Managing a Kubernetes Cluster on Raspberry Pi Devices

Ferenc Füstös*, Katalin Péter*, Norbert-Péter László§, Szilárd-Gábor Mátis§, Zsolt Szabó§ and Csaba Sulyok*
* Faculty of Mathematics and Computer Science, Babeș-Bolyai University
RO-400084 Cluj-Napoca, Romania
§ Softech SRL
RO-400458 Cluj-Napoca, Romania
ferenc.fustos@stud.ubbcluj.ro; katalin.peter1@stud.ubbcluj.ro; laszlo.norbert@codespring.ro; matis.szilard@codespring.ro; szabo.zsolt@codespring.ro; csaba.sulyok@ubbcluj.ro

Abstract—**This paper presents the step-by-step setup of a scalable Kubernetes cluster consisting of Raspberry Pi computers. This process is mostly automated with the help of Ansible. The power of Kubernetes is shown through two proof-of-concept applications: one is a stateless, easily parallelizable single workload web service, while the other is a microservice-oriented application which can demonstrate the workload handling and scaling abilities of the system. The previously mentioned data can be monitored using Grafana and Prometheus.**

Index Terms—**Raspberry Pi; Kubernetes; K3s; Rancher; Docker; Monitoring**

I. Introduction

Kubernetes [1], [2] is an open-source system for managing containerized applications. Containers provide runtime isolation for a piece of software without the overhead of a virtual machine. They are launched from image files, which are packaged versions of an application together with its dependencies and configuration.

A Kubernetes cluster is a set of nodes: a master node and one or more worker nodes. Deploying an application is equivalent to placing it under the supervision of the cluster, which launches, manages and monitors its status. If a breakdown or failure occurs either on the container or the node level, the cluster can detect it and restart/move the affected container(s), minimizing any downtime [3].

Kubernetes also supports different types of scaling, including the ability to fine-tune the amount of resources reserved for a container based on its traffic, as well as running multiple replicas of it to spread out the workload as well as provide redundancy. Monitoring capabilities are also provided to track the behaviour and resource consumption of the nodes and/or deployed services.

The Raspberry Kube project intends to explore the world of container orchestration by building a Kubernetes cluster using only Raspberry Pis–a contrast to the common approach of deploying clusters on large on-premise servers or cloud providers. The cluster is configured to handle enterprise-level applications.

The current work also presents two proof of concept applications deployed to the cluster to test its efficiency and performance. The Mandelbrot Image Generator tackles a simple parallelizable problem through a minimalistic single-container web application, while the Movies project mimics an enterprise-grade system by using a distributed, thoroughly monitored microservice-oriented architecture.

The paper is structured as follows: Section II is an introduction to the world of Kubernetes and Rancher, while Section III is about the physical components and configuration of the cluster. The deployed applications and their associated monitoring tools are presented in Sections IV and V, with Section VI listing conclusions and further development potentials.

II. Kubernetes and Rancher

Kubernetes, derived from the Greek word meaning “helmsman” or “pilot”, is an open-source orchestration system which automates the deployment, scaling and management of containerized systems. Originally released by Google in 2014, it was later adopted by the Cloud Native Computing Foundation [4].

Nowadays different companies can easily make their web applications accessible using Kubernetes, regardless of them having their own infrastructure or not. In either case, the Kubernetes system provides a unified solution for the deployment of applications.

Applications that are to be deployed to Kubernetes need to be standardized, so any system can run them in the exact same way. This process is called containerization. A container means a software package, that contains all necessary dependencies needed to run.

Originally, Kubernetes only supported Docker as a containerization technology. Later it became possible to run containers in it using Containerd and other programs, which implement the Container Runtime Interface.

Kubernetes provides a RESTful[1] HTTP API which can be used by a developer to create, modify or delete resources. This API is usually accessed using the kubectl command line utility. As a result of the nature of the API, it can be used through browser based user interfaces as well, like Kubernetes dashboard or Rancher.

[1]Representational State Transfer

Kubernetes has multiple lightweight distributions, that allow running a cluster on computationally weaker devices. These include Microk8s, Minikube, and K3s.

K3s

K3s is a lightweight, production-ready Kubernetes distribution. Some advantages of K3s are:

- straightforward installation process
- small executable file size
- support for multiple operating systems and CPU architectures (e.g. amd64, arm64, armv7, etc.)

Thanks to the easy installation process and the wide variety of supported platforms, a cluster, which is also usable for testing purposes, can be set up in the span of a few minutes on the computer of a developer.

K3s creates a pre-configured Kubernetes system during the installation process. This process includes the installation of Traefik as the ingress controller of the cluster.

Rancher

Rancher is a unified multi-cluster management system, which facilitates the maintenance and management of multiple Kubernetes clusters. It provides security and authorization solutions.

One of the essential services of Rancher is its web-based control interface, with whom Kubernetes system administrators can register Kubernetes clusters and afterwards configure them. A cluster can be registered after supplying the respective information by deploying a resource file to the managed Kubernetes cluster. After being registered, the cluster becomes available through its generated id and the corresponding HTTP endpoint. This approach completely mirrors the API of Kubernetes, allowing the uniform management of multiple clusters through the generated endpoints. This is mostly useful for command line utilities, such as kubectl.

User management and integration of external authentication providers is another one of the functionalities that Rancher provides. It can be integrated with a single external authentication provider (e.g. Google OAuth, GitHub, Microsoft Active Directory, OpenLDAP and others) and can manage the permissions of individual users and groups. For example, Rancher can be configured so members of a certain GitHub organization can access the cluster, based on their permission levels.

III. Assembling the system

This section contains the steps of setting up and configuring the actual cluster, which is made out of 4 Raspberry Pis.

Physical components

The presented system is made of 4 Raspberry Pi 4 Model B devices. Each of them boasts 8 GB of RAM, giving a total of 32 GB RAM and 16 1.5 GHz processor cores. Some of these resources are used by the operating systems and base processes, leaving 30.5 GB manageable by Kubernetes. The maximum power consumption of a Raspberry Pi is less than 15 Watts, bringing the total maximum consumption to at most 60 Watts.

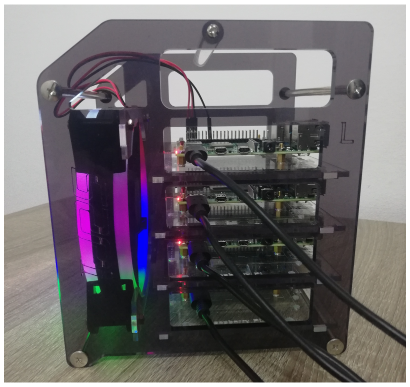

Fig. 1. The cluster in its powered on state

Each Pi has a USB type-C charging cable and 64 GB MicroSD card. The 4 Pis and their cooling system is held together by a rack.

As the first step, the MicroSD cards are inserted into the Pis with the help of extenders, which make it easier to access the MicroSD cards without disassembling the rack. Each of the Pis are afterwards attached to a transparent plane.

The next step is affixing the 4 Pis and the cooling fans to the frame. The fan is plugged into the Pi on top (the master node).

Finally, the charging cables are plugged in (as seen on Figure 1). From this point on, the cluster is ready to be configured.

Operating system

One of the cluster's most important attributes is the operating system (OS). The presented Raspberry Pi devices are running the official Raspberry PI OS Lite 32 bit OS. This is a non-graphical, open-source OS, which is a fork of Debian Bullseye, called Raspbian, optimized for Raspberry Pi computers.

The OS is installed onto the Pis using the Raspberry Pi Imager program.

Network configuration

Each computer that makes up the cluster needs to have internet access, since it has to download the Docker images it will run. Communication between computers also happens through the network.

The nodes are not connected directly to the public internet, since this would open multiple possible venues of attack. The Pis are plugged into a network switch instead, which assigns a single public IP address to the master node. The other nodes can access the public internet only through it. The firewall of the switch allows network packets through certain predefined ports (HTTP, HTTPS, SSH, Kubernetes API).

The Raspberry Pi designated as the master node creates a virtual network with the 192.168.1.0/24 subnet mask, and each Pi sets their IP address using static configuration:

```
master  192.168.1.200
worker1 192.168.1.201
worker2 192.168.1.202
worker3 192.168.1.203
```

This configuration makes the worker nodes available through the master node via Secure Shell (SSH).

The domain name `raspberryk3s.duckdns.org` is mapped to the public IP of the cluster by the free dynamic DNS service DuckDNS [5]. Since this is a wildcard DNS entry, all possible subdomains also point to the same address, ensuring availability and convenient routing opportunities for deployed applications.

The virtual cluster

The assembly of the virtual cluster consists in connecting the other three nodes to the master. This can be achieved by the use of tokens. Firstly, K3s must be installed to the master node, after which the token can be retrieved from the *k3s* directory. Afterwards, this can be supplied during the installation of K3s on each of the remaining three devices, as seen in the following code segment:

```
    curl -sfL https://get.k3s.io |
K3S_TOKEN="{{token.stdout}}"
K3S_URL="https://master.raspberry:6443"
K3S_NODE_NAME="raspberrypi-{{inventory_hostname}}"
sh -
```

After running the command, the node will automatically connect to the master. This can be verified via the kubectl command line utility on the master node.

Ansible

As it is demonstrated above, manually assembling the virtual cluster can be a monotonic procedure that consists of connecting to the nodes of the cluster one by one and running the installation command on each of them. The manual input can be reduced by using Ansible.

Ansible [6] is an open-source automation engine, designed for multi-tier deployments, enabling infrastructure as code. Taking advantage of the relations between different components, Ansible makes it possible to configure different schemes by using YAML files as playbooks. This way, it can be used for deploying applications and cloud provisioning too. It can be easily installed on the Raspberry Pis by running the following command:

```
sudo apt install ansible
```

The steps of assembling the virtual cluster can be listed in the playbooks. These can be run on one or more hosts, depending on the action. In the playbooks, every host is referenced by its name. Hosts can be collected into groups and referenced by group names.

The data required for connecting to the Raspberry Pis is stored in a hosts file. This contains the IP addresses, usernames and passwords (see section III for the introduced safety measures) needed for connecting via SSH. An important step is to disable strict host key checking:

```
ansible_ssh_extra_args=
'-o StrictHostKeyChecking=no'
```

SSH associates one key to every known host on default, which can be a problem if the key changes along the way. Associating new keys to the known hosts is possible by disabling the strict check.

Moving on to the playbooks, assembling the cluster can be divided into four steps (meaning four playbooks). Firstly, each Pi needs to have a user that is able to run sudo commands without needing a password. This being said, the first playbook consists of creating these users, named kube on each node. The second playbook initializes the IP addresses of the Pis. The next step is to install Kubernetes on the master node and to save the token. Lastly, Kubernetes is installed on the worker nodes using the token.

This assembly can be further simplified if the run commands of the playbooks are listed in a Makefile. Not only is the running order set in stone this way, but the method allows the password management to be implemented more effectively too.

Passwords and security

Assembling the virtual cluster as described in the III section is an effective, but a rather dangerous way to do so. Hard-coding the password is an obvious security risk that is better to be avoided.

Ansible has a built-in encryption tool, named Ansible Vault, for this very reason. It is intended for encrypting confidential information needed for running the playbooks. These passwords can be stored in a structured file. Ansible Vault can encrypt any kind of structured file that Ansible is able to process. This way, variables, tasks and even full playbooks can be encrypted.

In this case, the confidential information consists of the passwords used when accessing the Pis through SSH. These are collected in a file, named nodes-pass.yml, that is only accessible by the group (for security reasons), meaning that this is the file that someone, who wants to assemble the virtual cluster, needs.

Encrypting the file includes a password that is later on used to view:

```
ansible-vault view nodes-pass.yml
```

or to decrypt:

```
ansible-vault decrypt nodes-pass.yml
```

the file using Ansible Vault. Naturally, using the file also requires the knowledge of this password.

Now the hard-coded passwords in playbooks can be substituted with the names given to the password in the nodes-pass file. This way, the Makefile described in Section III also needs to be modified, otherwise one would have to manually type in the previously given password before each playbook starts. The most effective way to do this is to store this password in a file (named password in this case) and use that file when running the Makefile in the following way:

```
ansible-playbook -i hosts kube/kube-init.yml
--vault-password-file=password
```

Leaving the password in a file is another security risk, which is why this file needs to be deleted after every run and recreated only upon a future run. These steps can be added to the Makefile, so it is enough to type in the password once and then it takes care of the rest.

This way the encrypted file containing the passwords, and its own password are the two things required to assemble the virtual cluster. After inserting the nodes-pass.yml in the same directory as the Makefile, running the `make` command and typing in the password, the virtual cluster is assembled.

Deleting the cluster can also be implemented this way. It consist of two playbooks (to uninstall Kubernetes on the master and worker nodes). The Makefile can be expanded with a `clean` section under which you can manage the Ansible Vault password in a similar way to setting up the cluster.

Rancher configuration

The Rancher application used to help manage the cluster is also physically deployed alongside it on the master node, using a separate Docker container, behind a reverse proxy to ensure HTTPS communication.

For easier user management, the use of GitHub external users is enabled on Rancher. Members of the “raspberry-kube” organization created on GitHub are able to log in to Rancher, and members of the “raspberry-kube-admin” group within this organization have cluster admin privileges–only they can manage cluster-level resources such as persistent volumes.

Accessing nodes through the network

The easiest method of connecting to the nodes of the Kubernetes cluster is through SSH. This is only required in the case of maintenance or system configuration. This step requires that the connecting party has a user created on the master node and has their public key uploaded to their user. Using SSH you can only connect to the computers in the cluster if you have the private key, thus guaranteeing security. After connecting to the Pi, we have access to the internal network of the cluster and are able to connect to the other nodes using SSH. For the internal SSH connections only plaintext passwords need to be provided, ownership of private keys is not required. This connection method was selected because of convenience. It is not the best, but tolerable from a security perspective, because it assumes that the user owns a private key (needed for the initial connection).

There is another method as well, which uses Rancher to handle the Kubernetes cluster running on the Raspberry Pis as a local context. This way, local kubectl commands can be run on the cluster.

As the first step, the Kubeconfig, downloadable from Rancher, is put into the `.kube` directory (placed into the home directory of the user during the installation of Kubernetes). In order to make the certificate-authority-data recognizable, the certificate must be downloaded from Rancher. In the second step, we download and install the certificate of Rancher as its Base64 encoded version. As the last step, the certificate-authority-data field of the Kubeconfig file needs to be replaced with the certificate-authority field and its value needs to be the path of the installed certificate. In order to run commands on the Kubernetes cluster, we need to select the cluster as the current context:

```
kubectl config use-context kubernetes-cluster
```

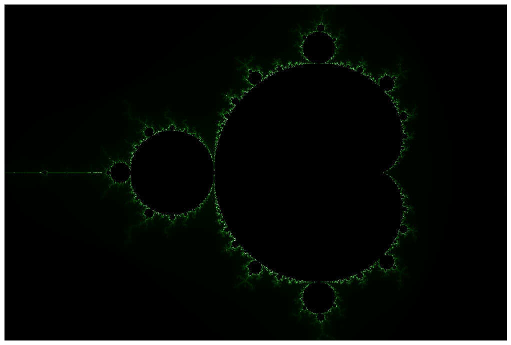

Fig. 2. Visual representation of the Mandelbrot set generated by the proof of concept project

IV. Mandelbrot Image Generator

The performance of the Kubernetes cluster built and configured as described in Section III may initially be studied by deploying small demonstrative workloads to it. In this section one such application is discussed.

The Mandelbrot Image Generator is a single-container stateless web application developed in Node.js, which aims to compute the elements of the Mandelbrot set on the cluster and display the results in a web browser.

The Mandelbrot set is a mathematical concept, defined as the set of complex numbers for which the iteration of the function $f_c(z) = z^2 + c$ does not diverge, where c is the complex number and z starts at 0.

The application includes a web-based user interface containing input fields for calculation parameters, as well as a canvas where the Mandelbrot set is plotted. The parameters include the size of the desired mathematical space, and the desired resolution of the elements. The most important parameter is how many parts the system should divide the mathematical space into–this directly impacts how many requests are sent to the backend. Since this task is unambiguously separable and parallelizable, it is well suited for measuring cluster load baring and performance.

The backend calculates the proximity of each complex number to the Mandelbrot set, returns this evaluation to the frontend, which will display the visual representation.

The project uses a public subdomain reserved by DuckDNS[2]. A generated visual representation of the Mandelbrot set can be seen in Figure 2.

V. The Movies project

The Movies project is a proof of concept meant to display the capabilities and usability of the created infrastructure. The application facilitates the sale of movie tickets, the

[2]`http://mandelbrot.raspberryk3s.duckdns.org/`

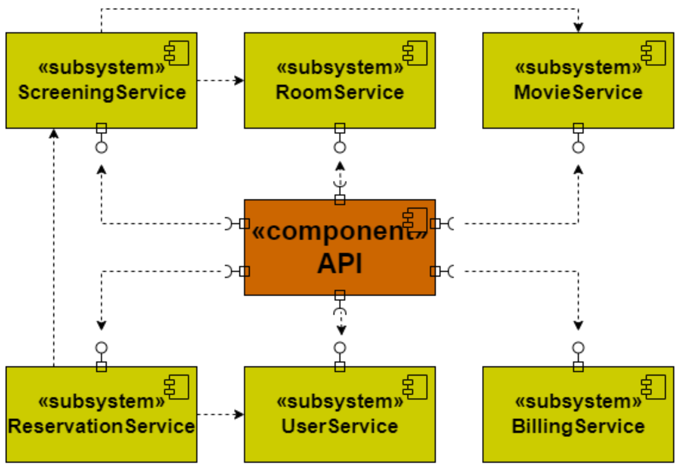

Fig. 3. The top-level architecture diagram of the Movies project

functionalities being separated into six different microservices. These are altogether able to manage user data, reservations for a specific movie at a pre-defined date and location, in addition to handling payments.

Architecture

The project is based on a microservice-oriented [7] architecture, meaning that it is divided into independent components that communicate with each other through well-defined protocols, usually HTTP REST APIs (see Figure 3).

This approach has numerous advantages, e.g. the microservices can be modified without having to update any of the others, meaning that in the case of a failure, downtime can be limited to the smallest possible fraction of the systemAnother advantage is scalability: since each microservice is deployed in its own container, resource requirements and replica counts may be independently assigned to them based on their load.

The microservices each have their own isolated data source, where they can persist relevant data and maintain a consistent state. They therefore conform to the design pattern "database per service" [7].

Technology

The primary reason driving the choice of technologies is their compatibility with microservice-oriented architectures. These include the Java programming language, Gradle as the build tool, as well as the Spring framework and its automatic configuration for bootstrapping a web application/API. The microservices in the Movies project follow the Controller-Service-Repository [8] pattern often used together with the mentioned technology stack. MongoDB, a document-based NoSQL database, is chosen for data persistence, since strict schema and relations are outside the focus of the project.

Deployment to the cluster is done via GitLab CI/CD (continuous integration/continuous development) pipelines reacting to version control changes. Upon detecting new commits, the CI jobs use Gradle to compile the code, and the Docker Buildx utility to package the resulting JAR files into a multi-architecture Docker image. The created images are then rolled out to the cluster, ensuring the changes are integrated into the deployments if they already exist.

Monitoring

One of the drawbacks of microservices is the difficulty of monitoring: the many involved containers all have isolated file systems and output streams, and although crashed pods are restarted automatically, their logs and resource usage information from before the incident may be lost, hindering worthwhile debugging. A common response to this issue is to deploy one or more auxiliary services to the cluster, whose responsibility lies in aggregating, storing and visualizing such metric data. Besides observing the behavior of the application(s) during runtime, such tools are also useful for demonstrating the capabilities of the created environment.

The monitoring stack deployed to aid the Raspberry Kube project consists of the open-source tools Prometheus and Grafana [9]. The former is a web application for general data aggregation and storage, while the latter is suitable for visually displaying the gathered data. The behavior of the application can be tracked simply using Grafana Dashboards to display the data, even down to the pod level.

In order for Prometheus to collect metrics data, it is necessary to configure how each microservice provides it. The configuration consists of defining counters for events (e.g. GET or POST request received) with unique names and optional tags. The data, using these two properties, can be visualized on Grafana Dashboards in the shape of tables, counters, flowcharts, etc (as seen on Figure 4).

VI. Conclusions and Further Development

This paper describes the steps for creating and configuring a monitored Kubernetes cluster with only Raspberry Pi computers. The main goal of the project is to explore the world of the scalable and redundant containerized systems in a miniature environment.

The usability of the created infrastructure is demonstrated by two deployed proof of concept applications. The single-container Mandelbrot Image Generator allows robust stress testing through a simple web interface. The more complex, microservice-oriented Movies application is closer to an enterprise-grade system due to its redundancy and monitoring through tools such as Prometheus and Grafana.

There are several opportunities to improve the functionality of the created infrastructure. One of the further development opportunities is to make the container deployment process faster and more confident with Spinnaker continuous delivery platform. For code quality control a SonarQube instance can be deployed to the cluster. Another development opportunity is to test the performance and behaviour of the cluster with event-driven applications. In addition to these, another enhancement can be done at the cluster level by deploying

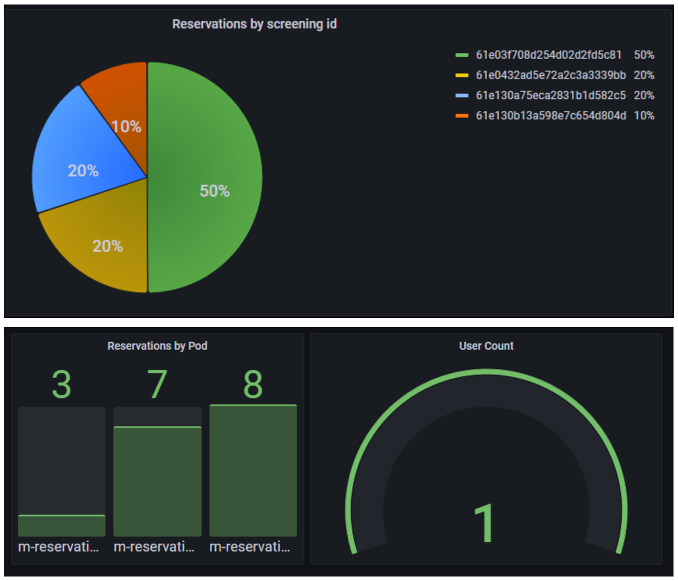
Reservations by screening id
🟩 61e03f708d254d02d2fd5c81 50%
🟨 61e0432ad5e72a2c3a3339bb 20%
🟦 61e130a75eca2831b1d582c5 20%
🟧 61e130b13a598e7c654d804d 10%
10%
20%
50%
20%
Reservations by Pod
3
7
8
m-reservati...
m-reservati...
m-reservati...
User Count
1

Fig. 4. Grafana Dashboards created for the Movies project. The upper image shows the reservations created by screenings, while the lower one shows two different Dashboards: the first one is for monitoring reservations by pods, and the second one is a counter for registered users.

cert-manager to the cluster to manage SSL certificates automatically. Doing so ensures secure data transfer via usage of the HTTPS protocol when communicating with applications deployed to the cluster.

References

[1] H. Saito, H. Lee, and C. Wu, *DevOps with Kubernetes: Accelerating software delivery with container orchestrators*. Packt Publishing, 2017.

[2] K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. O'Reilly Media, 2017.

[3] J. Arundel and J. Domingus, *Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud*. O'Reilly Media, 2019.

[4] The Linux Foundation, "Cloud Native Computing Foundation Announces Kubernetes as First Graduated Project," 2018. [Online]. Available: https://bit.ly/3t8YekG

[5] D. Gupta, "Duck DNS, the best free dynamic DNS service you can use," 2022. [Online]. Available: https://bit.ly/3t73EfL

[6] J. Freeman and J. Keating, *Mastering Ansible: Automate configuration management and overcome deployment challenges with Ansible*. Packt Publishing, 2021.

[7] C. Richardson, *Microservices Patterns with examples in Java*. Manning Publications, 2019.

[8] M. Fowler, *Patterns of Enterprise Application Architecture*. Pearson Education, 2003.

[9] J. Bastos and P. Araújo, *Hands-On Infrastructure Monitoring with Prometheus: Implement and scale queries, dashboards, and alerting across machines and containers*. Packt Publishing, 2019.