Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

One-Click Deployment

Simplifying Open-Source Software Deployment

Bachelor's Thesis in BsC Computer Science

Author: Jan Lauber
Advisor: Erik Graf
Expert: Reto Tinkeler
Project Partner: Natron Tech AG
Submission Date: 13.06.2024

I confirm with my signature that I have conducted my present Bachelor's thesis independently. All sources of information (specialist literature, discussions with experts, etc.) and other aids that have significantly contributed to my work are fully listed in my work report in the appendix. All content that does not originate from me is marked with the exact reference to its source.

Bern, 13.06.2024 Jan Lauber

Acknowledgements

Management Summary

TODO

Contents

1 Introduction

The increasing complexity and essential nature of open-source software (OSS) in the modern technology landscape have highlighted the need for simplified deployment and management solutions. Amidst a myriad of frameworks and applications, developers and organizations face significant challenges in deploying, managing, and scaling OSS. This thesis introduces a novel One-Click Deployment system designed to address these challenges, leveraging the power of Kubernetes and cutting-edge development practices to offer a streamlined, efficient, and user-friendly deployment experience.

1.1 Background and Context

Open-source software has become a cornerstone of innovation, offering a wealth of resources for building complex and reliable systems. However, the deployment and management of OSS remain daunting tasks for many, often requiring specialized knowledge and significant effort. The One-Click Deployment project emerges in response to this, aiming to democratize access to OSS by simplifying its deployment and management, thus fostering broader adoption and innovation.

1.2 Problem Statement

The deployment and ongoing management of OSS are fraught with complexities, from setting up environments and dependencies to ensuring secure, scalable, and efficient operation. These challenges can deter the use of OSS, especially among smaller organizations and individual developers who may lack the resources for intricate deployment strategies. This thesis identifies and addresses these barriers, proposing a solution that encapsulates the complexity of deployment and management processes.

1.3 Objectives of the Study

The primary objective of this study is to design, develop, and evaluate the One-Click Deployment system, focusing on:

- Simplifying the deployment process of OSS to a single click, regardless of the underlying technologies.
- Enabling seamless management and scaling of OSS deployments within a Kubernetes ecosystem.
- Assessing the impact of the One-Click Deployment system on the adoption and utilization of OSS.
- Contributing to the body of knowledge on OSS deployment and management best practices.

1.4 Thesis Structure

This thesis is structured as follows:

- **Chapter 1: Introduction** - Sets the stage by outlining the context, challenges, and objectives.
- **Chapter 2: Literature Review** - Examines existing literature on OSS deployment challenges and solutions, highlighting gaps the One-Click Deployment system aims to fill.
- **Chapter 3: Methodology** - Describes the research and development methodologies employed to create the One-Click Deployment system.
- **Chapter 4: System Design and Implementation** - Details the architectural design, technical stack, and implementation specifics of the One-Click Deployment system.
- **Chapter 5: Evaluation** - Presents the evaluation methodology and results, assessing the system's effectiveness, usability, and impact.
- **Chapter 6: Discussion and Conclusion** - Discusses the findings, implications for practice and research, limitations, and future work directions.

By following this structure, the thesis aims to provide a comprehensive insight into the challenges of OSS deployment and how the One-Click Deployment system proposes an innovative solution to overcome these barriers.

2 Literature Review

This chapter delves into the existing body of knowledge surrounding the deployment of open-source software (OSS), highlighting the technological landscape, the prevalent challenges, and the gaps that the One-Click Deployment system seeks to address.

2.1 Overview of Open-Source Software Deployment

The deployment of open-source software involves the delivery and installation of software solutions to an operational environment where they can be executed and utilized. This process encompasses a range of activities from configuration, provisioning, and orchestration to scaling and management. Unlike proprietary software, OSS deployment benefits from a vast, collaborative community contributing to its development, testing, and optimization. However, this advantage is also accompanied by unique challenges stemming from the diverse nature of OSS projects, their dependencies, and the need for specialized knowledge to deploy them efficiently.

2.2 Current Technologies and Tools

Several technologies and tools facilitate OSS deployment, each catering to different aspects of the deployment lifecycle:

- **Containerization Platforms (e.g., Docker)**: These platforms package software, libraries, and dependencies in containers, ensuring consistent environments across development, testing, and production.
- **Orchestration Tools (e.g., Kubernetes)**: Orchestration tools manage containers' deployment, scaling, and networking, supporting complex, scalable applications.
- **Continuous Integration/Continuous Deployment (CI/CD) Tools (e.g., Jenkins, GitLab CI)**: CI/CD tools automate the testing and deployment of software, enabling rapid iteration and deployment.
- **Infrastructure as Code (IaC) Tools (e.g., Terraform, Ansible)**: IaC tools automate the provisioning and management of infrastructure through code, improving deployment speed and consistency.
- **Platform as a Service (PaaS) Providers (e.g., Heroku, OpenShift)**: PaaS providers offer cloud-based platforms that simplify the deployment, management, and scaling of applications.

2.3 Challenges in OSS Deployment

Despite the advancement in deployment technologies, several challenges persist in OSS deployment:

- **Complexity**: OSS projects often involve complex dependencies and configurations, requiring specialized knowledge and significant effort to deploy and manage effectively.
- **Scalability**: Ensuring that OSS deployments can scale in response to demand without manual intervention is a significant challenge, particularly for organizations with limited resources.
- **Security**: The open nature of OSS necessitates vigilant security practices to manage vulnerabilities and updates, protecting against breaches and compliance issues.
- **Integration**: Integrating OSS into existing systems or with other OSS projects can be complicated by compatibility issues, requiring extensive customization and testing.
- **Community Support Variability**: While OSS benefits from community support, the level and quality of support can vary greatly between projects, affecting the reliability of deployment and maintenance efforts.

2.4 Gaps in Existing Solutions

A review of the current technologies and challenges reveals several gaps in the OSS deployment ecosystem:

- **Simplification and Accessibility**: There is a need for solutions that can simplify the deployment process, making it accessible to users without deep technical expertise in containerization, orchestration, or infrastructure management.
- **Unified Deployment Solution**: Current tools often address specific aspects of the deployment lifecycle, leading to the need for a unified solution that can manage the end-to-end deployment process in a cohesive manner.
- **Customization vs. Standardization**: Striking a balance between supporting customization and maintaining standard deployment practices is a persistent gap. Solutions must be flexible enough to accommodate the unique needs of different OSS projects while providing a standardized approach to simplify deployment.
- **Security and Compliance**: Enhanced tools for automating security checks, updates, and compliance validations within the deployment process are needed to address the evolving threat landscape and regulatory requirements.

The One-Click Deployment system is proposed as a solution to these gaps, aiming to simplify the deployment process, enhance accessibility, and provide a unified, secure, and compliant deployment platform for OSS projects. By addressing these identified gaps, the system seeks to support broader adoption and more efficient utilization of OSS.

3 Methodology

This chapter outlines the methodology employed in the development of the One-Click Deployment system, focusing on the research design, development approach, and the tools and technologies utilized. This comprehensive methodology ensures a systematic and efficient approach to addressing the challenges and gaps identified in OSS deployment.

3.1 Research Design

The research design adopted for this study is a mixed-methods approach, combining qualitative and quantitative research methods to gain a comprehensive understanding of the OSS deployment landscape and to evaluate the effectiveness of the One-Click Deployment system.

- **Literature Review**: An extensive review of existing literature on OSS deployment challenges, technologies, and tools was conducted. This helped identify the gaps in the current ecosystem and establish a theoretical foundation for the project.
- **Case Studies**: Multiple OSS projects were selected as case studies to understand the real-world challenges in deployment. These case studies involved detailed analysis of deployment processes, challenges faced, and the solutions adopted.
- **Surveys and Interviews**: Surveys and interviews with OSS developers, DevOps professionals, and organizations were conducted to gather insights into their deployment experiences, preferences, and the perceived challenges.
- **Prototype Evaluation**: The One-Click Deployment system prototype was evaluated through user testing sessions. Feedback was collected from users to assess usability, effectiveness, and overall satisfaction.

3.2 Development Approach

The development of the One-Click Deployment system followed an iterative and agile methodology, emphasizing continuous integration, testing, and feedback to ensure adaptability and responsiveness to user needs.

- **Requirement Analysis**: Initial requirements were gathered based on the literature review, case studies, and survey results. These requirements guided the design and development of the system.
- **Prototype Development**: A minimum viable product (MVP) was developed to demonstrate the core functionality of the One-Click Deployment system. This MVP served as a basis for initial user testing and feedback.
- **Iterative Development and Testing**: Based on the feedback and additional research findings, the system underwent several iterations of development and testing. Each iteration focused on refining features, improving usability, and integrating additional functionalities.
- **User-Centered Design**: Throughout the development process, a user-centered design approach was adopted to ensure that the system met the needs and expectations of its intended users. This involved regular user testing sessions and feedback loops to guide design decisions.

3.3 Tools and Technologies Used

The development of the One-Click Deployment system utilized a range of tools and technologies, selected for their efficiency, robustness, and compatibility with OSS deployment requirements.

- **Kubernetes [1]**: As the backbone of the system, Kubernetes was used for orchestrating container deployment, scaling, and management.
- **Docker [2]**: Docker was employed for containerizing applications, ensuring consistency across different deployment environments.
- **Operator SDK [3]**: The Operator SDK facilitated the development of the Kubernetes operator, a key component of the system that automates the deployment and management processes.
- **Svelte [4] and Pocketbase [5]**: The frontend of the system was developed using Svelte, a modern framework for building web applications, while Pocketbase served as the backend database and API server.
- **Git [6] and GitHub [7]**: Git was used for version control, with GitHub hosting the project's code repository and facilitating collaboration among developers.
- **CI/CD Tools**: Continuous Integration and Continuous Deployment were achieved using tools like GitHub Actions, automating the testing and deployment of code changes.

By leveraging these tools and technologies, the One-Click Deployment system aims to provide a simplified, efficient, and scalable solution for OSS deployment, addressing the identified challenges and gaps in the current ecosystem.

4 System Architecture and Design

This chapter presents the architecture and design of the One-Click Deployment system, detailing the system overview, architectural components including backend and frontend, database design, and security considerations that underpin the system's effectiveness and efficiency in deploying open-source software (OSS).

4.1 System Overview

The following diagram shows how and what the One-Click operator manages.

- In red you see everything responsible for the frontend / pocketbase backend.
- In blue you see everything which handles Kubernetes natively.
- In green you see what the One-Click operator manages and controls.

Every Kubernetes resource will get created and managed within a Kubernetes namespace named with the **project ID**

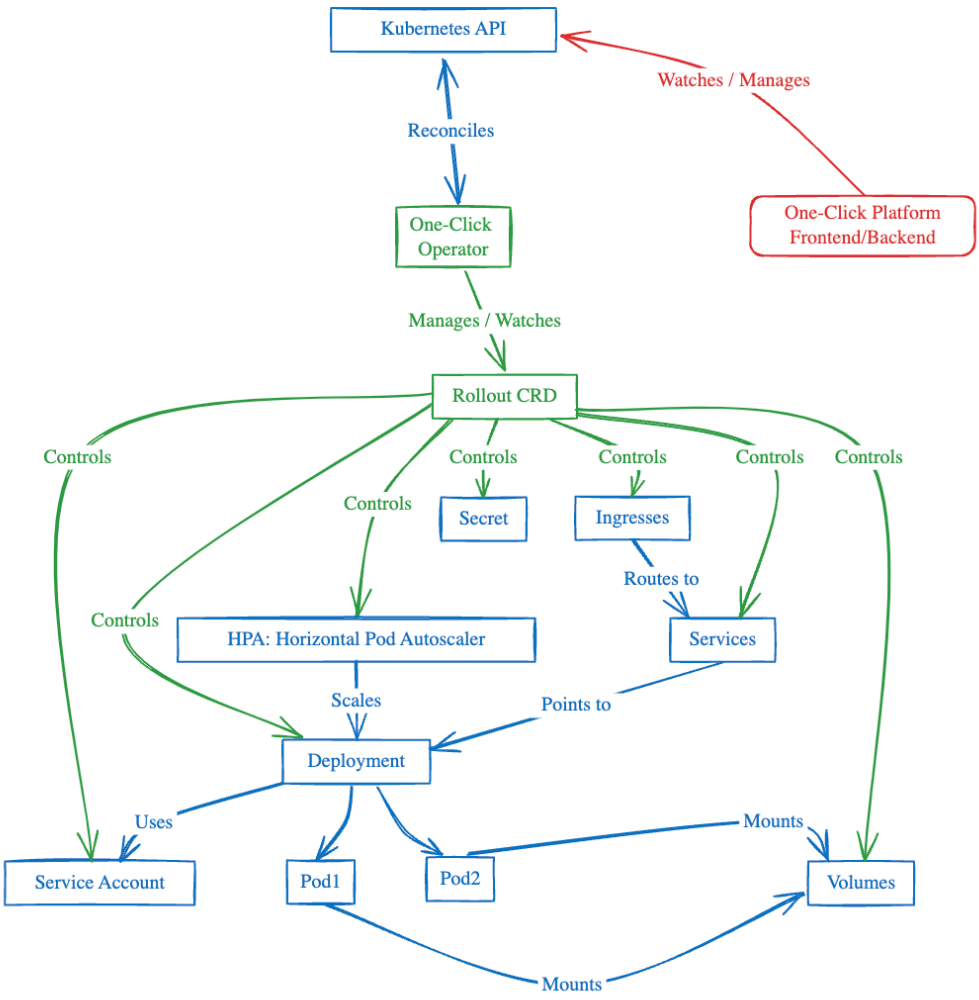

Figure 1: System Architecture and Design

The architecture of the solution is designed to operate within a Kubernetes ecosystem, focusing on simplicity and manageability for deploying containers. Here is a high-level view of its main components:

- **Frontend Component**: Developed with Svelte, the frontend provides the user interface. Its primary role is to facilitate user interaction and input, which it relays to the backend for processing.
- **Backend System**: The backend, powered by Pocketbase, acts as the central processing unit. It interprets requests from the frontend, managing the necessary API calls and interactions within the Kubernetes environment.
- **Kubernetes Cluster Interaction**: The backend is responsible for orchestrating various elements within the Kubernetes cluster. A key function includes the creation and management of namespaces, segregating projects to maintain orderly and isolated operational environments.
- **Custom Resource Management (Rollouts)**: Rollouts, defined as Custom Resource Definitions (CRDs) within Kubernetes, are managed by the backend. These are central to the deployment and operational processes, serving as bespoke objects tailored to the system's requirements.
- **One-Click Kubernetes Operator**: This component simplifies interactions with Kubernetes. It automates the handling of several Kubernetes native objects and processes, including deployments, scaling, and resource allocation. The operator is crucial for streamlining complex tasks and ensuring efficient system operations.
- **System Scalability**: The architecture is designed with scalability in mind, using Kubernetes' capabilities to handle a range of workloads and adapting as necessary for different project sizes and requirements.

This architecture aims to streamline the deployment process for OSS containers, offering an efficient and manageable system that leverages the strengths of Kubernetes, Svelte and Pocketbase.

4.2 Architecture Design

The system architecture is modular, comprising distinct backend and frontend components, and is built on a microservices architecture principle to ensure scalability, maintainability, and ease of updates.

4.2.1 Backend Architecture

The One-Click system leverages the open-source backend platform Pocketbase to manage authentication, data storage, and serving the frontend interface. In the release process, Pocketbase and frontend code are compiled together into a single container image, which is then pushed to the Github container registry, streamlining the deployment process.

Pocketbase's flexibility allows for the extension of its capabilities with custom Golang code. This feature is utilized to listen for specific events, upon which custom logic is executed, including making Kubernetes API calls and managing the Kubernetes resources initiated through the frontend interface. The project's code, demonstrating these integrations, can be accessed at https://github.com/janlauber/one-click/tree/main/pocketbase.

4.2.1.1 Authentication

Using JWT tokens, Pocketbase handles authentication processes efficiently. Upon receiving user credentials, the backend verifies them and returns a JWT token for successful authentications. This token is then stored in the local storage by the frontend and used for subsequent requests to the backend.

The system also supports authentication through various providers, including Google, GitHub, and Microsoft, with the frontend dynamically displaying login options based on the enabled providers in Pocketbase.

4.2.1.2 Database UML

The system's database structure is visualized using the PocketBaseUML tool, providing a clear representation of the data model and relationships.

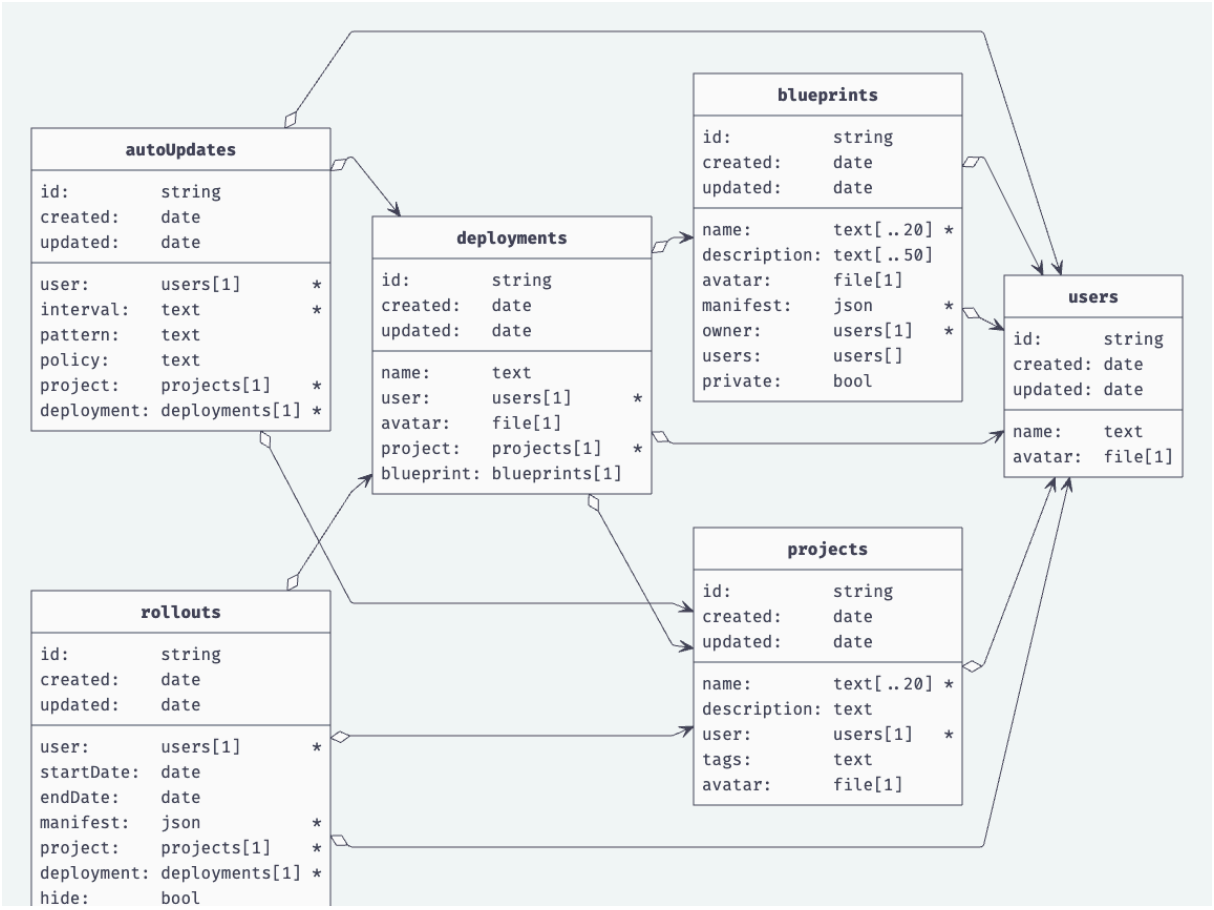

Figure 2: Database UML

4.2.1.3 Custom Endpoints

Pocketbase's capability to create custom endpoints is extensively utilized to facilitate frontend interactions. These custom endpoints, defined in the **main.go** file, support various operations, from serving the frontend and backend to fetching rollout metrics and managing Kubernetes resources. Each endpoint enforces JWT authentication to ensure secure access, except for the websocket endpoints, which are designed for real-time updates and logs.

4.2.1.4 Environment Variables

A set of environment variables is defined to configure the system's operation, such as specifying local deployment settings and configuring the auto-update feature's checking frequency.

4.2.2 Frontend Architecture

The frontend of the One-Click Deployment system is thoughtfully designed to offer a dynamic, user-friendly interface, allowing for an intuitive experience in managing open-source software deployment processes. Built on SvelteKit and augmented with TypeScript, it employs Tailwind CSS for styling and incorporates Flowbite-Svelte for UI components. The frontend leverages the Pocketbase JavaScript SDK for seamless backend interaction. Notably, the frontend's entire codebase—comprising JavaScript, HTML, and CSS—is compiled into optimized static files. These static assets are then served in conjunction with the Pocketbase backend within the same container, highlighting the system's streamlined deployment strategy.

4.2.2.1 SveteKit

Utilizing SvelteKit as the frontend framework enables the One-Click Deployment system to harness Svelte's reactivity and SvelteKit's versatility for building sophisticated web applications. SvelteKit's support for server-side rendering, static site generation, and single-page applications ensures that the frontend is fast, responsive, and accessible across all devices and network conditions.

4.2.2.2 TypeScript

TypeScript brings type safety to the development process, enhancing code reliability and maintainability. It helps in identifying and preventing potential issues early in the development cycle, facilitating easier code management and contributing to a more robust application.

4.2.2.3 Tailwind CSS and Flowbite-Svelte

The combination of Tailwind CSS's utility-first approach with Flowbite-Svelte's component library enables rapid, customizable UI development. This setup allows for the quick implementation of a visually appealing and responsive design that aligns with current web standards.

4.2.2.4 Pocketbase JavaScript SDK

The frontend extensively uses the Pocketbase JavaScript SDK to communicate with the backend efficiently. This SDK simplifies making API requests, managing authentication, and performing CRUD operations, ensuring that the frontend and backend interactions are smooth and secure.

4.2.2.5 Compiled Static Assets

A key feature of the system's frontend architecture is the compilation of its codebase into static assets. This process transforms the JavaScript, HTML, and CSS into highly optimized files that are ready for deployment. These compiled static assets are then

served alongside the Pocketbase backend within the same container. This approach ensures that the deployment of the frontend is not only efficient but also simplifies the overall system architecture by bundling both frontend and backend together.

This method of serving compiled static assets alongside the backend within the same container streamlines the deployment and hosting process, eliminating the need for separate setups for the frontend and backend. It encapsulates the essence of operational efficiency and ease of use, key principles that guide the design and implementation of the One-Click Deployment system. If needed, the frontend could also get separated from the backend and hosted independently.

4.3 Security Considerations

Security is a paramount concern in the system's design, incorporating best practices to safeguard user data and deployed applications.

- **Authentication and Authorization**: Implements secure authentication mechanisms, such as OAuth2, and role-based access control (RBAC) to ensure that users can only access and manage their projects.
- **Data Encryption**: Data at rest in the database and data in transit between the client and server are encrypted using industry-standard encryption algorithms. This is also a key feature of Pocketbase.
- **Regular Updates and Patch Management**: The system architecture is designed to facilitate easy updates and patches, ensuring that all components remain secure against known vulnerabilities.
- **Secure Development Practices**: Adheres to secure coding practices and regular security audits to preemptively address potential security issues.

5 Implementation

5.1 Development Environment Setup

The development of the One-Click Deployment system necessitates a specifically configured environment to support the technologies used. This setup includes a Kubernetes cluster, which is central to deploying and managing containerized applications. Developers need to install Docker to containerize the application, ensuring consistent operation across different environments. The backend development leverages Go, requiring a Go environment setup, while the frontend uses Node.js and SvelteKit, necessitating the installation of Node.js and the appropriate npm packages.
The development environment setup involves:

- A Kubernetes cluster either locally via Minikube or as a Managed Service at Natron Tech AG [8].
- Docker installation for building and managing containers.
- Node.js and npm to handle various frontend dependencies and build processes.
- Go environment for backend development, set to the appropriate version to ensure compatibility with all dependencies and libraries used.

These tools and setups form the backbone of the development infrastructure, providing a robust platform for building, testing, and deploying the system components efficiently.

Generally, the development environment are described in detail in the corresponding **README** files of the respective repositories.

5.2 Core Functionality Implementation

5.2.1 Deployment Module (Kubernetes Operator)

The Kubernetes Operator within the One-Click Deployment platform acts as a core component, designed to simplify the management of deployments within the Kubernetes ecosystem. It automates the process of deploying, updating, and maintaining containerized applications. Using Custom Resource Definitions (CRDs), the operator allows users to define their applications in a declarative manner.

The development of this module involved using the Operator SDK [3], which provides tools and libraries to build Kubernetes operators in Go. This SDK facilitates the monitoring of resource states within the cluster, handling events such as creation, update, and deletion of resources.

In the *controllers* directory of the *one-click-operator repository* [9] on GitHub, the core functionality of the operator is implemented. This includes the reconciliation loop, which continuously monitors the state of resources and ensures that the desired state is maintained. The operator interacts with the Kubernetes API to create and manage

resources, such as Deployments, Services, and ConfigMaps, based on the user-defined specifications.

The **rollout_controller.go** [10] is the primary controller responsible for managing Rollout resources. The following code snippets illustrate the core functionality of the deployment module:

```
// RolloutReconciler reconciles a Rollout object
func (r *RolloutReconciler) Reconcile(ctx context.Context, req ctrl.Request)
(ctrl.Result, error) {
  log := log.FromContext(ctx)

  // Fetch the Rollout instance
  var rollout oneclickiov1alpha1.Rollout
  if err := r.Get(ctx, req.NamespacedName, &rollout); err != nil {
    if errors.IsNotFound(err) {
      // Object not found
      log.Info("Rollout resource not found.")
      return ctrl.Result{}, nil
    }
    // Error reading the object - requeue the request.
    log.Error(err, "Failed to get Rollout.")
    return ctrl.Result{}, err
  }

  // Reconcile ServiceAccount
  if err := r.reconcileServiceAccount(ctx, &rollout); err != nil {
    log.Error(err, "Failed to reconcile ServiceAccount.")
    return ctrl.Result{}, err
  }

  // Reconcile PVCs, Secrets, Deployment, Service, Ingress, HPA
  [...]

  // Update status of the Rollout
  if err := r.updateStatus(ctx, &rollout); err != nil {
    if errors.IsConflict(err) {
      log.Info("Conflict while updating status. Retrying.")
      return ctrl.Result{Requeue: true}, nil
    }
    log.Error(err, "Failed to update status.")
    return ctrl.Result{}, err
  }

  return ctrl.Result{}, nil
}
```

```
// SetupWithManager sets up the controller with the Manager. This will tell the
manager to start the controller when the Manager is started.
func (r *RolloutReconciler) SetupWithManager(mgr ctrl.Manager) error {
  return ctrl.NewControllerManagedBy(mgr).
    For(&oneclickiov1alpha1.Rollout{}).
    Owns(&appsv1.Deployment{}).
    Owns(&corev1.Service{}).
    Owns(&networkingv1.Ingress{}).
    Owns(&corev1.Secret{}).
    Owns(&corev1.PersistentVolumeClaim{}).
    Owns(&autoscalingv2.HorizontalPodAutoscaler{}).
    Owns(&corev1.ServiceAccount{}).
    Complete(r)
}
```

The reconciliation loop continuously monitors the state of Rollout resources, ensuring that the desired state is maintained. The controller reconciles various resources, such as ServiceAccounts, PVCs, Secrets, Deployments, Services, Ingress, and HPAs, based on the user-defined specifications. The **SetupWithManager** function sets up the controller with the Manager, instructing the manager to start the controller when the Manager is started.

The deployment module is a critical component of the One-Click Deployment system, automating the deployment and management of containerized applications within the Kubernetes environment. The operator simplifies complex tasks, streamlining the deployment process and ensuring efficient system operations.

5.2.2 Backend Implementation

The backend of the One-Click Deployment system is built using Pocketbase, an open-source platform that simplifies backend development and deployment. The backend system handles user authentication, data storage, and interactions with the Kubernetes cluster. The backend codebase is written in Go, leveraging the flexibility and performance of the language to manage the system's core functionalities.

The backend interacts with the Kubernetes API to creates and manages the logical resources required to deploy the Rollout objects, which then trigger the Kubernetes Operator to handle the deployment process. The backend also manages user authentication, ensuring secure access to the system's functionalities.

The backend codebase is structured to handle various API endpoints, each responsible for specific operations, such as user authentication, project creation, and Rollout management. The backend interacts with the frontend through RESTful APIs, processing requests and returning appropriate responses based on the system's state and user input.

The following code snippet of the **main.go** [11] demonstrates the implementation of a backend API endpoint for creating a new project:

```
[...]
func main() {
  // Initialize the Pocketbase app
  app := pocketbase.New()

  [...]

   // Listen for incommint requests to create a new rollout and trigger the
rollout creation process in the Kubernetes cluster
  app.OnRecordBeforeCreateRequest().Add(func(e *core.RecordCreateEvent) error {
    switch e.Collection.Name {
    case "rollouts":
      return controller.HandleRolloutCreate(e, app)
    }
    return nil
  })

  [...]
}
```

The code snippet demonstrates the event handling mechanism in Pocketbase, where the backend listens for incoming requests to create a new Rollout object. Upon receiving the request, the backend triggers the Rollout creation process in the Kubernetes cluster, initiating the deployment of the specified application.

The backend implementation is designed to provide a robust and efficient foundation for the One-Click Deployment system, enabling seamless interactions between the frontend, backend, and Kubernetes environment.

5.2.3 User Interface Implementation

The user interface of the One-Click Deployment system is developed using Svelte, a modern web framework that simplifies frontend development and enhances user experience. The frontend interface serves as the primary interaction point for users, allowing them to define and manage deployment projects easily.

The frontend codebase is structured to provide a dynamic and intuitive user experience, with components designed to facilitate project creation, application deployment, and configuration. The frontend interacts with the backend through RESTful APIs with the Pocketbase Javascript SDK [12], enabling seamless communication between the user interface and the backend system. The frontend leverages Tailwind CSS [13] for styling and Flowbite-Svelte [14] for UI components, ensuring a consistent and visually appealing design.

With SvelteKit as the frontend framework, the One-Click Deployment system benefits from Svelte's reactivity and SvelteKit's versatility, enabling the development of fast, responsive, and accessible web applications. TypeScript is used to enhance code reliability and maintainability, providing type safety and early error detection during development.

5.3 Integration Strategies

Integration plays a critical role in the operational efficiency of the One-Click Deployment platform. The system integrates with existing CI/CD pipelines to automate the deployment and testing processes. This integration is facilitated through APIs and webhooks, which connect the One-Click Deployment system with tools like Jenkins, GitLab CI, and GitHub Actions.

These integration strategies enhance the platform's flexibility and adaptability, making it suitable for various operational environments and user requirements.

6 Evaluation and Testing

6.1 Evaluation Criteria

6.2 Testing Methodology

6.3 Test Results

6.4 Performance Analysis

7 Customer Use Cases and Feedback

7.1 Use Cases

7.1.1 Use Case 1: Streamlit Hosting for Data Science Projects

7.1.1.1 Description

7.1.1.2 Feedback

7.1.2 Use Case 2: Vercel Alternative for Node Projects

7.1.2.1 Description

7.1.2.2 Feedback

7.1.3 Use Case 3: Node-RED Deployment for IoT Projects

7.1.3.1 Description

7.1.3.2 Feedback

7.2 Conclusion

8 Discussion

8.1 Analysis of Findings

8.2 Comparison with Existing Solutions

8.3 Limitations and Challenges

9 Conlusion and Future Work

9.1 Summary of Contributions

9.2 Conclusions Drawn

9.3 Recommendations for Future Work

9.4 Potential Enhancements

List of Figures

Appendix A: Supplementary Material

9.5 Meetings Log

9.5.1 2024-02-14

Participants

- Jan Lauber
- Erik Graf

Topics

- Schedule and next steps
- Deliverables:
  - Task distribution
  - Solution
  - Documentation
  - Presentation
- Timeboxing for deliverables: Calculated back to 4-5 weeks
- Incorporating feedback into documentation at the beginning of the meeting
- Project 2 feedback: Positive on presentation and business perspective
- Discussion on Reflex Apps enhancing AI:
  - Autoconfiguration
  - Autocompletions
- Suggestions: Analyze latencies, requests, and configuration data for user-based recommendations
- Future meetings: Fridays in Biel, Wednesdays in Bern
- Examining use cases, Reflex deployment, chat solution integration

Action Items

- Define paper structure
- Define deliverables to be completed within 4-5 weeks
- Task planning in Github Project

9.5.2 2024-03-01

Participants

- Jan Lauber
- Erik Graf

Topics

- Expert posted on Moodle
  - Schedule meeting with expert
- Integration of business perspectives in thesis
- Discuss upcoming business trip

Action Items

- Schedule meeting with expert via email

9.5.3 2024-03-05

Participants

- Jan Lauber
- Emanuel Imhof (Founder of Unbrkn GmbH [15])

Topics

- Presentation of the project and MVP
- Emanuel Imhof's feedback on the project
  - Vercel Alternative for Node Projects
  - Remix project for Zermatt Tourism → https://ggb-map.unbrkn.dev
- Shared Kubernetes cluster hosting at Natron Tech AG [8]

Action Items

- User documentation for the project
- Setup of the project on the shared Kubernetes cluster

9.5.4 2024-04-17

Participants

- Jan Lauber
- Erik Graf

Topics

- Onboarding Coralie completed
- Creation of landing page and initial user documentation
- Discussion on business case and target audience
- Preparation for kanban setup and documentation website
- Implementation of CI/CD GitHub with testing and pre-commit hooks

Action Items

- Address inquiries regarding presentation and video requirements at defense

- Finalize features and documentation

9.5.5 2024-05-01

Participants

- Jan Lauber
- Reto Tinkler

Topics

- Review of current status
- Presenting the project to the expert
- Discussion on the presentation and defense

Action Items

- Show business case and target audience in the presentation
- Documentation should contain: Requirements, target audience, customers, project management, marketing

9.5.6 2024-05-03

Participants

- Jan Lauber
- Emanuel Imhof (Founder of Unbrkn GmbH [15])

Topics

- Onboaring Emanuel Imhof on the One-Click Deployment system
  - Teleport user
  - One Click user
  - One Click introduction
- Discussion on the project and its future
- Feedback on the project and contract for managed OneClick

Action Items

- Fix some bugs in the project
  - Typo in the delete page of the deployment
  - Auto Image Update is not working when configured for multiple deployments

9.6 Technical Documentation

9.7 Code

9.8 Additional Diagrams

Bibliography

[1] "Kubernetes." Accessed: May 04, 2024. [Online]. Available: https://kubernetes.io/

[2] "Docker." Accessed: May 04, 2024. [Online]. Available: https://www.docker.com/

[3] "Operator SDK." Accessed: May 04, 2024. [Online]. Available: https://sdk.operatorframework.io/

[4] "Svelte." Accessed: May 04, 2024. [Online]. Available: https://svelte.dev/

[5] "PocketBase." Accessed: May 04, 2024. [Online]. Available: https://pocketbase.io/

[6] "Git." Accessed: May 04, 2024. [Online]. Available: https://git-scm.com/

[7] "GitHub." Accessed: May 04, 2024. [Online]. Available: https://github.com/

[8] "Natron Tech AG." Accessed: May 04, 2024. [Online]. Available: https://natron.io/

[9] J. Lauber, "Janlauber/One-Click-Operator." Accessed: May 04, 2024. [Online]. Available: https://github.com/janlauber/one-click-operator

[10] "One-Click-Operator/Controllers/Rollout_controller.Go at Main · Janlauber/One-Click-Operator." Accessed: May 04, 2024. [Online]. Available: https://github.com/janlauber/one-click-operator/blob/main/controllers/rollout_controller.go

[11] "One-Click/Pocketbase/Main.Go at Main · Janlauber/One-Click." Accessed: May 04, 2024. [Online]. Available: https://github.com/janlauber/one-click/blob/main/pocketbase/main.go

[12] "Pocketbase/Js-Sdk." Accessed: May 04, 2024. [Online]. Available: https://github.com/pocketbase/js-sdk

[13] "Tailwind CSS - Rapidly Build Modern Websites without Ever Leaving Your HTML.." Accessed: May 04, 2024. [Online]. Available: https://tailwindcss.com/

[14] "Flowbite Svelte - UI Component Library." Accessed: May 04, 2024. [Online]. Available: https://flowbite-svelte.com/

[15] "Unbrkn GmbH." Accessed: May 04, 2024. [Online]. Available: https://www.unbrkn.ch/