Adaptable TeaStore*

Simon Bliudze
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Giuseppe De Palma
Alma Mater Studiorum - Università di Bologna, Bologna, Italy
Olas Team, INRIA, Sophia Antipolis, France

Saverio Giallorenzo
Alma Mater Studiorum - Università di Bologna, Bologna, Italy
Olas Team, INRIA, Sophia Antipolis, France

Ivan Lanese
Alma Mater Studiorum - Università di Bologna, Bologna, Italy
Olas Team, INRIA, Sophia Antipolis, France

Gianluigi Zavattaro
Alma Mater Studiorum - Università di Bologna, Bologna, Italy
Olas Team, INRIA, Sophia Antipolis, France

Brice Arléon Zemtsop Ndadji
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Modern cloud-native systems require adapting dynamically to changing operational conditions, including service outages, traffic surges, and evolving user requirements. While existing benchmarks provide valuable testbeds for performance and scalability evaluation, they lack explicit support for studying adaptation mechanisms, reconfiguration strategies, and graceful degradation. These limitations hinder systematic research on self-adaptive architectures in realistic cloud environments.

To cover this gap, we introduce *Adaptable TeaStore*, an extension of the renowned TeaStore architecture that incorporates adaptability as a first-class design concern. Our extension distinguishes between mandatory and optional services, supports multiple component versions—with varying resource requirements and functionality levels—considers the outsourcing of functionalities to external providers, and provides local cache mechanisms for performance and resilience. These features enable the systematic exploration of reconfiguration policies across diverse operational scenarios.

We discuss a broad catalogue of reference adaptation scenarios centred around Adaptable TeaStore, useful to evaluate the ability of a given adaptation technology to address conditions such as component unavailability, cyberattacks, provider outages, benign/malicious traffic increases, and user-triggered reconfigurations. Moreover, we present an open-source implementation of the architecture with APIs for metrics collection and adaptation triggers, to enable reproducible experiments.

1 Introduction

In modern cloud software architectures, the ability to adapt services to changing conditions is increasingly essential. Distributed systems must remain operational despite fluctuating workloads, partial fail-

*Work partially supported by French ANR project SmartCloud ANR-23-CE25-0012, by PRIN project FREEDA (CUP: I53D23003550006) funded by the frameworks PRIN (MUR, Italy) and Next Generation EU, by project PNRR CN HPC - SPOKE 9 - Innovation Grant LEONARDO - TASI - RTMER funded by the NextGenerationEU European initiative through the MUR, and by INdAM - GNCS 2024 project MARVEL, code CUP E53C23001670001.

Submitted to:
WACA 2025

ures, and evolving user requirements. While state-of-the-art approaches often adopt the microservice [5] or serverless [10] paradigms, the architectural benchmarks available today largely emphasise performance and scalability rather than explicit support for adaptation. This gap limits the capacity of researchers and practitioners to study reconfiguration strategies, graceful degradation, and the substitution of external providers in a systematic and comparable way.

TeaStore [16] has emerged as a reference benchmark for microservice-based systems. The benchmark provides a realistic yet tractable environment for studying performance trade-offs, resource management, and deployment variability. However, TeaStore's original design does not explicitly account for adaptation, leaving out many scenarios that arise in real-world deployments where applications must cope with service outages, degraded performance, cyberattacks, or surges in demand. Addressing these limitations requires a benchmark that supports both mandatory and optional services, multiple implementation variants, and external dependencies, together with mechanisms for fallback and reconfiguration.

To cover this gap, we propose *Adaptable TeaStore*, a variant of TeaStore that includes adaptability as a core feature. Adaptable TeaStore substantially extends the architecture of the original TeaStore benchmark with characteristics that support the reconfiguration of the application under varying operational conditions. Central to this design is the distinction between mandatory services, which guarantee baseline functionality, and optional ones, which can be selectively enabled or disabled to trade features for resilience or efficiency. Several services are offered in multiple "flavors", ranging from lightweight to fully featured implementations, thereby supporting fine-grained adaptation to resource availability and quality-of-service constraints. Moreover, functionalities, such as data persistence, may be outsourced to external providers, reflecting modern cloud deployment practices, while local cache mechanisms and static fallback solutions ensure continuity in the presence of provider failures or disconnections. Taken together, these characteristics make Adaptable TeaStore a flexible benchmark for studying reconfiguration strategies, graceful degradation, and provider substitution within a realistic microservice setting.

Looking at the state of the art, Adaptable TeaStore acts as a bridge between the landscape of microservice and cloud benchmarks and that of the adaptation-centric ones, complementing existing microservice-style performance-oriented benchmarks such as Acme Air [1], Sock Shop [17], and DeathStarBench [6] and adaptation-oriented ones, like mRUBiS [15].

The paper is structured as follows. We begin, in Section 2, by recalling the original TeaStore architecture, emphasising its strengths as a research-grade reference system and its limitations in the context of adaptive behaviour. Moving to Section 3, we present our extensions, which enrich TeaStore's architecture with configurable and outsourceable services, service flavours offering different levels of functionality, and local cache mechanisms that ensure graceful degradation when external dependencies fail. Building on these extensions, we discuss several configuration levels that capture different operating modes, from minimal barebone deployments to full-featured configurations relying on external providers. To demonstrate the range of adaptation situations Adaptable TeaStore can support, we present a catalogue of adaptation scenarios that stress a given adaptation technology's ability to manage diverse operational conditions. These scenarios include coping with unavailable data sources, mitigating cyberattacks on external services, handling benign or malicious traffic surges, and responding to user-triggered reconfigurations. Together, these scenarios provide a systematic basis for evaluating adaptive mechanisms, resilience strategies, and their trade-offs. As a second contribution, in Section 4, we present an implementation of Adaptable TeaStore that we provide as open source code and as images on Docker Hub to be used as an experimentation platform for the adaptation of microservice-based applications. We describe the interfaces provided for querying metrics and for triggering adaptation actions. In Section 5, we position our contribution w.r.t. related work. We conclude, in Section 6, by drawing closing remarks on the broader implications of Adaptable TeaStore and outlining community-contributed directions for

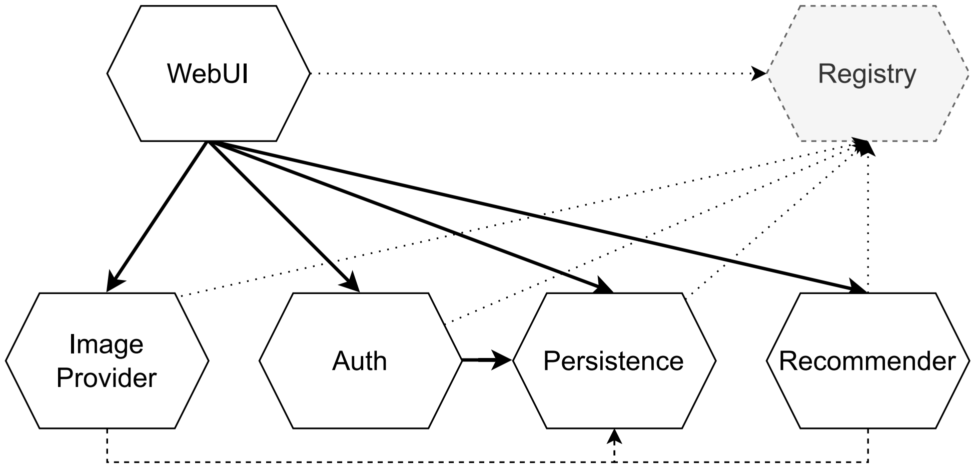

Figure 1: TeaStore Architecture, adapted from von Kistowski et al. [16]

future extensions.

2 TeaStore

TeaStore [16] is a well-known reference benchmark for microservice-based systems. It involves 5 different services: WebUI, Auth, Persistence, Image Provider, and Recommender.

In Figure 1, we show an overview of the TeaStore architecture. The WebUI service is the entry point for the user, and it is the main service that interacts with all other services. The Persistence service is the layer on top of the database. It is used by the WebUI to retrieve and store data and by the Authentication service to retrieve user data, which is then passed to the WebUI. On startup, the Persistence service populates the database with data. The Image provider and Recommender both connect to a provided interface at the Persistence service. However, this is only necessary on startup (dashed lines). The Image provider must generate an image for each product, whereas the Recommender needs the current order history as training data. Once running, only the Authentication and the WebUI access, modify, and create data using the Persistence. The Registry service will be described in Section 4.

We conclude this section by summarising the description of TeaStore's components from the benchmark's original proposal [16].

The WebUI service manages the user interface and contacts all the other services to retrieve and display data. It compiles and serves Java Server Pages (JSPs) with the categories, products, recommendations, and images. It performs preliminary validation on user inputs before sending them to the Persistence service.

The Image Provider service handles the serving and resizing of images in various sizes for the WebUI. It uses a cache to optimise performance, resizing and caching images if necessary. The system employs a least-frequently-used caching strategy to reduce resource demand, and response time depends on whether an image is already cached or needs resizing.

The Authentication service verifies user login and session data, using BCrypt for login verification and SHA-512 hashes for session validation. Session data include shopping cart content, login status, and order history. The performance depends on the volume of session data. The service remains stateless since all session data is passed to the client.

The Recommender service provides product recommendations based on items other customers bought, on the products in a user's current shopping cart, and on the product the user is viewing at the time. It maintains coherence between different instances by sharing training data (additional Recommender instances query existing instances for their training data-set). Several recommendation algorithms can be used, including two 'Slope One' variants and an order-based nearest-neighbourhood method, with different algorithms optimising for either memory or CPU performance. There is a fallback algorithm based on overall item popularity.

The Persistence service manages access and caching for the store's relational database, which stores data on products, categories, purchases, and users. There is an internal cache to improve scalability and reduce the load on the database. The cache is kept coherent across multiple Persistence service instances. All data inside the database itself is generated at the first start of the initial persistence instance.

3 Adaptable TeaStore

The original TeaStore architecture is static, lacking scenarios where the structure of the system needs to change due to failures and variations of environmental conditions or user requirements. To enhance the case study in this direction, we extend it with optional services and external dependencies. The latter kind are functionalities that the entity deploying the system to provide a service to clients, henceforth called the *company*, has no control over.

The Adaptable TeaStore architecture is shown in Figure 2. Below, we present our extensions to the original architecture and the proposed adaptation scenarios.

We first pinpoint some terms that we use in the description of the extension of the TeaStore architecture. Functionalities of an architecture are either *mandatory* or *optional*. Moreover, functionalities of a given architecture have different *provision* modalities: either by components managed by the company or by third parties. In the first case, we classify the functionality as offered by an *internal* component. Internal components can be either hosted on premises or deployed on Cloud. Otherwise, a third party offers that functionality as an *external* service. Some functionalities can be provided either through internal components or external services, and we call them *outsourceable*. Another dimension is that of *service flavours*, i.e., alternative implementations of a component that provides a given functionality and that one can deploy interchangeably.

Since (Adaptable) TeaStore adopts a microservice architecture, internal functionalities are provided by (micro)services. Therefore, below, we use the terms service and component interchangeably.

3.1 Extensions of the TeaStore Architecture

We categorise the functionalities of the Adaptable TeaStore as follows.

Mandatory and Optional Functionalities The WebUI, Image Provider, and Persistence services provide functionalities which are mandatory for the system to operate. The functionalities provided by the

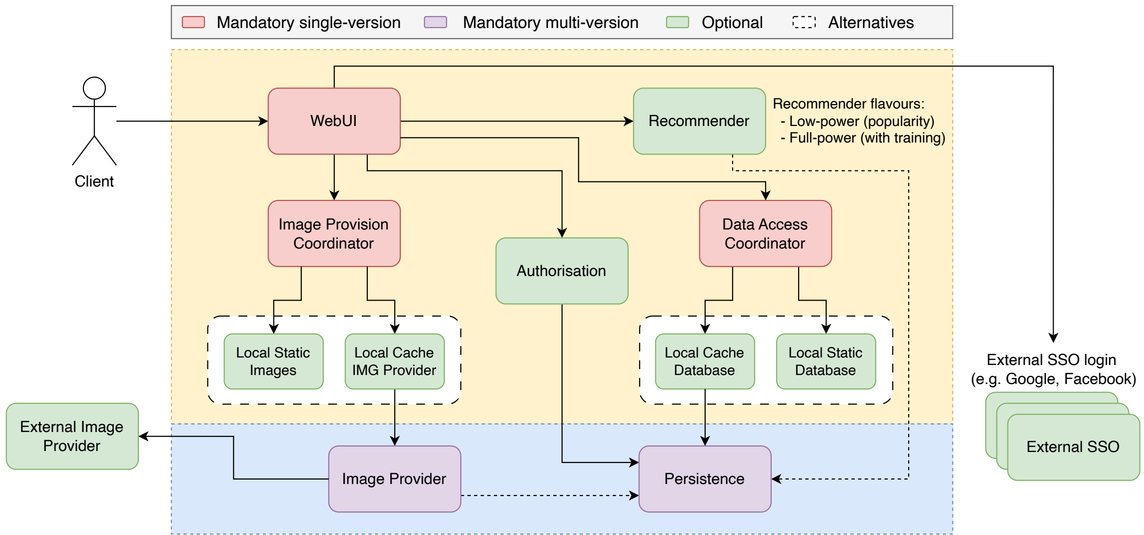

Figure 2: Adaptable TeaStore Architecture (functionalities shown within the coloured areas are internal, those in the blue area are outsourceable, in the yellow area—non-outsourceable (cf. Table 1))

Recommender and by the Authentication service are both optional. Moreover, we introduce in Adaptable TeaStore the optional functionality of Single Sign-On (SSO) authentication. Note that the two authentication techniques are not mutually exclusive. Indeed, while the two functionalities provide the same outcome (authenticating users) their modalities significantly change: the one provided by the Authentication service involves two parties, a client and a server, where the latter has the necessary information to authenticate the user, while SSO involves three parties, a client, a relying party, and an identity provider, where the relying party trusts the identity provider to give access to clients. An Adaptable TeaStore instance can work without either of the functionalities, allowing users to anonymously browse products, but it can also work with either or both functionalities available, since they manage the authentication of different sets of users.

Outsourceable Functionalities The functionalities provided by the Image Provider and Persistence services can come either from an external provider, e.g., Database-as-a-Service (DaaS) platforms for the Persistence functionalities (e.g., Firebase, Supabase), or from the respective homonymous internal services—this scenario includes also multi-tenant deployments where multiple applications of the same company share the same services.

External Functionalities As mentioned above, we introduce the possibility for users to authenticate through Single Sign-On as a coexisting alternative to the internal Authorisation service. This option reflects typical applications that integrate third-party login options (e.g., Google, Facebook, etc.).

Service Flavours In Adaptable TeaStore there are three components, each coming in two flavours. The Image Provider comes either in its full-featured flavour, named after the service itself, which can dynamically resize images according to request parameters (e.g., for smaller or larger screens) or its lightweight alternative, which provides only static images, called *Local Static Images* service. The Recommender

Table 1: Summary of the characteristics of the functionalities/components found in Adaptable TeaStore (a bullet • indicates that the element has the given property, while a circle ○ denotes its absence)

Functionality	Mandatory	Provision Modality	Outsourceable	# Flavours	Local Cache
WebUI	•	Internal	○	1	○
Image Provision Coordinator	•	Internal	○	1	○
Data Access Coordinator	•	Internal	○	1	○
Persistence	•	Internal	•	2	•
Image Provider	•	Internal	•	2	•
Recommender	○	Internal	○	2	○
Authorisation	○	Internal	○	1	○
SSO	○	External	-	-	○
External Image Provider	○	External	-	-	○

service has either a *Full-power* flavour, which runs a resource-intensive recommendation algorithm, or a *Low-power* one, which requires less resources but might provide less accurate recommendations—the latter corresponds to the fallback algorithm based on overall item popularity. The Persistence service also has two flavours: the one from the original TeaStore, named after itself, and a static version that does not support writing new data (e.g., it allows users to browse products but not to buy them), called *Local Static DB*.

Optional Local Cache Services We further extend the original TeaStore architecture by introducing *local* (internal) *cache services*, e.g., to buffer interactions with outsourced or external services. Performance-wise, these services can reduce latency and minimise redundant requests. Functionality-wise, the cache services can reply using cached information when the corresponding outsourced/external functionality is not available. We consider two such optional cache services, one for the Image Provider and one for the Persistence service, respectively called Local Cache Image and Local Cache DB.

Coordinators To avoid modifying the WebUI code to select, e.g., between Local Static Images and Local Cache Image Provider (similarly for database access), we introduce two new coordinator services—*Image Provision Coordinator* and *Data Access Coordinator*. The role of these services is to channel the WebUI requests according to the current configuration. In the absence of optional local cache services, the Image Provision Coordinator (respectively the Data Access Coordinator) interacts directly with the Image Provider (respectively Persistence).

An overview of the characteristics of the functionalities and components of the Adaptable TeaStore architecture, we summarise them in Table 1.

Configuration Levels Since the above dimensions allow one to identify multiple Adaptable TeaStore instance configurations, we fix three reference configuration levels.

1. **Barebone** contains only the mandatory services, to offer minimum functionality for anonymous users, in their low-level flavours, if any, i.e., the WebUI, the Local Static Images flavour of Image Provider, and the Local Static DB of Persistence.

2. **Barebone + Recommender** adds, along the services of the Barebone configuration, the Recommender using its Low-Power flavour.

3. **Full** contains Persistence and Image Provider in their high-level flavours, possibly deployed in another region than the rest of the other components (these services could be used in other applications from the same company). As an alternative, these functionalities can be outsourced from an external provider (e.g., Supabase). The configuration also includes the Authorisation service, the Recommender, which can be used in its Full-Power flavour, and an external SSO functionality. Either when outsourced or deployed in a different region, the Local Cache Image and Local Cache DB are used to buffer the requests to the related services. Note that, if both the SSO functionality and the Authorisation service are not available, the Recommender Full-Power flavour cannot be used due to missing user data.

3.2 Adaptation Scenarios

We now move to present a set of scenarios designed to evaluate a system's ability to adapt across various operational challenges. Each scenario tests specific capabilities of a given technology for managing adaptation, from handling infrastructure failures to responding to security threats and handling resource constraints. The scenarios cover different basic aspects. They can be combined to form sophisticated multi-cause scenarios for deeper evaluation of adaptation capabilities of the system.

3.2.1 Database Unavailable

The system is deployed in a barebone configuration with only local services. The queries to the local database start to timeout and the WebUI becomes unresponsive. The WebUI adapts to avoid querying the database and show a maintenance message while the system restarts the database service. Once the database is back online the WebUI resumes normal operation.

3.2.2 Cyberattack on External Providers

The system is deployed in the full configuration with an external provider for the Image Provider and Persistence services as well as authorisation through SSO. The external providers detect an attack (e.g., a privilege escalation) and have to take down and restart the provided functionalities. In response to this unavailability, due to a security threat, the system enables the Local Static Images and the Local Static DB services. The WebUI adapts by disabling the authentication functionality, such as new logins and registrations. At some point, the deployed external functionalities are back online and the system returns to the full configuration, as before the attack.

3.2.3 Cloud Provider Outage

The system is deployed in the full configuration. The services running within remote regions go offline due to outages (e.g., energy failure). The requests from the WebUI to these services start to timeout. The system adapts by first switching the configuration to barebone, to be able to provide the essential functionalities, while the unavailable services are being redeployed at a different provider. After the redeployed services are online, the system switches back to the full configuration.

As an example, the system is up and running with an instance of the Image Provider deployed in a different region than the rest of the architecture. The WebUI fetches images from the Image Provider via the Local Cache Image provider. A user accesses a product page and the WebUI tries to fetch the image from the local cache service. The local cache service does not have the image, it tries to fetch the image from the remote Image Provider, but the request times out. The system adapts the configuration by switching to the Local Static Images service. Meanwhile, the system deploys the remote service on a different provider and switches back to the full configuration when the service is online.

3.2.4 Sudden Traffic Increase

We propose the following three scenarios distinguishing among the benign and malicious causes for the traffic increase.

Benign Traffic Increase Incoming traffic towards the WebUI increases significantly due to a genuine increase in users. Since the traffic increase is benign, the system must adapt by scaling out the services to handle the increased load or taking other measures to the same purpose (e.g., switch to low-power versions of the services).

As an example, on a system with the Recommender service in Full-power mode, the Recommender service is under heavy load due to the sudden increase in user requests. The service quality degrades because the response time increases significantly (defined by a user-specified QoS threshold). The system adapts the configuration by switching the Recommender service to Low-power mode, which uses the fallback algorithm to provide recommendations based on item popularity.

Malicious Traffic Increase Incoming traffic towards the WebUI increases significantly due to a DDoS attack. The adaptations include deploying circuit breakers between services to prevent cascading failures, switch Authentication service to a more restrictive mode with additional verification, the Recommender switching to Low-power mode, and Local Cache services activating to reduce remote/external dependencies.

Conditional Handling of Traffic Increase Incoming traffic towards WebUI increases significantly. However, no explicit information is available about the reason for that: the increase can be either due to a genuine increase in the number of users or to a DDoS attack. The system must evaluate the situation and adapt accordingly. Notice that the decision and the adaptation need not be a mutually exclusive choice between the benign and malicious scenarios above, it can be a combination of the two.

3.2.5 DevOps Requirements Change

The system should be able to adapt to reconfiguration requests from DevOps. For instance, DevOps can decide to scale out a service or switch between different versions, thus increasing/decreasing the amount of resources available to it or modifying the features of the architecture (e.g., removing or adding services). This event could involve supporting additional external SSO login options or forcing a specific service (e.g., Recommender) flavour. Notice that the reconfiguration request might be incomplete, providing information only about the required changes but not necessarily about all their dependencies. The system should adapt by moving to a valid configuration, ideally without downtime.

For example, the administrator of the system requires switching between local and remote providers (e.g., moving the Persistence service from an on-premises to a on cloud location) with little to no downtime by supporting the smooth transition with local caching as a transition layer.

Table 2: REST API for metrics collection (HTTP method: GET)

Metric	**Endpoint**	**Providing service**
CPU Usage	/metrics/cpu	Core TeaStore services + Persistence
Memory Usage	/metrics/memory	Core TeaStore services + Persistence
Request details	/metrics/requests	Core TeaStore services + Persistence
Service status	/metrics/status	Core TeaStore services + Persistence
Service state	/metrics/state	Core TeaStore services + Persistence
Database status	/metrics/db	Persistence
Database response time	/metrics/db/responseTime	Persistence

Table 3: Metrics and adaptation actions exposed by the implemented Adaptable TeaStore services

Service	**Adaptation actions**
WebUI	OpenCircuitBreaker, CloseCircuitBreaker, DDoSAttackEventBroadcast, EnableMaintenanceMode, DisableMaintenanceMode
Recommender	OpenCircuitBreaker, CloseCircuitBreaker, HighPerformanceMode, LowPowerMode, NormalMode
Image Provider	OpenCircuitBreaker, CloseCircuitBreaker, DisableExternalImageProvider, EnableExternalImageProvider
Persistence	OpenCircuitBreaker, CloseCircuitBreaker, DatabaseAvailableEventBroadcast, DatabaseUnavailableEventBroadcast, EnableCache, DisableCache
Authentication	OpenCircuitBreaker, CloseCircuitBreaker
Registry	OpenCircuitBreaker, CloseCircuitBreaker

4 Experimentation Platform

We have extended the original TeaStore implementation [13] towards providing an implementation of the Adaptable TeaStore specification from Section 3.[1] The goal is to provide a reference implementation for experimental validation of tools and techniques for managing the adaptation in microservice-based applications.

Adaptable TeaStore relies on the same architecture as the original TeaStore: in addition to the core services—WebUI, Recommender, Image Provider, Persistence, and Authentication—discussed in the previous sections, it comprises a Registry service (see Figure 1) and a service running a MariaDB database (not shown in the figure). We have modified the implementations of the core TeaStore services and of the Registry service to allow collection of metrics and triggering the adaptation actions.[2]

Provided metrics are summarised in Table 2. All the core services provide the CPU usage, memory usage, request count, service state, and service status. The service state is always `RUNNING` for all services except Recommender and Image Provider. The Recommender service has an additional state `TRAINING_DATA`, whereas Image Provider has the additional state `GENERATING_IMAGES`. The `status` request returns the last heartbeat date and time, and the hosting server id. Additionally, the Persistence service provides the database status and the database response time. The database status report comprises the database response time, network status, and the numbers of active connections and pending queries.

[1] The implementation code is accessible from a public repository. Docker images are provided on Docker Hub.
[2] Coordinators, local cache services and static flavours proposed in Section 3.1 will be implemented in future work.

Table 4: REST API for adaptation action execution (HTTP method: POST)

Action	Endpoint	Request parameters
Execute a single action	/adapt/single	Query parameter: `actionName` — The name of the action to execute. Example: `.../adapt/single?actionName=OpenCircuitBreaker`
Execute multiple actions	/adapt	Body (JSON): A list of action names to be executed. Example: `["OpenCircuitBreaker", "EnableCache"]`

It should be noted that any of the above requests can be used to determine whether a service is available or not and to measure its response time.

Provided adaptation actions are summarised in Table 3. All services implement the circuit breaker functionality. The three event broadcast actions allow sending push notifications to selected services upon the corresponding events: WebUI can notify Persistence, Recommender, Image provider, and Authentication about DDoS attacks; Persistence can notify WebUI and Recommender about database availability changes. Other actions reflect the specifications in Section 3.[3]

Both the metrics and the adaptation actions can be accessed through a RESTful API, via corresponding `GET` and `POST` actions. For the metrics collection, each service in the Adaptable TeaStore exposes a REST API with a list of available endpoints summarised in Table 2. For the adaptation actions, each service exposes two endpoints (see Table 4). The `/adapt/single` endpoint allows a single adaptation action to be executed by passing its name. The `/adapt` endpoint allows several adaptation actions to be executed sequentially by passing the list of their names.

The usage of the metrics and adaptation actions discussed above is illustrated by `AdaptiFlow` [18], a framework that provides an abstraction layer for microservice-based applications focusing on the Monitor and Execute phases of the MAPE-K loop.

5 Related Work

Several benchmarks and reference architectures have been proposed over the last decade to evaluate cloud-native and distributed systems, each with distinct traits and emphases.

Acme Air [1] is one of the earliest microservice-based benchmarks, originally designed as a sample web application with Java and Node.js implementations, and later adopted for studies of autoscaling and cloud elasticity [14, 9]. The benchmark primarily highlights performance evaluation and scaling of services, but it does not incorporate explicit adaptation strategies beyond autoscaling policies.

Sock Shop [17] was subsequently introduced as a demonstration system for microservice tooling, resilience patterns and observability practices. While Sock Shop provides a compact, retail-oriented application that has become widely used for tutorials and chaos engineering exercises, it is not designed with adaptation aspects such as feature toggling or graceful degradation.

TrainTicket [19] represents a further step in realism, providing a complete ticket booking system that emphasises complexity, distributed transaction management and large-scale testing, designed for evaluating engineering practices and system integration rather than adaptation.

As mentioned, TeaStore [16] is a research-grade reference application, explicitly designed to facilitate benchmarking, modelling and resource management studies, with carefully controlled variability in

[3]The external image provider is currently implemented by querying `https://ui-avatars.com`.

performance and deployment options. TeaStore's original definition does not specify adaptation scenarios, which we provide in this proposal.

DeathStarBench [6] broadened the scope of benchmark applications by introducing a suite of end-to-end cloud architectures spanning social networks, media services, e-commerce, and banking. The suite enables the study of fan-out patterns, QoS, scalability and hardware-software co-design implications. While diverse and realistic, DeathStarBench's benchmarks are fixed in functionality and offer no adaptation scenarios like functional degradation and provider variability.

Adaptable TeaStore positions itself within this context by combining the realism and familiarity of an established microservice benchmark architecture (TeaStore) with explicit, targeted adaptation scenarios. Like Acme Air, Sock Shop, TrainTicket and DeathStarBench, Adaptable TeaStore targets performance and scalability experiments, but it goes beyond these aspects by evaluating reconfiguration policies, graceful degradation, and provider substitution.

Leaving microservices, proposals worth mentioning are mRUBiS [15], SPEC Cloud [2], FaaSdom [12], BeFaaS [8], and SeBS [3].

mRUBiS is a component-based e-commerce system explicitly built to evaluate self-adaptation. mRUBiS targets self-healing and self-optimisation techniques and makes adaptation the central evaluation dimension through the integration of failure injection, utility functions, and reconfiguration tactics.

SPEC Cloud, focusing on the infrastructure level, defines benchmarking for elasticity, scalability and provisioning time in Infrastructure-as-a-Service environments. While SPEC Cloud's scope is cloud platforms (moving it closer to Adaptable TeaStore), the benchmark captures elasticity behaviour and adaptation at the resource level rather than at the application one.

More recently, adaptation benchmarks have been proposed for the context of serverless computing [11]. FaaSdom and BeFaaS offer benchmarks for modular workloads and application-centric cases to study elasticity and cost-performance trade-offs in serverless environments, while SeBS defines a cross-provider benchmark suite for serverless platforms, evaluating performance, efficiency, scalability, and reliability.

We envision future versions of Adaptable TeaStore to draw inspiration from these work. For example, it could integrate utility functions, failure injection, and reconfiguration tactics, as present in mRUBiS, encompass elasticity metrics and cost-awareness from SPEC Cloud, and consider workflow resilience, typical of serverless benchmarks.

Another direction to consider in future versions of Adaptable TeaStore is that of energy-awareness. Drawing inspiration from recent work on energy-aware self-adaptive systems [4, 7], the architecture could integrate power meters to monitor energy consumption at the service level, enabling fine-grained visibility into the energy footprint of individual components. The architecture would then support reconfigurations that drive the switching among service flavours based on their energy profiles during different workload patterns. In general, the integrating of energy profiling of service flavours with runtime adaptation mechanisms would enable researchers to systematically explore the trade-offs between energy efficiency, performance, and availability in cloud-native architectures.

6 Conclusion

In this work, we introduced Adaptable TeaStore, a novel extension of the widely adopted TeaStore reference architecture that introduces adaptation as a first-class concern. Unlike existing benchmarks, which primarily target performance and scalability, Adaptable TeaStore defines explicit scenarios regarding reconfiguration, graceful degradation, and provider substitution. Through the introduction of mandatory and optional services, outsourceable dependencies, multiple service flavours, and local fallback caches, our proposal supports the systematic exploration of adaptation mechanisms under realistic, diverse operating conditions.

The considered adaptation scenarios, ranging cyberattacks, cloud provider outages, benign and malicious traffic surges, and DevOps-driven reconfigurations, constitute a foundation for evaluating the effectiveness, robustness, and trade-offs of adaptive strategies across heterogeneous cloud environments. In this way, Adaptable TeaStore bridges a long-standing gap between performance-oriented benchmarks and adaptation-centric ones.

Looking ahead, we see Adaptable TeaStore as a *community benchmark* that can evolve with emerging challenges in adaptive cloud systems. Several research directions naturally follow. For instance, the integration of explicit utility functions, failure injection, and reconfiguration tactics to enable principled trade-off analyses between availability, cost, and quality of service. Moreover, one can extend the architecture to include other resource-aware adaptation policies, such as energy, advancing the study of sustainable computing. In addition, hybridising the architecture to encompass serverless workloads, workflow-driven services, and cross-provider federation would make Adaptable TeaStore as an even more suitable and comprehensive testbed for the next generation of cloud-native systems.

We envision that Adaptable TeaStore can, over time, become a reference point for the evaluation of adaptive cloud systems. While it is not intended as a definitive solution, its role as a common testbed may help consolidate empirical practices, inspired and evaluate theoretical approaches, and support a more systematic comparison of adaptation techniques. In this sense, Adaptable TeaStore contributes to the longer-term goal of grounding self-adaptive software research in reproducible and practically relevant experimentation.

We see this effort as a community-driven one, where members can contribute scenarios, extensions, and evaluation methodologies, turning Adaptable TeaStore into a shared experimental ground for reproducibility, comparability, and innovation. The long-term aim is to establish an evolving benchmark that not only reflects today's needs but also anticipates tomorrow's demands in the engineering of self-adaptive software architectures.

References

[1] ACME AIR PROJECT. Acme Air sample and benchmark. `https://github.com/acmeair/acmeair`, 2015. Accessed September 2025.

[2] BASET, S., SILVA, M., AND WAKOU, N. SPEC cloud™ IaaS 2016 benchmark. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017* (2017), W. Binder, V. Cortellessa, A. Koziolek, E. Smirni, and M. Poess, Eds., ACM, p. 423.

[3] COPIK, M., KWASNIEWSKI, G., BESTA, M., PODSTAWSKI, M., AND HOEFLER, T. SeBS: a serverless benchmark suite for function-as-a-service computing. In *Middleware '21: 22nd International Middleware Conference, Québec City, Canada, December 6 - 10, 2021* (2021), K. Zhang, A. Gherbi, N. Venkatasubramanian, and L. Veiga, Eds., ACM, pp. 64–78.

[4] DE MEDEIROS, H., MUÑANTE, D., CHABRIDON, S., BATISTA, C., AND CONAN, D. Adaptable TeaStore with energy consumption awareness: A case study. In *Post-proceedings of the Workshop on Adaptable Cloud Architectures (WACA 2025)* (2025), G. De Palma and S. Giallorenzo, Eds., this volume of EPTCS, Open Publishing Association.

[5] DRAGONI, N., GIALLORENZO, S., LLUCH-LAFUENTE, A., MAZZARA, M., MONTESI, F., MUSTAFIN, R., AND SAFINA, L. Microservices: Yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*. Springer, 2017, pp. 195–216.

[6] GAN, Y., ZHANG, Y., CHENG, D., SHETTY, A., RATHI, P., KATARKI, N., BRUNO, A., HU, J., RITCHKEN, B., JACKSON, B., HU, K., PANCHOLI, M., HE, Y., CLANCY, B., COLEN, C., WEN, F., LEUNG, C., WANG, S., ZARUVINSKY, L., ESPINOSA, M., LIN, R., LIU, Z., PADILLA, J., AND DELIMITROU, C. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019* (2019), I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds., ACM, pp. 3–18.

[7] GAZZA, S., AMADINI, R., BROGI, A., D'IAPICO, A., FORTI, S., GIALLORENZO, S., PLEBANI, P., PONCE, F., SOLDANI, J., VITALI, M., AND ZAVATTARO, G. A constraint-based approach to optimise QoS- and energy-aware cloud-edge application deployments. *ACM Trans. Internet Technol.* (July 2025).

[8] GRAMBOW, M., PFANDZELTER, T., BURCHARD, L., SCHUBERT, C., ZHAO, M. X., AND BERMBACH, D. BeFaaS: An application-centric benchmarking framework for FaaS platforms. In *IEEE International Conference on Cloud Engineering, IC2E 2021, San Francisco, CA, USA, October 4-8, 2021* (2021), IEEE, pp. 1–8.

[9] GRANCHELLI, G., CARDARELLI, M., FRANCESCO, P. D., MALAVOLTA, I., IOVINO, L., AND SALLE, A. D. MicroART: A software architecture recovery tool for maintaining microservice-based systems. In *2017 IEEE International Conference on Software Architecture Workshops, ICSA Workshops 2017, Gothenburg, Sweden, April 5-7, 2017* (2017), IEEE Computer Society, pp. 298–302.

[10] JONAS, E., SCHLEIER-SMITH, J., SREEKANTI, V., TSAI, C., KHANDELWAL, A., PU, Q., SHANKAR, V., CARREIRA, J., KRAUTH, K., YADWADKAR, N. J., GONZALEZ, J. E., POPA, R. A., STOICA, I., AND PATTERSON, D. A. Cloud programming simplified: A berkeley view on serverless computing. *CoRR abs/1902.03383* (2019).

[11] JONAS, E., SCHLEIER-SMITH, J., SREEKANTI, V., TSAI, C., KHANDELWAL, A., PU, Q., SHANKAR, V., CARREIRA, J., KRAUTH, K., YADWADKAR, N. J., GONZALEZ, J. E., POPA, R. A., STOICA, I., AND PATTERSON, D. A. Cloud programming simplified: A berkeley view on serverless computing. *CoRR abs/1902.03383* (2019).

[12] MAISSEN, P., FELBER, P., KROPF, P. G., AND SCHIAVONI, V. FaaSdom: a benchmark suite for serverless computing. In *14th ACM International Conference on Distributed and Event-based Systems, DEBS 2020, Montreal, Quebec, Canada, July 13-17, 2020* (2020), J. Gascon-Samson, K. Zhang, K. Daudjee, and B. Kemme, Eds., ACM, pp. 73–84.

[13] TeaStore repository. `https://github.com/DescartesResearch/TeaStore`. Accessed September 2025.

[14] UEDA, T., NAKAIKE, T., AND OHARA, M. Workload characterization for microservices. In *2016 IEEE International Symposium on Workload Characterization, IISWC 2016, Providence, RI, USA, September 25-27, 2016* (2016), IEEE Computer Society, pp. 85–94.

[15] VOGEL, T. mRUBiS: an exemplar for model-based architectural self-healing and self-optimization. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems, SEAMSICSE 2018, Gothenburg, Sweden, May 28-29, 2018* (2018), J. Andersson and D. Weyns, Eds., ACM, pp. 101–107.

[16] VON KISTOWSKI, J., EISMANN, S., SCHMITT, N., BAUER, A., GROHMANN, J., AND KOUNEV, S. TeaStore: A micro-service reference application for benchmarking, modeling and resource management research. In *26th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2018, Milwaukee, WI, USA, September 25-28, 2018* (2018), IEEE Computer Society, pp. 223–236.

[17] WEAVEWORKS. Sock shop: A microservice demo application. `https://www.infoq.com/articles/sock-shop/`, 2017. Accessed September 2025.

[18] ZEMTSOP NDADJI, B. A., BLIUDZE, S., AND QUINTON, C. AdaptiFlow: An extensible framework for event-driven autonomy in cloud microservices. In *Post-proceedings of the Workshop on Adaptable Cloud Architectures (WACA 2025)* (2025), G. De Palma and S. Giallorenzo, Eds., this volume of EPTCS, Open Publishing Association.

[19] Zhou, X., Peng, X., Xie, T., Sun, J., Xu, C., Ji, C., and Zhao, W. Benchmarking microservice systems for software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018* (2018), M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds., ACM, pp. 323–324.