Provisioning, configuration, and monitoring of single-board computer clusters

Arvid Agne
Abstract

Single-board computers as hardware for container orchestration have been a growing subject. Previous studies have investigated their potential of running production-grade technologies in various environments where low-resource, cheap, and flexible clusters may be of use. This report investigates the appliance of methods and processes prevalent in cluster, container orchestration, and cloud-native environments. The motivation being that if single-board computers are able to run clusters to a satisfactory degree, they should also be able to fulfill the methods and processes which permeate the same cloud-native technologies. Investigation of the subject will be conducted through the creation of different criteria for each method and process. They will then act as an evaluation basis for an experiment in which a single-board computer cluster will be built, provisioned, configured, and monitored. As a summary, the investigation has been successful, instilling more confidence in single-board computer clusters and their ability to implement cluster related methodologies and processes.

Keywords: single-board computer, container orchestration, provisioning, configuration, monitoring.
1 Introduction

This report is written as a degree project for a computer science bachelor’s degree at Linnaeus University. It is written as a continuation of subjects and courses taught in the program Utveckling och drift av mjukvarusystem (Development and operation of software systems). The program encompassed subjects such as container technologies, cloud environments, and continuous delivery of both infrastructure and software.

The goal of this report is to investigate how tools used for provisioning, configuring, and monitoring container orchestration clusters behave on low-resource commodity single-board computers. During the investigation, a Kubernetes cluster will be provisioned onto a cluster of Raspberry Pis. The provisioned cluster will then act as a basis for the proceeding objectives of the investigation. Each objective will be investigated by applying one chosen tool in that area to the single-board computer cluster. For example, in the case of investigating configuration, a tool like Ansible or SaltStack could be chosen as an investigation basis. Multiple tools will not be investigated or compared in the same category.

The produced artifact by the report and investigation will, if successful, be a reproducible ready to be provisioned configuration of a single-board computer cluster. Built with Kubernetes as a container orchestration tool, configured to be fully functional and continuously monitored.

This report grounds itself on the belief that easily provisioned, and shareable clusters could help the adoption rate of cloud-native technologies by providing small teams or single developers accessible small scale cluster environments for experimentation, educational, development, or demonstration purposes. However, for this to hold, the currently unexplored area in academic studies regarding single-board computer clusters meaning the provision, configuration, and monitoring steps need to be further investigated.

1.1 Background

Single-board computers (SBCs) often come at a relatively low cost and are commonly used for demonstration/development systems, educational systems, or embedded computer controllers. Studies have investigated the suitability of these single-board computers as building blocks for cluster environments. They have shown that SBCs are sufficiently powerful to run mainstream workloads such as cluster environments [1], [2].

Much of today’s methodologies regarding the delivery process of software relate to container and cluster technologies. Usually, involving the idea of repeatable, reliable, and reproducible artifacts from code commit to deployment in production. Throughout methodologies such as Continuous Delivery (CD) and Development and Operation (DevOps), considerable weight is put upon automation, Infrastructure as Code (IaC), Configuration Management (CM), and monitoring for provisioning and managing the environment in which applications will run [3], [4]. These methodologies ensure repeatable and reliable systems through confidence in automated processes kept under version control [3]. Many tools and systems have, therefore, been developed to solve these kinds of problems. Some examples of these kinds of tools with various areas of responsibility are Kubernetes, Terraform, Ansible, and Prometheus.

In a survey conducted by the Cloud Native Computing Foundation, the adoption rate of cloud-native technologies between 2017-2018 had increased in production environments by 200% and by 375% in evaluation environments [5]. Today cloud-native applications, systems, and tools are widely spread, and cloud providers like Google, Amazon, and Microsoft all
provide robust container orchestration environments for global production settings. Further cloud-native applications and infrastructure that are most prominently showcased by Netflix have shown that microservice architectures are an effective solution to large, globally distributed services.

The interest for clusters is, therefore, both present and growing. Furthermore, the interest has spread beyond the walls of the traditionally large data centers to be something accessible for niche and specialized environments. Not surprisingly, several academic studies have investigated the subject [1], [2].

1.2 Related work

The suitability of single-board computer clusters has previously been investigated in academic studies such as *Commodity single board computer clusters and their applications* [1] and *Next generation single board clusters* [2]. These have shown that SBCs are capable of running cluster environments but do not focus on the provisioning and configuration of them in an automated and repeatable manner.

*Commodity single board computer clusters and their applications* focuses on the real-world application of SBC clusters. It introduces the concept of clusters powered by commodity SBCs in various settings. The report investigates different logistical hardware configurations and deployments, the pros and cons of single-board computing hardware in a cluster environment, and its relative cost to other solutions. It concludes by remarking the concept of SBC clusters as good candidates for use cases such as [1]:

- Educational, enabling firsthand interaction with cluster technologies from hardware to software.
- Edge compute, small commodity clusters could help move computing and storage closer to the end-user, for example, in rural and remote areas.
- Expendable compute, small commodity clusters could place efficient, cheap, and, therefore, expandable clusters in high-risk environments, such as volcanoes.
- Resource constrained compute, the relatively low resource consumption of single-board computers, and their small size, allow for clusters in constrained environments.
- Next-Generation data centers, large scale server solutions based on the ARM platform are being investigated and developed as an alternative to current architectures. Single-board computers, being ARM-based, could provide insight for these future data centers.
- Portable clusters, single-board computer clusters suitability for Edge, Expendable and Resource constrained compute, allows for small and resource-light portable clusters.

*Next Generation single board clusters* is a smaller investigation and demonstration of an SBC cluster by an implementation called Pi Stack. Its goal is to present accessible solutions for hardware, power distribution, and cluster maintenance. It highlights some of the inherent risks of SBC hardware contra traditional computers. Storage on SD-cards presents a more easily corrupted file system, and a fragmented OS selection results in unique security and performance considerations. Although the investigation highlights some detriments SBC clusters, it concludes its usefulness in small scale applications [2].

A subject regarding single-board computers and clusters which remains after the investigation of the studies above is the application of cluster deployment and maintenance
methodologies. These have been evolving alongside the same cluster technologies, which now single-board computers are capable of running.

As said while single-board computer clusters are proven candidates for cloud-native technologies, no academic study has explicitly investigated the combination of single-board computer clusters and the principles and methodologies applied to their production counterparts. As cloud-native technologies are so closely coupled to these methodologies, it would be incredibly beneficial if these clusters also could realize and adopt them wholly.

1.3 Problem formulation

Now when modern commodity single-board computers have been established as good candidates for fully functional container orchestration clusters well suited for development, demonstration, and educational purposes [1], [2]. They should be investigated and looked upon not only from a purely technical perspective but also on how well they can facilitate the processes closely coupled to them.

The development and operation of cloud-native infrastructure and applications have resulted in several practices and methodologies. These adopt the idea of entirely reproducible resources and artifacts by placing all configuration, management, and deployment under version control. Principles like Infrastructure as Code and Configuration management are the basis of Continuous methodologies (i.e., Integration, Delivery), which in turn are crucial to cloud-native deployment and development [3].

For a further explanation of the underlying methodologies and processes mentioned above and what kind of problems they aim to solve, see Chapter 2.

A summary of the high-level goals of the combined methodologies are:

- Everything which regards infrastructure and configuration should be version-controllable.
- Infrastructure and configuration should be fully buildable from scratch, only using files kept under version control.
- The process of building and configuration should be an automated process.

Therefore, the report aims to investigate the provisioning, configuration, and monitoring of single-board computer clusters with a focus on the main principles of Infrastructure as Code and Configuration Management. Examining if single-board computer clusters can be treated as equals to their production counterparts when commonly used cloud-native tools and principles are applied to them.

To further explain the investigation which will take place, three guiding research questions are presented below.

- **Question 1:** Is single-board computer hardware able to be provisioned to run cluster technologies according to the principles of *Infrastructure as Code*?
- **Question 2:** Are single-board computer clusters able to be configured and applications deployed unto them according to the principles of *Configuration Management*?
- **Question 3:** Are single-board computer clusters able to be monitored to such extent as to provide useful metrics for the feedback loop of *Continuous Methodologies*?
1.4 Objectives

The investigation will be divided into three logical parts: provisioning, configuration, and monitoring and presented as three objectives. The objectives below present a brief overview of the goals and examined aspects for the following report. For an introduction to the underlying concepts and methodologies which motivate the report's work and a more thorough description of each objective, see Chapter 2.

- **Objective 1, Provisioning.** Addresses research question 1 by investigating the creation/providing of resources and infrastructure onto the underlying hardware in an automated manner.
- **Objective 2, Configuration.** Addresses research question 2 by investigating the required configuration for the cluster and application deployments in an automated manner.
- **Objective 3, Monitoring.** Addresses research question 3 by investigation of the collection, consolidation, and presentation of data derived from the cluster environment.

1.5 Scope/Limitation

The investigation of all three objectives: provisioning, configuration, and monitoring will be restricted to a single tool or toolstack. For example, there will not be any comparison between different provisioning tools during the provisioning objective.

Further, only one kind of single-board computer version will be used to host the cluster, and the number of nodes will remain constant during the investigation. There will not be any kind of comparison between different single-board computer setups, versions, or hardware specifications.

1.6 Target group

The target group for the investigation is developers and operational personnel who are interested in incorporating Continuous methodologies and related tools into low resource educational, demonstrative, or developing clusters.

1.7 Outline

The outline for the rest of the report follows the following structure. Chapter 2 contains an introduction to concepts and principles which build the foundation for the investigation. Chapter 3 describes and details the chosen method to investigate the problem objectives. Chapter 4 contains the implementation and evaluation details of the investigation objectives.

Chapter 5 presents the results of the investigation and analyzes them using the evaluation criteria detailed under Chapter 4. After the analysis comes Chapter 6, a discussion on investigation findings and how well they fulfill the problem formulation. The chapter also contains comparisons of the result to other similar investigations.

Finally, Chapter 7 summarizes what has been shown under the project, if the results are relevant for the problem domain and how generally applicable the result is for future investigations. There will also be a discussion on what could have been done differently to achieve better results.
2 Continuous methodologies

2.1 Processes

*Continuous Delivery* written by J. Humble and D. Farley identifies the release process of software as a common point of failure in software development [3]. It is often seen as a high-risk endeavor. It is an endeavor involving months of work and preparations that culminate on the release date to an ‘all on the line’ make or break situation. The authors argue that this traditional way of handling release cycles fails to successfully deliver consistent software releases and instead only contributes to fear, resentment, and late work hours [3].

One solution to avoid these kinds of problems and to achieve a robust delivery pipeline are the different but closely related Continuous methodologies. Continuous Delivery is an engineering approach that focuses on how quickly and reliably any kind of value created can be available for production. In this context, the value can be defined as anything benefiting the overall quality of development, or product, for example, code refactoring, bug fixes, or new system features. Three guiding principles must be implemented into the release process to break down the traditionally massive releases of software into smaller digestible parts. Three guiding principles must be implemented into the release process:

- **Repeatable**, guaranteed repeatability means that automation of the process is required.
- **Reliable**, a reliable process needs to be well-tested. As it is repeatable, it should have been executed hundreds of times, proving its reliability.
- **Predictable**, with a repeatable and reliable process, the output it produces will be predictable. Which, in turn, should instill confidence in the product of the automated process.

To achieve a release process that embodies the three guiding principles, version-controlled automation should be used anywhere possible in the process. This leads to the foundation of any Continuous process, the configuration management of projects. *Continuous Delivery* puts the terms configuration management and version control as synonyms and defines it as the system in which all changes within the development effort are kept. Importantly the scope of the configuration management encompasses absolutely everything that is code, dependencies, software configuration, and environment [3].

Sound configuration management is defined by three key points by *Continuous Delivery*:

- **Encompassing**, the current production system should be entirely reproducible from what is kept under version control.
- **Revert-able**, any change added to any element contained in the version-control system should be easily revertible to an older version.
- **Consistent**, the configuration should always deploy identically configured systems.

2.2 Infrastructure and tools

With the growing adoption of cloud technologies [5], the same principles which drive continuous delivery of software can be applied to the infrastructure. This means that whole production environments encompassing web servers, databases, and load balancers can be automatically created from scratch.
They are bringing the benefits of repeatability, reliability, and predictability not only to software but to the underlying infrastructure as well. Nevertheless, there are also challenges involved in these kinds of infrastructures, challenges similar to those of software delivery. The infrastructures’ changeable nature can if mismanaged, lead to unmanageable and unrecoverable states. Resulting in considerable costs in time and resources through unnecessary work in recovering and managing the infrastructure [6].

The common problems and anti-patterns often identified in dynamic infrastructures are:

- **Configuration drift**, the configuration of servers starts to drift apart over time, which results in unmanaged server states.
- **Snowflake servers**, servers which have drifted so far from their original configuration that they are no longer replicable.
- **Fragile infrastructure**, infrastructure which, if disrupted, i.e., failure of a snowflake server requires significant time and effort to fix.

To combat this and put infrastructure under configuration management, one requirement stood out, the ability to completely version-control infrastructure. This resulted in Infrastructure as Code, a process which describes all parts of an environment through definition files. Definition files enable a provisioning tool to read the definition file(s) and automatically and reliably produce complete production-ready environments.

In *Infrastructure as Code: Managing Servers in the Cloud* by Keith Morris, the distinctive feature of effective teams and tools is their ability to handle changes and new requirements [6]. High effectiveness stems from the ability to break down requirements into small, low-risk low-impact pieces, which then can be deployed through a fast delivery pipeline.

Today there is a wide variety of different provisioning and configuration tools available on the market. Examples of such can be Puppet, SaltStack, Ansible, Terraform, and many more. It is not always a clear line where provisioning stops and configuration start, or if a tool is a provisioning tool or a configuration tool. Often a tool realizes both or parts of the other. However, in essence, all tools have four common traits and goals:

- **Source controlled infrastructure** for providing audit trails and rollbacks.
- **Testable infrastructure**, written code can be unit, functional, and integration tested.
- **Self-documented infrastructure**, through written configuration files with readable syntax.
- **Improved collaboration** to support the idea of collaboration between developers and operators.

An essential part of the cloud-native environment is the container orchestration tool, the facilitator of portability, scalability, and redundancy. In today’s cloud-native environment, Kubernetes is by a wide margin the most popular choice of container orchestration technology, spanning the most dominant cloud providers [7]. Kubernetes is the clear choice of cluster technology moving forward with the report and investigation as no alternative has a comparable adoption rate yet.

### 2.3 DevOps

DevOps is not a clearly defined term but is classified as a collection of different processes/practices that try to make the development and operation of software as a cohesive
experience as possible. Tying the two together (Dev and Ops) to reduce friction in the passing of software from release candidate to running in a production environment. Importantly DevOps is not a tool or technology but a set of disciplines regarding documentation, quality assurance, project management, user experience, and at last configuration management.

The book *The DevOps Handbook* written by some of the most influential people behind DevOps Gene Kim, Jez Humble, Patrick Debois and John Willis divide the processes into the fulfillment of three so-called flows [8]:

- **The Principle of Flow**, fast flow from idea/code to production.
- **The Principle of Feedback**, fast information flow from production to code.
- **The Principle of Continuous Improvement**, building organizational feedback loops to keep things moving

The principles of Flow and Feedback goes hand in hand with Continuous methodologies and configuration management. Where configuration management can be described as the embodiment of the two principles in their real-world application. Figure fig. 2.1 depicts a simplified view of the DevOps process, which relates to the investigation’s scope. It shows the circular process which drives a project forward, supported by the ideas of continuous thinking, version control, monitoring, and more.

Collaborative development using version control builds and tests the application or service to be. The application is continuously integrated into the main branch of development, and artifact management allows for easy revert-ability. Continuous Deployment is possible from working main branch releases to the cluster, which runs the applications in their portable container runtime. From cluster and containers metrics can be derived, and if required, alerted on. Metrics form the usage analytics, which ties the process together, allowing the next iteration of the application to build upon a solid foundation.
Figure 2.1: Overview of DevOps, model derived and simplified for clarity from IBM Cloud Architecture [9]
2.4 Single-board computer clusters

With the widespread adoption rate of DevOps and Continuous practices together with the growing adoption rate of cloud-native technologies [5]. Development and operation on production-like clusters as early as possible in the delivery pipeline would be according to and beneficial to the principles in question. Minimal changes throughout the development, test, and production environments are essential to keep a delivery process from code check-in to deployment as lean, error-free, and efficient as possible [3].

Being able to replicate production-like container clusters at a small scale while maintaining a local hardware deployment provides developers and operations with a fully controlled cluster environment. These could potentially be a great benefit to the delivery process but could also provide developers and operation with educational opportunities. As the cloud environment is so dominated by the processes and methodologies mentioned above, it is important that they are also applicable to single-board computer clusters. Which would allow similar educational and demonstrative opportunities for the methodologies as are available for the technologies.

All methodologies and technologies presented in this chapter rely on parts of the continuous and encompassing process, which starts at infrastructure and ends in software running in production. Provisioning, configuration, and monitoring are all part of this overarching process, so even though the investigation is limited in scope and mainly focuses on configuration management, it still supports the more encompassing methodologies of DevOps and Continuous Delivery.

The investigation of provisioning, configuration, and monitoring of SBC clusters will involve a number of challenges. Mainly which requirement needs to be fulfilled to achieve a successful result when investigating, for example, provisioning. Does provisioning only involve a working cluster environment, or should idempotent execution also be a requirement?

While there are several good practices and requirements for configuration management and Continuous methodologies, there is no single checklist or definition for complete fulfillment. Several examples of real-world applications of single-board computer clusters and their use-cases are presented in the study of *Commodity single board computer clusters and their applications* [1]. An excerpt from the study includes educational clusters constructed by the University of Southampton and the University of Edinburgh. Further, the Free University of Bolzano has taken single-board computer clusters a step further by deploying a 300 node cluster for education, research, and deployment purposes of clusters for developing countries [10]. Single-board computers have, as presented above, a broad range of applicable use-cases. For the report’s investigation, the focus will lie on problems and goals applicable and usable in most, if not all, use cases. Therefore by studying the requirements and good practices presented in literature, more specifically, *Continuous Delivery* [3] and *Infrastructure as Code: Managing Servers in the Cloud* [6] several goals, motivations, and criteria have been constructed for each investigation objective.

**Provisioning**

**Goal:** Automatically initialize a Kubernetes cluster on a given set of single-board computers with a chosen provisioning tool.

**Motivation:** Provisioning of infrastructure is an important part of easily replicated and repeated environments. A requirement for any robust infrastructure solution [6].
Examined aspects:

- Creation of an inventory/collection with the single-board computers as nodes.
- Connection to the single-board computers as remote nodes.
- Execution of provisioning tool modules on single-board computer nodes.
- Provisioning of Kubernetes resources on single-board computer nodes.

Configuration

Goal: Automatically configure a set of single-board computer nodes to run distributed workloads as a cluster with a configuration tool.

Motivation: Repeatable and idempotent configuration of resources is the foundation of configuration management [3].

Examined aspects:

- Continuous management of a single-board computer node inventory/collection.
- Continuous execution of configuration modules on single-board computer clusters.
- Continuous deployment of Kubernetes deployments, services, and jobs on a single-board computer cluster.

Monitoring

Goal: Examine the collection and presentations of metrics derived from the single-board computer cluster.

Motivation: Monitoring is an integral part of continuous methodologies, providing metrics and data which drive both development and infrastructure forward [6].

Examined aspects:

- Collection of single-board computer Kubernetes cluster node and pod metrics and statistics.
- Storage of single-board computer Kubernetes cluster node and pod metrics and statistics.
- Presentation of single-board computer Kubernetes cluster node and pod metrics and statistics.

Terminology

From here on out, commonly repeated terms like single-board computers and Infrastructure as Code will be shortened. Please refer to the terminology chapter below.

Single-board computer (SBC) Is a complete and fully functional computer built on a single circuit board, including microprocessor, memory, and I/O.

Infrastructure as Code (IaC) The process of managing and provisioning computer data centers through machine-readable definition files.

Configuration Management (CM) The process of systematically handling changes to a system ensuring its integrity over time.

Continuous methodologies (CI, CD) Engineering approaches that focus on how quickly and reliably any kind of value created by a developer can be readily available for production. Relies on build and delivery systems that are repeatable, reliable, and predictable [3].
3 Method

With the goal and examined aspects detailed for provisioning, configuration, and monitoring, an appropriate method for the investigation can be chosen. As described under Objectives, the problem domain will be divided into three parts/objectives. The chosen method used to investigate each of them will be a case study, an up-close detailed examination of each objective. Each objective will be investigated in the logical order of provision, configuration, and lastly, monitoring. If the opportunity presents itself, a revisit to an earlier goal for further investigation and refinement may occur. However, several iterations on the objectives are not planned for. Instead, the study of each objective is satisfied by a single implementation. See fig. 3.2 for an overview of the process, which will guide the investigation.

As fig. 3.2 displays, the investigation process is incremental, and each objective is dependant on the preceding. Therefore a small evaluation is conducted after the two first objectives. These evaluations’ main focus is to ensure the basic functionality required for the next objective to be investigated. With that said evaluation of Objective 2, Configuration could result in that monitoring is unfeasible or heavily restricted. This would not halt the investigation of Objective 3 Monitoring but would change the focus to what is achievable within its set boundaries. As the report aims to conclude a summary of the three objectives combined successfulness, that is the foremost concern of the investigation. Derivations and caveats related to each objective could still result in what is deemed a successful result.
The evaluation of the objectives will be somewhat open-ended as the investigation includes some exploratory aspects regarding tool appliances to SBC clusters. Results, implementation, and evaluation may, therefore, be expanded upon if an unexpected facet of SBC clusters or applied tools is discovered.

By both ideal and design, the methodologies and tools used in IaC and CM are driven by self-documenting code, reproducibility, and shareable output. This means that the implementation naturally will produce artifacts that are readable and testable by an external/outer part. Conformance to the expected standards of example naming and structure in IaC and CM will be of high priority during the implementation of the investigation. The machine-readable definition files will be provided as part of the presentation of the implementation and result chapters.

3.1 Reliability and Validity

The reliability and validity of the case study’s result can, to a certain extent, be promised as reproducible artifacts are one of the essential parts of both provisioning and configuration. An identical hardware configuration, together with the correct version or minor version changes of software and APIs, should provide reasonable grounds for validation of the implementation. Moreover, some implementation validity is gained from the machine-readable definition files, as they can be parsed by linters and used for the actual provisioning and configuration of the implementation itself.

What could affect the validity of the results is the natural configuration drift that is inevitable as time passes and software is updated. The current version of tools and dependencies are only guaranteed to be relevant for a short while. Although the results may act as useful guidelines for future versions of the various software involved, again, nothing can be guaranteed. If the provisioning, configuration, and monitoring of the cluster are deemed working with current software versions, it is unlikely that future versions will recess in functionality. For example, the possibility for provisioning will most likely remain valid for the future, while the actual definition of the said provision is destined to change.
4 Experiment

The method detailed in Chapter 3 describes how the general investigation of the clusters provisioning, configuration, and monitoring will take place. However, before the investigation can be started, an experiment environment needs to be setup. Moreover, there has to be a detailing of the characteristics and limitations of the environment. Then with the experiment environment complete and understood, a scenario for each investigation objective can be created. The goal of these scenarios is to provide a way of achieving the goal and to cover the evaluation aspects for each objective detailed in Chapter 2.4.

4.1 Design

The following section describes the implementation specifications of the cluster in terms of hardware and software. Hardware and software specifications simply describe the hardware specifications such as CPU cores, memory, and storage. Followed by a quick overview of the tools and packages used during the implementation, with software versions and short descriptions. Characteristics and limitations then describes the limitations and expectations the hardware configuration enables and imposes, i.e., what is possible to replicate and what-not. Characteristics and limitations also describes which of the design decisions effects the provisioning, configuration, and monitoring of the cluster.

Figure fig. 4.3 provides a visual representation of the cluster topology, which the evaluation and implementation will produce. The topology depicts the master node running the Kubernetes master plane and the two worker nodes running the distributed workload of pods, deployments, and services.

![Cluster topology](image)

Figure 4.3: Cluster topology

Hardware and software specifications

The model of single-board computer chosen as hardware for the Kubernetes cluster is the Raspberry Pi version 4 2GB.

Hardware specifications:
- Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 2GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 32 GB Micro-SD

Three of these single-board computers will run Ubuntu 19.10 server and together form the cluster. One of the three single-board computers acting as a control-plane master and the two remaining as worker nodes.

The table below tbl. 4.1.1 presents the versions of tools and packages used during the investigation. Software is always changing and updated, and the implementation is, therefore, heavily dependent on the current limitations of said packages and tools. Problems encountered during the investigation may be solved in newer versions, or different problems may be present.

<table>
<thead>
<tr>
<th>Package/Tool</th>
<th>Description</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ansible</td>
<td>IT automation engine that automates cloud provisioning, configuration management, application deployment.</td>
<td>2.9.6</td>
</tr>
<tr>
<td>Helm</td>
<td>A tool that streamlines installing and managing Kubernetes applications</td>
<td>3.1.2</td>
</tr>
<tr>
<td>kubeadm</td>
<td>A tool built to provide best-practice “fast paths” for creating Kubernetes clusters</td>
<td>1.17.4-00</td>
</tr>
<tr>
<td>kubectl</td>
<td>The Kubernetes command-line tool allows execution of commands against Kubernetes clusters.</td>
<td>1.17.4-00</td>
</tr>
<tr>
<td>kubelet</td>
<td>The primary “node agent” that runs on each node in a Kubernetes cluster.</td>
<td>1.17.4-00</td>
</tr>
<tr>
<td>docker-ce</td>
<td>Free and open-source containerization platform.</td>
<td>5:19.03.8-3-0</td>
</tr>
<tr>
<td>docker-ce-cli</td>
<td>Command-line interface for interaction with the Docker daemon.</td>
<td>5:19.03.8-3-0</td>
</tr>
<tr>
<td>containerd.io</td>
<td>An industry-standard container runtime with an emphasis on simplicity, robustness, and portability.</td>
<td>1.2.13-1</td>
</tr>
<tr>
<td>Flannel</td>
<td>Kubernetes CNI, a virtual network that gives a subnet to each host for use with container runtimes.</td>
<td>0.11.0</td>
</tr>
<tr>
<td>Prometheus</td>
<td>An open-source monitoring system with a dimensional data model, flexible query language, efficient time-series database.</td>
<td>2.16.0</td>
</tr>
<tr>
<td>Grafana</td>
<td>Open source analytics &amp; monitoring solution</td>
<td>6.7.1</td>
</tr>
</tbody>
</table>

### 4.2 Characteristics and limitations

The constructed experiment environment, mainly the cluster architecture, has specific characteristics and limitations which affect what is possible to achieve in it. Tools and software could also limit the experiment environment, but none of the tools explicitly states limitations
for ARM platforms even though such could exist. It is, therefore, impossible to predict these limitations or unexpected facets in the design stage of the environment. The following characteristics are therefore focused on the architectural side of the cluster.

**Capacity**

The total compute capacity of the cluster is the sum of all constituent nodes. In this case, the total capacity is eight cores of processing power and a total of 4GB memory. The master node does not contribute to the cluster capacity as the count and capacity of masters are considered a separate concern in Kubernetes cluster architecture [11].

**Node size**

As mentioned above, the master node size is one, and the worker node size two. One-master-two-workers setup is the minimal configuration that still provides some hardware redundancy for a cluster architecture. A caveat is that a second master node is required to replicate a High Availability (HA) environment, see High availability.

**Blast radius**

Two worker nodes are a limitation of the cluster, which means that a failing node results in a loss of 50% of available pods. The blast radius is therefore quite high, and the inherent risk of the low node count is a capacity shortage if an overbearing pod migration is required. This should always be in consideration when running demanding cluster workloads. In ordinary cases, when there is an apt leftover capacity for migration, Kubernetes automatic rescheduling should be triggered. Exploratory and demonstrative handling of failed nodes should, therefore, be accessible in the SBC cluster environment.

**High availability**

High Availability (HA) clusters are clusters which also provides redundancy for its master nodes. Further, HA clusters should span multiple zones, adding the need for another cluster for a correct implementation. The bottom line is that the current implementation of the cluster does not support interaction with a HA environment. However, the setup does not limit future expansion to a HA environment by virtual, cloud, or local means.

### 4.3 Evaluation scenarios

Given the experiment environment, scenarios for each objective of the investigation can be constructed. The goal and motivation for each of these scenarios are as detailed under Chapter 2.4 with the actual implementation. Showcasing the evaluation of each objectives examination criteria. Chapter 2.4 answered the question of what and why. The scenarios will answer how. To keep the implementation in line with Chapter 2.4 ’s goals, it needs some direct requirements fulfilled. First of all, the node count of different clusters may vary from a couple to a hundred [1]. The implementation of provisioning, configuration, and monitoring will, therefore, always be able to act on an unknown number of given nodes. Conveniently, the ideals of IaC and CM and the tools used to realize them works with unknown node counts not only in mid but
from the ground-up. A correct implementation, given a configuration or provisioning tool, will be able to act on as many nodes required given apt resources on the master machine. The translation of evaluation aspects to evaluation criteria, meaning, how each aspect will be implemented during the investigation, is detailed in Chapter 4.4 below.

**Provisioning scenario**
Provisioning in the experiment environment will consist of the construction and application of Ansible playbooks, tasks, and roles. Ansible as a tool will work through an inventory which exposes the hosts to be nodes of the cluster. Ansible is also an extensive provisioning tool allowing for multiple facets of the process to be examined. The scenario will begin with the bare experiment environment, i.e., the three Raspberry Pis in an inventory. Then provided the evaluation criteria for provisioning the implementation will be guided by them to cover each criterion while, in the end, produce a Kubernetes cluster ready for configuration.

**Configuration scenario**
The configuration scenario, also guided by its evaluation criteria will be based upon the earlier provisioning scenario. Configuration will examine the criteria by joining together the provisioned nodes to a cluster and then deploy applications and services to it. Ansible will also be used as the tool of choice for configuration, as it handles configuration of environments much like it does provisioning. This results in that the implementation of configuration will consist mostly of Ansible playbooks, tasks, and roles. Deployment of cluster application and services are also involved. Definition files for Helm and Kubernetes will, therefore, also play a part in the scenario. The scenario should result in the experiment environment running a set of monitoring services as a distributed workload on the Kubernetes cluster.

**Monitoring scenarios**
With monitoring services configured and running in the cluster, their data and presentation should now be parsable. The monitoring scenario is much more analysis dependant than the earlier scenarios and will most likely require more discussion if certain evaluation criteria are achieved or not. This scenario consists of two monitoring solutions and will be divided into two separate parts. The first scenario is consisting of an evaluation of the Kubernetes Dashboard (Web UI) service and deployment. And the second evaluation of a full monitoring stack consisting of data exporters, Prometheus, and Grafana.

The scenario of monitoring through the Kubernetes Dashboard will be focused on the deployment in question. As the deployment is limited in scope, mostly basic functionality will be evaluated against the evaluation criteria. Because of its limited functionality, not all criteria may be relevant.

The full monitoring stack is much more complex and will, therefore, allow for a more in-depth evaluation of monitoring according to the criteria. The complexity of both tools and their functionality will impose certain limitations; everything the tools have to offer in terms of monitoring potential can not be examined. Therefore the scenario will focus on the stream of data through the whole monitoring stack. Meaning if the chain of gathering, consolidation, and presentation is working for a sample of metrics, certain assumptions could be made of similar untested metrics.
Prometheus and Grafana are two widespread monitoring tools, usually combined in monitoring stacks as Grafana includes built-in support for Prometheus. The motivation for the choice of Prometheus and Grafana is the same as the choice of Ansible and Kubernetes. Both fulfill the need to achieve the investigations goal, and the tools are familiar to the conductor of the experiment.

4.4 Preparation

Preparation for the experiment is divided into two parts, one preparing tools and dependencies for implementation and the second creating the evaluation criteria, which will guide both implementation and analysis of the results.

Environment

Ansible
As Ansible will be used for both provisioning and configuration, it will play an essential part in the investigation. Ansible requires a master host, which runs its playbooks, as it is an agentless tool only an installation is required on the master. Ansible is available on all OS platforms, but as the master used in this investigation runs an installation of Arch Linux, it is installed through the Arch community repository.

An important note for some Linux distributions and potentially other platforms is that the Ansible’s k8s module requires an installation of the python dependency OpenShift, available through example the python package manager pip.

Hardware installation
Hardware installation is not the focus of the report and has previously been subject to investigation in studies [1], [2]. The description of the preparation of hardware installation will, therefore, be somewhat brief.

Raspberry Pis are given an SD card as storage with an Ubuntu 19.10 Server installation. They are then connected to an unmanaged network switch, which in turn connects to a router for DCHP, gateway, and IP allocation. The clusters hardware installation is now ready for the investigation, beginning with Ansible’s initial provisioning.

Criteria

Evaluation tools
These are the criteria that the investigation will base itself upon for implementation and analysis. For a presentation of the result of the evaluation see tibs. 5.1, 5.2, 5.3 under Chapter 5. The requirements have been derived from studying Continuous Delivery [3] and Infrastructure as Code: Managing Servers in the Cloud [6] which resulted in the evaluation goals and aspects of Chapter 2.4. Combined with process facets provided by the functionality of Ansible as a provisioning and configuration tool. Some criteria will carry over between the three objectives. These are the basic requirements for tools and processes regarding IaC, CM, and CI/CD; these include:
Table 4.4.2: Evaluation criteria for general investigation.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idempotent execution of processes.</td>
<td>Repeated execution of modules without side effects</td>
</tr>
<tr>
<td>Automated processes.</td>
<td>Full execution of modules without interaction.</td>
</tr>
</tbody>
</table>

Provisioning
Provisioning is the creation of resources and infrastructure in an existing space. In an automated manner, go from three bare Raspberry Pis to a running Kubernetes cluster. To be able to provision the cluster unto the hardware, some basic requirements need to be fulfilled and have been divided into four parts:

Table 4.4.3: Evaluation criteria for Objective 1, Provisioning.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation of an inventory/collection with the single-board computers as nodes.</td>
<td>Automatic execution of low-level modules to several hosts.</td>
</tr>
<tr>
<td>Connection to the single-board computers as remote nodes.</td>
<td>Automated remote access for provisioning tool to hosts.</td>
</tr>
<tr>
<td>Execution of provisioning tool modules on the single-board computer nodes.</td>
<td>Provisioning of OS-level configuration and cluster dependencies.</td>
</tr>
<tr>
<td>Automatic provisioning of Kubernetes resources on the single-board computer nodes.</td>
<td>Automatic provisioning of an initialized Kubernetes cluster.</td>
</tr>
</tbody>
</table>

Configuration
Configuration is what ties the provisioned infrastructure and nodes together. Master and worker nodes are joined together to form the actually distributed cluster. Furthermore, unto the cluster, the tools and services required for monitoring are deployed. After the configuration step, the cluster should not only be running, and it should have several deployments and workloads running as well. In terms of a cluster, it should now be ‘complete,’ i.e., fully capable of hosting web servers, load balancers, and more. The criteria for a functional and automated configuration of the Kubernetes cluster is divided into three parts.
Table 4.4.4: Evaluation criteria for Objective 2, Configuration.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous management of a single-board computer node inventory/collection.</td>
<td>Execution of configuration modules on the provisioned cluster.</td>
</tr>
<tr>
<td>Repeated execution of configuration modules on the single-board computer cluster.</td>
<td>Repeated configuration and deployment of modules on/to the cluster.</td>
</tr>
<tr>
<td>Continuous deployment of Kubernetes deployments, services, and jobs to the single-board computer cluster.</td>
<td>Execution of Kubernetes resources to the cluster.</td>
</tr>
</tbody>
</table>

**Monitoring**
In the last objective of the investigation, the collection of node, pod, and service metrics are in focus. Is it possible to see the memory usage of all nodes? Can the Kubernetes API be queried for its version? And more similar questions. Monitoring the SBC based Kubernetes cluster is divided into three parts.

Table 4.4.5: Evaluation criteria for Objective 3, Monitoring.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment.</td>
<td>Analysis of monitoring tool(s) output.</td>
</tr>
<tr>
<td>Storage of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment.</td>
<td>Analysis of monitoring tool(s) storage convention.</td>
</tr>
<tr>
<td>Presentation of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment.</td>
<td>Analysis of monitoring dashboard deployment and functionality.</td>
</tr>
</tbody>
</table>

### 4.5 Implementation
The evaluation criteria and their process of verification allow for the implementation of the three investigation objectives to begin. Each implementation is presented with an overview of the process followed by its procedure.

**Provisioning**

**Overview**
The goal of the provisioning evaluation is to have the three Raspberry Pis ready for attachment to a common cluster, which is then able to run a distributed workload. The provisioning will
be automated and defined with the tool Ansible. The evaluation will be guided by three loosely defined categories; these are:

- **OS Level**, what is the required provisioning for running a Kubernetes cluster and its dependencies on an SBC.
- **Dependencies**, what is the required provisioning for the dependencies of the Kubernetes cluster.
- **Kubernetes**, what is the required provisioning for the initialization of the Kubernetes cluster.

After said categories have been evaluated and implemented, there should be one host provisioned as a master node and the two remaining hosts provisioned as workers.

**Procedure**

The tool chosen for the implementation of the provisioning objective is Ansible. Ansible is an agentless provisioning and configuration tool which connects to nodes by the SSH protocol. In terms of accessibility, both of these features dramatically reduce any pre-provisioning requirements. As the chosen operating system Ubuntu 19.10 Server comes with SSH functionality built-in, only a master machine with an Ansible installation is required to start provisioning. Ansible’s definition files are divided into playbooks and roles, both written in YAML. The provisioning of the SBCs will be loosely divided into three steps or layers mentioned under **Overview**.

**OS level**

Out-of-the-box the Raspberry Pi 4 2GB with Ubuntu 19.10 Server as the operating system comes with a varied selection of pre-installed packages. As mentioned above, SSH connectivity is the only dependency for provisioning by an Ansible master. The types of host inventories which Ansible is able to provision to, both dynamically loaded and statically typed alternatives are supported. A static inventory will be used during this investigation as the node count is low and manageable. Frequent node additions and subtractions will be unlikely in small educational, demonstrative, or developing clusters. As the node count is limited by physical hardware, a dynamic inventory solution would be somewhat redundant. The choice between a static or dynamic inventory should not impact the result of the provisioning as it is a more tool related issue than a cluster related.

Static IP-addresses are created and allocated to each Raspberry Pi, and the static inventory can be implemented. Named by convention [12] `production.yml`, the inventory contains the three hosts divided into two roles: masters and workers. A complete architecture playbook (called `cluster.yml`) is created, which in turn imports two sub-playbooks called `masters` and `workers`.

There are some specific OS-level configurations that have to be done in order for a Kubernetes cluster to be fully operational. These requirements are:

- Swap space and paging need to be disabled.
- Kernel module Control groups (cgroups) must be active.
- Iptables must use its legacy drivers and not rely on nftables.
• Package repositories must be available over HTTPS.
• Every host needs to have a unique hostname.

Each of these requirements needs to be handled in one or more tasks in the implementation. There are also some optional steps which will be implemented, these are:

• SSH password authentication will be disabled.

Dependencies

The only binary dependency for Kubernetes is a Container Runtime Interface (CRI). There are some options to choose from, but Docker is the most commonly used CRI for Kubernetes and is, therefore, a prime candidate.

Docker comes with its own set of requirements:

• Access to the Docker repository.
• A running Docker daemon.
• A running Docker service.

The actual installation of the Docker packages is handled by the OS package manager. These results in the addition of three packages:

• docker-ce
• docker-ce-cli
• containerd.io

Kubernetes

To access the Kubernetes binaries, the external Kubernetes repository needs to be accessed. The installation of Kubernetes related packages results in the addition of three new packages:

• kubelet
• kubeadm
• kubecli

The provisioning needs will differ from master and worker nodes. Foremost, the master nodes are required to have a Container Network Interface (CNI) deployment running. Without a CNI, the cluster would be unable to operate correctly as no internal network traffic could be sent or received.

The Container Network Interface (CNI) chosen for the Kubernetes cluster is Flannel developed by CoreOS. There is a wide variety of CNIs available for Kubernetes. In terms of the required setup for a working deployment, Flannel is quite minimal. If bleeding-edge performance was of great importance, the choice of CNI could be a lengthy process. Choosing Flannel is part of the exploratory process of the implementation, as mentioned under Method. Again provisioning of any other CNI would mean different configuration, but the provisioning itself would be somewhat similar.

The requirement for a functional Flannel deployment is an initialized Kubernetes cluster with a Classless Inter-Domain Routing (CIDR) in the range of 10.244.0.0/16 for the
pod network. In terms of provisioning contra configuration, the CIDR specification could fall under both categories, but as it is part of cluster initialization its placed under Provisioning.

After implementing the three steps **OS level, Dependencies** and **Kubernetes** the SBC Kubernetes cluster should be fully provisioned and ready for **Configuration**.

**Configuration**

**Overview**

The Configuration step will include several different tasks divided into two categories.

- **Cluster configuration**, joining of nodes and cluster management.
- **Pod and service deployments**, deployment of Kubernetes pods and services.

The goal of the implementation of the configuration is to make interaction with the Kubernetes control-plane easier as well as the installation of the required pods for the **Monitoring** objective. Configuration will be realized by a combination of Ansible plays/tasks and Kubernetes deployments. After the evaluation, the cluster should be running a distributed workload of monitoring services that collect and is able to present different metrics derived from the cluster.

**Procedure**

**Cluster configuration**

The process of joining master and worker nodes together in a cluster requires the creation of administrative tokens for the master node. The token can, after that, be shared with the worker nodes, which authenticates with the given token against the master. This is all handled by the package **kubeadm**. However, the token exchange requires storage and transfer of the token and some other parameters to work in an automated manner.

To make interaction with the Kubernetes master-plane more accessible, the master node will be configured to allow the standard user to access the Kubernetes configuration file. This will allow for interaction with the cluster without root privileges.

Further configuration of the hosts will include the holding/marking of the installed Kubernetes packages. As Kubernetes requires special attention when upgrading packages [13], this ensures continued stability when configuring other parts of the hosts.

In terms of cluster configuration, the cluster should now be ready to run pods, deployments, and services as expected in a Kubernetes cluster. Utilizing the two worker nodes for hardware redundancy and workload distribution.

**Pod and service deployments**

The Kubernetes deployments which will be part of the base configuration of the cluster are:

- A cluster monitoring deployment
- A node and pod monitoring stack, which includes:
  - Gathering
Cluster monitoring will be handled by the deployment of Kubernetes Dashboard (Web UI). For proper interaction with the monitoring deployment, both a service account and a cluster role binding needs to be created in the Kubernetes cluster. As the deployment will create a new namespace called kubernetes-dashboard, the service account is required to access the dashboard with unimpeded access. A separate role called kubernetes-dashboard utilizing Ansible’s k8s module is therefore created to automate the needed cluster configuration. With the pods for the Kubernetes Dashboard up and running, the monitoring service can be accessed by the token associated with the service account previously created.

The node and pod monitoring stack will consist of the following tools:

- **Node exporter**, Prometheus exporter for hardware and OS metrics exposed by *NIX kernels.
- **Kube state metrics**, Prometheus exporter, which listens to the Kubernetes API server and generates metrics about the state of the objects.
- **Prometheus**, collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and trigger alerts.
- **Grafana**, allows queries, visualizations, alerts, and interactions with data metrics independent from data storage.

In short, in this stack Node exporter exposes the metrics which Prometheus consolidates and Grafana presents. For ease of deployment of the monitoring stack, Helm the package manager for Kubernetes can be utilized. Helm allows for Kubernetes deployments and services to be directly applied to a cluster without the need of writing custom definition files. Instead, a version-controlled repository with stable and official deployments often written by the tools developers themselves can be used. Any configuration needed for the deployment/service can be applied either in Helms installation command or as preferred provided as an accompanying standalone configuration file.

Deployment of Helm packages (called Charts) is easy, usually consisting of nothing more than a single command. It is here when deploying the Prometheus chart; an implementation problem is encountered. One of the metric exporters kube-state-metrics is not runnable on arm platforms. Further investigation on this issue reveals that the recompilation of the exporter’s source code makes it compatible. However, recompilation and distribution of a custom deployment would stray away from the original goal of this investigation. Therefore, the decision is made to configure the Prometheus chart to exclude the kube-state-metrics exporter and focus on the metrics available through the other exporters.

Prometheus includes a local on-disk time-series database, but can also integrate with remote storage systems. In this implementation, long-term storage is omitted altogether, as a robust storage solution involves further hardware additions to the cluster.

The cluster configuration is now fully implemented, with a distributed workload running the required deployments needed for Monitoring.
Monitoring

Overview

Monitoring will evaluate if the two monitoring solutions’ collection and presentation of different metrics are functional and working correctly. When evaluating the monitoring stack, its limitations and extendability will also be analyzed. Compared to the two preceding evaluation objectives Provisioning and Configuration, this objective is less black and white in its requirements and more analysis dependant. Evaluation of the cluster monitoring is divided into two categories:

- **Kubernetes Dashboard**, functionality of metrics collected and presented by the Web UI deployment.
- **Monitoring stack**, functionality of metrics and presentation of the monitoring stack.

Procedure

The Kubernetes Dashboard (Web UI) deployment is a preset solution providing a quick overview of Kubernetes clusters. Its neither very configurable or functionality heavy compared to custom solutions. This means it does not require any implementation during this objective and is ready for analysis straight away.

Proceeding to the custom monitoring stack. As mentioned under Configuration of Pod and service deployments, Prometheus includes a local on-disk time-series database, but also optionally integrates with remote storage systems. Configuration can also be made to skip long-term storage of data altogether. In this implementation, long-term storage is omitted, as a robust storage solution involves further hardware additions to the cluster. The lack of storage of collected metrics does not impede the immediate collection and presentation of them. Also mentioned, data from the metrics exporter `kube-state-metrics` will not be available for monitoring in the current implementation. This means that metrics will primarily be derived from `Node exporter`.

Prometheus consolidates metrics and allows for a vast number of querying options. Common queries that will be implemented in the evaluation of the clusters monitoring are CPU utilization, memory usage, and HTTP requests. As for more cluster-specific queries, the count of pods and nodes will be monitored.

Writing queries for Prometheus is done by what is called PromQL. As Grafana handles the presentation of metrics, the PromQL queries are written and queried from the Grafana client and associated with different visual representations i.e., graphs and gauges. Dashboards created with different visualizations can be exported in JSON, which makes them version-controllable.

The monitoring of the cluster is now fully functional, providing an overview of different aspects of the cluster itself and the workloads running within it.
5 Result analysis

The following chapter evaluates the result from the conducted experiment of provisioning, configuration, and monitoring of SBC clusters in Chapter 4. Each objective is presented along with an overview of both its implementation and results according to the criteria. After the overview is a motivation for each criterion and its result. The criteria are the same as detailed under Chapter 4 in tbls. 4.4.2, 4.4.3, 4.4.4, 4.4.5. Lastly, Section 5.4 summarizes the result of the combined objectives as a whole.

5.1 Provisioning

Result

The result of the provisioning objective is three Ansible playbooks called Ubuntu, Docker, and Kubernetes. Each playbook handles a different part of the provisioning steps. Figure fig. 5.1 provides an overview of the parts and appliance of the implementation unto the cluster. Where the Ansible master provisions the different roles to all of the available hosts in the production inventory.

Analysis

The provisioning objective criteria are presented below, along with the reason for its deemed success or failure. Table tbl. 5.1 gives an overview of each criterion evaluated during the investigation of the provisioning step.
Table 5.1: Result of provisioning evaluation.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
<th>Result (OK/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idempotent execution of processes.</td>
<td>Repeated execution of modules without side effects</td>
<td>OK</td>
</tr>
<tr>
<td>Automated processes.</td>
<td>Full execution of modules without interaction.</td>
<td>OK</td>
</tr>
<tr>
<td>Version-controllable resource and configuration definitions.</td>
<td>Machine-readable implementation output.</td>
<td>OK</td>
</tr>
<tr>
<td>Creation of an inventory/collection with the single-board computers as nodes.</td>
<td>Automatic execution of low-level modules to several hosts.</td>
<td>OK</td>
</tr>
<tr>
<td>Connection to the single-board computers as remote nodes.</td>
<td>Automated remote access for provisioning tool to hosts.</td>
<td>OK</td>
</tr>
<tr>
<td>Execution of provisioning tool modules on the single-board computer nodes.</td>
<td>Provisioning of OS-level configuration and cluster dependencies.</td>
<td>OK</td>
</tr>
<tr>
<td>Automatic provisioning of Kubernetes resources on the single-board computer nodes.</td>
<td>Automatic provisioning of an initialized Kubernetes cluster.</td>
<td>OK</td>
</tr>
</tbody>
</table>

**Idempotent execution of processes**

Verified by repeated execution of modules without side effects. Given correctly written definition files, it does not seem to be any side effects when executing the provisioning modules unto the hosts. Ansible successfully detects and returns changed and skipped values depending on the status of existing or non-existent resources.

**Automated processes**

Verified by full execution of modules without interaction. Deemed successful, the provisioning step does not require any user input between initialization and completion. In case of a process critical error, execution halts allowing for error handling or repeated execution.

**Version-controllable resource and configuration definitions**

Verified by machine-readable implementation output. Successful, all configuration needed for the Raspberry Pis to be able to run a Kubernetes is placable in definition files and, therefore, able to be kept under version control.
Creation of an inventory/collection with the single-board computers as nodes
Verified by automatic execution of low-level modules to several hosts. Successful, low-level modules such as ping can be run against the collection of hosts providing independent responses and statuses of each host.

Connection to the single-board computers as remote nodes
Verified by automated remote access for provisioning tool to hosts. Connection to the hosts as remote nodes is related to the previous criteria. Now instead of pinging the hosts, access to them with root privileges is requested, allowing for actual configuration of the machines. No problems are encountered during verification of the criteria.

Execution of provisioning tool modules on the single-board computer nodes
Verified by provisioning of OS-level configuration and cluster dependencies. Successful, both the Ubuntu and Docker playbooks are able to properly install and configure the hosts for a proceeding Kubernetes installation.

Automatic provisioning of Kubernetes resources on the single-board computer nodes
Verified by automatic provisioning of an initialized Kubernetes cluster. This step was able to be completed without any complications. Both initialization of a Kubernetes cluster with a given CIDR and given CNI deployment was successful. Providing a fully functional Kubernetes cluster ready for configuration.

5.2 Configuration

Result
As the configuration is split into two parts, the implementation is presented in two separate figures figs. 5.2, 5.3. The first figure fig. 5.2 depicts the configuration of the cluster to be configured to its one master two worker architecture. Allowing for hardware redundancy and distributed workloads.

The second figure fig. 5.3 depicts the configuration of the clusters running deployments. Instead of functioning as the previous provisioning and configuration against the hosts. The configuration is now directly applied to the Kubernetes cluster.

Analysis
For an overview of the result for the evaluation of the configuration, objective see tbl. 5.2.
Figure 5.2: Overview of the cluster configuration implementation

Figure 5.3: Overview of the pod and services configuration implementation
Table 5.2: Result of configuration evaluation.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
<th>Result (OK/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idempotent execution of processes.</td>
<td>Repeated execution of modules without side effects</td>
<td>OK</td>
</tr>
<tr>
<td>Automated processes.</td>
<td>Full execution of modules without interaction.</td>
<td>OK</td>
</tr>
<tr>
<td>Version-controllable resource and configuration definitions.</td>
<td>Machine-readable implementation output.</td>
<td>OK</td>
</tr>
<tr>
<td>Continuous management of a single-board computer node</td>
<td>Execution of configuration modules on the provisioned cluster.</td>
<td>OK</td>
</tr>
<tr>
<td>inventory/collection.</td>
<td>Repeated configuration and deployment of modules on/to the cluster.</td>
<td>OK</td>
</tr>
<tr>
<td>Repeated execution of configuration modules on the single-board</td>
<td>Execution of Kubernetes resources to the cluster.</td>
<td>OK</td>
</tr>
<tr>
<td>computer cluster.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous deployment of Kubernetes deployments, services,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and jobs to the single-board computer cluster.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Idempotent execution of processes**

Verified by repeated execution of modules without side effects. The Ansible playbooks constructed for the configuration objective behave as those created during provisioning. Depending on how they are written, repeated execution without side effects is easily achievable. The Helm charts and Kubernetes deployments applied to the cluster also works as expected against any other cluster environment. Successfully detecting already present deployments of applications and services, handling such cases accordingly.

**Automated processes**

Verified by full execution of modules without interaction. Successful, Ansible is able to both configure the cluster and execute Kubernetes related commands automatically using predefined definition files. Allowing a completely automated process.

**Version-controllable resource and configuration definitions**

Verified by machine-readable implementation output. Ansible playbooks, tasks, and roles are all version-controllable. It is more interesting that all configuration of services and applications deployed also could be put in separate definition files. Making both deployment of and the applications internal configuration version-controllable.
Continuous management of a single-board computer node inventory/collection

Verified by execution of configuration modules on the provisioned cluster and deemed successful. The configuration of the cluster allows for continued management of it, even after its initialization. Access to cluster nodes and Kubernetes API can be shared through the administrative configuration file of the cluster to external machines. Allowing for execution of modules unto the cluster from inside and outside the cluster.

Repeated execution of configuration modules on the single-board computer cluster

Verified by repeated configuration and deployment of modules on/to the cluster. Even after its initialization, the cluster can be further configured, for example, by joining of additional nodes to the cluster.

Continuous deployment of Kubernetes deployments, services, and jobs to the single-board computer cluster

Verified by execution of Kubernetes resources to the cluster. The Kubernetes cluster operates as expected, allowing new deployments and services to be applied and deleted as wished. Further, while using Helm as a deployment tool, incremental deployments and rollbacks are available for any future deployments of applications. The criteria are, therefore, fulfilled and successful.

5.3 Monitoring

Result

Monitoring of the SBC cluster is, as mentioned, implemented in two scenarios, both present in fig. 5.4. Scenario one regarding the Kubernetes Dashboard is represented by the single application listening in on the Kubernetes control plane.

The more advanced monitoring stack is represented by the three tools which deal with different areas of responsibility. The general flow of metrics can be seen in the fig. 5.4 with Node exporter gathering metrics from the two worker nodes and passing them forward in the stack.

Analysis

Table tbl. 5.3 provides an overview of the result of each criterion for the monitoring objective.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
<th>Result (OK/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idempotent execution of processes. Automated processes.</td>
<td>Repeated execution of modules without side effects Full execution of modules without interaction.</td>
<td>OK  OK</td>
</tr>
</tbody>
</table>
Criteria Verification

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Verification</th>
<th>Result (OK/FAIL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version-controllable resource and configuration definitions.</td>
<td>Machine-readable implementation output.</td>
<td>OK</td>
</tr>
<tr>
<td>Collection of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment.</td>
<td>Analysis of monitoring tool(s) output.</td>
<td>OK</td>
</tr>
<tr>
<td>Storage of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment.</td>
<td>Analysis of monitoring tool(s) storage convention.</td>
<td>OK</td>
</tr>
<tr>
<td>Presentation of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment.</td>
<td>Analysis of monitoring dashboard deployment and functionality.</td>
<td>OK</td>
</tr>
</tbody>
</table>

**Idempotent execution of processes**

Verified by repeated execution of modules without side effects. Most applicable to Objective 1 and 2, the idempotent execution of processes is somewhat limited in the monitoring step. Further, the line between monitoring and configuration is blurred in this regard. Nevertheless, no apparent side effects can be observed in the execution of either monitoring solution.

**Automated processes**

Verified by full execution of modules without interaction. Successful, no user interaction is required for monitoring. Complete dashboards and metric collection/presentation are configurable in predefined definition files.

**Version-controllable resource and configuration definitions**

Verified by machine-readable implementation output. When applicable, as in case of the monitoring stack, all configuration is conducted through definition files. For example, exclusion of the Kube state metrics exporter in the deployment of Prometheus as well as dashboards and graphs in Grafana.

**Collection of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment**

Verified by analysis of monitoring tool(s) output. Deemed successful with an important caveat. Both the Kubernetes Dashboard and the monitoring stack is fully functional in its collection of cluster metrics through their available exporters. Though the Kube state metrics exporter is not natively supported on arm64 platforms. This means that while metrics collection is possible, not all metrics may be available for monitoring. The issues and implications of this caveat are discussed in Chapter 6.
Storage of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment

Verified by analysis of monitoring tool(s) storage convention. Storage of the collected metrics can be automatically configured to suit the SBC cluster’s need. In the experiment environment where no long-term storage is present, a configuration can be made to Prometheus time-series database to work continuously without long-term storage through definition files. The criterion is therefore successful as it can be tailored to fit the SBC cluster needs without manual intervention.

Presentation of metrics and statistics concerning nodes and pods in an SBC based Kubernetes environment

Verified by analysis of monitoring dashboard deployment and functionality. Grafana can successfully present all metrics queryable through PromQL. No apparent limitations seem to be present in its presentation capabilities on arm64 platforms or SBC clusters. All configuration and dashboards are also available for specifications in definition files. Allowing complete monitoring solutions to be constructed and shared before deployment.

5.4 Findings

Summarizing the three objectives evaluation, the conclusion can be drawn that according to the criteria the experiment was a success. As Chapter 4.4 in table tbl. 4.4.2 stated, some criteria would be applicable to all objectives, in this case, idempotent execution, automated process, and version-controllable definitions. All the objectives fulfilled the general requirements. Further, every objective specific criteria have been deemed satisfactory according to their respective verification process. Only the collection of metrics and statistics during the monitoring step holds a point of contention, as mentioned in its motivation discussed in Chapter 6.
Nevertheless, every evaluation criteria have been fulfilled to a satisfactory degree, which could point to SBC clusters being good candidates for educational and demonstrative opportunities regarding DevOps, configuration management, and continuous methodologies. This subject is discussed in detail in the following Chapter 6.
6 Discussion

Analysis according to the criteria and results presented in tbls. 5.1, 5.2, 5.3 under Chapter 5 shows that all verification steps are deemed successful. It is important to realize that the investigation is limited in scope and cannot be interpreted as a comprehensive or final conclusion on SBC clusters and the possibility of applying continuous methodologies unto them.

What can be concluded is the possibility of further investigations on the matter based upon the results of this investigation. In the following discussion, the experiment will be discussed as a whole and not in terms of the three objectives.

In Chapter 1, the problem which would act as a base for the investigation was formulated as the following. Examining if single-board computer clusters can be treated as equals to their production counterparts when commonly used cloud-native tools and principles are applied to them. Using the original formulation, the experiment have investigated SBC clusters with the perspective of DevOps, configuration management, and continuous methodologies.

Beginning with the high-level goals of both the processes and methodologies as well as the experiment.

- Everything which regards infrastructure and configuration should be version-controllable.
- Infrastructure and configuration should be fully buildable from scratch, only using files kept under version control.
- The process of building and configuration should be an automated process.

The three goals have all been fulfilled during the investigation, from the provisioning of the SBCs to monitoring them. By using the same tool for provisioning and configuration, the certainty of properly handled configuration for both have been somewhat guaranteed. Monitoring has relied upon Kubernetes deployments and Helm chart definitions of applications and services, which are built with the three points above in mind. Also, the configuration of the monitoring stacks different parts such as service configuration, metric storage, and presentations of graphs have been put under version control.

Building upon those as a foundation, the tools used to realize the cluster’s configuration management, mainly Ansible, have been flawless in its execution unto the hardware and hosts. The agentless nature of Ansible may very well be a contributing factor to this, though not by any means proven as no provisioning or configuration tools which use agents have been tested. Though the reduced complexity of operation removes one potential point of failure. The experiment did not have a specific provisioning/configuration tool in mind, and the choice of Ansible was because of familiarity and execution model. Would the result of the experiment be different given another tool for provisioning and configuration? Possibly yes, it is not possible to determine if the experiment was successful out of sheer luck by choosing Ansible or if it would have remained the same with another tool. Nevertheless, with Ansible’s success, at least one point of reference has been created for future investigations with other tools.

Looking at the basic building blocks of the cluster, the Raspberry Pis and Ubuntu 19.10. There have not been any apparent problems due to hardware or operating system, which speaks well for the interchangeable nature of container technologies in general. This is more of confirmation on previous studies [1], [2] than a new finding, but still a point worth mentioning.
6.1 Cluster and experiment improvements and problems

The first question that will be discussed concerns the investigations experiment. In retrospect what could be improved, done differently or be the basis for future investigations?

In terms of hardware, there is almost an endless amount of combinations and additions available. Most likely, no cluster configuration would be the same if compared with each other. Long-term storage remains an unexplored facet of the SBC cluster which could impose different complications into the equation. Furthermore nodes, especially master nodes, opens up a lot more experimentation in terms of high availability och hardware redundancy. The Raspberry Pi 2GB version chosen as a basis is neither the most powerful SBC available or the most limited. Both ends of the spectrum would be candidates for a deeper dive into SBC clusters.

Moving on to software, an equal amount of investigation is left as with hardware. The possible variations and possibilities given the vast amount of software leave room for almost endless investigations. The software chosen for this investigation is at least capable of proving that the concepts can be applied to SBC clusters and provisioning, configuration, and monitoring. Is Ansible the be-all and end-all solution for provisioning and configuration? No, but it is a widely used tool in the problem domain that successfully competes with similar tools on the market. Likewise can be said about the monitoring stack using Prometheus and Grafana as building blocks. The point is that now, with some tools investigated as part of provisioning, configuration, and monitoring, another tool can be evaluated against it. For example, would SaltStack provide improvements to the provisioning and configuration of SBC clusters compared to Ansible? If so, what kind of benefits would they be? Furthermore, does the tool, therefore, handle the arm64 platform better or worse than the other?

Given that the implementation and experiment environment could be forever modified for improvement, what in the current implementation provides apparent problems that need to be solved? This is mainly answered by the missing exporter for monitoring of the running cluster. Missing the kube state metrics exporter is a sore point in the cluster, it does question if the cluster is adequately monitored. Both node exporter and kube state metrics are useful in the context of monitoring, but they have also been created to fill different roles in cluster monitoring. This results in that there is a noticeable gap in the data metrics exposed to the overlying applications of the monitoring stack. Does the missing exporter create a make-or-break situation? Again no, monitoring can still take place, though in a limited form. Furthermore, it is of interest to remember that a recompilation of the kube metrics exporter source seems to be a solution to the problem, allowing for complete metrics.

The encountered problem raises an interesting point about SBC clusters, where lies the responsibility or blame for tool compatibility? Is it the platform that restricts the choice of tools and functionality? Or should the tools be changed or redistributed as in this case with the arm64 platform in mind? Maybe as cloud-native technologies continue to grow [5] and the demand for SBCs as cluster hardware increases [1], [2], these rough edges will naturally smoothen with time and increased visibility.
6.2 Demonstrative and educational opportunities

Do SBC clusters provide demonstrative and educational opportunities for DevOps and configuration management? Absolutely, given both the prospects of the experiment result as well as personal experience implementing the cluster, there have been plentiful of educational opportunities. I would argue that SBC clusters fully allows for educational and demonstrative possibilities regarding configuration management and continuous management.

Where support and possibilities may lack, the SBC cluster still fulfills enough of the problem domain that the lack of example kube state metrics can be noticed, discussed, and evaluated. Moreover, as provisioning and configuration of SBC clusters seem to be fully functional when it comes to automation and version control. The potential accessibility of sharing and spreading SBC clusters to new people seems very high. Even if provisioning, configuration, and monitoring are of no interest to a developer, he/she could have a fully functional cluster automatically deployed within half an hour for running containerized applications. Reiterating on points made by [5], the results could increase the adoption and growth of cloud-native technologies further. Not only in professional settings but also with the low cost of SBCs allow for an easily accessible cluster for home use as well.

6.3 Findings in relations to related work

Do the results cohere with other academic studies in the same area? Looking mainly at the studies with SBCs as focus [1], [2], the result of this experiment seems to align with their drawn conclusions. SBC clusters are capable of running cluster workloads without a problem and provide the same capabilities when applying conventional processes and methodologies unto them. No facets of SBC clusters have been discovered during the experimentation that would challenge the result of either study.

As mentioned earlier, even though the conducted experiment only scratches the surface of container orchestration, tools, and single-board computers, it can hopefully still act as a small point of reference for other similar studies and investigations.
7 Conclusion and Future work

The report has covered the investigation and experimentation of single-board computer clusters and their appliance to popular cloud-native methods such as DevOps, configuration management, and continuous methodologies. A cluster was constructed with three Raspberry Pis as nodes and then had a Kubernetes cluster with an accompanying monitoring stack automatically and fully configured provisioned unto it.

During the work, tools like Ansible, Helm, Prometheus, and Grafana was used and applied to the cluster to investigate the actual implementation of the methodologies. The results of the experiment were by majority positive and the investigation an overall success. Combining the results, a conclusion can be drawn that configuration management and continuous methodologies can be applied to single-board computer clusters without difficulty. However, there is still room for improvement when it comes to some tool compatibilities and the arm64 platform.

The most obvious continuation based on the report’s findings is to apply the provisioning and configuration on a larger scale. Proving or disproving the actual accessibility of the results by sharing them and applying them to a large number of other single-board computer clusters. If the experiment itself would be continued and the work expanded, my opinion is that the environment should be expanded with another master node and a long term storage solution. Both of those would build upon the existing work and enable investigation of some unexplored cluster facets. Another master node would allow for a High Availability cluster and quite advanced pod migration and disaster scenario investigations. Long-term storage would allow for advanced logging services supplementing and expanding upon important cluster monitoring and management facets. Further, long-term storage would allow for investigations regarding conventional databases, database migrations, and database connectivity.

In the broader perspective of future work, there are a plethora of options available spanning from hardware choice to cluster specifics such as CNIs and CRIs. Provisioning, configuration, and monitoring are also three subjects that deserve investigations of their own. Tools can be compared, configuration can be benchmarked, and monitoring can be set up as a complete production-ready solution. Container orchestration and its related methodologies span such a large area that ideas and possibilities for future work are seemingly endless. The future for single-board computer clusters seems to be bright, and soon maybe they can be equivalents to their data center counterpart.
8 References


