

fluxCon

NORTH AMERICA

From Helmfile to Flux

GitOps for Telco at Enterprise Scale

fluxCon
NORTH AMERICA



Jesús González Royo
E2E CI/CD & Automation Architect
Ericsson



Carl-Fredrik Sundstrom
Senior Principal Developer
Ericsson

WHERE WE STARTED

The honest baseline

1	**Quarterly releases**	A cadence the whole portfolio was built around.
2	**Mature CI/CD pipelines**	Years of investment. They worked. People trusted them.
3	**Helmfile per product**	Already how we packaged and shipped everything.
4	**Customers on their own clock**	Every operator upgrading when their change window allowed.

None of this was broken. That is exactly what made it hard to change.

THE PROBLEM

We don't run the clusters

ERICSSON

150 years

The Situation

A telco operator migrates its own clusters to GitOps. One organization, one change-control process, one timeline. Hard but it is their decision when to move.

Our problem

We are the vendor.

Our software runs in many operator's clusters in their premises, their change control, their timelines. We ship GitOps.

We don't get to switch it on.

That one difference shaped every decision in this talk.

ADOPTION

Nobody gets forced across

Classic delivery	Customer keeps the Helmfile based deployment they have. Nothing breaks.
Hybrid	GitOps for some domains, classic Helmfile for the rest. Both at once.
Full GitOps	Continuous deployment, day-2 config in Git, reconciliation on.

One product.
One release.
One artifact set.

We support the runway without forking the product to allow it.

THE DECISION

Why we didn't rebuild

ERICSSON

150 years

✗

Rebuild the product pipelines

A multi-year program. Product teams blocked. Every customer forced onto a new model at once. The business does not wait, so the migration quietly dies.

✓

Convert what we ship

Keep the product pipelines. Generate Flux resources from the Helmfile assets we already produce. Let each customer move when their change board says yes.

Everything after this slide follows from that choice.

THE CONVERSION

Helmfile to Flux

ERICSSON 150 years

Helmfile

What our pipelines already produce

›

Convert

Generate Flux resources automatically

›

Git

Versioned per product, per environment

›

Reconcile

Flux applies it and keeps it applied

To be clear:

This is not Helmfile/helm versus Flux, helm-controller is Flux, and Flux 2.8 ships Helm v4 natively.
Our Helm charts stay Helm charts. What changed is that delivery became declarative instead of imperative.

THE CONVERSION

Helmfile to Flux manifest

INPUT what our pipelines already produce

helmfile.yaml

```
helmDefaults:
  wait: true   atomic: true   timeout: 900
releases:
- eric-sec-sip-tls      labels:{crd: admin}
- eric-sec-certm        labels:{crd: admin}
- eric-bss-bam-rbac-roles {rbac: admin}
- eric-bss-bam-cm-agent    app
- eric-bss-bam-authn-proxy app
- eric-bss-bam-cli         app

    # expand one release ↓
    - name: eric-bss-bam-cm-agent
      values/eric-bss-bam-cm-agent.yaml:
        enabled: true
        replicaCount: 1
        config: { gitClient: {...} }

eric-bss-bam-cm-agent:
  version: 1.52.0+2
  repository: proj-adp-gs-all-helm
eric-bss-bam-authn-proxy:
  version: 1.25.0+3
  ....
```

fluxtool
hf convert -b

labels crd/rbac:admin
→ platform; rest → tenant

+ 1000s more
HelmReleases converted

OUTPUT versioned Flux manifests

platform-admin repo
CRDs + RBAC · flux-system · admin-approved

HelmRelease **eric-sec-sip-tls-crd** 7.0.0+12
→ eric-crd-ns crds: CreateReplace
HelmRelease **eric-sec-certm-crd**
HelmRelease **eric-bss-em-rbac** 22.11.0+871

tenant repo
app HelmReleases + source · bam-essvt1 · offering-approved

HelmRelease **eric-bss-bam-cm-agent** 1.52.0+2
chart.sourceRef → HelmRepository bam-oci
values: { global.namespace: bam-essvt1 }
HelmRelease **eric-bss-bam-authn-proxy** 1.25.0+3
HelmRepository **bam-oci** → oci://.../bam/helm-charts

One Helmfile and its values become versioned Flux manifests — CRDs and RBAC in the platform-admin repo, application HelmReleases in the tenant repo. Same charts, same expected result: the legacy Helmfile procedure and GitOps delivery stay in step.

REVERSE RECONCILIATION

Letting Flux adopt a running cluster

ERICSSON 150 years

1 Brownfield

Hundreds of Helm releases, installed imperatively, already serving live traffic.

2 Generate

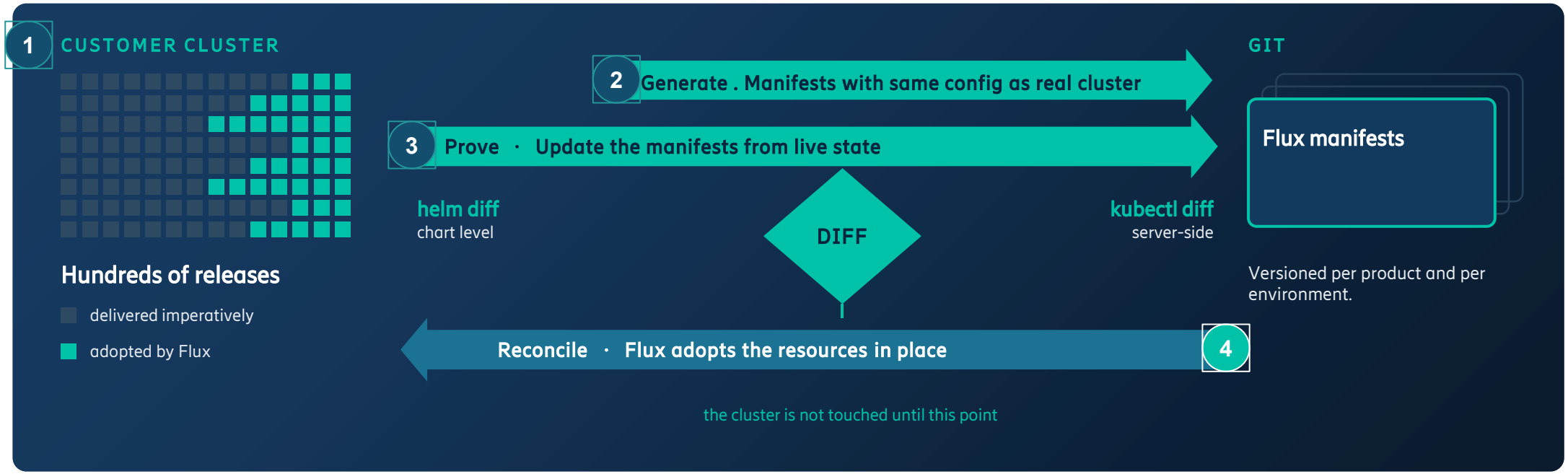
Flux manifests derived from the Helmfile, whole fleet of µService at once, not one by one.

3 Prove

Diff the generated intent against live cluster reality. Nothing applied, nothing resumed yet.

4 Reconcile

Reconciliation on. Flux adopts what is already there instead of redeploying it.



Read, publish, reconcile. The first two never touch the cluster — so by the time Flux is resumed, we already know exactly what will change.

THE SETUP

From suspended to reconciling

ERICSSON 150 years

What the generator writes



pre-resume check

```
$ flux create kustomization prod \
      --path ./generated/prod --suspend

$ diff-tool diff --server-side upgrade <release>
… diff-tool completed

  ownership annotations added ......... expected
  resources recreated ........................... 0
  resources deleted ............................. 0

$ flux resume kustomization prod
  reconciliation enabled
```

Two zeroes are the whole gate. Anything else and we stop.

One commit and one resume. Until that resume the cluster has not changed at all — which is what makes a brownfield takeover safe to attempt on a live system.

SCALE TESTING

Thousands of nodes that do not exist

ERICSSON 150 years

Not what we are testing

Application functionality and chart correctness are already proven when the software is delivered. Re-running that at scale tells us nothing new.

What we are testing

The delivery system itself — manifests, image availability, pipelines, and the conversion tooling that has to hold up across a whole fleet.

Kwok Open source. Simulates thousands of Kubernetes nodes with no kubelets and no real workloads. Flux reconciles against them exactly as it would against a customer fleet.

What has to come back green before we touch a real cluster

- **Charts & images** — Every chart and OCI artifact actually resolvable at fleet size.
- **Pipelines** — The automation runs end to end, not just against a toy cluster.
- **Conversion tooling** — Our generators still hold across thousands of objects.
- **Reconciled state** — HelmReleases installed, Kustomizations ready, resources healthy.

Kwok gives us a cluster with no kubelets and nothing actually running — which is the point. The workload was never the part we doubted. The machinery that delivers it was.

SCALE TESTING

Standing up the fake fleet

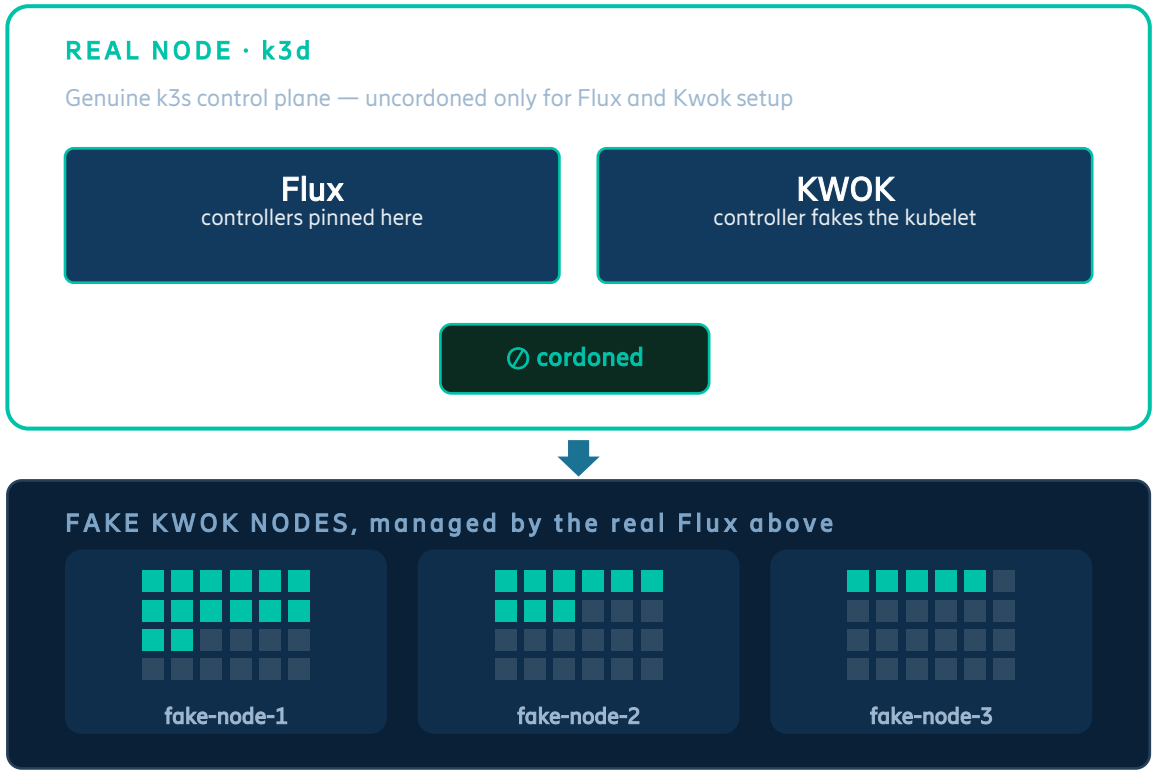
ERICSSON

150 years

FAKE CLUSTER SETUP · on the real node

1. **Create k3d cluster**
   One real node — genuine k3s API server.
2. **Deploy real Flux + KWOK**
   Controllers pinned to the real node; they actually run.
3. **Scale up 3 fake KWOK nodes**
   No kubelet — pods report Running instantly.
4. **Cordon the real node**
   So product workloads are kept off it.

5. **Product deployment**
   Releases managed by Flux are scheduled onto the fake KWOK nodes.*



* When a fake deployment needs a nudge to succeed — completing a hook Job, meeting a readiness or dependency gate — a KWOK Stage drives the resource to the right status, so the generated repo can reconcile

DAY-2 AND DAY-N

Operations as commits

ERICSSON

150 years

Independent delivery & upgrades

Each microservice evolves independently. Upgrades are pull requests; rollbacks are reverts.

Declarative configuration & drift control

Day-2 and Day-N configuration lives in Git. Flux continuously reconciles deviations.

Four-eye approval & auditability

Separate authors from approvers. Every approved change has a traceable history.

Separation of concerns

Shared clusters with clear boundaries between infrastructure administration and application tenants.

Compliance becomes a byproduct of operations where every change is reviewed and traceable, not an audit exercise.

OBSERVABILITY

Finding where the time actually goes

Upstream Otel contribution

ERICSSON 150 years

1 Duration

How long each Helm release actually took to reconcile.

2 Origin

Which Kustomization the release was delivered from.

3 Blocker

What it was waiting on, and what was holding it there.

4 Idle time

Waiting that should not be happening at all.

Flux reconciliation

Per-release timing and state

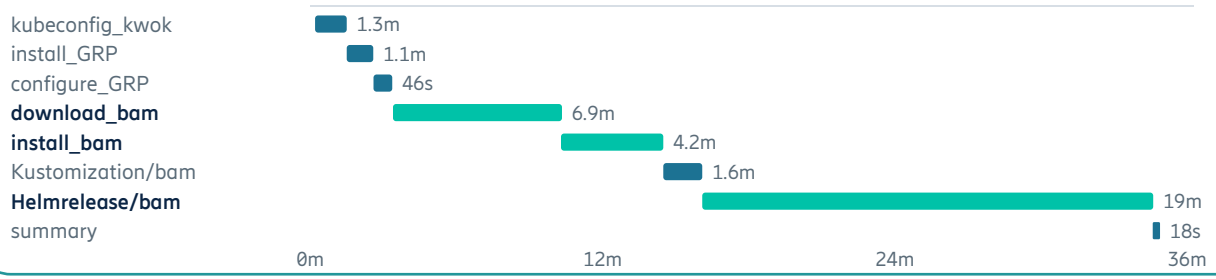
GitLab pipelines

Build and publish timing

›

One timeline across the automation loop (Gitlab & FluxCD)

One bam-regression run · 36 min end to end



The timing data we needed did not exist, so we are contributing it to Flux OTel distributed trace rather than keeping it in a fork every Flux user gets to see where reconciliation time goes.

If you take one thing

GitOps isn't simply a deployment mechanism.

It's an operating model for regulated environments.

Jesús González Royo · Carl-Fredrik Sundstrom · Ericsson

Find us in the hallway — we'd like to hear how you solved this



Q&A

From Helmfile to Flux: GitOps for Telco at Enterprise Scale

Share your feedback

https://www.ericsson.com/