A practical experience report on moving from VM to Kubernetes in an academic research group

Sebastian Copei[1]

[1] Kassel University, Kassel, Germany

✉ sco@uni-kassel.de

Abstract. Using Kubernetes in a productive environment increases the complexity of the needed software stack. Without further maintenance of the software infrastructure, technical debts can quickly arise. With this experience report, we want to outline our migration from classic VMs over Docker to a Kubernetes infrastructure for both providing software for our research projects and also for productive software. Despite all the challenges and barriers, the benefits after a successful migration, such as fast and easy CI/CD or the rollout of any service, are worth it.

1 Introduction

The constant transition from traditional deployment scenarios with plain virtual machines to distributed cloud systems with Kubernetes is omnipresent. Companies like Netflix, Spotify and Amazon are using cloud technologies at a large scale. Besides these large-scale applications, the usage of Kubernetes in a smaller environment is also associated with benefits [1, 10]. For a long time, in our research group, we deployed our software on a virtual machine hosting Ubuntu Server. For the deployment we used Jenkins and as an application server Tomcat [4]. As most of the software we were serving was written in Java, that stack was sufficient. Also in our teaching, we focused on the development of Java-based software or tools which runs in the Eclipse ecosystem like EMF[1]. However, the trend shows a migration from desktop application to web application (Software as a Service (SaaS)). Further, the students reached out at us to provide more web and cloud technologies in our teaching. This was not limited to our lectures, also the students wants to do more projects and theses within the cloud subject area. With this in mind, We decided to shift our teaching from classical desktop development to more modern approaches of web technologies. This also implies a change of the used programming language. While Java with Spring Boot[2] doubtless is a reliable choice for the implementation of backends, most modern frameworks require at least JavaScript for the frontend. To not lose the type system known from Java, we decided to use TypeScript instead of JavaScript. In addition, we utilize TypeScript in conjunction with NestJS[3] for backend implementation. The upcoming technology Docker was a perfect fit. With rising complexity of our applications, databases, proxies, microservices etc., we first used Docker Compose to manage applications in stacks. However, due to the lack of overview for all our services, we moved on to Rancher[4] as a management service for our

[1] https://projects.eclipse.org/projects/modeling.emf.emf
[2] https://spring.io/projects/spring-boot
[3] https://nestjs.com/
[4] https://www.rancher.com/

cluster. In this report, the author provides a brief introduction to our migration process from a virtual machine to a Kubernetes cluster. In particular, we want to show that Kubernetes is also a good choice for smaller deployment scenarios, despite its higher system complexity. In section 2, we will present the relevant requirements that needed to be considered during the migration process, as well as our general setting. After that, in section 3, the used technology stacks will be presented. Section 4 will highlight barriers and challenges during the migration process and section 5 focuses on the discussion, evaluation, and assessment of the migration process, as well as the acceptance of the new cloud environment. Finally, section 6 will give a summary and outlook.

2 Setting and Requirements

The establishment of an efficient IT environment is a fundamental requirement for our workgroup, enabling us to deliver and manage various applications like GitLab [5] or ownCloud[6] (later Nextcloud[7]). Beyond the essential infrastructure, the implementation of additional services becomes crucial to streamline interaction with students. Moreover, it is mandatory to ensure the availability of an environment capable of executing software associated with research projects and facilitating the research work conducted by our scientists. Furthermore, the infrastructure should support the seamless operation of services utilized in our teaching activities. This comprehensive IT environment will serve as a foundation for enhancing productivity, promoting collaboration, and accommodating diverse requirements within our workgroup. As described in the last section, we initially fulfilled this with virtual machines and hosted a GitLab server, where students could host their code, Tomcat as an application server, to host our Java applications that were produced during research projects or other research activities, and Jenkins as a tool for automated CI/CD pipelines. From this setting, we defined the following requirements which needed to be fulfilled by a new infrastructure after migration.

R1: All data has to be stored physically on devices inside of the University. As we are running software which is used by our university's administration, it is not permitted to save the data on servers which are provided by a hyperscaler like AWS.

R2: Applications and services need to be easily configurable. Each service is configurable through environment variables. There should be an understandable and readable way to manage and change these variables.

R2.1: Also students should be able to use the configuration tools. The used configuration tool should be user-friendly enough that even students can configure and deploy projects they develop during their studies.

R3: The infrastructure should be production-ready. As already described in **R1**, we host applications which are production-ready. After a migration, all used tools and software need to remain production-ready.

[5] https://docs.gitlab.com/ee/install/
[6] https://owncloud.com/de/
[7] https://nextcloud.com/de/

All in all, after the migration to Kubernetes, we were able to fulfill the defined requirements. The infrastructure after the migration will be introduced in the next section.

3 Used technology stack

We split the full migration path into smaller parts. We took the first step by migrating to Docker and simplifying the organization of applications with Docker Compose. Containers that require persistent storage use host binding to achieve this. Additionally, we copied the folders from the host binding to a network-attached storage (NAS) for backups. We installed nginx[8] as a proxy and for SSL termination. Our IT service center issued the SSL certificate. As we deployed more and more applications, we had to use a UI for easier administration. Consequently, we performed the subsequent step by migrating to an appropriate platform and choosing Rancher[9] as the option. By introducing Rancher, we replaced the previously used ngxin proxy with the integrated HAProxy [10] within Rancher. This transition included replacing the SSL certificate and using Let's Encrypt[11] for issuing certificates. In the final migration step, we moved from Docker to Kubernetes. In order to ensure high availability (HA) of the cluster, we implemented the recommended infrastructure as suggested by Rancher Labs [8, 9, 5, 6, 7]. Following Rancher Labs' recommendations, we established two distinct clusters. The initial cluster comprises three nodes for the Kubernetes cluster, with Rancher configured in a HA setup [6]. Additionally, a separate node is dedicated to serving as a nginx layer 4 load balancer [8, 7]. The second cluster, referred to as the downstream cluster [8], consists of two control plane nodes [3], three etcd nodes [3], two worker nodes [3], and an additional separate node serving as a nginx layer 4 load balancer [9, 5]. In this new setup, SSL termination is no longer performed by the nginx load balancers. Instead, the ingress controller[12] utilizes a certificate provided by cert-manager[13] and issued by Let's Encrypt. With the migration to Kubernetes, the transfer of persistent data to Kubernetes volumes[14] becomes necessary. However, in a HA setup, the specific node on which a container will be deployed is unknown. To ensure that the volumes remain consistently available on all worker nodes, we employed Longhorn[15] for data replication. Longhorn also handles the backups, which are further saved onto our NAS. For the monitoring of the nodes inside the downstream cluster, we used the recommended monitoring stack from Rancher Labs[16], which consists of Prometheus[17] for data collection and Grafana[18] for visualization. Lastly, we deployed a Keycloak[19] server for centralized user management and a Registry[20] to store our Docker images. From this point onwards, we transferred all our applications and projects to the downstream cluster. In addition to migrating the cluster, we transitioned from using a self-hosted GitLab to utilizing GitHub[21] and GitHub Classroom[22] for stu-

[8] https://www.nginx.com/
[9] https://rancher.com/docs/rancher/v1.6/en/
[10] https://rancher.com/docs/rancher/v1.6/en/cattle/adding-load-balancers/#adding-a-load-balancer-in-the-ui
[11] https://letsencrypt.org/de/
[12] https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
[13] https://cert-manager.io/
[14] https://kubernetes.io/docs/concepts/storage/volumes/
[15] https://longhorn.io/
[16] https://ranchermanager.docs.rancher.com/how-to-guides/advanced-user-guides/monitoring-alerting-guides/enable-monitoring
[17] https://prometheus.io/
[18] https://grafana.com/
[19] https://www.keycloak.org/
[20] https://docs.docker.com/registry/
[21] https://github.com
[22] https://classroom.github.com/

dent interaction and code project storage. In section 5, these applications will be described in detail.

4 Barriers and challenges

To fulfill requirement **R3**, we were forced to use Kubernetes in a high-availability setting. For this, Kubernetes requires a lot more server resources than a simple infrastructure with Docker and Docker Compose. At this point, we were facing our first barrier. Our target infrastructure contains 11 servers and two load balancers. Following requirement **R1**, all data has to be stored in our in-house datacenter, but there was not enough space for more hardware. The first attempt to solve this issue was to request multiple virtual machines at our IT service center. Unfortunately, they do not have the capacity for further hardware or virtual machines either. Because of missing knowledge, a direct hosting of a Kubernetes cluster also was not possible. The actual solution was to run the management components of Kubernetes at a cloud provider and the worker nodes in our in-house datacenter. With this setup, we are able to meet requirement **R3** without violating **R1**. To meet requirements **R2** and **R2.1** we decided to use Rancher as a cluster manager. Rancher is supposed to manage hundreds of clusters. The user interface (UI) has an integration for monitoring systems and provides with the catalog[23] an easy possibility to install certified helm charts. Also, unlike the default Kubernetes dashboard, Rancher has the capability to deploy applications and services through its UI. With this feature, requirement **R2.1** is fully satisfied. The main challenge of the migration process was the steep learning curve. This challenge can also be interpreted as a barrier. Currently, the author is the person who did the migration and built the infrastructure as it is. Due to the high complexity of the infrastructure (see section 3) and the steep learning curve, it is challenging to transfer the knowledge to another person who can effectively administer the cluster. In addition to administering the cluster, the person must also monitor its health and be capable of executing disaster recovery procedures in the event of major failures, such as a network outage. We were able to partly solve this challenge, the author served a lecture in our master computer science program and teaches current DevOps technologies, which also includes the creation and administration of Kubernetes clusters with Rancher.

5 Analysis

The applications we are running in our cluster can be grouped into the following categories.

1. Teaching
2. Student projects
3. Research projects
4. Real world projects

Teaching Student numbers naturally fluctuate, posing challenges to effectively responding to increased volumes without Kubernetes. However, Kubernetes' horizontal pod autoscaler[24] simplifies scaling, provided that the services being scaled are compatible with it [2]. In concrete terms, we provide a service system for a software engineering course in the bachelor's program, where the students

[23] https://ranchermanager.docs.rancher.com/pages-for-subheaders/helm-charts-in-rancher
[24] https://kubernetes.io/de/docs/tasks/run-application/horizontal-pod-autoscale/

need to implement the client to the server in teams. In the master's program, we simulate an industrial network for an Internet of Things (IoT) course. The students have to implement several simple services and deploy them inside the network, and the services have to cooperate.

Student projects With Rancher's user-friendly UI and integrated role-based user management, students have the ability to host their projects, which are required for their studies. Consequently, by only needing to familiarize themselves with writing a Dockerfile and utilizing CI/CD with GitHub Actions[25], students can allocate more time to focus on their projects rather than having to learn the intricacies of our cluster's entire stack.

Research projects The primary advantage of our research project is the increased visibility of our results. For instance, we have developed Fulib[26], a model generation tool for Java applications, as a further development of Fujaba[27]. By providing an online playground, individuals can experiment with or directly utilize the tool without the need for installation.

Real world projects We have developed during several student projects or with the work of our student assistant software which is used productively. The biggest example is a research newsletter[28] with round about 8000 user. With the capability abilities of the cluster, we can react fast and reliable onto traffic peaks.

Furthermore, for all our projects and applications, we establish multiple environments for development, staging, and production builds. Moreover, by utilizing GitHub Actions, we have created a unified CI/CD pipeline that can be reused. Although the initial complexity of the system is notably high, and the learning curve is steep, utilizing the cluster through Rancher significantly reduces complexity and lowers barriers for employing web technologies.

6 Conclusions

With this report, we outline our transition from virtual machines to a Kubernetes cluster, highlighting the barriers and challenges we encountered. Despite the steep learning curve and initial obstacles during the cluster setup, the reduced complexity of application deployment offers significant time savings that can be directed towards other engaging endeavors. Looking ahead, our aim is to adopt GitOps for our CI/CD pipeline, thereby diminishing configuration overhead.

7 Acknowledgement

I would like to pay my heartfelt tribute to the late Prof. Dr. Albert Zündorf, whose guidance, expertise, and unwavering support have been invaluable throughout my research journey. His profound impact

[25] https://github.com/features/actions

[26] https://fulib.org/

[27] https://web.cs.upb.de/archive/fujaba/home.html

[28] https://www.uni-kassel.de/uni/en/forschung/research-services/research-information-services/the-information-service-fit

and mentorship have shaped me into the researcher I am today, and I am forever grateful for his contributions to my academic and personal growth. I am also thankful to Jens Kosiol for his proofreading and input! I am deeply appreciative of his contributions to refining my research.

References

[1] DigitalOcean. Kubernetes for startups: Why, when, and how to adopt. https://www.digitalocean.com/resources/article/kubernetes-for-startups-why-when-and-how-to-adopt, last visited 22.06.2023.

[2] Susan J. Fowler. *Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization*. O'Reilly Media, Inc., 1st edition, 2016. https://www.oreilly.com/library/view/production-ready-microservices/9781491965962/ch04.html, last visited 22.06.2023.

[3] Google. Kubernetes components. https://kubernetes.io/docs/concepts/overview/components/, last visited 22.06.2023.

[4] Paul Jenkins and Jean Cassou. *Jenkins*. Gerd Hatje. https://www.theshagundave.com/static/pdfs/Jenkins%20-CI%20Tool-End%20to%20End%20Platform%20Design.pdf, last visited 22.06.2023.

[5] Rancher Labs. Architecture recommendations. https://ranchermanager.docs.rancher.com/reference-guides/rancher-manager-architecture/architecture-recommendations, last visited 22.06.2023.

[6] Rancher Labs. Checklist for production-ready clusters. https://ranchermanager.docs.rancher.com/pages-for-subheaders/checklist-for-production-ready-clusters, last visited 22.06.2023.

[7] Rancher Labs. Recommended cluster architecture. https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/kubernetes-clusters-in-rancher-setup/checklist-for-production-ready-clusters/recommended-cluster-architecture, last visited 22.06.2023.

[8] Rancher Labs. Set up infrastructure for a high availability rke kubernetes cluster. https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/infrastructure-setup/ha-rke1-kubernetes-cluster, last visited 22.06.2023.

[9] Rancher Labs. Setting up an nginx load balancer. https://ranchermanager.docs.rancher.com/how-to-guides/new-user-guides/infrastructure-setup/nginx-load-balancer, last visited 22.06.2023.

[10] Vivek Sharma. Managing multi-cloud deployments on kubernetes with istio, prometheus and grafana. In *2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS)*, volume 1, pages 525–529, 2022.