# 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

**Abstract**

This thesis explores how the One-Click Deployment system enhances the deployment of open-source software by leveraging Kubernetes. By simplifying the complexities involved, the system aims to democratize access to these powerful features, making it easier for users of all technical levels to benefit from advanced container orchestration.

The deployment of open-source software and frameworks often involves intricate and challenging processes, which can limit accessibility and adoption. Many users find it difficult to manage dependencies, scale applications, and ensure consistent environments across different stages of development. The One-Click Deployment system addresses these challenges by encapsulating Kubernetes' strengths within a user-friendly interface. This system simplifies deployment, scaling, and management processes, making these advanced capabilities available to a broader audience. Through iterative development and user feedback, the system has been refined to balance ease of use with robust functionality.

The primary objective of this project is to enhance the accessibility and usability of Kubernetes, enabling users to deploy and manage open-source software more efficiently. By providing a unified deployment solution that streamlines the process, the system empowers users to focus on their applications' development and innovation, rather than the complexities of deployment.

The One-Click Deployment system has successfully made the strengths of Kubernetes more accessible, significantly reducing the need for specialized knowledge. Users reported notable improvements in deployment efficiency and ease of management, highlighting the system's effectiveness in simplifying complex processes.

Future enhancements will focus on further improving the user experience, integrating additional Kubernetes features, and ensuring robust security measures. Ongoing user feedback will drive the continuous improvement of the system, ensuring it meets the evolving needs of the open-source community.

The system is open-source and available under the Apache 2.0 license on GitHub, with repositories for both the One-Click Operator and the One-Click system. This commitment to open-source principles fosters broad collaboration and continuous enhancement, further supporting its goal of making advanced deployment capabilities accessible to all.

# Table of Contents

---

[1]Results are attached in the appendices.

# 1 Introduction

Deploying and managing open-source software (OSS) has become increasingly complex, especially with the rise of Kubernetes, a powerful but often challenging tool for container orchestration. This thesis introduces the One-Click Deployment system, designed to make Kubernetes more accessible and straightforward for users of all technical backgrounds. The system has been developed under constant iteration and is available in a first fully functional version. It has even experienced commercial production use at the time of this writing.

## 1.1 Background and Context

Open-source software is a cornerstone of modern technology, driving innovation and providing essential tools for building robust systems. However, the complexity of deploying and managing these tools can be a significant barrier, particularly with Kubernetes. While Kubernetes offers powerful features for container management, its steep learning curve can be intimidating.

### 1.1.1 Concrete Example

Imagine a developer who has built a data visualization tool using the open-source framework Streamlit or a complex automation workflow using Node-RED. Initially, the developer considers deploying the application using Docker Compose, which involves creating and managing a **docker-compose.yml** file. However, this approach requires configuring a virtual machine (VM) and deciding on a cloud host or on-premise setup. The developer must also implement an SSL reverse proxy and consider vertical scaling and automatic updates every time a new Docker image is published.

Faced with these challenges, the developer might turn to Kubernetes for its built-in solutions to these problems. Kubernetes provides tools for container orchestration, scaling, and managing configurations. However, the user soon realizes the difficulty of managing the deployment through numerous YAML files required for Kubernetes resources, such as deployments, services, ingress controllers, and more. This is where the One-Click Deployment system comes in, streamlining the deployment process and abstracting the complexity involved.

**Challenges**

- **Complexity**: Kubernetes requires a deep understanding of its concepts and resources, making it challenging for beginners.
- **Configuration**: Managing YAML files for Kubernetes resources can be error-prone and time-consuming.
- **Scalability**: Ensuring that the deployment can scale horizontally and vertically requires additional configurations.
- **Security**: Implementing secure deployments with SSL certificates can be complex.
- **Maintenance**: Keeping the deployment up-to-date with the latest versions of the software and Kubernetes resources can be a manual process.

The One-Click Deployment project aims to democratize Kubernetes by simplifying its deployment and management processes, making these advanced capabilities available to everyone, from beginners to experienced developers. The system centralizes configuration and follows the principle of **"convention over configuration,"** allowing users to deploy and manage applications with minimal effort.

## 1.2 Problem Statement

The deployment and management of OSS using Kubernetes involve numerous challenges. These include setting up environments, managing dependencies, and ensuring security and scalability. These tasks often require specialized knowledge, which can limit the use of Kubernetes to larger organizations with dedicated resources. Smaller teams and individual developers may find these complexities overwhelming, hindering their ability to leverage the full potential of Kubernetes.

**Concretely, the challenges include:**

- **Complex Deployment Process**: The manual configuration of Kubernetes resources can be complex and error-prone.
- **Limited Accessibility**: Kubernetes is often perceived as difficult to learn and use, limiting its adoption.
- **Scalability Management**: Ensuring that deployments can scale efficiently requires additional configurations.
- **Security Maintenance**: Implementing secure deployments with SSL certificates and encryption can be challenging.
- **Operational Complexity**: Keeping deployments up-to-date with the latest software versions and Kubernetes resources can be time-consuming.

**Requirement by the End-User:**

- **Simplicity**: Users need an easy-to-use interface that abstracts away the complexities of Kubernetes.
- **Efficiency**: Deployments should be quick and efficient, allowing users to focus on building applications.
- **Reliability**: Deployments should be reliable, scalable, and secure without requiring manual intervention.
- **Customization**: Users should have the flexibility to customize deployment configurations based on their requirements.
- **Documentation**: Detailed documentation and support should be available to guide users through the deployment process.

The goal of One-Click Deployment is to address these challenges by providing a solution that centralizes configuration and follows the principle of "convention over configuration." This approach reduces the need for users to understand the intricate details of Kubernetes and allows them to deploy and manage applications with minimal effort. By encapsulating Kubernetes' strengths within a user-friendly interface, the One-Click Deployment system simplifies deployment, scaling, and management processes, making these advanced capabilities accessible to a broader audience.

## 1.3 Objectives of the Study

The main objectives of this study are to design, develop, and evaluate the One-Click Deployment system, focusing on:

- Simplifying the Kubernetes deployment process to a single click, making it accessible to users regardless of their technical expertise.
- Enabling easy 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 Kubernetes.
- Collecting feedback from users to refine and enhance the system's features continuously.
- Identifying opportunities for future research and development in Kubernetes deployment and management.

## 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: Analysis of the Current Landscape and Challenges** - Provides an in-depth analysis of the current OSS deployment landscape, highlighting challenges and gaps that the One-Click Deployment system aims to address.
- **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: Implementation** - Discusses the backend, frontend, and integration strategies of the One-Click Deployment system.
- **Chapter 6: Evaluation and Testing** - Presents the evaluation criteria, testing methodologies, and results of the One-Click Deployment system.
- **Chapter 7: Customer Use Cases and Feedback** - Explores real-world use cases and feedback from customers who have adopted the One-Click Deployment system.
- **Chapter 8: Discussion** - Analyzes the findings, implications, and future directions of the One-Click Deployment system.
- **Chapter 9: Conclusion** - Summarizes the key findings, contributions, and recommendations for future work.

By following this structure, the thesis aims to provide a clear and comprehensive understanding of the challenges associated with Kubernetes deployment and how the One-Click Deployment system offers a user-friendly solution to overcome these barriers, making Kubernetes more accessible to a wider audience.

# 2 Analysis of the Current Landscape and Challenges

This chapter analyzes the current landscape of open-source software (OSS) deployment, highlighting key technologies in use, prevalent challenges, and gaps that the One-Click Deployment system seeks to address. The analysis and insights are derived from **Project 2**, also conducted at the University of Applied Sciences at Bern by Jan Lauber. [1]

## 2.1 Overview of Open-Source Software Deployment

Deploying open-source software involves delivering and installing software solutions in an operational environment where they can be executed and utilized. This process includes activities such as configuration, provisioning, orchestration, scaling, and management. Unlike proprietary software, OSS deployment benefits from a collaborative community that contributes to its development, testing, and optimization. However, this advantage also presents unique challenges due to the diverse nature of OSS projects, their dependencies, and the need for specialized knowledge to deploy them efficiently.

Open-source software has been transformative, providing an array of tools and resources for building reliable systems. OSS projects are driven by community collaboration, which accelerates development and innovation. However, the diversity and complexity of these projects can pose significant challenges for deployment, as observed in Project 2.

## 2.2 Current Technologies and Tools

Several technologies and tools facilitate OSS deployment, each addressing different aspects of the deployment lifecycle. Some of the key technologies and tools include:

- **Containerization Platforms (e.g., Docker [2]):** 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 the deployment, scaling, and networking of containers, supporting complex, scalable applications. [3]

- **Continuous Integration/Continuous Deployment (CI/CD) Tools (e.g. GitLab CI [4]):** CI/CD tools automate the testing and deployment of software, enabling rapid iteration and deployment.
- **Infrastructure as Code (IaC) Tools (e.g., Terraform [5], Ansible [6]):** 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 [7], OpenShift [8]):** PaaS providers offer cloud-based platforms that simplify the deployment, management, and scaling of applications.

These technologies collectively streamline the deployment process but also introduce their own complexities. While OSS tools offer flexibility and customization, they often require significant technical expertise to manage effectively.

## 2.3 Challenges in OSS Deployment

Despite advancements 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.

These challenges highlight the need for solutions that can simplify and streamline OSS deployment, making it accessible to a broader range of users.

## 2.4 Gaps in Existing Solutions

A review of 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. Current tools often require a steep learning curve and significant manual intervention.
  - ‣ **Example**: Tools like Docker and Kubernetes are powerful but complex. Docker Compose can simplify multi-container applications but lacks advanced orchestration capabilities. Kubernetes provides extensive features but requires managing numerous YAML configuration files, which can be overwhelming for new users.
- **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 cohesively.
  - ‣ **Example**: Each tool in the deployment pipeline (e.g., CI/CD, containerization, orchestration) requires separate configurations and management, leading to fragmentation and complexity.
- **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.
  - ‣ **Example**: Customizing deployments for specific requirements can lead to inconsistencies and maintenance challenges. Standardizing deployment practices can simplify management but may limit flexibility.
- **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.
  - ‣ **Example**: Ensuring secure deployments with SSL certificates, encryption, and compliance with data protection regulations can be complex and time-consuming, requiring specialized knowledge and manual intervention.

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 section details how the One-Click Deployment system was developed, focusing on the research design, development approach, and the tools and technologies used. By adopting a structured and comprehensive methodology, we aimed to address the complex challenges of OSS deployment effectively.

## 3.1 Research Design

To understand the OSS deployment landscape and evaluate our system's effectiveness, we used a combination of different research methods:

- **Current Landscape Analysis**: We began by analyzing the current landscape of OSS deployment. This involved looking at existing tools, technologies, and methods used in the industry to identify common challenges and gaps.
- **Case Studies**: We examined several OSS projects to understand their deployment processes. These case studies provided insights into the practical issues faced during deployment and the solutions adopted to overcome them.
- **Prototype Evaluation**: A prototype of the One-Click Deployment system was created and tested by users. Feedback from these sessions was crucial in assessing the system's usability, effectiveness, and overall user satisfaction, guiding further development.

## 3.2 Development Approach

The development of the One-Click Deployment system followed an iterative and agile methodology, which allowed us to be flexible and responsive to user feedback:

- **Requirement Analysis**: Initial requirements were gathered based on our analysis of the current landscape and feedback from the case studies. These requirements helped shape the design and development of the system.
- **Prototype Development**: We developed a minimum viable product (MVP) to showcase the core functionalities of the One-Click Deployment system. This MVP was essential for initial user testing and feedback.
- **Iterative Development and Testing**: The system underwent multiple iterations of development and testing. After each iteration, user feedback was collected and used to refine features, improve usability, and add new functionalities.
- **User-Centered Design**: Throughout the development process, a user-centered design approach was adopted. Regular user testing sessions and feedback loops ensured that the system was intuitive and met user needs.

## 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 [9]**: 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 [10]**: The Operator SDK facilitated the development of the Kubernetes operator, a key component of the system that automates the deployment and management processes.
- **Svelte [11] and Pocketbase [12]**: 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 [13] and GitHub [14]**: 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.

## 3.4 Open Source Availability

The One-Click Deployment system is completely open source and is available under the Apache 2.0 license. The source code can be accessed through the following GitHub repositories:

- **One-Click Kubernetes Operator**: https://github.com/janlauber/one-click-operator
- **One-Click Main Application**: https://github.com/janlauber/one-click
- **One-Click Documentation**: https://github.com/janlauber/one-click-docs

By leveraging these tools and technologies, and making the system open source, we aim to provide a robust, efficient, and accessible solution for OSS deployment. This system simplifies the deployment process, making it accessible to a broader audience and addressing the key challenges identified in our research.

# 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**
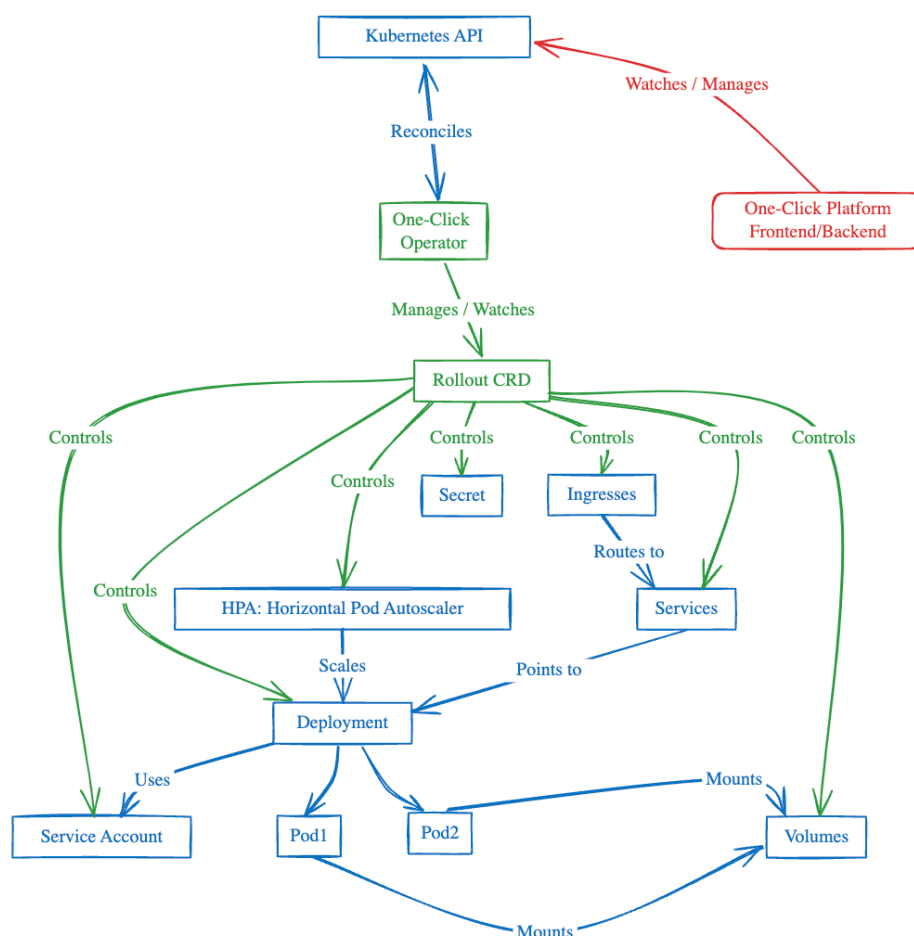


Figure 1: System Architecture and Design

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.
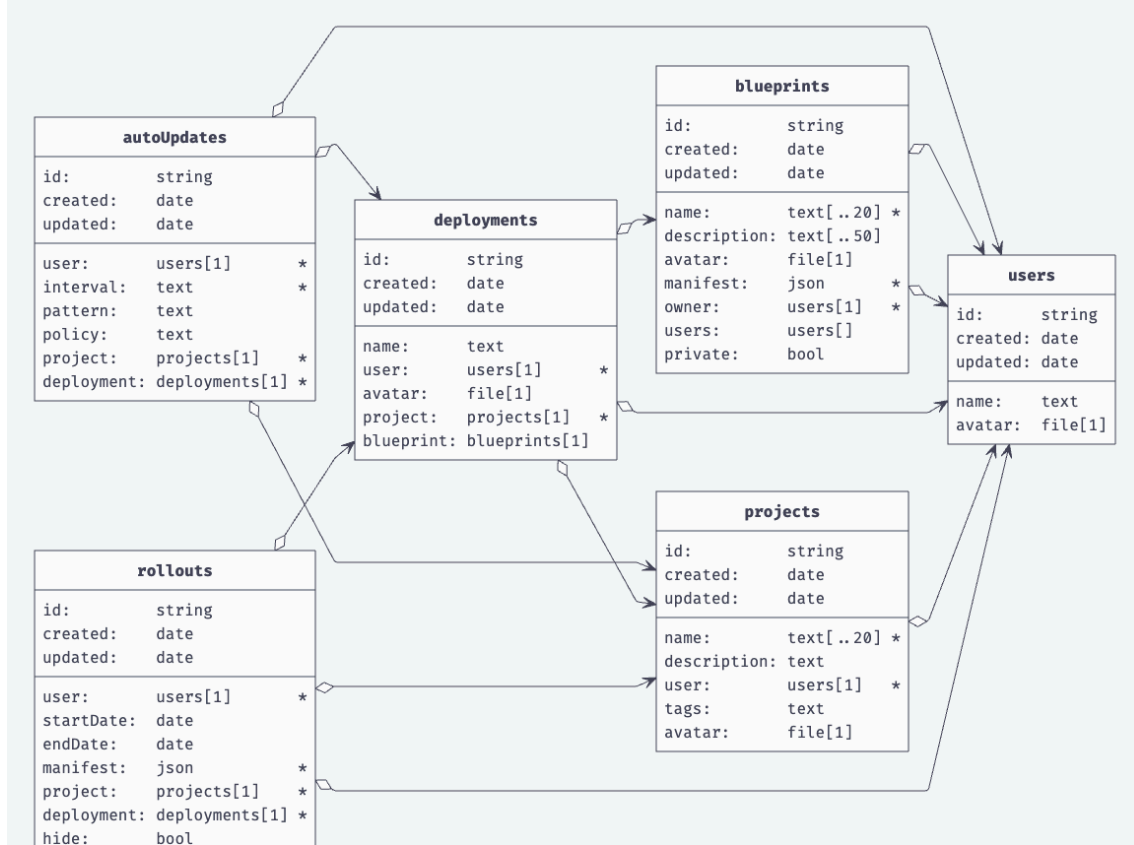


Figure 2: Database UML

### 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 [15].
- 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 Design Goals

The main goal of the One-Click Deployment system is to put **convention over configuration**. This means that the user should not have to deal with the details of Kubernetes resources like Deployments, Services, Ingresses, etc. The user should only have to define highly abstracted values like the amount of replicas, the container image, the environment variables, etc. To make it even easier for the user, the system provides a blueprint functionality of these abstracted values for certain use cases. The user can then create a new deployment based on a blueprint and only has to adjust the values which are different from the blueprint. For example, the user can create a blueprint for a Node-RED deployment and then create a

new deployment based on this blueprint and only has to adjust minor things like the URL where the Node-RED instance should be available. This way the user can deploy complex applications with only a few clicks.

The system also provides a real time monitoring of the deployed application. The user can see the current CPU and memory usage of the pods and can also see the Kubernetes resources generated by the deployment like pods, services, ingresses, etc. The user can also interact with these resources like viewing logs, events, and yaml configurations.

The system also provides a rollback functionality. Each time the user changes the deployment configuration a new rollout gets triggered. The user can then see the old rollouts and the current one. The user can also rollback to a previous rollout. This functionality is like a snapshot of the deployment configuration at a specific time.

With the implementation of the Kubernetes Operator, the system can by design be easily extended with new resources and functionalities. The system is also designed to be highly scalable and reliable. In the end the CRD (Custom Resource Definition) Rollout is the core abstraction of the system. Then the Kubernetes Operator takes care of the rest. To make this level of abstraction more accessible to the user, the system provides a web interface where the user can interact with the system and which it's core functionality is to create and manage Rollout resources.

With this design goals in mind, the core functionality of the system was implemented.

### 5.2.2 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 [10], 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* [16] on GitHub, the core functionality of the operator is implemented. This includes the

26

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.

**Kubernetes Resources managed by the operator include:**

- **ServiceAccount**[2]: A service account provides an identity for processes that run in a Pod.
- **PersistentVolumeClaim (PVC)**[3]: A PVC is a request for storage by a user.
- **Secret**[4]: A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.
- **Deployment**[5]: A Deployment provides declarative updates to Pods and ReplicaSets.
- **Service**[6]: A Service is a networking Layer 3/4 load balancer which exposes the pods within the Kubernetes cluster.
- **Ingress**[7]: An Ingress is similar to a Reverse Proxy and exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.
- **HorizontalPodAutoscaler (HPA)**[8]: An HPA automatically scales the number of pods in a deployment based on observed CPU utilization.
- **CronJob**[9]: A CronJob creates Jobs on a repeating schedule.

All these resources are managed by the operator based on the user-defined specifications in the Rollout resource explained in the Section 5.2.2.2.

### 5.2.2.1 Rollout Controller

The **rollout_controller.go** [17] is the primary controller responsible for managing Rollout resources. The following code snippets illustrate the core functionality of the deployment module:

```go
// 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
```

---

[2]https://kubernetes.io/docs/concepts/security/service-accounts/
[3]https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims
[4]https://kubernetes.io/docs/concepts/configuration/secret/
[5]https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[6]https://kubernetes.io/docs/concepts/services-networking/service/
[7]https://kubernetes.io/docs/concepts/services-networking/ingress/
[8]https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
[9]https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

```go
      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, CronJobs
  [...]

  // 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.
func (r *RolloutReconciler) SetupWithManager(mgr ctrl.Manager) error {
      if    err    :=    mgr.GetFieldIndexer().IndexField(context.TODO(),
&batchv1.CronJob{},      "metadata.ownerReferences.uid",      func(rawObj
client.Object) []string {
    cronJob := rawObj.(*batchv1.CronJob)
    ownerRefs := cronJob.GetOwnerReferences()
    ownerUIDs := make([]string, len(ownerRefs))
    for i, ownerRef := range ownerRefs {
      ownerUIDs[i] = string(ownerRef.UID)
    }
    return ownerUIDs
  }); err != nil {
    return err
```

```
    }

    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{}).
        Owns(&batchv1.CronJob{}).
        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, Cronjobs, Services, Ingress, and HPAs, based on the user-defined specifications. The **SetupWith-Manager** function sets up the controller with the Manager, instructing the manager to start the controller when the Manager is started.

### 5.2.2.2 Rollout Resource Specification

The core abstraction happens in the Rollout specification definition. This yaml structure was designed with our design goals in mind. The user or web interface then only has to interact with this structure to deploy and manage applications. The following yaml snippet illustrates the Rollout resource specification:

```yaml
apiVersion: one-click.dev/v1alpha1 # current version of the CRD
kind: Rollout # name of the CRD
metadata:
  name: nginx # name of the Rollout
  namespace: test # namespace where the Rollout should be created
spec:
  args: ["nginx", "-g", "daemon off;"] # (optional) arguments for the
container
  command: ["nginx"] # (optional) command for the container
  rolloutStrategy: rollingUpdate # or "recreate" (if not specified then
"rollingUpdate" is used)
  nodeSelector: # (optional) set specific nodes where the pods should be
scheduled
    kubernetes.io/hostname: minikube
  tolerations: # (optional) set specific tolerations for the pods
```

```yaml
      - key: "storage"
        operator: "Equal"
        value: "ssd"
        effect: "NoSchedule"
    image: # container image
      registry: "docker.io"
      repository: "nginx"
      tag: "latest"
      username: "test"
      password: "test3"
    securityContext: # (optional) security context for the container
      runAsUser: 1000
      runAsGroup: 1000
      fsGroup: 1000
      allowPrivilegeEscalation: false
      runAsNonRoot: true
      readOnlyRootFilesystem: true
      privileged: false
      capabilities:
        drop:
          - ALL
        add:
          - NET_BIND_SERVICE
    horizontalScale: # the horizontal scaling configuration
      minReplicas: 1
      maxReplicas: 3
      targetCPUUtilizationPercentage: 80
    resources: # the resource configuration like CPU and memory limits and
requests
      requests:
        cpu: "100m"
        memory: "128Mi"
      limits:
        cpu: "200m"
        memory: "256Mi"
    env: # environment variables for the container
      - name: "USERNAME"
        value: "admin"
      - name: DEBUG
        value: "true"
    secrets: # secret environment variables for the container
      - name: "PASSWORD"
        value: "admin"
      - name: "ANOTHER_SECRET"
```

```yaml
      value: "122"
  volumes: # persistent volumes for the container
    - name: "data"
      mountPath: "/data"
      size: "2Gi"
      storageClass: "standard"
  interfaces: # the network interfaces for the container
    - name: "http"
      port: 80
    - name: "https"
      port: 443
      ingress:
        ingressClass: "nginx"
        annotations:
          nginx.ingress.kubernetes.io/rewrite-target: /
          nginx.ingress.kubernetes.io/ssl-redirect: "false"
        rules:
          - host: "reflex.oneclickapps.dev"
            path: "/"
            tls: true
            tlsSecretName: "wildcard-tls-secret"
          - host: "reflex.oneclickapps.dev"
            path: "/test"
            tls: false
  cronjobs: # cronjobs for the container
    - name: some-bash-job
      suspend: false
      image:
        password: ""
        registry: docker.io
        repository: library/busybox
        tag: latest
        username: ""
      schedule: "*/1 * * * *"
      command: ["echo", "hello"]
      maxRetries: 3
      backoffLimit: 2
      env:
        - name: SOME_ENV
          value: "some-value"
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
```

```yaml
        requests:
          cpu: 100m
          memory: 256Mi
  serviceAccountName: "nginx"
```

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.

### 5.2.3 Backend Implementation

The backend of the One-Click Deployment system is built using Pocketbase[10], 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. Also go is the language of the whole Kubernetes ecosystem, so it was a natural choice to use it for the backend because all the standard libraries and tools are available in go.

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** [18] demonstrates the implementation of a backend API endpoint for creating a new project:

```go
[...]
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
  })
```

---

[10]https://pocketbase.io

```
    [...]
}
```

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.

### 5.2.4 User Interface Implementation

The user interface of the One-Click Deployment system is developed using Svelte [11], 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 [19], enabling seamless communication between the user interface and the backend system. The frontend leverages Tailwind CSS [20] for styling and Flowbite-Svelte [21] 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.

**5.2.5 User Interaction**

The user interface of the One-Click Deployment system features a clean and intuitive design, allowing users to create projects, manage deployments, and configure application settings easily. The following screenshots showcase the key components of the user interface.

**5.2.5.1 Projects Overview**

In One-Click the user can create a new project or manage existing ones. Each project will get a unique **ID** which will be used to identify the project inside the Kubernetes cluster. The project will also get a unique **namespace** in the Kubernetes cluster which has the same name as the project **ID**.
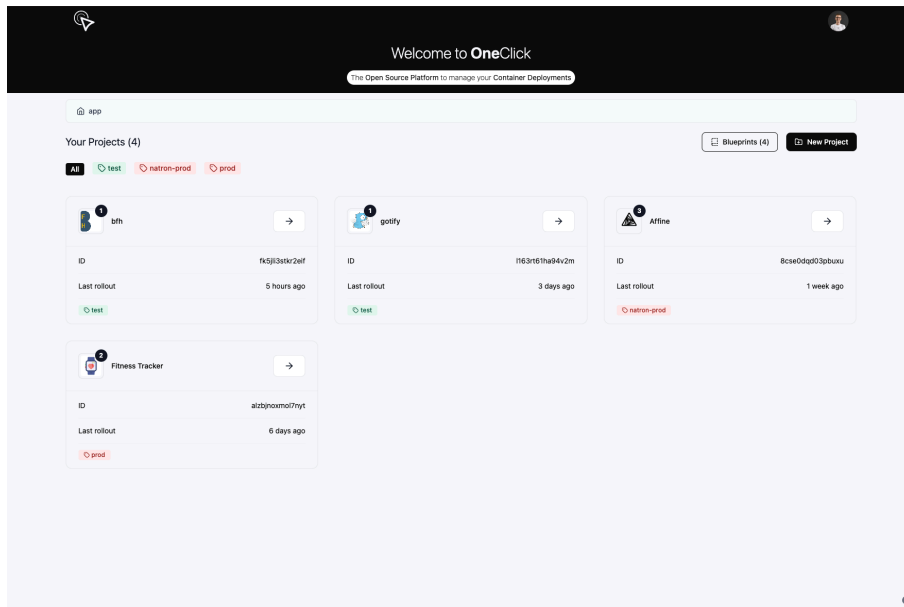


Figure 3: One-Click Deployment System Projects Overview

There will be a Kubernetes namespace each One-Click project with certain labels:

| Label | Description |
|---|---|
| `one-click.dev/displayName` | The users display name in pocketbase. |
| `one-click.dev/projectId` | The unique ID of the project. |
| `one-click.dev/userId` | The users ID of the project. |
| `one-click.dev/username` | The username of the project. |

### 5.2.5.2 Deployments Overview

In the project overview, the user can see all the deployments in this project. He can also navigate to the Blueprints, create a new Deployment or make some project settings. In the listed deployments, he can see the deployment name, status, URL (if configured), the last rollout time, number of replicas, and current deployed docker image.
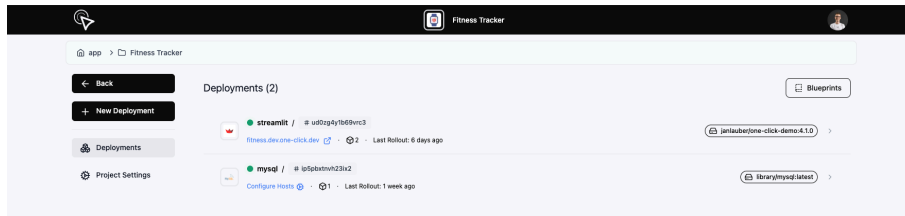


Figure 4: One-Click Deployment System Deployments Overview

### 5.2.5.3 Blueprints

Blueprints are stored configurations of a One-Click CRD (Section 5.2.2.2). With these blueprints the user can easily bootstrap new projects with no effort. The use cases are for example to predifine some enhanced configurations in his rollout crd yaml file which then get automatically applied when creating a new project out of this blueprint.
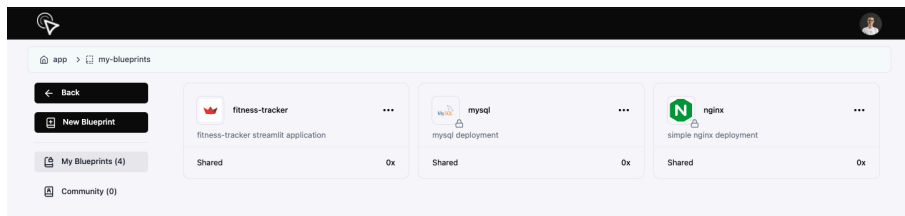


Figure 5: One-Click Deployment System Blueprints

### 5.2.5.4 Deployment Overview

When selecting or creating a new deployment within a project, users will land on the deployment overview page, as shown in the following screenshot:
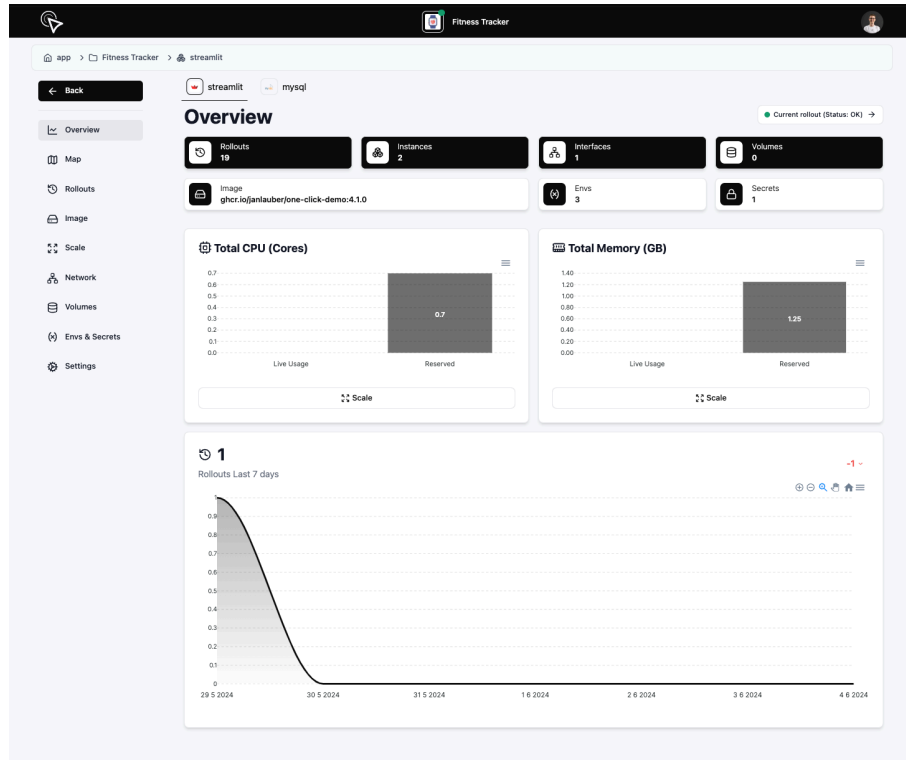


Figure 6: One-Click Deployment System Streamlit Deployment Overview

The deployment overview page displays high level information and stats about the deployment configuration, such as amount of rollouts, instances, interfaces, volumes, environment variables, secret variables and the current docker image. There is also real time CPU and memory usage monitoring.

### 5.2.5.5 Deployment Map

The map shows real time Kubernetes resources generated by the deployment, such as pods, services, ingresses, and persistent volume claims etc.
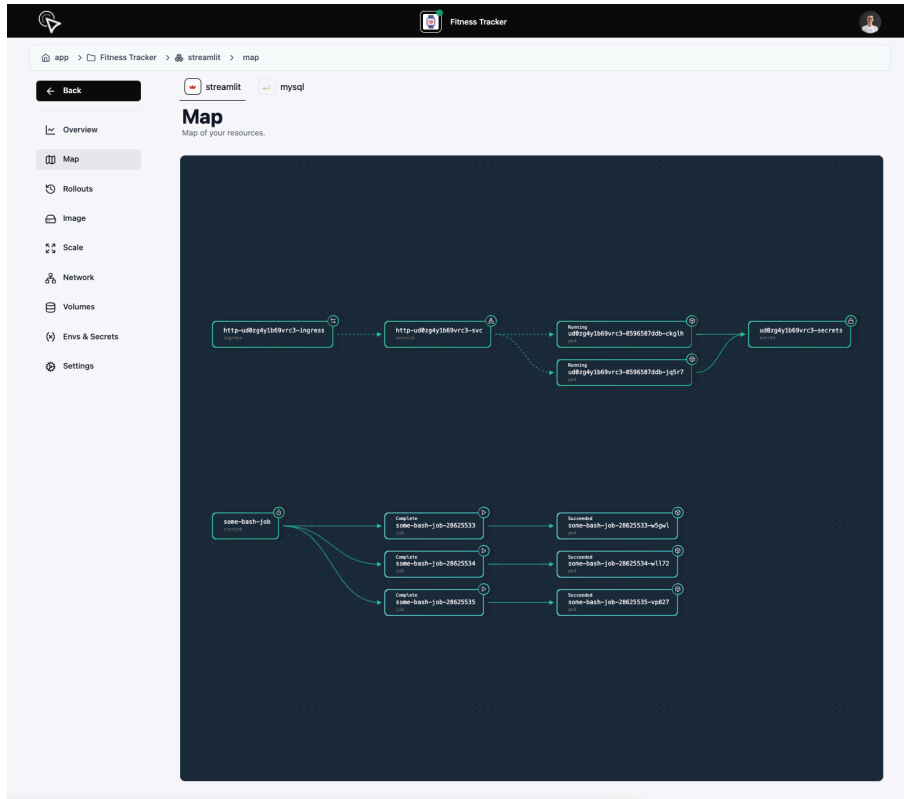


Figure 7: One-Click Deployment System Map

The map feature in a deployment uses svelte-flow[11] to graphically show the resources of the current rollout in the selected deployment. Everything gets updated in real time via a websocket endpoint to the backend.

---

[11]https://svelteflow.dev/

The user can move the components with his mouse, zoom in and out and dig into its `manifests` / `logs` / `events` when click on a component.
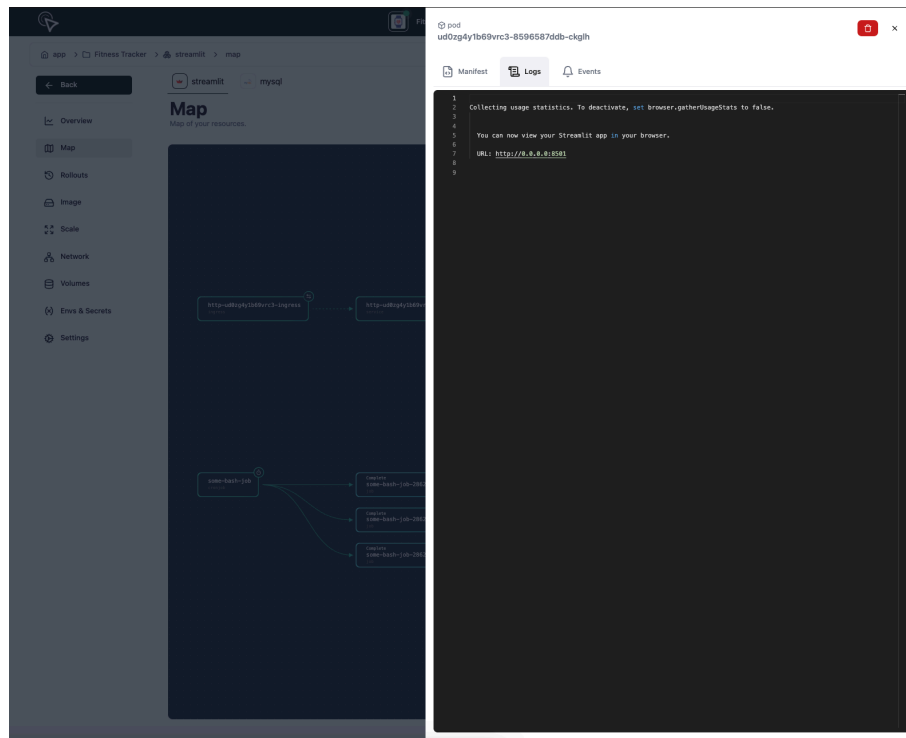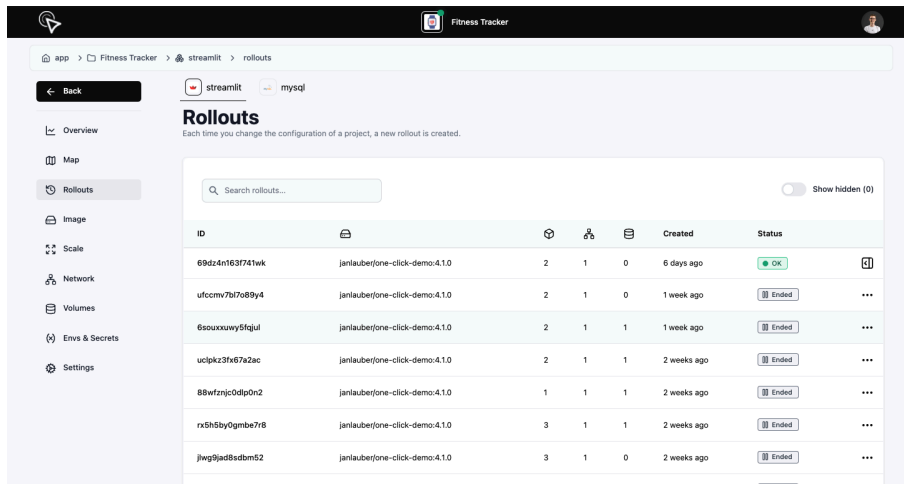


Figure 8: One-Click Deployment System Map Drawer

### 5.2.5.6 Rollouts

Each time the user edits and changes something in a deployment a new rollout will get created. This is like a **snapshot** of the CRD configuration. This gives the user the power to undo any changes he did to his deployment configuration like changing the port of an interface or updating the container image tag. He can see every rollout in the rollouts table. Through the frontend the user can either delete or hide a rollout snapshot. If he deletes a rollout then it won't pop up on the overview page anymore. If he hides a rollout then it will still be there but not visible on the rollouts table.



Figure 9: One-Click Deployment System Rollouts Table

When selecting a previous rollout the user can click on "rollback" and then a diff shows up which diffs the CRD files and show you exactly what will change:
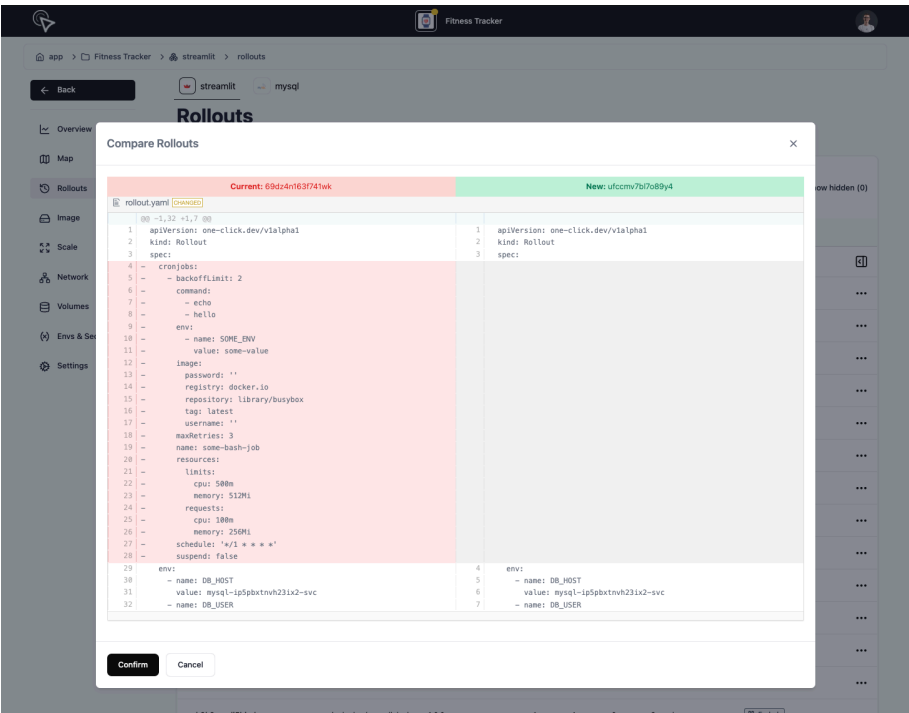


Figure 10: One-Click Deployment System Diff

### 5.2.5.7 Container Image

Under images, the management of the deployment image is possible. Configuration of the registry (e.g., ghcr.io, docker.io), along with specifying username and password for private registries, and the repository/image are supported. Additionally, defining the image tag is an option. For debugging purposes, copying the current rollout ID allows for searching components within the Kubernetes cluster.



Figure 11: One-Click Deployment System Container Image

**Auto update** To avoid manual updates of the image tag each time a new version is pushed to the registry, the Auto Update feature can be activated. This feature allows for specifying an interval (1m, 5m, 10m), a pattern, and a policy on how the image registry should be checked and updated.

- **Interval**: The cron ticker, defined as a Pocketbase environment variable, checks the registry at set intervals based on the modulo of the minutes. It is recommended to maintain the default 1-minute tick interval.
- **Pattern**: A regex pattern that parses the image tag, following the default semantic versioning format (x.x.x).
- **Policy**: This dictates the sorting method, which can be either semantic versioning (semver) or timestamp-based, with the latter requiring the image tag to be in unix timestamp format.

43

The concept and behavior mirror those found in fluxcd, as outlined in their guide to image updates: FluxCD Image Update Guide[12]

**Examples**

| Pattern | Policy | Note |
|---------|--------|------|
| `^\d+.\d+.\d+$` | semver | Default x.x.x semver pattern. e.g. 1.2.0 will get updated to 1.2.1 |
| `dev-\d+.\d+.\d+$` | semver | Custom pattern for **dev** versions. e.g. dev-1.2.0 will get updated to dev-1.2.1 |
| `.*` | timestamp | Any pattern will get updated with a **unix** timestamp. |
| `preview-*` | timestamp | A pattern with the **preview-** prefix which will get udpated with a **unix** timestamp. |

---

[12]https://fluxcd.io/flux/guides/image-update/

### 5.2.5.8 Scaling

On the scale page, configuration options are available for both horizontal and vertical scaling. Horizontal scaling adjusts the number of instances (replicas), while vertical scaling involves setting CPU and memory requests and limits.
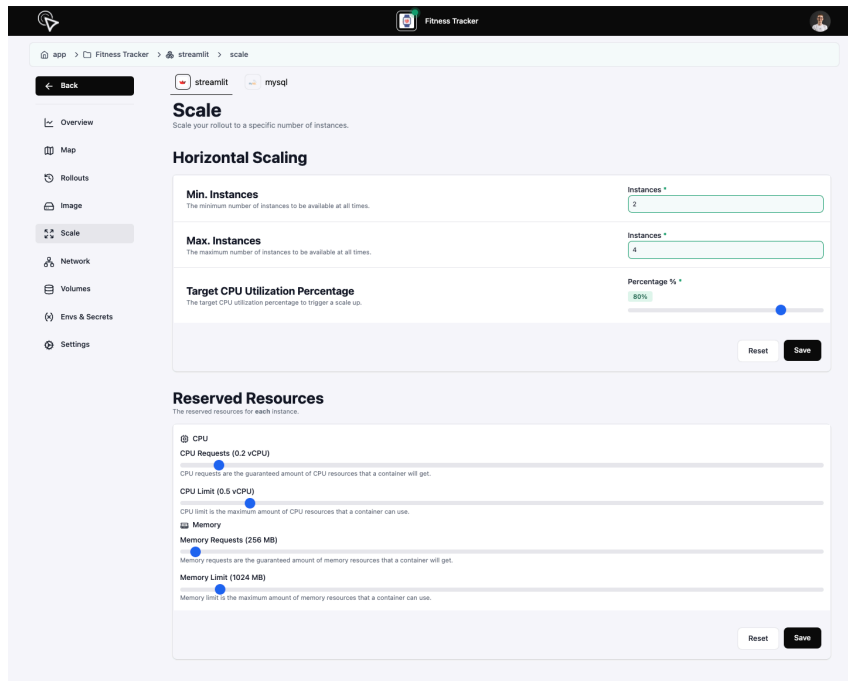


Figure 12: One-Click Deployment System Scaling

**Horizontal**

The number of minimum and maximum replicas can be defined. Autoscaling behavior is governed by the **target CPU** usage; if the CPU usage exceeds the target, the number of replicas will increase. Current CPU usage can be monitored on the deployment overview page.

**Vertical**

Vertical scaling involves setting CPU and memory **requests** and **limits**. The request specifies the minimum CPU and memory allocated to the pod, while the limit defines the maximum allowable CPU and memory. Exceeding these limits results in pod **termination** on the **memory** side and **throttling** on the **CPU** side. These measurements are specified in millicores and megabytes, with the limit set at or above the request level.

   Understanding the application's requirements is crucial for setting appropriate values. If the requirements are unknown, starting with default values and monitoring the system's behavior is advisable. Increases to the limits may be necessary if the pod is frequently terminated due to exceeding these thresholds.

### 5.2.5.9 Networking

Network configuration offers several customization options. An unlimited number of services and ingress interfaces can be configured. Services provide internal Kubernetes connectivity, while ingress interfaces handle external access. To create a new network interface, select the "New Interface" button. The configurable options include:

- **name**: Unique identifier within the deployment.
- **port**: Corresponds to the application port specified in the Dockerfile.
- **ingress** class: Select from available ingress classes within the Kubernetes cluster via a dropdown menu.
- **host**: The domain name, such as example.com.
- **path**: Access path for the interface (e.g., `/api`, `/`).
- **tls**: Toggle to enable TLS and specify the secret name for the TLS certificate.
  - ‣ **tls secret name**: If left unspecified, defaults to the host name. This setting allows for automatic TLS certificate generation using cert-manager annotations[13].

The DNS name corresponds to the Kubernetes Service name in the cluster, which other deployments can use for DNS lookups. The DNS name can be copied using the copy icon. Setting the ingress class to none, removing the host and path, and disabling TLS will result in the deletion of the ingress and creation of the service for internal network exposure only.
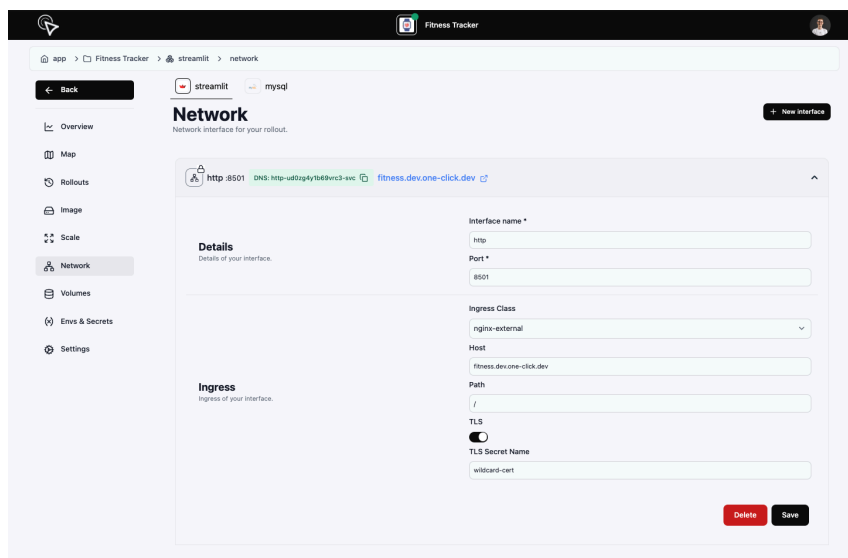


Figure 13: One-Click Deployment System Networking

---

[13]https://cert-manager.io/docs/usage/ingress/

### 5.2.5.10 Volumes

For persistent storage, volumes are available to store application data. Multiple volumes can be defined as needed. To add a new volume, click the "New Volume" button. Configurable options for volumes include:

- **name**: Unique identifier within the deployment.
- **mount path**: Location where the volume is mounted (e.g., `/data`, `/var/lib/mysql`).
- **size**: Volume size in GiB (gibibyte).
- **storage class**: Select from the available storage classes within the Kubernetes cluster using a dropdown menu.

Once created, the size and storage class of a volume cannot be changed. If a change is necessary, a new volume must be created, and the data migrated to it.



Figure 14: One-Click Deployment System Volumes

### 5.2.5.11 Environment Variables and Secrets

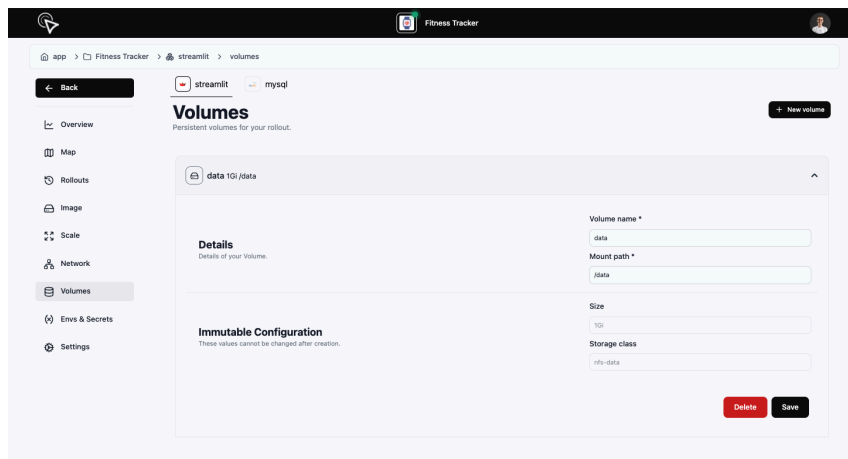On the envs & secrets page, users can configure both environment variables and secrets for their application. Environment variables are key-value pairs injected into the container, while secrets represent sensitive data stored as Kubernetes secrets and are base64 encoded. They can function as environment variables or be mounted as files within the container. Content from a `.env` file or a secret file can be directly pasted into the text area provided.

Environment variables and secrets are accessible within the container as environment going forward.
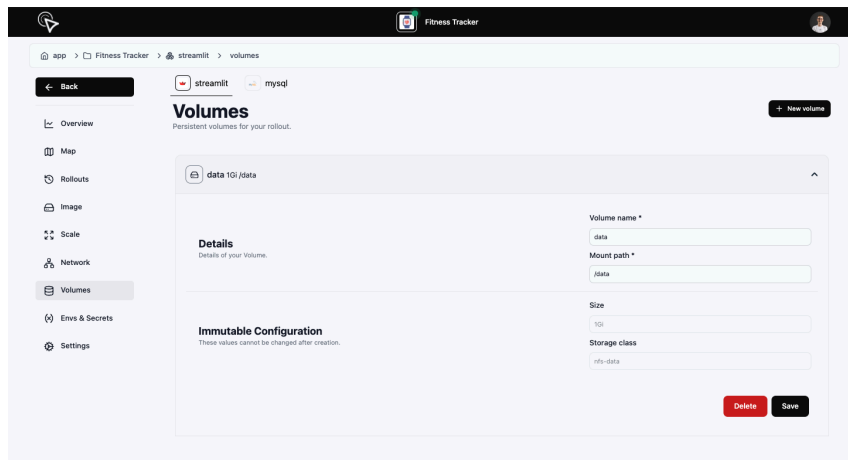


Figure 15: One-Click Deployment System Environment Variables

### 5.2.5.12 Deployment Settings

In the deployment settings the user can change it's name or avatar. He can also use the advanced editing mode to edit the CRD of the deployment directly. It's also possible to create new blueprint out of the current deployment configuration. The blueprint can then be used to create new deployments with the same configuration. The user can also delete the deployment.
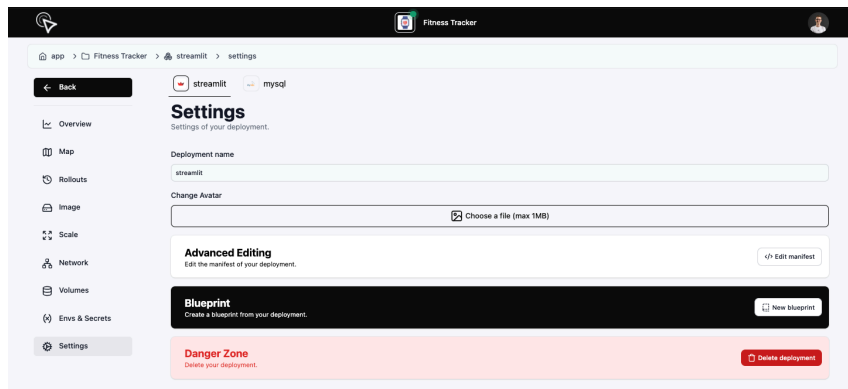


Figure 16: One-Click Deployment System Deployment Settings

## 5.3 Build and Deployment Process

Of course the One-Click System itself needs a deployment process. The core components like the Kubernetes Operator, frontend and backend needs to be accessible for the user to run it on his own. The operator is deployed in the Kubernetes cluster where the user wants to deploy his applications. It doesn't matter if it's a local Minikube cluster, a managed Kubernetes cluster at Natron Tech AG[14] or a cluster in a hyperscaler like AWS, Azure or Google Cloud. An example deployment can be found in the **config** directory of the **one-click-operator**[15] repository.

The frontend and backend are build in a single Docker container and can get deployed either in a Kubernetes cluster or on a server. It only needs a connection and kubeconfig (for authentication) to the Kubernetes cluster where the operator is running. An example to deploy it inside the Kubernetes cluster can be found in the **deployment** directory of the **one-click**[16] repository.

## 5.4 Project Management

To manage the development process efficiently, GitHub Projects was used, which provided a straightforward Kanban board for tracking progress. This tool was instrumental in organizing tasks, collecting feature requests, and prioritizing work based on user feedback and identified requirements. The use of GitHub Projects facilitated a clear and transparent workflow, enabling future contributors to collaborate effectively and stay aligned on project goals.

- **Kanban Board**: The Kanban board in GitHub Projects allowed us to visualize the development process, track the status of tasks, and manage the flow of work. Features and tasks were categorized into columns such as "To Do," "In Progress," "Review," and "Done," providing a clear overview of the project's progress.
- **Feature Collection**: Based on user feedback and requirements analysis, features were collected and added to the backlog. This ensured that user needs were continuously integrated into the development cycle, leading to a more user-centered product.
- **Prioritization**: Tasks were prioritized according to their importance and impact, ensuring that critical features and fixes were addressed promptly. This

---

[14]https://natron.ch
[15]https://github.com/janlauber/one-click-operator/tree/main/config
[16]https://github.com/janlauber/one-click/tree/main/deployment

approach helped maintain a focus on delivering high-value functionality in each iteration.

By leveraging GitHub Projects as a project management tool, it was possible to streamline the development process.

Also the use of GitHub Issues and Pull Requests was essential for tracking bugs, feature requests, and code changes, ensuring a structured and organized development process with a clear transparency for all users and contributors.

# 6 Evaluation and Testing

## 6.1 Evaluation Criteria

The evaluation of the One-Click Deployment system employs precisely defined criteria tailored to assess both system performance and user experience. These criteria ensure that results are both quantitatively measurable and qualitatively insightful, accurately reflecting the system's efficiency, usability, and scalability.

- **User Experience (UX):** The UX assessment primarily focuses on the interface usability as experienced by colleagues during a series of task-based tests. Key methodologies include:
  - ‣ **Task Completion:** Observing the ability of participants to complete predefined tasks using the system, noting any usability challenges or navigation issues.
  - ‣ **Lighthouse Testing:** Employing Google Lighthouse [22] to evaluate the frontend performance, assessing metrics such as load times, interactivity, and the overall efficiency of the web interface.

- **System Performance:**
  - ‣ **Code Quality Analysis:** Utilizing CodeFactor [23] to scrutinize the quality of the system's codebase, including the Kubernetes operator. This analysis helps identify potential issues in the code such as complexity, maintainability, and security vulnerabilities.
  - ‣ **Scalability Testing:** Conducting load tests to evaluate the system's performance under varying loads, ensuring that it can handle multiple concurrent deployments without compromising performance.

## 6.2 Testing Methodology

The methodology integrates specific software tools and manual testing to provide a comprehensive evaluation of the One-Click Deployment system:

- **User Experience Testing:** Conducted informally with colleagues, this involves direct interaction with the system's UI to complete specific deployment tasks, capturing real-time feedback and usability data.
- **Performance Testing:**
  ‣ **Frontend Performance:** Using Google Lighthouse[17] to systematically assess and document the frontend's performance under typical usage conditions.
  ‣ **Code Quality Assessment:** CodeFactor[18] is employed to analyze the entire codebase, providing a detailed report on code health, identifying areas for improvement, and ensuring adherence to best coding practices.
  ‣ **Scalability Testing:** Load tests are performed manually to simulate multiple concurrent deployments, monitoring the system's response time and resource utilization under varying loads.

## 6.3 Test Results[19]

The results from the testing phase are critical in highlighting the system's operational strengths and areas for improvement:

- **User Experience Results:**
  ‣ **Task Completion:** Most participants successfully completed the tasks with minimal guidance, indicating a generally intuitive user interface. However, some noted minor confusion with more complex functionalities.
  ‣ **Lighthouse Scores:** The frontend achieved the following Lighthouse scores:
    – Performance: 80
    – Accessibility: 92
    – Best Practices: 96
    – SEO: 91
    – First Contentful Paint: 1.7s
    – Largest Contentful Paint: 2.2s
    – Total Blocking Time: 0ms
    – Cumulative Layout Shift: 0
    – Speed Index: 1.9s

---

[17]https://developer.chrome.com/docs/lighthouse/overview
[18]https://codefactor.io/
[19]Results are attached in the appendices.

- **Performance and Reliability:**
  - ‣ **Code Quality:** CodeFactor analysis revealed an overall "A" grade for the system's codebase with specific suggestions for reducing complexity in certain modules of the Kubernetes operator. The scan will be used to refine the codebase further. The current score is always available in the repository README's.
  - ‣ **Scalability:** Load tests demonstrated that the system could handle up to 100 concurrent deployments without significant performance degradation, showcasing its scalability and reliability.

## 6.4 Recommendations

Based on the test results, several actionable recommendations are proposed:

- **UI Enhancements:** Refine the UI design to improve clarity and ease of use, especially for complex deployment configurations, based on user feedback and Lighthouse results. This includes enhancing error messages and providing more intuitive guidance for users using the advanced editing mode.
- **Optimize Frontend Resources:** Implement the changes suggested by Lighthouse to enhance load times and interactivity, such as optimizing images and reducing the impact of third-party scripts.
- **Code Refinement:** Address the issues highlighted by CodeFactor by refactoring complex code segments in the Kubernetes operator, improving maintainability and potentially enhancing performance.

These recommendations are designed to bolster the One-Click Deployment system's robustness, enhance user satisfaction, and ensure the system can efficiently meet anticipated demands. By implementing these suggestions, the system will not only perform better but also offer a more streamlined and user-friendly experience.

# 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

Coralie R. is a data scientist at the University of applied science Bern. She is working on a project in which she has developed a Streamlit application for getting data from the user by providing a user-friendly interface which uses a llm model to predict the branche of someones job. She is looking for a simple and efficient way to deploy her Streamlit application without worrying about the underlying infrastructure.

#### 7.1.1.2 Feedback

Coralie found the One-Click Deployment system to be a perfect solution for deploying her Streamlit application. She was impressed by the simplicity of the deployment process and the ability to customize the deployment configuration based on her requirements. The system's integration with Kubernetes allowed her to deploy the application seamlessly, ensuring reliable performance and scalability. Coralie appreciated the user-friendly interface and the detailed documentation provided, which guided her through the deployment process. Overall, she was satisfied with the system's performance and ease of use, making it an ideal solution for deploying her data science projects.

It is also a great way to deploy Streamlit applications rather than using the hosted solution from Streamlit. The One-Click Deployment system provides more flexibility and control over the deployment process, allowing users to customize the deployment configuration based on their requirements.

### 7.1.2 Use Case 2: Vercel Alternative for Node Projects

### 7.1.2.1 Description

Emanuel I. is a co-founder and full-stack developer for his web development agency unbrkn GmbH [24]. He is working on some Node.js projects and currently hosting them on Vercel. However, he is looking for an alternative Hosting partner in Switzerland to ensure data sovereignty and compliance with local regulations. So he is looking for a simple and efficient way to deploy his Node.js projects without worrying about the underlying infrastructure. Because at Natron Tech [15] we provide our customers managed Kubernetes clusters, he is looking for a solution that can be easily integrated with Kubernetes.

### 7.1.2.2 Feedback

Emanuel found the One-Click Deployment system to be an excellent alternative to Vercel for hosting his Node.js projects. He appreciated the system's seamless integration with Kubernetes, allowing him to deploy his applications with minimal effort. The ability to customize the deployment configuration based on his requirements was a significant advantage, enabling him to tailor the deployment process to suit his needs. Emanuel was impressed by the system's ease of use and features, ensuring that his applications ran smoothly and efficiently. He found the user interface intuitive and easy to navigate, making the deployment process straightforward and hassle-free. Overall, he was pleased with the system's capabilities and the support provided, making it an ideal solution for hosting his Node.js projects.

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

### 7.1.3.1 Description

At Natron Tech [15], we are working on simplifying the deployment of the Node-RED platform for IoT projects. The One-Click Deployment system offers a streamlined solution for deploying Node-RED instances on Kubernetes, enabling developers to create and manage IoT applications efficiently.

### 7.1.3.2 Feedback

The One-Click Deployment system has proven to be a valuable tool for simplifying the deployment of Node-RED instances for IoT projects. The system's user-friendly interface and customizable deployment configurations have made it easy for developers to deploy and manage Node-RED instances on Kubernetes. The seamless integration with Kubernetes has ensured reliable performance and scalability, allowing developers to focus on building innovative IoT applications without worrying about the underlying infrastructure. The detailed documentation and support provided have been instrumental in guiding developers through the deployment process, ensuring a smooth and hassle-free experience.

## 7.2 Conclusion

The One-Click Deployment system has received positive feedback from customers across various domains, highlighting its efficiency, reliability, and ease of use. The system's seamless integration with Kubernetes, user-friendly interface, and customizable deployment configurations have made it an ideal choice for deploying a wide range of applications, from data science projects to IoT applications. Customers have appreciated the system's performance, scalability, and detailed documentation, which have guided them through the deployment process. The feedback from customers underscores the system's value in simplifying the deployment and management of applications, ensuring a seamless and efficient experience for users. By addressing the diverse needs of customers and providing a robust deployment solution, the One-Click Deployment system has proven to be a valuable tool for developers and organizations seeking a streamlined deployment experience. The positive feedback from customers reflects the system's success in delivering on its promise of simplifying the deployment process and empowering users to focus on building innovative applications. The system's continuous improvement and customer-centric approach will further enhance its capabilities and ensure its relevance in the rapidly evolving technology landscape.

# 8 Discussion

## 8.1 Analysis of Findings

The deployment of the One-Click Deployment system across various use cases has revealed significant insights into its effectiveness, user adaptability, and operational robustness. The system's architecture, designed with a focus on simplicity and integration with Kubernetes, has proven to be highly effective in meeting diverse deployment needs. Users like Coralie R. and Emanuel I. have reported remarkable improvements in deployment efficiency, demonstrating the system's ability to cater to both data-intensive applications and complex web projects.

Key findings indicate that the system successfully reduces the technical overhead associated with deployment processes. This allows users to concentrate more on application development and less on deployment complexities. The flexibility in customization and configuration ensures that the system can adapt to specific user requirements, which is critical for applications requiring stringent compliance with data sovereignty laws, as highlighted by Emanuel's Node.js projects.

The scalability and reliability of the system, underscored by its seamless integration with Kubernetes, have been pivotal in handling fluctuating workloads, particularly in IoT applications deployed by Natron Tech AG. This scalability ensures that as user demands increase, the system can dynamically adjust to maintain performance without manual intervention. The positive feedback from users across different domains underscores the system's adaptability and robustness in diverse deployment scenarios.

## 8.2 Comparison with Existing Solutions

When compared to existing solutions like Vercel and traditional Kubernetes deployment methods, the One-Click Deployment system offers several distinct advantages:

- **Ease of Use**: Unlike traditional Kubernetes deployments, which often require detailed knowledge of container orchestration, the One-Click Deployment system provides a user-friendly interface that simplifies the entire process.
- **Customization and Flexibility**: The system allows more in-depth customization options compared to platforms like Vercel, making it suitable for a wider range of applications and compliance needs.

- **Integrated Management**: By combining application deployment with Kubernetes management, the system eliminates the need for separate tools or extensive configuration, streamlining the deployment pipeline.

These features make the One-Click Deployment system particularly appealing for developers and organizations looking for a reliable, scalable, and easy-to-use deployment solution that also adheres to stringent regulatory requirements.

## 8.3 Limitations and Challenges

Despite its strengths, the One-Click Deployment system faces several limitations and challenges:

- **Complexity in Advanced Configurations**: While the system excels in simplifying deployments, users with advanced needs may find the interface less intuitive when dealing with complex configurations. This is partly due to the system's design focusing on simplicity, which can limit detailed control in certain scenarios.
- **Dependency on Kubernetes Infrastructure**: The system's performance and scalability are heavily dependent on the underlying Kubernetes infrastructure. Any limitations or issues within the Kubernetes environment can directly impact the deployment system's effectiveness.
- **Documentation and Learning Curve**: Although the system is designed to be user-friendly, new users still face a learning curve. Comprehensive documentation[20] is available, but the initial setup and advanced feature utilization may require additional support and learning resources.
- **Security and Compliance**: While the system offers robust security features, ensuring compliance with evolving data protection regulations and security standards remains an ongoing challenge. Regular updates and enhancements are essential to address emerging threats and vulnerabilities. There is also a need for more advanced security concepts like **RBAC** and **Network Policies**.

---

[20]https://docs.one-click.dev

## 8.4 Transition to Native Kubernetes

The One-Click Deployment system simplifies Kubernetes deployments by abstracting the complexities involved. However, there are scenarios where transitioning to native Kubernetes might be necessary. These include:

- **Highly Customized Configurations**: When the users application requires highly customized configurations that go beyond the capabilities of the One-Click Deployment system. Native Kubernetes provides more granular control over configurations and resources.

- **Advanced Security Requirements**: For applications that require advanced security policies and compliance measures, native Kubernetes may offer more granular control. Features like **Network Policies** and **Pod Security Policies** can be better managed and enforced directly in Kubernetes.

- **Performance Tuning**: In cases where fine-tuned performance optimization is needed, direct access to Kubernetes configurations can be beneficial. Users can tweak resource allocations, scheduling policies, and other performance-related settings more effectively in native Kubernetes.

- **Large-Scale Deployments**: As the users application scales, you might encounter scenarios where native Kubernetes management provides better performance and resource utilization.

By recognizing these limitations and understanding when to transition to native Kubernetes, users can maximize the benefits of the One-Click Deployment system while preparing for future growth and complexity in their deployment needs.

# 9 Conlusion and Future Work

## 9.1 Summary of Contributions

The development and implementation of the One-Click Deployment system has significantly advanced the efficiency and accessibility of deploying applications, particularly within Kubernetes environments. This system stands out for its integration of a straightforward user interface with the complex mechanisms of container orchestration, making advanced deployment techniques accessible to developers regardless of their Kubernetes expertise. Its ability to seamlessly manage scalability ensures that both small-scale applications and large, dynamic workloads can operate with high reliability.

This project has contributed to the field by providing a deployment tool that not only simplifies the process but also enhances the deployment experience through customizable settings that cater to diverse operational needs. These settings enable strict compliance with data governance standards, which is critical for applications requiring stringent security measures and adherence to local data protection laws.

## 9.2 Conclusions Drawn

Evaluations across various real-world applications have demonstrated that the One-Click Deployment system effectively minimizes the time and effort required for deployments. The system's robustness in managing scalable solutions underlines its potential to support a range of applications from individual academic projects to large-scale enterprise solutions. The feedback from users like Coralie R. and Emanuel I. has been overwhelmingly positive, highlighting the system's ability to meet and exceed the operational requirements of diverse deployment scenarios.

The system's architecture is designed to evolve, anticipating future needs and technologies. Its current integration with Kubernetes provides a strong foundation for supporting emerging technologies and deployment strategies.

## 9.3 Recommendations for Future Work

To ensure the One-Click Deployment system continues to lead and innovate in deployment solutions, future work should focus on several strategic areas:

- **User Education and Engagement**: Developing comprehensive training programs and interactive tutorials that can help new users quickly understand and utilize the system effectively.
- **Community Development**: Establishing a robust community around the system can facilitate the sharing of best practices, custom configurations, and support, which are invaluable for the evolution of the platform.
- **Security Enhancements**: As cybersecurity threats evolve, so too should the security features of the deployment system. Investing in advanced security protocols and regular updates will be crucial to maintaining the integrity and trustworthiness of the system.

## 9.4 Potential Enhancements

To maintain its competitive edge and adaptability, the One-Click Deployment system can explore several enhancements:

- **Integration of AI and Machine Learning**: Implementing AI to analyze deployment patterns and predict potential issues before they affect the deployment could dramatically improve efficiency and uptime.
- **Support for Hybrid Cloud Environments**: As businesses increasingly adopt hybrid cloud strategies, the system could expand its capabilities to manage deployments across multiple cloud environments and on-premise infrastructure seamlessly.
- **Enhanced Customization for Enterprise Applications**: Developing more granular control features would allow larger organizations to fine-tune the system to their complex environments and workflows, thus broadening the system's applicability.

By focusing on these recommendations and potential enhancements, the One-Click Deployment system can continue to evolve and meet the changing needs of developers and organizations in the dynamic landscape of application deployment. The system's commitment to simplicity, scalability, and reliability positions it as a key player in the future of deployment solutions.

# 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 [24])

**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 [15]

**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 [24])

**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

### 9.5.7 2024-05-22

**Participants**

- Jan Lauber
- Erik Graf

**Topics**

- Improvements to One-Click.dev:
  - ‣ Highlighting its power: all the benefits of Kubernetes without the hassle, cloud-agnostic, non-proprietary
  - ‣ University's interest in a Kubernetes cluster that is easy to configure
    - – Automatic lifetime management of 12 months for VMs
  - ‣ One-Click as an enterprise edition:
    - – Automatic cleanup and policy management
    - – Pricing between 5-10k CHF/year
  - ‣ Identifying champions for the platform
- Preparation and practice for the presentation:
  - ‣ Emphasizing the business case
  - ‣ Preparing for a deeper business-technical defense presentation
- Administrative tasks:
  - ‣ Sending the PDF document
  - ‣ Book entry
  - ‣ Creating a marketing video featuring the BFH logo

**Action Items**

- Highlight improvements to One-Click.dev in the presentation
- Emphasize ease of use and enterprise features in marketing materials
- Prepare and practice the presentation with a focus on the business case and technical depth
- Send the required PDF document
- Make a book entry
- Create and distribute a marketing video with the BFH logo

**9.5.8 9.5.8 2024-06-05**

**Participants**

- Jan Lauber
- Erik Graf

**Story of the presentation**

- Begin the presentation from a developer's perspective.
  - ‣ The developer discovers Node-RED and wants to deploy it.
  - ‣ Show screenshots of overwhelming deployment setups (e.g., Azure Kubernetes Service).
    - – Highlight the complexity of Kubernetes.

– Transition to where people host their Kubernetes clusters.

‣ Start the second case with a custom development like Streamlit.

- **Proposal Feedback:** Erik finds the proposal excellent.
- **Iteration Process:** Discuss the numerous iterations the system has undergone.
- **Power User Representation:** How to depict power users.
- **Conventions:** What conventions the system adheres to.
- **Limitations and Transition:**

‣ When to switch to native Kubernetes.

- **Commercial Aspect:** Incorporating the commercial aspect into the story.

### Documentation

- **GitHub Repository:** [Bachelor Thesis Repository](https://github.com/janlauber/bachelor-thesis)

‣ Should a lot of code be included? Erik suggests linking it.

### Feedback

- **Commercial Deployment:** The system has been successfully deployed in a commercial setup.
- **Knowledge for Kubernetes Deployment:**

‣ What does one need to know to deploy in Kubernetes?

‣ Main steps in bullet points.

- **Gaps in Existing Solutions:**

‣ Provide concrete examples; it's quite complex.

- **UX Design and Application Logic Description:**

‣ Move "4. Design goal" to "1. User Experience."

‣ Obscure other background processes.

‣ Focus on why and how we are doing this; emphasize "convention over configuration" (centralizing configuration).

‣ Project creation and monitoring.

‣ Main value for the user.

‣ Opinionated use cases.

- **Screenshots:**

‣ Increase the prominence of user interaction.

‣ Better structure the presentation.

**Page Count:** Approximately 60 pages.

# Bibliography

[1] J. Lauber, "Project 2: One-Click Deployment." Jan. 19, 2024.

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

[3] "Kubernetes Documentation." Accessed: Jun. 02, 2024. [Online]. Available: https://kubernetes.io/docs/home/

[4] "Get Started with GitLab CI/CD | GitLab." Accessed: Jun. 02, 2024. [Online]. Available: https://docs.gitlab.com/ee/ci/

[5] "Terraform by HashiCorp." Accessed: Jun. 02, 2024. [Online]. Available: https://www.terraform.io/

[6] "Homepage | Ansible Collaborative." Accessed: Jun. 02, 2024. [Online]. Available: https://www.ansible.com/

[7] "Cloud Application Platform | Heroku." Accessed: Jun. 02, 2024. [Online]. Available: https://www.heroku.com/

[8] "Red Hat OpenShift | Container Platform für Unternehmen." Accessed: Jun. 02, 2024. [Online]. Available: https://www.redhat.com/de/technologies/cloud-computing/openshift

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

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

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

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

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

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

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

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

[17] "One-Click-Operator/Controllers/Rollout_controller.Go at Main · Janlauber/One-Click-Operator." Accessed: May 04, 2024. [On-

line]. Available: https://github.com/janlauber/one-click-operator/blob/main/controllers/rollout_controller.go

[18] "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

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

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

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

[22] R. Montti, "Google Lighthouse." Accessed: May 11, 2024. [Online]. Available: https://www.searchenginejournal.com/chrome-lighthouse/235465/

[23] "Codefactor, Analysis Tools (Open Source)." Accessed: May 11, 2024. [Online]. Available: https://support.codefactor.io/i24-analysis-tools-open-source

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