



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

SIG Auth: Right-sized Access Control & Pull with Proof

Stanislav Láznička, Microsoft
Lucas Käldeström, Upbound



KEP updates in 1.36



GA

- ☐ [KEP 740](#) External signing for SA tokens
- ☐ [KEP-2862](#) Fine-grained Kubelet API Authorization
- ☐ [KEP-5018](#) DRA: AdminAccess
- ☐ [KEP-5538](#) CSI driver: opt-in for service account tokens via secrets field



Beta

- ☐ [KEP 4317](#) Pod Certificates
- ☐ [KEP-3104](#) Kuberc allowlist for client-go credential plugins
- ☐ [KEP-5284](#) Constrained impersonation
- ☐ Secrets Store Sync Controller, in progress



Alpha

- ☐ [KEP-3926](#) Handling undecryptable resources
- ☐ [KEP-5681](#) Conditional Authorization - KEP merged, implementation in review
- ☐ [KEP-5917](#) (Still in review) Authenticator Provided Authorization Constraints





KubeCon



CloudNativeCon

Europe 2026

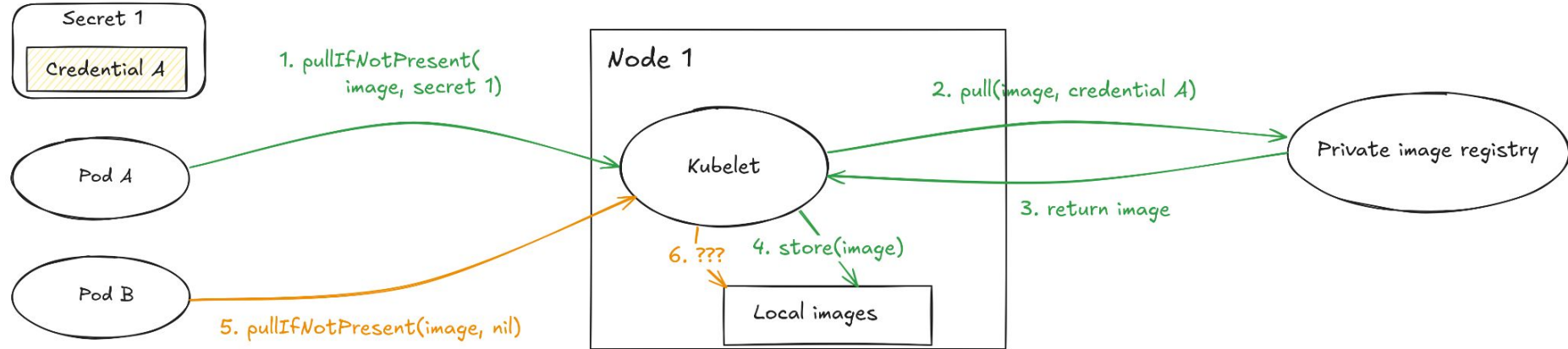
Auth changes to image pulling

Anish Ramasekar, Stanislav Láznička

Auth changes to image pulling

- Kube 1.32
- Authentication
 - Service Account Tokens for Kubelet Credential Providers (**KEP-4412**)
 - Continues push for ephemeral credentials
 - Cloud providers - workload identities
 - IfNotPresent/Never pull policy unsafe!
- Authorization
 - Ensure Secret Pulled Images (**KEP-2535**)
 - Authorize access to images on nodes

Accessing pulled images



KubeCon



CloudNativeCon

Europe 2026

Current solution

- AlwaysPullImages admission controller
 - Users must be aware of the issue
 - Puts cluster egress and remote registries in the critical path
 - Adds network overhead
 - Requires unnecessary authentications

What can we do?

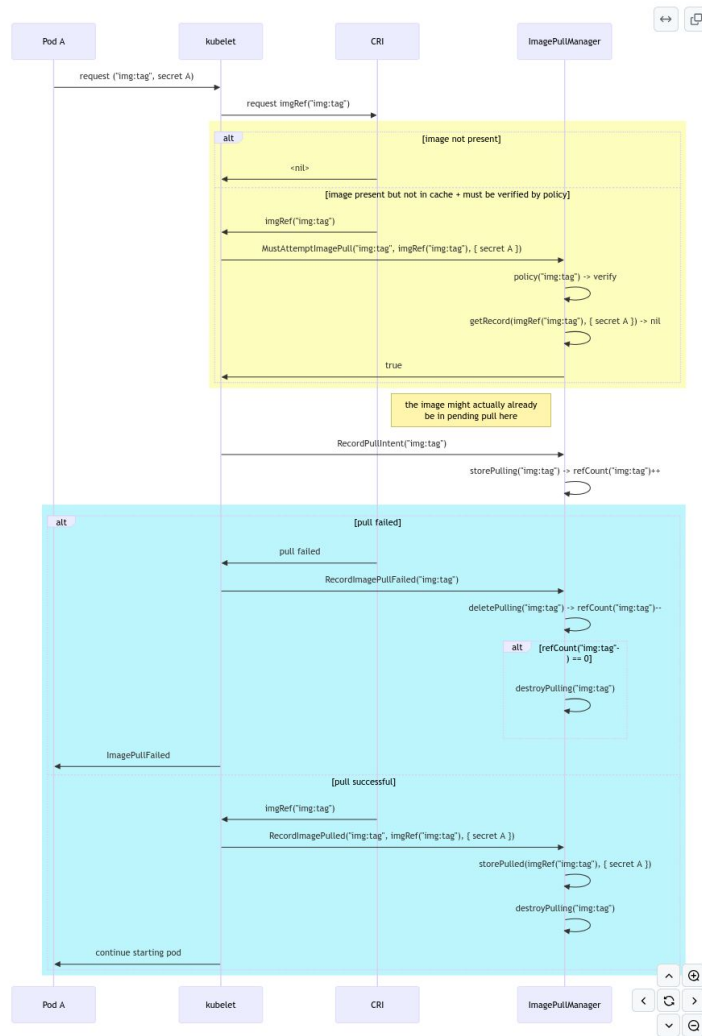
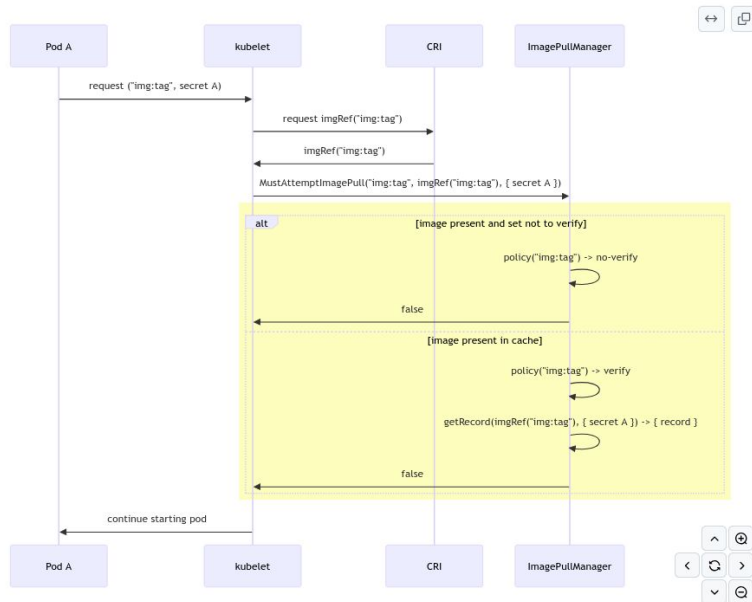
What information does Kubelet have when pulling images?

```
type ImageManager interface {  
    // EnsureImageExists ensures that image specified by `requestedImage` exists.  
    EnsureImageExists(ctx context.Context, objRef *v1.ObjectReference,  
        pod *v1.Pod, requestedImage string, pullSecrets []v1.Secret,  
        podSandboxConfig *runtimeapi.PodSandboxConfig, podRuntimeHandler string,  
        pullPolicy v1.PullPolicy,  
    ) (imageRef, message string, err error)  
}
```

The How

- **Don't require pulling unless really necessary**
- Build `image` $\Leftrightarrow$ `pull credentials` mapping
 - Persist it between kubelet restarts
 - Query it for each `IfNotPresent/Never` pull
 - Handle kubelet restarts mid-pull
 - i. **Pulling** records
 - ii. **Pulled** records
- Preloaded images?
 - 4 different policies:
 - i. `AlwaysVerify`
 - ii. `NeverVerifyPreloadedImages`
 - iii. `NeverVerifyAllowlistedImages`
 - iv. `NeverVerify`

The Real How (simplified)



Things users see

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
imagePullCredentialsVerificationPolicy: NeverVerifyAllowlistedImages
preloadedImagesVerificationAllowlist: ["test.test/repo/*"]
```

/var/lib/kubelet/image_manager

```
pulling/sha256-aef2af226629a35d5f3ef0fdbb29fdbebf038d0acd8850590e8c48e1e283aa56:
  {"kind": "ImagePullIntent", "apiVersion": "kubelet.config.k8s.io/v1beta1",
   "image": "docker.io/testing/test:latest"}
```

```
pulled/sha256-917e8b3439bf8a7a6f37ffd2d2ddfdafac8a251bf214a0be39675742b420b1a:
  {"kind": "ImagePulledRecord", "apiVersion": "kubelet.config.k8s.io/v1alpha1",
   "lastUpdateTime": "2024-10-21T12:26:40Z", "imageRef": "testimageref-sa",
   "credentialMapping": {"docker.io/testing/test": {
     "kubernetesServiceAccounts": [
       {"uid": "test-sa-uid", "namespace": "default", "name": "test-sa"}]}}} }
```

Read all about it

- [KEP 2535 design document](#)
- Kubernetes.io [documentation page on images](#)
- Kubernetes.io blog post: [Kubernetes v1.33: Image Pull Policy the way you always thought it worked!](#)



KubeCon



CloudNativeCon

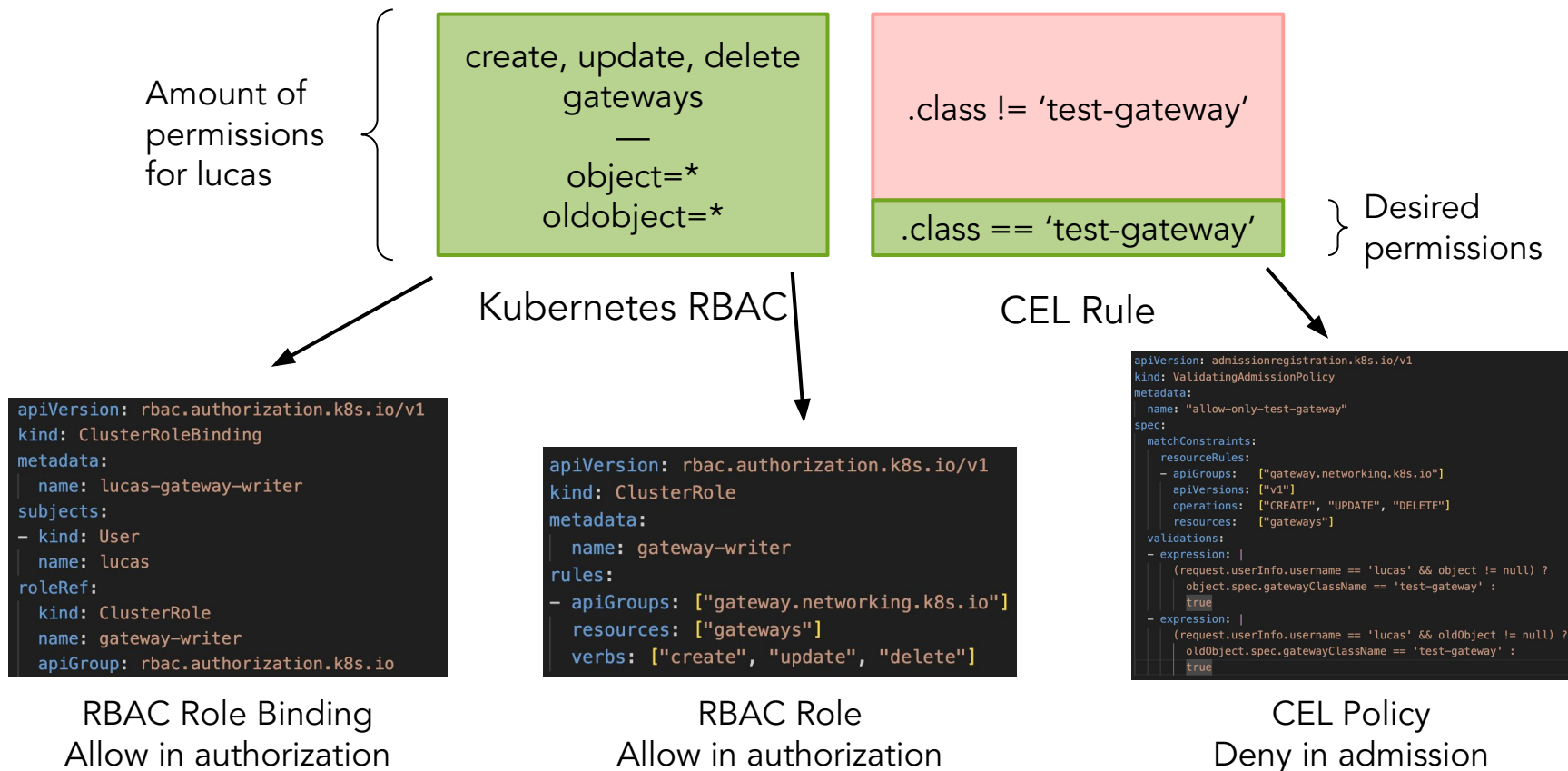
Europe 2026

Conditional Authorization

Lucas Käldeström



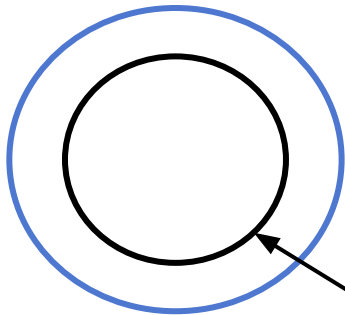
Current problem: “Over-grant” in RBAC, deny in VAP



Related: KEP-5284 Constrained Impersonation

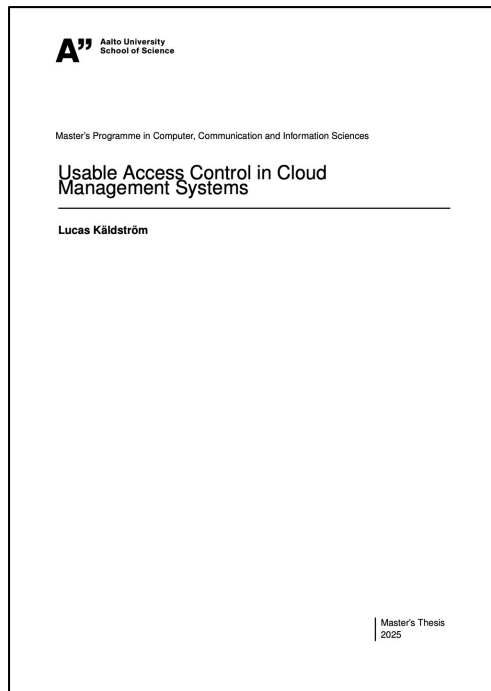
- Constrained Impersonation allows for a way to restrict what an impersonator can do.
- Can be used for “on behalf of” semantics
- Feature owners: @qiujian16, @enj

Your permissions

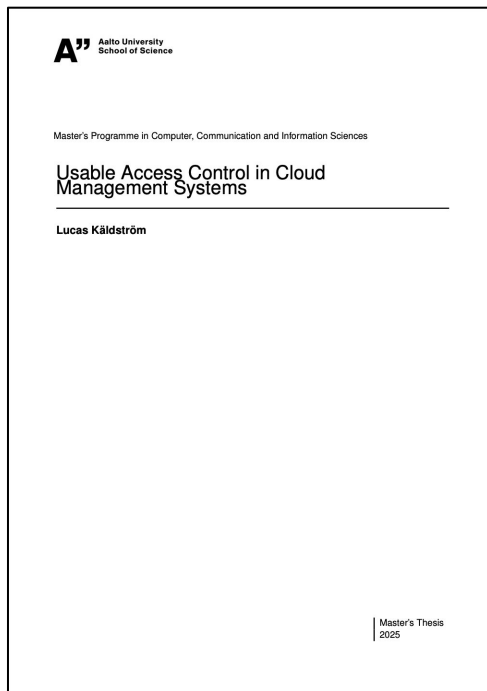


**Impersonator's
permissions**

MSc thesis: Usable Access Control in Cloud Management Systems



MSc thesis: Usable Access Control in Cloud Management Systems



[KEP-5681](#)

Kubernetes Enhancement Proposal 5681: Conditional Authorization



kubernetes

Howdy! Is the request
with <metadata>
authorized?



Authorizer

It depends. Only if
`secret.type == "tls"`

new

Use-cases

- Allow a principal to only read Secrets with a given label

Use-cases

- Allow a principal to only read Secrets with a given label
- **Allow a principal to update an object only when a sensitive field is unchanged**

Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- **Allow a principal to create objects only with certain names**

Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- Allow a principal to create objects only with certain names
- **Allow a node agent to only access objects referring to them**

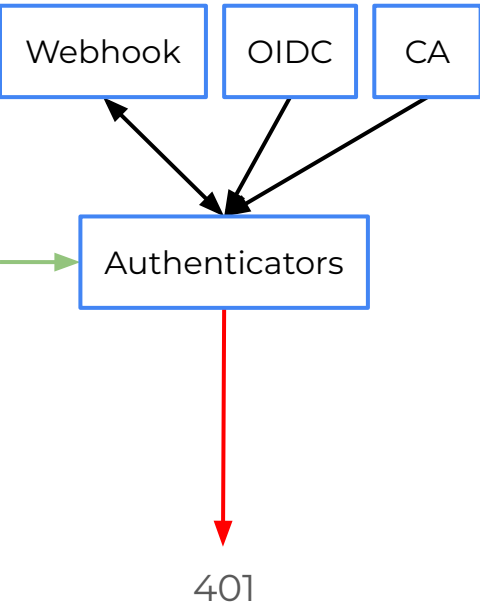
Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- Allow a principal to create objects only with certain names
- Allow a node agent to only access objects referring to them
- **Allow a controller to only add/remove its own finalizer**

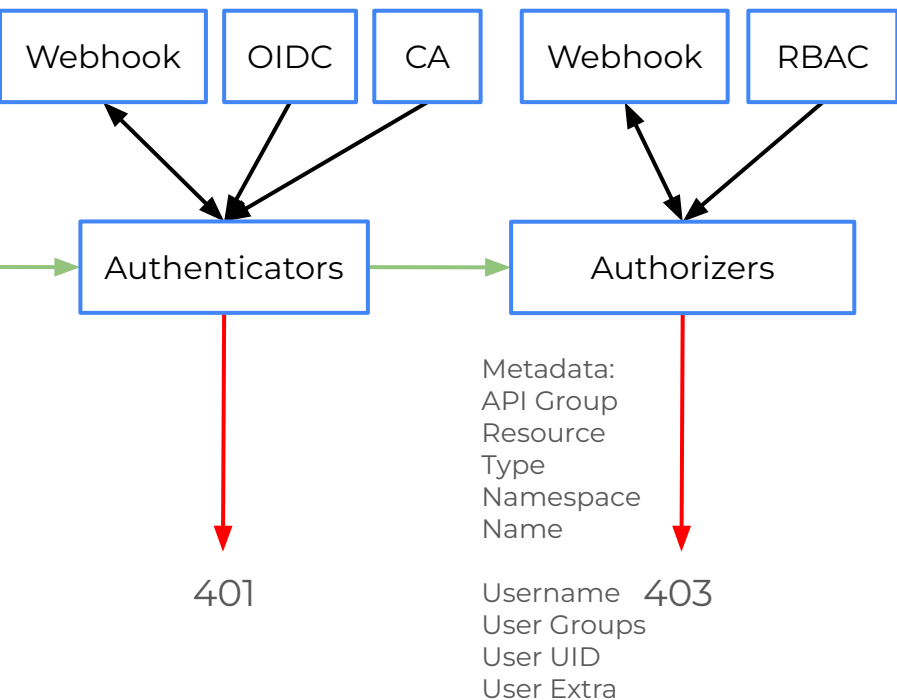
Use-cases

- Allow a principal to only read Secrets with a given label
- Allow a principal to update an object only when a sensitive field is unchanged
- Allow a principal to create objects only with certain names
- Allow a node agent to only access objects referring to them
- Allow a controller to only add/remove its own finalizer
- **Deny everyone except admins to set a sensitive field (e.g. GPU adminAccess)**

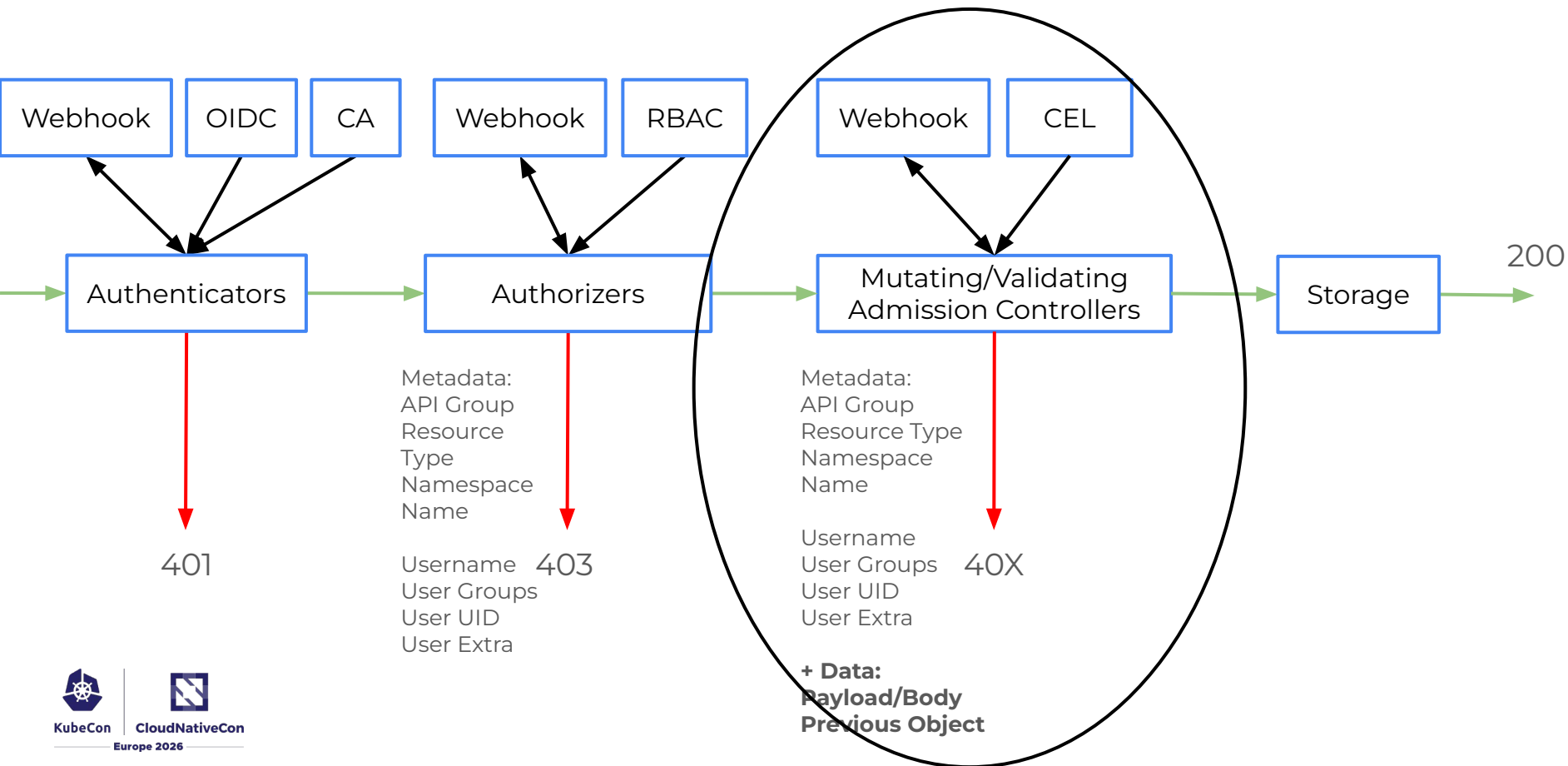
Recall admission control for write requests



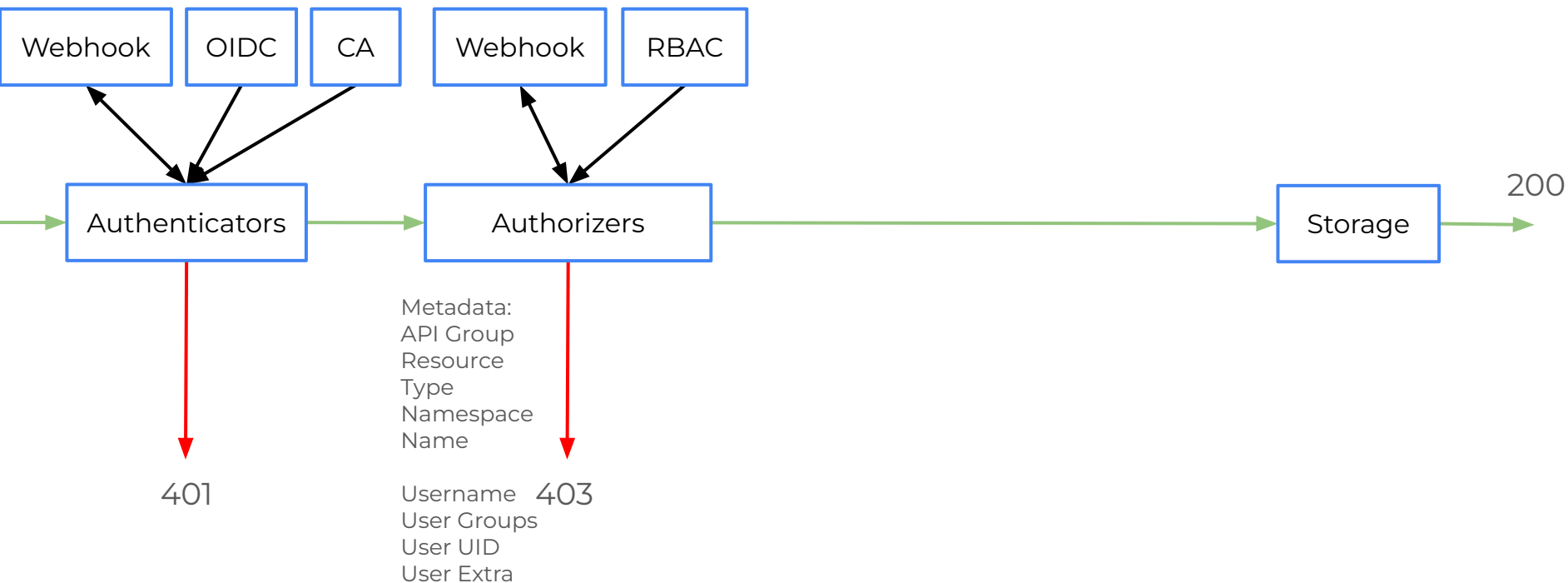
Recall admission control for write requests



Recall admission control for write requests



Recall admission control for read requests



Validating Admission Control

For a request to succeed, all admission controllers must consider the object “valid”
Admission control never gives permissions, but can only remove (deny policy)

Webhook

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  operation: CREATE
  name: foo-pod
  object:
    kind: Pod
    ...
response:
  allowed: true
```

CEL

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "allow-only-test-gateway"
spec:
  matchConstraints:
    resourceRules:
      - apiGroups: ["gateway.networking.k8s.io"]
        apiVersions: ["v1"]
        operations: ["CREATE", "UPDATE", "DELETE"]
        resources: ["gateways"]
  validations:
    - expression: |
        (request.userInfo.username == 'lucas' && object != null) ?
        object.spec.gatewayClassName == 'test-gateway' :
        true
    - expression: |
        (request.userInfo.username == 'lucas' && oldObject != null) ?
        oldObject.spec.gatewayClassName == 'test-gateway' :
        true
```

Unified policies through Partial Evaluation

What if it was possible to declare in one place that

“lucas can create gateways when class='test-gateway'”?

Unified policies through Partial Evaluation

What if it was possible to declare in one place that

“lucas can create gateways when class='test-gateway'”?

In pseudo-code, it could look something like:

```
“request.verb == ‘create’ &&
```

```
request.apiGroup == ‘gateway.networking.k8s.io’ &&
```

```
request.resource == ‘gateways’ &&
```

```
request.userInfo.username == “lucas” &&
```

```
has(request.object.v1) &&
```

```
request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark request **object** as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

```
“request.verb == ‘create’ &&  
request.apiGroup == ‘gateway.networking.k8s.io’ &&  
request.resource == ‘gateways’ &&  
request.userInfo.username == “lucas” &&  
has(request.object.v1) &&  
request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark request **object** as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

“**true** &&

`request.apiGroup == 'gateway.networking.k8s.io' &&`

`request.resource == 'gateways' &&`

`request.userInfo.username == "lucas" &&`

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == 'test-gateway'”`

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark `request.object` as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

`“true &&`

`true &&`

`request.resource == 'gateways' &&`

`request.userInfo.username == “lucas” &&`

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == 'test-gateway’”`

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark `request.object` as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

`“true &&`

`true &&`

`true &&`

`request.userInfo.username == “lucas” &&`

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == ‘test-gateway’”`

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark `request.object` as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

`“true &&`

`true &&`

`true &&`

`true &&`

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == 'test-gateway’”`

Unified policies through Partial Evaluation

One trick is to use **partial evaluation**. Mark `request.object` as “unknown”, but evaluate the expression as much as possible.

If lucas does a `kubectl create -f gateway.yaml`, the following happens:

“true &&

true &&

true &&

true &&

true &&

Condition on the request object,
defer evaluation to when decoded!

`request.object.v1.spec.gatewayClassName == 'test-gateway'”`



KubeCon



CloudNativeCon

Europe 2026

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

```
“request.verb == ‘create’ &&
```

```
request.apiGroup == ‘gateway.networking.k8s.io’ &&
```

```
request.resource == ‘gateways’ &&
```

```
request.userInfo.username == “lucas” &&
```

```
has(request.object.v1) &&
```

```
request.object.v1.spec.gatewayClassName == ‘test-gateway’”
```

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

“`true &&`

`request.apiGroup == 'gateway.networking.k8s.io' &&`

`request.resource == 'gateways' &&`

`request.userInfo.username == "lucas" &&`

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == 'test-gateway'”`

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

`“true &&`

`true &&`

`request.resource == 'gateways' &&`

`request.userInfo.username == "lucas" &&`

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == 'test-gateway’”`

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

`“true &&`

`true &&`

`true &&`

`request.userInfo.username == “lucas” &&`

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == ‘test-gateway’”`

Unified policies through Partial Evaluation

Note that even though the object is unknown, in some cases partial evaluation can evaluate to a concrete true or false:

For example, if user *hacker* executes `kubectl create -f gateway.yaml`

`“true &&`

`true &&`

`true &&`

`false &&`

Expression is always **false**, no matter what the object is! Don't decode, instead **Deny**

`has(request.object.v1) &&`

`request.object.v1.spec.gatewayClassName == 'test-gateway'”`

Kubernetes Authorization

Kubernetes considers an ordered list (chain) of authorizers.

Today, there are three decision modes: **Allow**, **Deny**, or **NoOpinion**.

If **Allow** or **Deny**, enforce decision. If **NoOpinion**, consult next authorizer.

Kubernetes Authorization with Conditions

Kubernetes considers an ordered list (chain) of authorizers.

Today, there are three decision modes: **Allow**, **Deny**, or **NoOpinion**.

If **Allow** or **Deny**, enforce decision. If **NoOpinion**, consult next authorizer.

Conditional Authorization adds a new decision mode: **Conditional**, along with a set of conditions.

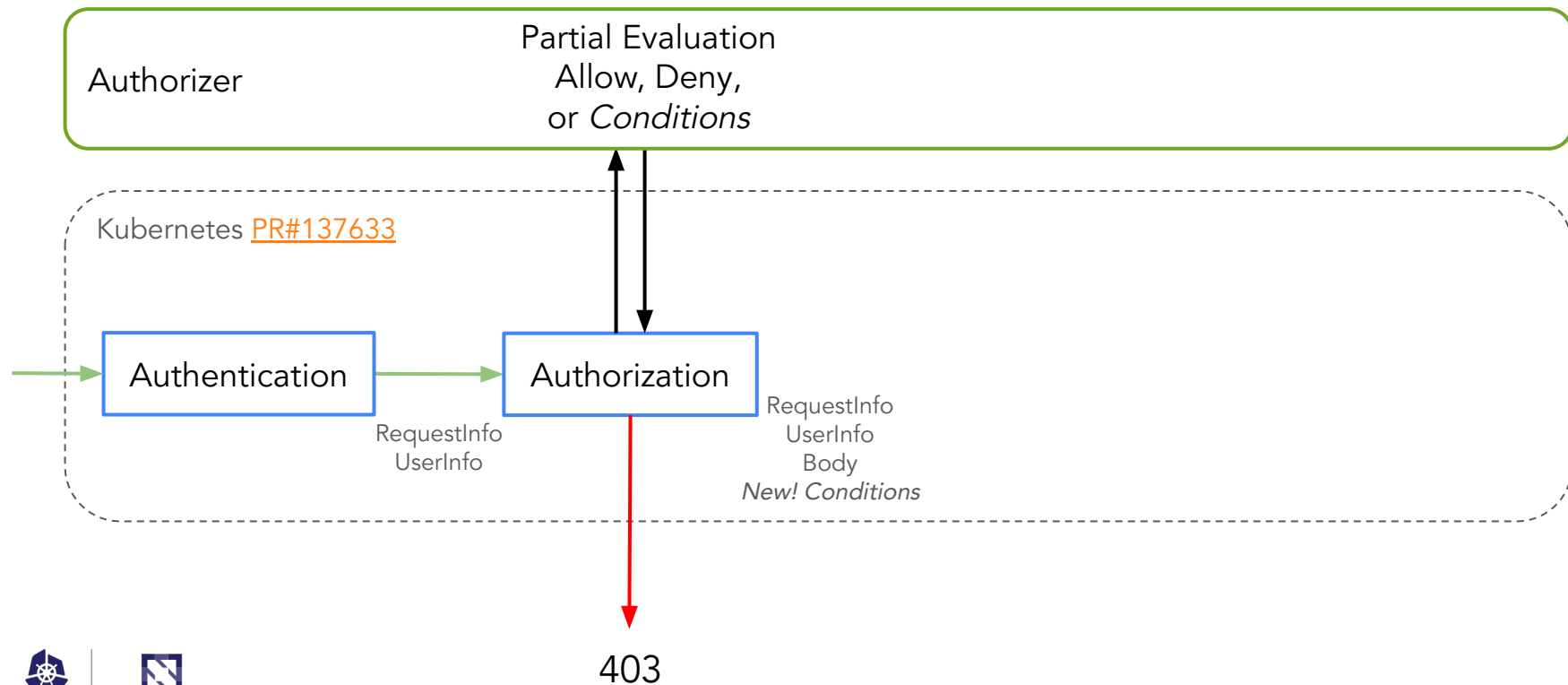
The request can proceed to the admission stage if the decision is **Allow** or **Conditional** with at least one condition that can make the request get authorized.

Conditional Authorization: Data model

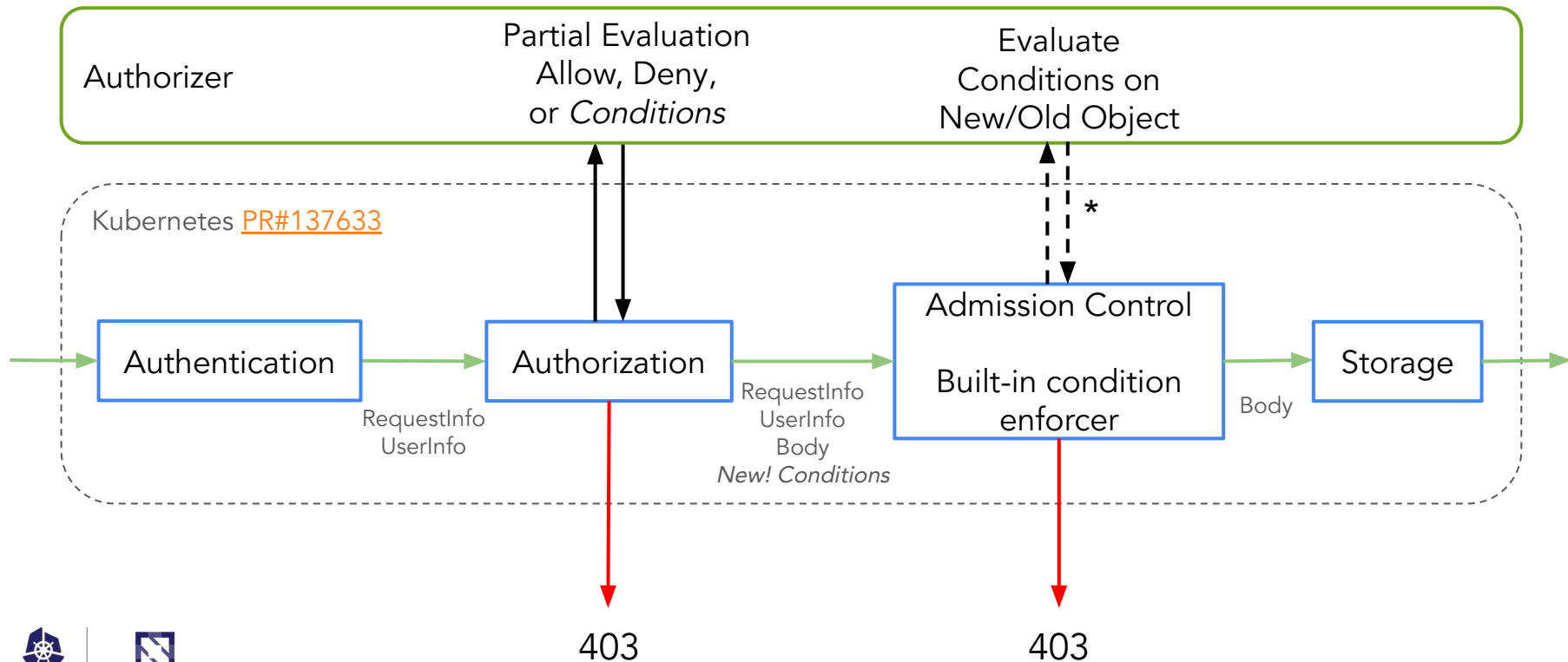
A condition has the following properties:

- **id**: Authorizer-unique identifier (for reason and error messages)
- **effect**: **Allow**, **Deny**, or **NoOpinion**. How to treat “true”.
- **condition**: String with some predicate on the request/stored objects.
- **type**: What language the condition is written in (e.g. CEL, OPA, or Cedar)

Conditional Authorization flow



Conditional Authorization flow



“Let lucas only write TLS secrets”

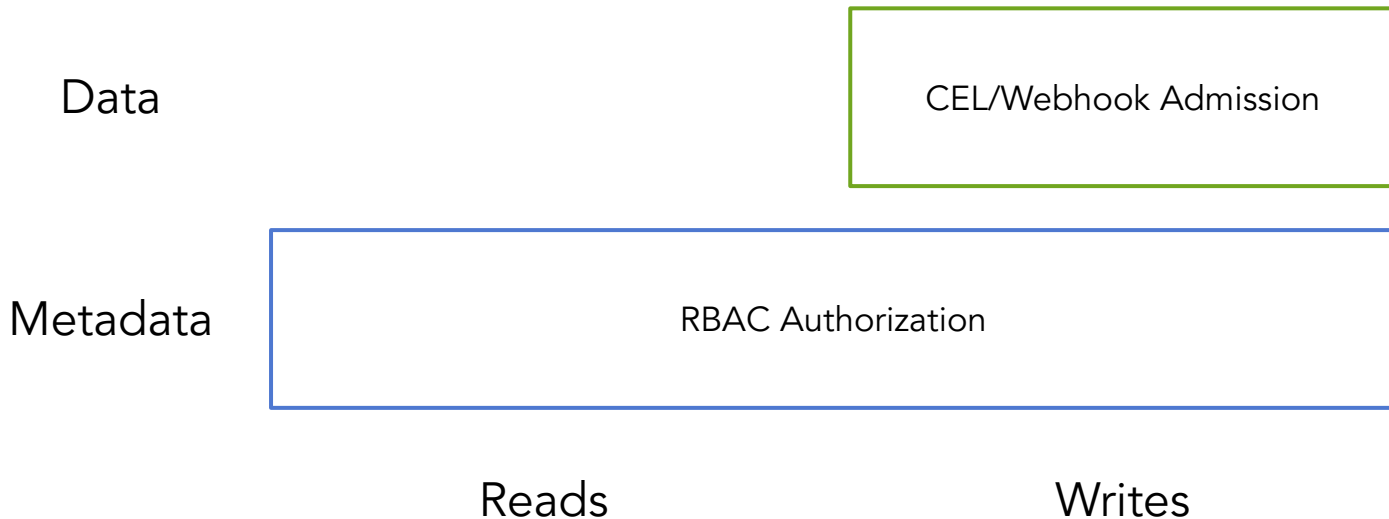
```
zsh
mhausler@6c7e67c5cc57(arm64)$ kubectl create --as lucas -f self-sar-create.json -o yaml
apiVersion: authorization.k8s.io/v1
kind: SelfSubjectAccessReview
metadata: {}
spec:
  resourceAttributes:
    namespace: default
    resource: secrets
    verb: create
    version: v1
status:
  allowed: false
  conditionsChain:
  - conditions:
    - condition: (true && (true && (true && (true && (true && (true && (has(object.type)
      && (object.type = 'kubernetes.io/tls'))))))))
    effect: Allow
    id: policy0
    type: k8s.io/authorization-cel
    reason: conditionally authorized
mhausler@6c7e67c5cc57(arm64)$
```

“Let lucas only read TLS secrets”

```
zsh
mhausler@6c7e67c5cc57(arm64)$ kubectl get secrets --as lucas
Error from server (Forbidden): secrets is forbidden: User "lucas" cannot list resource "secrets" in API group "" in the namespace "default": action list denied by policies [PolicyID("conditional_authorization")]
mhausler@6c7e67c5cc57(arm64)$ kubectl get secrets --as lucas --field-selector type=kubernetes.io/tls -o yaml
apiVersion: v1
items:
- apiVersion: v1
  data:
    password: c3VwZXJzZWNyZXQ=
    username: bWljYWhoYXVzbGVy
  kind: Secret
  metadata:
    creationTimestamp: "2025-11-13T15:35:32Z"
    name: basic-secret
    namespace: default
    resourceVersion: "1012"
    uid: 7e9438ef-1064-4021-9a16-0db5b6d94bf9
  type: kubernetes.io/tls
kind: List
metadata:
  resourceVersion: ""
mhausler@6c7e67c5cc57(arm64)$
```

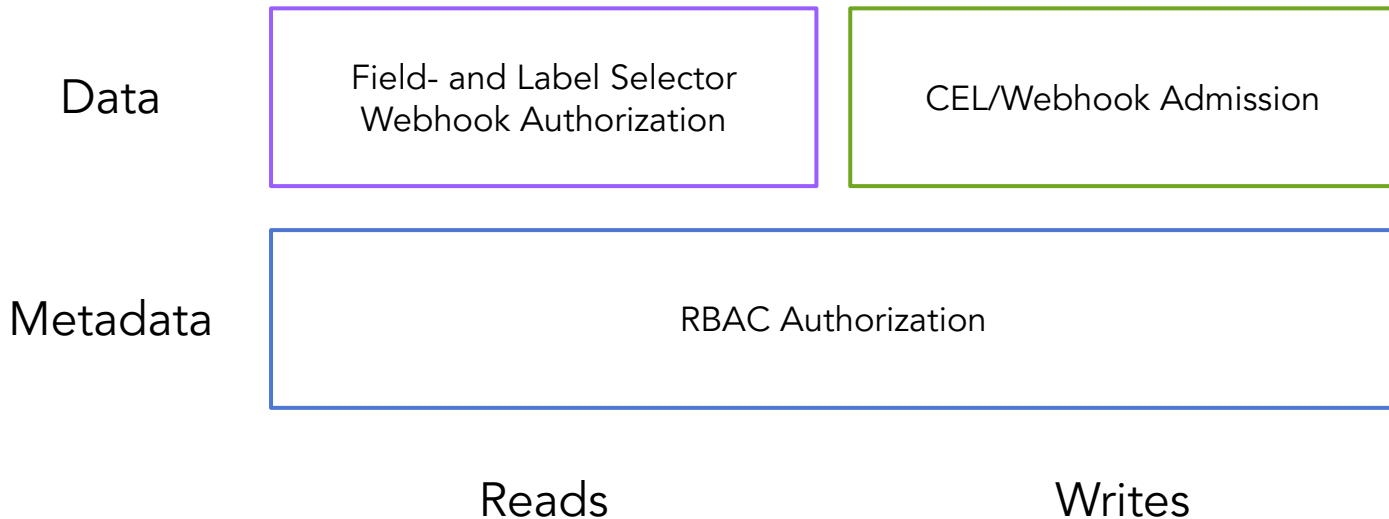
KEP-4601 Authorize with Selectors

Before KEP-4601, there was no way to say “authorize a read only with labelSelector ingress=true”.



KEP-4601 Authorize with Selectors

With KEP-4601, expressing “authorize a read only with labelSelector ingress=true” is possible using a webhook authorizer (not builtin RBAC).



KEP-4601 Authorize with Selectors

Utilizing both KEPs together, makes it possible to express all types of authorization using the same, unified interface for the user.

Data

Metadata

Unified Policy Authoring Interface
For example, using CEL or Cedar

Reads

Writes



KubeCon



CloudNativeCon

Europe 2026

Questions!

**Join the Kubernetes Slack
channel [#sig-auth](#)**



KubeCon



CloudNativeCon

Europe 2026

Thanks!

Contact us on Slack/Github:
@stlaz / @luxas





KubeCon



CloudNativeCon

Europe 2026

