KubeCon | CloudNativeCon

North America 2024

Expanding the Capabilities of Kubernetes Access Control

Lucas Käldström (Upbound) & Jimmy Zelinskie (authzed)

Who are you listening to?

SWE @ Upbound

Formerly at Weaveworks

Former Kubernetes Maintainer (kubeadm)

Former CNCF Ambassador

CPO/Cofounder @ authzed

Formerly at CoreOS, Red Hat

OCI Maintainer

Co-creator SpiceDB, Operator Framework

Our agenda to avoid getting lost in the weeds

1. Speedrun the foundations of authorization
2. Take stock of Kubernetes: what's it doing for authorization
3. Acknowledge the challenges with Kubernetes authorization
4. Propose future solutions to these problems

Let's get the terminology straight

AuthN vs AuthZ

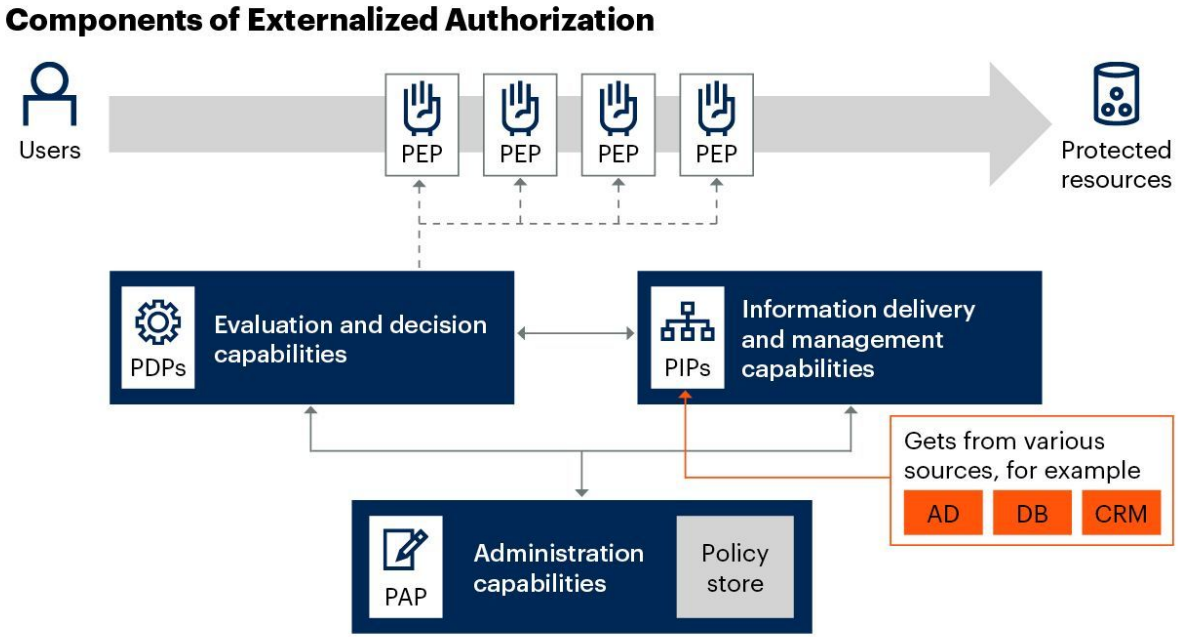
identity vs permissions

Anatomy of a Permissions Check

Can **$PRINCIPAL** perform **$ACTION** on **$RESOURCE**?

Subject	Verb	Object
User	Permission	Entity
Identity	Relation	
	Policy	

What does authorization software look like?



Source: Gartner

AD = active directory; CRM = customer relationship management; DB = database; PAP = policy administration point; PDP = policy decision point; PEP = policy enforcement point; PIP = policy information point

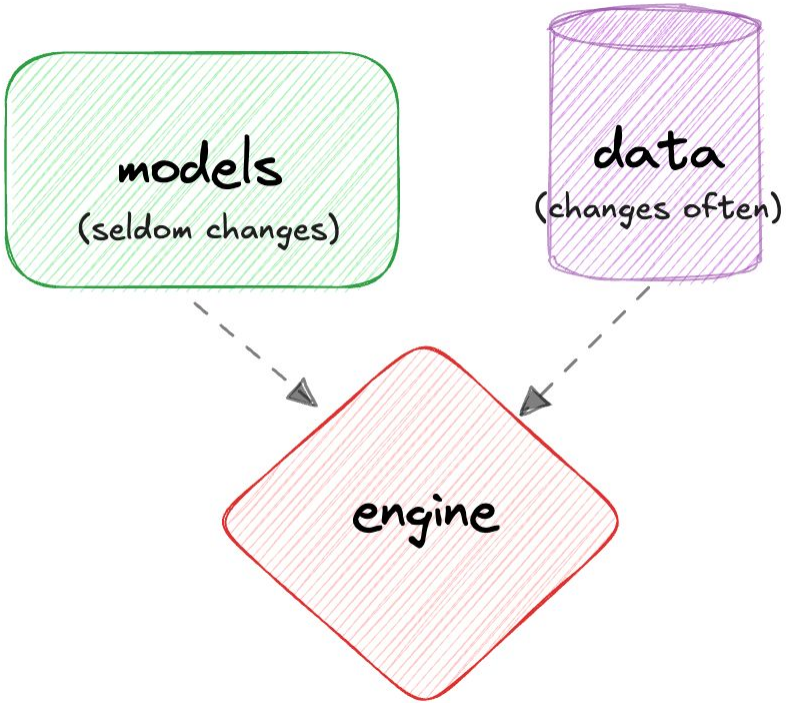
801920_C

Gartner

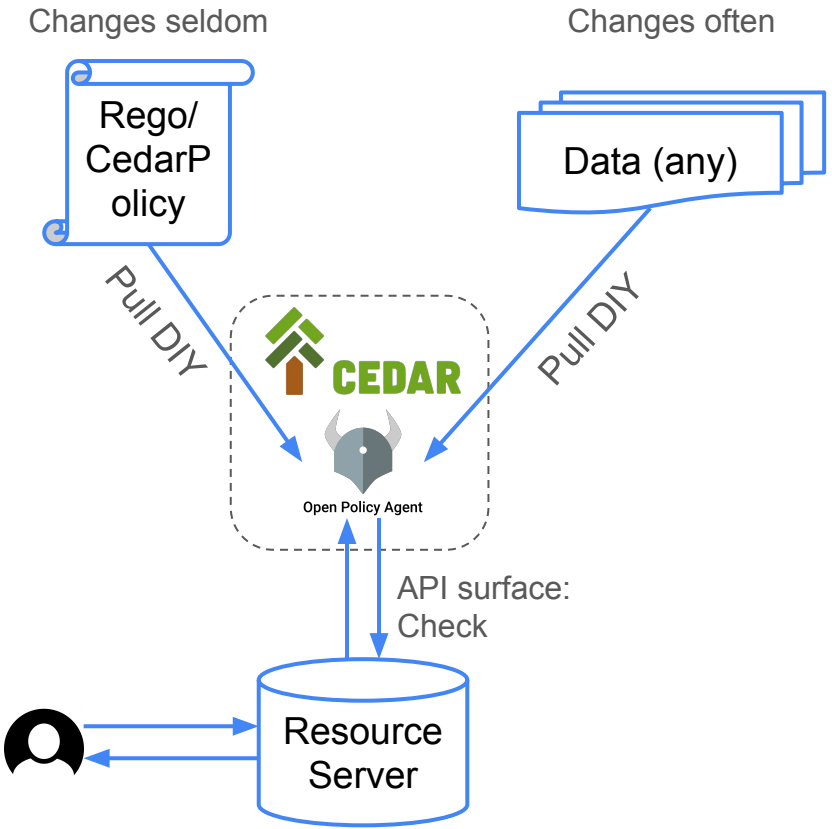
What does authorization software REALLY look like?

Courtroom Metaphor

- Models: laws
- Data: facts or evidence
- Engine: judge or jury



Permissions Engine, Model, and Data: Policy Engine



✅ Flexible policy schema

❌ Datastore for models and data not part of API surface, but DIY

❌ Consistency left as an exercise to the reader, best for fairly “static” policies

OPA is an open source, general-purpose policy engine

Cedar is an open source, general-purpose authorization engine

Data Consistency: Here be dragons

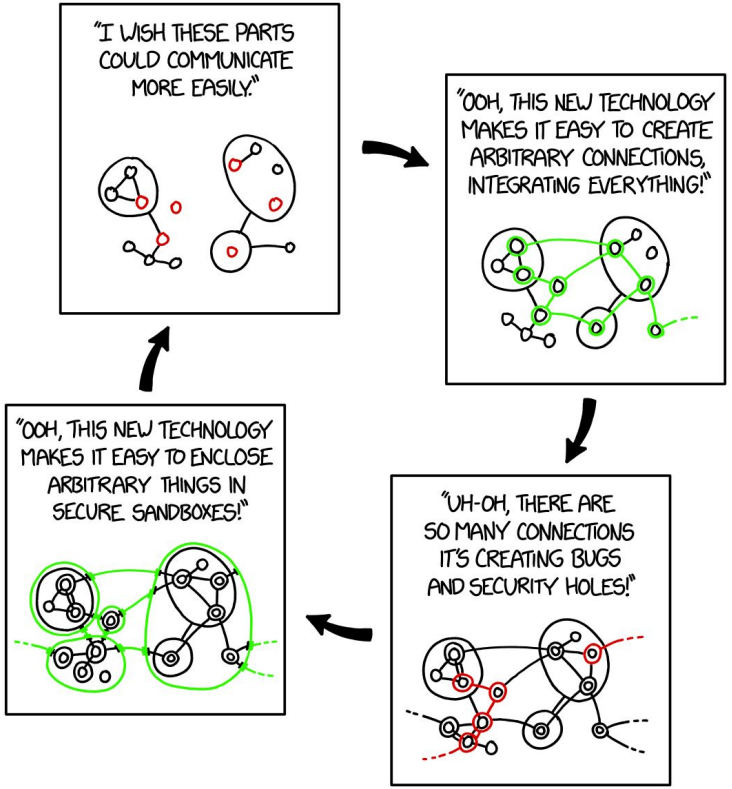
What if the jury applies laws from 1901 and below, although the crime happened 1984?

What if the jury applied laws from 2010 although the crime happened 1984?

What if only facts before the actual crime happened were considered?

What is privilege escalation?

- When a principal is able to perform unintended actions
- Often occurs (by mistake) when traversing **Trust Domains**

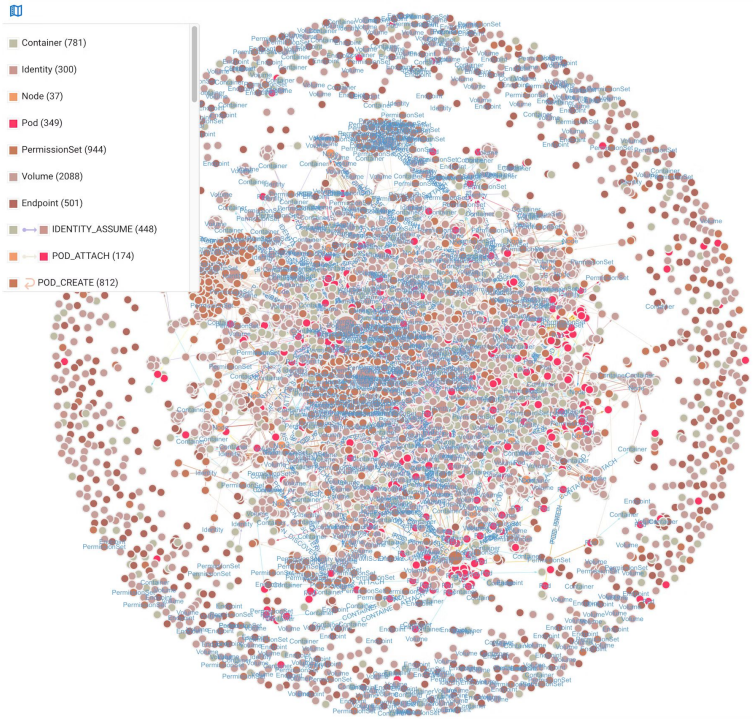


Example: “Privilege escalation” through controllers

User effective permissions are extended by controllers’ permissions in sometimes unexpected ways.

1. User U can only create deployments
2. U creates a deployment referring to a Secret U cannot otherwise read
3. Deployment controller creates a ReplicaSet
4. ReplicaSet controller creates a Pod
5. Kubelet downloads the Secret on U’s behalf and gives to the workload controlled by U

Controllers should be “secure monitors”, but this is hard to implement



Attack paths in a Kubernetes cluster. Image from *KubeHound: Identifying attack paths in Kubernetes clusters* by DataDog.

Research on Authorization Models tl;dr

Concept	Published	Implemented
DAC/MAC	1983	beginning of time
RBAC	1992	impossible to tell
ABAC	2015	at least 1965 (multics)
ReBAC	2019	at least 1998, popular in early 2000s

KubeCon

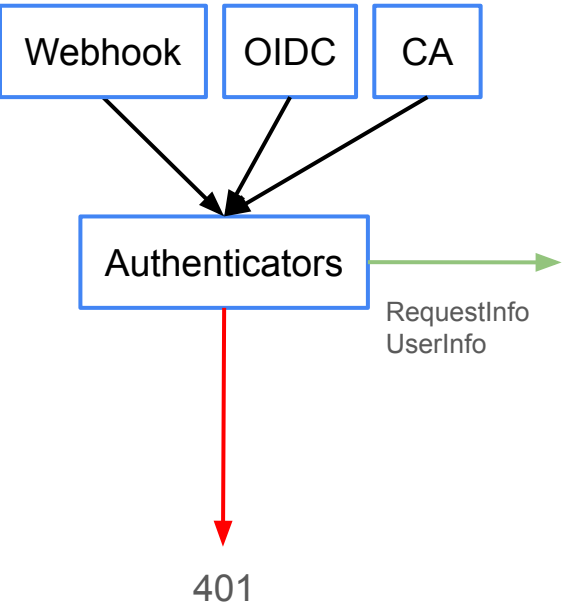
CloudNativeCon

North America 2024

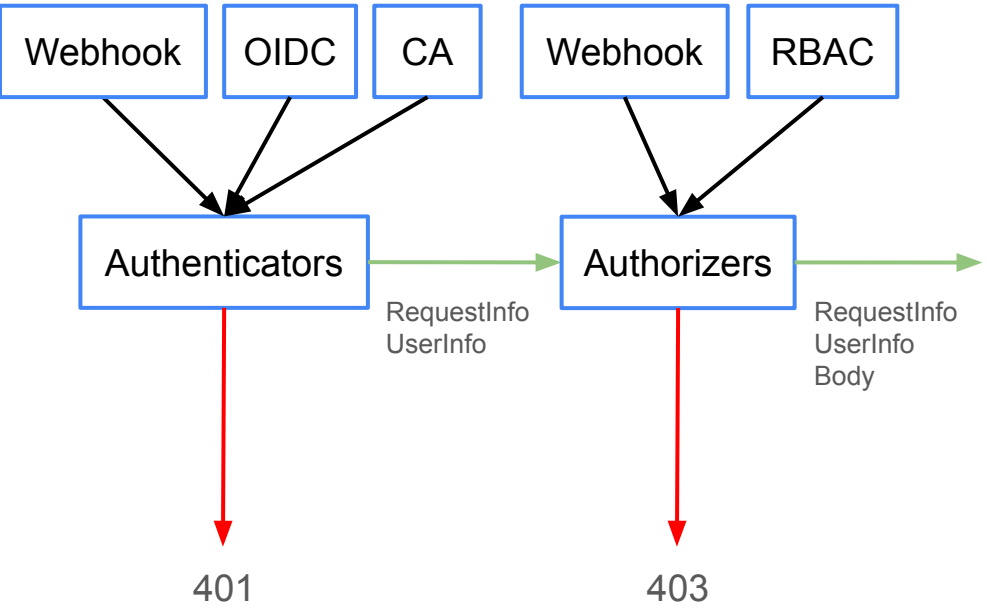
but, what about Kubernetes?

Isn't this a (half) Kubernetes conference?

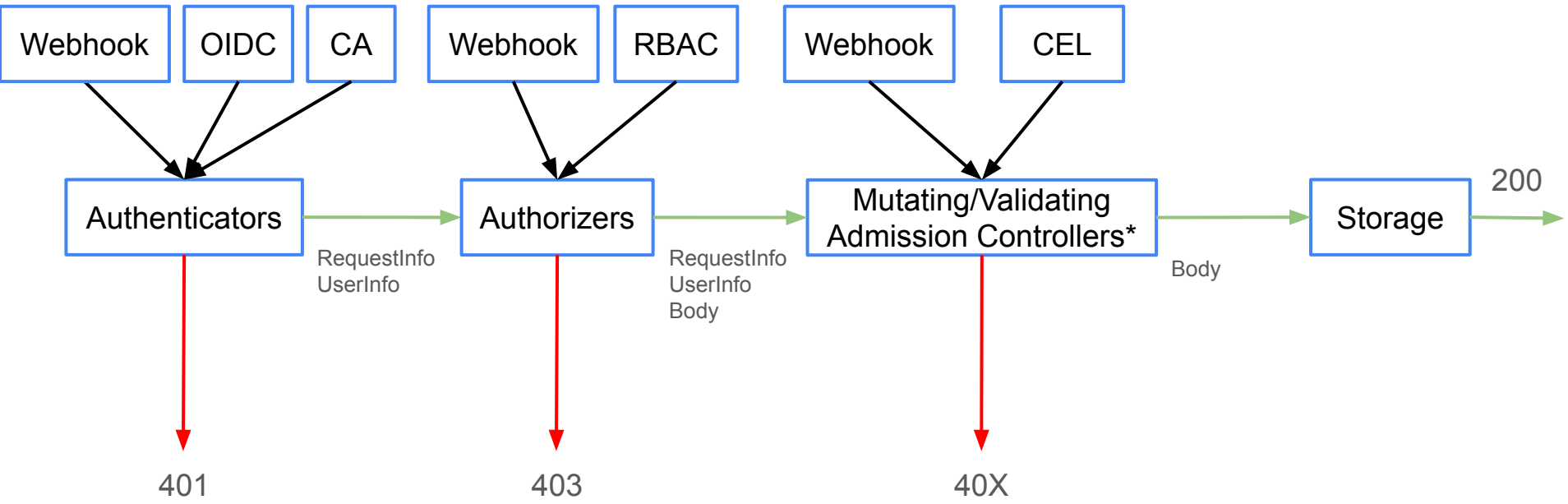
Kubernetes API Server structure



Kubernetes API Server structure



Kubernetes API Server structure



* Admission only for CREATE/UPDATE/PATCH/DELETE

Kubernetes RBAC

✅ “Can principal P perform action A on resource R?” (`SubjectAccessReview`)

✅ Authorization grants stored in the API server as `(Cluster)Role(Binding)`s

→ `(Cluster)Role`: Grants verb permissions to resources matched by **attributes** (apiGroup, resource, namespace, name) (“ABAC”)

→ `(Cluster)RoleBinding`: Binds principals to roles (“RBAC”)

✅ Extensible: custom resources, subresources and verbs supported

✅ Privilege escalation prevention (RBAC editors cannot expand their rights)

❌ Not a “language” in which arbitrary expressions can be written

Common Expression Language (CEL)

Used for extending various part of the policies and functionality of the API server.
Non-turing complete (a feature), can analyze “cost” of expressions.

Kubernetes v1.29+: Structured Auth Configuration

```
kube-apiserver
--oidc-issuer-url=https://foo
--oidc-client-id=my-app
--oidc-username-claim=sub
--oidc-username-prefix=k8s:
--oidc-ca-file=ca.crt
```


```yaml
apiVersion: apiserver.config.k8s.io/v1beta1
kind: AuthenticationConfiguration
jwt:
- issuer:
    url: https://example.com
    certificateAuthority: <CA certificates>
    audiences:
    - my-app
  claimMappings:
    username:
      expression: 'claims.username + ":k8s"'
```

```
kube-apiserver
--authorization-mode=Webhook,
Node,RBAC
--authorization-webhook-confi
g-file=authz.yaml
--authorization-webhook-cache
-authorized-ttl=1m
```


```yaml
apiVersion: apiserver.config.k8s.io/v1beta1
kind: AuthorizationConfiguration
authorizers:
- type: Webhook
  webhook:
    authorizedTTL: 1m
    connectionInfo:
      kubeConfigFile: authz.yaml
  matchConditions:
  - expression:
request.resourceAttributes.namespace !=
'kube-system'
- type: Node
- type: RBAC
```

New in Kubernetes v1.31: Label & Field Authorization

✅ In v1.31, there is the `AuthorizeWithSelectors` alpha feature which adds label and field selectors to the authorization attributes for `list`, `watch` and `deletecollection` requests.

✅ With the feature, the kubelet can only see “its own” Pods (before: all)

✅ One can now give an operator to only list/watch Secrets with a given label

❌ RBAC support for selectors is not planned, requires a webhook

❌ Generic list filtering is deemed out of scope for the time being

Kubernetes Authorization Gotchas

❌ Limited querying capabilities / discovery

→ No “Who can access this resource?” API

→ Limited "What can P do in the system?" (namespace-scoped, only RBAC)

❌ No deny roles (by design, as that would require “tiering”)

❌ No framework to assume extra or drop privileges without DIY impersonation

❌ No builtin time-to-live feature of roles or bindings for temporary access

🎱 Roles inheriting from other roles only supported cluster-wide (aggregation)

🎱 New Enemy Problem (authorizer is eventually consistent/AP)

KubeCon

CloudNativeCon

North America 2024

What about future Kubernetes foundations?

Towards generic control planes, not just containers

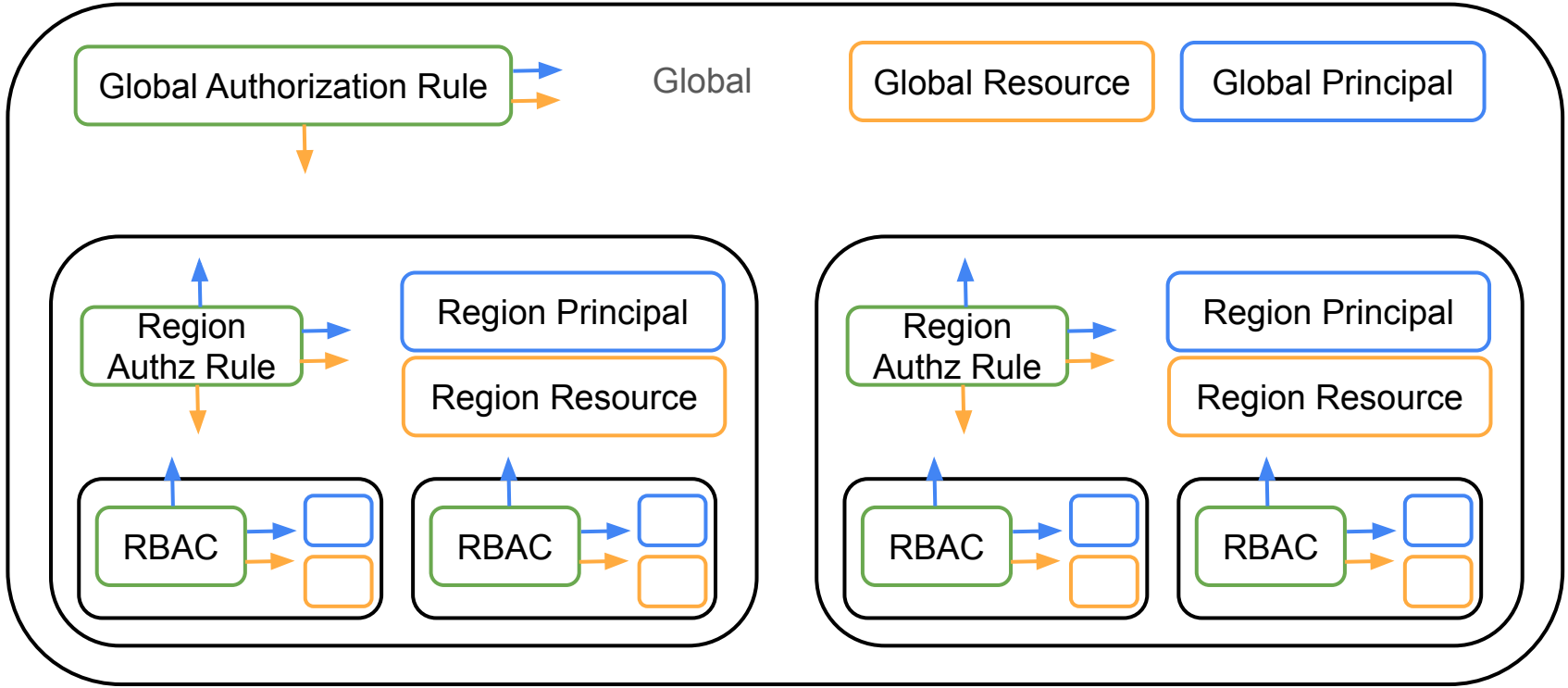
Principles of federated authorization

1. **It all starts from authentication:** Use SPIFFE (or a similar framework) which supports federation, s.t. trust domain A can validate credentials from trust domain B (and possibly vice versa). Map users into globally unique names.

2. **Sync authorization data between trust domains**: We're yet to make a "controller-runtime for authorization data"

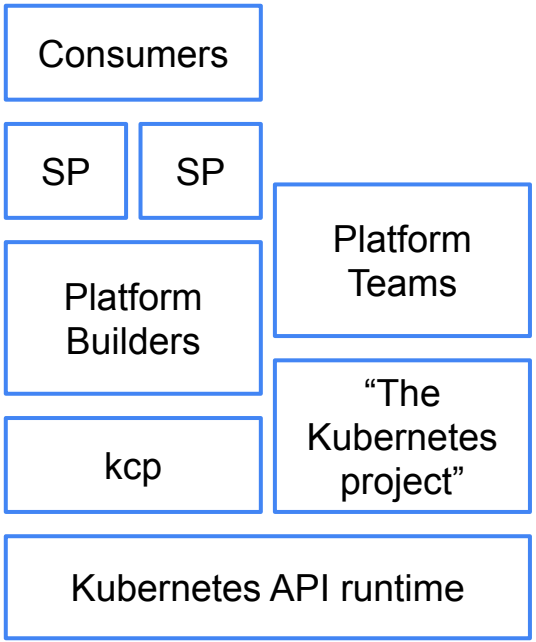
3. **"Bind to principals upwards, resources downwards":** This rule helps enforce isolation, while allowing inheritance.

“Bind to principals upwards, resources downwards”



KCP: Kubernetes re-imagined platforms

- Kubernetes was designed for one persona (ops), but, today, we need at least 3: platform owner, service provider, and API consumers
- KCP is a framework for building multi-tenant k8s control plane experiences that can be managed by a central platform owner, easily extended by service providers, and usable by consumers
- Kubernetes authorization primitives cannot model these new personas or any other future workflows
  - Experimentation with Warrants: kcp-dev/kcp#3156
    - => seteuid instead of setuid
  - Permissions inheritance with scoping across trust domains



KubeCon

CloudNativeCon

North America 2024

If we're going to move forward, what other alternatives exist?

Remember this slide?

Concept	Published	Implemented
DAC/MAC	1983	beginning of time
RBAC	1992	impossible to tell
ABAC	2015	at least 1965 (multics)
ReBAC	2019	at least 1998, popular in early 2000s

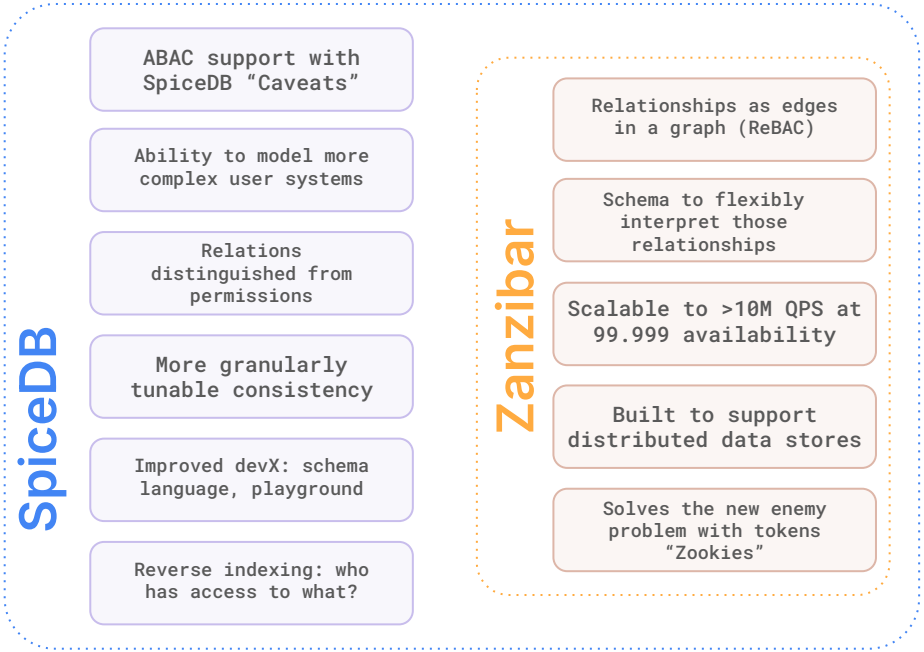
Zanzibar: Google's Consistent, Global Authorization System

Ruoming Pang, Ramón Cáceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner*, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, Mengzhi Wang

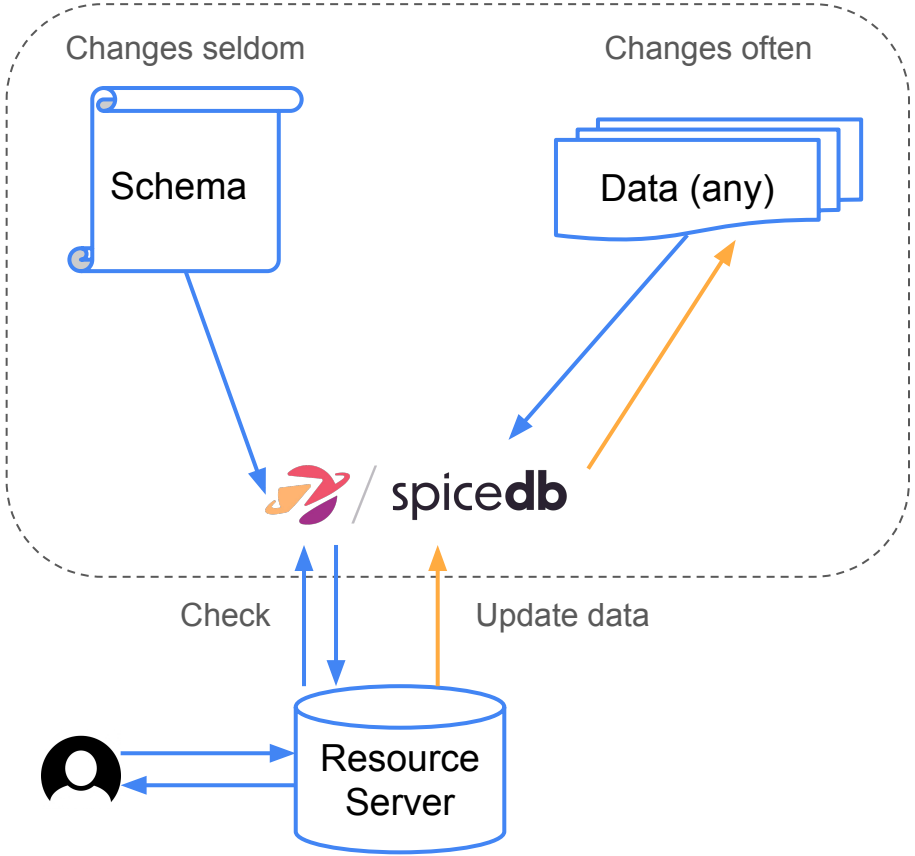
Google, LLC; Humu, Inc.; Carbon, Inc.

https://zanzibar.tech

- Database specifically for authz data
- Most mature OSS project using ReBAC principles
- Superset of Google's Zanzibar
- Operated as a centralized service (often by platform teams) shared across a product suite/microservice architecture
- Alternative: Okta's OpenFGA



Permissions Engine, Model, and Data: ReBAC



✅ Flexible permissions model
✅ Datastore for permissions data and model
✅ Handles consistency & safe caching on a request-basis

KubeCon | CloudNativeCon

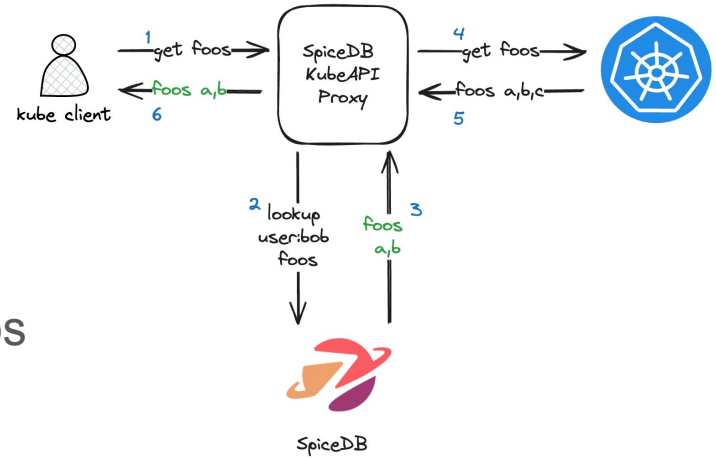
North America 2024

What could this look like integrated with Kubernetes?

⇒ Map Kube data & policies to generic engines

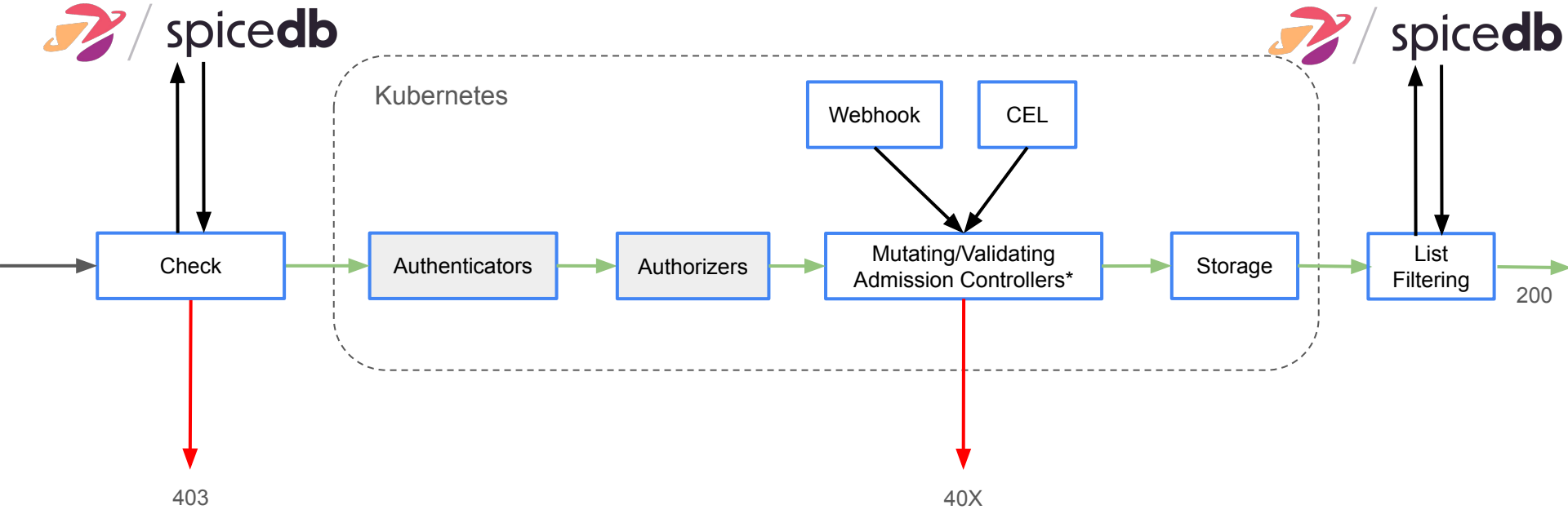
SpiceDB KubeAPI Proxy

Experimental k8s API proxy to enforce custom, fine-grained authorization



- ✅ Filters lists of Kubernetes resources
- ✅ Customizable permissions model
- ✅ “What can I see?”, “Who can see this?” lookups
- ✅ No synchronization required
- ✅ Easy to delegate a user’s permission
- ❌ Must be the only Kube API server endpoint

SpiceDB KubeAPI Proxy Architecture



Cedar-Kubernetes Authorizer

Experimental webhook that combines authorization and validating admission using one unified language (Cedar) instead of two (RBAC, CEL)

✅ Non-turing complete => Analyzable SMT
✅ Deny roles with policy tiering
🎱 Stores only permission models, no data
🎱 Experimental lookups
❌ No loops (not analyzable)

✅ Fine-grained impersonation & label authz
✅ Uses Kubernetes CRDs as policy data store

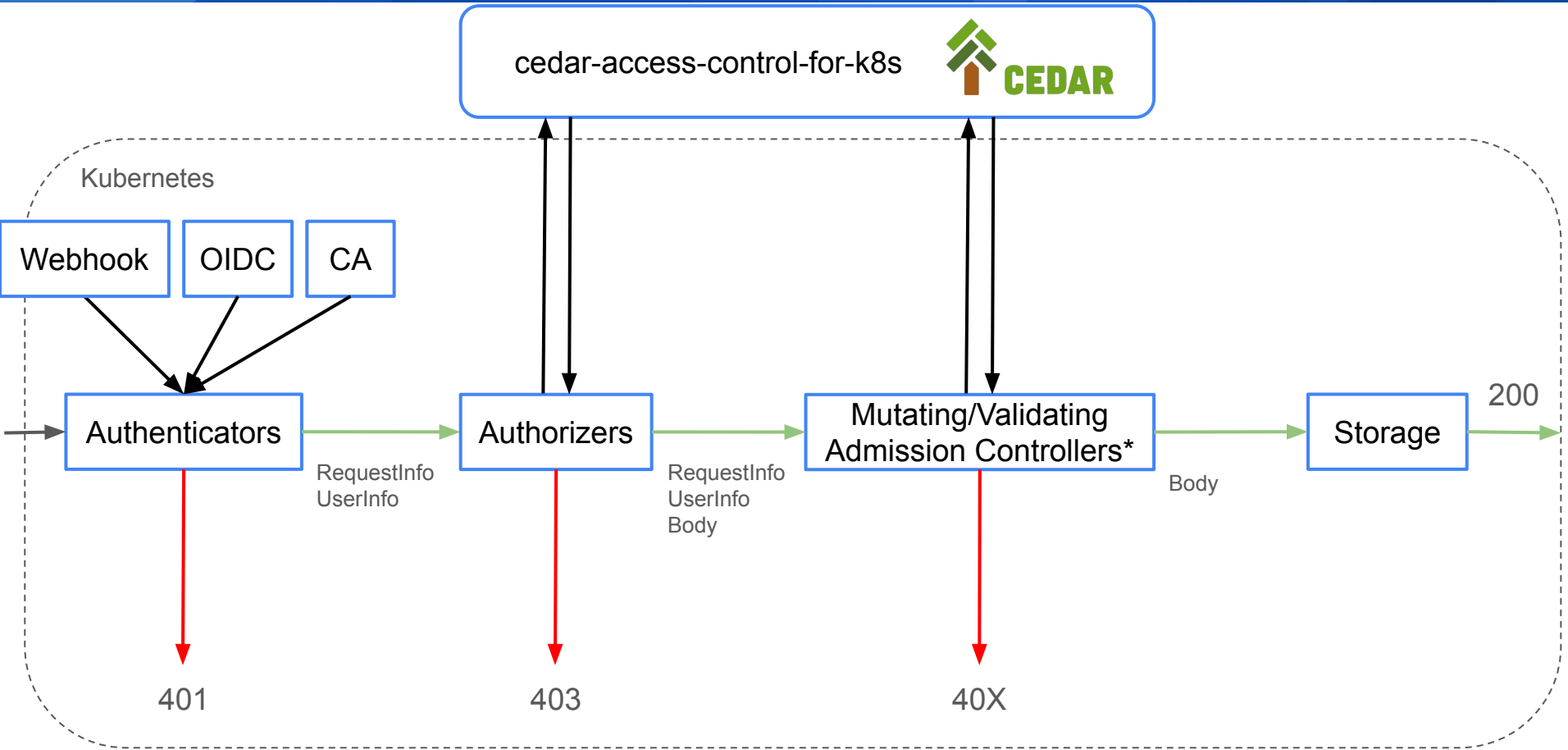
Implementation of `AuthorizeWithSelectors`

```
forbid (
    principal is k8s::User,
    action in [k8s::Action::"list", k8s::Action::"watch"],
    resource is k8s::Resource
) when {
    principal in k8s::Group::"requires-labels"
} unless {
    resource has labelSelector &&
    resource.labelSelector.contains({
        "key": "owner",
        "operator": "=",
        "values": [principal.name]
    })
};
```

Require an owner label to be set

```
forbid (
    principal is k8s::User,
    action in [
        k8s::admission::Action::"create",
        k8s::admission::Action::"update"],
    resource
) when {
    principal in k8s::Group::"requires-labels"
} unless {
    resource has metadata &&
    resource.metadata has labels &&
    resource.metadata.labels.contains({"key": "owner", "value": principal.name})
};
```

Cedar-Kubernetes Authorizer Architecture



Build more controllers?

One could build controllers for the following, regardless of “backend”, that:

- Pull off-cluster data (e.g. “global” policies) into local authorization context
- Delete Roles / RoleBindings after a defined TTL
- Implement “role inheritance” (aggregation) also inside of namespaces
- Based on object-to-object relationships (e.g. Pod refers to a ConfigMap), craft “computed” roles based on some logic
- Implement “delegation” through copying RoleBindings
- Implement “drop privs” thru impersonation into a “user copy” with less privs

Authorize references?

Recall this example of a user creating a Deployment referring to a Secret they don't have access to directly (privilege escalation through references).

One can deny the "create deployment" call at admission time using this CEL expression.

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
 name: "refcheck-core.pods"
spec:
 matchConstraints:
   resourceRules:
   - apiGroups:   [""]
     operations:  ["CREATE", "UPDATE"]
     resources:   ["pods"]
 validations:
   - expression: |
      !has(object.spec) || !has(object.spec.containers) ||
      object.spec.containers.all(container,
        !has(container.envFrom) || container.envFrom.all(envFrom,
          !has(envFrom.secretRef) || !has(envFrom.secretRef.name) ||
          authorizer.group("").
            resource("secrets").
            namespace(namespaceObject.metadata.name).
            name(envFrom.secretRef.name).
            check("get").allowed()))
```

KubeCon | CloudNativeCon

North America 2024

Kubernetes is just the first step! We've got the whole ecosystem to fix, too!

Conclusion

1. “I wish all of these things integrated, *securely*”
   – every person looking at the CNCF landscape

   - Example: Argo has its own RBAC system, but uses its “root” privileges when talking to clusters. By integrating a general-purpose authorization model, one might integrate more deeply with other projects.

2. Authorization is complex. Don't roll your own without evaluating existing tools.

3. Can we get to a “docker” moment where common patterns and practices emerge with generic solutions?


KubeCon
CloudNativeCon
North America 2024
Please, join the communities
around the mentioned projects!
Thank you!