Suture: Stitching Safety onto Kubernetes Operators

Akshat Mahajan
Brown University
akshat_mahajan@brown.edu

Theophilus A. Benson
Brown University
tab@cs.brown.edu

Abstract

Kubernetes operators allow custom automation for applications to be packaged with the application in a cluster-agnostic manner. This unique property eliminates the need for in-house operational expertise with the application — such domain knowledge, encoded once, can be distributed to any environment — but requires trusting the operator to run arbitrary actions across an entire cluster. Little is known about the security or reliability implications of this paradigm. We present results from a survey of 54 Kubernetes developers and an analysis of 215 feature requests against 19 operator repositories demonstrating the ways users have experienced nontrivial safety issues with operators. We further propose the development of Suture, an access-control mechanism that seeks to prevent the majority of these safety issues with operators.

ACM Reference Format:
Akshat Mahajan and Theophilus A. Benson. 2020. Suture: Stitching Safety onto Kubernetes Operators. In *Student Workshop (CoNEXT'20), December 1, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3426746.3434055

1 Introduction

Managing an application in production brings with it infrastructure and operations challenges, ranging from the routine (e.g. backup scheduling, container deployment and traffic management) to the complex (auto-scaling resources, leader election, datacenter failover, etc.). Solving these in an automated fashion typically requires writing infrastructure code unique to the environment and the application. But subtle environmental differences (the use of a different kernel, for example, or differing network policies) prevent deploying one-size-fits-all solutions. Operations automation is further complicated in a microservices context, where standardization on toolchains, dependency versions and runtime environments may not be enforced across microservices, or where automation to preserve links *between* vastly different applications (e.g. service discovery) may be needed.

The Kubernetes operator paradigm[4] is a promising attempt to solve writing generic automation solutions across Kubernetes clusters. It allows operations teams to write custom scripts (called *controllers*) that react to different events monitored by the cluster, and apply actions against Kubernetes abstractions for hardware and software resources. These so-called "operators" can set default security settings, perform autoscaling in response to system pressure, resolve leader election issues for distributed worker services, generate application-specific secrets, and carry out numerous tasks that previously required domain knowledge from cluster administrators. Written once, operators for individual microservices relieve the need for a common platform team that has to know how every service works in depth.

Unfortunately, the operator paradigm is currently imperfect. In August and September 2020, our group conducted a diverse survey of 54 Kubernetes operator users across industry (divided roughly equally between startups, small-to-medium businesses, enterprise and Fortune 500 companies) and analyzed 215 feature requests filed against 19 open-source operators written by third-party software vendors. Our work in this space led us to identify three issues that threaten safety:

Lack of configuration safety. 47% of our survey takers reported providing operators configuration that led to unwanted consequences by accident, including deprovisioning of containers/services and even user data loss (an event one-sixth of these specific respondents were victim to).

Lack of transparency. Just 58% of all users surveyed felt they had the tools and intuition to understand how their operators would behave under unusual situations (such as network hijacks, split brain scenarios with databases or datacenter loss). Several users highlighted frustration with operator event logs as a means for diagnosis and debugging, stating they did not understand why an event may have occurred.

Lack of operator isolation. Operators are designed with the idea they are virtually isolated from other operators — unfortunately, this isn't true in practice. 48% of our survey takers reported using multiple operators in production, but only a third of those who did stated they used standard resource isolation techniques (segregating operator resources in a different cluster or in different virtual sandboxes) to prevent operators from negatively interacting with each other. We documented at least one case where absence of such safeguards would

CoNEXT'20, December 1, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-8183-3/20/12…$15.00
https://doi.org/10.1145/3426746.3434055

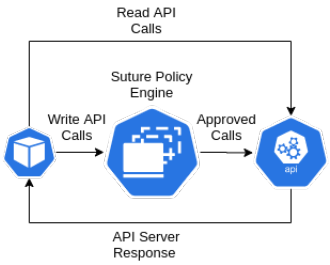

Figure 1: Proposed Architecture of Suture. Suture runs as a Kubernetes service on the Kubernetes cluster.

have led to aggresive deletion of database resources that were *not* managed by the operator.

Existing mitigations for all of these issues seem to be ineffective. Namespace isolation is used only by a minority of users, despite being the official way to enforce resource and operator isolation. 30% of operator users felt the recommended access control authorization (making use of the existing Kubernetes role-based access control[3] mechanism) suggested by their operators were overly broad or insufficient. To date, no solution for preventing incorrect behaviour from misconfiguration has been implemented for operators, and no satisfactory diagnostic tool for why operator events happen appears to be forthcoming.

We believe all of these issues can be traced back to deficiencies with the Kubernetes authorization model[2]. The Kubernetes authorization system is the primary way users can regulate the behaviour of applications in their environment. Currently, however, the system only verifies whether the action is statically whitelisted. This design decision does not work for operators, which dynamically allocated resources without a pre-existing name, and therefore may need whitelisting for their actions conditionally revoked. As a result, the Kubernetes permission model cannot constrain operators from having accidental side-effects on the cluster — its focus on pure identity and access management is insufficient for this purpose.

2 Design

We propose to build Suture, a system that models whether an operator has permission to perform an action against a resource by validating its *impact* on the cluster, rather than by what the action is or what the resources are. This is achieved by extending the Kubernetes permission model using an engine armed with rules about what properties of the system should be kept invariant under which circumstances. When an operator emits an API call, we intercept it (using a pass-through proxy, more specifically a Kubernetes mututing admission webhook) and compute whether the consequences of performing such an action would violate our invariants, optionally modifying the API call to prevent or avert the unwanted consequence if needed. Such a system directly solves the issue of configuration safety by blocking calls that would violate a desired invariant in the system.

Suture's design and architecture is still under development. In the absence of a formal semantics for Kubernetes behaviour, we do not know which design completely covers all desired interactions — it remains to be seen what works. We present below some open research questions we aim to tackle and design decisions we currently believe make sense given the landscape of operators today.

How do we model system impacts of an API call? For an initial pass, we intend to model all API calls from a specific source as undesirable modifications unless certain conditions pass. These conditions could be as simplistic as time of day, or more complex and heuristic as taking service level indicators into account. It will be the cluster administrator's responsibility to decide whether or not the conditions are valid. Once this mode of operation is established, we will then be in a position to iterate and adopt more thorough measures as a whole.

What are the ideal policy semantics that map to real-world behaviour? Based on our analysis of issues, we believe users will currently benefit if Suture at least supports allowing policies to specify being relaxed under certain cluster conditions as well as enforcing mutually exclusive access to resources between operators.

3 Future Evaluation

The OperatorHub website[1] grades operators on the basis of what sort of operational tasks an operator is capable of performing. To validate whether Suture correctly blocks unwanted side-effects as a consequence of misconfiguration, we intend to randomly select operators ranked among the most mature, and set up test conditions where an operator might perform an invalid action. Using this, we hope to be able to demonstrate correctness guarantees if Suture correctly flags and reports these events. Our hope is that this work will lay the foundation for a cleaner approach to managing operational challenges in all contexts.

4 Acknowledgments

We thank the anonymous CoNEXT reviewers for their invaluable comments. This work is supported by the National Science Foundation (through grant CNS-1816340).

References

[1] Red Hat Inc. 2020. OperatorHub. (2020). https://operatorhub.io/
[2] Kubernetes. 2020. Authorization Overview. (2020). https://kubernetes.io/docs/reference/access-authn-authz/authorization/
[3] Kubernetes. 2020. Using RBAC Authorization. (2020). https://kubernetes.io/docs/reference/access-authn-authz/rbac/
[4] Brandon Philips. 2016. Introducing Operators: Putting Operational Knowledge Into Software | Coreos. (2016). https://coreos.com/blog/introducing-operators.html