

Mercedes-Benz
Tech Innovation



Securing 900 Kubernetes Clusters Without Pod Security Policies (PSP)

by Tobias Giese & Tjark Rasche
Software Engineers

Mercedes-Benz



Who am I?



Tobias Giese

Software Engineer

- Certified Kubernetes Security Specialist
- Involved into Kubernetes since v1.7
- Former maintainer: Cluster API
Provider OpenStack
- Vinyl music enthusiast
- Loves to optimize his home office desk setup

Working at Mercedes-Benz Tech Innovation since 2017

Who am I?



Tjark Rasche

Software Engineer

- Working with Kubernetes since v1.13
- Initiator of the Kubernetes Meetup Ulm
- Loves making and teaching music

Working at Mercedes-Benz Tech Innovation since 2023

We are Mercedes-Benz Tech Innovation!

#TECH-SAVVY

#FUTURE-ORIENTED

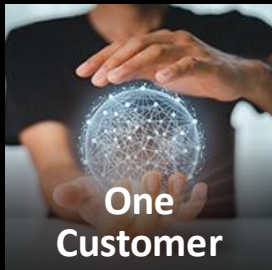
#PASSIONATE

We are a 100 % subsidiary and strategic partner exclusively for Mercedes-Benz.

100 | 25

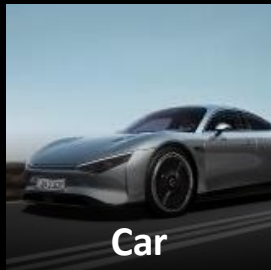
We have been developing digital products and software solutions for over 25 years.

Projects & Products



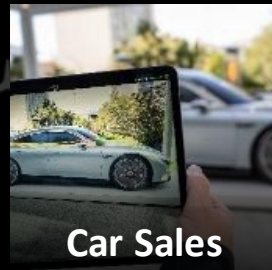
One Customer

We lead the management of customer data across all channels and sales levels at Mercedes-Benz.



Car

We drive the development of car software embedded, onboard and offboard.



Car Sales

We design the luxury customer experience and drive vertical functional integration for car sales.



Car After-Sales

We push digital targeting to engage customers and leverage connected car capabilities.



Technologies & Security

We foster cyber security and develop latest technologies as the basis for new products and innovations.



Daniel Geisel

CEO

> 1,400

Employees
(August 2023)

Where we are

Ulm (headquarters),
Stuttgart, Berlin, Karlsruhe

Worldwide Group network



Fully Managed Kubernetes Platform

On Prem & Public Cloud



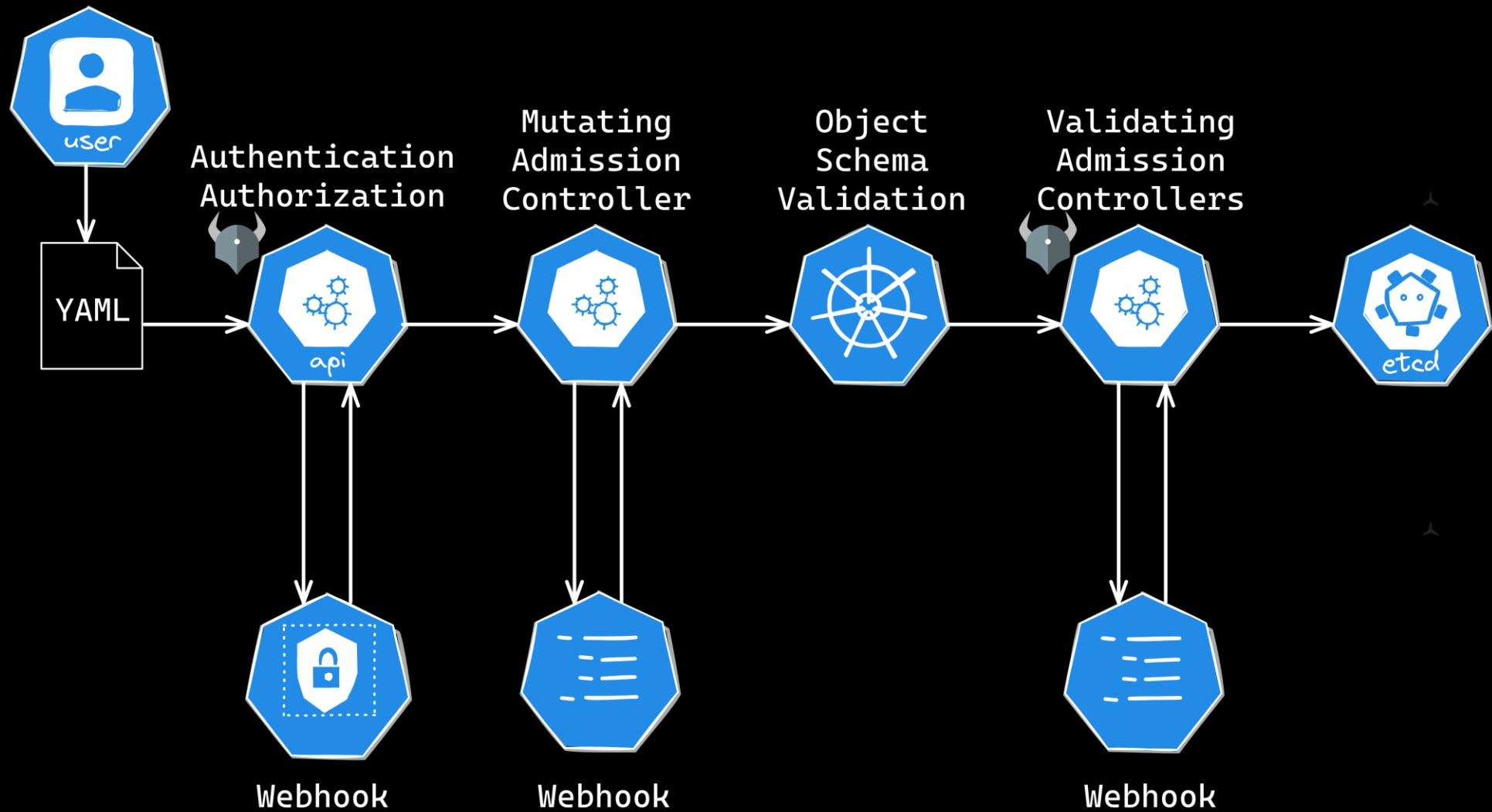
What Managed Means

- Customer does not have to think about Kubernetes operations at all
- Customer does not have SSH access to Kubernetes nodes
- Security lock down (what we are talking about here)
 - No root containers
 - No host path volumes
 - No access to system Namespaces
 - ...

The Status Quo on Kubernetes v1.24

- PSP used for enforcing Pod Security during runtime
- Open Policy Agent running as static Pod manifest
- Handling validating webhook
- Handling authorization webhook

The How and Why of OPA in a Static Pod



Destination: Validating Admission Policies



Pod Security Policies

All in one: defaulting and validation. But confusing, bug-prone, opt-in and beta status.

Kyverno, Gatekeeper

Kubernetes Native, great user experience, great tooling.

Performance Issues

Validating Webhook delays API requests up to 11s.

Validating Admission Policy

Declarative, fast, highly configurable.
Available with v1.26.

Pod Security Standards



Great feature

Secure by default
Enforcing best practices
Not opt in
– If you choose a pod security standard changes in Pod specifications of future releases will automatically be considered
Easy to use and understand



Very unflexible

e.g. NET_ADMIN
Disabled in baseline already
↪ More flexible solution needed

What the Solution Should Provide



Allow flexible definition
of policies for pods (and
ideally other resources as
well)

"Pollute" customer
experience as little as
possible

Allow mutation of
resources (to overwrite
insecure defaults from
Kubernetes)

Why Do We Need Mutation?

```
1 // AddNoNewPrivileges returns if we should add the no_new_privs option.
2 func AddNoNewPrivileges(sc *v1.SecurityContext) bool {
3     if sc == nil {
4         return false
5     }
6
7     // handle the case where the user did not set the default and did not explicitly set allowPrivilegeEscalation
8     if sc.AllowPrivilegeEscalation == nil {
9         return false
10    }
11
12    // handle the case where defaultAllowPrivilegeEscalation is false or the user explicitly set
13    // allowPrivilegeEscalation to true/false
14    return !*sc.AllowPrivilegeEscalation
15 }
```

<https://github.com/kubernetes/kubernetes/blob/release-1.26/pkg/securitycontext/util.go#L199-L211>

OPA/Gatekeeper vs. Kyverno



PRO

can mutate

not Kubernetes native
as used in current setup

CON

very complex
(own language)

when used with
Gatekeeper pollutes
customer cluster



PRO

can mutate

Kubernetes native via CRD

CON

"pollutes" customer cluster
→ may not be able to use
Kyverno on it's own

Our Kyverno Experience

Great User Experience

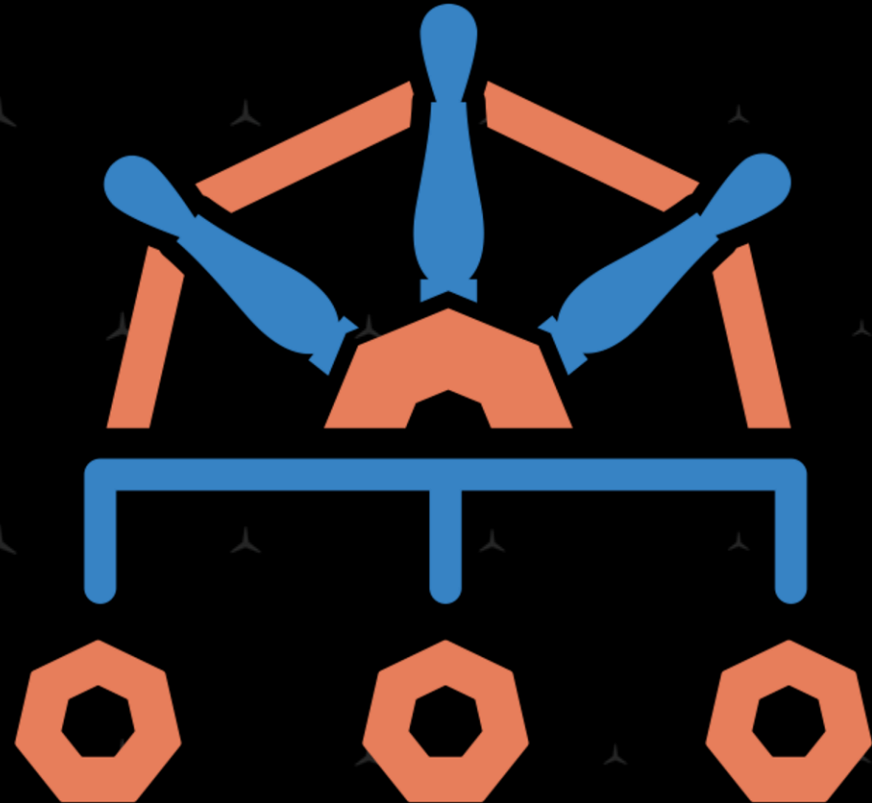
Good readability and maintainability

All Policies implemented in a few days

All E2E Tests passing

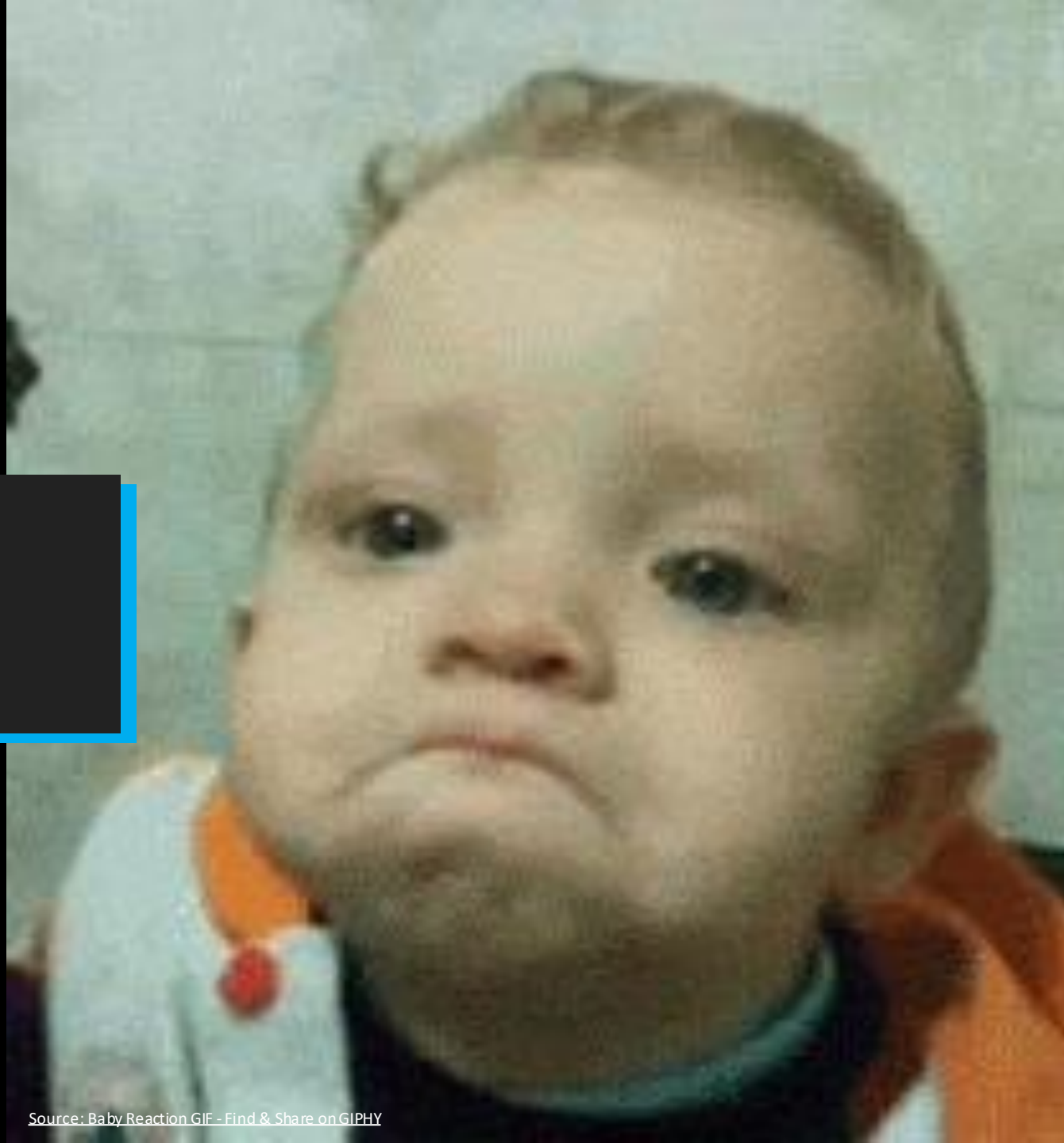


We are ready to go!



The First Day of Test Deployments

EVERYTHING IS BROKEN 🤖

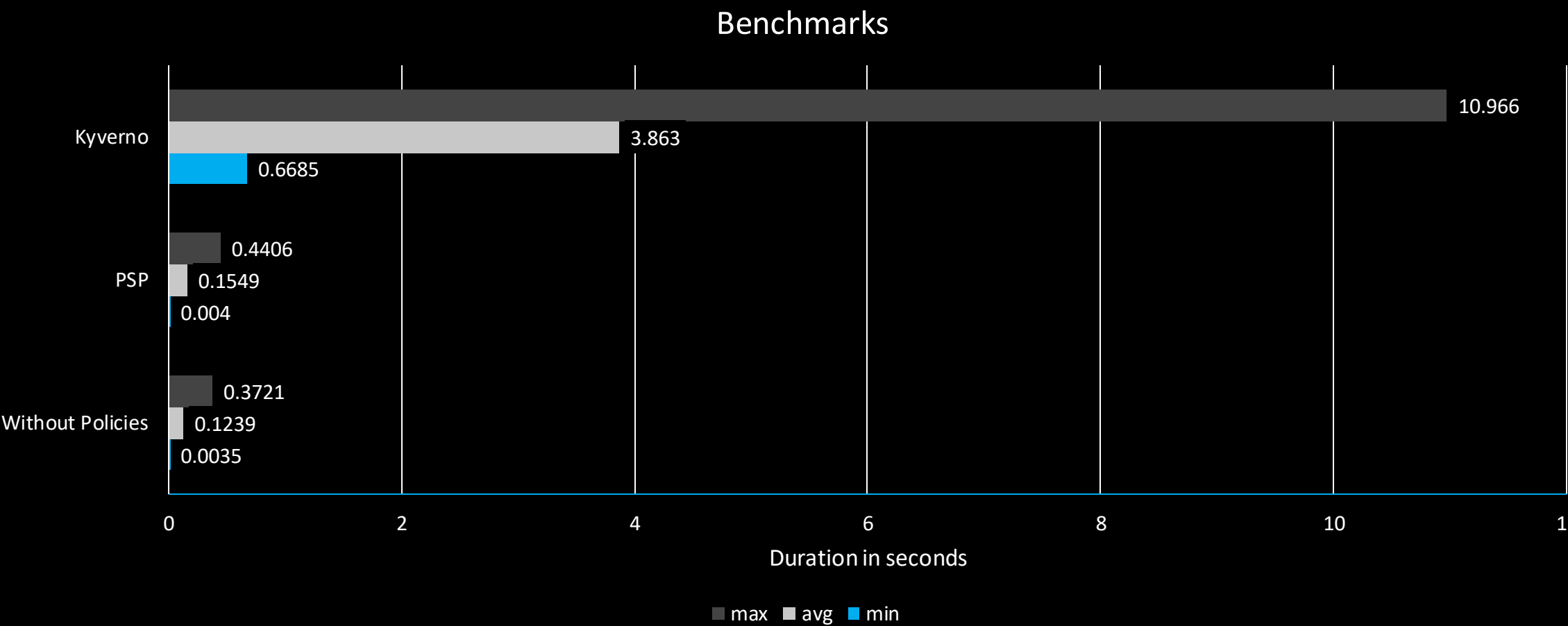


Source: Baby Reaction GIF - Find & Share on GIPHY

So We Started Benchmarking...

- Benchmarking with k6 from Grafana Labs
- Local test cluster
 - Kind
 - 2 CPUs
- With following k6 parameters
 - Virtual users (vus): 100
 - Iterations: 1000
- Simple Pod creation
 - Valid Pod manifest
 - Only as dry-run

Comparison to PSP and No Policies



The Battle Plan

OPA

Implement all policies in OPA
using existing setup
(without Gatekeeper
→ no easy mutation)

Custom Controller

Implement a Proof-of-Concept
custom controller to evaluate
performance

Benchmarking Suite

Improve K6 load testing suite
to better match real world
scenarios

We Are Running Out of Time



v1.26

is already around the corner

We will Do It in One Go

v1.24

v1.25

v1.26



Validating Admission Policies



Validating Admission Policies



Allow custom
admission logic

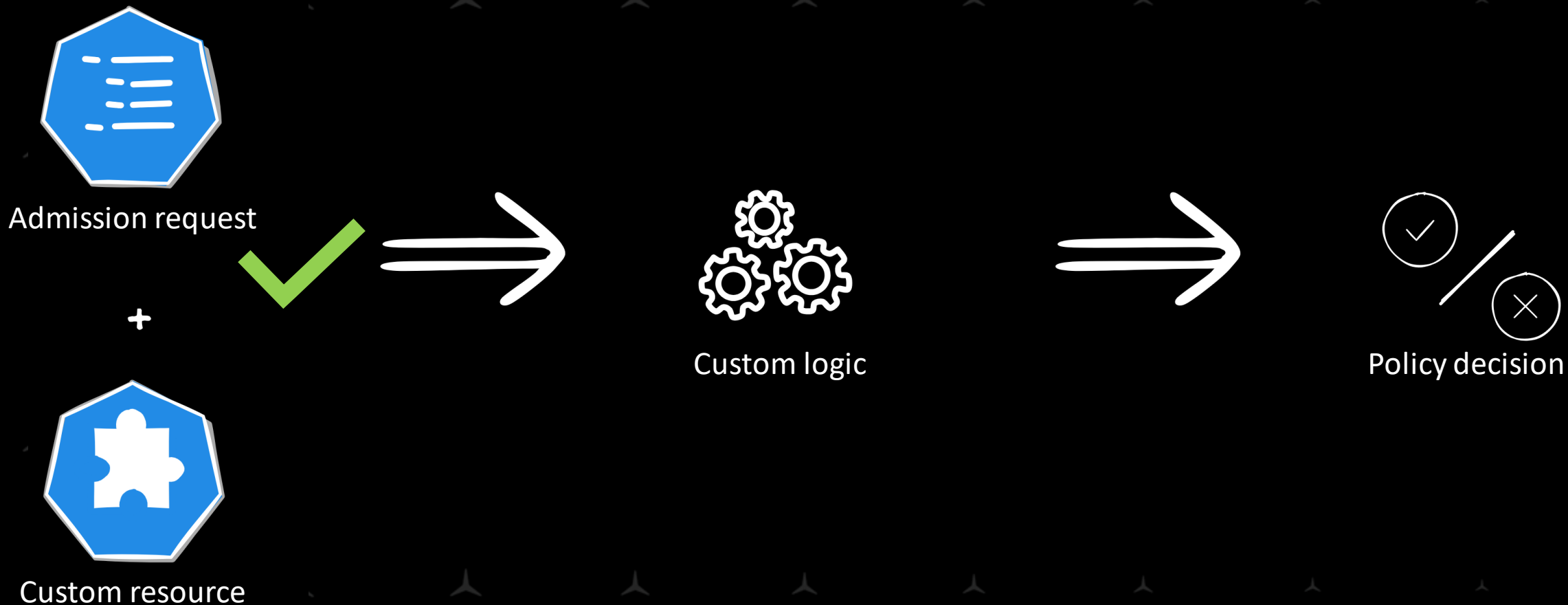
Lightning fast thanks
to in memory AST

No additional
controller or CRD
needed

So, Let's Build Some
Validating Admission Policies!



So, Let's Build Some Validating Admission Policies!



Anatomy of a Validating Admission Policy

```
1 apiVersion: admissionregistration.k8s.io/v1alpha1
2 kind: ValidatingAdmissionPolicy
3 metadata:
4   name: "foo-policy"
5 spec:
6   failurePolicy: Fail
7   paramKind:
8     apiVersion: addons.caas.mercedes-benz.com/v1alpha1
9     kind: AdmissionException
10  matchConstraints:
11    resourceRules:
12      - apiGroups: [ "apps" ]
13        apiVersions: [ "v1" ]
14        operations: [ "CREATE", "UPDATE" ]
15        resources: [ "deployments" ]
16  validations:
17    - expression: "!(object.spec.template.spec.serviceAccountName in params.spec.excludedServiceAccounts)"
18      reason: ServiceAccount not allowed!
```

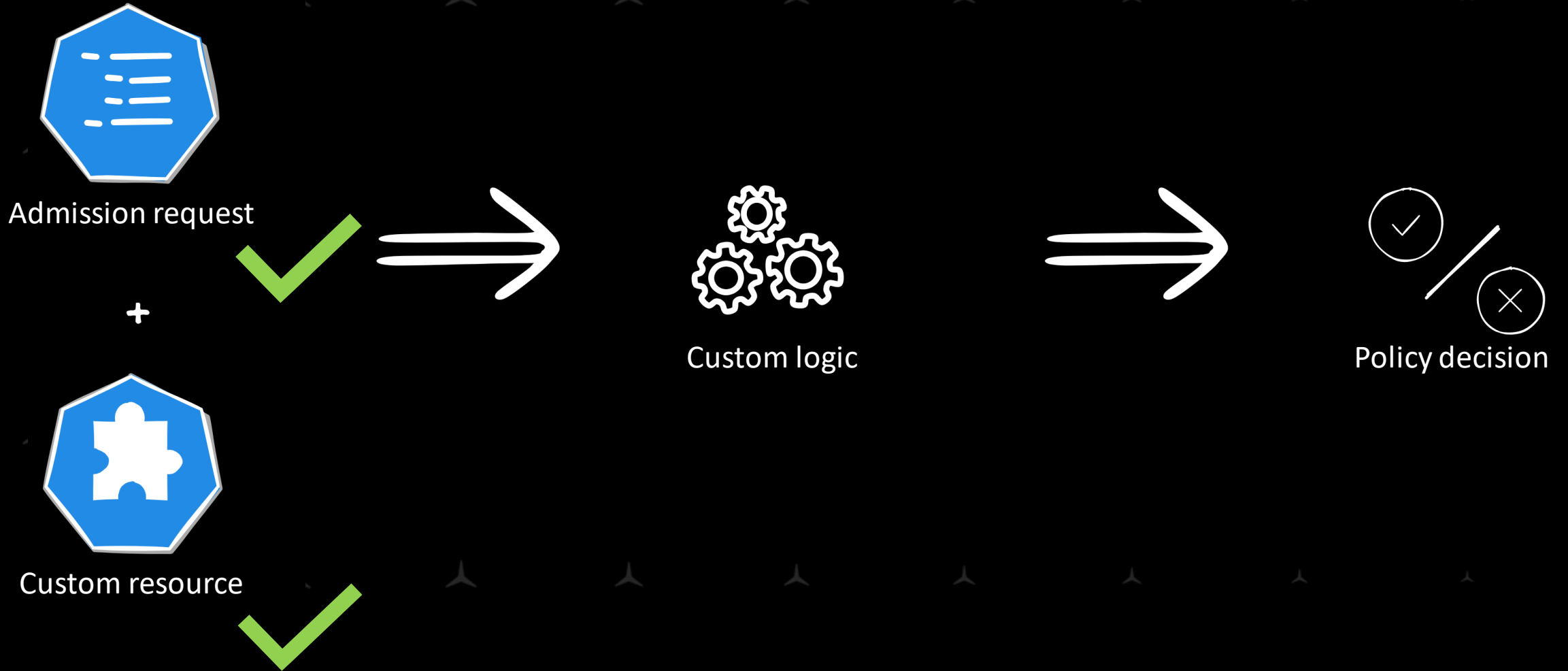
Anatomy of a Validating Admission Policy

```
1 apiVersion: addons.caas.mercedes-benz.com/v1alpha1
2 kind: AdmissionException
3 metadata:
4   name: foo-exception
5 spec:
6   excludedServiceAccounts:
7     - "kube-system:foo-baz"
```

Anatomy of a Validating Admission Policy

```
1 apiVersion: admissionregistration.k8s.io/v1alpha1
2 kind: ValidatingAdmissionPolicyBinding
3 spec:
4     paramRef:
5         name: foo-exception
6     policyName: foo-policy
```

So, Let's Build Some Validating Admission Policies!



Anatomy of a Validating Admission Policy

```
1 apiVersion: admissionregistration.k8s.io/v1alpha1
2 kind: ValidatingAdmissionPolicy
3 metadata:
4   name: "foo-policy"
5 spec:
6   failurePolicy: Fail
7   paramKind:
8     apiVersion: addons.caas.mercedes-benz.com/v1alpha1
9     kind: AdmissionException
10  matchConstraints:
11    resourceRules:
12      - apiGroups: [ "apps" ]
13        apiVersions: [ "v1" ]
14        operations: [ "CREATE", "UPDATE" ]
15        resources: [ "deployments" ]
16  validations:
17    - expression: "!(object.spec.template.spec.serviceAccountName in params.spec.excludedServiceAccounts)"
18      reason: ServiceAccount not allowed!
```

Common Expression Language

- Evaluated in linear time
- Mutation free
- Not Turing complete

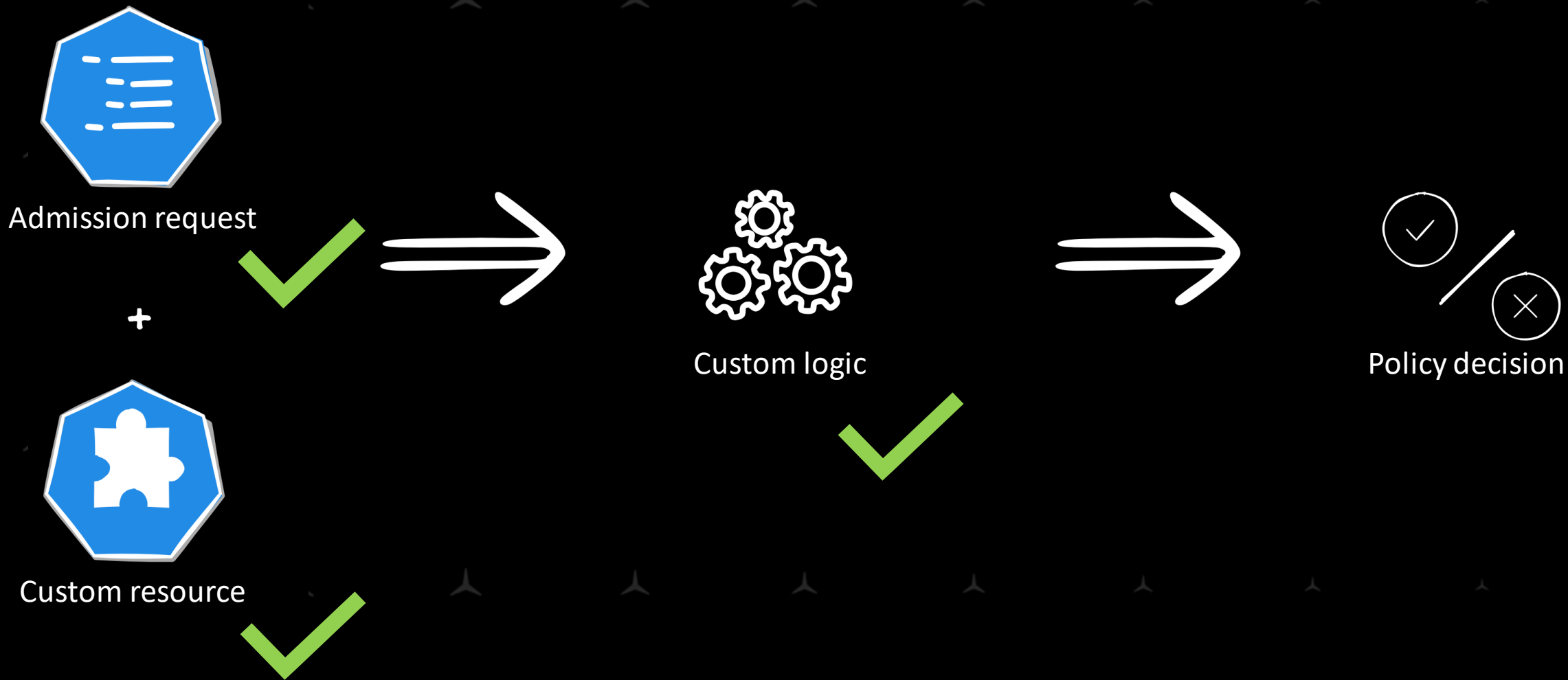
It's fast!

CEL Can Be a Hassle

```
1  validations:
2    - expression: "!(object.spec.template.spec.serviceAccountName in params.spec.excludedServiceAccounts)"
3    reason: ServiceAccount not allowed!
```

```
1  - expression: |
2    (
3      "system:bazs" in request.userInfo.groups ||
4      request.userInfo.username in ["foo-baz"] || (
5          has(object.spec.serviceAccountName) &&
6          request.namespace + ":" + object.spec.serviceAccountName in ["example-namespace:foo-controller-
7manager", "example-namespace:foo-baz-agent"]
8      ) || (
9          has(object.spec.template) &&
10         has(object.spec.template.spec.serviceAccountName) &&
11         request.namespace + ":" + object.spec.template.spec.serviceAccountName in ["example-namespace:foo-
12controller-manager", "example-namespace:foo-baz-agent"]
13     ) ) || ['Deployment', 'ReplicationController', 'ReplicaSet', 'DaemonSet', 'StatefulSet', 'Job'].all(kind,
14         object.kind != kind)
15     || !has(object.spec.template.spec.ephemeralContainers) ||
16     object.spec.template.spec.ephemeralContainers.all(container, !has(container.ports))
17     || !(container.ports.all(port, has(port.hostPort)))
18 )
```

So, Let's Build Some Validating Admission Policies!



Validation is Working

But what about mutating Pod resources to secure defaults?

kubernetes / enhancements

<> Code Issues 256 Pull requests 94 Actions Projects 3 Security Insights

Mutating Admission Policies #3962

Open 1 of 4 tasks jpbetz opened this issue on Apr 24, 2023 · 7 comments

jpbetz commented on Apr 24, 2023 · edited by cici37

Enhancement Description

- One-line enhancement description (can be used as a release note): Introduce CEL based Mutating Admission Policies
- Kubernetes Enhancement Proposal: [KEP-3962: Add MutatingAdmissionPolicy as provisional #3962](#)
- Discussion Link: <https://groups.google.com/g/kubernetes-sig-api-machinery/c/VY1oAl7a6nU>
- Primary contact (assignee): jpbetz, cici37
- Responsible SIGs: API Machinery
- Enhancement target (which target equals to which milestone):
 - Alpha release target (x.y): 1.30
 - Beta release target (x.y):
 - Stable release target (x.y):

☐ Alpha

- ☒ KEP ([k/enhancements](#)) update PR(s): [\[KEP 3962\]MutatingAdmissionPolicy: alpha in 1.30 #4399](#)
- ☐ Code ([k/k](#)) update PR(s):
 - [\[WIP\] MutatingAdmissionPolicy implementation](#) [kubernetes#123332](#)
- ☐ Docs ([k/website](#)) update PR(s): [\[KEP-3962\]Placeholder PR](#) [website#45248](#)

Please keep this description up to date. This will help the Enhancement Team to track the evolution of the enhancement

added the **needs-sig** label on Apr 24, 2023

jpbetz changed the title **KEP: Mutating Admission Policies** Mutating Admission Policies on Apr 24, 2023

Throwback to What Was Going on in Our Team at the Time

OPA

Implement all policies in OPA
using existing Setup
(without Gatekeeper
-> no mutation)

Custom Controller

Implement a Proof-of-Concept custom
controller to evaluate performance

Benchmarking Suite

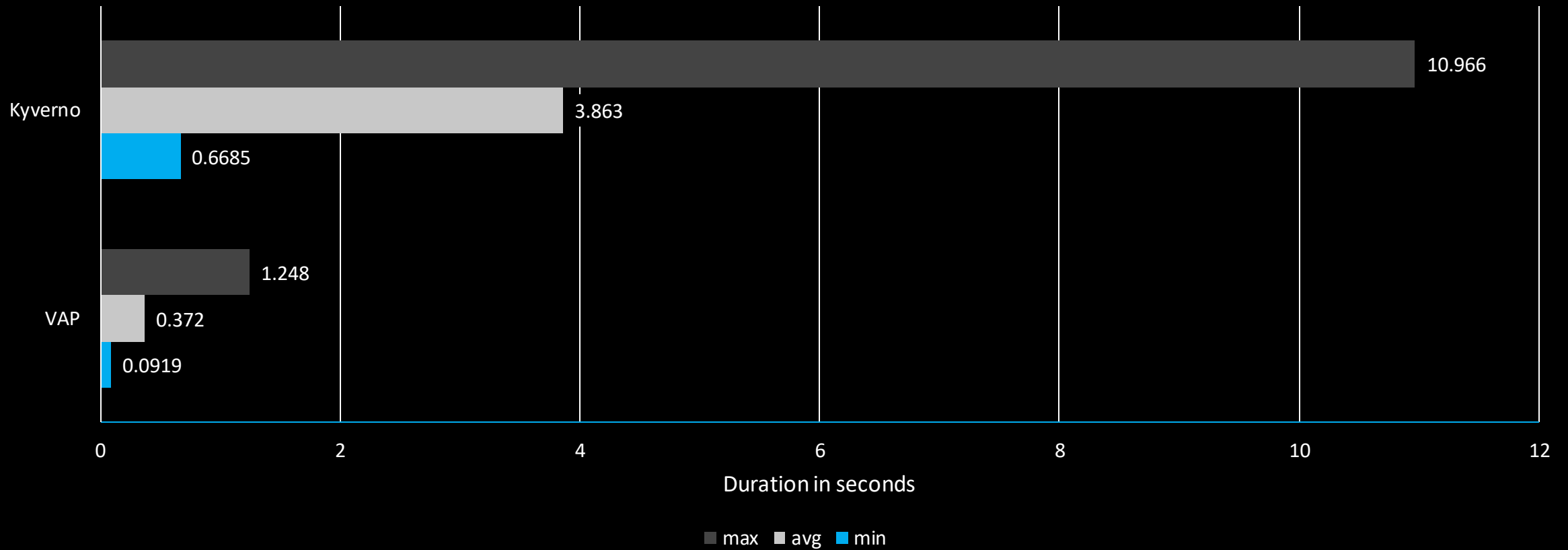
Improve K6 load testing suite
to better match real world
scenarios

Final Setup

- Validation in Lightning speed via Validating Admission Policies
- Mutation via tiny custom controller

Performance Comparison Between Kyverno and Validating Admission Policies

Benchmarks



But Wait...

Validating Admission Policies are only available starting with Kubernetes v1.26

What about the transitionary state for v1.25?



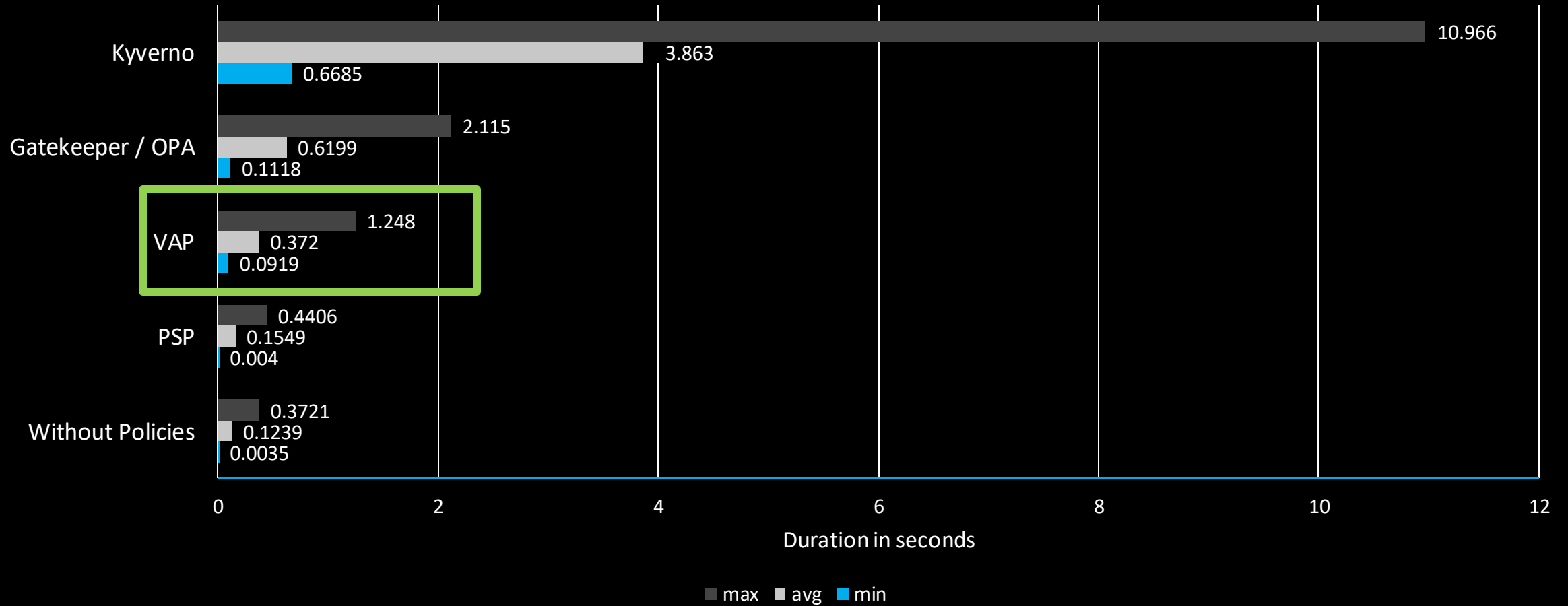
Throwback to What Was Going on in Our Team at the Time

OPA	Custom Controller	Benchmarking Suite
Implement all policies in OPA using existing Setup (without Gatekeeper -> no mutation)	Implement a Proof-of-Concept custom controller to evaluate performance	Improve K6 load testing suite to better match real world scenarios

Upgrade From Kubernetes v1.24 to v1.26

- Kubernetes v1.25 with OPA and admission-controller
- Kubernetes v1.26 with Validating Admission Policies and admission-controller
- Future plans: Validating and Mutating Admission Policies without the need of an admission-controller
 - All built-in, no need of HTTP overhead

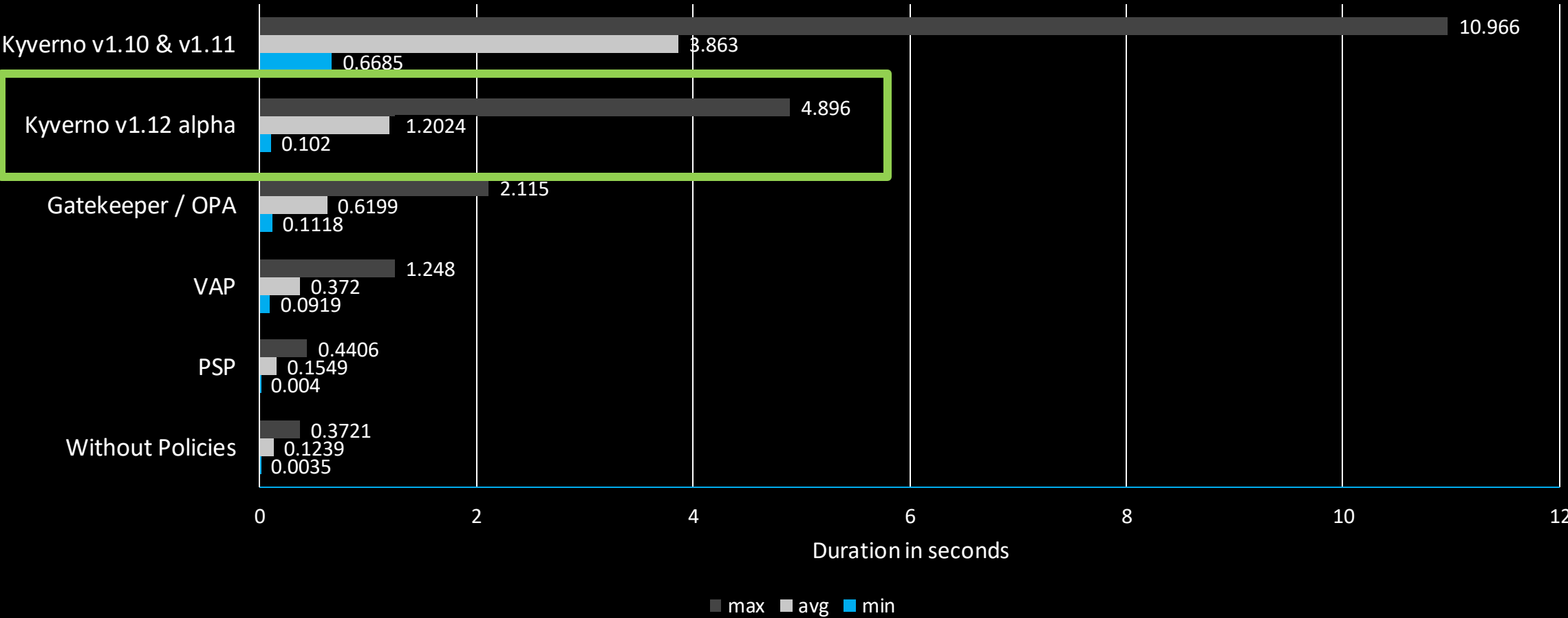
Final Benchmark – Comparison



What Happened Since Then

- Kyverno improved performance for their upcoming v1.12.0 release
- **Thanks to our colleague Robin Afflerbach**
- Kyverno can generate Validating Admission Policies

Final Benchmark – Comparison



Lessons Learned

- Benchmark your policies
- No really BENCHMARK YOUR POLICIES, it will save a lot of work
- Good and tool independent e2e-Tests are crucial for quick iterations and comparability
- Take care of sub resources like pods/ephemeralcontainers
- We payed the price of early adoption #techinnovators
- A pure Kubernetes solution is feasible and maintainable

We're Happy

BELIEVE IT

Mercedes-Benz
Tech Innovation



We're hiring!



www.mercedes-benz-techinnovation.com/en/jobs/

Mercedes-Benz

