

Gerred Dillon

Staff Software Engineer & KUDO Product Owner, D2iQ

Website: <https://kudo.dev>

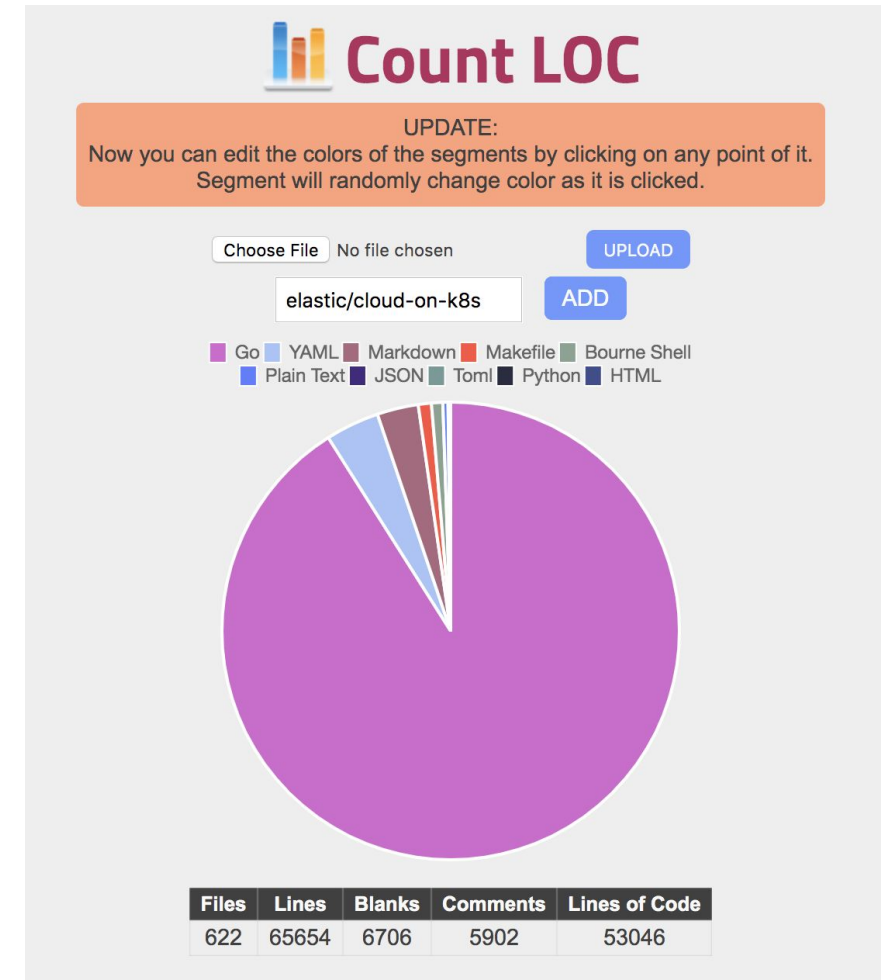
Git Repo: <https://github.com/kudobuilder/kudo>

Slack: <https://kubernetes.slack.com/messages/kudo>

Demo: <https://www.youtube.com/watch?v=MIJG3sPXf08>

Operator Development Challenges

- Advanced Kubernetes and distributed systems knowledge required
 - Distributed systems programming in Go
 - Challenging for software vendors to hire these skills
- Building a good Operator takes 10k+ lines of code and many months
- Code duplication between Operators
- No easy integration with CNCF ecosystem tools
- High maintenance burden



Example: Elastic Operator is ~53k LOC

Operator User Challenges

- Running stateful workloads on Kubernetes is still too complicated for many users
- Different operators have different workflows and APIs
- Inconsistent debugging tools
- Controller sprawl



Kelsey Hightower ✓

@kelseyhightower

Following



I'm always going to recommend people exercise extreme caution when running stateful workloads on Kubernetes. Most people who are asking "can I run stateful workloads on Kubernetes" don't have much experience with Kubernetes and often times the workload they are asking about.

3:10 AM - 24 Mar 2019

286 Retweets 901 Likes



What is KUDO?

A **toolkit and runtime** for building operators optimized for complex, stateful applications.

Increases developer productivity when **building operators**.

Increases operator productivity when **operating services**.

Why KUDO?

Building operators is hard

This requires a deep understanding of Kubernetes, API development, Operator API development, and a language that fits into the K8s environment.

Maintaining operators is hard

Keeping an operator implementation up-to-date with the latest K8s version is a non-trivial effort and, again, requires a deep understanding of the changes going on in that space.

Operating services is hard

No existing solution provides easy to customize means of operating stateful services in production

There's value in making all of that easier.

How KUDO Helps Developers

- KUDO provides abstractions for sequencing lifecycle operations using Kubernetes objects and “plans” - conceptually similar to runbooks
- Reduces boilerplate and code duplication between Operators
- Reduces the number of controllers in a cluster
- Provides an extension mechanism to create “flavors” of a base Operator for customization to a user’s environment
- Provides ISVs with a tool to ship best practices for day 2 operations alongside their software
- Ships with testing tool to enable TDD of Kubernetes resources

How KUDO Helps Users

- KUDO provides the `kubectl kudo` plugin to deploy, debug, and manage workloads.
- It is common to deploy multiple Operators on a cluster. KUDO provides a similar API/CLI/workflow experience for all.
- All workloads are managed as Custom Resource Definitions (CRDs) in Kubernetes, allowing users to manage workloads via standard GitOps tooling
- Existing operators can be managed by KUDO, natively understanding how to deploy CRDs, custom resources, and other operators, enabling these dependencies to be part of other workloads
- (Upcoming) Centralized supportability, metrics/alerting, and security/RBAC features, enabling KUDO use for enterprise platform teams

```
Zwerg:kudo tobi$ kubectl kudo plan status --instance kafka
Plan(s) for "kafka-rvf9wq" in namespace "default":
├─ kafka-rvf9wq (Operator-Version: "kafka-0.2.0" Active-Phase: "deploy")
│  └─ Plan deploy (serial strategy) [IN_PROGRESS]
│     └─ Phase deploy-kafka (serial strategy) [IN_PROGRESS]
│        └─ Step deploy (IN_PROGRESS)
└─ Plan not-allowed (serial strategy) [NOT ACTIVE]
   └─ Phase not-allowed (serial strategy) [NOT ACTIVE]
      └─ Step not-allowed (serial strategy) [NOT ACTIVE]
         └─ not-allowed [NOT ACTIVE]
```

Lifecycle Orchestration with KUDO

```
plans:  
  deploy:  
    strategy: serial  
    phases:  
      - name: deploy-master  
        strategy: parallel  
        steps:  
          - name: deploy-master  
            tasks:  
              - deploy-master  
      - name: deploy-data  
        strategy: parallel  
        steps:  
          - name: deploy-data  
            tasks:  
              - deploy-data  
      - name: deploy-coordinator  
        strategy: parallel  
        steps:  
          - name: deploy-coordinator  
            tasks:  
              - deploy-coordinator  
      - name: deploy-ingest  
        strategy: parallel  
        steps:  
          - name: deploy-ingest  
            tasks:  
              - deploy-ingest
```

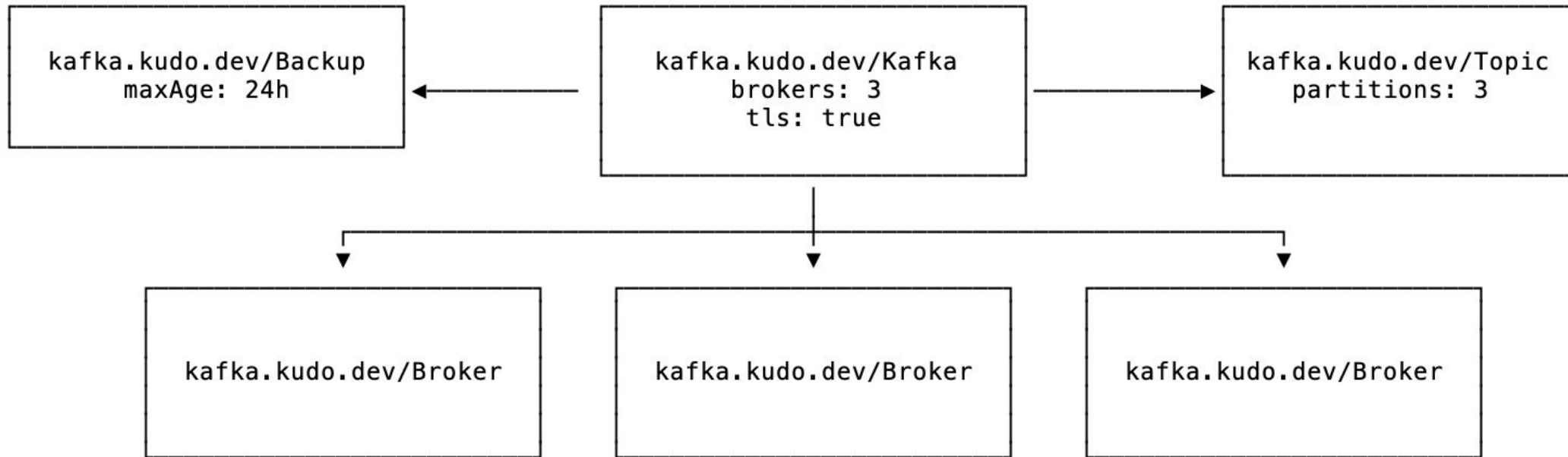
Plans: define lifecycle operations such as deploy, backup, restore, ...

Phases: grouping mechanism for tasks
Strategy: defines the execution order of the phases

Steps: each step consists of one or more tasks
Strategy: defines the execution order of the steps

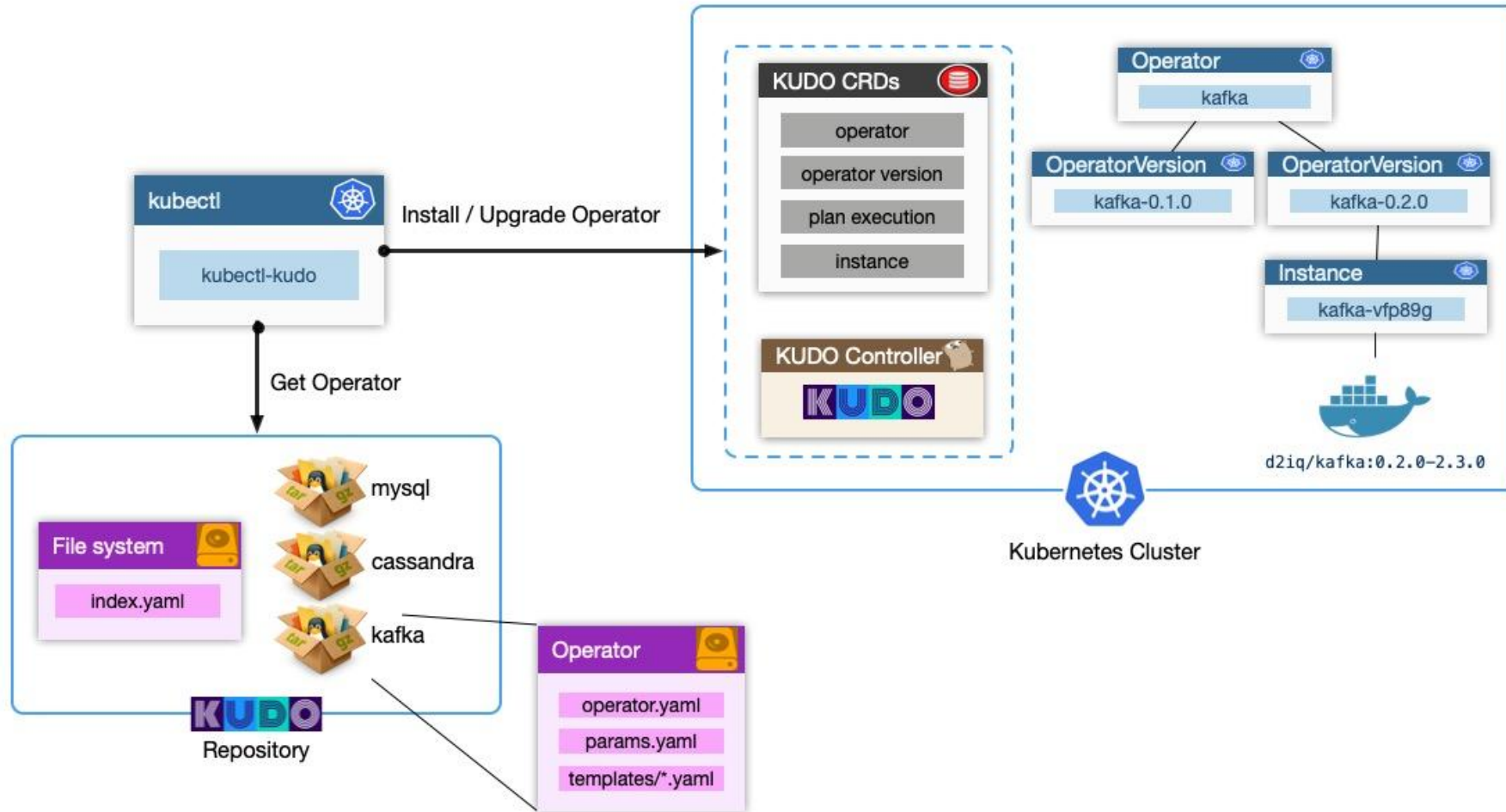
Tasks: each task creates a Kubernetes object from a templated manifest

Lifecycle Orchestration with KUDO - Next



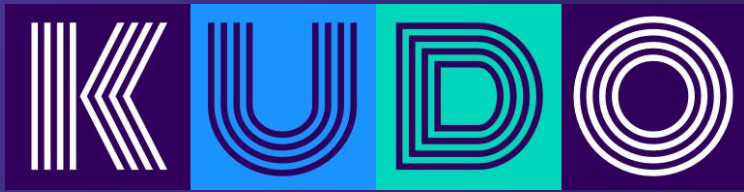
- Operator developer declares a component that they want to operate on
 - Writes create, update, upgrade, delete, and custom plans for each component
 - Writes CLI extensions for each component (optional)
 - e.g.: **kubectl kudo kafka topic add analytics-ingest --partitions=3**
- KUDO converts operator definition into a set of CRDs
- User installs operator, gets custom CLI, CRDs
- Application components and topology (Kafka Topics, MySQL Tables, etc) now are CRDs, and can be treated like any other Kubernetes resource (backup, GitOps, RBAC, etc)
- Leaky abstractions can still be imperative (Restores are hard to make declarative)

KUDO Architecture



KUDO Community

- Open Governance model based around Kubernetes Enhancement Proposals (KEPs)
- Focus on ease of contribution to core codebase and operators
- 4 reference operators, multiple community operators (including Elastic) in progress
- 0.7.5 with minor releases every 2 to 4 weeks
- 326 Github Stars
- Multiple organizations interested in building operators using KUDO
- 57 contributors with 6 working full time on KUDO



compared to...*

KUDO vs. Operator SDK / Kubebuilder

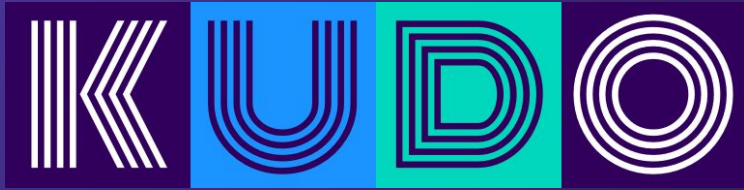
- Polymorphic controller, supports running multiple types of operators
- Configured via CRDs, extensible via Webhooks
- Oriented towards using existing clients and tooling for software rather than re-building ops-related functionality in Go
- Build operators using Kubernetes primitives rather than software development in Go

KUDO vs. Metacontroller

- Both are polymorphic controllers for operators
- KUDO manages CRDs for you and is an “operator” for dynamic CRDs as well as software
- KUDO supports dependencies and has dependency resolution for other operators. Existing Metacontroller operators force you to bring your own dependencies to the table
- KUDO allows sequencing of your application during different lifecycles
- KUDO supports custom plans and day 2 operations for software

KUDO vs. Helm (w/ Tiller)

- Controller managing not just the “deploy” step, but also day 2 operations
- Drift detection, repair, and alerting (WIP)
- Sequencing of steps for complicated lifecycles
- “Higher level” features for supportability - get logs, configuration files, metrics from a running instance
- Automatic sandboxing of instances. Working on dynamic KUDO RBAC, possible OPA and other policy integrations to dynamically sandbox KUDO itself.



in the Wild

Community Operators

- Zookeeper
- Kafka
- MySQL
- Elastic
- More coming soon

KUDO Ecosystem

- Builds on other OSS projects: kubebuilder, controller-runtime
- Users can extend (progressively enhance) existing Helm charts and CNAB bundles with orchestration provided by KUDO (under development)
- KUDO does not aim to solve the “application definition” problem, instead focusing on lifecycle and day 2 operations. “I have my application deployed, what now?”
- Future versions of KUDO are focused on enabling developers to write application-aware declarative operations and workflows around stateful applications (Backups, restores, scaling, disaster recovery)
- Package ecosystem that enables users to
- Community governance model based on the Kubernetes process, built from the start to easily be vendor neutral

KUDO Roadmap

Dynamic CRDs: Dynamic CRDs provide the ability for KUDO to manage the lifecycle of operator CRDs for the operator developers and users. This will enable more expressiveness around declarative components and day 2 operations.

Application Aware Operations: KUDO already provides for, and will be expanding the ability for operators to perform operations such as backups, restores, scaling, upgrades, and application-specific tasks in an application-aware way.

Alternate Definition Formats: Evaluating CUE and Starlark as high level, opinionated alternatives to writing operators. Currently collecting user data on the needs.


Operator Dependencies: Ability for KUDO to support a wide range of dependencies (from existing instances and connection strings to entirely new dependencies that are KUDO managed), and for tighter control of dependency specification by operator developers.

Operator Extensions: Operator extensions provide a mechanism for developers to extend from other formats such as other KUDO operators, Helm charts, or CNAB bundles without forking an operator. This enables operators to be enhanced for specific environments, such as Kubernetes distributions, monitoring software, or service meshes without multiple maintained versions of the same base tech.

Airgap Support: KUDO will soon support deploying itself and operators in airgapped environments.

Supportability: Features such as robust support bundles, metrics, and alerting that allows operator developers to define how their applications should be monitored and alerted upon by default.

Get Involved

- Install: <https://kudo.dev> and try it out!
- Give Feedback / Open Issues (and check out our **Hacktoberfest**  label!):
 - <https://github.com/kudobuilder/kudo>
 - <https://github.com/kudobuilder/operators>
- **#kudo** channel on the Kubernetes Slack (<https://slack.k8s.io>)