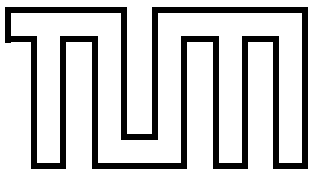


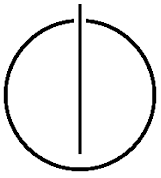
FAKULTÄT FÜR INFORMATIK

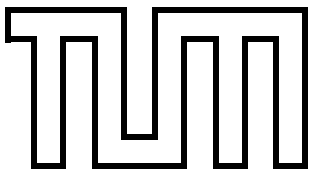
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Towards a Kubernetes Supported Learning Infrastructure at Scale

Matthias Linhuber





FAKULTÄT FÜR INFORMATIK

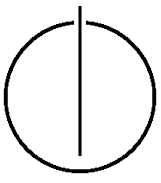
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Towards a Kubernetes Supported Learning Infrastructure at Scale

Kubernetes-gestützte Lerninfrastruktur im großen Maßstab

Author:	Matthias Linhuber
Supervisor:	Prof. Dr. Stephan Krusche
Advisor:	Prof. Dr. Stephan Krusche
Date:	15.11.21



1 Introduction

Artemis is an open-source learning platform that implements an interactive learning approach [KS18]. Educators create exercises that provide immediate and automated feedback to students, which significantly reduces manual correction efforts [Kru21]. As the number of students grows every year[1], Artemis became indispensable for large computer science classes at several universities. As both students and educators rely on Artemis to conduct graded exercises and exams, the system must be highly available and reliable.

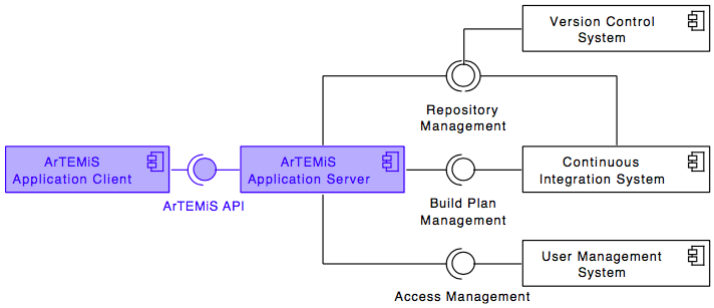


Figure 1: Artemis Top-Level Architecture (UML Component Diagram) Figure taken from Github Documentation[2]

Artemis is comprised of different subsystems and uses external services like Version Control and Continuous Integration Systems. Figure 1 shows an overview of an example Artemis deployment. Depending on the number of users and the system environment, the `Artemis Application Server` can be scaled vertically (providing more resources to a single server) or horizontally (extending to multiple `Artemis Application Server` forming a cluster) [Lei20]. At the Technical University Munich (TUM), Artemis and all the external systems are deployed to Virtual Machines (VM) using Ansible[3], VMware vSphere[4], and Proxmox[5]. Administrators have to provision each VM individually and perform regular security updates, which is inherently slow and error-prone.

[1] `https://www.in.tum.de/die-fakultaet/profil-der-fakultaet/die-fakultaet-in-zahlen/daten-und-fakten-2020/`

[2] `https://github.com/ls1intum/Artemis/blob/develop/docs/dev/system-design/TopLevelDesign.png`

[3] `https://docs.ansible.com`

[4] `https://www.vmware.com/de/products/vsphere.html`

[5] `https://proxmox.com/en/`

2 Problem

We identified four problems with the current deployment that we want to address in this thesis.

First, the current deployment at TUM with a total number of 102 VMs in multiple data centers is complex, heterogeneous, and challenging to maintain. Table 1 shows the number of VMs used for each Artemis-related system. Administrators must provide and maintain each VM by hand, imposing significant overhead if more resources are needed.

Second, Artemis is actively developed and needs to be deployed to the production environment regularly. The current deployment workflows for initial setup and version updates are distinct. The initial setup is executed using Ansible; updates are deployed via Bamboo and custom deployment scripts. This imposes a distinction between deployment activities performed by administrators and deployment activities performed by release managers. Both actors have a different understanding of the deployment strategy and are only loosely coupled. While the initial setup is automated, it is executed only once. Hence, it is not mandatory to adapt the setup configuration regularly to reflect necessary changes. This is a classic problem in software configuration management.

Third, the testing and production environments are different. Due to the complexity of a complete Artemis installation with all auxiliary services, providing representative testing environments is challenging and time-consuming. Thus the administrators at TUM provide only one staging environment used to execute both automated and manual tests before a release is deployed to production. This environment is shared and can only be used exclusively by one developer or development team to test changes. This slows down the development process and induces other problems caused by deviations between testing and production environments.

Fourth, horizontal scaling involves many manual steps in the current setup at TUM. Administrators have to provision and set up VMs and reconfigure Artemis to use the new resources. This process is cumbersome and too slow to respond to varying system load in a reasonable time.

3 Motivation

In recent years containerized deployment has become state of the art in software engineering [Ber14]. The need for a reliable and scalable container orchestration system quickly became evident [CI20]. Google introduced Kubernetes as a solution to container orchestration. Kubernetes is the de facto

System	vSphere VMs	Proxmox VMs	Sum
Artemis Application Server	7	4	11
Proxy / Broker / Storage / Database	4	0	4
Version Control System	0	7	7
Continuous Integration Server	0	1	1
Build Agents	20	50	70
User Management	0	1	1
Additional Supporting Services	5	3	8
Sum	**36**	**66**	**102**

Table 1: Number of VMs per System

standard in the industry and is also used in education already[6].

With the transition to Kubernetes, administrators no longer need to maintain Virtual Machines which run workloads. Kubernetes automatically provisions the necessary resources for the Artemis components and maintains software updates. The desired deployment state can be rigorously defined as configuration files that guarantee reproducibility and allow quick infrastructure changes [Bai17]. As Kubernetes is inherently hardware-agnostic, developers or administrators no longer need to maintain the components' network, storage, or operating systems. Kubernetes provides an easy-to-use abstraction for all necessary resources.

Kubernetes allows the definition of a complete Continuous Integration and Continuous Delivery pipeline. As a result, Artemis can be instantiated in a new environment with the exact mechanisms used to update or downgrade the system. This further improves the portability and allows external educators to instantiate a complete Artemis installation easily and in a standardized fashion.

As the complete deployment can be defined in a structured and rigorous way, deploying production-equivalent testing environments becomes easy. Each developer can instantiate a distinct testing environment the same way the production installation is deployed. This unified deployment strategy improves consistency and prevents software defects caused by deviations in execution environments.

Finally, Kubernetes allows Artemis to scale out or in depending on the system load dynamically. During lectures or exams, more Artemis Application Servers are rolled out to achieve the best possible system performance - During the night or the semester break, resources are free to be used in other

[6]Berkley JupyterHubs `https://docs.datahub.berkeley.edu/en/latest/admins/cluster-config.html`

systems. This allows a more cost-effective and environmentally friendly infrastructure design.

4 Objective

In this thesis, we introduce an on-premise Kubernetes deployment strategy tailored to teaching environments. We will outline the system architecture and guidelines for On-Premise Kubernetes clusters constrained by external systems and services like network and storage. We will propose hardware and software components to support Kubernetes and provide detailed information about the Kubernetes instantiation and management considerations. On-premise Kubernetes clusters are challenging to set up and maintain. The cluster needs to be deployed to many (virtual) servers, provided with public and private network address spaces and both local and shared storage. Ku-Kubernetes is very versatile and can be adapted to many use cases. While this is powerful, it introduces complexity that must be managed by choosing the best fit for our environment and use case. Consequently, we have to address the following challenges:

- **Cluster Networking**: Containers that are deployed in Kubernetes need IP addresses. We need to provide intra Kubernetes communication and addressing (Containers communication with each other) as well as inter Kubernetes communication (Containers interact with external entities). We have to support both IPv4 and IPv6 to be future prove and to be able to interface with additional infrastructure. As we don't have control over the university routers, we have to design the network configuration accordingly.

- **Name Resolution**: The Kubernetes cluster should allow arbitrary deployments and hence needs to support external name resolution. Kubernetes only provides DNS within the cluster, so we must implement additional external name resolution outside of the cluster.

- **Persistent Storage**: To be able to store data persistently in the cluster, a storage provider is needed. We have to choose a storage provider and deploy the solution to the cluster. Furthermore, we have to implement a backup strategy for the cluster data.

- **Certificate Management**: Services in the cluster have to be able to request a TLS certificate to support secure HTTPS connections. We need to design a Certificate Management process that is able to interact with the Certificate Authority at TUM.

- **User Management**: We want to allow developers to access the Kubernetes cluster for testing purposes. However, we need to make sure that they cannot interfere with the production deployment. Hence, we need a capable solution for user and role management.

We will compare the On-premise Kubernetes installation with Kubernetes provides in the Cloud (Amazon Elastic Kubernetes Service / Google Kubernetes Engine (GKE) / Linode Kubernetes Engine (LKE)) and present the benefits and challenges of both solutions in detail.

Furthermore, we will implement a Kubernetes base deployment workflow for Artemis, allowing easy and consistent installations in any Kubernetes cluster.

Finally, we will suggest a new Continuous Delivery workflow for Artemis to further improve the development lifecycle.

5 Schedule

- 15.11.2021 - Thesis Kickoff
- Until the end of November
  - Familiarize with the Artemis code base and previous work
  - Create a repository containing the necessary Infrastructure as Code definitions to setup a the underlaying infrastructure for a On-Premise Kubernetes Cluster
  - Decide on a Kubernetes Distribution
- Until the end of December - On-Premise Kubernetes
  - Functional Rancher instance
  - Functional Cluster Networks
  - Functional Kubernetes User Management
- Until the end of January
  - Functional External Name Resolution
  - Functional Persistent Storage
  - Functional Certificate Management
- Until the end of February - Deploy Artemis in Kubernetes

  - All Artemis sub-systems available as containers
  - Artemis can be deployed to a Kubernetes cluster
  - Artemis Application Servers can be scaled up and down
- Until the end of March - Cloud based Kubernetes Clusters
  - Research and document benefits and challenges of Cloud-Based Kubernetes clusters:
  - Deploy Artemis to different Cloud-based Kubernetes clusters
- Until the end of April
  - Implement a Kubernetes based Continuous Delivery workflow for Artemis
  - Final Presentation
- Until 15.05.22
  - Implement Feedback
  - Resolve possible follow up issues
- 15.05.22 - Final Delivery

References

[Bai17] Jonathan Baier. *Getting Started with Kubernetes*. Packt Publishing Ltd, 2017.

[Ber14] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[CI20] Emiliano Casalicchio and Stefano Iannucci. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, 32(17):e5668, 2020.

[Kru21] Stephan (Dr.) Krusche. Interactive learning - a scalable and adaptive learning approach for large courses, 2021.

[KS18] Stephan Krusche and Andreas Seitz. Artemis: An automatic assessment management system for interactive learning. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 284–289, 2018.

[Lei20] Simon Leiß. Securing and scaling artemis websocket architecture, Aug 2020.