KAPETÁNIOS: Automated Kubernetes Adaptation through a Digital Twin

Johannes Zerwas[1], Patrick Krämer[1], Răzvan-Mihai Ursu[1], Navidreza Asadi[1],
Phil Rodgers[2], Leon Wong*[2], Wolfgang Kellerer[1]
[1]Technical University of Munich, [2]Rakuten Mobile Inc.

***Abstract*—This demo presents a self-operating Kubernetes (K8s) cluster that uses digital twinning and machine learning to autonomously adapt its Horizontal Pod Autoscaler (HPA) to workload changes. The demo uses a digital twin of a K8s cluster to gather performance statistics and learn a model for the workload. With the model, the cluster autonomously adjusts HPA parameters for better performance. The demo illustrates this process and shows that the requested pod seconds decrease by $\sim$37 %, while the request latency stays mostly unaffected.**

***Index Terms*—Kubernetes, Digital Twin, Machine Learning**

I. INTRODUCTION

Modern networks such as 5G cloud-native cores become distributed service meshes, running containerized on top of Kubernetes (K8s) clusters [1]. Managing those clusters in a dynamic environment requires a high degree of automation. Automation is challenging as, e.g., adjusting configurations requires an understanding of how the cluster responds to changes. Today, this knowledge is provided through highly trained experts. This practice becomes a limiting factor to the evolution of modern networks as they grow in size and become more diverse, and dynamic. Thus, in the future, the clusters must continuously measure and analyze themselves to detect and react to changing conditions.

We present KAPETÁNIOS[1], a self-operating K8s cluster that adaptively tunes its scaling policy to dynamic workloads. Fig. 1 gives an overview of KAPETÁNIOS. At a high level, KAPETÁNIOS uses a digital twin (DT) to represent the cluster state, collect measurements, and manage the cluster. KAPETÁNIOS learns models of the cluster behavior with Machine Learning (ML) from the gathered data, and uses the learned models to enhance simulations, allowing KAPETÁNIOS to autonomously search for configurations. Simulations and ML models thereby replace human experience. In this demo, the ML models allow KAPETÁNIOS to predict the cluster utilization in response to the request arrival rate, and adjust the cluster configuration to new profiles. Specifically, KAPETÁNIOS automatically tunes the configuration of K8s' built-in horizontal pod autoscaler (HPA) with the goal to reduce the number of deployed application instances (pods) over time while keeping request completion time low.

* This work was conducted as a part of Beyond5G R&D Promotion Project (#01701), supported by National Institute of Information and Communication Technology (NICT), Japan.

[1]Kubernetes originates from the greek word for helmsman. We further automate the orchestration of containerized applications, reliving the operator from his position as the captain of K8s.

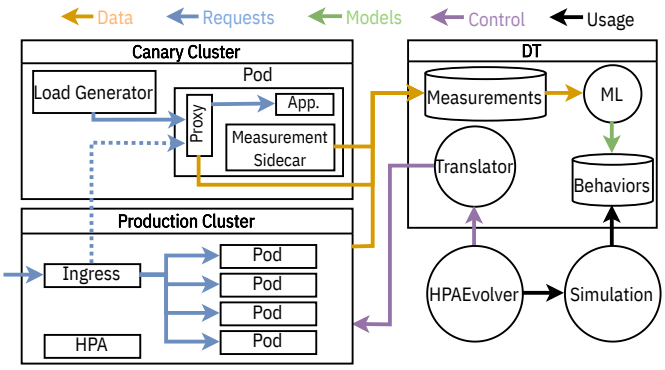

Fig. 1: Major components of KAPETÁNIOS.

II. BACKGROUND

K8s has emerged as the de facto standard to orchestrate service meshes. K8s employs a horizontal pod autoscaler (HPA) to realize scalable service management. The HPA monitors a predefined metric, e.g., CPU utilization of pods, and tries to meet Quality of Service (QoS) requirements by adding or removing pods, while avoiding over-provisioning [2], [3]. To achieve these goals, the HPA can optimize two parameters: (1) the frequency of checking conditions to scale; (2) the utilization thresholds at which the HPA scales [2], [3]. If the workload varies, e.g., the application's resource requirements, or the arrival patterns of the requests change, HPA settings are likely to become invalid [2].

Verifying existing settings and finding new ones is challenging. The operator has to know requests patterns, i.e., how requests arrive over time, as well as the application's resource profile, i.e., how resource demands change with the pattern. Both can only be obtained empirically, and do not directly translate into HPA settings. Obtaining the necessary information and translating them into HPA setting are considered time-consuming and tedious [2], [3].

III. KAPETÁNIOS

KAPETÁNIOS has five building blocks: production cluster, canary cluster, DT, simulation and `HPAEvolver`.

Production Cluster: a K8s cluster that provides services (applications), and has an HPA for each service. The ingress controller distributes requests to running pods. KAPETÁNIOS collects data via available APIs and logs. For example, the resource utilization through K8s' cAdvisor, and HPA actions and the deployment time of pods from the cluster's logs.

Canary Cluster: KAPETÁNIOS uses it to gather data which the production cluster does not provide. To this end, the canary cluster runs applications from the production cluster in a pod together with two sidecar containers. The first sidecar is a reverse proxy that gathers arrival, departure and service times per request. The second sidecar gathers CPU utilization at high resolutions (<1 s) and makes them available in real-time. Further, it parses the request information from the proxy logs.

Requests originate from two sources: (1) mirroring or forwarding random samples of requests from the production cluster; (2) synthetic workloads from the load generator. Sampling from the production cluster allows KAPETÁNIOS to detect changes in request patterns, e.g, request arrival rate or the distribution over services. For instance, if users start to access a more resource-intensive service more often, it might necessitate an update in the cluster configuration.

KAPETÁNIOS uses the *load generator* to generate diverse synthetic workloads to benchmark the applications with patterns that are different from the production traffic. For example, it synthesizes specific distributions over request types. Generating such patterns is necessary to provide ample data for subsequent data analysis tasks and ML. Especially ML needs a variety of behaviors for accurate predictions that might rarely occur in day-to-day usage scenarios.

Digital Twin: serves as an interface to the production and the canary clusters. DT collects data from both clusters, and monitors *and* controls the clusters. Further, it uses ML to extract models from data. The models describe aspects of the clusters, e.g., how the request arrivals affect CPU utilization, how long the deployment of pods takes, how the request arrival rate changes over time, etc. DT also provides control over the production cluster via a Translator that turns high-level commands into configurations and applies them.

Simulation: evaluates different configurations and estimates their impact on resource consumption and QoS parameters. It uses the behavioral models from DT to mimic the cluster as closely as possible. For example, the simulation exploits application profiles and usual request arrivals to simulate how a specific HPA setting might affect resource consumption and QoS. Using simulations instead of testbeds allows KAPETÁNIOS to concurrently evaluate multiple configurations.

`HPAEvolver`: searches for the configurations that minimize the resources a cluster uses while satisfying QoS requirements. The `HPAEvolver` uses simulations to evaluate different HPA configurations and selects the best one. It then utilizes the DT to apply the new configuration. KAPETÁNIOS can trigger the `HPAEvolver` automatically, e.g., if KAPETÁNIOS detects a change in the request pattern, or through external triggers, e.g., through a scheduled update of the application.

IV. DEMO SCENARIO

Fig. 2 shows a screenshot of the demo. The demo showcases the automatic tuning of the HPA by the `HPAEvolver`. For this, the demo runs a dummy web application. In the app, each request occupies a single CPU core for a given amount of time (request processing time).

The illustrated use-case is a change in the application's request processing time: the application running in the production cluster is updated, changing a requests' processing time, and making the HPA configuration inefficient. The demo has four application versions that the user can select with the slider in the upper left corner. Each version corresponds to a different request processing time (upper left in Fig. 2). The demo proceeds in three steps. For each step, the demo highlights the involved components in the left of Fig. 2.

Firstly, KAPETÁNIOS uses the canary cluster together with the load generator to generate measurements on the resource consumption of the updated application. The measurements are collected by the DT, where the ML engine learns a new profile from the collected data. The demo shows the mismatch between the old profile (orange), the actual cluster utilization (blue), and how the new profile (green) correctly predicts the utilization (upper right of Fig. 2). The demo uses the Random Forest Regressor from the sci-kit learn library [4].

Secondly, the `HPAEvolver` employs the simulation, the behavior models, and historical data on request patterns from the DT to optimize the HPA configuration. The simulation initializes itself with the current configuration of the production cluster. The `HPAEvolver` then updates the parameters that the simulation should evaluate. The simulation reports the expected QoS and pod resources over time of the cluster to `HPAEvolver`. `HPAEvolver` selects the best configuration and uses the DT to apply it on the cluster.

Finally, the demo deploys the updated application to the production cluster, and shows the resulting QoS, and pod resources of the application. Further, the demo compares the values with bar-plots against a scenario in which the configuration is not changed to showcase the benefits of adapting the HPA's scaling behavior. Fig. 2 shows examples at the bottom: whereas the average request completion time remains unchanged at ~ 0.07 s, the used pod seconds decreases by $\sim 37\,\%$, i.e., from ~ 476 s to ~ 298 s.

V. RELATED WORK

Tamiru et al. [2] show that scaling performance depends heavily on the workload and that using multiple node pools increases performance. Toka et al. [3] design an ML-based autoscaler. It predicts the request arrival rate for the next minute, and demonstrates that with a slightly higher resource over-provisioning more requests can be served compared to the default K8s HPA. Libra [5] is an adaptive autoscaler for K8s. It scales horizontally until reaching a predefined maximum number of pods. Afterwards, it behaves as a vertical scaler and predicts a CPU limit per pod. KubeKlone [6] uses a DT to model and simulates resource management methods for microservice-based applications, and abstracts infrastructure details by providing high-level interfaces. In contrast to previous work on adaptive HPA scaling, KAPETÁNIOS learns a model of the applications' resource consumption and uses the learned models to realistically model the cluster's behavior in simulations. KAPETÁNIOS then uses the simulations to autonomously update the cluster's configuration.

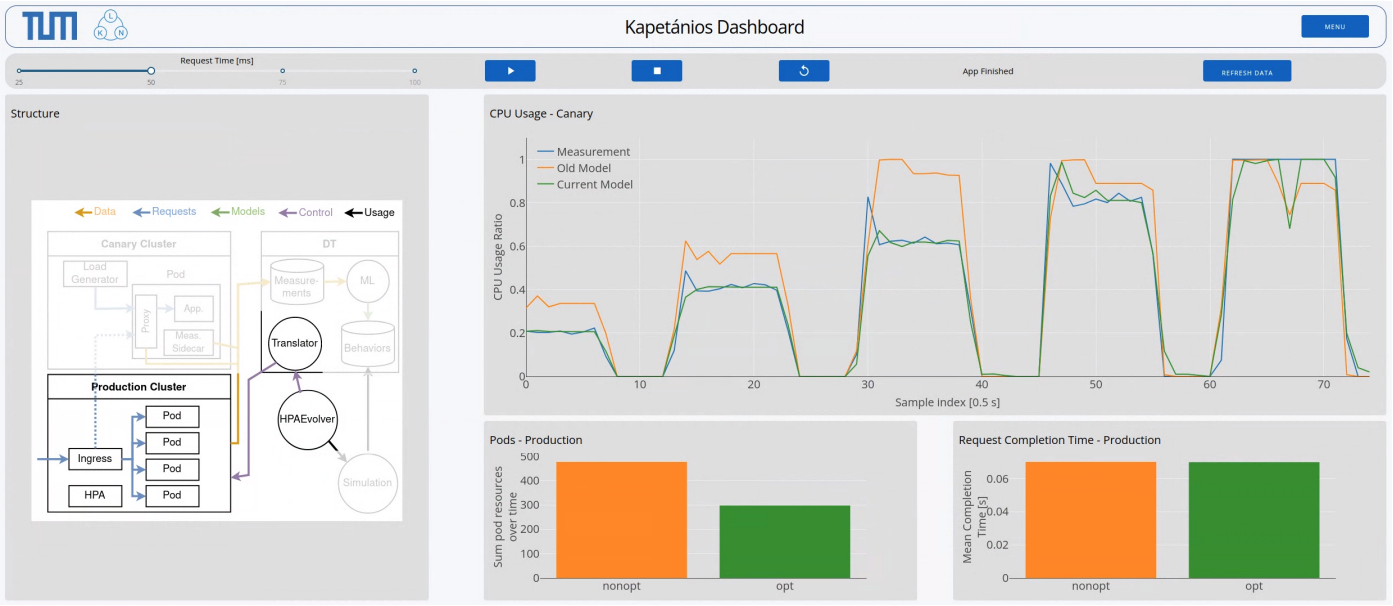

Fig. 2: Demo GUI. **Left**: Structure of KAPETÁNIOS, indicating the demo's state. **Upper right:** Mismatch between new and old ML model. **Bottom right:** KPIs for old and optimized configuration.

Recently, DTs have gained attention as a facilitator to close knowledge gaps for operators [7], [8]. For instance, Hui et al. [9] observe that packet-level network simulators are slow and thus not adequate to use in digital network twins. Instead, they observe that Neural Networks can model complex behaviors and suggest to model the impact of various factors on performance. The authors identify requirements for the performance evaluation to be useful in DTs. Sun et al. [10] try to assure service level agreements (SLAs) via a DT. They use information from the physical network to make decisions. However, it is unclear what the network looks like, and how data is retrieved. ChaosTwin [11] uses fuzzing to optimize cloud-based services, specifically, to find fault tolerant locations of VMs. Zhang et al. [12] provides fast performance estimation for data center networks at scale, based on packet-level simulations and ML. KAPETÁNIOS does not focus on establishing a novel concept of DTs. Instead, this demo shows a new avenue of how DTs can help the automation of next generation networks.

VI. CONCLUSION

This demo presents KAPETÁNIOS. KAPETÁNIOS automates the configuration of Kubernetes (K8s) clusters by combining Canary testing, digital twinning, ML, and discrete event simulations in a novel way. The demo shows how KAPETÁNIOS adapts the configuration of a K8s cluster in real-time to a changing behavior of the application, decreasing requested pod seconds by $\sim$37 % while keeping request latency low.

REFERENCES

[1] 5G Core (5GC) network: Get to the core of 5G. [Online]. Available: https://www.ericsson.com/en/core-network/5g-core/en/core-network/5g-core

[2] M. A. Tamiru, J. Tordsson, E. Elmroth, and G. Pierre, "An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud," in *CloudCom 2020*. Bangkok, Thailand: IEEE, 2020, pp. 17–24.

[3] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine Learning-Based Scaling Management for Kubernetes Edge Clusters," *IEEE TNSM*, vol. 18, no. 1, pp. 958–972, Mar. 2021.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] D. Balla, C. Simon, and M. Maliosz, "Adaptive scaling of Kubernetes pods," in *IEEE/IFIP NOMS 2020*, Apr. 2020, pp. 1–5.

[6] A. Bhardwaj and T. A. Benson, "KubeKlone: A Digital Twin for Simulating Edge and Cloud Microservices," *Asia-Pacific Workshop on Networking (APNet 2022)*, p. 7, 2022.

[7] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital Twin: Enabling Technologies, Challenges and Open Research," *IEEE Access*, vol. 8, pp. 108 952–108 971, 2020.

[8] M. M. Rathore, S. A. Shah, D. Shukla, E. Bentafat, and S. Bakiras, "The role of ai, machine learning, and big data in digital twinning: A systematic literature review, challenges, and opportunities," *IEEE Access*, vol. 9, pp. 32 030–32 052, 2021.

[9] L. Hui, M. Wang, L. Zhang, L. Lu, and Y. Cui, "Digital Twin for Networking: A Data-driven Performance Modeling Perspective," *CoRR*, vol. abs/2206.00310, 2022, arXiv: 2206.00310. [Online]. Available: https://doi.org/10.48550/arXiv.2206.00310

[10] X. Sun, C. Zhou, X. Duan, and T. Sun, "A digital twin network solution for end-to-end network service level agreement (sla) assurance [version 1; peer review: awaiting peer review]," *Digital Twin*, vol. 1, no. 5, 2021.

[11] F. Poltronieri, M. Tortonesi, and C. Stefanelli, "Chaostwin: A chaos engineering and digital twin approach for the design of resilient it services," in *CNSM 2021*, 2021, pp. 234–238.

[12] Q. Zhang, K. K. W. Ng, C. W. Kazer, S. Yan, J. Sedoc, and V. Liu, "Mimicnet: fast performance estimates for data center networks with machine learning," in *ACM SIGCOMM' 2021*, 2021, pp. 287–304.