

Kubernetes Security at Shopify Scale

Automating security across an infrastructure monorepo





We secure Shopify's infrastructure for millions of merchants



Jie Wu



Pulkit Garg

Infrastructure Security

****Today** We are Professional Cat Herders**



Kubernetes security at scale is like herding cats



...except these cats can accidentally expose your entire infrastructure.

SERVICE LANDS

CLUSTER ISLANDS

THE DEPLOY ROAD

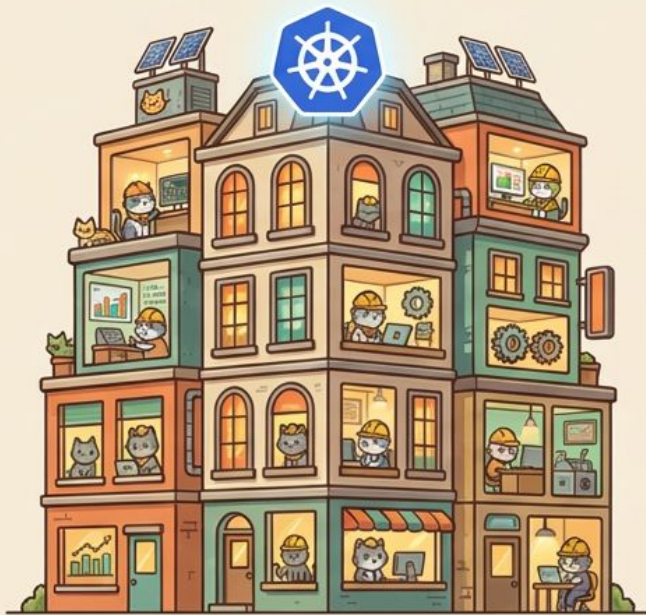
THE MONOREPO
FORTRESS

HELM WORKSHOP

THE DEPLOY ROAD

HELM WORKSHOP

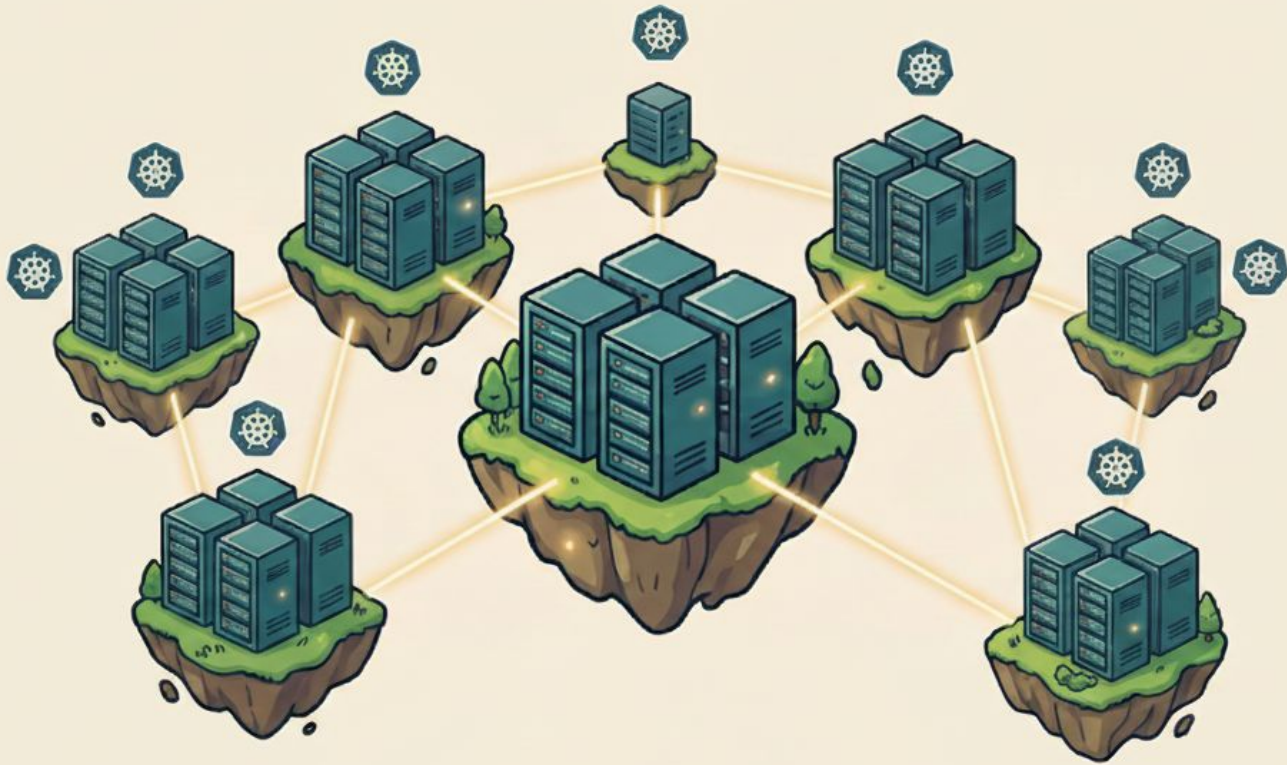




Shopify runs hundreds of services on Kubernetes



Our teams push ~60,000 deploys every week

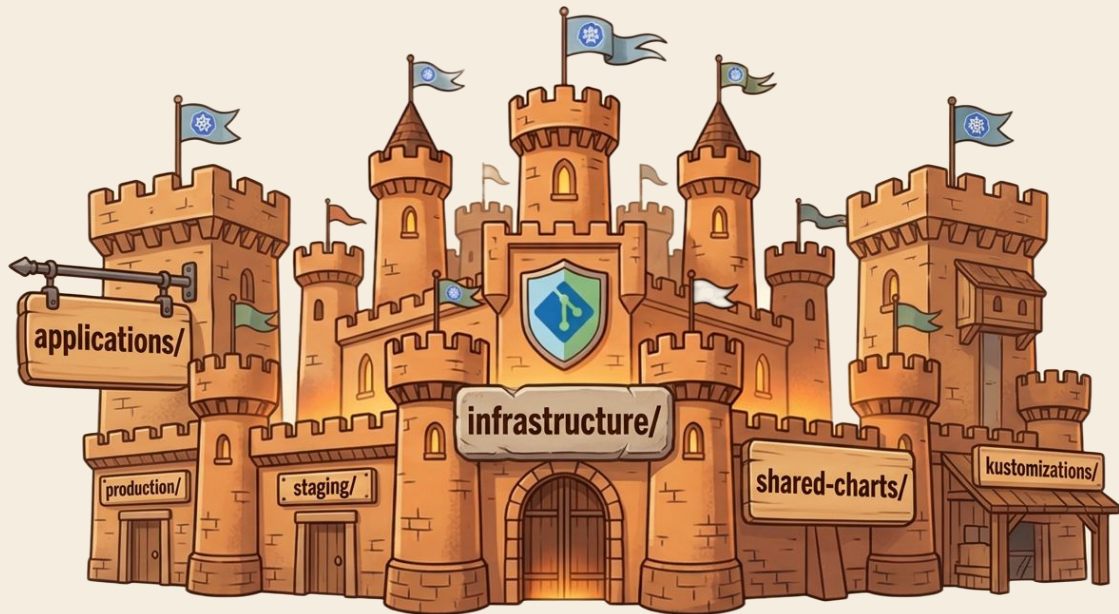


The kingdom spans ~500 clusters worldwide

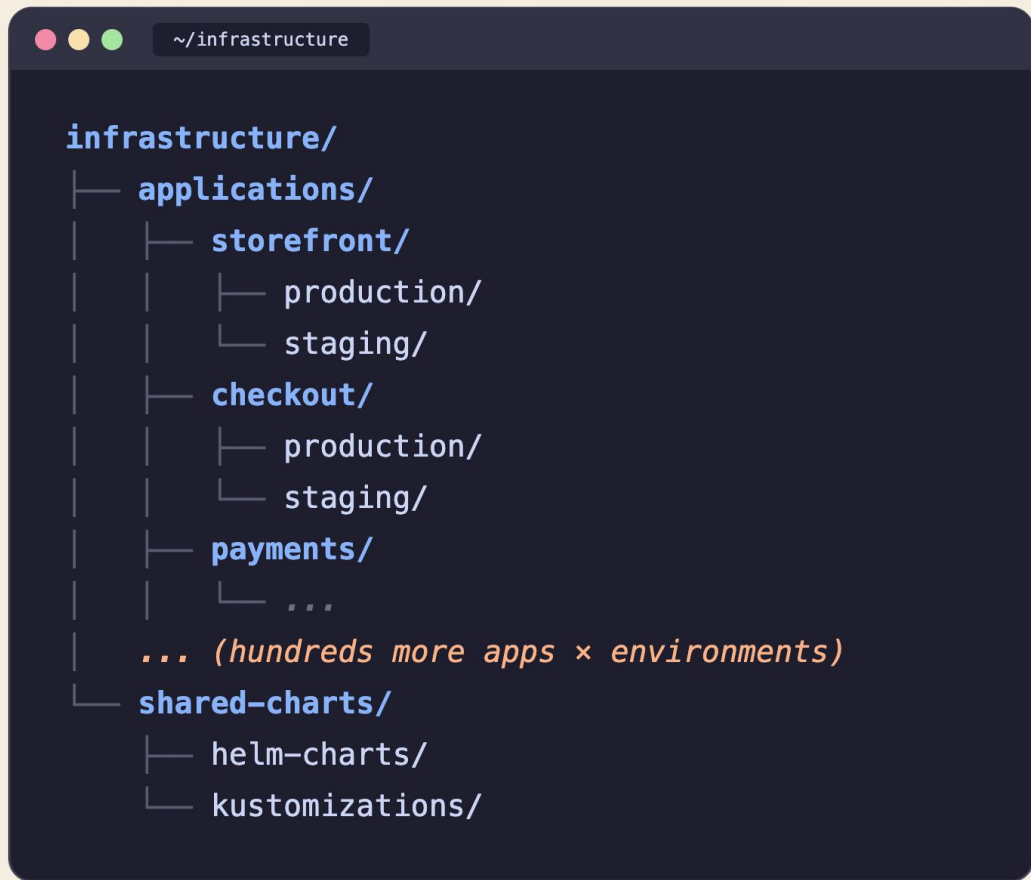


Millions of merchants depend on this infrastructure

Everything lives in one infrastructure monorepo



one source of truth, one source of risk



Developers ship configs
through the monorepo
every day





1 helm chart



Hi, I'm Hazel

(The Helm Hedgehog)

**100s of
rendered
manifests**



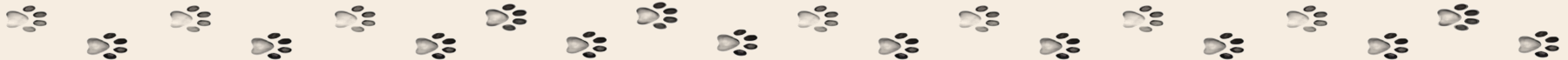
Change one template → update hundreds of services

Dev writes it



Hazel multiples it

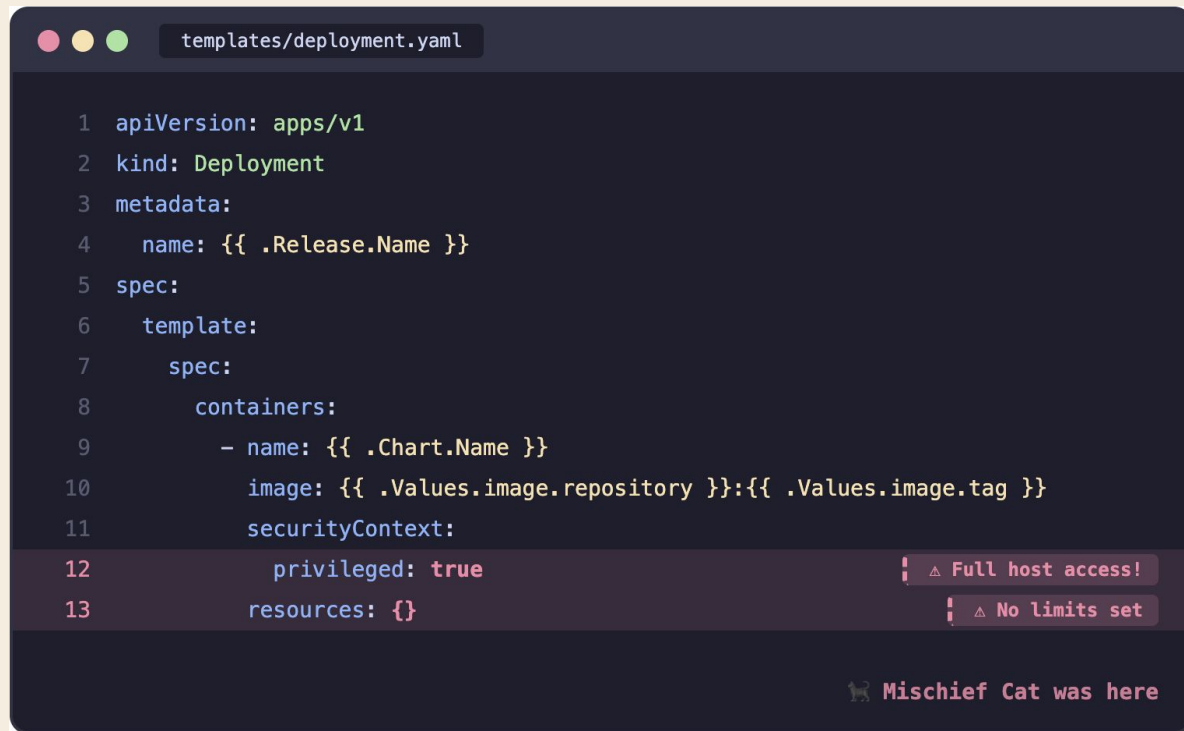
But not everyone follows the rules...



Not every change that ships is a good one



A single bad config in a shared template can compromise hundreds of services



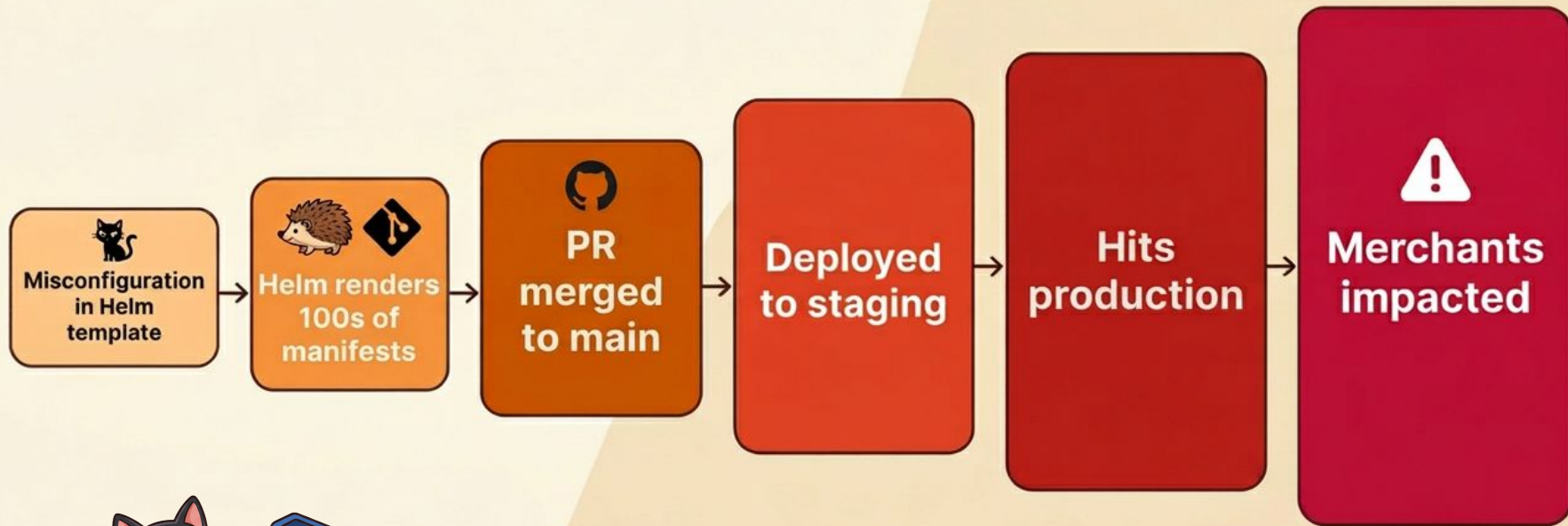
The image shows a code editor window titled 'templates/deployment.yaml'. The code is a Kubernetes Deployment manifest. Lines 12 and 13 are highlighted in red, indicating a security issue. Line 12 has a warning 'Full host access!' and line 13 has a warning 'No limits set'. At the bottom right, there is a message 'Mischief Cat was here' with a cat icon.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: {{ .Release.Name }}
5  spec:
6    template:
7      spec:
8        containers:
9          - name: {{ .Chart.Name }}
10            image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
11            securityContext:
12              privileged: true
13              resources: {}
```

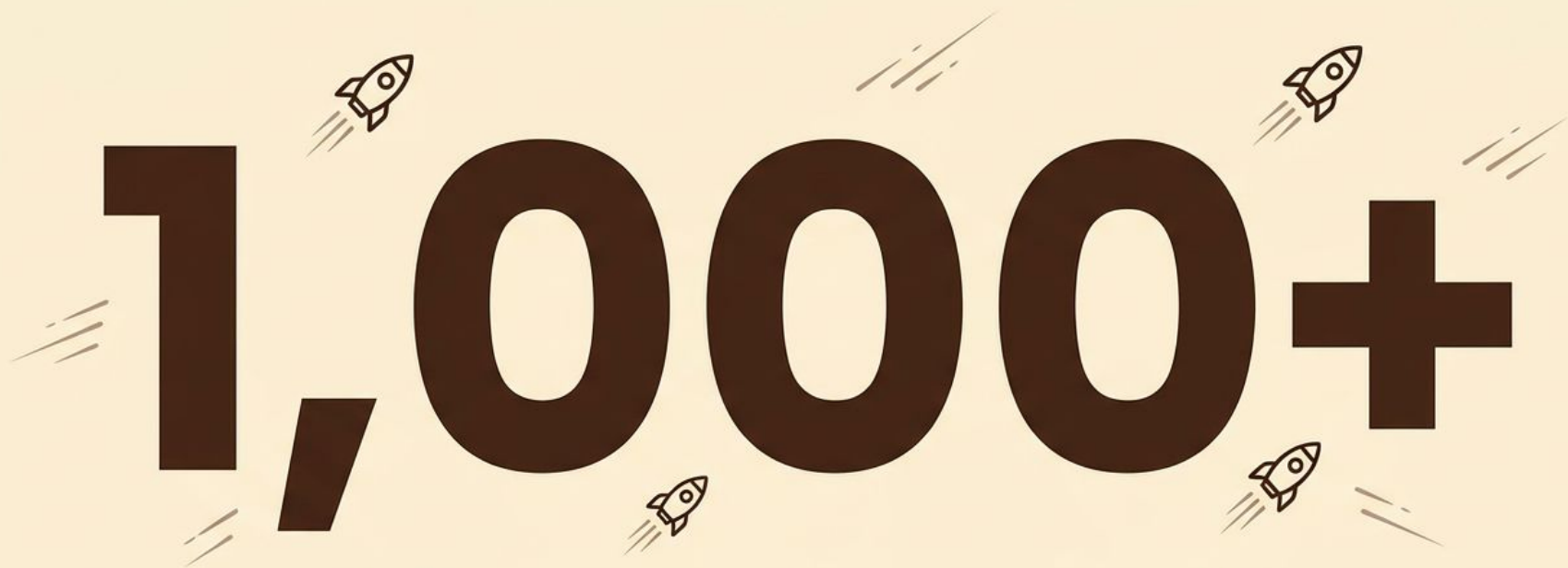
⚠ Full host access!

⚠ No limits set

🐱 Mischief Cat was here

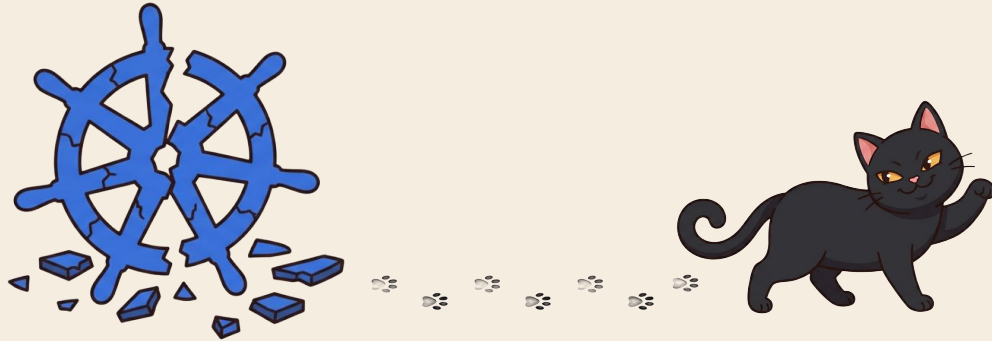


1,000+

The number '1,000+' is rendered in a large, bold, dark brown font. It is decorated with four small rocket ship icons, each with three motion lines behind it, suggesting speed. The rockets are positioned around the number: one above the first zero, one above the second zero, one below the first zero, and one below the second zero. There are also several short, diagonal motion lines scattered around the number, enhancing the sense of rapid growth or achievement.

PRs merged every week

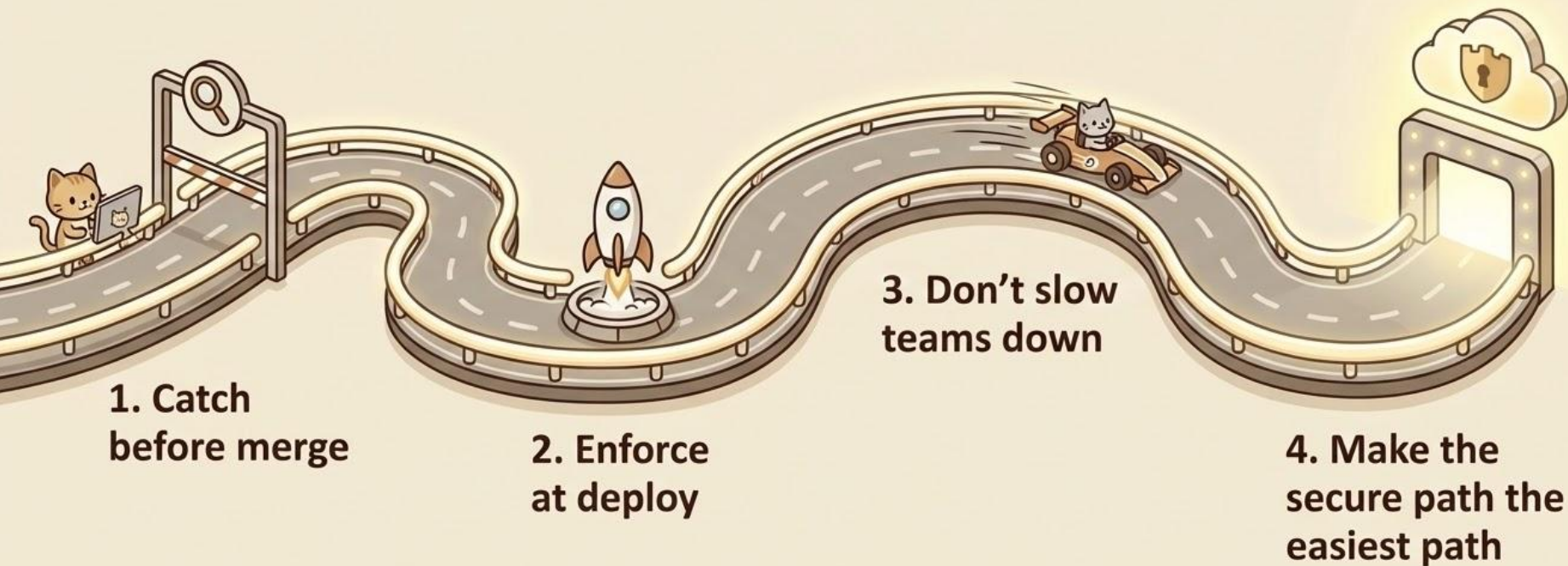
We hit a tipping point. Misconfigurations were slipping through.





Something had to change

We needed automated guardrails that enables speed, not fight it



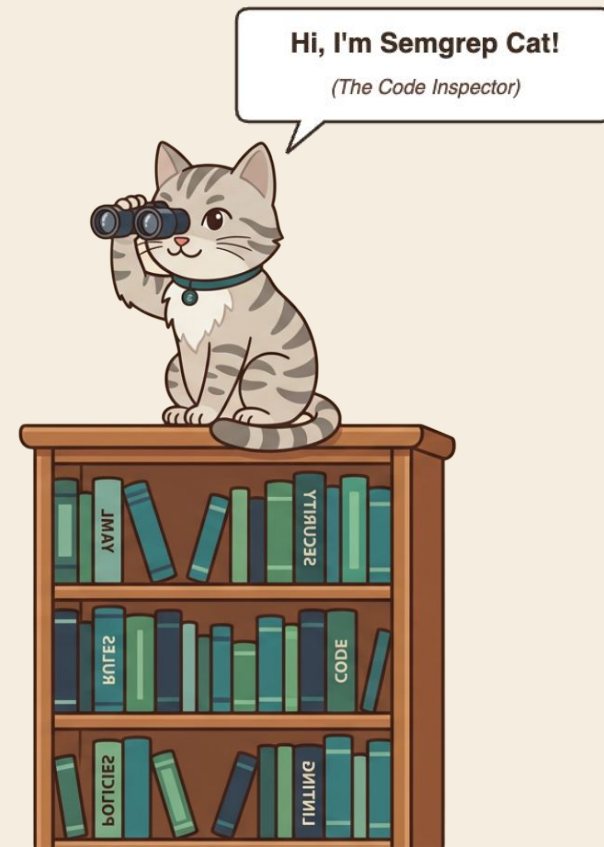
Catch it before it merges



```
templates/deployment.yaml

1 # templates/deployment.yaml (fixed)
2 spec:
3   template:
4     spec:
5       containers:
6         - name: {{ .Chart.Name }}
7           image: {{ .Values.image.repo }}
8           securityContext:
9             privileged: false                ! ✓ Unprivileged container
10          resources:
11            limits:
12              cpu: "500m"
13              memory: "256Mi"              ! ✓ Resource limits defined
14          readinessProbe:
```

🐱 Semgrep Cat approved ✓



Three lines of YAML catch the exact misconfiguration from earlier

```
.semgrep/rules/no-privileged-containers.yaml

1 rules:
2   - id: no-privileged-containers
3     pattern: |                                     ! ← 3 lines catch it
4       securityContext:
5         privileged: true
6     message: >
7       Containers must not run in privileged mode.
8       This grants full access to the host.
9     severity: ERROR
10    languages: [yaml]
```

🐱 Semgrep Cat – 1 rule, privileged containers blocked ✓

Semgrep can catch what's missing, not just what's wrong

```
.semgrep/rules/require-resource-limits.yaml

1 rules:
2   - id: require-resource-limits
3     patterns:
4       - pattern: |                                     ! ← find containers
5         containers:
6           - name: $NAME
7       - pattern-not: |                                  ! ← but NOT this
8         containers:
9           - name: $NAME
10          resources:
11            limits:
12              ...
13      message: "Container '$NAME' is missing resource limits"
14      severity: WARNING
15      languages: [yaml]
```

🐱 Semgrep Cat catches what's missing, not just what's wrong ✓

One rule catches three ways a container can access the host

```
.semgrep/rules/no-host-network.yaml

1 rules:
2   - id: no-host-network
3     patterns:
4       - pattern-either:
5         - pattern: "hostNetwork: true"
6         - pattern: "hostPID: true"
7         - pattern: "hostIPC: true"
8     message: >
9       Pods must not share host namespaces.
10      This allows lateral movement across nodes.
11    severity: ERROR
12    languages: [yaml]
```

← network stack
← process space
← IPC namespace

🐱 Semgrep Cat – 1 rule, 3 vectors blocked ✓



- ✓ privileged containers
- ✓ host namespaces
- ✓ resource limits

You don't know what you're deploying if you use :latest"

```
.semgrep/rules/no-latest-image-tag.yaml

1 rules:
2   - id: no-latest-image-tag
3     patterns:
4       - pattern: |                                     ← capture the image
5         image: $IMAGE
6       - metavariable-regex:
7         metavariable: $IMAGE
8         regex: ".*:(latest)$|^[:]+$"                 ← :latest or no tag
9     message: >
10      Pin image tags to a specific version.
11      ':latest' or untagged images are a supply chain risk.
12     severity: WARNING
13     languages: [yaml]
```

🐱 image: nginx:latest → 🚫 supply chain risk

A missing annotation means your service has no auth

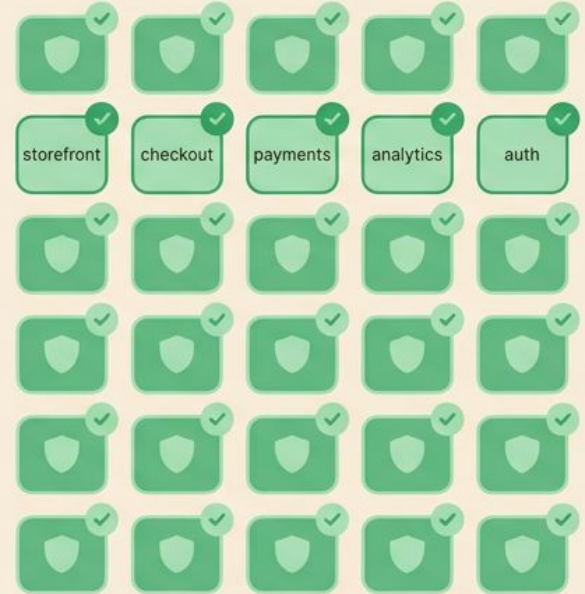
```
.semgrep/rules/require-auth-annotation.yaml

1 rules:
2   - id: require-auth-annotation
3     patterns:
4       - pattern: |                                     ! ← match any Ingress
5         kind: Ingress
6         metadata:
7           ...
8       - pattern-not: |                                ! ← but NOT this
9         metadata:
10          annotations:
11            security.shopify.io/auth: ...              ! ← missing auth!
12     message: >
13       Ingress is missing security.shopify.io/auth annotation.
14       If missing, auth may not be enforced on this ingress.
15     severity: ERROR
16     languages: [yaml]
```

🐾 Missing annotation = improperly configured auth

Fix one Helm template, secure hundreds of services

1 Helm Template Fix



Developers get instant, actionable feedback right in their PR

 production/build/deployment.yaml

@@ -280,6 +280,10 @@

280280

ports:

281281

- containerPort: 8080

282282

name: metrics

283283

securityContext:

284284

privileged: true

 Check warning

 Code scanning / Semgrep OSS

Semgrep Finding: privileged-true-pod ⚠ Warning

privileged is set to true in securityContext of container or pod.

Dismiss alert

Show more details

285285

resources: {}

Self Serving Guidance

Avoid Running as Privileged

A privileged container has almost the same access as processes running directly on the host machine. It can access all host devices and bypass most security protections.

Best Practice: Avoid setting `privileged: true` in container security contexts.

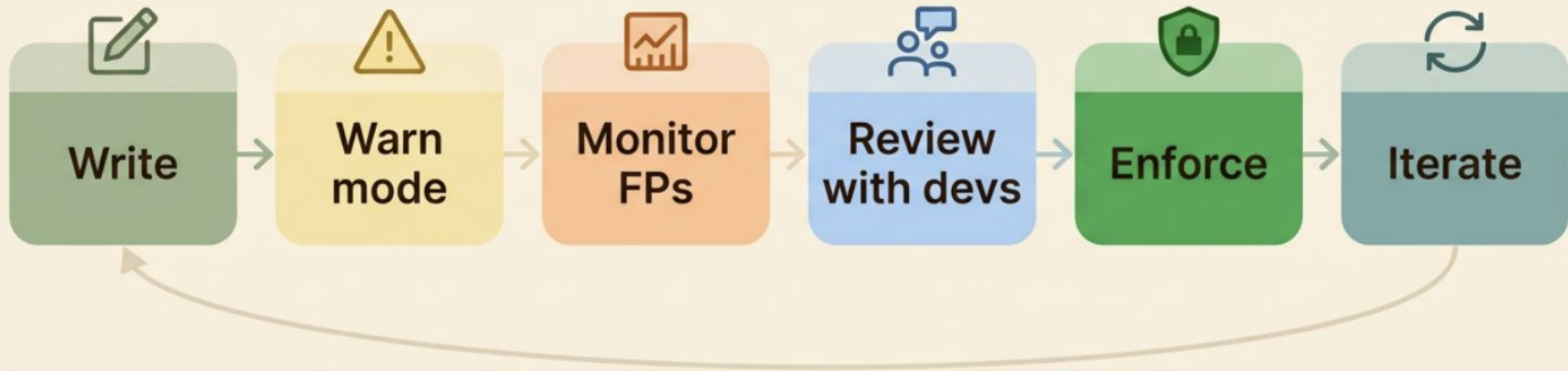
Why: Privileged containers can access all host devices, have nearly the same privileges as the host OS, and can potentially compromise the entire Kubernetes node.

❌ Don't do this:

```
securityContext:  
  privileged: true
```

✅ Do this:

```
securityContext:  
  allowPrivilegeEscalation: false  
# privileged defaults to false – just don't set it to true
```



We treat security rules with the same rigor as production code

Why not OPA for K8s?

here's how we decided



Semgrep

- Scans templates at the source
- Runs in CI, no cluster infra
- One fix → hundreds secured

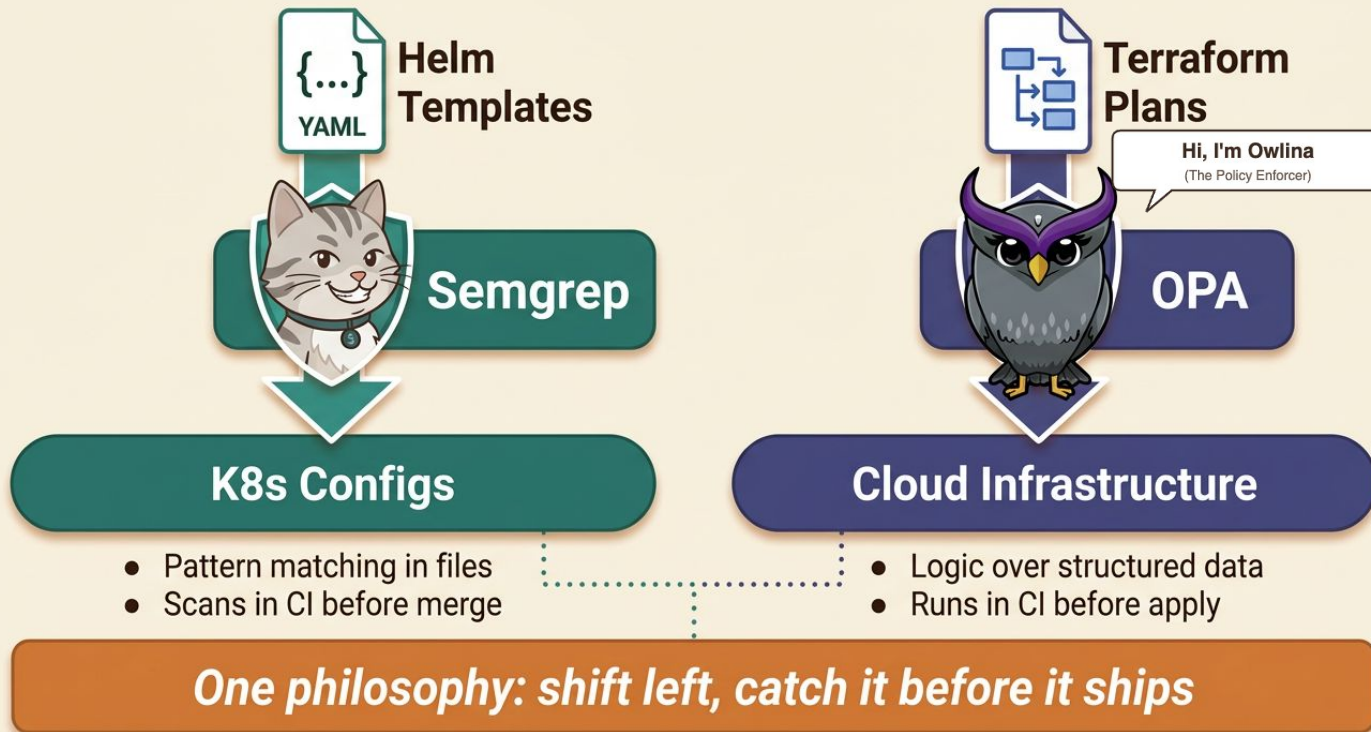
vs



OPA

- kubectl deploys, not CI
- Runtime admission control
- Audits running workloads

The same policy-as-code mindset secures our terraform



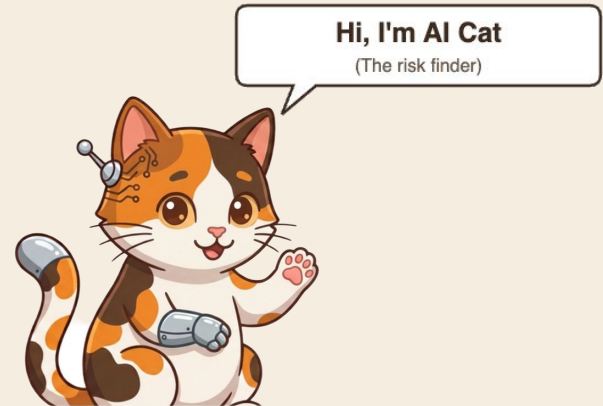
OPA blocks restricted GPU types before infrastructure reaches production

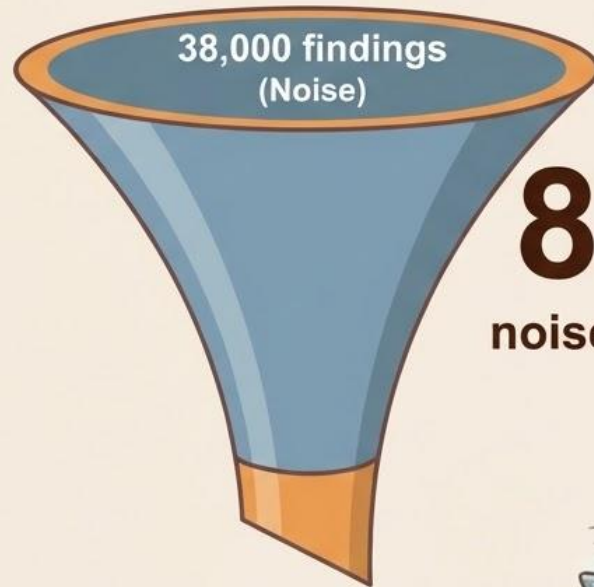
```
polices/terraform/restrict-machine-types.rego

1 package terraform.gke
2
3 deny[msg] {                                     ! ← blocks the pipeline
4   resource := input.planned_values.root_module.resources[_]
5   resource.type == "google_container_node_pool"
6
7   machine_type := resource.values.node_config[_].machine_type
8   restricted := {"n2d-standard-128", "c3-highcpu-176"} ! ← costly oversized VMs
9   restricted[machine_type]                          ! ← match check
10
11   not has_approval(resource.name)
12
13   msg := sprintf(
14     "Node pool '%s' uses restricted machine type '%s'. "
15     "Requires infrastructure-security approval.",
16     [resource.name, machine_type]
17   )
18 }
```

OPA — runs in CI against terraform plan

AI helped us find the blind spots we didn't know we had





38,000 findings
(Noise)

88%
noise reduction

~4,500 findings
(Actionable)



AI turns a security idea into a working rule in minutes



AI-Powered Rule Development:



Developer



Helm

Risk Finder



Code Inspector



Misconfiguration

Runs Securely



code → scan → deploy → cluster

Code/PR

CI/Scanning

Cluster

So... did it work?

Identify, Craft, Ship & Repeat

~50

Semgrep rules



13

Terraform OPA policies



Industry Framework Coverage



CIS Benchmarks



MITRE



OWASP

Before & After

What changed when we turned on automatic guardrails

BEFORE

0

automated K8s security checks

— Manual reviews only

— Inconsistent coverage across thousands of PRs

— Misconfigs reach production undetected

AFTER

Every PR

scanned automatically before merge

1,000+ misconfigs

caught before reaching production

0 ✓

security rollout incidents

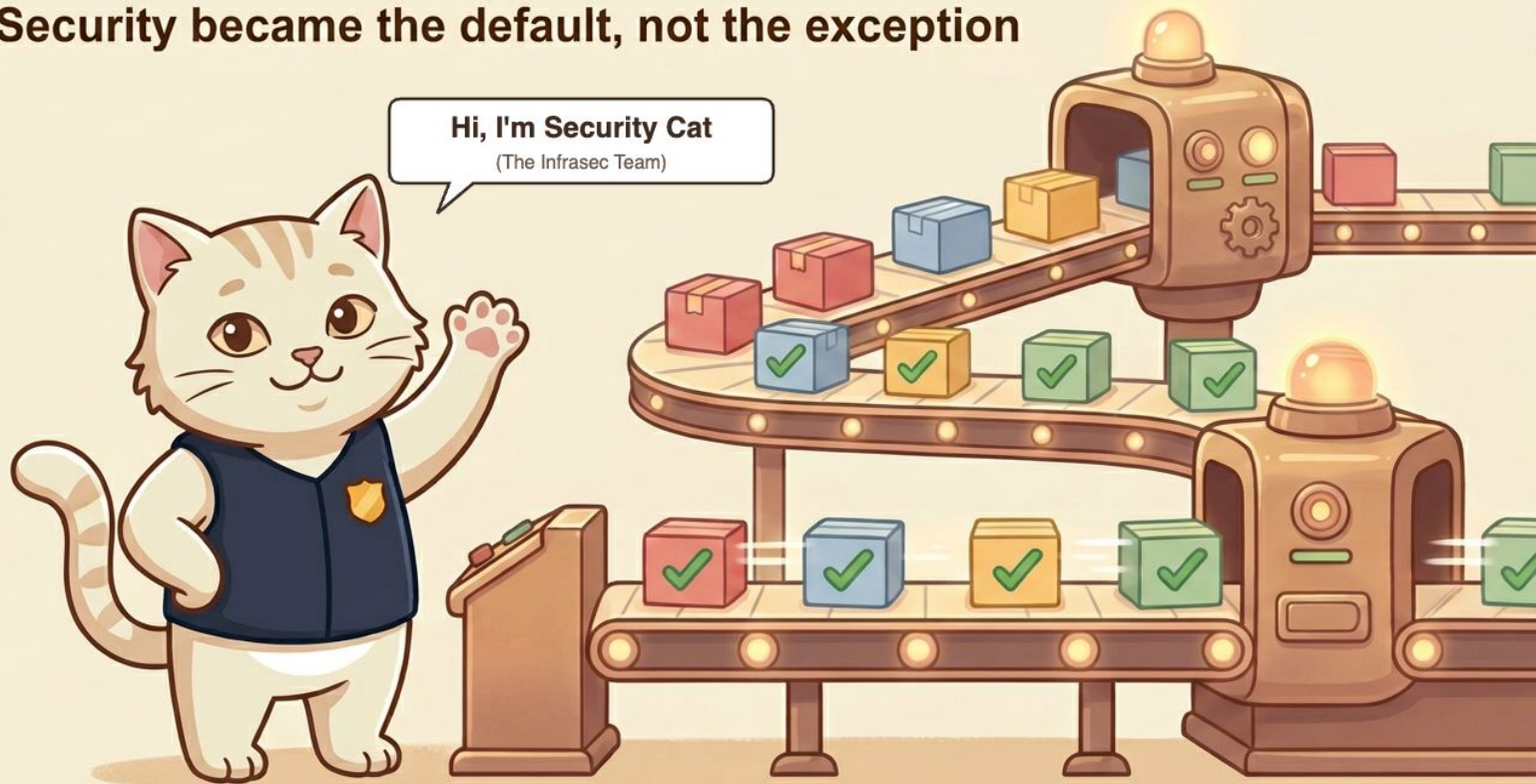
privileged containers

capability drops

host namespaces

root users

Security became the default, not the exception



Together, we have



Guardians of the Infrastructure Monorepo



It wasn't all smooth sailing

False positives



False positives

Developer
pushback



False positives

Developer
pushback

Course correction



False positives

Developer pushback

Course correction

We learned the hard way



**Start in warn
mode (always)**



**False positives
everywhere**



**Clear steps for
remediation**



**Pave the road
first**

Pave the secure road for your teams to run on



The End





Stay in Touch

<https://www.linkedin.com/in/wu-jie>

<https://www.linkedin.com/in/garg-pulkit>