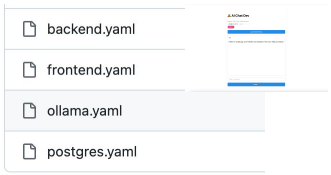

https://kccncna2025.sched.com/event/27FeG/gitops-without-variables-brian-grant-alexis-richardson-confighub-inc?iframe=no&w=100%&sidebar=yes&bg=no

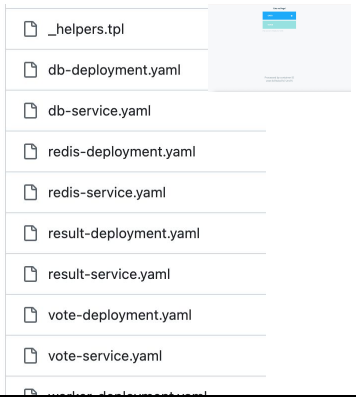
Brian Grant: original lead architect of Kubernetes
Alexis Richardson: founder of WeaveWorks, coined “GitOps”

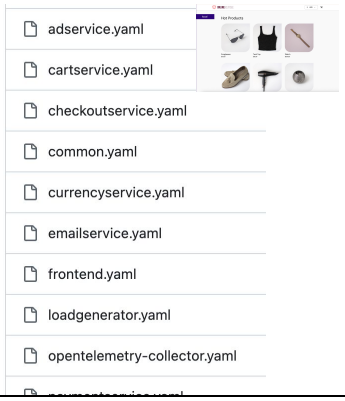
Today we’re going to talk about what’s possible when configuration is always fully rendered with no parameters or variables, rather than stored as templates, patches, or configuration generator code.

# GitOps Scenario

- Multiple applications, owned by different teams, + ops and security
- Multiple environments (dev, prod) and clusters
- Multiple repos, organized differently




KubeCon CloudNativeCon North America 2025

Imagine you have multiple applications running in multiple environments.

Their helm charts, YAML files, and other configuration files are spread over multiple git repositories.

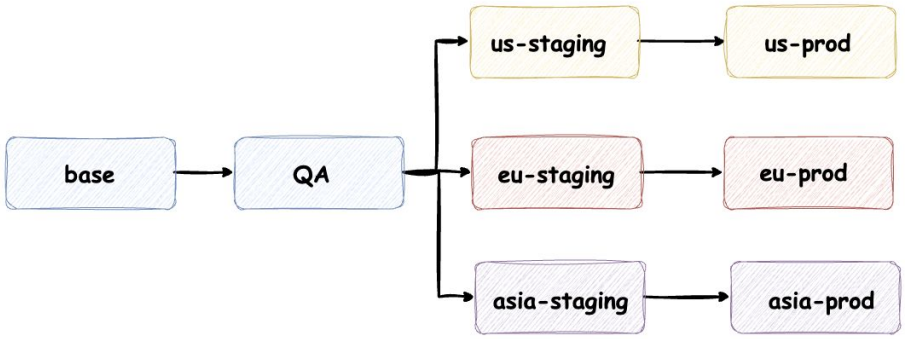
They are in different directories and are organized differently. Some are in repositories also containing application source code. Some are in repos containing configurations from multiple applications.

Here we have 3 sample applications. An AI chatbot, a voting application, and an online store.
(show the apps running quickly)

They have their own repos with their helm charts broken down into files in different ways.

Individual app teams know where their configuration source is, where the input values are for each environment, how to change environment variables, and so on. But what about operations and security personnel, who may be responsible for many applications? They may be less familiar with these details.

# Your app might look something like this




KubeCon CloudNativeCon North America 2025

May have multiple stages, multiple regions.

## What do you do when an **urgent** update is required?

Example causes:

- Container OOMs
- A new feature that must be disabled
- A security issue is found in a resource in the cluster


KubeCon CloudNativeCon North America 2025

Let's say something breaks due to an update or operational issue, or an application need to be updated urgently to address a security concern.

It could be:

- a recurring container OOM,
- A bug in a new feature that needs to be disabled, or
- A new security issue, to name a few cases.

## Operations People often say that GitOps is ‘hard to use’




NOTE that these are OPERATIONAL issues → this is why we don’t want to fight eg templates to get there.  Our message that tools like Helm are not “for operations”

**#1 GitOps complaint to me in 8 years has been from ops people who find using multiple dev tools to be high risk**

## Step 1: Find the configuration and input values

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
    meta.helm.sh/release-name: apptique
    meta.helm.sh/release-namespace: apptique
  creationTimestamp: "2025-10-07T19:35:46Z"
  generation: 1
  labels:
    app: emailservice
    app.kubernetes.io/managed-by: Helm
    helm.toolkit.fluxcd.io/name: apptique
    helm.toolkit.fluxcd.io/namespace: apptique
  name: emailservice
  namespace: apptique
...
```


If you want to update the configuration in git, first you need to find the templates and the right input values.

Helm adds some labels and annotations, but they don’t really help with that.

# Find the config+values: Follow GitOps breadcrumbs

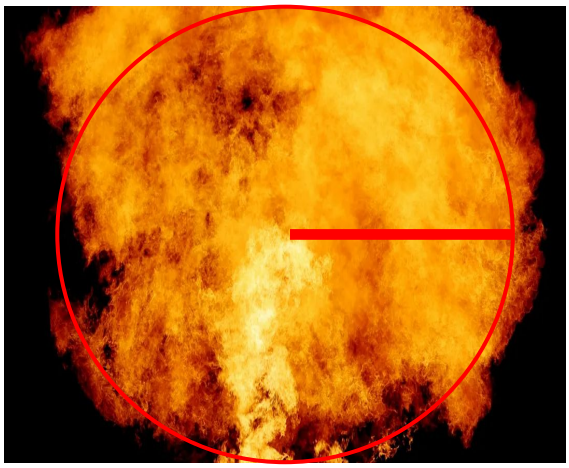
```
% flux trace -n apptique pod/emailservice-7d4b8cd7d6-5kw5h
Object:        Pod/emailservice-7d4b8cd7d6-5kw5h
Namespace:     apptique
Status:        Managed by Flux
...
---
HelmChart:     apptique-apptique
Namespace:     apptique
Chart:         ./helm-chart
Version:       *
Revision:      0.10.3+1
Status:        Last reconciled at 2025-10-07 12:35:46 -0700 PDT
Message:       packaged 'onlineboutique' chart with version '0.10.3+1' and merged values files [helm-chart/values.yaml helm-chart/values-dev.yaml]
---
GitRepository: apptique
Namespace:     apptique
URL:           https://github.com/confighub-kubecon-2025/apptique
Branch:        main
Revision:      main@sha1:addb49ce028439a00dea980f2c1ff30d38b8a662
Status:        Last reconciled at 2025-10-07 12:35:46 -0700 PDT
Message:       stored artifact for revision 'main@sha1:addb49ce028439a00dea980f2c1ff30d38b8a662'
```


One of the underappreciated benefits of GitOps is the ability to map resources in a cluster back to their configuration.

The Flux trace command can follow labels back to the corresponding Flux resources that identify where the Helm chart came from and where the values are in git.

## Step 2: Changing the right values → blast radius

```
...
      readinessProbe:
        periodSeconds: 5
        grpc:
          port: 8080
      livenessProbe:
        periodSeconds: 5
        grpc:
          port: 8080
...
```



KubeCon CloudNativeCon North America 2025

https://medium.com/itnext/modularity-and-blast-radius-in-infrastructure-as-code-9a1f6432196e

Next you need to change the right values for just the intended environments.

In simple cases, finding the right values to change isn't too hard.

But in some cases there are values overlays, value overrides on command lines or in GitOps configurations, post-rendering steps, or multiple tools chained together.

Or, if the values that need to be changed are hardcoded, like these probe settings, or unspecified, then the template needs to be changed, which can affect all environments.

The potential scope of impact of a change is sometimes called the "blast radius".

—------

From
https://github.com/confighub-kubecon-2025/apptique/blob/main/helm-chart/templates/emailservice.yaml#L92
https://medium.com/itnext/modularity-and-blast-radius-in-infrastructure-as-code-9a1f6432196e

## Changing values: Parameters taken to the extreme

```
{{- if .Values.probe.enabled }}
readinessProbe:
  httpGet:
    path: {{ .Values.probe.path }}
    port: {{ .Values.containerPort }}
    scheme: HTTP
  {{- with .Values.probe.settings }}
  {{- toYaml . | nindent 2 }}
  {{- end }}
{{- end }}
```

```
probe:
  enabled: true
  settings:
    periodSeconds: 30
```

https://github.com/gimlet-io/onechart/blob/master/charts/common/templates/_container.yaml#L31
https://www.youtube.com/watch?v=HzJ9ycX1h0c



To avoid that problem, a number of organizations preemptively parameterize every possible attribute.

In that case, figuring out which input values correspond to the values you want to change can sometimes be challenging.

For instance, some values inject snippets of YAML containing multiple fields, such as for these probe settings.

https://github.com/gimlet-io/onechart/blob/master/charts/common/templates/_container.yaml#L31
https://github.com/gimlet-io/onechart/blob/master/values.yaml#L26
https://www.youtube.com/watch?v=HzJ9ycX1h0c

## Changing values: Off-the-shelf charts

```
{{- $healthchecksPort := (default (.Values.ports.traefik).port .Values.deployment.healthchecksPort) }}
{{- $healthchecksHost := (default (.Values.ports.traefik).hostIP .Values.deployment.healthchecksHost) }}
{{- $healthchecksScheme := (default "HTTP" .Values.deployment.healthchecksScheme) }}
{{- $readinessPath := (default "/ping" .Values.deployment.readinessPath) }}
{{- $livenessPath := (default "/ping" .Values.deployment.livenessPath) }}
readinessProbe:
  httpGet:
    {{- with $healthchecksHost }}
    host: {{ . }}
    {{- end }}
    path: {{ $readinessPath }}
    port: {{ $healthchecksPort }}
    scheme: {{ $healthchecksScheme }}
  {{- toYaml .Values.readinessProbe | nindent 10 }}
```



https://github.com/traefik/traefik-helm-chart/blob/master/traefik/templates/_podtemplate.tpl#L83

Configurations for off-the-shelf components are often even more heavily parameterized.

This is an excerpt from the well maintained Traefik helm chart.

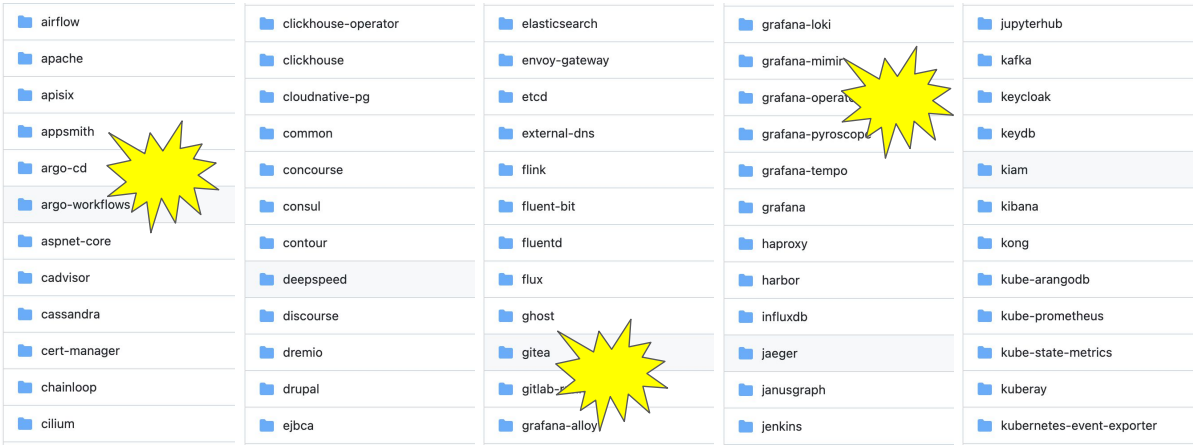
The chart has about 2000 lines of templates and another 3000 lines of default values and values schema.

We use Traefik for ingress, and for our use case the rendered configuration is only 300 lines of standard Kubernetes YAML.

Such Helm charts are installers that need to handle every scenario the authors want to support, whereas the rendered output is just for one scenario, so it's not surprising that it's much shorter and simpler.

https://github.com/traefik/traefik-helm-chart/blob/master/traefik/templates/_podtemplate.tpl#L83
https://itnext.io/i-shouldnt-have-to-read-installer-code-every-day-4dc1e5f9ee1a

## But we don’t want live changes to blow up other systems!




KubeCon CloudNativeCon North America 2025

Now imagine you have dozens or hundreds of applications in your organization, and each runs in a dozen or more environments with different input values for each environment.

You need to make a change to one component in one cluster. How do you do it?

Do you attempt to make a change through git and hope it works when it reaches the cluster, and that you don’t unintentionally change other clusters?

## Operations currently require “live” config changes

```
% flux suspend helmrelease emailservice
% kubectl edit -n apptique deployment emailservice
```

https://itnext.io/why-configuration-drift-is-so-hard-to-avoid-in-practice-443248cafc9c


A number of people have told me that they believe it’s not only faster and simpler but even safer to “break glass” and just change resources in the cluster directly.

If they use a GitOps tool, such as Flux or ArgoCD, they may need to suspend or disable it first and remember to re-enable it later.

If the change needs to be made permanent, then it needs to be backported to the configuration source.

You could try to ask an AI agent to do it, but you may need to provide very detailed instructions and review the results carefully.

## Rendered Manifest Pattern

### Chart + values

```
…
      image: {{ .Values.images.repository }}/{{
.Values.emailService.name }}:{{
.Values.images.tag | default .Chart.AppVersion }}
…
```

```
images:
  repository:
us-central1-docker.pkg.dev/google-samples/microse
rvices-demo
  tag: ""
…
# Override in prod
images:
  tag: "v0.10.3"
…
```

### Rendered manifest

```
…
      image:
us-central1-docker.pkg.dev/google-samples/
microservices-demo/emailservice:v0.10.3
…
```

KubeCon CloudNativeCon North America 2025

https://github.com/confighub-kubecon-2025/apptique/blob/main/helm-chart/templates/emailservice.yaml#L72

The difficulty in determining whether a change is correct in a complex configuration is what has led some people to render the configuration in advance.

That makes the configuration more straightforward to review and also ready to run through linting and policy tools like kubeconform and kyverno.

Instead of needing to understand templates and values …

You could just review the Kubernetes resources with all of the desired values in place.

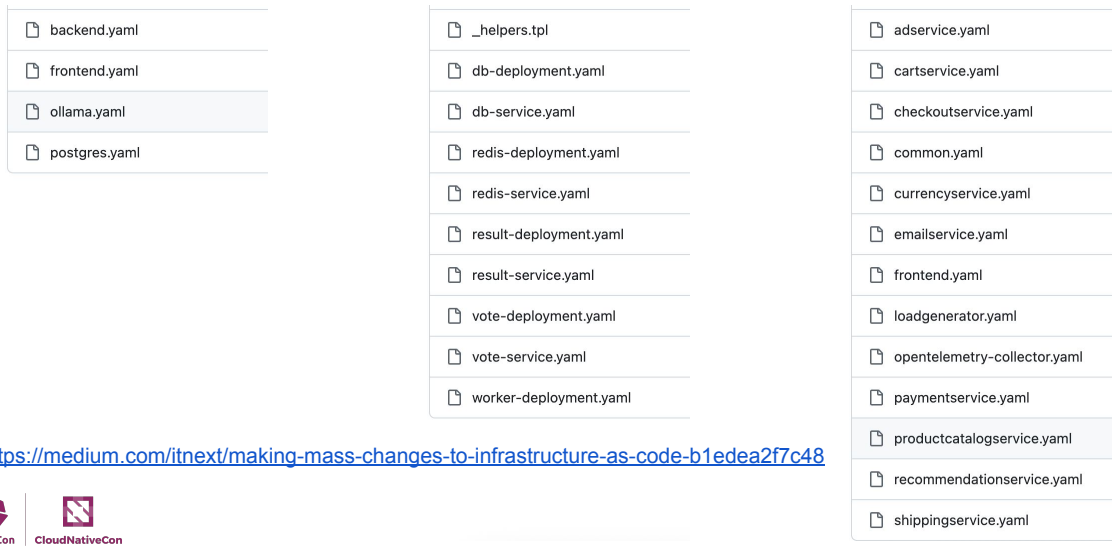
That can increase the confidence in changes.

But doesn't make changes any faster.

Rendering the manifests can even slow down the process due to bottlenecks in CI workers.

https://github.com/confighub-kubecon-2025/apptique/blob/main/helm-chart/templates/emailservice.yaml#L72
https://akuity.io/blog/the-rendered-manifests-pattern

## Another challenge: Mass changes




https://medium.com/itnext/making-mass-changes-to-infrastructure-as-code-b1edea2f7c48



The rendered manifest pattern also doesn't help make similar changes across many applications.

Because of the syntactic differences in helm charts and sprawl across multiple repositories, that can't generally be accomplished by a simple search and replace.

Even this simple example has 18 YAML files with Deployments. If more than a couple of them need to be changed, that's already enough changes to be error prone.

## DRY vs WET

**DRY (Don't Repeat Yourself) Configuration generators**

- Helm
- Kustomize
- CDK8s
- Jsonnet
- CUElang
- ...

**WET (Write Every Time) Configuration**

- Plain YAML
- Porch (Nephio)
- ConfigHub

https://itnext.io/variant-generation-is-the-primary-capability-of-infrastructure-as-code-773e6b404b49


Helm charts, Kustomize overlays, and CDK code are sometimes called DRY formats, or Don't Repeat Yourself.

Rendered manifests are plain YAML, which is sometimes called WET configuration, for Write Every Time.

DRY formats enable generation of multiple similar configurations, which we call variants. For example, you probably have slightly different variants of your application's deployment configuration for each environment.

Configuration generators can make configuration for each variant appear more concise on the surface, but the generator needs to support all desired variants, so it is more complicated than any individual variant.

This complexity can make understanding the configuration more difficult and changing the configuration less straightforward.

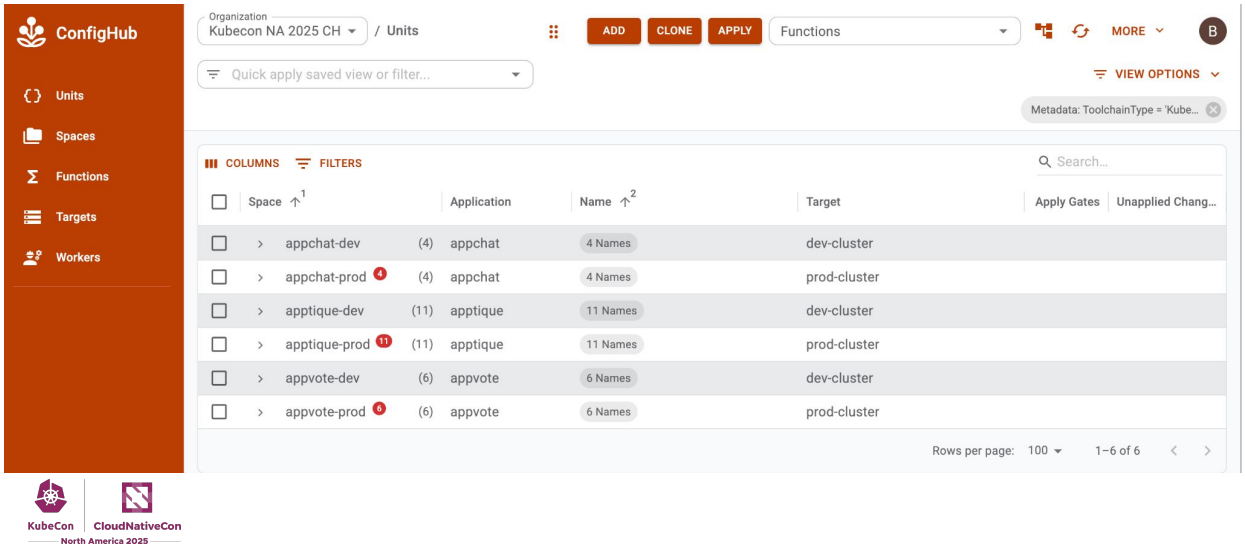
To make these kinds of operational and security changes easier, we propose that the solution is to use WET configuration, such as plain YAML.

Since Infrastructure as Code has been considered a best practice for decades, that

probably seems counterintuitive, so let's see what this looks like.

You could perhaps use Porch, which part of the Nephio project.  But we're going to use ConfigHub.

# ConfigHub Demo



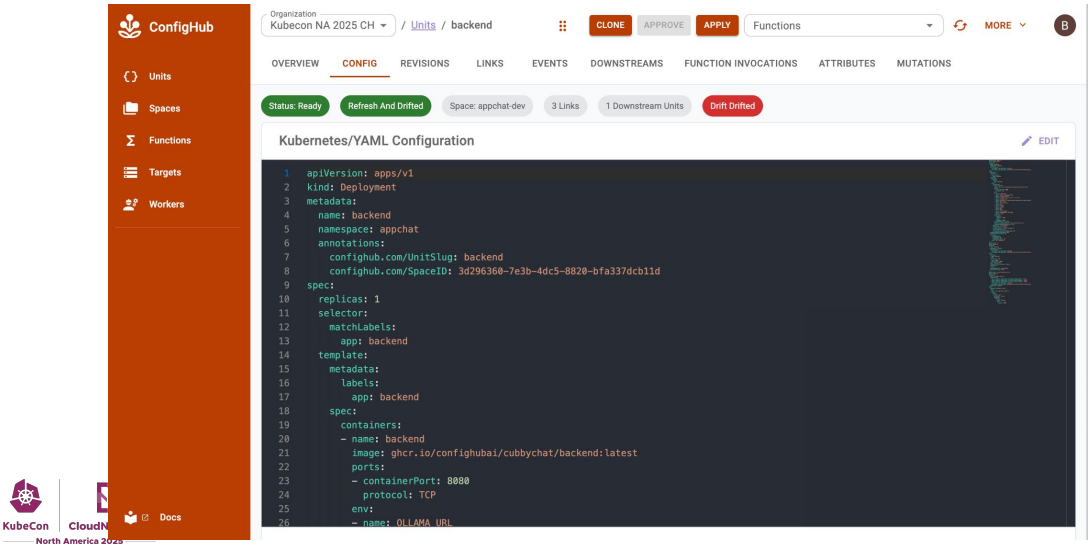

ConfigHub stores configuration data as objects in a database instead of storing configuration generation code in files in git.

These are the 3 applications I showed earlier, deployed into the two environments, dev and prod.

In ConfigHub, there's a 1:1 correspondence between live resources and configuration data in ConfigHub. There is one copy corresponding to each live instance.

It's similar to the rendered manifest pattern, but it's always fully rendered.

# Programmatic changes, queries, validation




Let's revisit the break-glass scenarios I mentioned earlier.

To make a change through ConfigHub, I need to find the corresponding Unit. As with GitOps, I can trace a resource back to its configuration source.

cub k8s source --kubeconfig prod.kubeconfig Deployment backend -n appchat

Unlike typical GitOps, the configuration is plain Kubernetes YAML.

That enables it to be modified programmatically, so we can make changes precisely and quickly.

In the case of the container OOM, I need to change the container's resources. I don't have to change them by hand. I can use a tool to do it that validates the inputs and changes or inserts the correct paths. I could invoke it via a UI, CLI, or API.
(set-container-resources backend floor 100m 200Mi 2)

In the case of a feature that need to be disabled, I could change an environment variable.
(set-env-var backend EXPERIMENTAL_FEATURE false)

If someone discovered inadequate Security settings, I could set the securityContext to recommended values automatically.
(set-pod-defaults –security-context)

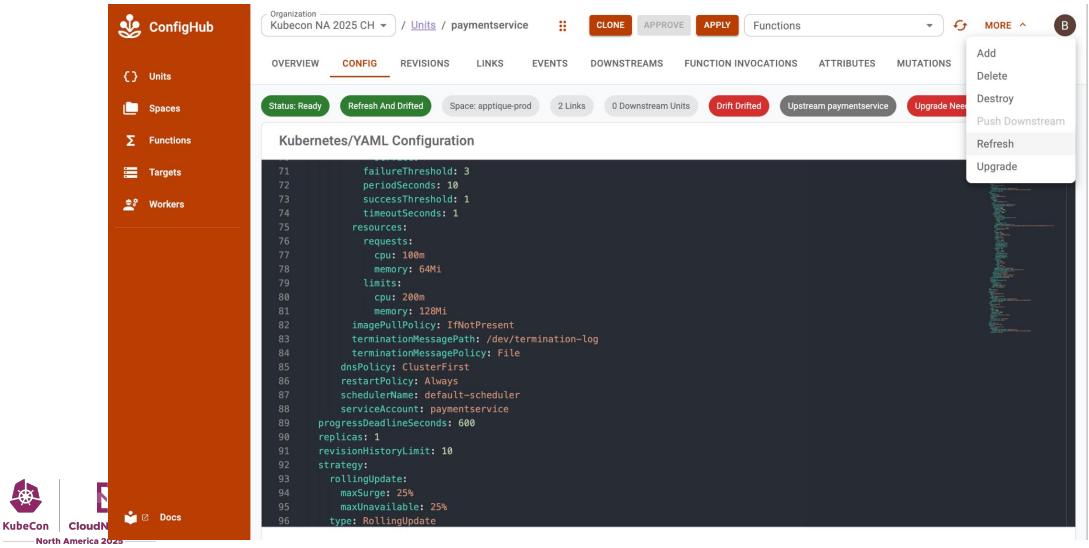
Not only that, but I can find all resources that don't have the values set appropriately and change them all as easily as I can change one.
(filter by run-as-root, select all, and run set-pod-defaults –security-context on them)

This doesn't get deployed automatically. Teams have an opportunity to review the changes. You could think of it as an open pull request.

I can also use something like Kubernetes dynamic admission control to ensure no new instances with the wrong settings are deployed.
(enable the Trigger, make a change to the unit, and show the apply gate)

The same mechanism can be used to automatically run kubeconform or other linting and validation mechanisms to continuously validate the configuration.
(add bogusPodField to a pod, switch to dashboard, run valid k8s trigger and show the error message.)

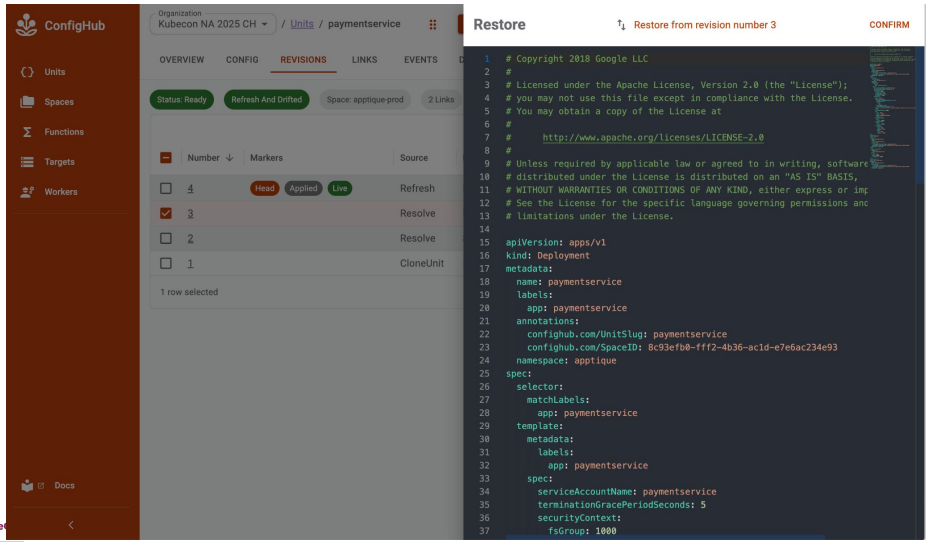
# Bidirectional GitOps




The 1:1 correspondence between ConfigHub and the live state not only makes changes more straightforward with a more contained direct blast radius, but it also enables us to bidirectionally synchronize with the state in the cluster – bidirectional GitOps.

1. kubectl --kubeconfig prod.kubeconfig edit deploy -n apptique paymentservice
    a. Double cpu and memory
2. cub unit refresh --space apptique-prod paymentservice
3. cub k8s source --kubeconfig prod.kubeconfig Deployment paymentservice -n apptique

# Decide which revision to keep




KubeCon CloudNative North America 2025

Although the configuration data isn't stored in git, there's still change history.

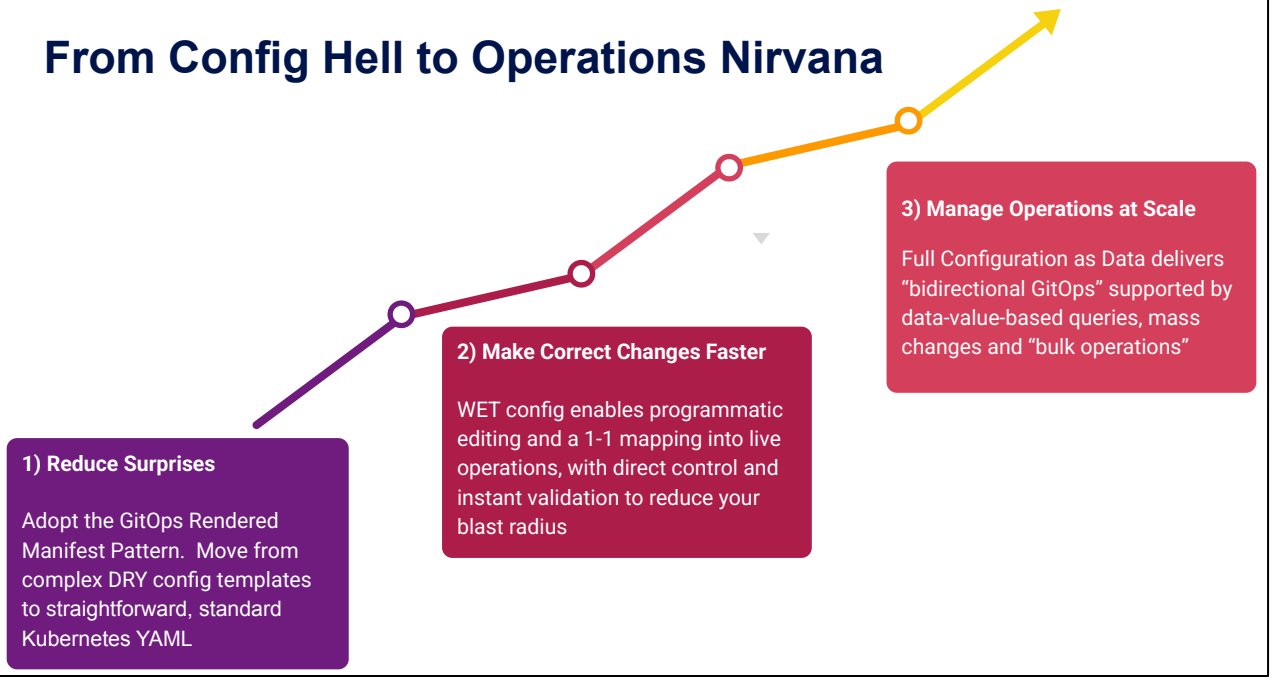
If you didn't want to keep that change, you could restore the prior revision and apply again. (restore it).

But now you have the option to keep the live changes rather than spending a lot of time integrating them back into your values and templates.

# Configuration is Data


We might say "so what's the takeaway from all this?  well we want you to focus on one thing and that is: configuration is data".

# From Config Hell to Operations Nirvana



**1) Reduce Surprises**

Adopt the GitOps Rendered Manifest Pattern.  Move from complex DRY config templates to straightforward, standard Kubernetes YAML

**2) Make Correct Changes Faster**

WET config enables programmatic editing and a 1-1 mapping into live operations, with direct control and instant validation to reduce your blast radius

**3) Manage Operations at Scale**

Full Configuration as Data delivers “bidirectional GitOps” supported by data-value-based queries, mass changes and “bulk operations”

# Config as data: See what will happen before it happens

Speed up predictable changes at critical moments

\+ Reduce config sprawl from systems, repos, files, formats

= Operational happiness!

**Try the demo**

http://docs.confighub.com

https://github.com/orgs/confighub-kubecon-2025/repositories


KubeCon
CloudNativeCon
North America 2025
Try it out:
https://www.confighub.com/
Look at examples:
https://docs.confighub.com/examples/
Contact us:
hello@confighub.com