Cloud-Native Scalability Benchmarking with Theodolite: Applied to the TeaStore Benchmark

Sören Henning, Benedikt Wetzel, Wilhelm Hasselbring
{soeren.henning@email, stu126940@mail, hasselbring@email}.uni-kiel.de
Software Engineering Group, Kiel University, Germany

Abstract

Theodolite is a framework for benchmarking the scalability of cloud-native applications such as microservices. It automates deployment and monitoring of a cloud-native application for different load intensities and provisioned cloud resources and assesses whether specified service level objectives (SLOs) are fulfilled. Provided as a Kubernetes Operator, Theodolite integrates with the cloud-native ecosystem and runs existing deployment configurations of various systems-under-test, load generators, and benchmarks. We give a presentation on Theodolite and exemplify its usage by benchmarking the scalability of the TeaStore microservice reference application.

1 Introduction

Over the last decade, *cloud-native* became a preferred way of designing, implementing, and operating large-scale software systems [3]. Microservice architectures are a pattern, particularly suited for building cloud-native applications [2]. Supported by declarative APIs, immutable infrastructure, and containerization, individual microservices are independently scalable [4]. In fact, scalability is often mentioned as a key driver for adopting microservices and cloud-native architectures [5, 7, 8]. Despite this, however, we found that research is lacking a commonly accepted method to benchmark scalability [14]. Since benchmarking is an essential empirical standard in software engineering research [11], such methods and tools are required to benchmark scalability of cloud-native applications.

After presenting our roadmap toward efficient scalability benchmarking [10] and advocating Kubernetes Operators for cloud-native benchmarking [13] in the previous two editions of SSP, we now give a demonstration of Theodolite.[1] Implemented as a Kubernetes Operator, Theodolite automates scalability benchmarking of cloud-native applications, running in Kubernetes. It is open-source research software and can easily be installed via Helm. To exemplify Theodolite's usage, we benchmark the TeaStore [6], a microservice reference application for benchmarking, modeling, and resource management research.

[1] `https://www.theodolite.rocks`

2 Theodolite's Benchmarking Method

Theodolite adopts established definitions of scalability in cloud computing for its benchmarking method [14]. It quantifies scalability by running isolated experiments for different load intensities and provisioned resource amounts, which assess whether specified SLOs are fulfilled. Two metrics are available: The *resource demand* metric describes how the amount of minimal required resources evolves with increasing load intensities, while the *load capacity* metric describes how the maximal processable load evolves with increasing resources.

The terms load, resources and SLOs are consciously kept abstract as Theodolite leaves it to the benchmark designer to define what type of load, resources, and SLOs should be evaluated. For example, horizontal scalability can be benchmarked by varying the amount of Kubernetes Pods, while vertical scalability can be benchmarked by varying CPU and memory constraints of Pods.

To balance statistical grounding and time-efficient benchmark execution, Theodolite comes with different heuristics for evaluating the search space of load and resource combinations. Other configuration options include the number of repetitions, the experiment and warm-up duration, as well as the amount of different load and resource values to be evaluated.

Theodolite distinguishes between benchmarks and their executions. Benchmarks define the deployment of the system-under-test (SUT) and the load generator as well as supported load dimensions, resource dimensions, and SLOs. Executions configure the experimental setup of a single benchmark execution, including options such as experiment durations and repetitions, the scalability metric, or the search heuristic. Both benchmarks and executions are defined in YAML files (Kubernetes custom resources), which can be deployed to the Kubernetes API, from where they are picked up by Theodolite [13].

3 Benchmarking the TeaStore's Scalability with Theodolite

Theodolite aims to support scalability benchmarking of arbitrary cloud-native applications without making Theodolite-specific adjustments to the SUT and with-

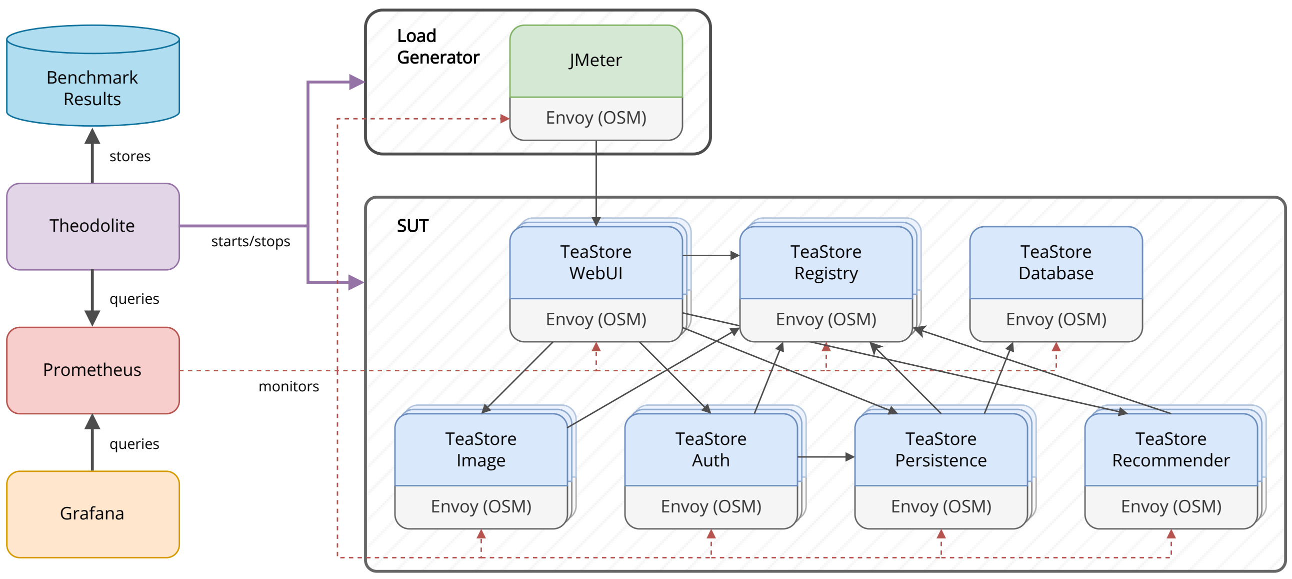

Figure 1: Benchmark deployment of the Theodolite stack, the TeaStore as SUT, and the JMeter load generator.

out SUT-specific adjustments to Theodolite. This includes real-world systems, publicly available benchmarks, and load generator tools. We demonstrate how Theodolite can benchmark third-party applications by benchmarking the TeaStore [6], a microservice reference application consisting of six microservices and a MariaDB database. It resamples a web shop for tea, allowing customers, for example, to browse the shop catalog, receive product recommendations, or place orders.

3.1 Benchmark Design

A Theodolite benchmark defines both the SUT and the load generator as references to Kubernetes resource files, specifying their deployments. Fig. 1 shows the overall benchmarking deployment, consisting of the TeaStore as SUT, JMeter as load generator, and the Theodolite stack. The latter consists of the Theodolite Kubernetes operator itself, a Prometheus instance that collects and manages performance metrics of the SUT, and a Grafana instance for benchmark observability. Finally, benchmark results are written to a Kubernetes volume, from where they can be further analyzed and visualized with Jupyter notebooks.

Supported load and resource dimensions are defined as transformation functions on these resource files. When running a benchmark, Theodolite uses these functions to adjust the SUT and the load generator deployments for different load and resource values, before starting and stopping these deployments.

A benchmark defines SLOs as PromQL queries associated with aggregation functions and a threshold. For each evaluated load-resource combination, Theodolite uses these queries to request monitoring data from Prometheus and checks the aggregated data against the defined threshold. Our Theodolite benchmark for the TeaStore is defined as follows:

SUT The TeaStore comes with Kubernetes files, which we can directly use in our benchmark. While the TeaStore integrates application-level monitoring with Kieker [1, 9], we require higher-level metrics such as latency of requests between services. Therefore, we deploy the TeaStore along with Open Service Mesh (OSM). OSM injects an *Envoy* proxy to each TeaStore service, which monitors incoming and outgoing network traffic and expose corresponding performance metrics in Prometheus' data format. To actually see scaling effects, we restrict the containers of the WebUI, Image, Auth, and Recommender microservices to 0.5 CPU cores and 1 GB memory. For demonstration purposes, we do not use separate nodes for the SUT, the load generation, and experiment controlling. However, this could easily be achieved by configuring Kubernetes node affinities.

Load Generator For generating load on the TeaStore, we deploy JMeter within our cluster and use the "browse" profile [6], provided as part of the TeaStore.

Load Dimension The load with which we scale is the number of concurrent users configured in JMeter. Each user creates a sequence of requests to the WebUI service, resulting in approximately 25–30 requests per user and second.

Resource Dimensions We choose two types of resource scaling: 1) We scale the number of instances for each of the WebUI, Image, Auth, and Recommender services (benchmarking horizontal scalability). 2) We stick to one instance per service, but scale the amount of provided CPU cores and memory for each service (benchmarking vertical scalability).

SLO To consider a certain load intensity to be handleable by a certain amount of instances, we require

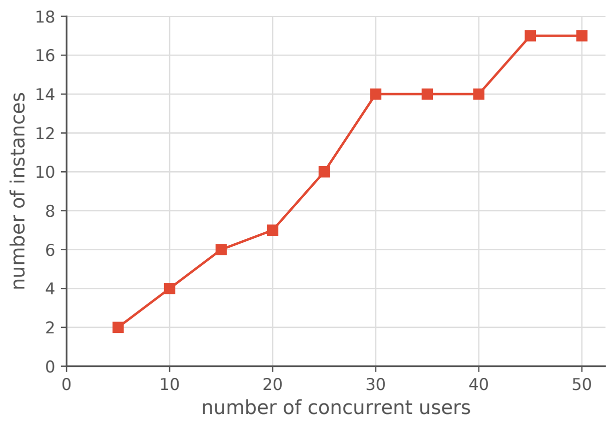

Figure 2: Results for benchmarking the horizontal scalability of the TeaStore.

that the 95th percentile latency of requests to the WebUI service does not exceed 200 ms.

3.2 Benchmark Execution

We run our experiments in a Kubernetes cluster, consisting of 5 bare-metal nodes with each 384 GB memory and 32 CPU cores. For all benchmark executions, we use the *load capacity* metric with Theodolite's lower bound, linear search strategy [14]. In manual experiments, we found that after 10 minutes of warmup the WebUI response latencies are quite stable. Thus, we run each experiment of a certain load intensity with a certain amount of resources for 20 minutes, while discarding the measurements of the first 10 minutes. For benchmarking horizontal scalability, we vary the number of pod instances from 1 to 20, while for vertical scalability we vary the pod's CPU resources from 0.5 to 8 cores and, proportionally, the pod's memory from 1 GB to 16 GB. We generate load with 5 to 50 concurrent users, resulting in a runtime of over 12 hours per benchmark execution.

3.3 Benchmark Results

Fig. 2 shows the results of our horizontal scalability benchmark. We can see that the required number of pods (for each microservice) scales approximately linearly with the amount of concurrent users. In our benchmark execution for vertical scalability, we observed that 5 concurrent users could be served by providing 1.5 CPU cores and 3 GB memory to each service. Higher amounts of concurrent users could not be handled, irrespective of the provisioned resources. While a detailed analysis is beyond the scope of this demonstration, we suspect that higher loads could still be processed by tuning the number of threads or accepting connections.

4 Conclusions and Outlook

In this paper, we demonstrated Theodolite's benchmarking method by benchmarking horizontal and vertical scalability of the TeaStore. We will publish our experimental setup as an example in Theodolite's online documentation and submit a pull request to the TeaStore's GitHub repository. Ongoing research in the Theodolite project includes exploring scalability along multiple load and resource dimensions. Besides "classical", REST-based microservices, we are also actively studying the scalability of event-driven microservices and provide corresponding benchmarks [12].

References

[1] A. van Hoorn, J. Waller, and W. Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". *International Conference on Performance Engineering.* 2012.

[2] A. Balalaie, A. Heydarnoori, and P. Jamshidi. "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". *IEEE Software* 33.3 (2016).

[3] D. Gannon, R. Barga, and N. Sundaresan. "Cloud-Native Applications". *IEEE Cloud Comput.* 4.5 (2017).

[4] W. Hasselbring and G. Steinacker. "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce". *International Conference on Software Architecture.* 2017.

[5] N. Kratzke and P.-C. Quint. "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study". *Journal of Systems and Software* 126 (2017).

[6] J. von Kistowski et al. "TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research". *IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems.* 2018.

[7] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel. "The pains and gains of microservices: A Systematic grey literature review". *Journal of Systems and Software* 146 (2018).

[8] H. Knoche and W. Hasselbring. "Drivers and Barriers for Microservice Adoption – A Survey among Professionals in Germany". *Enterprise Modelling and Information Systems Architectures (EMISAJ) – International Journal of Conceptual Modeling* 14.1 (2019).

[9] W. Hasselbring and A. van Hoorn. "Kieker: A monitoring framework for software engineering research". *Software Impacts* 5 (2020).

[10] S. Henning and W. Hasselbring. "Toward Efficient Scalability Benchmarking of Event-Driven Microservice Architectures at Large Scale". *Softwaretechnik-Trends* 40.3 (2020). Symposium on Software Performance.

[11] W. Hasselbring. "Benchmarking as Empirical Standard in Software Engineering Research". *Evaluation and Assessment in Software Engineering.* 2021.

[12] S. Henning and W. Hasselbring. "Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures". *Big Data Research* 25 (2021).

[13] S. Henning, B. Wetzel, and W. Hasselbring. "Reproducible Benchmarking of Cloud-Native Applications With the Kubernetes Operator Pattern". *Symposium on Software Performance.* 2021.

[14] S. Henning and W. Hasselbring. "A Configurable Method for Benchmarking Scalability of Cloud-Native Applications". *Empirical Software Engineering* 27.6 (2022).