AnvilOps: Increasing Kubernetes Accessibility via an Open-Source Platform-as-a-Service

BRENDAN SWANSON*, North Carolina State University, USA
EMMA ZHENG*, Purdue University, USA
LJ LUMAS, Purdue University, USA
HANIYE KASHGARANI, Purdue University, USA

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, an open-source platform-as-a-service that simplifies application deployment and management on Kubernetes clusters. AnvilOps, developed at the Rosen Center for Advanced Computing (RCAC), simplifies application deployment on both the Geddes and Anvil Composable Subsystems. It enables users to deploy applications from GitHub repositories or public images with minimal configuration, abstracting image building and Kubernetes resource management, and supports continuous deployment via GitHub webhooks. This paper presents the design and architecture of AnvilOps, along with 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.

Additional Key Words and Phrases: Kubernetes, CI/CD, HPC, containerization, PaaS

ACM Reference Format:
Brendan Swanson, Emma Zheng, LJ Lumas, and Haniye Kashgarani. 2026. AnvilOps: Increasing Kubernetes Accessibility via an Open-Source Platform-as-a-Service. In *Proceedings of Practice and Experience in Advanced Research Computing Conference 2026 (PEARC '26)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

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 [10]. To meet these evolving needs within high-performance computing (HPC) environments, Kubernetes provides a flexible, scalable platform for orchestrating containerized services that complement traditional batch workloads.

Kubernetes enables dynamic allocation of compute and storage resources and provides features such as autoscaling, load balancing, and self-healing. These capabilities make it well-suited for

*Both authors contributed equally to this research.

Authors' Contact Information: 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.

PEARC '26, Minneapolis, MN, USA

ACM ISBN 978-1-4503-XXXX-X/2026/02
https://doi.org/XXXXXXX.XXXXXXX

modern scientific workloads, particularly those requiring elasticity or interactivity. However, this flexibility comes with added complexity. Kubernetes has a steep learning curve that makes it difficult to get started. Its complexity can obscure its benefits, especially for researchers and domain experts who prefer to focus on their work rather than infrastructure management [9]. These challenges become even more pronounced in environments without dedicated DevOps support.

To address these barriers, we developed AnvilOps, an open-source platform-as-a-service (PaaS) that provides a streamlined, user-friendly interface for deploying and managing applications on Kubernetes clusters. With built-in support for continuous deployment, users can deploy and update applications with minimal configuration. By abstracting the underlying infrastructure, AnvilOps allows users to focus on their core research and development tasks. This paper presents the design and implementation of AnvilOps, including its architecture, features, and development model.

2 Background and Motivation

Traditional HPC systems offer access to computing resources primarily through batch jobs. A user submits a script to a batch system, which runs when the requested resources are available, then exits. This workflow is suitable for one-off jobs such as simulations and computations, but it is a poor fit for hosting the persistent services, interactive tools, and custom software stacks increasingly required in modern research. To address this gap, institutions and computing centers have adopted composable infrastructure [10], where shared resources are pooled and dynamically provisioned based on workload requirements [1]. Anvil [11], Purdue's NSF-funded Category I HPC system, exemplifies this approach with a Kubernetes-managed composable subsystem and Rancher for centralized management and access control [6, 10]. This setup allows researchers to deploy persistent and interactive applications while leveraging Kubernetes' automation, efficiency, portability, and scalability [8]. AnvilOps builds on this foundation by automating application deployment on Kubernetes clusters like the Anvil Composable Subsystem, reducing effort and letting researchers focus on their work. AnvilOps simplifies three key aspects of the application deployment process:

2.1 Kubernetes

The first challenge is interacting with Kubernetes. Deploying applications such as databases or web services requires defining multiple resources (e.g., Namespace, Deployment/StatefulSet, Service, and Ingress), typically through YAML configuration or tools like `kubectl`. While powerful, these approaches require familiarity with Kubernetes concepts and are prone to errors, making them difficult for non-expert users. In addition, standard graphical tools like Rancher simplify access but still expose underlying abstractions, requiring users to understand core concepts such as Deployments and Services. AnvilOps reduces this complexity by automating the creation and deployment of Kubernetes resources.

2.2 Containerization

Before an application can be deployed on Kubernetes, it must be packaged as a container image, typically using a Dockerfile that defines the application's environment, dependencies, and build steps. While Docker provides a standardized and portable workflow, writing Dockerfiles requires familiarity with syntax and best practices. To simplify this process, AnvilOps supports two approaches. Users can provide a Dockerfile directly or use Railpack [5], which automatically generates build configurations based on the structure of the codebase.

2.3 Continuous Deployment

To improve development efficiency and enable faster feedback, developers rely on Continuous Integration and Continuous Delivery/Deployment (CI/CD) to automate building, testing, and

rolling out applications. By integrating with GitHub webhooks, AnvilOps automatically provides continuous deployment. When a commit is pushed to a specified branch or a configured GitHub Actions workflows succeeds, the application is rebuilt and deployed to the cluster.

2.4 Related Work

2.4.1 Repo-to-image tools. Several tools automate building source repositories into container images, including `repo2docker` [4], Buildpacks [2], and Railpack [5]. These tools detect technologies from project files and generate build configurations, environment variables, and startup commands accordingly. `repo2docker` focuses on reproducible Python, R, and Julia environments for interactive data science platforms, while the others support a broader range of applications. AnvilOps uses Railpack as its primary repo-to-image tool due to its speed and convenient integration with the build engine BuildKit, which AnvilOps also uses to build from Dockerfiles.

2.4.2 Kubernetes-based platforms. A variety of Kubernetes-based platforms aim to streamline application deployment on managed or on-premises infrastructure.

Red Hat OpenShift [7] is an opinionated Kubernetes distribution that includes networking, security features, an integrated image registry, and CI/CD tooling. OpenShift Builds and Pipelines support building images and automating deployments, but these tools are limited to OpenShift clusters. In contrast, AnvilOps is designed for portability and can run on any Kubernetes cluster.

Kubero [12] is an open-source Kubernetes-based PaaS. It is more mature than AnvilOps, featuring support for multiple Git providers, Helm-based deployments, resource monitoring, and convenient templates for storage services like databases. However, while Kubero supports multi-tenancy, it lacks mechanisms for applying resource quotas across groups of namespaces at the team level, making cluster administration more complex. Rancher addresses this through its Project abstraction, which enables shared quotas and access control across namespaces, but Kubero does not integrate with Rancher.

3 System Overview

3.1 Frontend and Backend

The backend is a stateless HTTP server implemented in Node.js, and the frontend is a React application. Users deploy and manage applications through the web dashboard, while the backend spawns build jobs and dynamically generates and applies Kubernetes resource manifests.

3.2 Data Model

All user data is stored in a PostgreSQL database. When a user signs in through CILogon, a record is created in the users table along with an organization derived from the user's display name. Organizations contain Apps, each representing an application or microservice, and each App can have multiple Deployments. An AnvilOps Deployment, distinct from the Kubernetes Deployment resource, represents a snapshot of the configuration required to deploy an application by generating the necessary manifests. Only one Deployment is active for an application at a time.

3.3 Image Building

AnvilOps uses BuildKit, an image building engine, 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 [3], making builds fast and efficient. AnvilOps uses Railpack [5], a third-party BuildKit frontend which analyzes source files to automatically generate a build plan for BuildKit without relying on a Dockerfile. Users who are developing applications not supported by Railpack, or who require more customized build

configuration, can provide a Dockerfile instead. When a build completes, the resulting image is pushed to a Harbor image repository before being deployed on the cluster.

4 Components of a Deployment

AnvilOps deploys each application in a new Kubernetes `Namespace`, which can be selected by the user. When Rancher integration is enabled, users can place the namespace within a `Project` so that resource usage is tracked under an existing allocation.

Applications are deployed as Kubernetes `StatefulSets`, which support horizontal and vertical scaling. A `StatefulSet` is used instead of a `Deployment` to support stateful workloads such as databases that require ordered initialization. Environment variables are stored in a Kubernetes `Secret` within the same namespace. Each application is assigned a `ClusterIP` `Service` for internal communication. Optionally, users can expose the application via a custom subdomain, in which case AnvilOps creates an `Ingress` resource. Finally, AnvilOps creates a `NetworkPolicy` in each tenant namespace to restrict access. Only AnvilOps apps in the same app group, or other services in namespaces enumerated by the administrator, will be able to communicate with the app.

5 User Flow

5.1 Initial Deployment

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

5.1.1 Application source. An application may be deployed from either a linked Git repository or from a publicly available container image.

5.1.2 Project. AnvilOps can integrate with Rancher, allowing users to deploy applications within any Project for which they have the `manage-namespaces` permission. A Rancher Project is a Rancher feature that allows multiple Kubernetes namespaces to be managed as a unit, simplifying role-based access control and resource assignments [6].

5.1.3 Configuration. The user specifies the port number that the application is served on and (optionally) a public subdomain name they want the app to use. They can also add environment variables and volume mounts. All configuration options, except volume mounts and the namespace name, can be updated after the app is created. When deploying from a Git repository, the user specifies a branch and then selects either the Dockerfile or Railpack builder along with a webhook event (push or GitHub Actions workflow success) to trigger rebuilds and redeployments. Automatic redeployment can be disabled later.

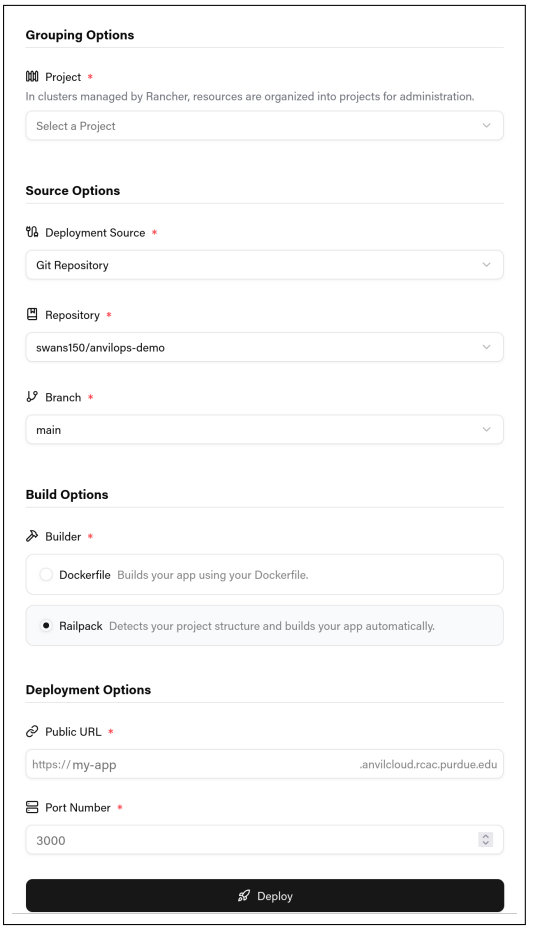

Fig. 1. A subset of the configuration options for new applications.

5.1.4 Build and Deployment. For Git repositories, AnvilOps must complete an image build process before deployment. These steps are not necessary for deployments from existing images.

(1) The AnvilOps backend uses the Kubernetes API to create a `Job` (the "build job").
(2) The build job clones the repository, optionally invokes Railpack to generate build instructions in place of a Dockerfile, and then sends the build context to the BuildKit daemon.
(3) As the application builds, AnvilOps streams logs from the build job to the dashboard.
(4) When the build completes, the BuildKit daemon requests credentials from the build job and uses them to push the image to a Harbor registry.
(5) The build job sends an HTTP request to the AnvilOps backend to confirm that the build is complete and then terminates.

AnvilOps then proceeds with deploying the container image, generating Kubernetes manifests from the configuration parameters and applying them in a new namespace. As the resources are created, AnvilOps updates the dashboard with the app's status.

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.

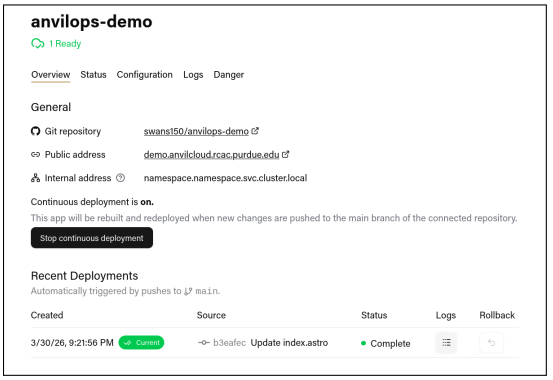

Fig. 2. The AnvilOps dashboard. Users can monitor status, deployments, logs, and configuration options.

5.2 CI/CD Pipeline

After an AnvilOps user has created an app on the platform and linked it to a GitHub repository, every time they push a commit, the deployment process runs:

(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 5.1.4.
(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.
(4) Kubernetes creates or updates resources as needed.

Applications may also be deployed directly from an existing container image. In that case, only steps 3-4 are completed.

6 Installation Steps

AnvilOps can be installed from a Helm chart. Through the `values.yaml` file and Kubernetes `Secrets`, administrators configure various features such as network policy creation, Rancher integration, and storage class. Detailed instructions are available on the AnvilOps GitHub repository.

7 Security Considerations

This section lists some of our design decisions in areas crucial to security.

The platform runs all builds on the same BuildKit instance, which could lead to cache poisoning attacks. For compatibility reasons, the default installation of AnvilOps includes a privileged daemon that runs as root in its container, but we highly recommend that system administrators run it in rootless mode and/or enable user namespaces to alleviate the risk of container escape vulnerabilities.

AnvilOps currently uses the same `imagePullSecret` in all namespaces, which means one user could run another user's image if they knew its name. We made this tradeoff to maximize compatibility between container registry implementations, but if we mandated the use of one particular registry, we could easily automate credential generation and granular access control.

When Rancher integration is enabled and AnvilOps needs to interact with the Kubernetes API on behalf of a user, the platform impersonates that user. This limits the attack surface to the set of actions that the AnvilOps user can perform in the cluster.

8 Conclusion and Future Work

AnvilOps lowers the barrier to entry to Kubernetes by automating builds and deployments. In addition, it provides a powerful continuous delivery system tightly integrated with GitHub. By deploying applications via AnvilOps, researchers can avoid manual configuration of infrastructure, freeing up time for application development and use.

AnvilOps is under active development. As it is adopted by more users at the Rosen Center, features will continue to be added and refined. The source code for the platform is publicly available at https://github.com/PurdueRCAC/AnvilOps, and we invite readers to leave feedback.

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] 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

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

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

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

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

[6] 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

[7] Red Hat. 2026. *OpenShift Container Platform 4.21 Documentation.* Red Hat, Inc. https://docs.redhat.com/en/documentation/openshift_container_platform/4.21 Accessed: 2026-02-09.

[8] Khaldoun Senjab, Sohail Abbas, Naveed Ahmed, and Atta ur Rehman Khan. 2023. A survey of Kubernetes scheduling algorithms. *Journal of Cloud Computing* 12, 1 (2023), 87. doi:10.1186/s13677-023-00471-1

[9] 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

[10] 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

[11] 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

[12] Kubero Team. 2025. *Kubero Documentation.* https://www.kubero.dev/docs Accessed: 2026-02-09.