AnvilOps: Increasing Accessibility of Kubernetes with Automated Builds and Deployments

Brendan Swanson*
bdswans2@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Emma Zheng*
zheng861@purdue.edu
Purdue University
West Lafayette, IN, USA

LJ Lumas
dlumas@purdue.edu
Purdue University
West Lafayette, IN, USA

Haniye Kashgarani
hkashgar@purdue.edu
Purdue University
West Lafayette, IN, USA

Abstract

Kubernetes is a container orchestration system that offers reliable deployment of containerized applications. However, its steep learning curve and complex configuration requirements present barriers to its adoption. To address these challenges, we present AnvilOps, a platform-as-a-service designed to streamline the deployment and management of applications on a Kubernetes cluster. AnvilOps is developed at the Rosen Center for Advanced Computing (RCAC) to simplify application deployment on Anvil's Composable Kubernetes Subsystem. Through a web interface, AnvilOps enables users to deploy applications from GitHub repositories (including GitHub Enterprise) or publicly available images with minimal configuration, abstracting the details of image building and Kubernetes resource definitions. It also supports continuous deployment through GitHub webhooks, automatically redeploying apps on continuous integration events. This paper will provide an architectural overview of AnvilOps, then discuss the results and future work.

CCS Concepts

• **Software and its engineering → Software as a service orchestration system**; *Development frameworks and environments*; Software configuration management and version control systems.

Keywords

Kubernetes, CI/CD, container build automation, high-performance computing, deployment orchestration, platform-as-a-service

ACM Reference Format:
Brendan Swanson, Emma Zheng, LJ Lumas, and Haniye Kashgarani. 2025. AnvilOps: Increasing Accessibility of Kubernetes with Automated Builds and Deployments. In *HUST '25 (Not in proceedings)*. ACM, New York, NY, USA, 5 pages.

*Co-first-authors contributed equally to this research.

SC HUST '25, St. Louis, MO

1 Introduction

Modern domain science increasingly relies on composable infrastructure to support usage patterns such as interactive use, custom software stacks, web applications, and databases[17]. To meet these evolving needs within high-performance computing (HPC) environments, Kubernetes[1] provides a flexible, scalable platform for orchestrating containerized services that complement traditional batch computing and workloads.

As a standard for deploying, scaling, and managing containerized applications in both industry and research, Kubernetes allows dynamic allocation of compute and storage resources. It also provides a rich set of container management features such as autoscaling, load balancing, and self-healing, empowering teams to deploy robust and scalable applications [1]. These features provide a flexible approach to accommodating modern scientific workloads, especially those requiring elastic scaling or interactivity. See Appendix A for more information.

Despite its power and flexibility, the steep learning curve is frequently cited as a barrier to adopting Kubernetes. Many practitioners comment on the complexity of the configuration and the significant time required for training[16]. Similarly, containerization of applications is another area that often demands dedicated training; additional information is included in Appendix B. While technologies such as Kubernetes and containerization offer remarkable flexibility and scalability, their complexity can obscure their benefits, especially for researchers and domain experts whose primary focus lies outside infrastructure management [16]. An additional layer of of automation and abstraction is needed to make these technologies more accessible to non-expert users.

To bridge this gap between powerful infrastructure and practical usability, we developed AnvilOps, a platform-as-a-service (PaaS) that provides a streamlined, user-friendly interface for deploying and maintaining applications on on-premises Kubernetes clusters. AnvilOps is deployed on the Composable Subsystem of the Anvil HPC system[18] at the Rosen Center for Advanced Computing (RCAC). With built-in support for continuous deployment, the platform allows non-expert users to deploy and update applications with minimal initial configuration. By abstracting the complexities of the underlying infrastructure, AnvilOps allows users to focus on their core research and development tasks.

This paper presents the design and implementation of AnvilOps, detailing its architecture, core features, and development model.

2 Background and Motivation

2.1 Anvil

Anvil [18], Purdue University's NSF-funded Category I HPC system, exemplifies the composable approach to HPC infrastructure[17]. The Anvil Composable Subsystem is managed using Kubernetes [1], with Rancher providing centralized management and access control for the cluster [15]. AnvilOps was designed to provide end-to-end automation of the deployment process to Kubernetes clusters such as the Anvil Composable Subsystem.

2.2 Simplifying Kubernetes

AnvilOps aims to simplify three major aspects of the application deployment process. The first is interaction with Kubernetes.
To deploy applications such as databases, web servers, or other custom services, users define a number of Kubernetes resources. A typical deployment involves creating a Namespace,
Deployment or StatefulSet, Service, and Ingress rule, each with their own YAML or JSON specifications.

To reduce this complexity, AnvilOps automates the generation and deployment of required Kubernetes resources. It abstracts away the low-level details, allowing users to deploy applications without needing to understand or manage YAML files, container orchestration, or resource linkage. The fundamental Kubernetes resources that AnvilOps generates meet the needs of typical research and data science workflows, but users can deploy additional resources in the same namespace manually to work alongside those generated and managed by AnvilOps.

2.3 Simplifying Containerization

Before an application can be deployed on Kubernetes, it must be packaged as a container image. This process typically requires users to write a `Dockerfile`, a file that specifies the application's environment, dependencies, and build instructions. Docker is the most widely used tool for building and running containers, providing a standardized format and workflow that improves portability and reproducibility across systems. However, containerizing applications requires familiarity with the Dockerfile syntax and best practices.

To simplify this process, AnvilOps automatically builds a container image from a GitHub repository using one of two approaches. Users may either provide a Dockerfile directly or rely on Railpack [13] , a tool that automatically generates one based on the structure and contents of a Git repository. Railpack detects the application language and framework (e.g., Python, Node.js, Go) and infers build instructions accordingly. This automation provides a fast path to containerization for commonly used research software stacks while retaining flexibility for users who require custom images.

2.4 Providing Continuous Deployment

Traditional software development is a linear progression through stages such as design, coding, testing, and deployment, in which each stage must be completed before progressing to the next. To accelerate the development lifecycle, catch bugs earlier, and collect feedback more quickly, developers now rely on Continuous Integration and Continuous Delivery/Deployment (CI/CD) to automatically build, test, and deploy their applications, often several times a day. There are several existing frameworks for implementing these workflows, including GitHub Actions [7], CircleCI [3], and Jenkins [9].

By integrating with GitHub webhooks, AnvilOps automatically provides continuous deployment. When a commit is pushed to the specified branch of a previously deployed GitHub repository, the application is automatically rebuilt and rolled out to the cluster. AnvilOps also maintains the flexibility to integrate with user-defined GitHub Actions, as users may instead deploy after a successful workflow run rather than after each push.

2.5 Related Work

2.5.1 Repo-to-image tools. Several tools exist to automate the process of building a source repository into a container image, like `repo2docker`[12], Buildpacks[4], Nixpacks[14], and Railpack[13]. These tools scan a directory for common file names to identify the technologies used, run preset build commands, and configure environment variables, startup commands, and port numbers. `repo2docker` is designed to build reproducible Python, R, and Julia environments for interactive data science sessions, such as on JupyterHub or BinderHub, while the others are targeted at a wider range of technologies with less support for interactive scientific workflows.

AnvilOps incorporates the Railpack builder into the build and deployment process, but can also build images from Dockerfiles if specific build configuration is needed. In addition to building images, AnvilOps also automates deployment to a Kubernetes cluster, integrates with CI/CD workflows, and provides a suite of management features through its web dashboard.

2.5.2 Kubernetes-based platforms. There are many existing container-based PaaS systems that use Kubernetes for orchestration, including Epinio[19], Harpoon[8] and Devtron[5]. All three provide sleek user interfaces that make it easier for users to deploy applications without in-depth understanding of Kubernetes infrastructure and allow users to deploy from publicly available container images or linked GitHub repositories.

Devtron and Harpoon are commercial offerings which include continuous deployment capabilities. Devtron focuses on comprehensive Kubernetes management at the cost of user interface simplicity, while Harpoon offers an intuitive drag-and-drop interface but lacks Dockerfile-free image building. Epinio is an open-source option that provides a simplified web interface and command-line tool for Kubernetes interaction. It uses Buildpacks to automatically create container images; however, it lacks built-in CI/CD pipelines for user apps.

In comparison to these existing offerings, AnvilOps aims to further streamline the deployment experience for non-technical users by cutting manual configuration: build configuration, deployment configuration, and CI/CD setup. By inferring infrastructure needs, AnvilOps allows non-experts to rapidly create deployments with reasonable configurations.

3 System Overview

All components of AnvilOps are deployed on Purdue's Anvil Composable Subsystem, a Kubernetes cluster managed using Rancher.

3.1 Frontend and Backend

The backend is a stateless HTTP server implemented in Node.js, while the frontend interface was developed using the React frontend library. The request and response format is defined with an OpenAPI[20] specification containing an expected schema for each message type.

Users deploy and manage applications through the web dashboard. Then, the backend is responsible for launching build jobs for applications as well as dynamically generating and applying Kubernetes resource manifests from saved configurations. The Kubernetes Node.js client is used to manage resources in the cluster via the Kubernetes API.

Each AnvilOps application is created in a new Kubernetes namespace for isolation. In addition, AnvilOps integrates with the Kubernetes management platform Rancher to create resources in end users' Rancher projects. This allows users' workloads to run within the resource constraints of their allocation, eliminating the "noisy neighbor" problem commonly seen on shared infrastructure.

3.2 Data Model

All user data is stored in a PostgreSQL database. An application and its deployment history is represented as an App entry with many Deployments. An AnvilOps Deployment, not to be conflated with the Kubernetes Deployment object, is a snapshot of all the configuration data needed to generate Kubernetes manifests for the application. When an application's configuration changes, AnvilOps generates Kubernetes manifests on the fly using the application's name, subdomain, and a Deployment record, so the full text of manifests does not need to be stored. Because AnvilOps maintains a history of the application through these Deployment records, the user can rapidly revert to a previous build or configuration.

3.3 Image Building

AnvilOps uses the image building engine BuildKit to build container images from user-provided Git repositories. BuildKit is the container builder used by Docker and offers advanced build optimization capabilities, such as layer caching and aggressive parallelization of build steps[6], making builds fast and efficient.

BuildKit has a variety of frontends, which are responsible for converting the build context (source files) into the Low-Level Build (LLB) binary format. LLB is an intermediate representation of a "solved" build as a set of steps. AnvilOps incorporates the Dockerfile frontend, as well as the Railpack frontend [13], which analyzes source files to automatically generate a build plan for BuildKit. By incorporating Railpack, AnvilOps further streamlines the build process for applications using supported languages and frameworks (shown in Figure 1) by eliminating the need for Dockerfiles or other custom build configuration.

When a build completes, the resulting image is pushed to a container registry to make it accessible across the Kubernetes cluster.

Figure 1: The list of languages and frameworks that Railpack can build into a container image with zero configuration from the user, eliminating the need to write a Dockerfile.

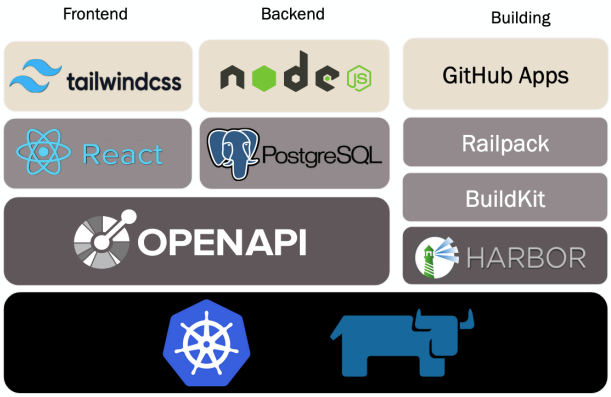

Figure 2: Graphical overview of the AnvilOps architecture. An OpenAPI specification links the React frontend with a Node.js server, with data persisted in a PostgreSQL database. A GitHub App watches connected repositories and notifies the server to start a build job with BuildKit.

4 Application Flow

4.1 Configuration

To deploy an application with AnvilOps, users complete a series of form fields through the web interface. Key fields are described below.

4.1.1 Application source. An application may be deployed from either a linked Git repository or from a publicly available container image. AnvilOps also provides a collection of template apps, each of which includes a Git repository and configuration parameters.

4.1.2 Deployment Configuration. Required deployment configuration is minimal. The user specifies the port number that the application is served on and enters a custom subdomain that the application should be made publicly available at. Environment variables and volume mounts can also be added. This information is used to deploy the application as a StatefulSet, along with a ClusterIP Service, required Secrets, and logging resources.

Git deployments require additional CI/CD configuration. The user specifies a branch of the repository and then selects either the Dockerfile or Railpack builder along with a webhook event (push or workflow run) to trigger rebuilds and redeployments.

All configuration parameters can be updated later except the subdomain name and volume mounts, which are forbidden due to Kubernetes limitations.

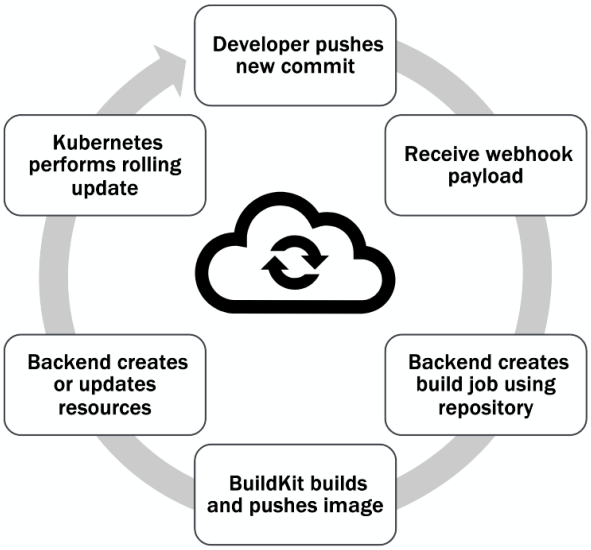

Figure 3: Each time a developer pushes a commit to the connected GitHub repository, AnvilOps can rebuild and redeploy the application.

4.2 Build and Deployment

For Git repositories, AnvilOps must complete an image building process before deployment. Using the Kubernetes API, the AnvilOps backend creates a `Job` (the "build job"). The build job clones the repository, optionally invokes Railpack to generate build instructions in place of a Dockerfile, and then uses `buildctl` to send the build context to the BuildKit daemon. When the build completes, the BuildKit daemon pushes the image to a container registry and notifies the backend that the build is complete.

AnvilOps then proceeds with deploying the container image in a new Kubernetes namespace, updating the dashboard with the application status as resources are created.

Finally, the application is running and can be accessed at the specified subdomain. From the dashboard, the user can monitor the status of pods, view real-time logs, modify configuration parameters, and scale the application as desired.

4.3 CI/CD Pipeline

An application deployed on AnvilOps can be rebuilt when a commit is pushed, or when a workflow completes successfully. When the specified event occurs on the linked GitHub repository, the deployment process begins, as shown in Figure 3:

(1) GitHub sends a webhook payload to the AnvilOps backend informing it of the new commit.
(2) The AnvilOps backend begins the build process described in Section 4.2. A status check is added to the commit, indicating that a build is in progress.
(3) The backend redeploys the application with the latest image by using the latest configuration to generate Kubernetes manifests. These are sent to the Kubernetes API in `PATCH` requests. The status check is updated to indicate success.

5 Conclusions, Limitations, and Future Work

AnvilOps lowers the barrier to entry to Kubernetes by automating build and deployment. In addition, it provides a powerful continuous delivery system tightly integrated with GitHub, a tool developers are already familiar with. By deploying applications via AnvilOps, researchers can avoid manual configuration of infrastructure, freeing up time for application development and use. After the application has undergone further improvements, we plan to open-source AnvilOps as well as release a Helm chart for rapid installation, such that the service can be used to streamline Kubernetes deployments in other environments. Ultimately, by providing a seamless interface to advanced Kubernetes infrastructure, we envision AnvilOps empowering researchers to leverage HPC resources with greater independence. In order to achieve this, we have identified six main areas of improvement.

5.0.1 Logging and observability. Users would benefit from more extensive monitoring capabilities. For instance, CPU and memory usage statistics could be provided through the dashboard, and users could receive email or Slack alerts for app crashes.

5.0.2 Access Control. AnvilOps provides access control by building on Kubernetes and Rancher's role-based access control (RBAC) features, creating a user's application only inside a Rancher Project which they were allocated. This allows resource limits to be applied to the Project as a whole. Further research is needed to understand how AnvilOps can best integrate with the access control mechanisms of other clusters.

5.0.3 Collaboration. AnvilOps currently organizes users and applications into Organizations, providing a basis for collaboratively managing deployments. Additional user roles with fine-grained permissions would allow Organization owners to manage a team more effectively.

5.0.4 Integrations. A command-line tool and/or GitHub Action for deploying applications via AnvilOps would allow users to integrate the platform into existing CI/CD workflows more effectively. In addition, supporting other Git providers such as GitLab or BitBucket would provide greater flexibility.

5.0.5 Application Support. Although applications that do not use HTTP can run on AnvilOps, they cannot have a public subdomain provisioned. Providing support for non-HTTP applications would expand the use cases for the platform.

5.0.6 Testing. We are seeking feedback from users of the Anvil Composable Subsystem to better understand how AnvilOps can effectively bridge knowledge gaps between researchers and composable infrastructure. We acknowledge that a full evaluation of user impact is still ongoing. Future work will focus on measuring usage and benefits more clearly.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2005632. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] The Kubernetes Authors. 2024. Kubernetes Documentation. https://kubernetes.io/docs/home

[2] I-Hsin Chung, Bulent Abali, and Paul Crumley. 2018. Towards a Composable Computer System. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region* (Chiyoda, Tokyo, Japan) *(HPCAsia '18)*. Association for Computing Machinery, New York, NY, USA, 137–147. doi:10.1145/3149457.3149466

[3] Circle Internet Services, Inc. 2025. *CircleCI Docs*. https://circleci.com/docs/

[4] Cloud Native Computing Foundation. 2025. *Cloud Native Buildpacks*. https://buildpacks.io/docs/

[5] Devtron. [n. d.]. *About Devtron - Building the Future of Kubernetes Delivery*. https://devtron.ai/about

[6] Docker Docs. 2025. *BuildKit*. https://docs.docker.com/build/buildkit/

[7] GitHub Docs. 2025. *GitHub Actions*. https://docs.github.com/en/actions

[8] Harpoon. [n. d.]. *About*. https://www.harpoon.io/about

[9] Jenkins Project. 2025. *Jenkins*. https://www.jenkins.io/doc/

[10] Lance Long, Timothy Bargo, Luc Renambot, Maxine Brown, and Andrew E. Johnson. 2022. Composable Infrastructures for an Academic Research Environment: Lessons Learned . In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE Computer Society, Los Alamitos, CA, USA, 1209–1214. doi:10.1109/IPDPSW55747.2022.00208

[11] Dirk Merkel. 2014. Docker: lightweight Linux containers for consistent development and deployment. *Linux J.* 2014, 239, Article 2 (March 2014).

[12] Project Jupyter Contributors. 2025. *Welcome to repo2docker's documentation*. https://repo2docker.readthedocs.io/en/latest/

[13] Railpack Docs. 2025. *Railpack*. https://railpack.com/

[14] Railway Corp. 2025. *Nixpacks*. https://nixpacks.com/docs

[15] SUSE Rancher. [n. d.]. *Projects and Kubernetes Namespaces with Rancher*. https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/manage-clusters/projects-and-namespaces

[16] Shazibul Islam Shamim, Jonathan Alexander Gibson, Patrick Morrison, and Akond Rahman. 2022. Benefits, Challenges, and Research Topics: A Multi-vocal Literature Review of Kubernetes. arXiv:2211.07032 [cs.SE] https://arxiv.org/abs/2211.07032

[17] Preston M. Smith, Erik Gough, Alexander Younts, Brian Werts, Thomas J. Hacker, Norbert Neumeister, and Jennifer Wisecaver. 2020. The "Geddes" Composable Platform - An Evolution of Community Clusters for a Composable World. In *2020 IEEE/ACM International Workshop on Interoperability of Supercomputing and Cloud Technologies (SuperCompCloud)*. 33–38. doi:10.1109/SuperCompCloud51944.2020.00011

[18] X. Carol Song, Preston Smith, Rajesh Kalyanam, Xiao Zhu, Eric Adams, Kevin Colby, Patrick Finnegan, Erik Gough, Elizabett Hillery, Rick Irvine, Amiya Maji, and Jason St. John. 2022. Anvil - System Architecture and Experiences from Deployment and Early User Operations. In *Practice and Experience in Advanced Research Computing 2022: Revolutionary: Computing, Connections, You* (Boston, MA, USA) *(PEARC '22)*. Association for Computing Machinery, New York, NY, USA, Article 23, 9 pages. doi:10.1145/3491418.3530766

[19] SUSE. 2025. *Epinio*. https://docs.epinio.io/

[20] Swagger. [n. d.]. *OpenAPI Specification - Version 3.1.1*. https://swagger.io/specification/

A Composable Infrastructure

Traditional HPC systems have historically focused on batch-style computation and workloads, such as large-scale modeling and simulations. However, modern research increasingly requires interactive workflows, on-demand analytics, custom software stacks, and web applications. These are use cases that batch HPC does not adequately support. Institutions and computing centers have demonstrated the value of composable infrastructure as a solution [10, 17]. Composable systems treat part of the shared computing resources as a flexible pool and dynamically provision them according to the workload requirements [2]. This combination of composability and container orchestration significantly reduces barriers for researchers to focus on their domain-specific goals rather than infrastructure management.

B Containers

The fundamental unit of an application deployed on Kubernetes is a container. Containers package applications and their dependencies into standardized portable units that can run consistently across various computing environments. This brings portability and reproducibility to academic and industry settings. Docker is the most widely used technology for containerization, simplifying software delivery and improving reproducibility, which is critical for scientific computing workflows [11]. Users may face friction when manually writing Dockerfiles, a process that can be error-prone and time-consuming.

Received 11 August 2025