Stop Writing Operators

KubeCon | CloudNativeCon
North America 2020
Virtual

Joe Thompson

Senior Solutions Engineer, HashiCorp

Stop Writing Operators

by default ^

Joe Thompson

Senior Solutions Engineer, HashiCorp

In IT since my first job helping out with computers in my high school in 1994

Past employers: Mesosphere, Capital One, CoreOS, Red Hat, among others

Exposed to Kubernetes in early 2015 and working with it full-time since late 2015

Currently a Solutions Engineer for HashiCorp (we're hiring!)

Pronouns: he/him
Blood type: Caffeine-positive

Contact info:
- jthompson@hashicorp.com
- Twitter: @caffeinepresent
- Kubernetes Slack: @kensey
- LinkedIn

Everybody's thinking it...

28 Jun 2020

Talk submitted

➡

12 Sep 2020

Devan Goodwin: When Not to Write a Kubernetes Operator

➡

22 Sep 2020

The New Stack: Kubernetes: When to Use, and When to Avoid, the Operator Pattern

➡

29 Sep 2020

Talk accepted!

Pre-Operator Checklist

- Should this operator be written at all?
- Should I write it?
- What functions should my operator perform?
- What API permissions will my operator need to perform those functions?
- What framework should I write my operator in?

Injunction vs. information

Are you serious?

- Pretty serious

- Not like, we-can't-be-friends-unless serious though

Are you saying operators are bad?

- No! Operators are good!

...in the proper places

What an operator is and isn't

Operators vs. controllers

2016 -- CoreOS: Introducing Operators: Putting Operational Knowledge into Software

"...application-specific operational knowledge encoded into software that leverages the powerful Kubernetes abstractions to run and manage the application correctly..."

Meant for managing stateful apps

- All operators are controllers but NOT all controllers are operators!
- A controller that does nothing stateful is just a controller (and that's OK)

Deciding if you want an operator

Kubernetes
Your operator
Your app

Case #1: No state needs managing

No statefulness

- or-

Kubernetes has all the necessary state handling

Case #2: Apps you maintain

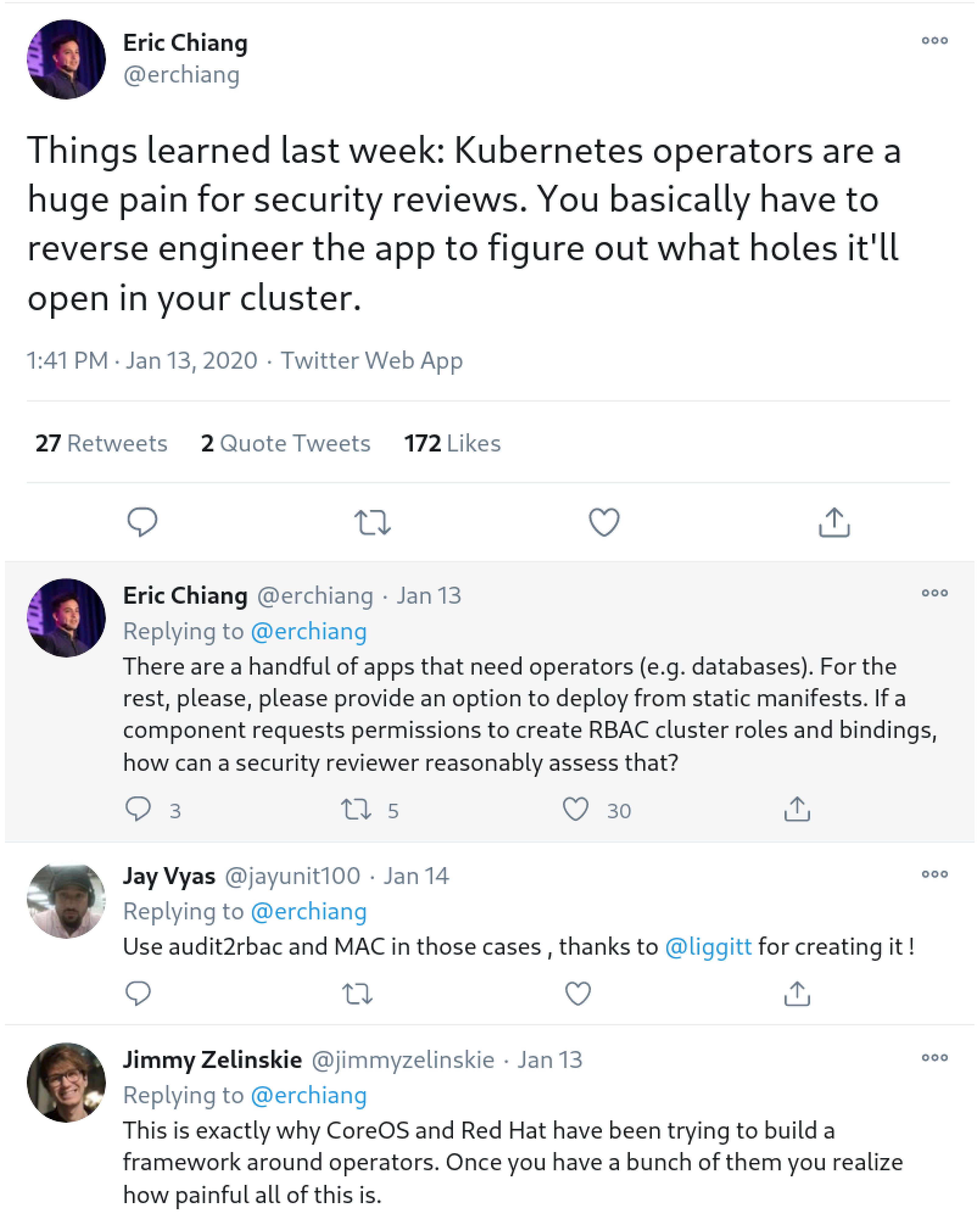
Creating an operator is over-abstracting the problem

You now have an extra component susceptible to failure

Case #2: Apps you maintain

Isn't an operator just another microservice?

- No: microservices remain abstracted from one another. Operators inherently don't.



Eric Chiang
@erchiang

Things learned last week: Kubernetes operators are a huge pain for security reviews. You basically have to reverse engineer the app to figure out what holes it'll open in your cluster.

1:41 PM · Jan 13, 2020 · Twitter Web App

27 Retweets 2 Quote Tweets 172 Likes

Eric Chiang @erchiang · Jan 13
Replying to @erchiang
There are a handful of apps that need operators (e.g. databases). For the rest, please, please provide an option to deploy from static manifests. If a component requests permissions to create RBAC cluster roles and bindings, how can a security reviewer reasonably assess that?
3 5 30

Jay Vyas @jayunit100 · Jan 14
Replying to @erchiang
Use audit2rbac and MAC in those cases , thanks to @liggitt for creating it !

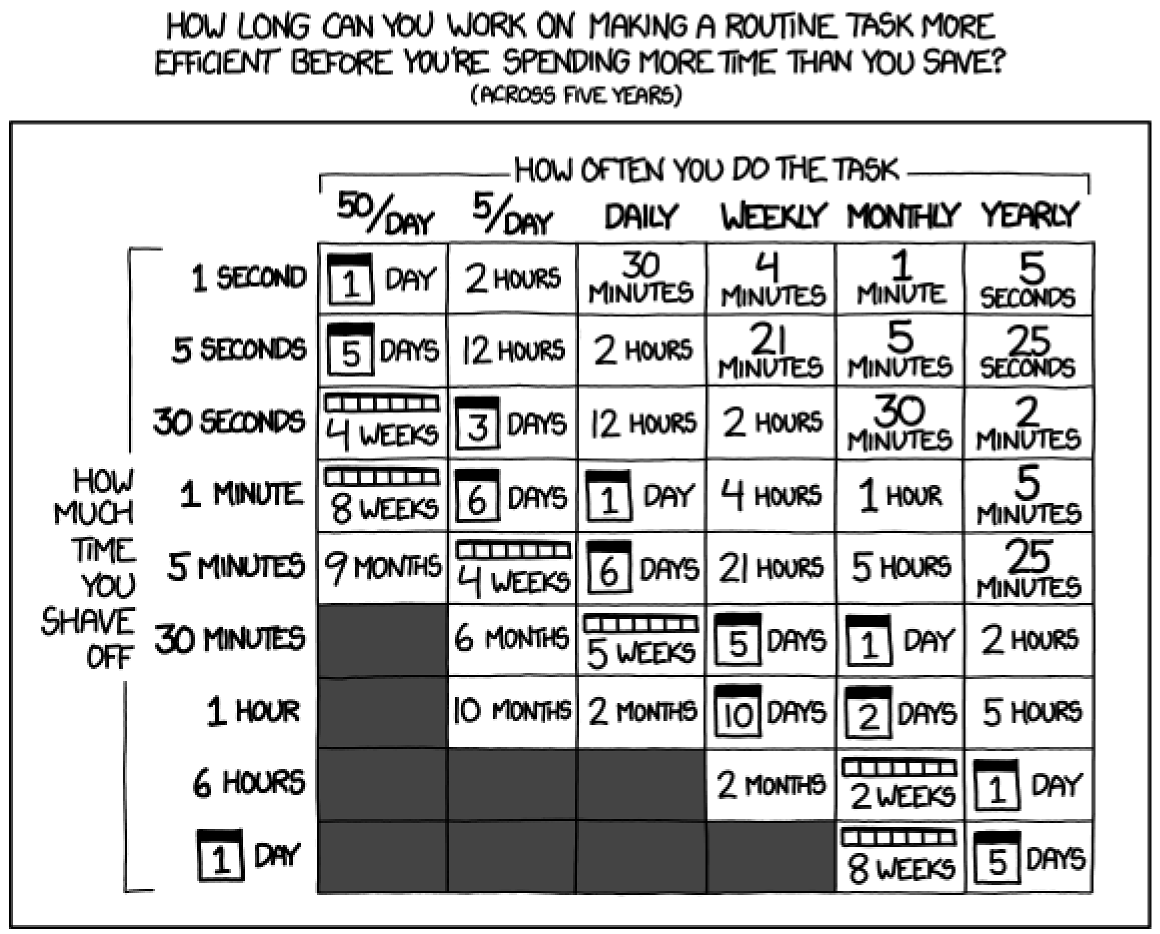
Jimmy Zelinskie @jimmyzelinskie · Jan 13
Replying to @erchiang
This is exactly why CoreOS and Red Hat have been trying to build a framework around operators. Once you have a bunch of them you realize how painful all of this is.

Operators (and CRD controllers) make your security burden heavier

Alternatives to operators

CRD controller

Not trying to manage state at all is easier to do safely



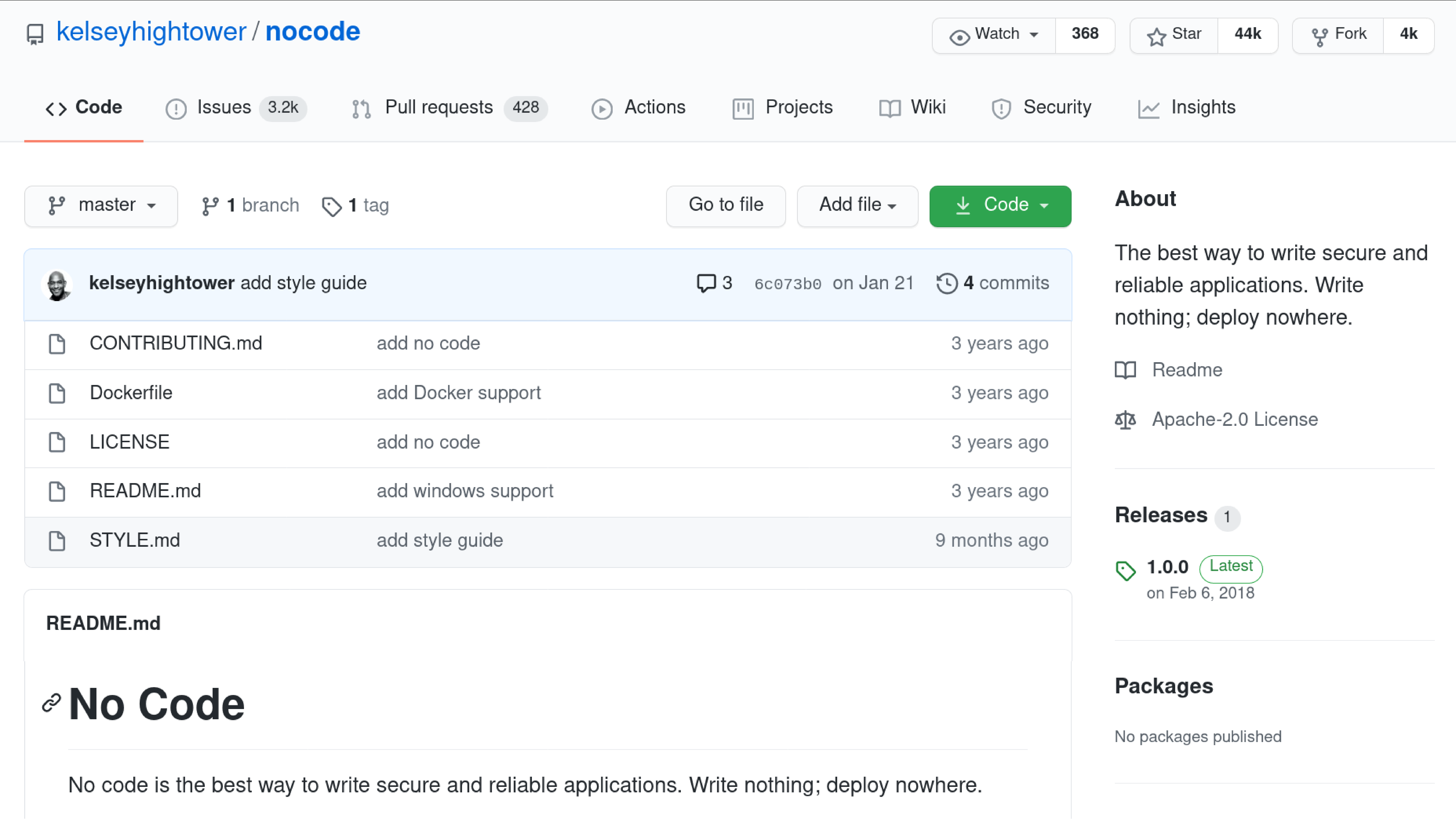
HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

HOW MUCH TIME YOU SHAVE OFF	HOW OFTEN YOU DO THE TASK: 50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES	■	6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR	■	10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS	■	■	■	2 MONTHS	2 WEEKS	1 DAY
1 DAY	■	■	■	■	8 WEEKS	5 DAYS

(source: XKCD #1205)

Do nothing

- Automation isn't worth it if you aren't saving time



kelseyhightower / nocode

Watch 368 · Star 44k · Fork 4k

Code · Issues 3.2k · Pull requests 428 · Actions · Projects · Wiki · Security · Insights

master · 1 branch · 1 tag · Go to file · Add file · Code

kelseyhightower add style guide · 3 · 6c073b0 on Jan 21 · 4 commits

CONTRIBUTING.md	add no code	3 years ago
Dockerfile	add Docker support	3 years ago
LICENSE	add no code	3 years ago
README.md	add windows support	3 years ago
STYLE.md	add style guide	9 months ago

README.md

No Code

No code is the best way to write secure and reliable applications. Write nothing; deploy nowhere.

About

The best way to write secure and reliable applications. Write nothing; deploy nowhere.

Readme

Apache-2.0 License

Releases 1

1.0.0 Latest
on Feb 6, 2018

Packages

No packages published

AUGUST 07, 2017

Gitops | Kubernetes | Product features

GitOps - Operations by Pull Request

At Weaveworks, developers are responsible for operating our Weave Cloud SaaS. "GitOps" is our name for how we use developer tooling to drive operations. This post talks about GitOps, which is 90% best practices and 10% cool new stuff that we needed to build. Fair warning: this is what works for us, and dear reader, you may disagree.

Git is a part of every developer's toolkit. Using the practices outlined in this post, our developers operate Kubernetes via Git. In fact, we manage and monitor all of our applications and the whole 'cloud native stack' using GitOps. It feels natural and less intimidating to learn, and the tools themselves are very simple.

Git as the Source of Truth

For the last two years, we've been running multiple Kubernetes clusters and Prometheus telemetry databases on Amazon Web Services. You can read more about how we provision Kubernetes in the blog post, "Provisioning And Lifecycle Of A Production Ready Kubernetes Cluster".

What exactly is GitOps? By using Git as our source of truth, we can operate almost everything. For example, version control, history, peer review, and rollback happen through Git without needing to poke around with tools like kubectl.

- Our provisioning of AWS resources and deployment of k8s is declarative
- Our entire system state is under version control and described in a single Git repository
- Operational changes are made by pull request (plus build & release pipelines)
- Diff tools detect any divergence and notify us via Slack alerts; and sync tools enable convergence
- Rollback and audit logs are also provided via Git

No-code operator frameworks

Top-level controller implements the management primitives

You write declarative config to implement your operator logic

Run outside the cluster

No longer limited in scale or architecture by needing to fit into the cluster

- But you will still have to manage RBAC, and handle authentication

When operators are needed

Case #1: Someone else's app

…and its state *has* to be managed inside Kubernetes

Definitely not ideal

Engage the app maintainers as early and often as possible

- They might even be willing to take on smaller state-handling feature requests

Case #2: Compatibility shims

Your app can take fuller advantage of Kubernetes features

- The operator provides state handling to run your app in older versions of k8s

- The operator provides functions for your app that have not landed in Kubernetes yet

This is fine on a *temporary* basis

You need an operator… but maybe you shouldn't write it

Case #1: You don't have a map yet

You should not write an operator for things until:

- you've spent as long *recovering from failure or dealing with lack of management*

- as it would reasonably take you to write the operator

Failure is where we learn

- Learning from failure is even more critical when we automate

What about routine operations?

Is that all you're doing?

- If so, look at some of the non-operator methods of handling these tasks first

Case #2: 0.x app versions

An operator with no forward compatibility assurance is a time sink waiting to happen

Don't assume 1.x means SemVer-level stability is in effect

Case #3: Did you talk to the owner?

The app owner knows their app better than anyone

…in fact, they might already be writing what you need

I'm writing an operator, now what?

Pre-Operator Checklist (continued)

- Understand CRDs and Controllers
- Get started looking at the operator landscape

Choosing to use a framework

- Can you write in one of your preferred languages?
- Does its model fit both into your environment and into your head?
- Does the framework maintainer provide framework-related services you want to use?

Final thoughts

When you write that operator:

- Maintain loose coupling
- Remember, no code is often better
- Write what you know (then stop)

More info

References:

Devan Goodwin: When Not to Write a Kubernetes Operator

The New Stack: Kubernetes: When to Use, and When to Avoid, the Operator Pattern

CoreOS: Introducing Operators: Putting Operational Knowledge into Software

WeaveWorks: GitOps - Operations by Pull Request

Kubernetes: Custom Resources, Operator Pattern

Further reading:

WeaveWorks: What Is GitOps

Matt Butcher: Is there a Helm and Operators showdown?

THANK YOU!

https://bitly.com/2HuBJlE+