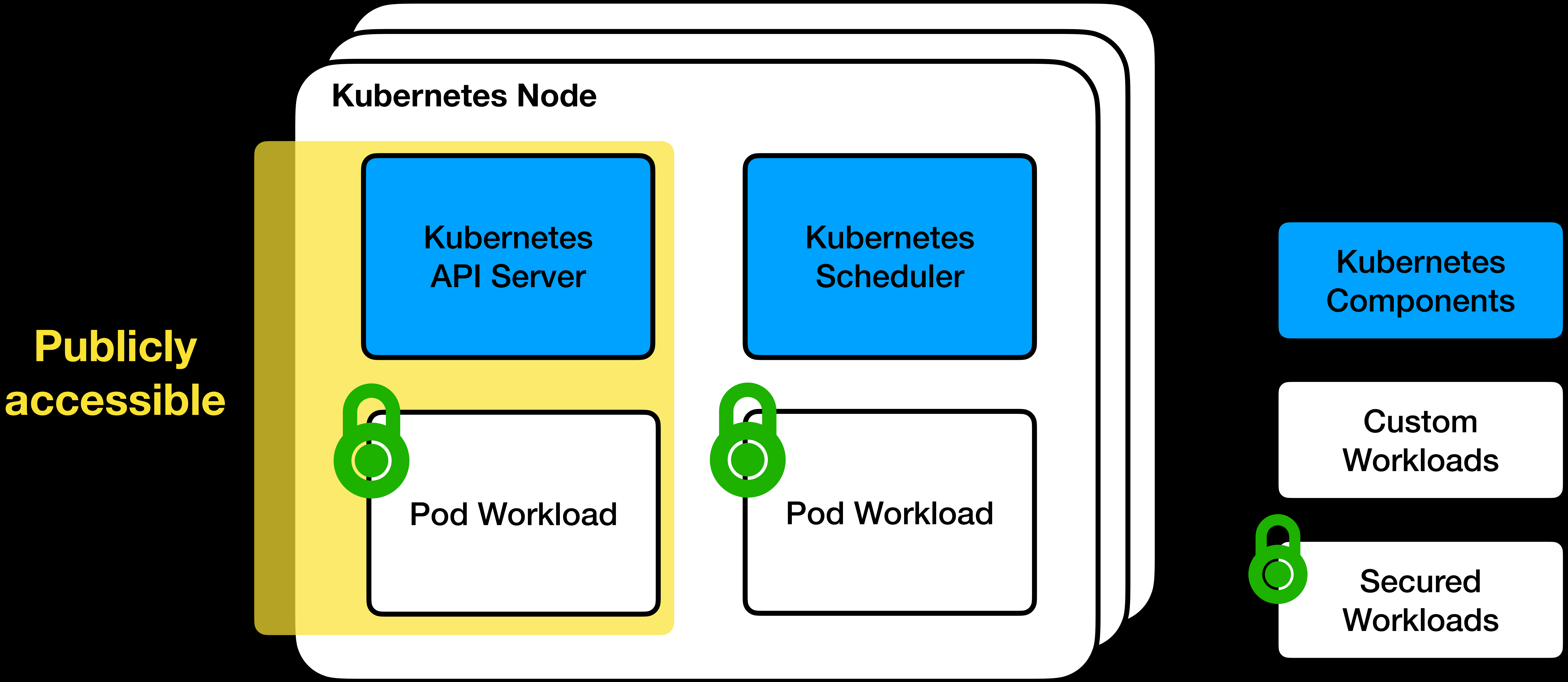
Open-source Container Orchestration tool

The de-facto standard for deploying applications in 2022

But this comes with some drawbacks

https://kubernetes.io/

Bigger attack surface



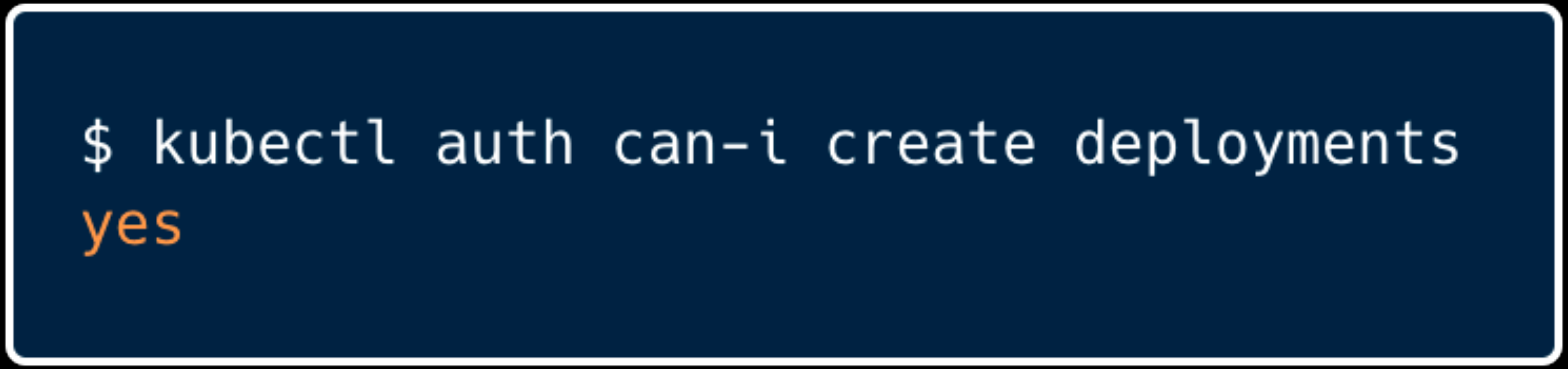
a_sankov

But Kubernetes has build-in security, right?

Yes.

Kubernetes RBAC

- Kubernetes has build-in RBAC (who can do what)
- Actions: **get**, **list**, **watch**, **delete**, **create**, **update**, etc.
- Resources: **pods**, **deployments**, **services**, etc.



```
$ kubectl auth can-i create deployments
yes
```

https://kubernetes.io/docs/reference/access-authn-authz/rbac/

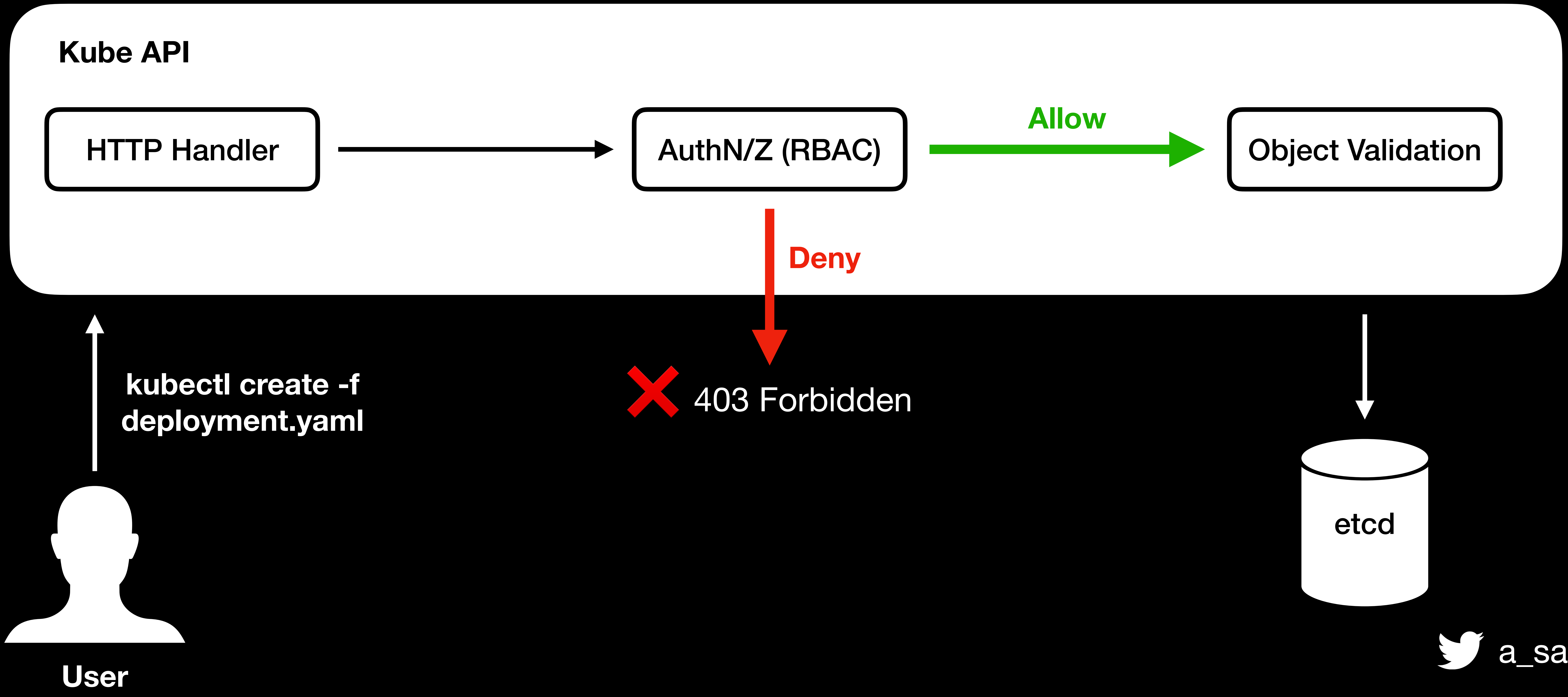
Kubernetes RBAC

	Get/List Deployments	Create/Update Deployments	Create RoleBindings
Developer	✅	❌	❌
DevOps / SRE	✅	✅	❌
Admin	❌	❌	✅

https://kubernetes.io/docs/reference/access-authn-authz/rbac/

a_sankov

Kubernetes RBAC



a_sankov

Problem: No granularity

- A user that can create Deployments can create ANY kind of Deployments

Problem: No granularity

- Each organisation has rules that want to be enforced on the Kubernetes resources
- Examples:
  - All workloads should have team labels (for cost and ownership measuring)
  - All workloads should have resource limits (so that a rogue workload does not bring the whole cluster down)
  - Only images from trusted repositories should be used (so that an attacked cannot deploy a malicious image)

But Kubernetes has build-in solution for that problem, right?

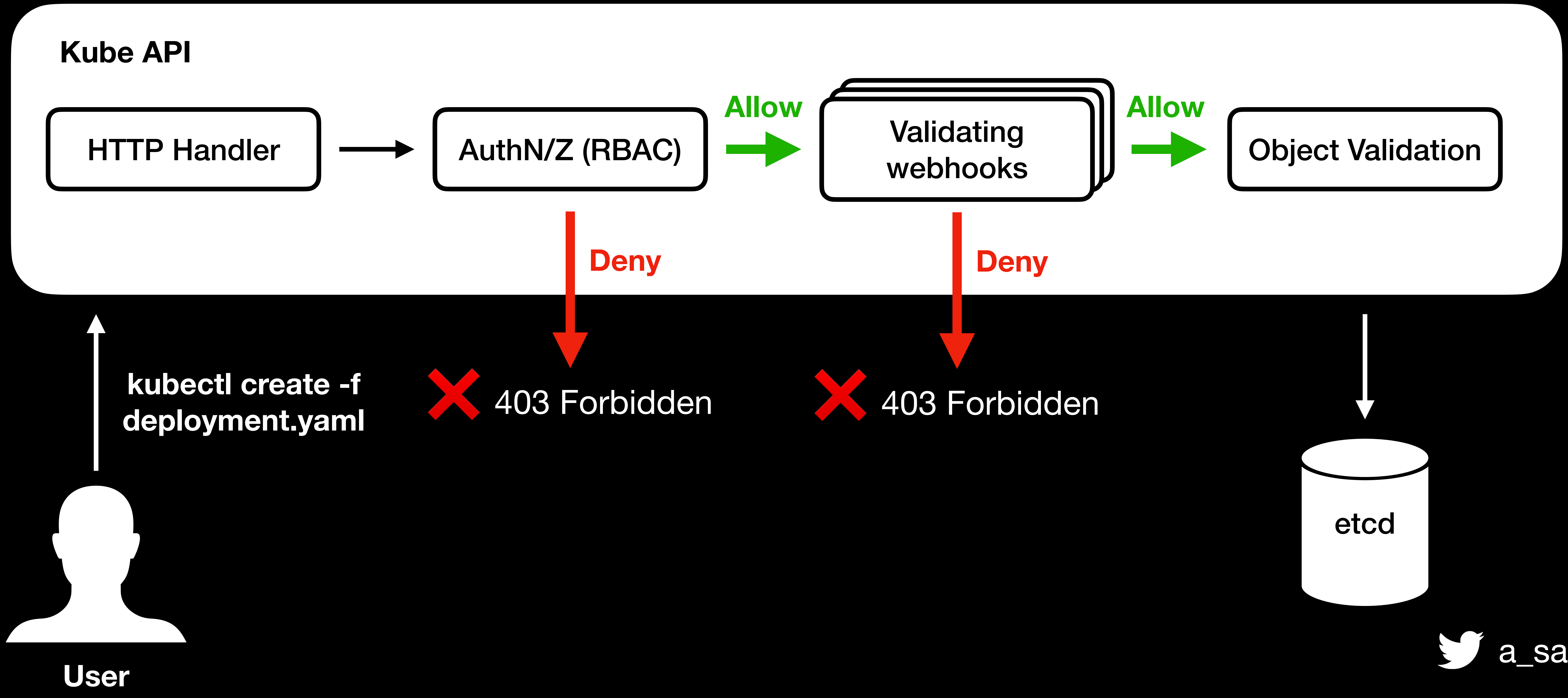
Sort of.

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.oscal.openlabs.cc”
webhooks:
- name: “admission.oscal.openlabs.cc”
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  ["CREATE"]
    resources:   ["Deployments"]
  clientConfig:
    url: "https://admission.oscal.openlabs.cc/admission"
  admissionReviewVersions: ["v1", "v1beta1"]
  sideEffects: None
  timeoutSeconds: 5
```

Solution: Validating Webhooks

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

Solution: Validating Webhooks

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

Solution: Validating Webhooks

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

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

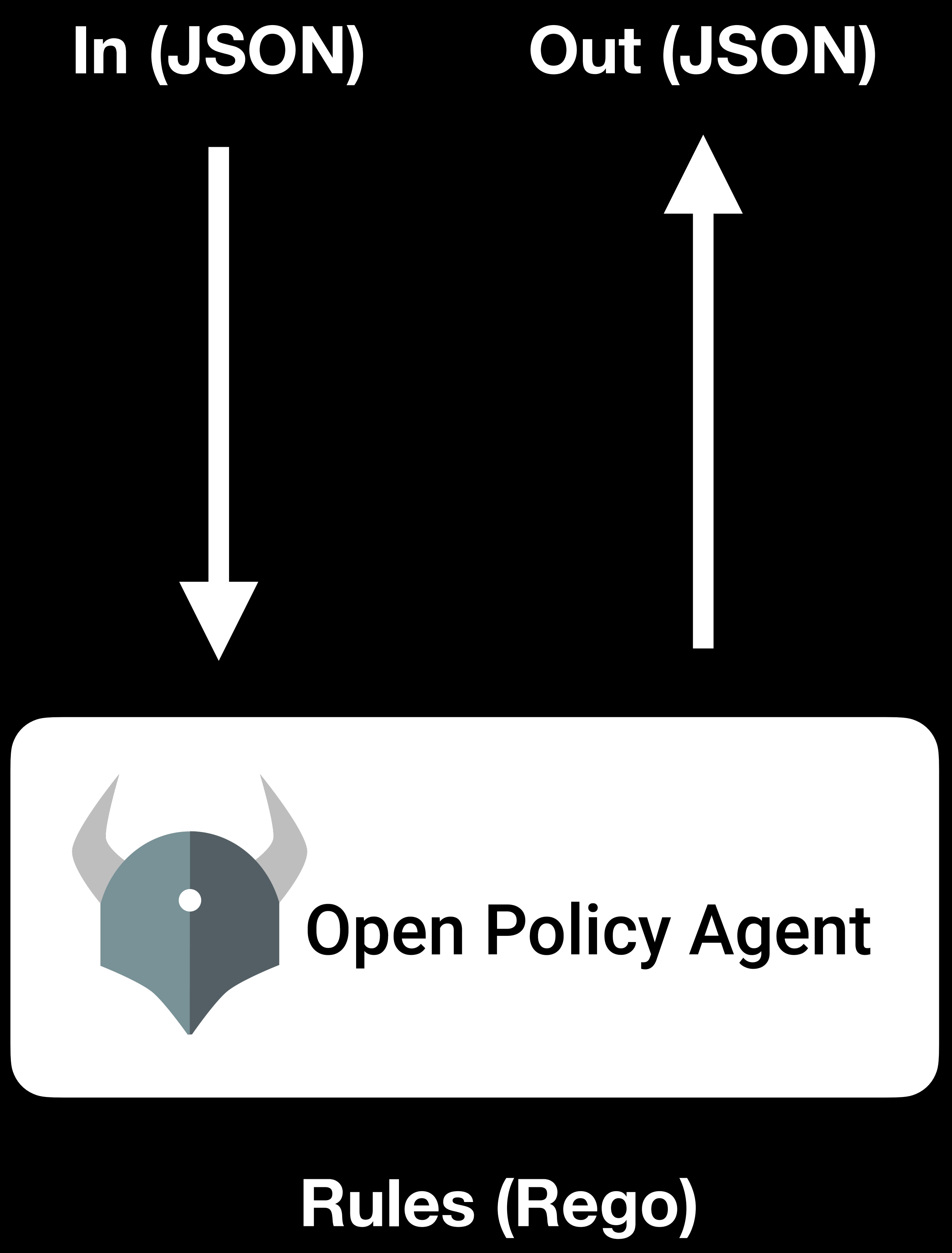
So… should I write my own Validating Webhook?

Not necessarily.

Open Policy Agent (OPA)

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

- Does not have anything to do with Kubernetes



In (JSON)

Out (JSON)

Rules (Rego)

https://www.openpolicyagent.org/

A (really) simple Rego policy

Input (JSON):

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

Output (JSON):

```
{
  “allow”: true
}
```

Policy (Rego):

```
package bsidesdemo

default allow = false

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

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 bsidesdemo

default allow = false

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

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 bsidesdemo

violations[{"msg": msg}] {
    input.conference.name != "OSCAL"
    input.conference.venue != “Tirana”
    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 bsidesdemo

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

Translates to

```
if input.conference.name != “OSCAL" AND input.conference.venue != “Tirana” {
    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 “OSCAL” 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 bsidesdemo

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

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

A (less) simple Rego policy

Input (JSON):

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

Output (JSON):

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

Policy (Rego):

```
package bsidesdemo

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

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

Good, but…

- Nothing so far was Kubernetes related
- The rules had some hard-coded values, which are not suitable for real environments

enter … Gatekeeper

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

Writing Gatekeeper policies

Gatekeeper represents Policies as Kubernetes objects (CRDs)

ConstraintTemplate - describes the Rego rules and the provided data

Constraint - shows how the **ConstraintTemplate** should be enforced

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

Writing Gatekeeper policies

In programming terms:

ConstraintTemplate - a function that describes the policy, accepts arguments and returns a response

Constraint - shows how and when to invoke the function (what arguments to pass)

Writing Gatekeeper policies

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            labels:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          provided := {label | input.review.object.metadata.labels[label]}
          required := {label | label := input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("you must provide labels: %v", [missing])
        }
```

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: deployments-must-have-gk
spec:
  match:
    kinds:
      - apiGroups: [“*"]
        kinds: [“Deployments"]
  parameters:
    labels: ["gatekeeper"]
```

a_sankov

Writing Gatekeeper policies

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            labels:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          provided := {label | input.review.object.metadata.labels[label]}
          required := {label | label := input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("you must provide labels: %v", [missing])
        }
```

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: deployments-must-have-gk
spec:
  match:
    kinds:
      - apiGroups: [“*"]
        kinds: [“Deployments"]
  parameters:
    labels: ["gatekeeper"]
```

Tells Kubernetes to invoke this rule ONLY when Deployments are being created. It will not be invoked for any other kind.

a_sankov

Writing Gatekeeper policies

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            labels:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          provided := {label | input.review.object.metadata.labels[label]}
          required := {label | label := input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("you must provide labels: %v", [missing])
        }
```

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: deployments-must-have-gk
spec:
  match:
    kinds:
      - apiGroups: [“*”]
        kinds: [“Deployments"]
  parameters:
    labels: ["gatekeeper"]
```

Parametrizes some parameters in the rule so that we can reuse ConstraintTemplates by create new Constraints (much like we reuse function)

a_sankov

Writing Gatekeeper policies

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        # Schema for the `parameters` field
        openAPIV3Schema:
          type: object
          properties:
            labels:
              type: array
              items:
                type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          provided := {label | input.review.object.metadata.labels[label]}
          required := {label | label := input.parameters.labels[_]}
          missing := required - provided
          count(missing) > 0
          msg := sprintf("you must provide labels: %v", [missing])
        }
```

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: deployments-must-have-gk
spec:
  match:
    kinds:
      - apiGroups: [“*"]
        kinds: [“Deployments"]
  parameters:
    labels: ["gatekeeper"]
```

The Kubernetes object being created.
Input provided by Gatekeeper

a_sankov

Demo

Alternatives

- Just use RBAC
- Lower the visibility of your Kubernetes as much as possible and hope someone does not get into your private network
- 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
- Go write some policies

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

Summary

- Build-in Kubernetes security (RBAC) is not enough for most organisations
- 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

Questions?

Thank you!

a_sankov

asankov.dev

Anton Sankov