INS | Institute for Network and Security

OST Ostschweizer Fachhochschule

18.12.2025, 19:09 - Semester Thesis - Fall Term 2025

Enterprise Container Platform

Development of a streamlined platform installation process

Project Team: **Valentino Diller**
valentino.diller@ost.ch

Daniel Schatzmann
daniel.schatzmann@ost.ch

Roman Weber
roman.weber@ost.ch

Project Advisor: **Jan Untersander**
jan.untersander@ost.ch

Version: **v1.1**

Git Version: **e4a5e00b**

Abstract

Initial Situation

For Software-as-a-Service providers who want to run their software in the cloud, provisioning a container orchestration platform can be completed within minutes, whereas deploying such a platform on local hardware remains significantly more complex and requires knowledge across multiple infrastructure layers. This thesis addresses this challenge by developing a minimal viable product (MVP) that automatically deploys an Enterprise Container Platform on bare-metal hardware.

Approach / Technology

In a first step, a conceptual layer model inspired by the OSI stack was designed to identify the required components. The evaluation showed that a combination of Talos, Kubernetes and complementary plugins for networking, storage and observability is well suited for enterprise environments, serving the needs of a wide range of software providers.

Based on these insights, an automation tool was written in Go. Through a terminal-based user interface, the tool collects user input and generates the required configuration files. These files define the desired state of the plugins, which then get deployed automatically.

Conclusion

The product of this thesis successfully provisions a complete, functional Kubernetes-based container platform. It allows a cluster to be deployed in a few minutes and without requiring detailed knowledge of the components involved. The tool could be extended by adding support for more components, for example a second network or storage plugin. Additional future work includes integrating lifecycle management features to support clusters throughout their entire operational lifetime.

Management Summary

Container-based platforms have become a key foundation of modern IT infrastructures. Cloud service providers enable organizations to provision container platforms within minutes, offering high flexibility and changes in an instant. As a result, many software providers and enterprises rely heavily on these services.

In contrast, deploying comparable orchestration platforms on local hardware remains significantly more complex. Such deployments require deep knowledge across multiple infrastructure layers, including hardware, networking, operating systems, and platform orchestration. This complexity leads to longer deployment times, higher costs, and a dependency on specialized staff. For many organizations, especially those with cost or data protection requirements, this represents a major barrier to adopting container technologies outside the public cloud.

This thesis addresses this gap by exploring how the deployment of an enterprise-grade container platform on bare-metal hardware can be simplified and accelerated through automation.

Objectives

The primary objective of this work was to design and implement an **MVP** that enables the automated deployment of an enterprise container platform on bare-metal infrastructure. The focus was on achieving the following goals:

- Reduction of manual installation effort
- Minimal required technical expertise from the user
- Enabling a quick deployment within minutes
- Suitability for enterprise environments
- Modular design for component interchangeability

The result was intended not only as a technical proof of concept, but also as a foundation for further development toward production-ready solutions.

Aproach and Technology

As a first step, a conceptual layer model was created. This model was used to identify and structure all required components of an Enterprise Container Platform (**ECP**) in a clear and comprehensible way. By separating responsibilities across layers, the model helped to reduce complexity and split the workloads. Based on this analysis, a combination of established and widely adopted technologies was selected:

- Talos as a lightweight, security-focused operating system designed specifically for container workloads
- Kubernetes as the central platform for container orchestration
- Additional components to meet enterprise requirements:
  - Cilium as the Container Network Interface (**CNI**)
  - Longhorn as the Container Storage Interface (**CSI**)
  - A customized observability stack providing the ability to gather logs and metrics
  - A further set of plugins to enable vulnerability scanner, secrets management, and certificate provider functions if wanted.

This technology stack provides a robust, standardized, and vendor-independent foundation suitable for a broad range of software providers and enterprise use cases.

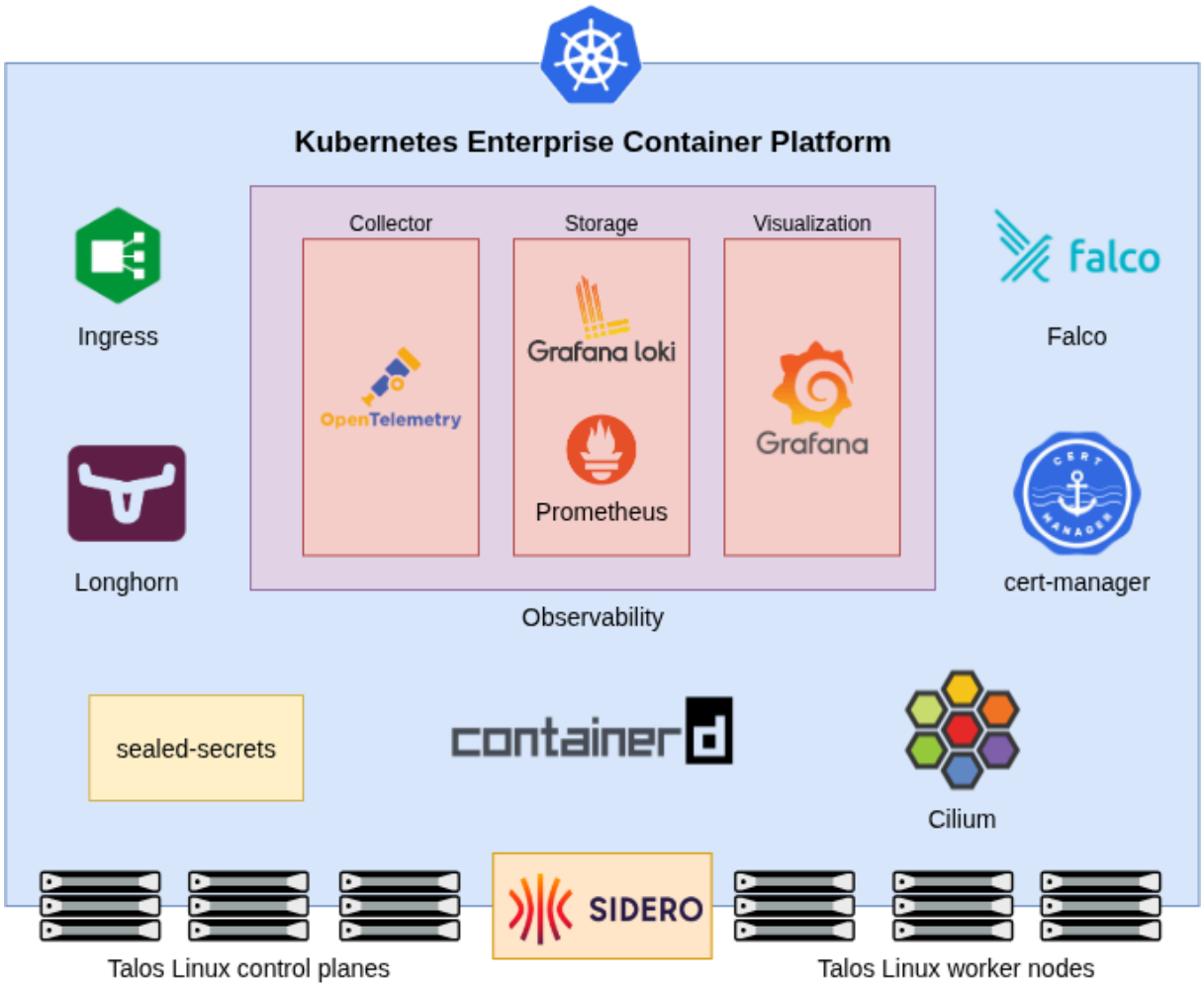

Figure 1: Overview of the Kubernetes cluster with the chosen components.

Implementation and Results

Building on the conceptual model, an automation tool was developed using the Go programming language. The tool provides a Terminal User Interface (TUI) that guides users step by step through the deployment process.

Instead of requiring the deployer to specify full configuration files, the tool collects only high-level inputs from the user. Based on this input, it automatically generates all required configuration files and initiates the full provisioning process.

The resulting MVP successfully deploys a fully functional Kubernetes-based container platform from scratch on bare-metal hardware. The automated approach drastically reduces deployment time and complexity, enabling cluster provisioning within minutes and without in-depth knowledge of the underlying components.

Figure 2: Screenshot of the TUI requesting input from the user.

Conclusion and Outlook

The work demonstrates that automated deployment of enterprise container platforms on bare-metal hardware is not only feasible but highly effective. The developed Minimal Viable Product (MVP) proves that complex infrastructure can be abstracted in a way that makes it accessible to a broader audience.

Future work could extend the tool by supporting additional components, such as alternative networking solutions, as well as by integrating lifecycle management capabilities by deploying a custom operator on the cluster. These enhancements would allow the solution to support container platforms throughout their entire operational lifecycle, from initial deployment to ongoing maintenance and updates.

In the long term, the presented approach has the potential to bring the simplicity and speed of cloud-based container provisioning to on-premises environments, while preserving full control over infrastructure and data.

Contents

Part I

Project Overview

1. Project Background

1.1. Challenge

For enterprises, running their containerized software in the cloud is a pretty straight forward process. They get their credit card, go to a cloud provider such as Amazon AWS or Microsoft Azure, and order their solution, be it a single container or an orchestration solution, like a Kubernetes Cluster. Within minutes, their software is up and running and accessible from the whole world.

However, there are also businesses, where running software in the cloud isn't always an option. These could be banks, healthcare institutions or other firms dealing with confidential data. These companies rely on running their software in house, which brings another whole set of challenges with it. They need to have a physical infrastructure, and more important, someone who installs and maintains it. Employing additional staff is often times cost intensive and the right people are hard to find.

1.2. Project Proposal

Knowing this challenge, the goal of this project is to evaluate and develop a solution, which streamlines the process of installing and maintaining an **ECP**. The clients should be able to have a guided installation process, much like when they would install a software on their computer, and their infrastructure gets provisioned to their likings. This provides them an platform, where they can run their software for themselves or their tenants, much like a Platform as a Service (**PaaS**) solution in the cloud.

This involves installing the Operating System (**OS**) on their servers and configuring the container orchestration solution. The different plugins needed for the networking, storage etc. will be evaluated and configured by the proposed solution. All to the point where the customer can start deploying their software, without needing to deal with the underlying infrastructure.

1.3. Project Conditions

This project is being carried out in the context of the Semester Thesis (**SA**) at the Eastern Switzerland University of Applied Sciences. Having a scope of 8 European Credit Transfer System (**ECTS**) credits, each team member needs to invest 240 hours into this project, resulting in a total of 720 hours.

1.4. Project Management

The project management method chosen is Scrumban, due to the flexibility it offers while also providing the visual element of the Kanban board.

2. Requirements

The initial project assignment states the following goals:

1. Identify essential components
2. Conduct market research and evaluate available technologies
3. Develop a Minimal Viable Product (**MVP**) of an automation framework

The target group needs to be defined with these goals in mind. For this case, a scenario with a fictional enterprise named "Software DW" with the following key points was created:

- Swiss-based company with 250 employees
- Single office location
- IT infrastructure consists of 30 physical servers that run Virtual Machines (**VMs**) on VSphere
- A new software product they want to launch should be scalable, which requires them to use an enterprise container platform
- They bought five new bare-metal servers that they intend to run said container platform on
- The new platform should use open-source tools backed by a known vendor or big community, where they can get help if needed.

This scenario helps to define the functional and non-functional requirements.

2.1. Functional Requirements

Based on the given scenario, there are two actors defined for this project:

- Deployer
- Operator

The deployer is the person who wants to deploy an **ECP**. Instead of installing the operating system on their hardware and configuring all the tools from scratch, they use the tool developed in this SA which automatically bootstraps the cluster for the operator to use it. The operator then has the freedom to deploy applications, as well as monitor and maintain the platform. Based on this, the following use case diagram was created.

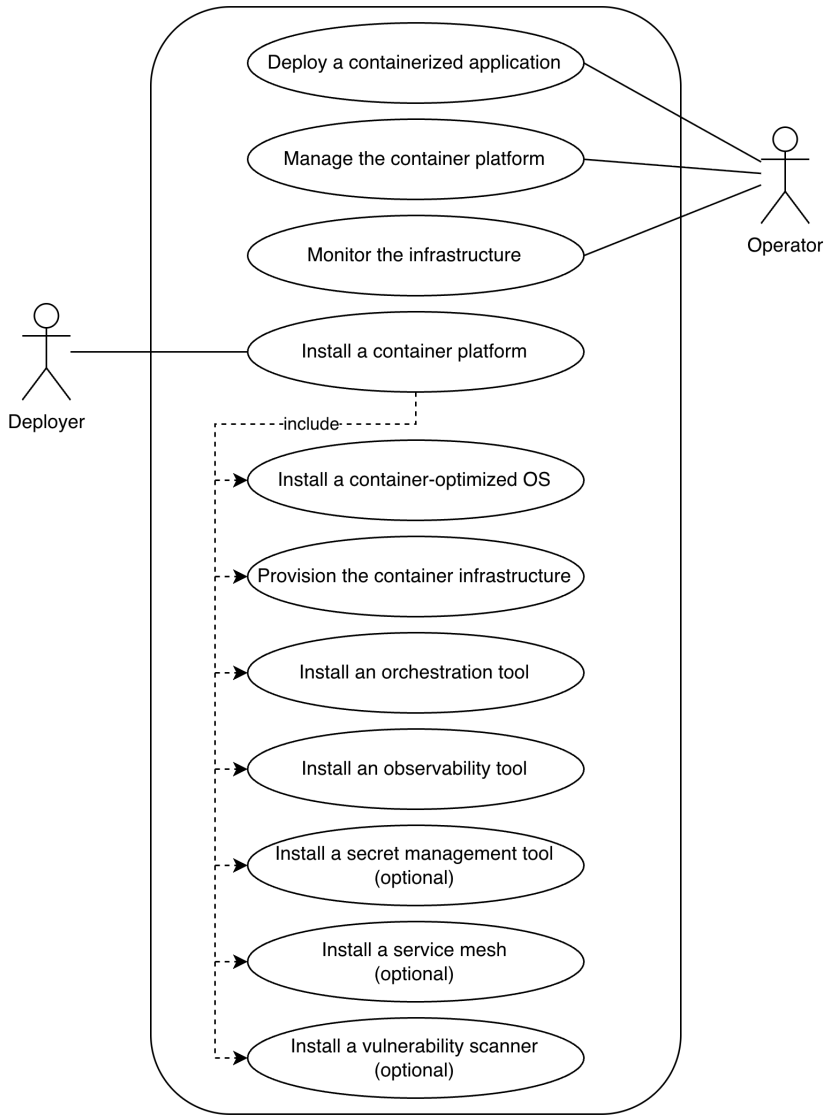

Figure 3: Use case diagram

In this diagram, the optional use cases are available as options, and the deployer can choose to install them or not.

The goal of this **SA** is to write a tool, which automates the part of the deployer. Anything that an operator would do or is a nice to have for him, is out of scope for the evaluation and the developed software itself.

Considering these use cases, the functional requirements were defined:

FR 1 - Install OS Layer

ID	Title	Priority
FR 1.1	Setup Operating System	High

Table 1: Functional Requirements: Install OS Layer

FR 2 - Install Provisioning Layer

ID	Title	Priority
FR 2.1	Install vulnerability scanner	MVP
FR 2.2	Install secret management system	High
FR 2.3	Install key management system	Medium
FR 2.4	Install key management system	Medium

Table 2: Functional Requirements: Install Provisioning Layer

FR 3 - Install Runtime Layer

ID	Title	Priority
FR 3.1	Install Cloud Native Storage plugin	MVP
FR 3.2	Configure persistent storage	MVP
FR 3.3	Install Container Runtime	MVP
FR 3.4	Install Cloud Native Network Plugin	MVP
FR 3.5	Configure container networking capabilities	MVP

Table 3: Functional Requirements: Install Runtime Layer

FR 4 - Install Orchestration Layer

ID	Title	Priority
FR 4.1	Install container orchestration tool	MVP
FR 4.2	Setup orchestration cluster	MVP
FR 4.3	Install observability stack	MVP
FR 4.4	Configure observability dashboards	MVP
FR 4.5	Install service mesh	Medium

Table 4: Functional Requirements: Install Orchestration Layer

FR 5 - Software Features

ID	Title	Priority
FR 5.1	Provide options to install features	MVP
FR 5.2	Advanced configuration	Medium

Table 5: Functional Requirements: Software Features

2.2. Non-Functional Requirements

The non-functional requirements were determined based on ISO/IEC 25010 standard. However, only the characteristics usability and maintainability were considered within scope of this project, other quality characteristics mentioned in the standard were explicitly set out of scope due to resource and time constraints. The non-functional requirements are described using SMART criteria, which stand for Specific, Measurable, Achievable, Relevant, and Time-bound. These criteria help ensure that the requirements are well-defined and can be effectively implemented and evaluated.

ID	Description	
1	**Usability**	
1.1	**Subject**	User Interface
	Requirement	Usability
	Priority	Medium
	Specific	Error messages must be visible to the user and provide clear indication about the cause of the issue.
	Measurable	All error messages are displayed in a consistent format and include clear indication about the cause.
	Achievable	The development team has the skills and resources to implement user-friendly error messages.
	Relevant	Clear error messages enhance user experience and reduce frustration.
	Time-bound	Implemented prior to the MVP release.
	Validation	Usability testing with target users.
1.2	**Subject**	User Interface
	Requirement	Usability
	Priority	Medium
	Specific	Loading indicators must be present during installation.
	Measurable	During the installation a loading indicator shows the status until the operation is complete.
	Achievable	The development team can integrate loading indicators into the user interface.
	Relevant	Loading indicators improve user experience by providing feedback during operations.
	Time-bound	Implemented prior to the MVP release, subject to time availability.
	Validation	Usability testing with target users.
2	**Maintainability**	
2.1	**Subject**	Code Quality
	Requirement	Maintainability
	Priority	High
	Specific	The codebase must be descriptive.
	Measurable	Code reviews ensure that the code is understood by a second developer.
	Achievable	The development team follows coding standards and best practices.
	Relevant	A descriptive codebase reduces maintenance effort and facilitates future development.
	Time-bound	Ongoing throughout the development process.

ID	Description	
	Validation	Code reviews and peer feedback on merge requests within the ecp-deployer repository

Table 6: Non-Functional Requirements

Part II

Product Documentation

3. Evaluation

Container platforms were used by the project team before, but the sheer size of the cloud native landscape and the process of deciding the required components and which ones to evaluate posed a challenge. This chapter shows the decision process from the theoretical evaluation to what was deployed in the cluster.

3.1. Evaluation Method

The evaluation methodology comprised the following steps:

1. Group cloud native landscape into categories (Chapter 3.3).
2. Define criteria relevant to category and use cases (Chapter 3.4).
3. In-depth analysis of components per category with decision matrix and conclusion (Chapter A).
4. Mid-term presentation of evaluated components (Chapter 3.5).
5. Convert evaluation into Architectural Decision Records (**ADRs**) (Chapter B).

After completing this evaluation, a few decisions needed updating with new learnings in the implementation phase. These new findings are represented as updates in the corresponding **ADR**.

3.2. Cloud or Bare-Metal?

One of the main questions at the beginning was whether the installation tool should work for on-premise or cloud-based infrastructure, as this influences the installation process. Because the process of getting a running Kubernetes cluster in the cloud is relatively quick and easy, the primary focus was set on bare-metal servers. This focus was decided in agreement with the advisor.

3.3. Initial Categories

Initially an overview of the tools to be evaluated was required. Inspired by the categories from the cloud native landscape [1], these are the groups and categories defined. The ones marked as optional were not evaluated.

- *Infrastructure*: Containing the **OS** and hardware
- *Provisioning*: The base which lays the foundation for the subsequent layers
  - Automation & Configuration
  - Security & Compliance
  - Key Management
  - Container Registry (optional)
- *Runtime*: The different parts which provide all the necessities to run containers
  - Cloud Native Storage
  - Container Runtime
  - Cloud Native Network
- *Orchestration & Management*: Tools to orchestrate and manage containers and applications
  - Scheduling & Orchestration
  - Coordination & Service Discovery
  - Observability & Analysis
  - Chaos Engineering (optional)
  - RPC (optional)

  - Service Proxy (optional)
- *App Definition & Deployment*: Possible optional additions
  - Database
  - Streaming & Messaging
  - App Definition & Image Build
  - CI/CD template

3.4. Evaluation Template

To have a structured approach of evaluating the categories, an evaluation template was created. The template consists of the following parts:

- *Selection criteria*: Selection criteria used to filter what components to investigate closer. For instance, the operating system is required to be 'optimized for container runtimes,' thereby limiting the scope of Linux distributions.
- *Evaluation criteria*: Criteria that components are evaluated by. Individual for each category. For instance 'licensing' or 'enterprise readiness'.
- *Components*: List of components to be evaluated per category.
  - *Features*: Feature list of the component.
  - *Challenges*: Challenges when using or implementing component.
  - *Criteria*: Evaluation criteria for each component.
- *Decision matrix*: Decision matrix for visual understanding.
- *Conclusion*: Argumentation of component selection.

The detailed evaluations per category are in the appendix under Chapter A.

3.5. Mid-Term Presentation

Following the theoretical evaluation phase, the preliminary findings were presented during the mid-term presentation on October 25, 2025. This session served to validate the selected components and gather feedback.

- The evaluated components were confirmed to be compatible and appropriate.
- The use of Architectural Decision Records (**ADRs**) was recommended to track the evolution of design decisions. This approach ensures that changes to initial decisions are formally documented throughout the project lifecycle. These **ADRs** are documented in Chapter B.

3.6. Results

The following table summarizes the evaluation results. The 'App Definition & Deployment' category was not evaluated and is therefore excluded.

Category	Group	Evaluated components	Selection
Infrastructure	Operating System	Talos Linux, Flatcar	**Talos**
Provisioning	Automation & Configuration	KubeEdge, Terraform, Onmi by Sidero, Metal³	-
	Identity provider	Zitadel, Keycloak	**Zitadel**
	Vulnerability scanner	Grype, Kubecape, Falco	**Falco**
	Linter & static code checker	KubeLinter, Checkov, Clair(by Redhat)	-
	Secret Management	sealed-secrets, Secrets OPerationS (**SOPS**)	**sealed-secrets**
Runtime	Cloud Native Storage	CubeFS, Rook, Longhorn	**Longhorn**
	Container Runtime	containerd, CRI-O	**containerd**
	Cloud Native Network	Cilium, Calico, Flannel	**Cilium**
Orchestration & Management	Scheduling & Orchestration	Kubernetes, Nomad Enterprise	**Kubernetes**
	Obvservability & Analysis	• *Visualization*: Grafana, OpenSearch Dashboards • *Log aggregation*: Fluent Bit, Fluentd, OpenTelemetry Collector, Alloy, Vector • *Log storage*: OpenSearch Core, Grafana Loki • *Metrics collection*: Prometheus Operator, OpenTelemetry Collector, Vector • *Metrics storage*: Prometheus (Thanos), Grafana Mimir	• **Grafana** • **Open-Telemetry Collector** • **Grafana Loki** • **Prometheus**

Table 7: Evaluation Results

4. Implementation

Since most of the work up to this point has been theoretical, it was necessary to verify that all components function as intended. To achieve this, a cluster was deployed manually serving as a Proof of Concept (**PoC**) of the stack. This process provides insights into which components require specific configurations and which tasks need automation. Based on these findings, the automation tool was developed.

4.1. Architecture

The architecture described in this chapter focuses on the ecp-installer and not on the components in the Enterprise Container Platforms (**ECPs**). The documenation is based on the arc42 template.

4.1.1. Introduction and Goals

The installer written in this project is similar to a regular software installer for any kind of program. Instead of installing software, it will provision an Enterprise Container Platforms (**ECPs**).

1. The deployer starts the tool ecp-deployer from the command line on his computer and enters the settings according to his needs.
2. The ecp-installer bootstraps configuration files based on the entered settings and automatically deploys and configures the cluster.
3. After the successful cluster deployment, the operator can take over managing the container platform.

4.1.2. Constraints

The ecp-deployer shall be:

- Runnable from the command line.
- Platform-independent and support Windows, Linux and macOS.
- Fulfill the FR and NFRs according to Chapter 2.1 and Chapter 2.2.
- Fulfill the requirements of the fictional scenario Software DW.

4.1.3. Context and Scope

The context diagram uses the deployer and operator as defined in Chapter 2.1.

- The deployer runs ecp-deployer and enters required input.
- ecp-deployer is the installation tool developed in this project.
- The Enterprise Container Platform (**ECP**) is the end product after running the ecp-deployer.
- The operator uses the Enterprise Container Platform (**ECP**) after the successful deployment.
- Artifact repositories are external software sources needed during the cluster installation.

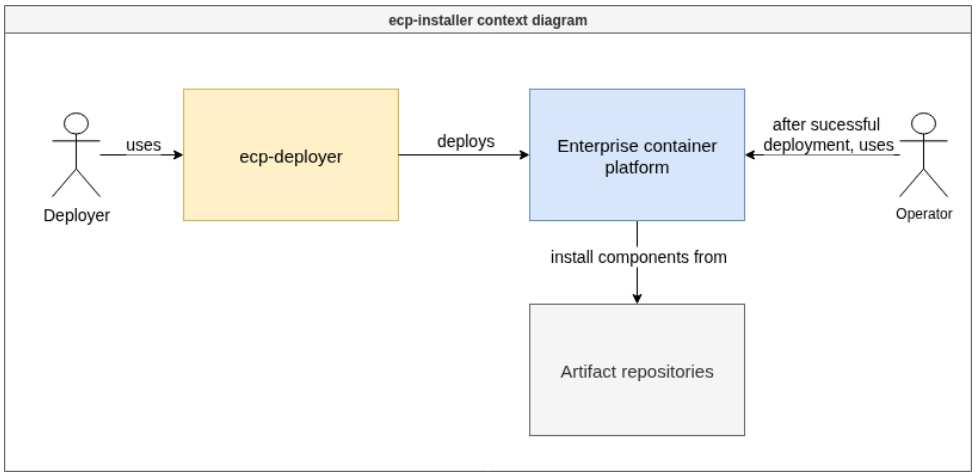

Figure 4: arc42 Context Diagram

4.1.4. Solution Strategy

1. As defined in ADR-13, the tool is written using Go.
2. To avoid import cycles, a model was defined that is implemented by the other code parts.
3. Said model uses interfaces to make the code adaptable and minimize coupling.

4.1.4.1. Level 1

The diagram shows the structure within the Go project. This overview shows how the code is separated into different packages. The packages call each other by calling Go functions.

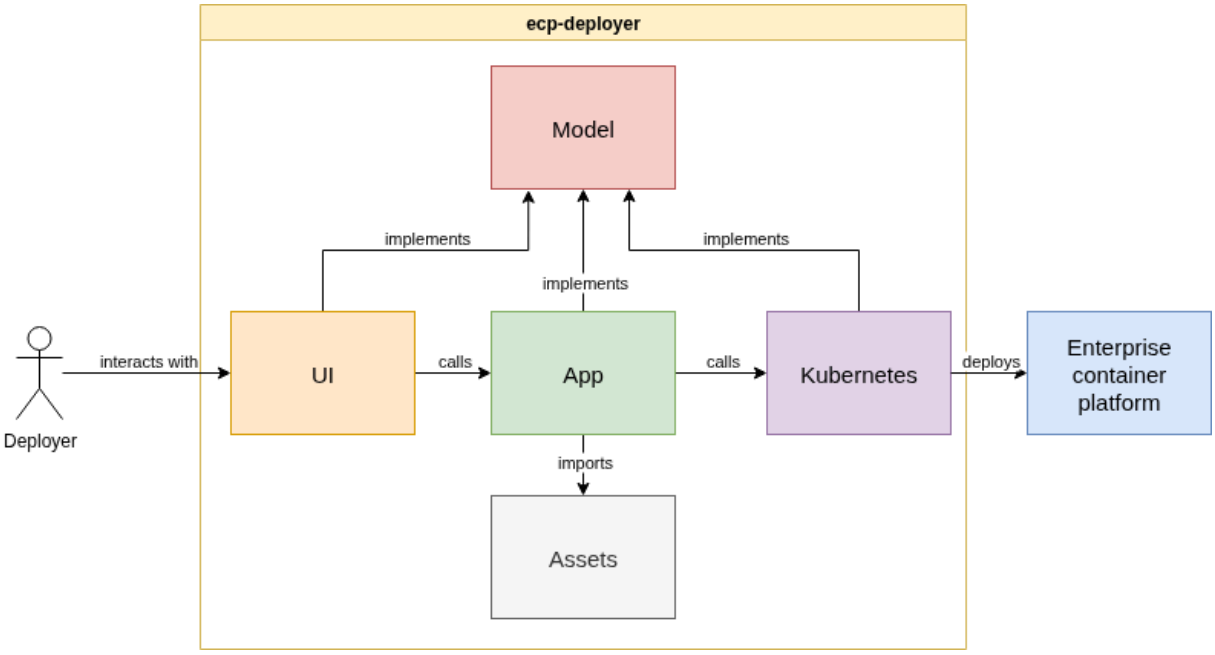

Figure 5: Architecture Level 1 Diagram

Building blocks	Description
Model	Defines interface StackComponent and installation order. The model is implemented across all parts of our software product.
UI	Capture user input and shows cluster deployment progress.
App	Parses UI input and contains business logic. Calls installer to deploy tools on container platform.
Kubernetes	Kubernetes doesn't refer to k8s, but to code written in this project. It contains functions that deploy the cluster using Kustomize or Helm.
Assets	Contains static (yaml) files.

Table 8: Architecture Level 1 Building Blocks

4.1.4.2. Level 2

The level 2 diagram focuses on the app part, as this contains the application logic of the script.

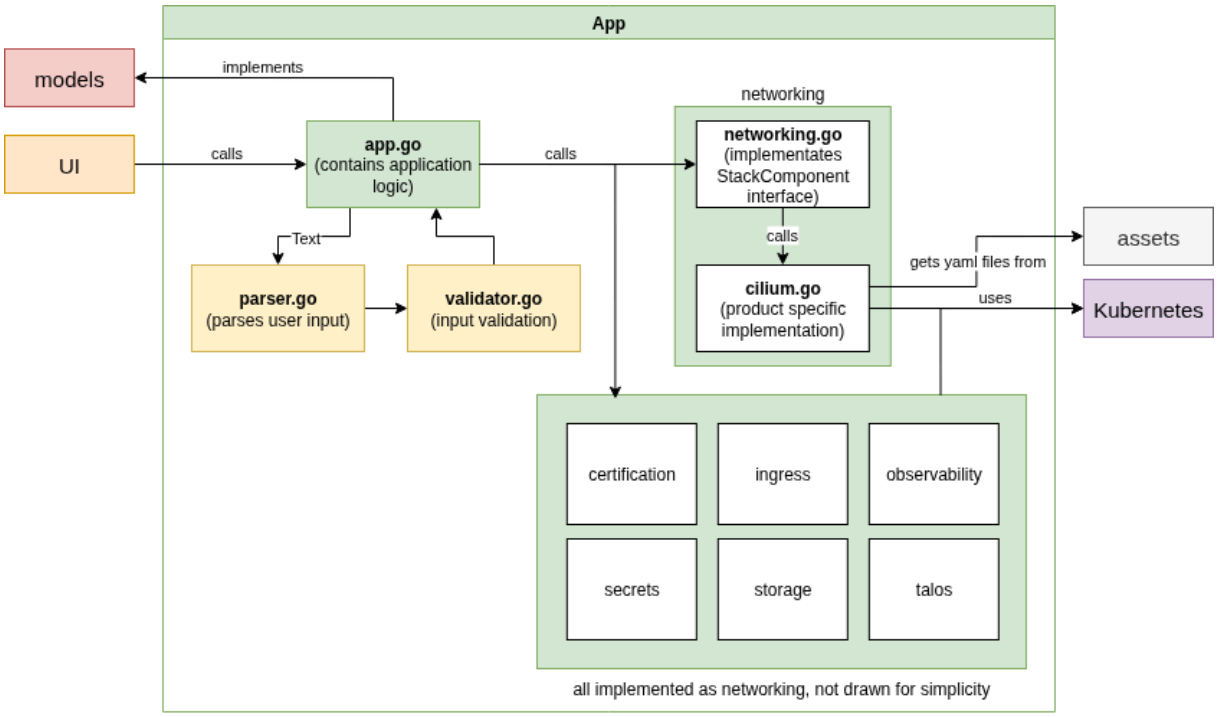

Figure 6: Architecture Level 2 Diagram

Building blocks	Description
app.go	Implements two main functions called NewStackComponent and DeployECP. StackComponents are added to a slice, DeployECP iterates over it and calls prepare and deploy functions.
parser.go	Parses user input into correct format. For example by splitting the string to IPs.
validator.go	Validates user input to only accept valid IPs or URLs.
networking.go	Implements Stack component interface with the intention to easily add more network plugins.
cilium.go	Prepares cilium related yaml files and deploys them on the cluster.

Table 9: Architecture Level 2 Building Blocks

The components in the purple box are implemented identical to networking, but have been left out to keep the diagram simple.

4.1.5. Runtime View

The runtime view of our installer is fairly simple. Once the user has entered all the settings, the app and the installer will sequentially install all selected components.

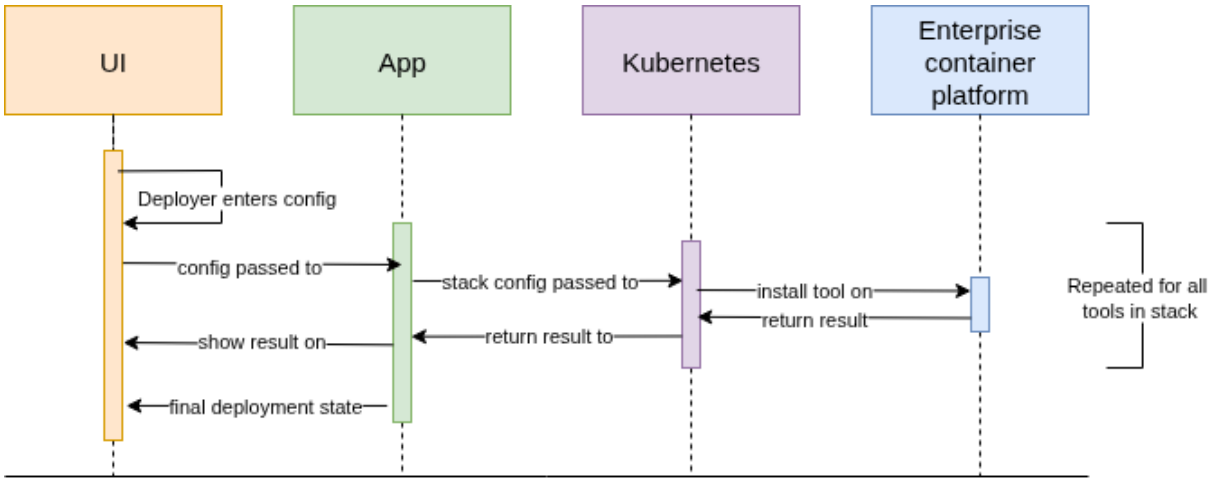

Figure 7: arc42 Runtime View Diagram

4.1.6. Deployment View

Our deployment uses proxmox to virtualize the servers needed to run our project.

- ecp-deployer: run locally on the device of the deployer.
- Proxmox: six virtual machines configured, three worker nodes and control panels each.
  - Snapshot feature of proxmox used to jump back or forward to a working configuration.

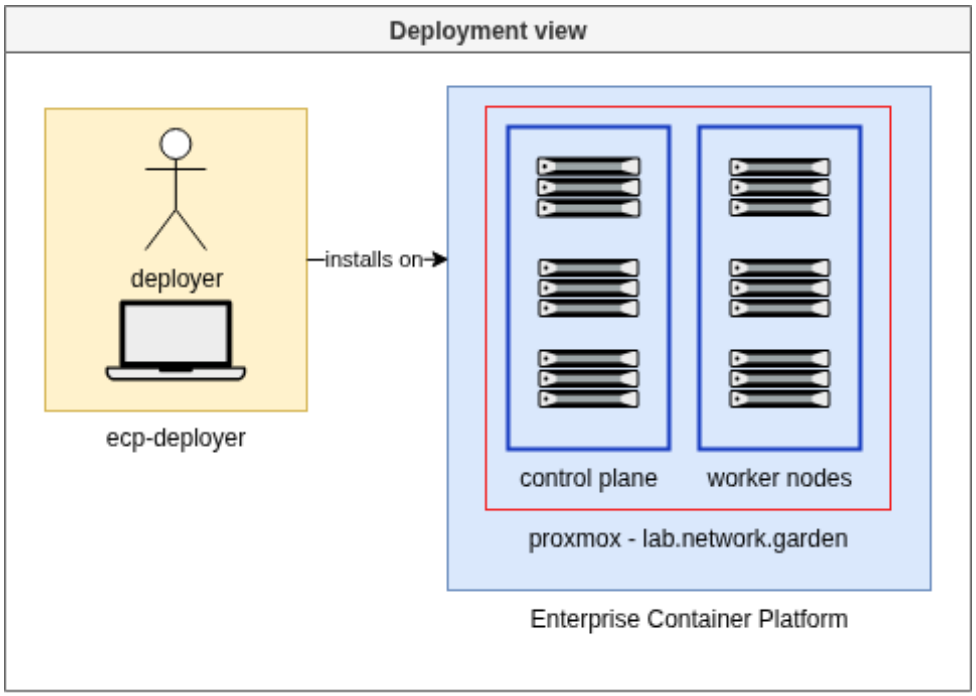

Figure 8: arc42 Deplyoment View Diagram

4.2. Code Structure & Packages

This chapter describes the project's code structure and lists the main packages. The project organizes its code into distinct packages. Each package implements a focused responsibility: data models, asset management, core application logic, Kubernetes integration, and User Interface (UI).

- Models: The `models` package defines domain types and the Yet Another Markup Language (YAML) representation of components. These types support validation and serialization.
- Assets: The asset subsystem centralizes embedded templates and static resources. It loads and transforms assets for template rendering.
- App (core logic): The `app` package implements parsing, validation, component composition, and the deployment orchestration workflow.
- Kubernetes (integration): The `kubernetes` package provides helpers to apply manifests and to integrate with Helm. It encapsulates Kubernetes operations to make higher-level deployment steps reproducible.
- UI: The `ui` implements a terminal-based interface that collects user input, displays deployment status and log feedback, and communicates with the application logic.

4.2.1. Models

The package models defines project wide structures, which then define the rules for the different packages importing from it. The most important one is the interface `StackComponent`, which gets implemented by each of the components packages. These components are also defined in the models, which are:

- OS
- Networking
- Ingress Controller
- Certificate Manager

- Storage
- Secrets Management
- Observability Stack
- Vulnerability Scanner.

Besides several getters and setters, the interface defines two key functions: `StackComponent.Prepare()` and `StackComponent.Deploy()`. These are used to deploy each stack component during the installation process. While this structure comes with code redundancy, it also provides a highly modular setup, so that new stack components or plugins can be added without interfering with existing code. More on that will be covered in Chapter 4.2.3.

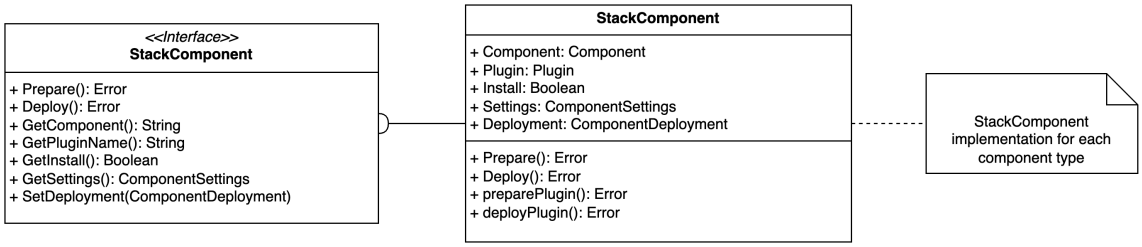

Figure 9: Interface StackComponent

Another struct defined in the models is the `ComponentDeployment`. It is assigned as a struct within the `StackComponent` and holds the plugin name which is to be deployed, the in-memory filesystem which holds the needed files, and the logging object for the **TUI** output. Because `ComponentDeployment` is implemented as a struct rather than an interface, component-specific objects are not required, and it can be used directly.

At last, some configuration files, which are written as **YAML**, were portrayed as structs. Using an external package, it's possible to load the files into these structs, change the values within the object, and rewrite it to the file. This is needed to set the user input within the configuration files. The Extras field in the struct is included so that unknown or additional YAML keys at the same level are merged into the struct during unmarshalling and are written at the top level during marshalling.

```
1 type Metadata struct {
2   ...
3   Extras map[string]interface{} `yaml:",inline"` // preserve any other keys
4 }
```

Code Listing 1: Extras field in metadata struct of generic.go

4.2.2. Assets

The integrated `embed` package is used by the assets package to ship configuration files within the binary. The files are placed in a folder and are exposed through a filesystem of type `embed.FS`. Two filesystems are defined: one for all components and another for **UI** resources. A function is defined that copies all files from a specified folder within the source filesystem recursively into a target filesystem.

4.2.3. App

The package `ecp/deployer/app` represents the whole business logic of the installer and is thus the biggest one. It is responsible for the input validation, parsing and coordinating the installation process of the container platform.

For every stack component, when the Deployer submits the corresponding settings, input validation occurs. These inputs can include Internet Protocol (**IP**) addresses, hostnames or Helm chart versions. If the validation function doesn't return an error, the tool creates an instance of a stack component with the given settings and stores it in the global array `StackComponents`. Afterwards, it creates an in-memory filesystem and uses the function defined in `assets` to copy all the files required for this plugin into the new filesystem. This process is repeated for each configuration page. As soon as the last stack component is submitted, the program starts with the installation routine.

The main function `DeployECP()` is called, which iterates through the array exactly once. It checks each component if it needs to be installed. If so, it modifies the configuration files in the virtual filesystem and installs the component. Since each component may require different preparation and installation steps, a mechanism to handle these variations was necessary. This is where the interface plays it's part, by delegating the implementation to the different components such as networking or storage. In the `Prepare()` and `Deploy()` implementations, each component differentiates between the possible plugins and then calls another function, which implements a custom routine for the specific plugin. At this stage, the Helm chart and additional Kubernetes manifest files are applied.

After the last object was installed, a post installation routine runs. This routine is decoupled from the other components and only consists of a few manifests. Once this is complete, the program exits successfully.

The following subchapters focus on the implementation of different components, which are more complex than just deploying a Helm chart with a set of values.

4.2.3.1. Talos

The standard process to create a Kubernetes cluster on Talos is as follows:

1. Create a set of configuration files using the Talos Command-Line Interface (**CLI**)
2. Boot the servers using a Talos image
3. Apply the according configuration files to the nodes
4. Bootstrap one of the control plane nodes.

At the end, the user can run the Talos **CLI** again to retrieve the `kubeconfig` file and use this file to connect to the Kubernetes Application Programming Interface (**API**). The package `talos` implements this workflow.

To get Talos running on the hardware, the Deployer needs to prepare a **(?? USB ??)** stick with the **OS** image flashed onto it. Using this as the boot device, the servers are running in Maintenance mode and ready to be provisioned into a cluster. From this point on, the program executes the remaining deployment steps.

To generate the Talos configuration, a patch file can be created to override specific values in the default configuration. These patch files can be written for either the control plane or the worker nodes, or for both types together. The patches defined for this deployment specify that the default

CNI *Flannel* should not be installed and that image extensions created with Talos Image Factory[1] should be included. To ensure the extensions match the Talos version, the Deployer specifies the Talos version so the tool can adjust the image tag accordingly.

In addition, the Deployer sets the cluster name, the IP addresses of control plane and worker nodes, the virtual IP shared by the Talos cluster, and the endpoint under which the Kubernetes API should be reachable. This information is required to generate the configuration files and execute the preparation step. The `Deploy()` method then performs the described process and saves the `kubeconfig` file in the working directory.

4.2.3.2. Networking

As mentioned in the Talos section, the **OS** must to be configured so that the default **CNI** is not present on the cluster. Consequently, when Talos is bootstrapped and the Kubernetes **API** becomes reachable, the nodes report their status as "not ready", because some pods aren't running due to the missing networking capabilities.

The base installation of the selected plugin *Cilium*, follows the official documentation. However, an additional feature was implemented to enable **IP** announcement on Layer 2. The Deployer has the option to enable this setting and directly define an **IP** block, which services can use and listen to. If this feature is selected, additional values are applied during the Helm installation, and two Kubernetes manifest files are deployed using a Kustomization.

4.2.3.3. Storage

With *Longhorn* as the chosen storage plugin, the installation follows the official Longhorn docs. It is essential that the Talos installation is patched accordingly to define the image and folder location for the worker nodes. Beyond this adjustment, the Deployer only needs to specify the hostname where the Longhorn frontend should be accessible. This frontend is deployed as an NGINX ingress.

4.2.3.4. Observability

Since the log collector requires access to node metrics and host-level access for Kubernetes logs, the observability namespace is configured as privileged.

The base installation of the selected plugin stack, `OTEL_LOKI_PROMETHEUS_GRAFANA_STACK`, consists of four main components: *OpenTelemetry Collector*, *Grafana Loki*, *Prometheus*, and *Grafana*. Each component is deployed using Helm charts with custom values to ensure seamless integration.

OpenTelemetry Collector

The *OpenTelemetry Collector* is configured to collect logs and scrape metrics from various sources within the cluster. Two deployment modes are used:

- DaemonSet: Runs on every node to collect container logs and host-level metrics using the Prometheus receiver.
- Deployment: Acts as a centralized aggregator for telemetry data.

Both configurations export logs to *Grafana Loki* and metrics to *Prometheus*. The setup includes of receivers for telemetry collecting, presets for log and metric enrichment and exporters for *Grafana Loki* and *Prometheus*.

[1] https://factory.talos.dev

Loki

Loki stores and indexes logs collected by the *OpenTelemetry Collector*. The Helm chart deploys Loki as a single binary with a persistent volume for log storage, ensuring durability. The configuration includes log retention policies and indexing optimizations, such as enabling `chunksCache` to speed up queries by caching recently accessed log chunks. To secure log ingestion, the exposed *Grafana Loki* service is protected with basic authentication.

Prometheus

Prometheus is configured as a metrics storage backend. Scraping is intentionally disabled because the *OpenTelemetry Collector* performs this task. The Helm chart disables service discovery and scrape intervals, while retention policies and persistent storage are configured to ensure efficient metric storage.

Grafana

Grafana visualizes both logs from *Grafana Loki* and metrics from *Prometheus*. The Helm chart provisions data sources for *Grafana Loki* and *Prometheus* and sets up default dashboards for monitoring Kubernetes cluster health, performance, and Longhorn storage. The *Grafana* service is exposed via a LoadBalancer and secured with basic authentication.

4.2.3.4.1. Improvements

Future enhancements could include:

- Integrating Alertmanager to notify administrators of critical events.
- Implementing authentication for metric ingestion into Prometheus.
- Adding an option in the ecp-deployer to specify the version of each component during deployment.

4.2.4. Kubernetes

All components, except for the **OS**, require deployment using Kubernetes resources. Both Kustomize and Helm provide extensive features for streamlining such deployments. The initial concept was to use a single `kustomization.yaml` file per component, which would apply all required resources. The main Kubernetes objects would be generated using Helm's templating capabilities.

However, when considering maintainability, a significant drawback of this approach became evident. Applying raw manifests from Helm charts without using Helm's installation process prevents Helm from registering the deployed chart on the cluster. This limitation complicates future maintenance tasks, such as upgrading the chart version or modifying configuration values. Based on this insight, the functionality of the package was adapted to support Helm installations using charts and values provided as assets. Additionally, it allows either deploying a single Kubernetes manifest or applying a Kustomization.

Another challenge was handling version changes in Helm releases. Hardcoding versions into configuration files would require providing a new binary of the ecp-deployer for every update, which is not a viable solution. To implement dynamic version handling, the tool checks the `index.yaml` of the corresponding chart, which must always be available[2]. If the entered version isn't listed in this file, the validation fails. This approach allows the Deployer to select the version for each Helm release and ensures that outdated versions are not used.

4.2.5. UI

The UI is implemented using BubbleTea, a Go framework for building terminal applications, and styled with Lip Gloss.

Figure 10: User interface of ecp-deployer

The UI primarily contains code related to BubbleTea. To minimize coupling, the application is invoked only twice:

- `app.NewStackComponent()` to create StackComponent objects per page upon user submission.
- `go app.DeployECP()` to start the cluster deployment.

4.2.5.1. Channel

`go app.DeployECP()` starts the cluster deployment in a separate goroutine. This second goroutine must stream log data back to the UI so the installation state can be rendered. This doesn't work with function calls, as this results in a cyclic import. In Go, channels are used for two goroutines to communicate. The app part sends log entries into the channel, where the UI consumes these entries as shown in the code snippet below and renders them correctly into the BubbleTea viewport.

```go
type model struct {
  // left out for brevity
  logChannel         chan string
  logs               []string
}

// waitForLog creates a Bubble Tea command that listens for log messages from a channel.
func (m *model) waitForLog(ch chan string) tea.Cmd {
  return func() tea.Msg {
    msg := <-ch
    if msg == "ECP Installation completed" {
      return deployCompleteMsg{}
    }
    return logMsg(msg)
  }
}

```

Code Listing 2: Channel in ui.go

4.2.5.2. Improvements

Two UI enhancements were identified but not implemented due to time constraints:

- A installation summary page.
- Improved visual distinction between the deployment steps of individual components.

The current communication channel utilizes the `chan string` data type, and the implementation renders messages as plain text. To enhance functionality, a custom data structure could be introduced to encapsulate additional metadata, such as the log severity level and the originating component. This modification would enable rendering features like including color-coded output based on log severity or a better distinction between deployment steps.

4.3. Testing

Testing of our code and the deployed cluster components was threefold:

- **Go tests**: Go unit testing of the code.
- **Manual cluster verification**: Checking that deployed components are healthy.
- **Usability test**: Feedback from test users.

4.3.1. Go Tests

In the pipeline that builds the Go project, `go test` is run and a code coverage report is created by the test stage. A failing test causes a failed pipeline, which prevents merging. No minimal code coverage threshold was defined, instead tests were written on a best-effort basis. To give an example, tests were written for the parser and the validator, but not for the deployment of the cluster. Mocking an entire Kubernetes cluster was deemed unnecessary.

4.3.2. Manual Cluster Verification

Each component deployed on the cluster was verified manually. The specific verification procedure depended on the component; for example, Cilium was considered healthy if all pods were in the `Running` state when `kubectl -n kube-system get pods -l k8s-app=cilium` was executed.

Once a component had been deployed successfully, the codified deployment process ensured that the component was deployed identically in subsequent runs and could therefore be expected to be healthy.

4.3.3. Usability Test

The usability test was kept small, as there is little input required from the deployer. The goal was to verify the user interface, installation documentation and if the non-functional requirements (**NFRs**) from Chapter 2.2 are achieved.

4.3.3.1. User Selection

People with an IT background were selected as the target audience. The usability test was conducted with two people working/studying in computer science.

4.3.3.2. User Story

Six servers have Talos installed, but no further configuration has been done. With the ecp-deployer and documentation provided, the user is asked to deploy an enterprise container platform. At the end, the Kubernetes cluster should be running with all components deployed.

4.3.3.3. Success Definition

The test is considered successful if the **ECP** can be deployed by the user with minimal input (ideally only the IPs) from the project team.

4.3.3.4. Conducting Test

The following steps were involved in conducting the test:

0. Servers are reset by the project team.
1. User is given an overview what the tool does.
2. A guided run-through is performed, the user checks the documentation and enters the settings himself. IPs are provided by the project team.
3. User is asked to answer questionnaire.

4.3.3.5. Questionnaire

- *Question1*: Is the **TUI** interface intuitive?
- *Question2*: What is missing to make it better or easier to use?
- *Question3*: Were any error messages encountered, and did they provide guidance for troubleshooting?
- *Question4*: What in the documentation can be improved to make it easier to understand?

4.3.3.6. Conducting the Usability Test

Name	**Gian Hunold**
Date	**04.12.2025**
Question1	The **TUI** is understandable and navigating through the configuration steps works as expected.
Question2	• Being able to input IP ranges like 10.8.38.57-59. • Use tab to autocomplete placeholders. • Show installation results after the cluster is set up.
Question3	The error message informed me that the entered IP was of invalid format.
Question4	The documentation does a good job of explaining the settings to enter. Without the explanation from the project team before the run-through, the context of what the tool and what each component does is lacking.

Table 10: Usability Test Gian Hunold

Name	**Yanik Breitenmoser**
Date	**12.12.2025**
Question1	Yes, the **TUI** is easy to understand and buttons work as expected.
Question2	• Mark optional features as optional (suggestion *) in the sidebar. • Provide information about the plugin that is being installed. • Show what plugin is being installed, for example the network plugin doesn't mention cilium in the installer.
Question3	The error message showed that the IP of the Talos server was not reachable. This was the only error encountered.
Question4	Having little know-how of Kubernetes, it would be great if the description of the components would be more detailed. Also an overview of the cluster that is deployed in the end is missing.

Table 11: Usability Test Yanik Breitenmoser

4.3.3.7. Results

The usability tests were evaluated against the **NFRs** outlined in Chapter 2.2:

- **1.1** Error messages: This **NFR** was successfully validated; the error messages displayed in the user interface were comprehended by the users.
- **1.2** Loading indicators: This **NFR** was validated with partial success. Although logs are displayed in the frontend, both test participants suggested improvements, specifically:
  - Displaying the final installation results.
  - Providing a clearer indication of the component currently being installed.

Due to time constraints, not all findings could be addressed. The following changes were implemented:

- A cluster overview was added to the `README.md` in the `ecp-deployer` repository.
- Implementation of the tab autocompletion feature.

- Revision of the documentation to provide additional context regarding the `ecp-deployer` and the associated components.

4.4. Running ecp-deployer

The ecp-deployer repository contains a detailed markdown with the prerequisites and explanations to run it. It runs on Linux, macOs and even Windows.

5. Results

This chapter draws a conclusion about the whole project. It focuses on the final product and if the project goals have been achieved, future work, personal reports and draws an overall conclusion.

5.1. Final Product

The final product is a command-line application written in Go that guides users through the configuration and deployment of a Kubernetes cluster based on Talos. The application takes inputs from the user, enabling anyone to deploy a container platform without deeper knowledge of the underlying technologies. The program logic is divided into three stages:

- Parsing and validating user input.
- Generating and modifying configuration files.
- Deploying components in the Kubernetes cluster.

The application uses a **TUI** to collect deployment parameters from the user. These parameters can range from general settings like the cluster name to settings on networking and storage levels. The input gets validated and parsed for each individual configuration step, ensuring correctness and completeness before starting with the installation process.

Based on the input, configuration files get modified to comply with the entered settings. For **YAML** files, it alters values and rewrites them to the filesystem, to make them available for the deployment step.

With the configuration files in place, the component gets installed in the defined order, from Talos to networking and storage to all the optional components like a secrets management tool or a custom-built observability stack. The installation logs are shown to the end user, so that the installation progress can be tracked and errors detected.

5.2. Project Outlook

With the current status of the project, a Kubernetes cluster can be deployed, but not managed. Considering the use-case of Software DW, a software developer might not be able to maintain the cluster properly without additional knowledge.

A possible extension to deal with this problem would be a lifecycle management tool, also automatically installed in the cluster by the ecp-deployer. This operator could provide features like upgrading Talos and Kubernetes or the installed Helm charts. It could also manage the installed Helm charts in terms of creating and deleting new ones or changing the installation values.

To reach a broader user group, the implementation of additional components like a second **CNI** would be possible. Because the code was written with the adaptability in mind, this is an easy addition. Currently the ecp-deployer deploys a ready-to-go cluster if one does not want to deal with the Kubernetes installation. More available components also allow the usage of ecp-deployer when having requirements in terms of installed components.

5.3. Personal Reports

In the personal reports, every team member reflects on the project.

5.3.1. Daniel Schatzmann

Through the evaluation phase, I explored a wide range of tools, particularly in the area of observability. Creating an evaluation that was meaningful and well-structured proved challenging given the large number of available options. The feedback received during our mid-term presentation was extremely helpful and guided my final decision-making process. As a result, I feel confident and satisfied with the choices I made.

Implementing the observability stack was demanding and required significant time. Since there was no complete documentation on how to integrate all components, I had to figure out the connections step by step to ensure a coherent setup. Completing the observability implementation was a relief, although I experienced a certain level of configuration fatigue due to the extensive use of YAML files. This fatigue only motivated me further to move on to automating it in the Go code, which was generally easier because much groundwork had already been done by my colleagues. Nevertheless, I faced unique challenges with saving files while working on Windows, which my colleagues on Linux or macOS did not have.

I greatly appreciated working with Roman and Valentino. We complemented each other well, combining different skills and personalities, which made collaboration both productive and enjoyable. I am confident that we will continue this strong cooperation in our bachelor thesis. Overall, I am very satisfied with the work we accomplished and the results we achieved.

5.3.2. Roman Weber

Even having used Kubernetes before, I was overwhelmed by the amount of tools and components available. At the beginning, with limited knowledge, it is hard to differentiate between relevant and nice to have. The task to create our own cluster and deciding what components use, was the most challenging part of this project. The secret management evaluation turned out to be my arch nemesis. The feedback received from the mid-term presentation proved very valuable. It gave us the confidence that our evaluation was technically correct (for the most part).

Personally, I had the most fun writing the Go code. At first, Valentino and myself sat together and started discussing application architecture. After agreeing on the what is now known as the model, but not being overly confident in our approach, he started deploying the cluster and I started with the **TUI**. The first moment I though that the ecp-deployer will work, was when the **TUI** and the Kubernetes code were married (as in connected). We were able to deploy Talos and Cilium with user input from the **TUI**. Connecting the two parts was easy and proved our earlier decisions right. I am happy with the coded delivered, it is clean, well documented and easily expandible.

The best part was to work in a team with Daniel and Valentino. Our team chemistry is really good, we are a well-balanced team and it is just fun to work with them. I am sure we can carry this momentum over to our Bachelor thesis!

5.3.3. Valentino Diller

When I heard about the project and the task we were given, on the one hand I was excited, on the other I had great respect. While I had made some experiences with Kubernetes during the studies up to this point, it felt like it was way beyond those. Also considering that we also had to create the installer from scratch.

The first few weeks were a bit tedious. We weren't really sure where to start. Once we did, the evaluation work wasn't what I would call fun, but it had to be done. Due to this, and also a few days of military service, I was about 50 hours behind at the half way point. This proved to be an advantage, because I had more time to implement.

Once started, my feeling and motivation for this project grew steadily. I was able to gather new experiences in Go and together with the team, we were finally able to produce what felt like real output. I enjoyed coding and it felt easier to do nearly 130 hours of programming compared to the earlier 70 hours of project management and evaluation.

Generally, the work environment was always great, be it with our advisor or between the team members. Everyone was flexible in scheduling meetings or providing feedback or help.

At the end, we can be proud of the accomplished in this project.

5.4. Conclusion

The final product successfully demonstrates an automated approach to deploying a Kubernetes cluster, without requiring deeper Kubernetes knowledge. After roughly 135 hours of analysis and 300 hours of implementation, the defined **MVP** has been successfully developed.

The project has enabled the team to strengthen their competences in working with Kubernetes and configuring components. It also provided a great opportunity to gather experiences in unknown territory, such as writing a software project in Go.

Bibliography

[1] Cloud Native Foundation, "CNCF Landscape guide." Accessed: Oct. 08, 2025. [Online]. Available: https://landscape.cncf.io/

[2] Helm Authors, "Helm Charts." Accessed: Dec. 17, 2025. [Online]. Available: https://helm.sh/docs/topics/charts/#chart-repositories

[3] Sidero Labs, "Talos Linux." Accessed: Oct. 09, 2025. [Online]. Available: https://www.talos.dev/

[4] Sidero Labs, "Talos Linux." Accessed: Oct. 09, 2025. [Online]. Available: https://www.talos.dev/v1.11/

[5] Sidero Labs, "Talos git repository." Accessed: Oct. 09, 2025. [Online]. Available: https://github.com/siderolabs/talos

[6] Sidero Labs, "Sidero Support & Services." Accessed: Oct. 30, 2025. [Online]. Available: https://www.siderolabs.com/omni/

[7] Flatcar, "Flatcar documentation." Accessed: Oct. 09, 2025. [Online]. Available: https://www.flatcar.org/docs/latest

[8] Flatcar, "Flatcar git repository." Accessed: Oct. 09, 2025. [Online]. Available: https://github.com/flatcar/Flatcar

[9] Flatcar, "Flatcar." Accessed: Oct. 09, 2025. [Online]. Available: https://www.flatcar.org/docs

[10] Charles Mahler, "Kubernetes on the edge getting started with KubeEdge and Kubernetes for edge computing." Accessed: Oct. 16, 2025. [Online]. Available: https://www.cncf.io/blog/2022/08/18/kubernetes-on-the-edge-getting-started-with-kubeedge-and-kubernetes-for-edge-computing/

[11] Kube Edge Project Authors, "KubeEdge Documentation." Accessed: Oct. 31, 2025. [Online]. Available: https://kubeedge.io/docs/

[12] Sidero Labs, "Omni documentation." Accessed: Oct. 16, 2025. [Online]. Available: https://www.siderolabs.com/omni/

[13] Metal³, "BareMetal Operator." Accessed: Oct. 16, 2025. [Online]. Available: https://github.com/metal3-io/baremetal-operator?tab=readme-ov-file

[14] CNCF, *Metal³ Kubernetes-Native Bare Metal Host Management*. Accessed: Oct. 16, 2025. [Online Video]. Available: https://www.youtube.com/watch?v=-BvQWz2cfCg

[15] Open Policy Agent contributors, "Open Policy Agent." Accessed: Nov. 01, 2025. [Online]. Available: https://www.openpolicyagent.org/

[16] Kyverno, "Kyverno Introduction." Accessed: Oct. 20, 2025. [Online]. Available: https://kyverno.io/docs/introduction/

[17] TUF Authors, "The update framework." Accessed: Nov. 01, 2025. [Online]. Available: https://theupdateframework.io/docs/overview/

[18] Slimtoolkit contributors, "Slimtoolkit." Accessed: Nov. 01, 2025. [Online]. Available: https://github.com/slimtoolkit/slim

[19] cerbos, "cerbos." Accessed: Nov. 01, 2025. [Online]. Available: https://www.cerbos.dev/product-cerbos-hub

[20] Keycloak, "Keycloak guides." Accessed: Oct. 19, 2025. [Online]. Available: https://www.keycloak.org/guides

[21] Keycloak, "Keycloak git repository." Accessed: Oct. 19, 2025. [Online]. Available: https://github.com/keycloak/keycloak

[22] Zitadel, "ZitadelGit." Accessed: Oct. 19, 2025. [Online]. Available: https://github.com/zitadel/zitadel

[23] Zitadel, "Zitadel Website." Accessed: Oct. 19, 2025. [Online]. Available: https://zitadel.com/pricing

[24] cert-manager, "Cloud native certificate management." Accessed: Oct. 19, 2025. [Online]. Available: https://cert-manager.io/

[25] cert-manager, "cert-manager." Accessed: Oct. 19, 2025. [Online]. Available: https://github.com/cert-manager/cert-manager

[26] Grype, "Grype." Accessed: Oct. 20, 2025. [Online]. Available: https://github.com/anchore/grype

[27] Kubescape, "Kubescape." Accessed: Oct. 20, 2025. [Online]. Available: https://kubescape.io/

[28] Kubescape, "Kubescape git repository." Accessed: Oct. 20, 2025. [Online]. Available: https://github.com/kubescape/kubescape

[29] Falco, "Falco." Accessed: Oct. 20, 2025. [Online]. Available: https://falco.org/docs/

[30] Falco, "Falco GitHub." Accessed: Oct. 20, 2025. [Online]. Available: https://github.com/falcosecurity/falco

[31] KubeLinter, "KubeLinter." Accessed: Oct. 20, 2025. [Online]. Available: https://docs.kubelinter.io/#/

[32] Grype, "Grype." Accessed: Oct. 20, 2025. [Online]. Available: https://www.checkov.io/1.Welcome/What%20is%20Checkov.html

[33] RedHat, "What is Clair?." Accessed: Oct. 20, 2025. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-clair#overview

[34] RedHat, "Clair." Accessed: Oct. 20, 2025. [Online]. Available: https://github.com/quay/clair

[35] R. S. Jann Fischer, "A Guide to Secrets Management with GitOps and Kubernetes." Accessed: Dec. 11, 2025. [Online]. Available: https://www.redhat.com/en/blog/a-guide-to-secrets-management-with-gitops-and-kubernetes

[36] Sealed Secrets contributors, "Sealed Secrets for Kubernetes." Accessed: Nov. 04, 2025. [Online]. Available: https://github.com/bitnami-labs/sealed-secrets

[37] SOPS Maintainers, "SOPS Secrets OPerationS." Accessed: Oct. 20, 2025. [Online]. Available: https://github.com/getsops/sops

[38] The Linux Foundation, "cubefs.io." Accessed: Oct. 19, 2025. [Online]. Available: https://cubefs.io/community

[39] Matt Saunders, "CubeFS Graduates from CNCF." Accessed: Oct. 30, 2025. [Online]. Available: https://www.infoq.com/news/2025/03/cubefs-cncf-graduation/

[40] Pradeep Singh, "CubeFS Graduates from CNCF: A Milestone in Cloud-Native Distributed Storage." Accessed: Oct. 30, 2025. [Online]. Available: https://mojoauth.com/blog/news-2025-03-cubefs-cncf-graduation#key-features-and-capabilities/

[41] CubeFS Authors, "CubeFS Docs - Intelligent Tiering." Accessed: Oct. 30, 2025. [Online]. Available: https://cubefs.io/docs/master/feature/hybridcloud.html

[42] CubeFS Authors, "CubeFS Docs - Authentication." Accessed: Oct. 30, 2025. [Online]. Available: https://cubefs.io/docs/master/design/authnode.html#feature-rich

[43] CubeFS Authors, "CubeFS Docs - Monitoring Metrics CubeFS_Docs_MetricsCollection." Accessed: Oct. 30, 2025. [Online]. Available: https://cubefs.io/docs/release-3.2.1/maintenance/metrics/collect.html

[44] CubeFS Authors, "CubeFS Docs - Kubernetes Deployment." Accessed: Oct. 30, 2025. [Online]. Available: https://cubefs.io/docs/master/deploy/k8s.html#edit-the-configuration

[45] Rook Authors 2022, "Rook Documentation." Accessed: Oct. 19, 2025. [Online]. Available: https://rook.io/docs/rook/latest-release/Getting-Started/intro/

[46] Rook Authors, "rook.io." Accessed: Oct. 30, 2025. [Online]. Available: https://rook.io/

[47] Jonas Sterr, "Ceph Performance Guide - Sizing & Testing." Accessed: Oct. 30, 2025. [Online]. Available: https://www.thomas-krenn.com/en/wiki/Ceph_Perfomance_Guide_-_Sizing_%26_Testing

[48] Ceph Foundation, "ceph.io." Accessed: Oct. 30, 2025. [Online]. Available: https://ceph.io/en/discover/benefits/

[49] Rook Authors, "Rook Docs - Snapshots." Accessed: Oct. 30, 2025. [Online]. Available: https://rook.io/docs/rook/v1.10/Storage-Configuration/Ceph-CSI/ceph-csi-snapshot/

[50] Federico Lucifredi, "Data Securit and Storage Hardening in Rook and Ceph." Accessed: Oct. 30, 2025. [Online]. Available: https://ceph.io/assets/pdfs/events/2024/ceph-days-nyc/DATA%20SECURITY%20AND%20STORAGE%20HARDENING%20IN%20ROOK%20AND%20CEPH%20(CEPH%20DAYS%20NYC%202024).pdf

[51] Rook Authors, "Rook Docs - Prometheus Monitoring." Accessed: Oct. 30, 2025. [Online]. Available: https://rook.io/docs/rook/latest/Storage-Configuration/Monitoring/ceph-monitoring/

[52] Longhorn Authors, "What is Longhorn?." Accessed: Oct. 19, 2025. [Online]. Available: https://longhorn.io/docs/1.10.0/what-is-longhorn/

[53] Longhorn Authors, "Longhorn Docs - Creating a Longhorn Backup." Accessed: Oct. 30, 2025. [Online]. Available: https://longhorn.io/docs/1.10.0/snapshots-and-backups/backup-and-restore/create-a-backup/

[54] Longhorn Authors, "Longhorn Docs - Volume Longhorn Encryption." Accessed: Oct. 30, 2025. [Online]. Available: https://longhorn.io/docs/1.10.0/advanced-resources/security/volume-encryption/

[55] Longhorn Authors, "Longhorn Docs - Setting up Prometheus and Grafana to monitor Longhorn." Accessed: Oct. 30, 2025. [Online]. Available: https://longhorn.io/docs/1.10.0/monitoring/prometheus-and-grafana-setup/#high-level-overview

[56] Longhorn Authors, "Longhorn Docs - Install with Helm." Accessed: Oct. 30, 2025. [Online]. Available: https://longhorn.io/docs/1.10.0/deploy/install/install-with-helm/

[57] containerd Authors, "containerd.io." Accessed: Oct. 12, 2025. [Online]. Available: https://containerd.io/

[58] Sidero Labs, "Talos Docs - Talos Components." Accessed: Oct. 30, 2025. [Online]. Available: https://docs.siderolabs.com/talos/v1.10/learn-more/components

[59] CRI-O Project Authors, "cri-o.io." Accessed: Oct. 30, 2025. [Online]. Available: https://cri-o.io/

[60] Cilium Authors, "Cilium." Accessed: Oct. 30, 2025. [Online]. Available: https://cilium.io/

[61] Cilium Authors, "Clilum Docs - Overview of Network Policy." Accessed: Oct. 30, 2025. [Online]. Available: https://docs.cilium.io/en/latest/security/policy/index.html

[62] ArtifactHUB Authors, "ArtifactHUB - Cilium." Accessed: Oct. 30, 2025. [Online]. Available: https://artifacthub.io/packages/helm/cilium/cilium

[63] Cilium Authors, "Cilium - Security." Accessed: Oct. 30, 2025. [Online]. Available: https://cilium.io/industries/security/

[64] Tigera Inc., "About Calico." Accessed: Oct. 20, 2025. [Online]. Available: https://docs.tigera.io/calico/latest/about/

[65] Tigera Inc., "Calico Docs - Get started with Calico network policy." Accessed: Oct. 30, 2025. [Online]. Available: https://docs.tigera.io/calico/latest/network-policy/get-started/calico-policy/calico-network-policy

[66] Tigera Inc., "Calico Docs - Install using Helm." Accessed: Oct. 30, 2025. [Online]. Available: https://docs.tigera.io/calico/latest/getting-started/kubernetes/helm

[67] Flannel Authors, "Flannel Github packages." Accessed: Oct. 30, 2025. [Online]. Available: https://github.com/flannel-io/flannel

[68] HashiCorp Inc., "Nomad Pricing." Accessed: Oct. 22, 2025. [Online]. Available: https://www.hashicorp.com/en/pricing?tab=nomad

[69] The Apache Software Foundation, "Apache Attic - Apache Mesos." Accessed: Oct. 22, 2025. [Online]. Available: https://projects.apache.org/project.html?attic-mesos

[70] CoreOS, "CoreOS fleet git repository." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/coreos/fleet

[71] The Kubernetes Authors, "Kubernetes Community." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/community/

[72] The Kubernetes Authors, "Considerations for large clusters." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/setup/best-practices/cluster-large/

[73] The Kubernetes Authors, "Kubernetes Documentation." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/home/

[74] HashiCorp Inc., "Nomad Git Repository." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/hashicorp/nomad

[75] HashiCorp Inc., "Nomad Use Cases." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs/use-cases

[76] HashiCorp Inc., "The Million Container Challenge." Accessed: Oct. 22, 2025. [Online]. Available: https://www.hashicorp.com/de/c1m

[77] HashiCorp Inc., "Introduction to Nomad." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs/what-is-nomad

[78] The Kubernetes Authors, "Kubernetes." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/

[79] The Kubernetes Authors, "Kubernetes Partners." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/partners/

[80] The Kubernetes Authors, "Kubernetes Git Repository." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/kubernetes/kubernetes

[81] The Kubernetes Authors, "Kubernetes Security." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/security/

[82] The Kubernetes Authors, "Kubernetes Overview." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/overview/

[83] The Kubernetes Authors, "Command line tool (kubectl)." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/reference/kubectl/overview/

[84] Philipp Strube (Kubestack), "Working with Terraform and Kubernetes." Accessed: Oct. 22, 2025. [Online]. Available: https://v1-32.docs.kubernetes.io/blog/2020/06/working-with-terraform-and-kubernetes/

[85] The Kubernetes Authors, "Extending Kubernetes." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/

[86] The Kubernetes Authors, "Kubernetes Other Tools." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/reference/tools/

[87] The Kubernetes Authors, "Kubernetes Multi-tenancy." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/concepts/security/multi-tenancy/

[88] HashiCorp Inc., "Nomad." Accessed: Oct. 22, 2025. [Online]. Available: https://www.hashicorp.com/products/nomad

[89] HashiCorp Inc., "Install Nomad." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs/deploy

[90] HashiCorp Inc., "Nomad Architecture." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs/architecture

[91] HashiCorp Inc., "Nomad Documentation." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs

[92] HashiCorp Inc., "Nomad HTTP API." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/api-docs

[93] HashiCorp Inc., "Nomad command-line interface (CLI) reference." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs/commands

[94] HashiCorp Inc., "Nomad Integrations." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/integrations

[95] HashiCorp Inc., "Nomad Ecosystem." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs/ecosystem

[96] HashiCorp Inc., "Nomad secure multi-tenancy with namespaces." Accessed: Oct. 22, 2025. [Online]. Available: https://developer.hashicorp.com/nomad/docs/ecosystem

[97] The Kubernetes Authors, "Kubernetes Configure DNS for a Cluster." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/tasks/access-application-cluster/configure-dns-cluster/

[98] The Apache Software Foundation, "Apache Zookeeper." Accessed: Oct. 22, 2025. [Online]. Available: https://zookeeper.apache.org/

[99] etcd Authors, "etcd." Accessed: Oct. 22, 2025. [Online]. Available: https://etcd.io/

[100] The CoreDNS Authors, "CoreDNS: DNS and Service Discovery." Accessed: Oct. 22, 2025. [Online]. Available: https://coredns.io/

[101] The Kubernetes Authors, "Creating a cluster with kubeadm." Accessed: Oct. 22, 2025. [Online]. Available: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

[102] Cloud Native Foundation, "CNCF Projects." Accessed: Oct. 22, 2025. [Online]. Available: https://www.cncf.io/projects/

[103] Grafana Labs, "Grafana Git Repository." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/grafana/grafana

[104] Grafana Labs, "Grafana Success stories and case studies." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/success/

[105] Grafana Labs, "Grafana Plan your IAM integration strategy." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/docs/grafana/latest/setup-grafana/configure-security/planning-iam-strategy/

[106] Grafana Labs, "Deploy Grafana on Kubernetes." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/docs/grafana/latest/setup-grafana/installation/kubernetes/

[107] Grafana Labs, "Grafana Help and support." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/help/

[108] Grafana Labs, "Grafana Dashboards." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/grafana/dashboards/

[109] Grafana Labs, "Grafana Alerting." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/docs/grafana/latest/alerting/

[110] The Linux Foundation, "About OpenSearch." Accessed: Oct. 22, 2025. [Online]. Available: https://opensearch.org/about/

[111] The Linux Foundation, "OpenSearch Getting Started Guide." Accessed: Oct. 22, 2025. [Online]. Available: https://docs.opensearch.org/latest/getting-started/

[112] The Linux Foundation, "OpenSearch Solutions Providers." Accessed: Oct. 22, 2025. [Online]. Available: https://opensearch.org/solutions-providers/

[113] The Linux Foundation, "OpenSearch Dashboards Git Repository." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/opensearch-project/OpenSearch-Dashboards

[114] The Linux Foundation, "OpenSearch Additional plugins." Accessed: Oct. 22, 2025. [Online]. Available: https://docs.opensearch.org/latest/install-and-configure/additional-plugins/index/

[115] The Linux Foundation, "About Security in OpenSearch." Accessed: Oct. 22, 2025. [Online]. Available: https://docs.opensearch.org/latest/security/

[116] The Linux Foundation, "OpenSearch Kubernetes Operator." Accessed: Oct. 22, 2025. [Online]. Available: https://docs.opensearch.org/latest/install-and-configure/install-opensearch/operator/

[117] The Linux Foundation, "Join the OpenSearch Community." Accessed: Oct. 22, 2025. [Online]. Available: https://opensearch.org/community/

[118] The Linux Foundation, "Integrations in OpenSearch Dashboards." Accessed: Oct. 22, 2025. [Online]. Available: https://docs.opensearch.org/latest/dashboards/integrations/index/

[119] Grafana Labs, "Grafana Loki Git Repository." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/grafana/loki

[120] Grafana Labs, "Send log data to Loki." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/docs/loki/latest/send-data/

[121] Grafana Labs, "Loki Manage authentication." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/docs/loki/latest/operations/authentication/

[122] Grafana Labs, "Loki Multi-tenancy." Accessed: Oct. 22, 2025. [Online]. Available: https://grafana.com/docs/loki/latest/operations/multi-tenancy/

[123] Grafana Labs, "Kubernetes Monitoring Helm tutorial." Accessed: Oct. 23, 2025. [Online]. Available: https://grafana.com/docs/loki/latest/send-data/k8s-monitoring-helm/

[124] Grafana Labs, "Install Grafana Loki with Helm." Accessed: Oct. 23, 2025. [Online]. Available: https://grafana.com/docs/loki/latest/setup/install/helm/

[125] Grafana Labs, "Loki Manage storage." Accessed: Oct. 23, 2025. [Online]. Available: https://grafana.com/docs/loki/latest/operations/storage/

[126] kube-logging authors, "Logging Operator Git Repository." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/kube-logging/logging-operator

[127] kube-logging authors, "Logging Operator Adopters." Accessed: Oct. 22, 2025. [Online]. Available: https://github.com/kube-logging/logging-operator/blob/master/ADOPTERS.md

[128] The Fluent Bit Authors, "Fluent Bit Inputs." Accessed: Oct. 22, 2025. [Online]. Available: https://docs.fluentbit.io/manual/4.0/data-pipeline/inputs

[129] Fluentd Authors, "Fluentd Plugins." Accessed: Oct. 22, 2025. [Online]. Available: https://www.fluentd.org/plugins/all

[130] kube-logging authors, "Logging Operator Security." Accessed: Oct. 22, 2025. [Online]. Available: https://kube-logging.dev/docs/logging-infrastructure/security/

[131] kube-logging authors, "Logging Operator Nodegroup-based multitenancy." Accessed: Oct. 22, 2025. [Online]. Available: https://kube-logging.dev/docs/examples/multitenancy/

[132] Prometheus Authors, "Prometheus Git Repository." Accessed: Oct. 23, 2025. [Online]. Available: https://github.com/prometheus/prometheus

[133] Prometheus Authors, "Prometheus Integrations." Accessed: Oct. 23, 2025. [Online]. Available: https://prometheus.io/docs/operating/integrations/

[134] Prometheus Authors, "Prometheus Security model." Accessed: Oct. 23, 2025. [Online]. Available: https://prometheus.io/docs/operating/security/

[135] Thanos authors, "Thanos." Accessed: Oct. 23, 2025. [Online]. Available: https://thanos.io/

[136] Prometheus Community, "kube-prometheus-stack Helm Charts Git Repository." Accessed: Oct. 26, 2025. [Online]. Available: https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack

[137] Prometheus Authors, "Prometheus Support and Training." Accessed: Oct. 26, 2025. [Online]. Available: https://prometheus.io/support-training/

[138] Thanos authors, "Thanos Git Repository." Accessed: Oct. 23, 2025. [Online]. Available: https://github.com/thanos-io/thanos

[139] Prometheus Community, "Prometheus Operator Git Repository." Accessed: Oct. 26, 2025. [Online]. Available: https://github.com/prometheus-operator/prometheus-operator

[140] Prometheus Community, "Prometheus Operator Adopters." Accessed: Oct. 26, 2025. [Online]. Available: https://prometheus-operator.dev/adopters/

[141] Prometheus Community, "Prometheus Operator RBAC." Accessed: Oct. 26, 2025. [Online]. Available: https://prometheus-operator.dev/docs/platform/rbac/

[142] Prometheus Community, "Prometheus Operator Git Issues." Accessed: Oct. 26, 2025. [Online]. Available: https://github.com/prometheus-operator/prometheus-operator/issues

[143] Prometheus Community, "Installing Prometheus Operator." Accessed: Oct. 26, 2025. [Online]. Available: https://prometheus-operator.dev/docs/getting-started/installation/

[144] Grafana Labs, "Grafana Tempo Git Repository." Accessed: Oct. 26, 2025. [Online]. Available: https://github.com/grafana/tempo

[145] Grafana Labs, "Tempo Manage authentication." Accessed: Oct. 26, 2025. [Online]. Available: https://grafana.com/docs/tempo/latest/operations/authentication/

[146] Grafana Labs, "Tempo Enable multi-tenancy." Accessed: Oct. 26, 2025. [Online]. Available: https://grafana.com/docs/tempo/latest/operations/manage-advanced-systems/multitenancy/

[147] Grafana Labs, "Deploy Tempo with Helm." Accessed: Oct. 26, 2025. [Online]. Available: https://grafana.com/docs/tempo/latest/set-up-for-tracing/setup-tempo/deploy/kubernetes/helm-chart/

[148] OpenTelemetry Authors, "OpenTelemetry." Accessed: Oct. 26, 2025. [Online]. Available: https://opentelemetry.io/

[149] OpenTelemetry Authors, "OpenTelemetry Adopters." Accessed: Oct. 26, 2025. [Online]. Available: https://opentelemetry.io/ecosystem/adopters/

[150] OpenTelemetry Authors, "OpenTelemetry Git Repository." Accessed: Oct. 26, 2025. [Online]. Available: https://github.com/open-telemetry/opentelemetry.io

[151] OpenTelemetry Authors, "OpenTelemetry Integrations." Accessed: Oct. 26, 2025. [Online]. Available: https://opentelemetry.io/ecosystem/integrations/

[152] OpenTelemetry Authors, "OpenTelemetry Helm Charts." Accessed: Oct. 26, 2025. [Online]. Available: https://opentelemetry.io/docs/platforms/kubernetes/helm/

[153] OpenTelemetry Authors, "OpenTelemetry Collector configuration best practices." Accessed: Oct. 26, 2025. [Online]. Available: https://opentelemetry.io/docs/security/config-best-practices/

[154] OpenTelemetry Authors, "OpenTelemetry Community." Accessed: Oct. 26, 2025. [Online]. Available: https://opentelemetry.io/community/

List of External Tools and Resources

Competence	**External Tools / Resources**
Component Research	Google, CNCF, ChatGPT, Copilot
Visualization	Excel, draw.io, Microsoft Whiteboard, PowerPoint
Idea Generation	Microsoft Whiteboard, ChatGPT, Copilot
Translation	Google Translate, DeepL, ChatGPT
Coding & Developing	Visual Studio Code, Go docs, Proxmox, ChatGPT, Copilot
Text Creation & Verification	ChatGPT, Copilot
Collaboration	Microsoft 365, GitLab, YouTrack
DevOps	GitLab, Kubernetes

Table 12: List of Tools and Resources

Glossary

arc42 A template for documenting software and system architectures.

Bare-metal Refers to running software directly on physical hardware without an intermediate virtualization layer.

BubbleTea Go framework to build rich, and interactive terminal user interfaces

cert-manager A native Kubernetes certificate management controller that automates the issuance and renewal of TLS certificates.

Cilium A networking, observability, and security solution for Kubernetes based on eBPF.

containerd An industry-standard container runtime with an emphasis on simplicity, robustness, and portability.

Control plane nodes Machines in a Kubernetes cluster that manage the cluster state, scheduling, and coordination of worker nodes.

CRD (Custom Resource Definition) Kubernetes extension mechanism that allows users to create custom resource types.

Docker Platform to build and run containers

Falco A cloud-native runtime security tool that detects anomalous activity in applications and containers.

Go A statically typed, compiled programming language designed at Google, known for its concurrency support and efficiency.

Grafana An open-source platform for monitoring and observability that allows querying, visualizing, alerting on, and exploring metrics, logs, and traces.

Helm Package manager for Kubernetes that packages resources as charts and manages releases.

Kubeconfig A configuration file used by Kubernetes clients to access clusters, containing cluster, user, and context information.

Kubernetes Open-source system for automating deployment, scaling, and management of containerized applications.

Kustomization Kubernetes native configuration management tool

Lip Gloss Go library to style terminal applications such as BubbleTea

Loki A horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus.

Longhorn Cloud-native distributed block storage system for Kubernetes clusters.

Manifest A file (typically YAML) that declares the desired state of resources to be applied to a system (for example, Kubernetes manifests).

OpenTelemetry An observability framework and toolkit designed to create and manage telemetry data such as traces, metrics, and logs.

Prometheus An open-source systems monitoring and alerting toolkit.

sealed-secrets A Kubernetes controller and tool for one-way encrypted Secrets, allowing them to be stored safely in public repositories.

Talos A modern, API-driven, immutable, and minimal Linux distribution designed specifically for Kubernetes.

Terminal user interface A user interface that uses text and symbols in a terminal to allow user interaction.

Worker nodes Machines in a Kubernetes cluster that run containerized applications and are managed by the control plane.

YAML Human-readable data-serialization format commonly used for configuration and Kubernetes manifests.

List of Abbreviations

ACL	Access Control List	**MQTT**	Message Queueing Telemetry Transport
ADR	Architectural Decision Record	**MVP**	Minimal Viable Product
AGPL	Affero General Public License	**NFR**	non-functional requirement
AI	Aritificial Intelligence	**OCI**	Open Container Initiative
API	Application Programming Interface	**OS**	Operating System
BSL	Business Source License	**PXE**	Preboot eXecution Environment
CAS	Content-Addressed Storage	**PaaS**	Platform as a Service
CC-BY-4.0	Creative Commons Attribution 4.0 International License	**PoC**	Proof of Concept
CLI	Command-Line Interface	**RAID**	Redundant Array of Independent Disks
CNCF	Cloud Native Computing Foundation	**RBAC**	Role-Based Access Control
CNI	Container Network Interface	**RED**	Rate Errors Duration
CRD	Custom Resource Definition	**SA**	Semester Thesis
CRI	Container Runtime Interface	**SAAS**	Software-as-a-Service
CSI	Container Storage Interface	**SIEM**	Security Information and Event Management
DNS	Domain Name System	**SLA**	Service Level Agreement
ECP	Enterprise Container Platform	**SOPS**	Secrets OPerationS
ECTS	European Credit Transfer System	**SSO**	single sign-on
HTTP	Hypertext Transfer Protocol	**TUI**	Terminal User Interface
IAM	Identity and Access Management	**UI**	User Interface
IP	Internet Protocol	**VM**	Virtual Machine
IaC	Infrastructure as Code	**YAML**	Yet Another Markup Language
IdP	Identity Provider	**eBPF**	Extended Berkeley Packet Filter
JSON	JavaScript Object Notation	**mTLS**	mutual Transport Layer Security

List of Tables

List of figures

Code Listings

Appendix

A. Evaluations

A.1. Evaluation of Operating System

The OS runs on the hardware and makes a computer useful.

A.1.1. Selection criteria

There are a lot of OSs, but most of them are not optimized for our ECP use case. Only OSs that fulfill the selection criteria below have been analyzed.

- The OS has to be optimized for container runtimes.
- It has to be as minimal, immutable and security hardened.

Initial research has shown two OSs that fulfill these requirements. They are Talos Linux, Flatcar Container Linux.

A.1.2. Evaluation criteria

- Maintenance: The code is updated regularly (as in not stale).
- Security: The OS has processes in place to ensure security requirements are met.
- Support: The technology is backed by an enterprise that offers support.
- Licensing: Code is for commercial use (e.g. Apache-2.0 License).
- Enterprise readiness: Is the technology mature and runs in the enterprise setting of Software DW.
- Ease of use: Interaction with the OS is intuitive and documentation is up to date.
- Installation options: The OS is installable with .iso file and has support for Preboot eXecution Environment (PXE) boot.
- Virtualization support: Support to run VMs - optional feature.

A.1.3. Talos Linux

Talos Linux is an operating system designed to run Kubernetes. It is an open-source project managed by Sidero Labs.

Features:

- "Talos reduces your attack surface. It's minimal, hardened and immutable". [3]
- No SSH access, configuration done via an API.
- "Delivers the latest stable versions of Kubernetes and Linux". [3]
- `talosctl cluster create` installs a local kubernetes cluster for development. [4]

Challenges:

- Automate the configuration to use the `talosctl`. [4]
- Dependency that Talos bootstraps an etcd cluster. [4]

Criteria:

- Maintenance: Code is actively maintained by Sidero labs. [5]
- Security: "Talos reduces your attack surface. It's minimal, hardened and immutable. All API access is secured with mutual TLS (mTLS) authentication". [3]
- Support: For mission critical environments, Sidero labs offers support contracts with Service Level Agreements (SLAs). [6]

- Licensing: Talos is licensed under the General Public License. [5]
- Enterprise readiness: This is guaranteed, Talos runs some of the biggest Kubernetes clusters in the world and can easily handle the 5 designated servers of Software DW. [4]
- Ease of use: Talos provides a documentation which is easy to understand, and reviews also suggest that the adaptation time is low.
- Installation options: Talos supports installation via ISO images and PXE boot support using matchbox. [4]
- Virtualization support: Support for KubeVirt. [4]

A.1.4. Flatcar Container Linux

"Flatcar Container Linux is a container optimized OS that ships a minimal OS image, which includes only the tools needed to run containers". [7] It is maintained by the Flatcar project contributors and Kinvolk and Microsoft.

Features:

- "Flatcar Container Linux runs on most cloud providers, virtualization platforms and bare metal servers". [7]

Challenges:

- Configuration via butane YAML files[2].

Criteria:

- Maintenance: Code is actively maintained by the Flatcar contributors. [8]
- Security: Flatcar is minimal and focused, while always ensuring automated updates to provide a secure operating system. [9]
- Support: There is no enterprise support agreement for Flatcar. Support might be provided through an active community.
- Licensing: Flatcar is licensed through Apache-2.0 License. [8]

[2]Example Butane config:

```
variant: flatcar
version: 1.0.0
systemd:
  units:
    - name: nginx.service
      enabled: true
      contents: |
        [Unit]
        Description=NGINX example
        After=docker.service
        Requires=docker.service
        [Service]
        TimeoutStartSec=0
        ExecStartPre=-/usr/bin/docker rm --force nginx1
        ExecStart=/usr/bin/docker run --name nginx1 --pull always --log-driver=journald --net host docker.io/nginx:1
        ExecStop=/usr/bin/docker stop nginx1
        Restart=always
        RestartSec=5s
        [Install]
        WantedBy=multi-user.target

```

Code Listing 3: Example butane config

- Enterprise readiness: Flatcar supports enterprise ready production clusters, a scale a lot bigger than the scope of Software DW. [7]
- Ease of use: Configuration is done with butane yaml files It is declarative and will work for our project. More complicated to get into than Talos, but looks understandable once one gets the hang of it.
- Installation options: Bare Metal installation options are ISO images, PXE boot, Installation script on existing system. [7]
- Virtualization support: Nothing specifically found for this, therefore currently marked as not supported.

A.1.5. Decision matrix

Criterion	Talos Linux	Flatcar Container Linux
Maintenance	✓ Fulfilled	✓ Fulfilled
Security	✓ Fulfilled	✓ Fulfilled
Support	✓ Fulfilled	✓ Partial fulfillment
Licensing	✓ Fulfilled	✓ Fulfilled
Enterprise readiness	✓ Fulfilled	✓ Fulfilled
Ease of use	✓ Fulfilled	✓ Partial fulfillment
Installation options	✓ Fulfilled	✓ Fulfilled
Virtualization support	✓ Fulfilled	✗ Unfulfilled

Table 13: OS Decision Matrix

A.1.6. Conclusion

Based on the evaluation criteria, both **OS**s would work for the scenario of Software DW. They are both optimized to run container platforms, although their approaches differ, which will have implications on how our product is implemented.

Talos has no SSH, which poses some challenges to get the cluster set up. But the documentation looks great and understandable. Setting up a Talos cluster during the evaluation was easy and intuitive, which bodes well for further implementing it. Flatcar on the other hand works a bit different with the butane **YAML** file. But they are all declarative and it would be easy to bootstrap the **YAML** files from the settings selected by the user of our product. For an enterprise-grade platform, support can be crucial, but isn't always provided. Not having it doesn't immediately break the project, but it's certainly not ideal.

As said, both **OS**s would work, but the decision ultimately is to use **Talos Linux**.

A.2. Evaluation of Automation & Configuration

Automation & Configuration is about codifying the environment setup, so that it becomes reproducible. The desired state is specified in **YAML** files, and the Container Platform will match that desired state. This way of specifying infrastructure is less error-prone and handles rapid development cycles better.

CNCF states that "Fundamentally, you'll need one or more tools in this space as part of laying down the computing environment, CPU, memory, storage, and networking, for your Kubernetes clusters. You'll also need a subset of these to create and manage the Kubernetes clusters themselves". [1]

A.2.1. KubeEdge

Edge computing describes the process of processing data where it is generated and only then sending it to the cloud. As an example, a surveillance camera uses artificial intelligence to analyze video and then only sends snippets of when something is happening to the cloud for further analysis. [10]

Description: Kubernetes by default doesn't support edge computing. KubeEdge is an extension to Kubernetes that makes the implementation for developers to use edge computing easier.

Features:

- Offers support for protocols like Hypertext Transfer Protocol (**HTTP**) and Message Queueing Telemetry Transport (**MQTT**). [10]
- Enables autonomous operation of edge nodes even when disconnected from the cloud. [11]

Use cases: KubeEdge will help developers to implement edge computing.

KubeEdge is a tool for the operator that wants to deploy containerized applications. Therefore it falls out of scope for this evaluation.

A.2.2. Terraform

TerraForm is one of the tools that can be used for codifying an environment, but it doesn't provide a way to bootstrap the servers as it's planned. Hence, it falls out of scope for this evaluation.

A.2.3. Omni

Omni is listed in the CNCF landscape and briefly looked at because Talos Linux is from the same vendor SideroLabs. [12]

Description: Omni provides a powerful graphical interface and a tight integration with Talos Linux. It simplifies the creation and management of Kubernetes.

Features:

- Dashboard to manage Kubernetes cluster.

Use cases:

- "On-premise bare metal clusters that can be scaled up with machines in the cloud". [12]
- "Edge clusters that are supported by machines in the data center". [12]

Our fictional company Software DW intends to use open-source solutions. In conflict with this, SideroLabs runs Omni as Software-as-a-Service (**SAAS**) or on-premise with a licensing mode. [12]

A.2.4. Metal³

Description: Metal³ provides components that allows bare-metal host management for Kubernetes.

Features:

- Inspect the host's hardware details and report them on the corresponding bare metal host. This includes information like disk or processor usage. [13]
- Provision hosts with a desired image. [13]
- Clean a host's disk contents before or after provisioning. [13]

The description of Metal³ implied it to be suitable for this use case, as its intended to host Kubernetes on bare-metal. After some resources about Metal³ have been considered [13], [14], it still isn't entirely clear what all features of Metal³ are used for. But as the deployer does not need Metal³ to install a container platform, it falls out of scope and thus isn't evaluated further.

A.2.5. Conclusion

The evaluation of the automation & configuration was not completed, because it is apparent that they are not applicable to the use cases defined for the project. These tools are installed and managed by the operator and not needed during the deployment process.

A.3. Evaluation of Security & Compliance

With the rapid release cycles of cloud native applications, tools in this category help to secure and harden the environment.

A.3.1. Pre-selection criteria

Cloud Native Computing Foundation (CNCF) lists a lot of tools in this category. As some of them serve different use cases, they were grouped. The tools were chosen based on these criteria:

- Only projects listed in the CNCF landscape were evaluated.
- They had to be graduated and incubating projects in the security & compliance section.
- The projects had to be of high popularity, with >10k GitHub stars.

This resulted in the following groups and tools:

- **Identity providers**: Keycloak, Zitadel
- **Certificate manager**: cert-manager
- **Vulnerability scanners**: Grype, Kubescape, Falco
- **Linter and static code checker**: KubeLinter, Checkov, Clair (by RedHat)
- **Hors catégorie**: SOPS

There are tools that fit these pre-selection criteria, but were filtered out because they were not relevant for the given use case of automating container platform deployment:

- Open Policy Agent: streamlines policy management across the stack. [15]
- Kyverno: A cloud native policy engine. [16]

Both Open Policy Agent and Kyverno are policy management solutions and used by the operator.

- The updated framework: Makes updating software you publish secure. Not needed in our stack as this isn't part of deployment. [17]
- SlimToolkit: Makes containers slimmer, smaller and more secure. This is useful for the operator once the enterprise container platform is set up. [18]
- Cerbos: Cloud native authorization tool, similar to tools in identity providers. Filtered out because more popular solutions exist. [19]

A.3.2. Evaluation criteria

To keep the scope of this evaluation in an acceptable size, the number of evaluation criteria was kept relatively small:

- Maintenance: The code is updated regularly (as in not stale).
- Support: The technology is backed by an enterprise that offers support.
- CNCF status: The tools in this section are security tools, therefore it is important to use mature tools and productive-tested tools. This is also a requirement of Software DW.
  - Graduated: ✓ Fulfilled
  - Incubating: ✓ Partial fulfillment
  - If it isn't a CNCF project, it must be well accepted and used in the community to achieve ✓ Fulfilled.
- Licensing: Code is for commercial use (e.g. Apache-2.0 License).

- Use in this project: The tool is either a required part of deploying the container platform or helps fulfill one of the use cases.

A.3.3. Identity providers

A.3.3.1. Keycloak

Keycloak is an open source identity provider that is an incubating project of **CNCF**.

Features:
- Runs on Kubernetes and OpenShift.
- Used to configure single sign-on (**SSO**) with protocols like OAuth 2.0 or SAML. [20]

Criteria:
- Maintenance: Keycloak is actively maintained. [21]
- Support: Other than an active community, there is no option for enterprise support.
- CNCF status: Incubating project. [1]
- Licensing: Licensed under Apache-2.0. [21]
- Use in our project: While not every container platform needs an Identity Provider (**IdP**), as soon as a login functionality is built the user management is needed. The **IdP** can be added as an option during cluster creation.

A.3.3.2. Zitadel

Zitadel is similar to Keycloak, they even say so themselves. It is an **IdP** that can do multi-tenant user management. [22] Additionally, they were founded in Switzerland.

Features:
- "Multi-tenancy with team management". [22]
- Protocols like OAuth 2.0, SAML and LDAP provide **SSO** support.

Challenges:
- Self-hosted option exists, but doesn't seem to be as straight forward.

Criteria:
- Maintenance: Code is actively maintained by Zitadel. [22]
- Support: Zitadel does offer enterprise grade support options. [23]
- CNCF status: Zitadel isn't a **CNCF** project, but is used frequently and is well maintainted, therefore partial fulfillment
- Licensing: under AGPL-3.0 licensing. [22]
- Use in this project: The same argument as for Keycloak applies.

A.3.3.3. Decision matrix

Criterion	Keycloak	Zitadel
Maintenance	✓ Fulfilled	✓ Fulfilled
Support	✓ Partial fulfillment	✓ Fulfilled
CNCF status	✓ Partial fulfillment	✓ Partial fulfillment
Licensing	✓ Fulfilled	✓ Fulfilled
Use in this project	✓ Fulfilled	✓ Fulfilled

Table 14: IdP Decision Matrix

A.3.3.4. Conclusion

Based on this evaluation, Zitadel is the preferred IdP due to the option of enterprise support. The IdP isn't the deployers concern, but it can quickly become use case for the software providers. Therefore, Zitadel will be an optional tool to select for the deployer, if the time budget allows for it.

Keycloak is a viable solution an can fulfill the same use case as Zitadel. This is kept on the backlog, it acts as a backup if problems arise with Zitadel or it could be added a second option during deployment.

A.3.4. Certificate manager

Due to the requirement that Software DW only wants to use open-source tools backed by a known vendor or a big community, the only option available in this category is cert-manager.

A.3.4.1. cert-manager

cert-manager is an X.509 certificate manager. It is a graduated CNCF project that was created by JetStack and donated to CNCF in 2020. [24]

Features:

- Obtains X.509 certificates from a variety of issuers and renews them to ensure they are valid. [24]
- Runs in Kubernetes and Openstack. [24]

Criteria:

- Maintenance: The code is actively maintained by CyberArk. [24]
- Support: Option for support by CyberArk. [24]
- CNCF status: Graduated CNCF project. [1]
- Licensing: Licensed under Apache 2.0. [25]
- Use in our project: cert-manager is the goto tool, when it comes to X.509 certificate management.

Criterion	cert-manager
Maintenance	✓ Fulfilled
Support	✓ Fulfilled
CNCF status	✓ Fulfilled
Licensing	✓ Fulfilled
Use in our project	✓ Fulfilled

Table 15: Certification Decision Matrix

A.3.4.2. Conclusion

Installing certificate management is not strictly part of deployment, but similarly to the **IdP** feature, it is a requirement in a lot of scenarios. Thus, cert-manager gets added as an optional feature.

A.3.5. Vulnerability scanners

It is a use case defined in the scope of this project to install a vulnerability scanner.

A.3.5.1. Grype

Grype is a "vulnerability scanner for container images and filesystems". [1] It is not part of **CNCF**, but popular with almost 11k stars on GitHub.

Features:

- Checks container images for known vulnerabilities.
- Support for multiple **OSs** like Alpine, Amazon Linux, Ubuntu, etc.
- Support for languages like Ruby, Java, Golang, etc.

Criteria:

- Maintenance: The code is actively maintained by Anchore. [26]
- Support: Commercial support options with Anchore. [26]
- **CNCF** status: Not a **CNCF** project, but popular by the community, therefore partial fulfillment.
- Licensing: Apache-2.0 licensed. [26]
- Use in this project: Vulnerability scanning of container images does improve the security of the cluster.

A.3.5.2. Kubescape

"Kubescape is an open-source Kubernetes security platform designed to provide practical, end-to-end security for Kubernetes environments". [27] It is an incubating project of **CNCF** with 11k stars on GitHub.

Features:

- "Configuration and vulnerability scanning". [27]
- Does not only provide static analysis, but also runtime monitoring.

Challenges:

- Runs as a micro-service within a Kubernetes cluster.

Criteria:

- Maintenance: The code is actively maintained by Kubescape authors. [28]
- Support: Support can be obtained through ARMO. [27]
- **CNCF** status: Incubating project. [1]
- Licensing: Apache-2.0 Licensed. [28]
- Use in this project: Adding Kubescape to our Kubernetes cluster will greatly improve the security of the platform, thus it adds a considerable benefit.

A.3.5.3. Falco

"Falco is a cloud native security tool that provides runtime security across hosts, containers, Kubernetes, and cloud environments. It is designed to detect and alert on abnormal behavior and potential security threats in real-time". [29]

Features:
- Monitoring and detecting agent.
- Mainly monitors the Linux Kernel
- Alerts can be sent to a Security Information and Event Management (**SIEM**).

Challenges:
- Other data sources like cloud events must be configured with plugins.

Criteria:
- Maintenance: The code is actively maintained by Sysdig. [29]
- Support: Enterprise support provided by Sysdig. [29]
- **CNCF** status: Graduated **CNCF** project. [1]
- Licensing: Apache-2.0 licensed. [30]
- Use in this project: Falco would improve the security of the container platform with it's runtime security threat analysis.

A.3.5.4. Decision matrix

Criteria	Grype	Kubescape	Falco
Maintenance	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Support	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
CNCF status	✓ Partial fulfillment	✓ Partial fulfillment	✓ Fulfilled
Licensing	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Use in this project	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled

Table 16: Vulnerability Scanner Decision Matrix

A.3.5.5. Conclusion

Grype, Kubescape und Falco all are similar, but with **Falco** being the only **CNCF** graduated project it is the winner out of the three. Grype and Kubescape would work too, and if there is enough time, they could be added as additional options.

A.3.6. Linter and static code checker

A.3.6.1. KubeLinter

"KubeLinter analyzes Kubernetes YAML files and Helm charts and checks them against various best practices". [31] It is currently in development by RedHat.

Features:
- Checks **YAML** and Helm files
- Used to check early and often for DevOps best practices and common misconfigurations.

Challenges:
- In an early state of development. [31]

Criteria:
- Maintenance: Still in an early stage of development. [31]
- Support: Nothing mentioned on the KubeLinter side, potentially through RedHat.
- **CNCF** status: Not a **CNCF** project and also not ready for production use. [31]

- Licensing: Apache-2.0 licensed. [31]
- Use in this project: Because KubeLinter is in early development, it is not ready for enterprise use yet and therefore doesn't meet the requirements of Software DW.

A.3.6.2. Checkov

"Checkov is a static code analysis tool for scanning Infrastructure as Code (**IaC**) files for misconfigurations that may lead to security or compliance problems". [32]

Features:

- Scans **IaC** files from Kubernetes, Terraform, Kubernetes etc.
- Checks and finds common misconfigurations.

Challenges:

- Runs on static code and is typically used at build time. Can be run in Kubernetes, but seems not commonly used that way. [32]

Criteria:

- Maintenance: Code is maintained by Prisma Cloud. [32]
- Support: No enterprise support agreement found.
- **CNCF** status: Not a **CNCF** project, less popular than Checkov and therefore Unfulfilled.
- Licensing: Apache-2.0 licensed. [32]
- Use in this project: Limited use for the **ECP** as it is more commonly used during build time in a pipeline.

A.3.6.3. Clair (by RedHat)

Clair is an open source tool that does static analysis of vulnerabilities in containers. It is **API**-driven and inspects containers layer-by-layer. [33]

Features:

- "Static analysis of vulnerabilities". [33]

Challenges:

- Requires the Kubernetes platform to be RedHat OpenShift.

Criteria:

- Maintenance: Code is actively maintained. [34]
- Support: Enterprise support is available through RedHat.
- **CNCF** status: Not a **CNCF** project, but popular choice from RedHat and therefore fulfilled.
- Licensing: Apache-2.0 licensed. [34]
- Use in our project: As this project will not use OpenShift, this tool is of no use.

A.3.6.4. Decision matrix

Criteria	KubeLinter	Checkov	Clair
Maintenance	✗ Unfulfilled	✓ Fulfilled	✓ Fulfilled
Support	✓ Partial fulfillment	✗ Unfulfilled	✓ Fulfilled
CNCF status	✗ Unfulfilled	✗ Unfulfilled	✓ Fulfilled
Licensing	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Use in our project	✗ Unfulfilled	✓ Partial fulfillment	✗ Unfulfilled

Table 17: Linter Decision Matrix

A.3.6.5. Conclusion

The evaluation of linters and static code checker showed, that none of the options passed the defined evaluation criteria. After reading into these static code analysis tools, it became apparent that they are more commonly used in pipelines and not run in a cluster. This is not part of the deployer role and also not a use case covered, which is why linter and static code checkers are ignored.

A.4. Evaluation of Secret Management

CNCF defines the category secrets management as: "The tools and projects in this category cover everything from how to securely store passwords and other secrets (sensitive data such as API keys, encryption keys, etc.) to how to safely eliminate passwords and secrets from your microservices environment". [1] They list tools from both key and secret management in the same category. There is an overlap, but they are not equal. In this project, they are defined as:

Key management: Key management focuses on the lifecycle of cryptographic keys, including their generation, distribution, rotation, and revocation. An example would be a certificate manager like cert-management.

Secret management: Secret management provides secure storage and access to secrets, ensuring that applications can retrieve them programmatically without exposing them in code, configuration files or in a repository. Examples would be sealed-secrets and Mozilla's SOPS.

This evaluation focuses on secret management as it addresses Software DW's primary use case: managing configuration secrets for containerized applications. As a mid-sized company launching their first container platform, implementing comprehensive key management infrastructure besides the cert-manager would add unnecessary complexity. [35]

A.4.1. Selection criteria

The selection criteria for this section are:

- Maintenance: Code is updated and maintained actively.
- Security considerations: Provides a secure way to store secrets stored in git.
- Licensing: Code is for commercial use (e.g. Apache-2.0 License).
- Ease of use: Provides a way for developers to create and manage their secrets.
- Enterprise readiness: Solution is mature and fits the use case of the scenario of Software DW.

A.4.2. sealed-secrets

sealed-secrets stores a custom resource SealedSecret in git. A controller running in Kubernetes has knowledge of the private and the public key which are used to decrypt the SealedSecret back to a Kubernetes secret. The developer can use the CLI `kubeseal` to encrypt secrets before comitting them to a repository. [35]

Features:

- The developer deploys the SealedSecret in git and the sealed-secret node in Kubernetes.
- "The Sealed Secrets controller supports automatic secret rotation for the private key". [35]
- A random nonce is used during encryption that makes brute-foce attacks very expensive. [35]

Challenges:

- The private keys are stored in the etcd store of the cluster and must be secured and backed up accordingly.
- It doesn't scale well when using multiple Kubernetes clusters, but that is not relevant for this use-case.

Criteria:

- Maintenance: The code is actively maintained by Bitnami. [36]

- Security considerations: Adding a SealedSecret object in a repository is a secure way to store a secret, as only the contoller and not even the creator himself can access it. [36]
- Licensing: Apache-2.0 licensed. [36]
- Ease of use: A developer can easily create a SealedSecret easily using `kubeseal`. [36]
- Enterprise readiness: sealed-secrets is in use in many clusters and fits the scenario of Software DW. Especially since they want to launch a new software product in their cluster.

A.4.2.1. SOPS: Secrets OPerationS

SOPS is an editor of encrypted files to edit **YAML** or JavaScript Object Notation (**JSON**) files. It was initially created by Mozilla and has been donated to **CNCF** as a sandbox project. It has almost 20k stars on Github and is very popular in the community. [37]

Features:

- Encrypts values of **YAML** or **JSON** file, that can be stored securely in a Git repository. [37]
- The keys and structure of the file remain, while the values are encrypted. [37]
- Special case encryption/decyption tool that is not limited to Kubernetes. [35]

Challenges:

- There is no service needed in the Kubernetes cluster, as there is nothing to deploy in the cluster. Instead, every developer must set up **SOPS** on their local devices.
- As **SOPS** is a **CLI** tool, the usability is limited to the Kubernetes-native world. [35]

Criteria:

- Maintenance: The code is actively maintained. [37]
- Security considerations: Storing secrets encrypted by **SOPS** is secure.
- Licensing: MPL-2.0 Licensing. [37]
- Ease of use: The ease of use of the CLI is given, using it in a Kubernetes cluster requires plugins and is more difficult than with sealed-secrets.
- Enterprise readiness: **SOPS** is a mature software and loved by the community. This category is only partial fulfillment because it requires more configuration to use it for their upcoming software product.

A.4.3. Decision matrix

Criterion	Sealed Secrets	SOPS
Maintenance	✓ Fulfilled	✓ Fulfilled
Security	✓ Fulfilled	✓ Fulfilled
Licensing	✓ Fulfilled	✓ Fulfilled
Ease of use	✓ Fulfilled	✓ Partial fulfillment
Enterprise readiness	✓ Fulfilled	✓ Partial fulfillment

A.4.4. Conclusion

The decision is to use **sealed-secrets** for secret management. It allows the deployment of the sealed-secret component in the cluster and the developer can be instructed to use `kubeseal`. This fits the Software DW scenario better than using **SOPS**.

A.5. Evaluation of Cloud Native Storage

Any changes made within containers are lost when they get deleted. To provide a shared storage between containers, e.g. config files for an application which runs on multiple ones, you need to provide a persistent storage. This is where the cloud native storage plugin helps out.

A.5.1. Selection criteria

To have the plugins work the intended way, the **CSI** was defined. Since this is a cloud native technology, it should be deployable and manageable by an Operator (e.g. Helm chart).

The full list of selection criteria is as follows:
- Comply with **CSI**
- Cloud native integration
- Handle Persistent Volume lifecycle
- Must support our goal platform (bare-metal)

Using these criteria and taking a look at the **CNCF** landscape, these are the products which seem suitable for this purpose:
- CubeFS
- Rook
- Longhorn

A.5.2. Evaluation criteria

As a storage interface, performance is an important criterion. To fully evaluate the performance, a standardized test with defined workloads etc. would need to be performed. Since this isn't feasable for this evaluation, we try to evaluate the performance in relation to the scale, in which we would deploy the platform, based on community experience and claimed infrastructure goals.

Otherwise, features such as reliability/fault tolerance are vital, to allow for some error before the whole system crashes. Other storage-related features like snapshots and encryption should also be part of the package. Otherwise, standard criteria include observability, ease of deployment and maintenance/support.

This results in the following list of evaluation criteria:
- Suited for project scenario: Does the product work well with our intended infrastructure scale or is ot over- or underprovisioned?
- Reliability: Does the product have fault tolerance/high availability features?
- Snapshots: Are snapshots or backups supported?
- Encryption: Can data at-rest (volumes) or in-transit be encrypted?
- Observability: Does it integrate into well-known observability tools?
- Ease of deployment: Can it be deployed and configured easily in our scenario?
- Maintenance/Support: Is it actively maintained and supported?

A.5.3. CubeFS

CubeFS is a storage plugin native to Kubernetes. Besides container platforms, it's also suitable for big data, AI and other types of unstructured data. It is fully open source and community driven. [38]

Features: A list of key features is present on their main page: [38]
- Multi-Protocol: It supports different network protocols like S3 or POSIX.

- Highly Scalable: It supports distributed storage solutions up to exobyte levels.
- High Performance: With caching levels and replication protocols and good performance even for small files.
- Multi-Engine: Supports different types of fault tolerance using replication or redundancy simlar to a Redundant Array of Independent Disks (**RAID**).
- Multi-Tenancy: Fine-grained isolation policies ensure a full sepration between tenants.
- Cloud-Native: Specifically built for Kubernetes and its **CSI**.

Challenges: While CubeFS is powerful and cloud-native, a few potential challenges should be considered:

- Maturity: CubeFS is relatively new compared to other solutions and hence could pose some problems with newer implementations.
- Operational Complexity: Even though it's Kubernetes-native, managing and tuning distributed file systems (metadata nodes, data nodes, object storage integration) can introduce complexity.
- Documentation & Community Size: Although growing, the community is smaller than other CNCF projects, meaning fewer troubleshooting resources and integration guides exist.
- Limited Ecosystem Integrations: Some enterprise backup or observability tools may require custom configuration or lack direct support.

Criteria:

- Suited for project scenario:
  CubeFS achieves good performance for distributed workloads through metadata caching and replication optimizations. It is designed to handle both small-file and large-object scenarios efficiently. It works especially well in big data scenarios. [39]
- Reliability:
  Reliability is provided through fault-tolerance mechanisms, either replication or erasure coding, which ensures data availability during node failures. Its metadata and data nodes can be deployed redundantly across multiple Kubernetes nodes for high availability. [40]
- Snapshots:
  Since CubeFS supports the **CSI**, it needs to provide a snapshot mechanism, enabling volume snapshots and restore operations. It also supports tiered storage, which can simplify backup strategies across multiple storage backends. [41]
- Encryption:
  CubeFS doesn't fully support encryption at-rest and in-transit. Data in-transit can be secured via TLS, while storage encryption depends on configuration of the underlying storage layer. [42]
- Observability:
  The system exposes metrics and logs that can be integrated with popular tools like *Prometheus* and *Grafana* for monitoring. [43] While basic observability is supported, detailed dashboards and alerting templates are limited and may require manual setup.
- Ease of Deployment:
  CubeFS can be deployed using a Helm chart and the official docs provide a manual to implement it with Kubernetes. [44] Automation shouldn't be a problem.

- Maintenance/Support:
  The project is actively maintained on GitHub with regular commits and version releases. Support is community-based via Slack and GitHub Issues.

A.5.4. Rook

Rook is a storage orchestrator, which in itself doesn't provide the storage solution. It relies on *Ceph*, which acts as the distributed file system. Rook integrates it into a cloud native environment while adding self-managing and self-healing capabilities. [45]

Features: Rook provides an abstraction layer that automates deployment, configuration, scaling, and management of the storage backend (Ceph) inside Kubernetes. Key features listed on the website [46] include:

- Simple and reliable automated storage management
- Hyper-scale or hyper-converge your storage clusters
- Efficiently distribute and replicate data to minimize loss
- Provision file, block, and object storage
- Manage open-source Ceph storage
- Easily enable elastic storage in your datacenter
- Open source software released under the Apache 2.0 license
- Optimize workloads on commodity hardware

Challenges: Despite its maturity and strong feature set, Rook introduces some operational and resource challenges:

- Complexity: Ceph is a large and complex system with many configuration options, making it harder to operate compared to simpler CSI backends.
- Resource Requirements: It requires significant CPU, memory, and disk resources and isn't well-suited for lightweight or edge environments, as some users report.
- Upgrade Management: Since this is based on two separate tools, version compatibility always needs to be considered for upgrades.
- Learning Curve: Administrators must understand Ceph concepts, which adds to operational overhead.
- Performance Tuning: To achieve optimal performance, a lot of fine-tuning is needed.

Criteria:

- Suited for project scenario:
  Given an optimal scenario with enterprise-grade hardware and a large scale infrastructure, this combination performs really well. However, for more basic workloads, this may be overkill and can't be sustained properly. [47]
- Reliability:
  Ceph provides replication, erasure coding, and self-healing, ensuring high availability and fault tolerance. Data is automatically redistributed if a node or disk fails. [48]
- Snapshots:
  It fully supports snapshots and clones through the Ceph CSI driver. Snapshots can be taken at the block or filesystem level and are natively integrated with Kubernetes VolumeSnapshot resources. [49]

- Encryption:
  Data at-rest and in-transit can both be encrypted. Rook allows configuration of encryption keys and key management via external solutions like Kubernetes secrets. [50]
- Observability:
  This solution provides observability through native integration with Prometheus and Grafana. [51] Metrics are exposed for all Ceph components, and Rook adds health checks and cluster status dashboards.
- Ease of Deployment:
  Rook offers a Helm chart and an Operator that automate most of the deployment and configuration. Initial setup requires defining Custom Resource Definitions (**CRDs**) for Ceph clusters and storage pools, which adds complexity but also flexibility.
- Maintenance/Support: Since Rook is a CNCF graduated project, it is actively maintained and has a large community. Enterprise-grade support is available indirectly through Red Hat Ceph Storage or SUSE Enterprise Storage.

A.5.5. Longhorn

Longhorn is a distributed block storage system for Kubernetes. It was originally developed by Rancher Labs and is now part of the **CNCF** as an incubating project. [52]

Features: The developers list the following features as part of Longhorn: [52]

- Persistent Volumes: With Longhorn volumes, you can provide persistent storage to your stateful applications.
- Partitioning: Split your block storage into different volumes for Kubernetes to use.
- Replication: Data gets replicated accross multiple nodes for better availability.
- Backup: Your data can get saved onto external storages like S3. Snapshots can also get scheduled.
- Disaster Recovery: Using disaster recovery volumes, you can restore your data onto a secondary Kubernetes cluster.
- Maintainability: Longhorn can be upgraded without a service disruption.

Challenges:

- Performance Overhead: Longhorn replicates data at the block level, which can introduce latency.
- Resource Consumption: The synchronous replication mechanism increases network and disk usage, which can be quite a lot on smaller clusters or nodes with limited resources.
- Scalability Limits: While Longhorn works well for small to medium clusters, it may not scale efficiently in very large environments compared to other solutions.
- Feature Maturity: Although actively developed, some advanced enterprise features like tiered storage are less mature than in older systems.

Criteria:

- Suited for project scenario:
  Longhorn provides decent I/O performance for general Kubernetes workloads, with overhead mainly due to synchronous replication between nodes.
- Reliability:
  It is designed for high availability with self healing features in case of disk or node failures. [52]

- Snapshots:
  It supports snapshots and backups also with scheduling and restoring options. Connection to external storages is also possible for backup purposes. [53]
- Encryption:
  Longhorn allows for volume and block level encryption. [54] Transport encryption can be achieved through standard Kubernetes security mechanisms.
- Observability:
  Longhorn also has native Prometheus and Grafana integrations and provides dashboards for volumes etc.. [55]
- Ease of Deployment:
  The installation process is straightforward using the official Helm chart. The web UI simplifies volume management and troubleshooting, although there may arise some challenges during automation. [56]
- Maintenance/Support:
  Longhorn is an Incubating CNCF project maintained by SUSE (Rancher team) and the open-source community. It is actively developed and has regular releases, as well as a strong documentation.

A.5.6. Decision matrix

Criterion	CubeFS	Rook	Longhorn
Suited for project scenario	✓ Fulfilled	✓ Partial fulfillment	✓ Partial fulfillment
Reliability	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Snapshots	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Encryption	✗ Unfulfilled	✓ Fulfilled	✓ Fulfilled
Observability	✓ Partial fulfillment	✓ Fulfilled	✓ Fulfilled
Ease of deployment	✓ Fulfilled	✓ Partial fulfillment	✓ Fulfilled
Maintenance/Support	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled

Table 18: Storage Decision Matrix

A.5.7. Conclusion

Generally, CubeFS would be a great option for our scenario, since it is Kubernetes native and integrates seamlessly. However, some of the features may not be fully developed and it isn't as mature as the other solutions. On the other hand, Rook in combination with Ceph is the most potent solution, but it aims for bigger clusters than the use case suggests, and hence may be overkill for this scenario. This makes Longhorn seem like the optimal solution in the scope of this project, as it's a good compromise between the other two and provides all the needed functionality in a single package.

A.6. Evaluation of Container Runtime

The Container Runtime is responsible for running and managing containers on each node. It provides the environment in which application containers are executed.

A.6.1. Selection criteria

After taking a look at the **CNCF** Landscape, it becomes clear that there are two main container runtimes:

- containerd
- cri-o

These are both in the graduated state and so far also the only ones the team has ever made any experiences with.

A.6.2. Evaluation criteria

As the container runtime isn't something the operator will directly interact with, the evaluation criterias will be based on installation and quality of life considerations. Hence, the following criterias were defined:

- Ease of automation: Does it implement well with the desired solution?
- Support for orchestration platform: How well does it work with the suggested platform?
- Security considerations: What is the default security standard or how well can it be optimized?
- Supported by container-optimized **OS**: Does it work well in combination with e.g. Talos or Flatcar?

A.6.3. containerd

Containerd is a widely used and often times default container runtime, especially in combination with *Docker*. It is an industry-standard and focuses on simplicity, robustness and portability. [57]

Features: These features are supported by the runtime according to their website [57]:

- OCI Image Spec support
  The Open Container Initiative (**OCI**) provides a standardized format for containers and containerd fully supports this standard.
- Image push and pull support
  The image management is a task of the container runtime. containerd supports different network protocols and image caching for enhanced performance.
- Network primitives for creation, modification, and deletion of interfaces
  containerd can handle the commands of the orchestration tool and enable low level interaction with the Linux networking.
- Multi-tenant supported with CAS storage for global images
  As container images are built in layers, some can contain the same content accross different tenants. Using Content-Addressed Storage (**CAS**) to store image layers, they can be reused independently.
- OCI Runtime Spec support (aka runC)
  Similar to the image spec, **OCI** defines a standard for running containers. containerd implements the reference standard *runc*.
- Management of network namespaces containers to join existing namespaces
  containerd allows for fine-grained control over namespaces.

Challenges: While containerd is widely adopted and stable, there are several aspects that may pose challenges when integrating it into a custom installer environment:

- Limited Operator Visibility: As a low-level system component, troubleshooting problems with containerd can be quite hard and the right tools are required.
- Configuration Management: Although the installation is straightforward, configuration files differ slightly between distributions. Ensuring correct integration to the other plugins needs to be kept in mind.
- Version Synchronization: Kubernetes and containerd versions must remain compatible. Using mismatched releases can lead to communication issues or unexpected behavior.
- Security Hardening: While containerd supports all modern security mechanisms, applying and validating these configurations across multiple hosts can be operationally demanding.

Criteria:

- Ease of automation:
  containerd packages are widely available. It integrates seamlessly with common automation tools (Ansible, Terraform) and Kubernetes setup workflows and comes preinstalled on some container-optimized **OS**s like *Talos*. [58]
- Support for orchestration platform:
  containerd is the default runtime in most Kubernetes distributions and managed services. It is fully **OCI**-compliant and actively maintained by CNCF. [57]
- Security considerations:
  It supports rootless mode, seccomp, AppArmor, and integrates well with container signing frameworks. Some security tuning still requires OS-level adjustments.
- Supported by container-optimized **OS**:
  It is the default runtime in Talos, Bottlerocket, and Ubuntu-based distributions and otherwise also widely supported.

A.6.4. CRI-O

CRI-O is an open-source container runtime built specifically for Kubernetes. It directly implements the Container Runtime Interface (**CRI**) provided by Kubernetes, offering a minimal and secure runtime layer between Kubernetes and the underlying operating system. [59]

Features: According to its maintainers and documentation, CRI-O provides the following functionality:

- Native CRI implementation: CRI-O was built specifically to implement Kubernetes' **CRI** without additional shims or translation layers.
- OCI Image and Runtime Spec support: Like containerd, CRI-O fully supports the OCI Image and Runtime Specifications.
- Runtime flexibility: CRI-O can use multiple OCI-compliant runtimes, including runc (default), crun (lightweight), and Kata Containers (hardware virtualization), enabling different isolation and performance profiles.
- Image management and caching: It uses skopeo and containers/storage libraries for pulling, caching, and managing images efficiently. Shared layer storage optimizes resource usage across pods.

- Integrated security frameworks: Deep integration with Linux security modules. It supports rootless operation, read-only filesystems, and drop-in policies for hardening container isolation.
- Logging and metrics integration: Native support for structured logging (JSON, journald) and integration with Kubernetes node metrics through CRI stats API, allowing for observability without external tooling.
- Network namespace and CNI support: Like containerd, CRI-O delegates networking to the **CNI**, allowing creation and management of namespaces and network interfaces as required by Kubernetes pods.

Challenges: While CRI-O is efficient and secure, there are several potential challenges when integrating it into a custom Kubernetes installer:

- Smaller ecosystem outside Red Hat: CRI-O is primarily adopted in OpenShift and Fedora/RHEL CoreOS environments, leading to fewer pre-built integrations for other **OS**s.
- Version coupling with Kubernetes: CRI-O releases are tightly bound to specific Kubernetes versions. This simplifies alignment but complicates multi-version support and upgrades.
- Automation variability: Only a few off-the-shelf automation scripts or Ansible roles exist, requiring additional customization during installation.
- Community size: While it is CNCF graduated and stable, the CRI-O community is relatively small, resulting in fewer third-party guides and integrations.

Criteria:

- Ease of automation:
  Generally, it isn't pre-installed a lot of the times and automating the installation may be harder.
- Support for orchestration platform:
  Since CRI-O is specifically for Kubernetes, it implements the CRI directly with no intermediary layers. [59]
- Security considerations:
  Native SELinux integration, seccomp, AppArmor, and support for rootless mode make it very strong. It follows Red Hat security hardening standards.
- Supported by container-optimized **OS**:
  It is associated with SUSE/Red Hat systems, but not really common on **OS**s like Talos or Flatcar.

A.6.5. Decision matrix

Criterion	containerd	cri-o
Ease of automation	✓ Fulfilled	✓ Partial fulfillment
Support for orchestration platform	✓ Partial fulfillment	✓ Fulfilled
Security considerations	✓ Fulfilled	✓ Fulfilled
Supported by container-optimized OS	✓ Fulfilled	✓ Partial fulfillment

Table 19: Container Runtime Decision Matrix

A.6.6. Conclusion

In most of the criterias, both of the runtimes are pretty similar. containerd provides better support for automation, while cri-o was developed specifically for Kubernetes. However, the underlying **OS**

needs to work well in combination with the chosen runtime. Since cri-o works best with Red-Hat based **OSs**, we will further proceed using **containerd**.

A.7. Evaluation of Cloud Native Network

The Cloud Native Network plugin is responsible for providing network connectivity between containers and handling ingress/egress traffic. It assigns IP adresses to the pods and acts as an abstraction layer, so that the container management software can interact with it.

A.7.1. Selection criteria

When taking a look at the **CNCF** landscape, there are two graduated projects:

- Cilium
- **CNI**

However, **CNI** only defines a "standard" and implementations need to comply with it. From the other projects in the landscape, together with Cilium, two other implementations will be regarded:

- Calico
- Flannel

These have been covered during the bachelor studies and provide a good base for an evaluation.

A.7.2. Evaluation criteria

Since the networking plugin is an abstraction layer, it needs to comply with certain standards given by the orchestration platform. This will influence the evaluation criteria, which were defined as:

- **CNI** support: Is the standard fully implemented?
- Ease of automation: Is it possible to configure using YAML files or maybe Helm charts?
- Security features: Possibility to define features like network policies or traffic encryption?
- Compatibility with Ecosystem: Is the solution compatible with e.g. Service Meshes and is it version independent?
- Licensing: Are the features needed available under free licensing?

A.7.3. Cilium

Cilium is a networking solution built on top of the Extended Berkeley Packet Filter (**eBPF**) technology.[60] It is a graduated CNCF project, actively developed by a broad community.

Features: A selection of key Cilium features includes:

- **eBPF**-based Data Plane: It eliminates the need for iptables, providing high performance and low latency network operations.
- Network Policies: Kubernetes native NetworkPolicies with L3/L4 filtering and L7 policies for protocols like HTTP are supported.
- Observability (Hubble): It has a built-in observability platform that provides deep network visibility, including flow tracing, service dependencies, and security audit trails.
- Kube-proxy Replacement: Cilium can replace kube-proxy entirely, simplifying the networking layer and improving service scalability.
- Encryption Support: Native support for node-to-node encryption using IPSec or WireGuard.
- Service Mesh Integration: Cilium can function as a lightweight service mesh or integrate with existing ones like Istio or Linkerd.
- Multi-cluster Support: It allows secure connectivity and policy enforcement across multiple Kubernetes clusters.

Challenges:

- Kernel Dependency: Requires a relatively recent Linux kernel with eBPF support. Older or custom OS images may face compatibility issues.
- Complexity: The large feature set can make configuration more complex.
- Learning Curve: The eBPF-based approach introduces new concepts that may require additional operational knowledge.
- Resource Overhead: The control plane components such as Hubble and Cilium agents consume some resources.

Criteria:

- CNI support:
  Cilium is fully compliant with the CNI specification and deeply integrated into Kubernetes. It supports native Kubernetes NetworkPolicies and can replace kube-proxy. [61]
- Ease of Automation:
  Cilium can be installed and configured declaratively using Helm charts, Cilium CLI, or YAML manifests. [62]
- Security Features:
  It provides extensive security capabilities including L3/L4/L7 network policy enforcement, identity-based access control, and encryption in transit. [63]
- Compatibility with Ecosystem:
  Cilium integrates natively with service meshes (Istio, Linkerd), observability stacks (Prometheus, Grafana), and container runtimes (CRI-O, containerd). [60] It is version-independent from the Kubernetes API and actively tested against major releases, ensuring consistent compatibility.
- Licensing:
  Cilium itself is open source and licensed under Apache 2.0. However, it provides enterprise support through partners if needed.

A.7.4. Calico

Calico is a single platform for all the networking desires of containerized workflows. It is developed and maintained by Tigera Inc. and part of the **CNCF**. [64] It offers four different tears, each with different feature sets:

- Calico Open source
- Calico Cloud Free Tier
- Calico Cloud
- Calico Enterprise

Features: As an allround solution, these are a few of the features listed on the Calico page [64]:

- High performance, scalable pod networking
- Advanced IP address management
- Different data planes (iptables, nftables, **eBPF**)
- Ingress Gateway
- Support for Kubernetes and custom network policies
- Data in-transit encryption

Challenges:

- Complex Configuration: While Calico offers rich functionality, it comes with a complex configuration model.
- Operational Overhead: Managing BGP routing or switching between data planes (iptables vs eBPF) can increase operational complexity.
- Resource Consumption: The control plane adds quite a bit of CPU and memory overhead compared to simpler CNIs.
- Feature Fragmentation: Some advanced features (multi-cluster management, enhanced observability, compliance dashboards) are only available in Calico Enterprise.

Criteria:

- CNI support:
  Calico is fully compliant with the CNI specification and deeply integrated into Kubernetes. It provides native support for Kubernetes NetworkPolicies while also extending them with Calico-specific policy definitions. [65]
- Ease of Automation:
  Installation and configuration are well supported through Helm charts, YAML manifests, and Operators. Calico can be managed declaratively, making it suitable for automation through Terraform, Ansible, or cluster installers. [66]
- Security Features:
  Calico includes comprehensive security controls: NetworkPolicies, Global NetworkPolicies, and optional WireGuard/IPsec encryption for node-to-node traffic. Enterprise editions add compliance and threat-detection capabilities. [64]
- Compatibility with Ecosystem:
  Calico integrates with major Kubernetes distributions, supports service meshes (Istio, Linkerd), and provides Prometheus/Grafana observability integrations. [64]
- Licensing:
  While the basic features can be used in the free open source tier, there are definitely some features, especially when considering an enterprise-level infrastructure, which need an extra license. [64]

A.7.5. Flannel

Flannel provides a simple Layer 3 fabric for Kubernetesa and its pods. It also implements the **CNI** standard and is part of the **CNCF**.

Features:

- Simple Architecture: Flannel provides straightforward pod networking without complex routing or policy components.
- Multiple Backends: It supports several data plane backends such as VXLAN, host-gw, AWS VPC, or UDP, allowing flexibility depending on the infrastructure.
- Lightweight and Easy to Deploy: It has minimal dependencies and configuration requirements.
- CNI-Compliant: Flannel is fully compatible with Kubernetes and standard container runtimes (CRI-O, containerd).
- Automatic IP Address Management: Built-in IPAM that assigns subnet leases to each node, ensuring non-overlapping pod CIDRs.

- Cloud and Bare-Metal Compatibility: It doesn't matter if you're on an on-premise or cloud environment.

Challenges:

- Limited Feature Set: Flannel focuses only on Layer 3 connectivity. It does not support NetworkPolicies, encryption, or advanced observability.
- No Native Security Controls: Lacks built-in mechanisms for isolating traffic or defining policies.
- Scalability Constraints: VXLAN encapsulation can cause network overhead at large scales and performance degrades when clusters grow significantly.
- No Native High Availability: It relies on etcd or Kubernetes API for state management.
- Minimal Troubleshooting Tools: There's only limited visibility into network flows or performance metrics compared to modern CNIs.

Criteria:

- CNI support:
  Flannel is fully CNI-compliant and integrates neatly with Kubernetes. It provides basic pod networking and works out-of-the-box with kubelet and standard runtime configurations.
- Ease of Automation:
  Flannel is extremely easy to deploy using simple YAML manifests or via Helm charts. It requires minimal configuration and few dependencies, making it ideal for automated installers and bootstrapping clusters. [67]
- Security Features:
  Security capabilities are minimal. Flannel provides no encryption or policy enforcement by itself. Clusters requiring isolation or compliance features typically combine Flannel with another policy engine.
- Compatibility with Ecosystem:
  As one of the oldest and simplest CNI implementations, Flannel enjoys broad compatibility with Kubernetes, container runtimes, and cloud environments. It integrates well with managed Kubernetes services.
- Licensing:
  Benig licensed under Apache 2.0, the full feature set is available commercially.

A.7.6. Decision matrix

Criterion	Cilium	Calico	Flannel
CNI support	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Ease of automation	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Security features	✓ Fulfilled	✓ Fulfilled	✗ Unfulfilled
Compatibility with Ecosystem	✓ Fulfilled	✓ Fulfilled	✓ Fulfilled
Licensing	✓ Fulfilled	✓ Partial fulfillment	✓ Fulfilled

Table 20: Cloud Native Networking

A.7.7. Conclusion

As the most versatile and complete option, being a **CNCF** graduated project and fully open source, we choose to work with **Cilium**. Flannel is too simple for our needs and doesn't scale well in bigger scenarios, and Calico doesn't offer the full feature set without having the enterprise licenses.

A.8. Evaluation of Scheduling and Orchestration Technologies

Orchestration and scheduling refer to running and managing containers across a cluster. Container orchestrators automate container management. They handle tasks like deployment, scaling, and networking. Scheduling is about deciding where to run containers based on resource availability and requirements. [1]

A.8.1. Selection criteria

- **Maturity**: The product is widely adopted, actively maintained, and has a proven track record in production environments.
- **Scalability**: Capable of supporting large-scale deployments to accommodate potential future scalability requirements, and support dynamic, automated application scaling in alignment with our fictional scenario.
- **Purpose-Built**: The tool is built for container orchestration and scheduling.

Products excluded by these criteria:

- **HashiCorp Nomad (open-source)**: The open-source version lacks ability for automated application scaling. [68]
- **Apache Mesos & Marathon**: Apache Mesos has been moved "in the Attic" and is no longer actively maintained. [69]
- **CoreOS fleet**: Has been deprecated and is no longer maintained. [70]
- **Docker Swarm**: No built-in dynamic application scaling.
- **OpenShift**: Enterprise distribution of Kubernetes, not a separate orchestrator.
- **Rancher**: Management platform for Kubernetes, not an orchestrator itself.
- **Amazon ECS/EKS, Azure AKS, Google GKE**: Managed Kubernetes services, not standalone orchestrators.

Products to evaluate:

- Kubernetes:
  - Maturity: Has become the de facto standard for container orchestration with a large ecosystem and active community support. [71]
  - Scalability: Designed to handle large-scale deployments with built-in features for scaling applications. [72]
  - Purpose-Built: Specifically built for managing containerized applications. [73]
- HashiCorp Nomad Enterprise:
  - Maturity: Gaining traction with a growing user base and active development. [74] Significant adapters include companies like Cloudflare, eBay, and SAP. [75]
  - Scalability: Capable of scheduling 1 million containers on 1'000 machines within 5 minutes. [76]
  - Purpose-Built: Designed to orchestrate both containerized and non-containerized applications. [77]

A.8.2. Evaluation criterias

- Support: The technology is backed by an enterprise that offers support.
- Licensing: Code is for commercial use (e.g. Apache License 2.0).
- Ease of Use: The time needed to set up the product, the state of the documentation, the complexity of the configuration and maintenance.

- Security: Provides the security features authentication, authorization, access-based control like Access Control Lists (**ACL**s), auditing and container isolation.
- Automation: The technology supports automation through **API**, **CLI** tools, and infrastructure as code.
- Extensibility: It should be extensible via plugins or integrations to adapt to various use cases.
- Multi-tenancy: The technology should support multiple isolated environments within a single cluster.

A.8.3. Kubernetes

Kubernetes is an open source platform for automating deployment, scaling, and management of containerized workloads. The project is maintained by the **CNCF**. [78]

Features:

- Designed for extensibility, integrates with a vast ecosystem.
- Built-in service discovery, load balancing, and self-healing.
- Automated rollouts and rollbacks.
- Horizontal scaling and resource optimization.
- Secrets and configuration management.
- Storage orchestration.
- Batch execution.

[78]

Challenges: Steep learning curve and complexity.

Criteria:

- Support: Kubernetes is backed by the **CNCF** and supported by multiple enterprises offering commercial support options. [79]
- Licensing: Apache-2.0 license [80]
- Ease of Use: Kubernetes has a steep learning curve due to its complexity. However, it has well-maintained and well-written documentation with many examples and also a large community that provides tutorials, guides, and support. [73]
- Security: Kubernetes offers control plane and workload protection, a secret API, and auditing. [81]
- Automation: Kubernetes supports automation through its **API** [82], **CLI** tools like kubectl [83], and declarative configuration. It integrates well with infrastructure as code tools like Terraform [84].
- Extensibility: Kubernetes offers extensibility through its **API**, custom controllers, operators, plugins, and integrations for scheduling, networking, and storage. [85] It has also a large ecosystem of third-party tools and extensions. [86]
- Multi-tenancy: Kubernetes supports multi-tenancy through control and data plane isolation mechanisms such as namespaces, quotas, node isolation and some others. [87]

A.8.4. HashiCorp Nomad Enterprise

Nomad is a simple and flexible orchestrator to deploy and manage any containerized or legacy application. [74]

Features:

- Supports containerized and legacy applications.

- Runs as a single binary, requiring no external services.
- Is distributed and resilient, providing high availability through built-in leader election and state replication.
- Able to run 1 million containers on a 1'000 machines within 5 minutes.
- Integrates with HashiCorp Ecosystem like Vault, Consul and Terraform.

Challenges:
- Smaller community and ecosystem compared to Kubernetes.
- Payed solution needed for enterprise features.

Criteria:
- Support: Nomad is backed by HashiCorp, which offers commercial support and enterprise features. [88]
- Licensing: Nomad is available under the commercial Business Source License (**BSL**). [74]
- Ease of Use: "Nomad is available as a pre-compiled binary or as a package for several operating systems." [89] Its setup is more straightforward than kubernetes, as state replication across clustered nodes removes the need for external storage systems like etcd. [90] The documentation is comprehensive and provides clear examples and guides. [91]
- Security: Nomad offers access control, **SSO**, auditing, and container isolation. [68]
- Automation: Nomad supports automation through its **HTTP API** [92], **CLI** tools [93], and declarative configuration. It also integrates well with infrastructure as code tools like Terraform. [94]
- Extensibility: Nomad is extensible through its plugin system, allowing custom drivers for different workloads and integrations with other tools. However, its ecosystem is considerably smaller compared to Kubernetes. [95]
- Multi-tenancy: Nomad offers multi-tenancy through namespaces. [96]

A.8.5. Decision matrix

Criterion	Kubernetes	HashiCorp Nomad Enterprise
Maintenance	✓ Fulfilled	✓ Fulfilled
Support	✓ Fulfilled	✓ Fulfilled
Licensing	✓ Fulfilled	✗ Unfulfilled
Ease of Use	✓ Partial fulfillment	✓ Fulfilled
Security	✓ Fulfilled	✓ Fulfilled
Ecosystem	✓ Fulfilled	✓ Partial fulfillment

Table 21: Container Orchestration Decision Matrix

A.8.6. Conclusion

There are not many maintained container scheduler and orchestrator left. Kubernetes stands out as the most viable orchestration solution. While HashiCorp Nomad is lightweight and simple it lacks the ecosystem maturity and relies on external integrations like Consul and Vault to achieve the extensibility and robustness. Additionally, its **BSL** restricts enterprises from competing in commercial offerings. Kubernetes, despite its operational complexity, benefits from a big open-source community, strong scalability, and full flexibility across infrastructures. This makes it an ideal foundation for

a customizable, flexible, enterprise-grade container platform suitable for organizations even if they have strict and complex requirements.

A.9. Evaluation of Service Discovery Technologies

In order for application to collaborate and communicate with each other, they need to be able to locate each other. Service discovery addresses this problem by providing a common place to find and identify services:

- Service discovery engines store information on all services and how to locate them.
- Name resolution tools receive service location requests and return network address information. [1]

A.9.1. Selection criteria

- **Best-practice**: The technology is considered best-practice for Kubernetes, our selected scheduler and orchestration technology.
- **Active maintenance**: The technology is actively maintained.

Products excluded by these criteria:

- **KubeDNS**: KubeDNS is not the recommended Domain Name System (**DNS**) solution for Kubernetes anymore. [97]
- **Zookeeper**: Zookeeper is not the recommended **DNS** solution for Kubernetes. [97] Zookeeper is primarily designed as a distributed coordination service. [98] Its complexity and operational overhead make it less suitable.

Products to evaluate:

- etcd + CoreDNS

A.9.2. etcd + CoreDNS

etcd is a distributed key-value store that provides a reliable way to store data across a cluster of machines. [99] CoreDNS is a fast and flexible **DNS** server that integrates with Kubernetes for service discovery. [100]

Criteria:

- Best-practice: Both are natively integrated into Kubernetes and are considered best-practices for service discovery in Kubernetes environments. [101]
- Active maintenance: Both etcd and CoreDNS are actively maintained and are **CNCF** graduated projects. [102]

A.9.3. Conclusion

Since we chose Kubernetes as our orchestration and scheduling platform, we select etcd and CoreDNS for service discovery and coordination, as they are natively integrated into Kubernetes and align to Kubernetes' architectural best practices.

A.10. Evaluation of Observability and Analysis Technologies

Observability is the ability to understand the state of a system based on the data it produces, such as logs, metrics, and traces. Analysis is the process of examining this data to gain insights, identify patterns, and make informed decisions. [1]

A.10.1. Selection criteria

- Maturity: The product is widely adopted, and has a proven track record in kubernetes production environments.
- Maintenance: The code is updated regularly (as in not stale).
- Integration: The product is Cloud-Native and has a rich integration ecosystem.
- Enterprise Readiness: Access control and compliance features.
- Cost: Core features should be free at least for non-commercial use.
- Deployment Environment: The stack should be deployable in the kubernetes cluster.
- Visualization: The product should provide a unified visualization for telemetry data (logs, metrics and traces). The product should provide the features dashboards, alerts, and notifications with templates for kubernetes observability and analysis.
- Log and metrics collection and aggregation: The product should support various input sources and output destinations.
- Log storage: The product should be able to index and query data in an efficient manner, and also provide a retention feature.
- Storage Scalability: The product can handle large volumes of log data and scale horizontally.
- Metrics storage: The product should store data as time-series data and support querying and aggregation.

Products excluded by these criteria: Inspector Gadget, M3, Nagios core, Falcon, Kindling, loggie, NexClipper, OpenTSDB, LinDB, deepflow, OpenObserve, HertzBeat, Zipkin, SigNoz, coroot, Anteon Stack (Ddosify, Alaz, Anteon Self-Hosted), Splunk Observability including AppDynamics, SolarWinds Observability, AppSignal, Applications Manager, Aspecto (BugSnag), Aternity (Riverbed), Blue Matador, Catchpoint, Causely, checkmk, Chronosphere, Dash0, DataSet, Datadog, DX O2 by Broadcom, Lightrun, Dynatrace, elastic, Embrace, Epsagon, Nightingale, Honeybadger, honeycomb, Humio, Icinga, Instana, Circonus, Kubetail, LogicMonitor, Mackerel, Mezmo, Middleware, Pinpoint, Pixie, Rizhiyi, Sensu, SkyWalking, StackState, TelemetryHub, Tanzu Observability, graylog, Zabbix, parseable, Centreon, LightStep, Influxdata, Logstash, Beats, Elasticsearch, Kibana, SigLens, VictoriaMetrics, Perses

Selected Products:

- Visualization: Grafana, OpenSearch Dashboards
- Log collection and aggregation: Fluent Bit, Fluentd, OpenTelemetry Collector, Alloy, Vector
- Log storage: OpenSearch Core, Grafana Loki
- Metrics collection: Prometheus, OpenTelemetry Collector, Vector
- Metrics storage: Thanos (Prometheus), Grafana Mimir
- Tracing collection: OpenTelemetry, Odigos
- Tracing storage: Grafana Tempo, OpenSearch Core

A.10.2. Evaluation criterias

- Support: The technology is backed by an enterprise support or strong, active community.

- Licensing: Code is for commercial use (e.g. Apache License).
- Ease of use: The product is easy to deploy, configure, and maintain.
- Integration: The product integrates well with other tools and systems in the observability ecosystem.

A.10.3. Visualization

A.10.3.1. Grafana

Grafana is maintained by Grafana Labs and is an open-source platform for monitoring and observability. It is used to visualize and analyze metrics, logs, and traces from various data sources. [103]

Features: Grafana Labs provides the following key features:

- Visualizations: Fast and flexible client side graphs with a multitude of options. Panel plugins offer many different ways to visualize metrics and logs.
- Dynamic Dashboards: Create dynamic & reusable dashboards with template variables that appear as dropdowns at the top of the dashboard.
- Explore Metrics: Explore your data through ad-hoc queries and dynamic drilldown. Split view and compare different time ranges, queries and data sources side by side.
- Explore Logs: Experience the magic of switching from metrics to logs with preserved label filters. Quickly search through all your logs or streaming them live.
- Alerting: Visually define alert rules for your most important metrics. Grafana will continuously evaluate and send notifications to systems like Slack, PagerDuty, VictorOps, OpsGenie.
- Mixed Data Sources: Mix different data sources in the same graph! You can specify a data source on a per-query basis. This works for even custom datasources.

[103]

Challenges:

- Access Control: More advanced access control features like Role-Based Access Control (**RBAC**) and **SSO** are only available in the Enterprise version.

Criteria:

- Maturity: The product is trusted and used by many big companies and organizations including Nvidia, Microsoft, CERN. [104]
- Maintenance: The product is actively maintained with regular updates and releases. [103]
- Integration: The product has a rich ecosystem of plugins and integrations for various data sources. [103]
- Enterprise Readiness: Grafana includes a user management system and supports various Identity and Access Management (**IAM**) solutions, though many advanced features are only available in the Enterprise Edition. [105]
- License/Cost: The product is licensed under the Affero General Public License (**AGPL**), which allows for commercial use. [103]
- Deployment Environment: The product can run in Kubernetes environments and can be deployed with Helm. [106]
- Support: In addition to Grafana Labs supporting it, Grafana has a very active community. [107]
- Ease of use: Grafana provides an intuitive web interface for creating and managing dashboards and has also many pre-built dashboards available. [108]

- Visualization: Grafana provides a unified visualization for telemetry data and can also be integrated with Alertmanager for alerts and notifications. [109]

A.10.3.2. OpenSerach Dashboards

OpenSearch Dashboards is an open-source data visualization tool designed to work with OpenSearch. The OpenSearch Project is a project of The Linux Foundation forked from Elasticsearch and Kibana. [110]

Features:

- Search: For searching through data using keywords, filters, and smart queries.
- Analytics: For analyzing data with interactive charts, dashboards, and tools to spot trends, patterns, and anomalies.
- Generative Aritificial Intelligence (**AI**): For conversational searches and data insights.

[111]

Challenges:

- Dependency: While OpenSearch Dashboards is specifically designed to integrate with OpenSearch, integration support is limited in tools such as Grafana, where OpenSearch is only accessible through third-party plugins.

Criteria:

- Maturity: The product is backed by many big companies and organizations including AWS, CERN, and Canonical. [112]
- Maintenance: The product is actively maintained with regular updates and releases. [113]
- Integration: The product is built specifically for OpenSearch and includes plugins that extend its core functionality with additional features and capabilities. [114]
- Enterprise Readiness: The product includes extensive features for authentication, authorization and even multi-tenancy. [115]
- License/Cost: The product is licensed under the Apache-2.0 license, which allows for commercial use. [113]
- Deployment Environment: The product provides an Kubernetes operator to automate the deployment and provisioning of OpenSearch. [116]
- Support: The OpenSearch Project provides many support options including a community forum, and solution providers. [117]
- Ease of use: The product provides an intuitive web interface for creating and managing dashboards.
- Visualization: The product provides a unified visualization for telemetry data with alerting. It also has a template for Kubernetes visualization, querying and projection of resource data such as flow logs. But in comparison with Grafana it is less powerful and flexible. [118]

A.10.3.3. Decision matrix

Criterion	Grafana	OpenSerach Dashboards
Maturity	✓ Fulfilled	✓ Fulfilled
Maintanance	✓ Fulfilled	✓ Fulfilled
Integration	✓ Fulfilled	✓ Partial fulfillment
Enterprise Readiness	✓ Partial fulfillment	✓ Fulfilled
License/Cost	✓ Fulfilled	✓ Fulfilled
Deployment Environment	✓ Fulfilled	✓ Fulfilled
Support	✓ Fulfilled	✓ Fulfilled
Ease of use	✓ Fulfilled	✓ Fulfilled
Visualization	✓ Fulfilled	✓ Partial fulfillment

Table 22: Visualization Decision Matrix

A.10.3.4. Conclusion

Both Grafana and OpenSearch Dashboards are mature, actively maintained, and widely adopted visualization platforms. Grafana stands out for its flexibility, intuitive user experience, and rich plugin ecosystem, making it the preferred choice for our dynamic and varied requirements. In contrast, OpenSearch Dashboards is tightly integrated with the OpenSearch stack and offers robust enterprise features, but its limited flexibility and less advanced customization make it less ideal for our use case. Considering these factors, we have choose Grafana for its versatility, ease of use, and strong community support.

A.10.4. Log storage

A.10.4.1. Grafana Loki

"Loki is a horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus." [119]

Challenges:

- Loki uses an object storage in the scalable architecture.

Criteria:

- Maturity: The product is trusted and used by many big companies and organizations including Nvidia, and Sky. [104]
- Maintenance: The product is actively maintained with regular updates and releases. [119]
- Integration: Loki integrates with Grafana natively and supports log shippers like Fluent Bit and Fluentd. [120]
- Enterprise Readiness: Authentication should be achieved with an reverse proxy in front of Loki as it does not provide authentication natively. [121] Multi-tenancy is supported. [122]
- License/Cost: The product is licensed under the **AGPL**, which allows for commercial use. [119]
- Deployment Environment: The product can run in Kubernetes environments and can be deployed with Helm. [123]
- Support: In addition to Grafana Labs supporting it, Grafana has a very active community. [107]
- Ease of use: The product is simple to set up and manage with Helm charts. [124]

- Log storage: The product is able to index and query data in an efficient manner, and also provides a retention feature. [125]
- Storage Scalability: The product can handle large volumes of log data and scales horizontally. [119]

A.10.4.2. Conclusion

OpenSearch Core is a powerful and flexible log storage solution. However, it integrates less seamlessly with our chosen visualization tool, Grafana. Grafana Loki is designed to work natively with Grafana, it is horizontally scalable and efficient in log indexing but comes with the challenge of needing a object storage in the scalable architecture. All things considered, we have chosen Grafana Loki for log storage in our observability stack mainly because of the seamless integration.

A.10.5. Log collection

A.10.5.1. Logging operator (Fluent Bit + Fluentd)

The Logging operator manages the log collectors and log forwarders of the logging infrastructure. It deploys and configures Fluent Bit as a log collector and Fluentd as a log forwarder. [126]

Criteria:

- Maturity: The product is trusted and supported by many big companies and organizations including Cisco, Rancher, and Dailymotion. [127]
- Maintenance: The product is actively maintained with regular updates and releases. [126]
- Integration: Fluent Bit and Fluentd have a rich ecosystem of plugins and integrations for various data sources and destinations. [128], [129]
- Enterprise Readiness: The logging operator supports **RBAC** by default [130] and allows to have Nodegroup-based multi-tenancy. [131]
- License/Cost: The product is licensed under the Apache License 2.0, which allows for commercial use. [126]
- Deployment Environment: The product is designed to run in Kubernetes environments. [126]
- Support: The product has an active community discord channel. [126]
- Ease of use: The logging operator simplifies the deployment and management of Fluent Bit and Fluentd in Kubernetes environments. [126]

A.10.5.2. Conclusion

As we want to focus on **CNCF** projects as much as possible we choose the Logging Operator to manage our log collection and aggregation. It provides a Kubernetes-native way to deploy and manage Fluent Bit and Fluentd. Grafana Alloy is a promising alternative especially for a grafana centric stack, the vendor-neutral OpenTelemetry Collector and Vector are also promising alternatives with much higher metadata enrichment capabilities, but alle these tools are less mature and seam more complex to set up in comparison to the Logging Operator.

A.10.6. Metrics storage

A.10.6.1. Prometheus

Prometheus, a **CNCF** project, is an open-source monitoring system and time series database. [132]

Features:

- A time series database with time series defined by metric name and set of key/value dimensions
- PromQL as a flexible query language

- No dependency on distributed storage; single server nodes are autonomous
- An HTTP pull model for time series collection
- Pushing time series is supported with an intermediary gateway for batch jobs
- Targets are discovered with service discovery or static configuration
- Support for hierarchical and horizontal federation

[132]

Criteria:

- Maturity: The product is trusted and used by many big companies and organizations including SAP, and Grafana Labs.
- Maintenance: The product is actively maintained with regular updates and releases. [132]
- Integration: There are many integrations and also many observability systems that integrate Prometheus itself. [133]
- Enterprise Readiness: The product supports authentication, authorization, and encryption implemented with basic authentication, mutual Transport Layer Security (**mTLS**), and bcrypt. [134] For more advanced features like multi-tenancy Thanos should be considered. [135]
- License/Cost: The product is licensed under the Apache-2.0 license, which allows for commercial use. [132]
- Deployment Environment: The product can run in Kubernetes environments and can be deployed with Helm. [136]
- Support: There are courses and training as well as commercial support. [137]
- Ease of use: The product is simple to set up and manage with Helm charts. [136]
- Storage Scalability: For scalability the product can be extended with Thanos. [138]

A.10.6.2. Conclusion

As we want to focus on **CNCF** projects as much as possible we choose Prometheus as our metric storage. It provides a Kubernetes-native way to collect and store metrics. For scalability as well as for multi-tenancy support Thanos can be added to the stack. While Grafana Mimir is a promising alternative especially for a grafana centric stack, it requires more resources, operational expertise and is less mature.

A.10.7. Metrics collection

A.10.7.1. Prometheus Operator

The Prometheus Operator provides Kubernetes native deployment and management of Prometheus and related monitoring components simplifies and automates the configuration of a Prometheus based monitoring stack for Kubernetes clusters. It is maintained by the Prometheus community. [139]

Features:

- Kubernetes Custom Resources: Uses Kubernetes custom resources to deploy and manage Prometheus, Alertmanager, and related components.
- Simplified Deployment Configuration: Configures the fundamentals of Prometheus from a native Kubernetes resource.
- Prometheus Target Configuration: Automatically generates monitoring target configurations based on Kubernetes label queries.

[139]

Criteria:

- Maturity: The product is used by many big companies and organizations including CERN, OpenShift, and SUSE Rancher. [140]
- Maintenance: The product is actively maintained with regular updates and releases. [139]
- Integration: The product integrates well with Prometheus and Alertmanager. [139]
- License/Cost: The product is licensed under the Apache-2.0 license, which allows for commercial use. [139]
- Deployment Environment: The product runs natively in Kubernetes environments and can be deployed with Helm. [136]
- Enterprise Readiness: The product supports **RBAC**. [141]
- Support: There is only the github repository where an issue can be created. [142]
- Ease of use: The product is simple to set up and manage with Helm charts. [143]

A.10.7.2. Conclusion

As we use Prometheus as our metric storage the Prometheus Operator is the natural choice for metrics collection as it is also maintained by the Prometheus community. It provides a Kubernetes-native way to deploy and manage Prometheus and related components. If more enrichment is needed an option like the vendor-neutral OpenTelemetry Collector or Vector with much higher metadata enrichment capabilities, but also more complexity to set up and maintain could be considered.

A.10.8. Tracing storage

A.10.8.1. Grafana Tempo

Grafana Tempo is an open source, easy-to-use, and high-scale distributed tracing backend, requiring only object storage to operate, and is deeply integrated with Grafana, Prometheus, and Loki. [144]

Features:

- Intuitive Trace Analysis: Spot slow or error-prone traces with easy, point-and-click interactions.
- Rate Errors Duration (**RED**) Metrics Overview: Use Rate, Errors, and Duration metrics to highlight performance issues.
- Automated Comparison: Identify problematic attributes with automatic trace comparison.
- Simplified Visualizations: Access rich visual data without needing to construct TraceQL queries.

[144]

Challenges:

- Tempo uses an object storage in the scalable architecture.

Criteria:

- Maturity: The product is trusted and used by many big companies and organizations including Sky. [104]
- Maintenance: The product is actively maintained with regular updates and releases. [144]
- Integration: The product integrates well with Grafana, Loki, and Prometheus. [144]
- License/Cost: The product is licensed under the **AGPL**, which allows for commercial use. [144]
- Deployment Environment: The product runs in Kubernetes environments and can be deployed with Helm. [144]
- Enterprise Readiness: As Loki, authentication should be achieved with an reverse proxy in front of Tempo as it does not provide authentication natively. [145] Multi-tenancy is supported. [146]

- Support: In addition to Grafana Labs supporting it, Grafana has a very active community. [107]
- Ease of use: The product is simple to set up and manage with Helm charts. [147]

A.10.8.2. Conclusion

Since we're already using Grafana and Grafana Loki, Grafana Tempo meets all our requirements, it's a logical choice to use Grafana Tempo for a more unified and consistent observability stack. That said, if we had decided to go with the OpenSearch stack instead, using OpenSearch Core for storing traces, logs, and metrics would also have been a solid and reasonable choice.

A.10.9. Tracing collection

A.10.9.1. OpenTelemetry Collector

OpenTelemetry, a **CNCF** project, is a framework and toolkit designed to instrument, generate, collect, and export telemetry data. [148]

Features:

- Traces, Metrics, and Logs: Create and collect telemetry from services and software and forward them to analysis tools.
- Drop-in Instrumentation & Integrations: Supports auto-instrumentation for popular libraries and frameworks and code-based and zero-code instrumentation.
- Open-Source and Vendor Neutral

[148]

Criteria:

- Maturity: The product is a **CNCF** incubating project [148], but also already has some significant adopters like Alibaba, eBay, and GitHub. [149]
- Maintenance: The product is actively maintained with regular updates and releases. [150]
- Integration: The product has a rich ecosystem with many integrations. [151]
- License/Cost: The product is licensed under the Creative Commons Attribution 4.0 International License (**CC-BY-4.0**) and Apache-2.0, which allows for commercial use. [150]
- Deployment Environment: The product runs in Kubernetes environments and can be deployed with Helm. [152]
- Enterprise Readiness: The product supports authentication, encryption and **RBAC**. [153]
- Support: Besides GitHub issues and Stack Overflow there is currently no other support listed. [154]
- Ease of use: Even though the product is not that easy to set up it provides good documentation and helm charts to simplify the deployment and management. [152]

A.10.9.2. Conclusion

To collect and receive traces from Kubernetes and our applications, we've chosen the OpenTelemetry Collector. As a **CNCF** project, it offers a robust ecosystem and broad integration support. It provides a vendor-neutral approach to trace collection and integrates seamlessly with Grafana Tempo for trace storage and Grafana for visualization. For simpler management, Odigos could be considered, as it leverages the OpenTelemetry Collector internally. However, its smaller community and the fact that it's not a **CNCF** project are key reasons we opted against it.

A.10.10. Observability and Analysis Stack Conclusion

While Logging Operator, kube-prometheus-stack, OpenTelemetry Collector, and Grafana Tempo would each be our preferred products when considered individually for their enterprise-grade capabilities, we opted for a more streamlined approach to reduce overall stack complexity. By consolidating telemetry data collection through OpenTelemetry Collector, we simplify integration and maintenance while still achieving comprehensive observability across logs, metrics, and traces.

The chosen components are:

- ***Telemetry Data Collection & Aggregation***: OpenTelemetry Collector
- ***Log Storage***: Grafana Loki
- ***Metrics Storage***: Prometheus
- ***Tracing Storage***: Grafana Tempo

B. Architectural Decision Records

The **ADR**s in this chapter capture the design decisions and serve as reasoning for the component selection.

ADR-01: Operating System

Name	**Operating System**
Status	Accepted
Initial proposal date	23. October 2025
Last updated	28. October 2025

Table 23: ADR Operating System

Context and problem statement

Which **OS** is optimal to run our enterprise container platform on?

Decision drivers

- The **OS** is the fundamental part that all other technologies rely on.
- Should be optimized to run container applications.

Considered options

- Talos Linux
- Flatcar

Decision outcome

In the context of selecting the optimal **OS**, facing the need to have a reliable and compatible software stack, we decided to use **Talos Linux** and against Flatcar to achieve a solid infrastructure base, accepting that is has certain dependencies like using etcd or Flannel being preinstalled.

Consequences

- Good, because Talos fulfills all evaluation criteria as per Chapter A.1.
- Neutral, because Flatcar was not selected although being a valid alterative.

Confirmation

The use of Talos is confirmed when:

- A viable installation on bare-metal servers has been defined.
- All technologies used in the enterprise container platform work with Talos.

More information

An in-depth evaluation of the operating system options can be found in Chapter A.1.

ADR-02: Automation and Configuration

Name	Automation and Configuration
Status	Rejected
Initial proposal date	23. October 2025
Last updated	31. October 2025

Table 24: ADR Automation and Configuration

Context and problem statement

The tools in automation & configuration are about codifying the environment. The desired state is specified and the the container platform will match that desired state.

Decision drivers

- Evaluate options that are required to run an enterprise container platform.

Considered options

- KubeEdge
- Terraform
- Onmi by Sidero
- Metal³

Decision outcome

In the context of selecting the fitting automation and configuration tools, it was decided that none of these tools are relevant for this use-case, accepting that they are deployed and used by an the operator and are not part of the deployer role who installs a container platform.

Consequences

- Neutral: The deployer needs to install required tools listed in this section himself after the container platform has been set up.

More information

An in-depth evaluation of the operating system options can be found in Chapter A.2.

ADR-03: Identity Providers

Field	Identity Providers
Status	Rejected
Initial proposal date	23. October 2025
Last updated	27. November 2025

Table 25: ADR Identity Providers

Context and problem statement

Installing an **IdP** in the container platform is a use case defined for this project.

Decision drivers

- Having an **IdP** in the **ECP** is requirement that occurs frequently when running a container platform. Therefore it should be made available as an optional feature during deployment.

Considered options

- Zitadel
- Keycloak

Decision outcome

In the context of finding the most fitting **IdP**, facing the need to add it as an optional feature during deployment, it was decided to use Zitadel and not Keycloak, for having an **IdP** in the container platform, accepting that Keycloak would also fulfill most evaluation requirements and remains as a fallback option.

Consequences

- Good, Zitadel fulfills all requirements as per Chapter A.3.3.
- Neutral, Keycloak remains as a fallback option in case of technical problems during the implementation.
- Neutral, our team has no prior technical experience with Zitadel.

Confirmation

Zitadel is confirmed as the correct **IdP** once it can be successfully deployed in the **ECP** .

More information

A detailed in-depth evaluation of the operating system options can be found in Chapter A.3.3.

Update 27. November 2025

The **IdP** was an optional feature and due to time constraints it will not be implemented.

ADR-04: Certification Manager

Name	Certification Manager
Status	Accepted
Initial proposal date	23. October 2025
Last updated	3. November 2025

Context and problem statement

Installing a certification manager during the deployment of **ECP** is one of the use cases defined for this project. This **ADR** justifies the certification manager selection.

Decision drivers

- Having a cert-manager in the **ECP** is a requirement that occurs frequently when running a container platform. It should be made available as an optional feature.

Considered options

- cert-manager

Decision outcome

In the context of finding a fitting certificate manager, facing the need to add it as an optional feature during deployment, it was decided to use **cert-manager**, to achieve having a working cert-manager in the container platform, accepting that cert-manager is the only option fitting the requirements in this category.

Consequences

- Good, cert-manager is the goto option with a lot of resources available.
- Bad, no alternative tool available in this category.

Confirmation

cert-manager is confirmed as soon as can automatically be deployed in the **ECP**.

More information

A detailed in-depth evaluation of the operating system options can be found in Chapter A.3.4.

ADR-05: Vulnerability Scanners

Name	Vulnerability Scanners
Status	Accepted
Initial proposal date	23. October 2025
Last updated	3. November 2025

Context and problem statement

Installing a vulnerability scanner during the deployment of **ECP** is one of the use cases defined for this project. As security is essential for an enterprise, a vulnerability scanner can be pre-installed. This **ADR** justifies the vulnerability scanner selection.

Decision drivers

- Having a vulnerability scanner in the **ECP** improves the security of the cluster.
- It is one of the requirements defined for this project.

Considered options

- Grype
- Kubescape
- Falco

Decision outcome

In the context of finding the most fitting vulnerability scanner, facing the need to add it as feature during deployment, it was decided to use **Falco** and not Grype or Kubescape, to achieve having the most mature vulnerability scanner in our platform.

Consequences

- Good, because Falco is a graduated **CNCF** project with enterprise support, ensuring long-term stability and reliability.

Confirmation

Falco is confirmed as soon as it can be deployed automatically in our **ECP**.

More information

A detailed in-depth evaluation of the operating system options can be found in Chapter A.3.5.

ADR-06: Linter and Static Code Checker

Name	Linter and Static Code Checker
Status	Rejected
Initial proposal date	23. October 2025
Last updated	3. November 2025

Context and problem statement

Static code checkers and linters check the code for vulnerabilities.

Decision drivers

- Linter and static code checker are not part of the deployer role and therefore not implemented.

Considered options

- KubeLinter
- Checkov
- Clair(by Redhat)

Decision outcome

In the context of finding a fitting linter and static code checker, it was decided not to use any of the evaluated tools, accepting that installing these tools is part of the operator role and not in scope for the project.

Consequences

- Neutral, the operator must install a linter and static code checker after the deployment of **ECP**.

Confirmation

This **ADR** is not awaiting any further confirmation.

More information

A detailed in-depth evaluation of the operating system options can be found in Chapter A.3.6.

ADR-07: Secret Management

Name	Secret Management
Status	Accepted
Initial proposal date	23. October 2025
Last updated	11. December 2025

Table 26: ADR Secret Management

Context and problem statement

Software DW intend to deploy a new application in the cluster and are therefore required to store secrets securely in git.

Decision drivers

- Securely storing secrets in git is required in the Software DW scenario.

Considered options

- sealed-secrets
- **SOPS**

Decision outcome

In the context of selecting an ideal secret management tool, facing the need to securly store secrets in git, it was decided to use **sealed-secrets** and not **SOPS**, to achieve an easy to use way for developers to store their secrets, accepting that **SOPS** remains a viable option that doesn't need a controller in the cluster.

Consequences

- Good, sealed-secrets works for the application Software DW is deploying.

Confirmation

The choice of sealed-secrets is confirmed once it can be deployed automatically in the **ECP**.

More information

The secret management evaluation proved to be tricky because a clear definition and understanding of which tools fall into this category was initially lacking. This confusion led to multiple technically wrong versions. Consequently, the chapter Chapter A.4 was rewritten multiple times.

Update 06. November 2025

Decision by the project team to use sealed-secrets as the secret management stack component. Proper documentation postponed to a later point, focus currently on development.

Update 11. December 2025

Properly documented **ADR** and Chapter A.4.

ADR-08: Cloud Native Storage

Name	**Cloud Native Storage**
Status	Accepted
Initial proposal date	23. October 2025
Last updated	1. November 2025

Table 27: ADR Cloud Native Storage

Context and problem statement

What cloud native storage plugin is best suited to provide persistent storage accross different nodes?

Decision drivers

- Data needs to be available to multiple containers (shared storage).
- Minimization of the risk of data loss.

Considered options

- CubeFS
- Rook
- Longhorn

Decision outcome

In the context of choosing a cloud native storage plugin facing the need to provide persistent storage to containers, it was decided to use **Longhorn** and not CubeFS or Rook to achieve a good allround storage solution accepting that other tools like Rook may have a more complete feature set.

Consequences

- Good, because Longhorn provides all needed features in a single tool.

Confirmation

The use of Longhorn is confirmed when:
- The installation of the plugin in combination with the rest of the stack was validated.
- The solution works as intended.

More information

An in-depth evaluation of the cloud native storage options can be found in Chapter A.5.

ADR-09: Container Runtime

Name	Container Runtime
Status	Accepted
Initial proposal date	23. October 2025
Last updated	1. November 2025

Table 28: ADR Container Runtime

Context and problem statement

What container runtime fulfills the needs to run containers on an enterprise-grade platform?

Decision drivers

- Functionality in combination with chosen OS
- Is it suited for a container orchestration platform?

Considered options

- containerd
- CRI-O

Decision outcome

In the context of choosing an optimal container runtime, facing the need to run containers and fetch images, it was decided to proceed with **containerd** and not CRI-O, to achieve an easy installation and ensured compatibility with Talos, accepting that CRI-O may be slightly better suited for Kubernetes.

Consequences

- Good, because containerd comes preinstalled on Talos
- Bad, because CRI-O is created specifically for Kubernetes

Confirmation

The use of containerd is confirmed when:
- The container runtime doesn't pose any problems when deploying and running containers.

More information

An in-depth evaluation of the container runtime options can be found in Chapter A.6.

ADR-10: Cloud Native Networking

Name	**Cloud Native Networking**
Status	Accepted
Initial proposal date	23. October 2025
Last updated	1. November 2025

Table 29: ADR Cloud Native Networking

Context and problem statement

Which cloud native network plugin is the best for assigning IP adresses to containers, as well as implementing security and other networking-related features?

Decision drivers

- Fully supports the **CNI** standard.
- Are security features provided?
- Licensed for commercial use.

Considered options

- Cilium
- Calico
- Flannel

Decision outcome

In the context of choosing a cloud native network plugin facing the need of providing networking capabilities to the container platform it was decided to use **Cilium** and not Calico or Flannel to achieve a highly capable networking solution which is for commercial use accepting that it may have a steeper learning curve than other tools.

Consequences

- Good, because Cilium fulfills all requirements in terms of networking and security.
- Good, because all features are available without any license costs.

Confirmation

The use of Cilium is confirmed when:
- The installation in combination with the rest of the stack was successful.
- There are no major issues which prevent network connectivity between containers.

More information

An in-depth evaluation of cloud native network can be found in Chapter A.7.

ADR-11: Scheduling and Orchestration Technology

Name	**Scheduling and Orchestration Technology**
Status	Accepted
Initial proposal date	23. October 2025
Last updated	2. November 2025

Table 30: ADR Scheduling and Orchestration Technology

Context and problem statement

Which container scheduling and orchestration technology is best suited for our enterprise container platform?

Decision drivers

- The technology should be designed to manage containerized applications at scale.
- It should mature and widely adopted in the industry.

Considered options

- Kubernetes
- Nomad Enterprise

Decision outcome

In the context of selecting the optimal scheduling and orchestration technology, facing the need to manage containerized applications at scale, it was decided to use **Kubernetes** and not Nomad Enterprise to achieve a more widely adopted, flexible and mature solution, accepting its complexity and steeper learning curve.

Consequences

- Good, because Kubernetes fulfills all evaluation criteria as per Chapter A.8.
- Good, because Kubernetes as well as Talos work with etcd, simplifying integration.
- Bad, only in case of having an HashiCorp-centric infrastructure, where Nomad Enterprise would integrate better.

Confirmation

The use of Kubernetes is confirmed when:
- A viable configuration has been defined.
- All technologies used in our enterprise container platform work with Kubernetes.

More information

An in-depth evaluation of the scheduling and orchestration technologies can be found in Chapter A.8.

ADR-12: Observability and Analysis Stack

Name	**Observability and Analysis Stack**
Status	Accepted
Initial proposal date	23. October 2025
Last updated	4. November 2025

Table 31: ADR Observability and Analysis Stack

Context and problem statement

To prevent service disruptions and ensure optimal performance in the enterprise container platform, telemetry data must be observed and analyzed. This raises the question: which observability and analysis stack is best suited for the enterprise container platform?

Decision drivers

- Integration with Kubernetes and cloud-native environments.
- Free for non-commercial use.
- Simplicity in deployment and maintenance.
- Scalability options to handle increasing data volumes.
- Supports log and metric collection, storage, and visualization.
- Capability to handle distributed tracing data.

Considered options

- Visualization: Grafana, OpenSearch Dashboards
- Log aggregation: Fluent Bit, Fluentd, OpenTelemetry Collector, Alloy, Vector
- Log storage: OpenSearch Core, Grafana Loki
- Metrics collection: Prometheus Operator, OpenTelemetry Collector, Vector
- Metrics storage: Prometheus (Thanos), Grafana Mimir

Decision outcome

In the context of selecting the optimal observability and analysis stack, facing the need to collect, store, and visualize logs and metrics, it was decided to use a combination of **Grafana, OpenTelemetry Collector, Grafana Loki, and Prometheus** and against all others to achieve a well-integrated, cloud-native, and simple observability stack with fewer components, accepting the trade-offs in performance and maturity.

Consequences

- Good, because Grafana integrates well with the components and has rich visualization possibilities.
- Good, because OpenTelemetry Collector can be used to aggregate logs, metrics, and traces, reducing the overall stack complexity.
- Good, because Grafana Loki is designed for cloud-native environments and integrates well with Grafana.
- Good, because Prometheus is the de-facto standard for metrics collection in Kubernetes environments.

- Bad, because Vector, Prometheus Operator, Fluent Bit, and Fluentd are more mature log collectors with better performance.
- Bad, because the need of an proxy to restrict access to Grafana Loki and Prometheus.
- Bad, because the need of an s3 bucket storage to scale log and metric storage horizontally arises.

Confirmation

The use of the components is confirmed when:
- A viable configuration has been defined for the whole stack.
- The stack logs and metrics are visualized in the Visualization tool.

More information

A detailed in-depth evaluation of the Scheduling and Orchestration Technologies can be found in Chapter A.10.

ADR-13: Programming Language

Name	Programming Language
Status	Accepted
Initial proposal date	6. November 2025
Last updated	6. November 2025

Table 32: ADR Programming Language

Context and Problem Statement

The options of programming languages for a stand-alone application were discussed and two options considered:

- Python: The team has experience with Python, the syntax is easy with modules for almost all use cases.
- Go: There is little to no experience present with Go, but it has native integration of Kubernetes and Talos (both of which are also written in Go).

This was followed by a short **PoC**, where the talosctl and kubectl implementations of Go were tested against a local Talos cluster running in a docker container.

Decision Drivers

- Having a stand-alone application that runs on Windows, Linux and macOS.
- Facing the concern to implement cli tools like talosctl, kubectl or helm.
- Facing the concern to implement a GUI.

Considered Options

- Python
- Go

Decision Outcome

In the context of selecting the optimal programming language facing the need create a stand-alone application similar to a software installer it was decided to use Go and not Python to achieve a native implementation of aforementioned CLI tools accepting that our team has little to no experience with Go.

Consequences

- Good, Go has native packages for all required CLI tools.
- Neutral, the GUI selection is to be determined.
- Bad, steep learning curve due to lack of experience.

Confirmation

A proof of concept has been done to confirm that kubectl and talosctl work directly from Go code. This confirms the selection of the programming language.

More Information

Go GUI options were briefly looked at, at the moment of writing this **ADR**, these include wails, fyne.io and bubbletea.