SEAN CORCORAN · ERIN WALTER

RBAC as a Platform Capability

Applying platform engineering to Kubernetes access

11 NOVEMBER 2026



ABOUT US

Erin Walter

SENIOR SOFTWARE ENGINEER, PLAYSTATION - 6.9 YEARS

My experience spans infrastructure and Kubernetes with a focus on improved Developer Experience and Productivity Tooling. My team empowers engineers for excellence, making golden paths accessible, secure, and streamlined.

Hobbies

- Board Games
- Bass Guitar
- Hiking & Backpacking



ABOUT US

Sean Corcoran

SENIOR SOFTWARE ENGINEER, PLAYSTATION - 4.9 YEARS

Roughly a decade of experience working in Kubernetes, and about 7 years in “Developer tooling” oriented roles.

Hobbies

- Gaming (MMOs & FPS)
- Golf & Baseball
- Dealing with my cats and their nonsense



How we managed Namespaces and access before

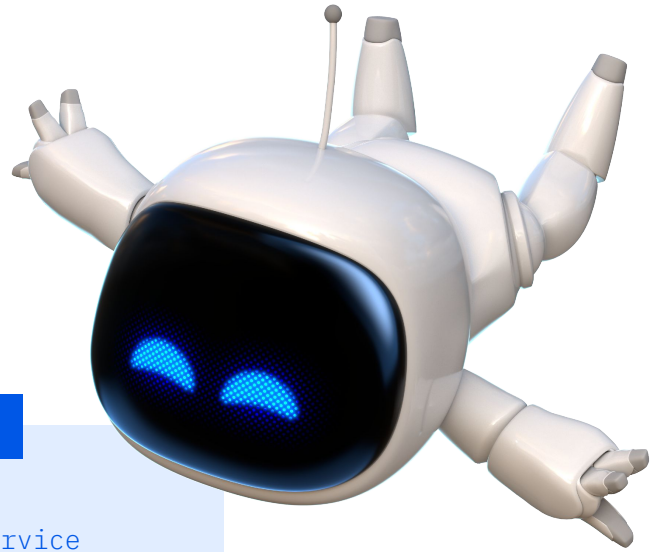
In-house tool we call “Maeslant”

- CronJob that runs on each individual Kubernetes Cluster
- Cluster and Namespace configuration files define what resources needed to be created



The combined YAML format

A namespace change and an access change shared the same review and apply path.



01 One file carried three jobs

Namespace metadata

RoleBinding subjects for users and ServiceAccounts

Arbitrary Kubernetes Resource application

namespaces/my-service.yaml

```
annotations:
  description: Basic API Service
resources:
  - limit-range-ns
  - pod-quota-ns
rolebindings:
  cr/edit:
    - raw:jenkins
    - github:team-dev
```

Growth exposed the coupling

New clusters also needed CronJob installation and repository onboarding.

01 More inputs

More clusters, namespaces, teams, and people

New employee RBAC source

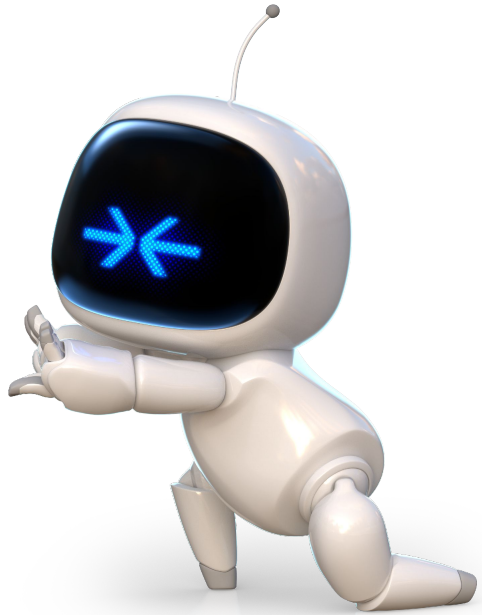
New service onboarding flows

02 Where it hurt

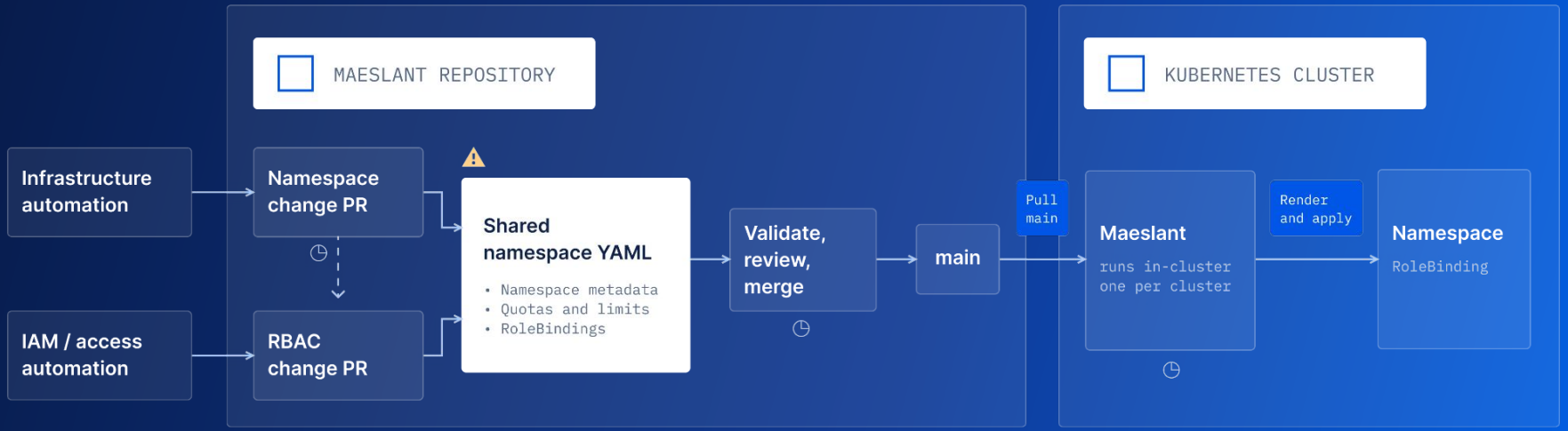
Concurrent pull requests collided

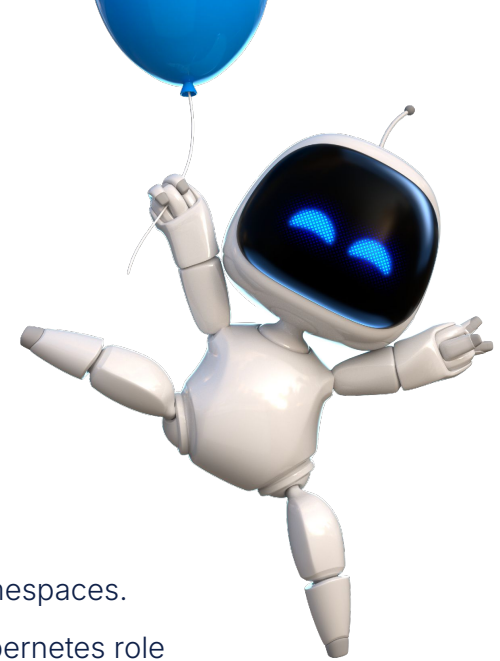
Validation and merge conflicts remained

Onboarding had to sequence namespace and access changes



In practice, the flow looked like





The new pattern

Utilize the Kubernetes controllers to expose these capabilities to clients while abstracting away infrastructure complexity.

01 Hub cluster

Clients create Namespace and NamespaceBinding resources.

Controllers resolve targets and report status.

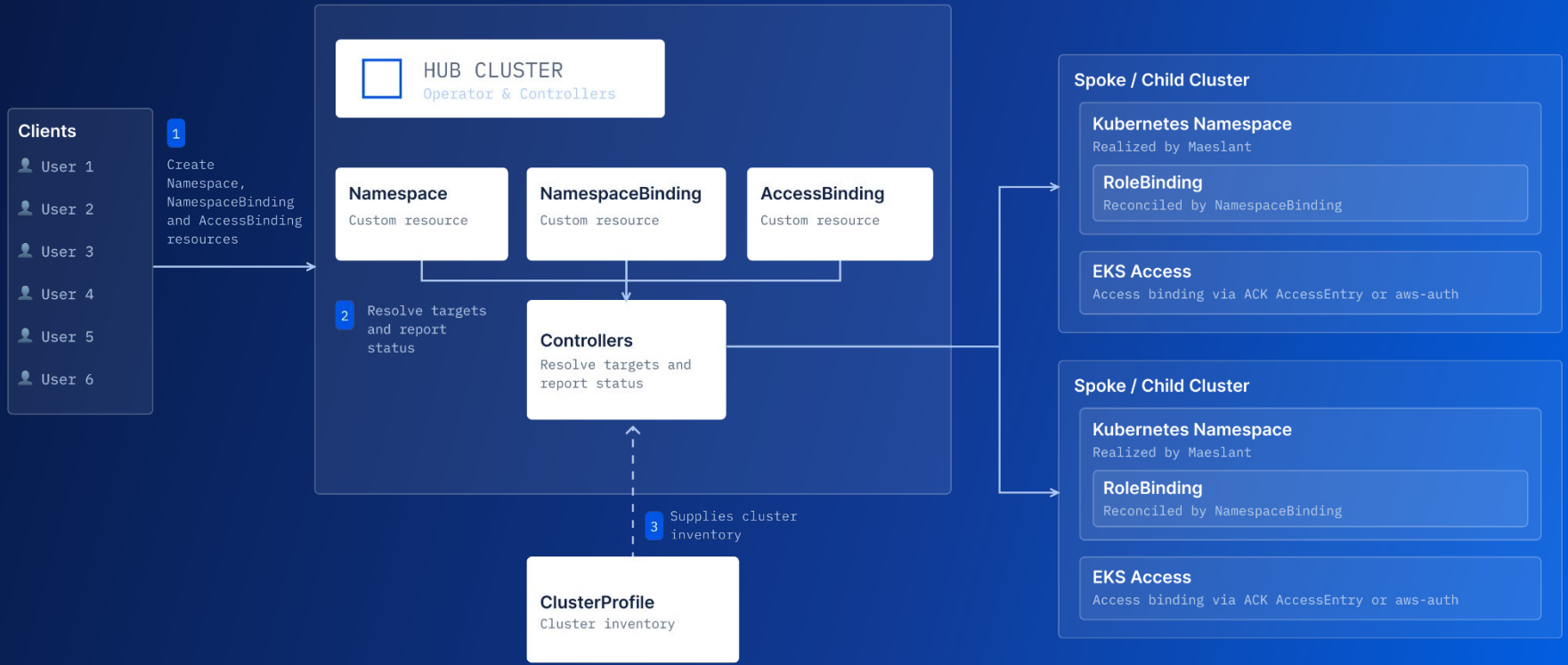
ClusterProfile supplies cluster inventory.

02 Child clusters

Controller realizes Kubernetes namespaces.

NamespaceBinding reconciles Kubernetes role bindings. For IAM subjects, it can compose an AccessEntry for EKS identity mappings.

One controller deployment manages resources across target clusters.


Clients
User 1
User 2
User 3
User 4
User 5
User 6
1
Create Namespace, NamespaceBinding and AccessBinding resources
HUB CLUSTER
Operator & Controllers
Namespace
Custom resource
NamespaceBinding
Custom resource
AccessBinding
Custom resource
2
Resolve targets and report status
Controllers
Resolve targets and report status
3
Supplies cluster inventory
ClusterProfile
Cluster inventory
Spoke / Child Cluster
Kubernetes Namespace
Realized by Maeslant
RoleBinding
Reconciled by NamespaceBinding
EKS Access
Access binding via ACK AccessEntry or aws-auth
Spoke / Child Cluster
Kubernetes Namespace
Realized by Maeslant
RoleBinding
Reconciled by NamespaceBinding
EKS Access
Access binding via ACK AccessEntry or aws-auth

What clients create

01 Namespace

One desired Kubernetes namespace in one named child cluster.

Can also declare ownership and app labels.

02 NamespaceBinding

Declares a role and subjects for referenced namespaces.

The controller creates role bindings on their target clusters.



From user intent to the desired resource

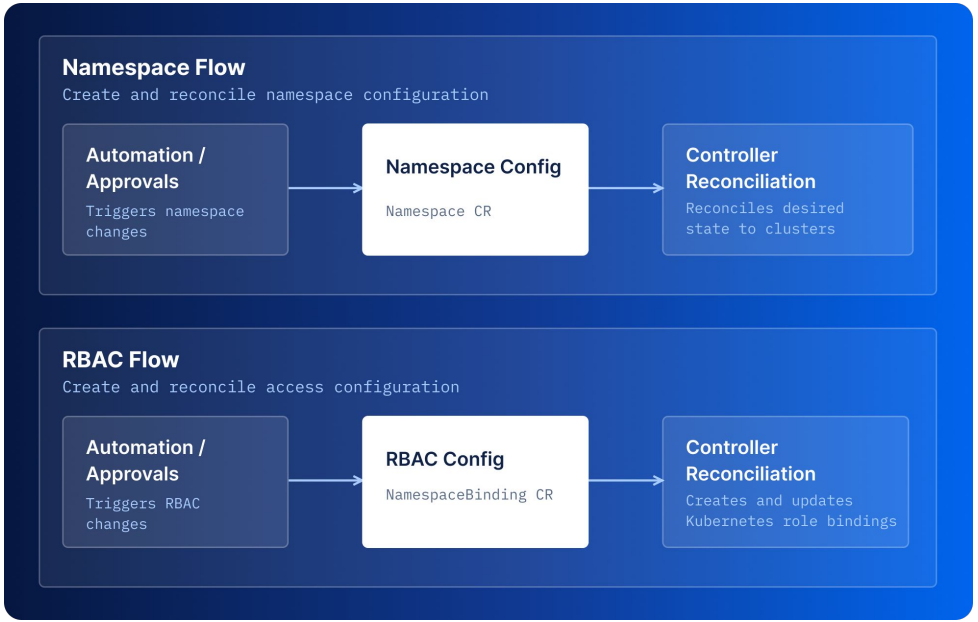
01 Client creates a Namespace or NamespaceBinding in the hub

02 Controller resolves the referenced ClusterProfile

03 Controller applies its owned target objects

04 Controller reports downstream Resource status and retries failed reconciliations

Each resource has its own reconcile and cleanup path.



Namespace

The hub custom resource is distinct from the child Kubernetes Namespace.

01 Operator action

Resolve the target cluster

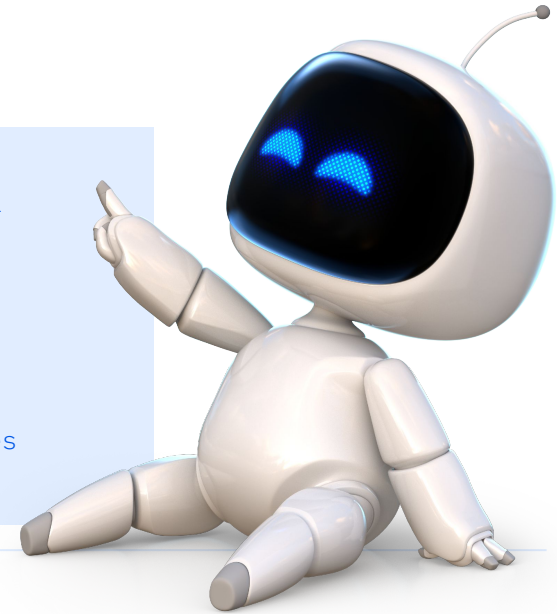
Create or update the child namespace

Apply labels and annotations

Report status on the hub resource

Client-created resource

```
apiVersion: engine.sie.sony.com/v1
kind: Namespace
metadata:
  name: team-dev
  namespace: platform-services
spec:
  name: team-dev
  type: Custom
  clusterProfileRef:
    name: cluster-a
    namespace: uks-cluster-profiles
  labels:
    owner: demo-team
```



NamespaceBinding

01 Operator action

Resolve the referenced hub Namespace resource.

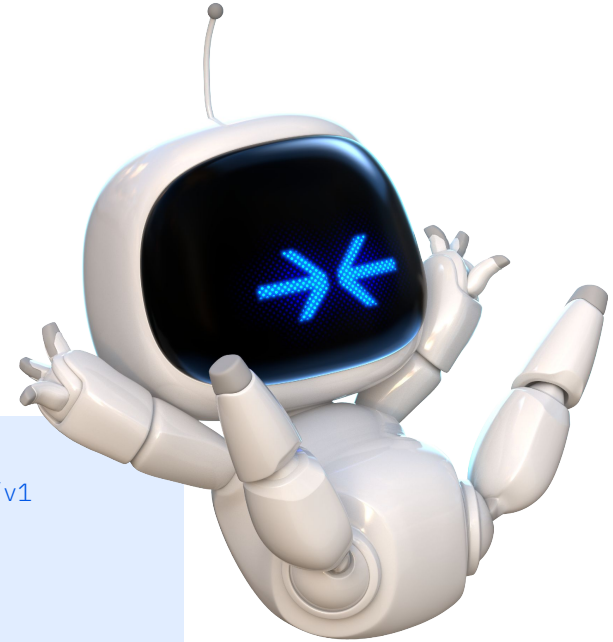
Find its target cluster and namespace.

Create or update the child RoleBinding.

Report status and retry when needed.

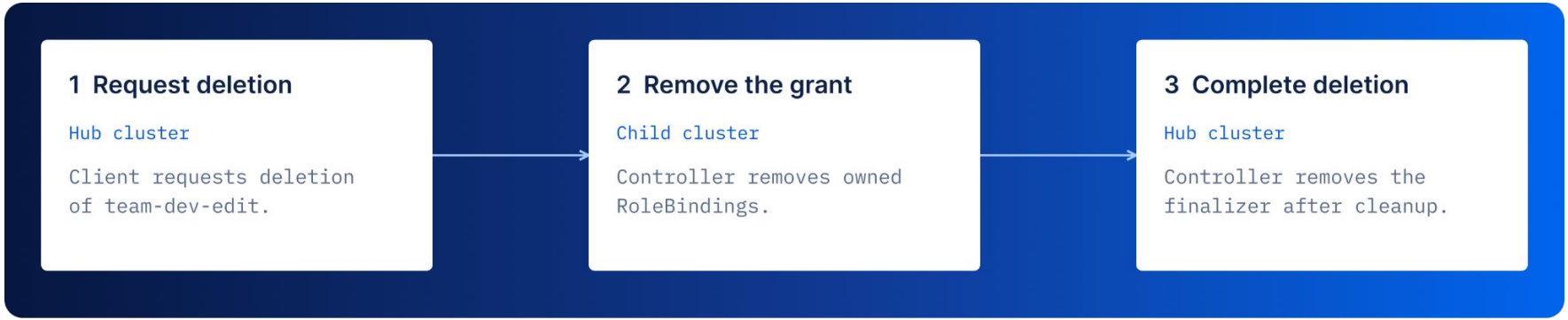
Client-created resource

```
apiVersion: engine.sie.sony.com/v1
kind: NamespaceBinding
metadata:
  name: team-dev-edit
  namespace: platform-services
spec:
  namespaceRefs:
    - name: team-dev
      namespace: platform-services
  binding:
    type: RoleBinding
    roleKind: ClusterRole
    roleName: edit
    subjects:
      - kind: Group
        name: team-deployers
```



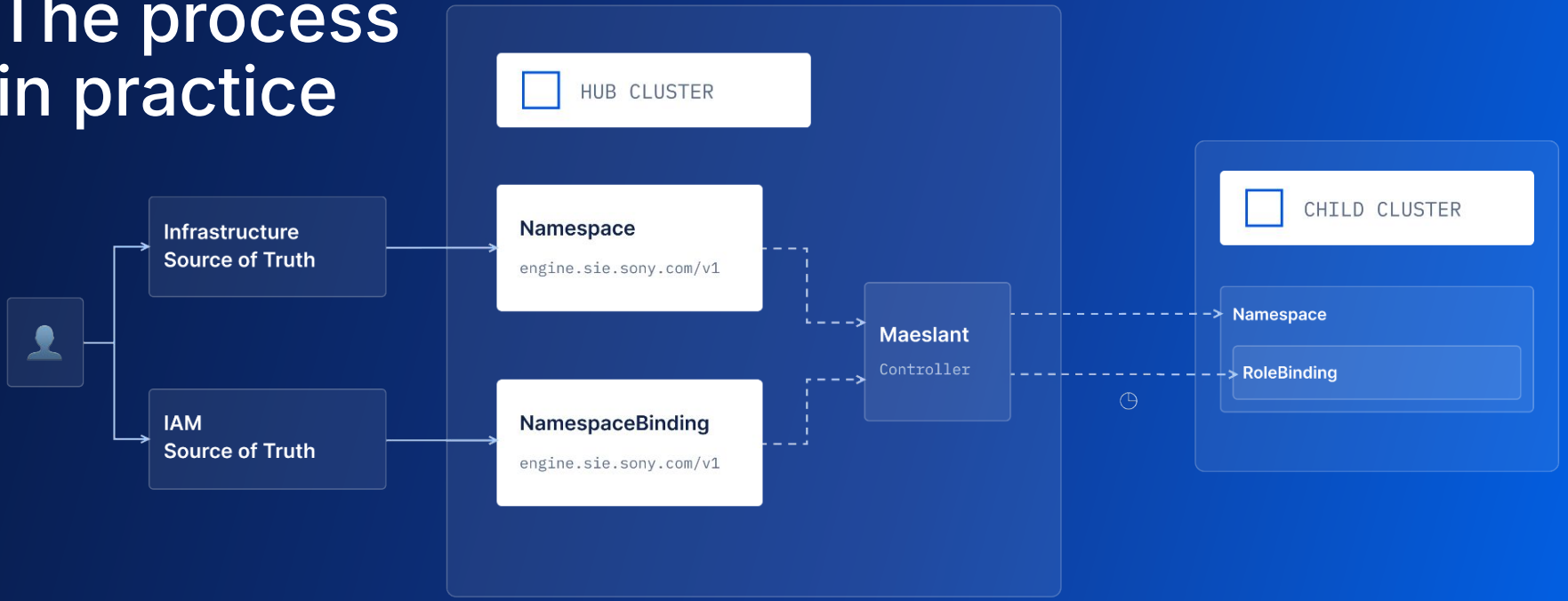
Removing a permission grant

Example: deleting NamespaceBinding team dev-edit



If the target is unavailable, cleanup waits and the controller retries.
The Namespace and unrelated RoleBindings remain in place.

The process in practice



What we've gained:

Independent requests, visible progress

Clients manage their own resources in the hub.

Controllers handle dependencies.

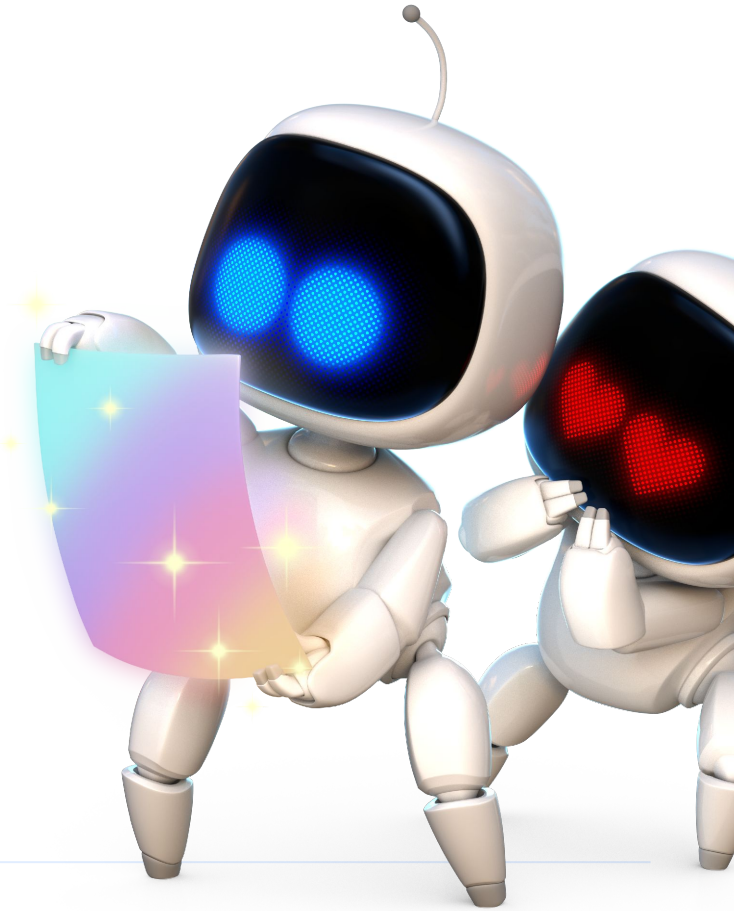
Reconciliation handles:

- NamespaceBindings submitted before their referenced Namespace is ready
- Temporary cluster connection failures

The controller reports status and retries reconciliation.

Monitor reconciliation metrics and alert on persistent failures.

Each request remains visible as a resource with status.



What changed for teams

	WORKFLOW	REPOSITORY + CRONJOB	PLATFORM API
Permission change	Edit shared namespace YAML through a pull request.	Submit a separate or directly edit an existing NamespaceBinding.
Dependency handling	Caller sequences namespace and access changes.	Controller waits for readiness and retries reconciliation.
Progress visibility	Inspect pull requests and scheduled apply results.	Read resource status and conditions.

Operational responsibility

The platform team maintains controller health, cluster connectivity, and reconciliation alerts.

METRICS - PRODUCTION CLUSTER RUNTIMES

Managed namespaces	Avg cronjob runtime (min)	Interval (min)
7	3.22	30
8	3.69	5
8	3.89	5
10	3.58	5
76	15.96	30
89	12.24	30
95	15.73	30
196	30.49	5
282	43.94	30
345	59.33	5

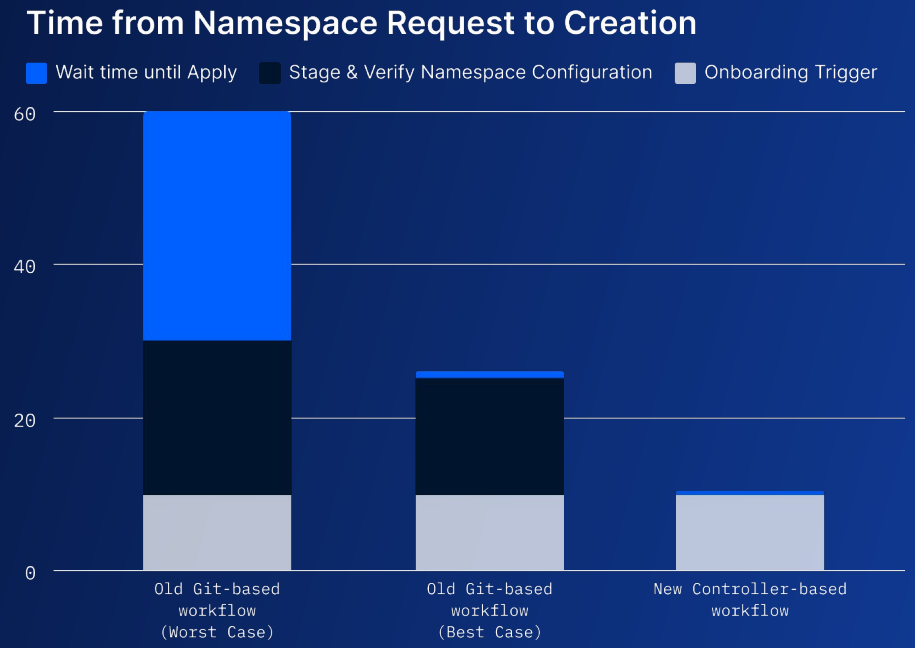
BENCHMARK RUNTIMES - 100MS LATENCY INCLUDED

Namespaces	Avg creation (min)	Avg update (min)
30	0.31	0.23
150	1.21	1.06
300	2.3	2.12

ONBOARDING TIME → NAMESPACE CREATION (CURRENT STATE)

Onboarding trigger	PR merged (min)	Cron Job window (min)
10 min	~20 min	~30 min

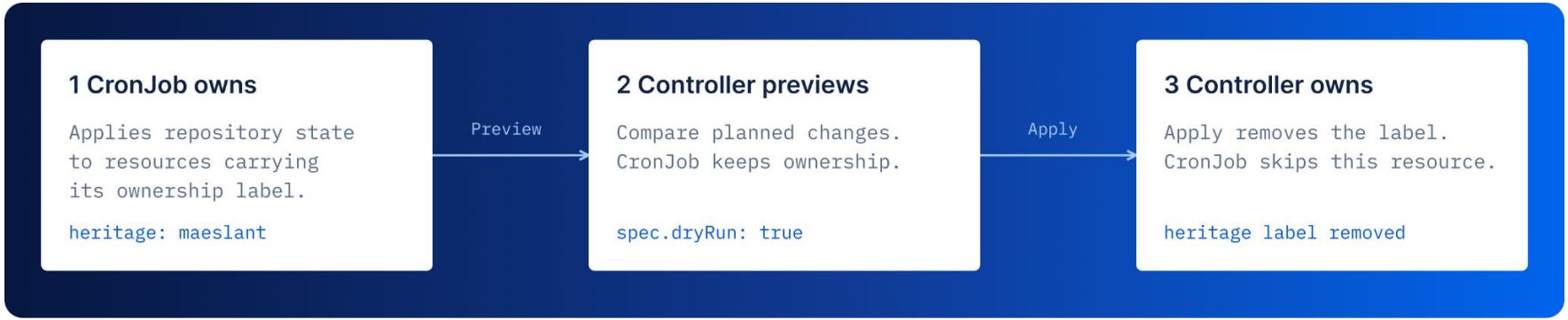
The time we got back



- Not waiting around for an infrequently run cronjob saves a lot of time!
- Moving verifications directly into the controller much faster than launching tests as part of Continuous Integration
- Significant optimization moved the workflow's bottleneck

Migration through ownership handoff

One writer per target resource



Return ownership if needed

Enable dry-run for affected custom resources and confirm apply has stopped.
Verify repository state, restore heritage=maeslant, and confirm the next CronJob run.

Patterns to take home

01 **Separate namespace lifecycle from permission intent**
Give each concern its own resource and owner.

02 **Let controllers handle dependencies and retries**
Callers submit intent independently while controllers manage dependencies.

03 **Make status part of the API**
Give clients a clear view of progress and failures.

04 **Define ownership for apply and cleanup**
Scope changes to owned objects and keep one writer per target resource.



Thank you!



RBAC as a Platform Capability: Applying Platform Engineering to Kubernetes Access