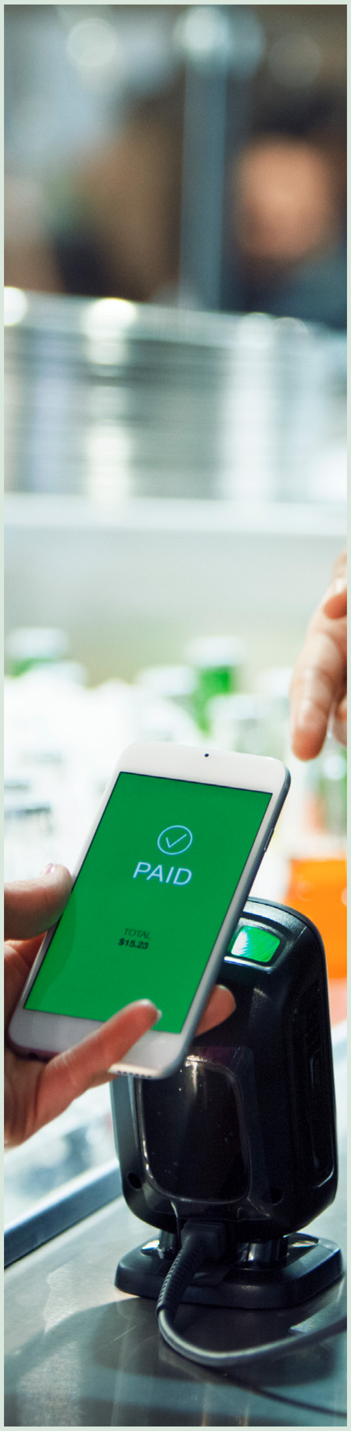
WHITE PAPER



IMPORTANCE OF BENCHMARKING FOR MICROSERVICES APPLICATIONS IN FINANCIAL SERVICES

Abstract

Applications are getting migrated from monolithic systems to microservices at a rapid rate in the Financial Services industry. However, microservices are often not benchmarked during the process of development and performance measurement. It may be due to lack of awareness about benchmark frameworks, paucity of time due to tight deadlines or additional work required to maintain these. This leads to limited predictability of project milestones delivery during development phase and application performance once microservices have been developed.

This whitepaper introduces the concept of benchmarking and its need. We also explored a few open-source benchmarks and its associated use cases. Additionally, we enumerated different scenarios where benchmarks could help us.

Infosys®
Navigate your next

Introduction

In the financial services industry, monolithic applications are being re-architected into microservices (small, manageable services implementing a particular use case) at a rapid rate. Typically, services in financial domain have complex business rules and deal with high volumes of data. Therefore, there needs to be more focus on benchmarking development and performance of the applications. This would lead to increased productivity during development and improved quality of the delivered products after deployment. By following these guidelines, one should be able to meet the planned deadlines and achieve the expected NFRs in performance.

Why Benchmarking?

Financial systems should have predictable SDLC for enhancements and be able to scale based on business demands. Having standard guidelines for SDLC helps development. Building a synthetic load generation tool to mimic actual production loads helps fine tune performance. To eliminate variances due to environment, all the benchmark runs should start in the same state (i.e. Caches, VMs, Databases, Files, States, Queues etc.). In the subsequent runs, data should be collected by varying only one input at a time so that it can be measured in isolation.

Benchmarking should be started earlier in the development phase because the cost of fixing issues at later SDLC stages is much higher. Any change in design could cause significant impact that necessitates thorough testing. It would be better to intercept them during initial stages.

Development Benchmarks

Development of microservices in financial domain on a predictable timetable is a challenge due to complexity of visualizing the system in its entirety. Troubleshooting across services makes it a time-consuming exercise. Hence, estimation of time and adherence to deadlines are of critical importance.

Community-owned standard benchmarks help solve these problems by measuring development against pre-defined and recommended characteristics. Implementing a solution in this manner prevents known issues from occurring and avoids rework. They can be grouped under three categories viz. Architectural, DevOps and General. Additional details regarding development benchmarks are in Table 1.

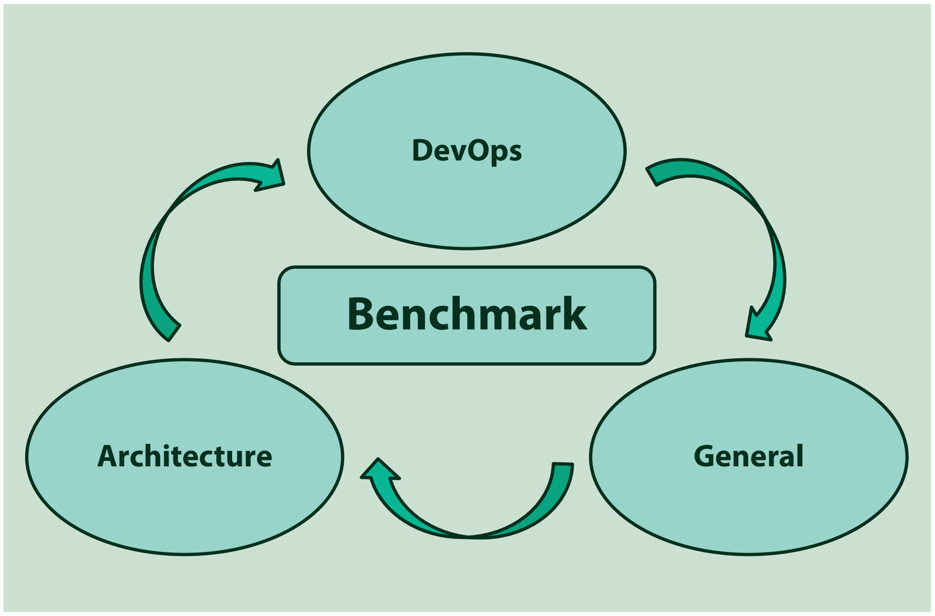


Figure 1.

Req.	Usage Table 1
Architectural	Things to be considered are: • **Explicit Topological View:** An explicit topological view, which specifies the main service elements and the communications between them, should be provided. • **Pattern Based Design:** Well-known patterns like **Circuit Breaker, Service Discovery, Database Per Service, Messaging** etc. should be implemented.
DevOps	Any code that is production ready should have the following practices from a development benchmark perspective. • **Easy Access From a Version Control Repository:** Version Control Repository such as **GitHub** or **Bitbucket** is a mandatory requirement that helps to have easy access to source code and release history. • **Continuous Integration:** Tools such as **Jenkins, TeamCity, etc.** help with automatic integration of existing software code with newly developed ones. Additionally, they help with code quality check and testing • **Automated Testing:** Tools like **Cucumber, Selenium** etc. should be used for automated testing on a repeated basis. • **Dependency Management:** For automatic download and installation of external software artifacts, one should use management tools like **Maven, Gradle, NPM etc.** • **Reusable Container Image:** Virtualized infrastructure tools like Docker are used for easy deployment independent of underlying physical infrastructure. • **Automated Deployment:** To alleviate the problem of variations related to application deployments across different environments, automated deployment tools such as **Chef** and **Ansible** should be used • **Container Orchestration:** Challenges faced due to service discovery, load balancing and rolling upgrade can be isolated using container orchestration tools like **Docker Swarm, Kubernetes and Mesos.**
General	This category is not mandatory from a technical perspective but it's a nice to have feature. It may not be always possible due to duplication of efforts and varied skill set requirement. • **Independence of Automation Technology:** An ideal solution would be to have support for multiple tools for dependency management, automated testing, continuous integration, and container orchestration • **Alternate Version:** It's good to have multiple implementations using different programming languages or different architectural designs. This helps to compare the design decisions and technology choices made. • **Community Usage and Interest:** A well-documented benchmark, having good customization and ease of deployment, is likely to be used more often. Its popularity would increase confidence in its adequacy and allows repeatability of results.

Performance Benchmarks

Performance is a dominant factor in all financial applications. Benchmarking would help us monitor and analyze changes to workload which ensures timely corrective actions. It also provides a comprehensive view of the parameters affecting the overall system performance on different platforms. This allows applications to face fluctuations due to market events as well as technology and business bubbles with confidence. Benchmarks can be considered reliable and useful based on the following points:

- Design of workload should be based on stress scenarios such that it can be tested against spikes seen in the real world. Additionally, in cloud environment, it is preferred to test the system stability by enforcing random instance failures to introduce chaos.
- Critical services in the financial firms must be tested by benchmarks and used as a basis for SLA's.

Benchmark Frameworks

DeathStarBench Suite

DeathStarBench is an open-source suite which studies microservices in Cloud and IoT. It contains few standardized use cases (like E-Commerce, Social Network, Video Streaming, Media Service, Movie Reviewing, Swarm Cloud, Swarm Edge and Banking System) which explores the implication of microservices on cloud stack and application design.

With increased digitization of financial services, we see usecases similar to standardized benchmarks (E.g. streaming of virtual townhalls vs video streaming, shopping carts in financial product transactions vs e-commerce, etc.). Hence benchmarks could be used to simulate the banking domain usecases.

Following are some of the commonly used principles:

- Representativeness
- End-to-end operations
- Heterogeneity
- Modularity
- Reconfigurability

One of the benchmarks in the suite is for a **Banking System**. Its services implement a secure banking system, which users leverage to process payments, request loans or balance their credit card. This should be a recommended benchmark to be used in financial domain. This benchmark uses RPC and has 34 unique microservices.

µSuite

Financial applications tend to have features such as product search, search across different integrated data categories, comparisons of similar competitive products, videos depicting detailed research, product catalogs of different funds, real time stock quotes, historical data and predictions, financial modelling and risk computations, etc. which are data intensive. They may be categorized as Online Data Intensive (OLDI) applications. For a better user experience, it is imperative to meet deadlines in the form of Service Level Objectives (SLO). In a typical monolithic application, one would have a latency benchmark of < 100ms but in microservices-based architecture one would expect a latency benchmark of < 10ms.

µSuite consists of 4 microservices based on different properties used in a real-world scenarios which are spread across 3 tiers.

- High Dimensional search (HDSearch)
- Router.
- Set Algebra
- Recommendation Engine.

This benchmark also uses RPC. Actually, it would make sense for financial systems to adopt RPC as the communication protocol as it tends to be much faster than REST.

How Benchmarks help us?

Development Benchmarks - Integration Issues

Applications in different departments in a financial firm may have different technology stacks and evolutionary paths - composed of a mix of open source, proprietary software, and bespoke implementations. When executing a cross-divisional project, there is a lack of visibility and estimations across teams. Using a standardized benchmark thus provides a reference for implementation and estimation. It also helps identify potential bottlenecks.

Development Benchmarks – Devops

Sometimes DevOps processes (E.g. Version Control, CI/CD, Automated Testing, Dependency Management, Containerization, Automated Deployment etc.) may have been sidelined due to various reasons. This would lead to development logjam and extended development lifecycle. Development benchmarks have requirements that should be supported for DevOps to prevent such issues from occurring. In financial domain, with tight regulatory deadlines, this would help market the product on time and within budget.

Development Benchmarks - Best Practices

Applications which are developed as rapid prototypes, minimum viable products, proof of concepts and internal applications may not have implemented all the best practices such as Circuit Breaker, Service Discovery, API Gateway, Database per service, Messaging, etc. This may inadvertently pose potential risks to data security, stability, and scalability of the system. In financial domain, data security is of paramount importance as it has a direct financial, regulatory, and reputational impact. A topological view that specifies the main service elements and the communications between them - provides a visual representation of the architecture and design. This should help identify the deficiencies in developmental life cycle while moving from initial stages into production implementation.

Performance Benchmarks - Scalability

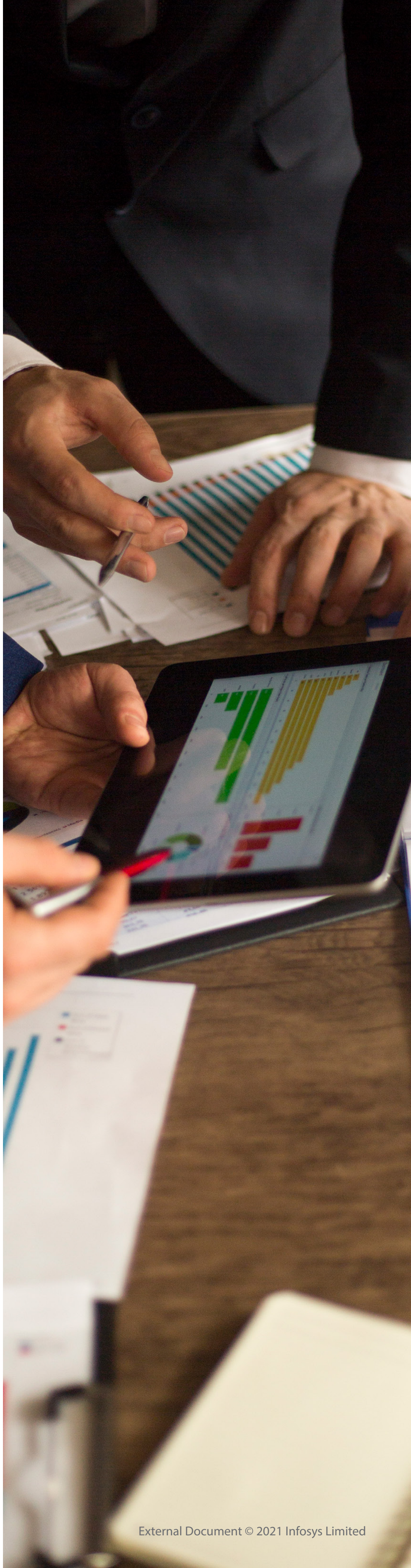
Financial applications are rich in functionality and possess complex business rules. Current workloads normally take precedence and higher focus. However, potential failure points due to higher workloads get sidelined. Having performance benchmarks gives us insights into how systems would behave under modelled workloads. It allows us to predict performance especially during peak load conditions and usage spikes.

Performance Benchmarks – Availability

Financial systems are expected to have fault tolerance and should be highly available for business continuity. However, due to market events, new products/ clients etc. the workload patterns change. This leads to unforeseen behaviors and faults due to hidden bugs. If outage or failure occurs due to this, the availability would be severely impacted. Having performance benchmarks would allow the design to be validated during testing and catch such issues ahead of time.

Performance Benchmarks - Infrastructure Updates

Infrastructure updates due to OS, hardware, and networks - presents a constantly changing environment. Financial firms perform regular updates for performance and security reasons. This may lead to failures for a few runtime scenarios and present risks to production environment. Having performance benchmarks and testing on target environment would give a prediction of system behavior under enclosed, low-risk conditions.



Financial Domain Benchmarks

The use cases and patterns for data sourcing in financial domain are unique. Currently, there are limited benchmarks available to represent them. There is a need for more benchmarks that model financial world issues such as trade sourcing, financial modelling, settlements, open banking use cases and financial transactions to name a few. Some of the patterns we can see are:

Trade processing Requirements

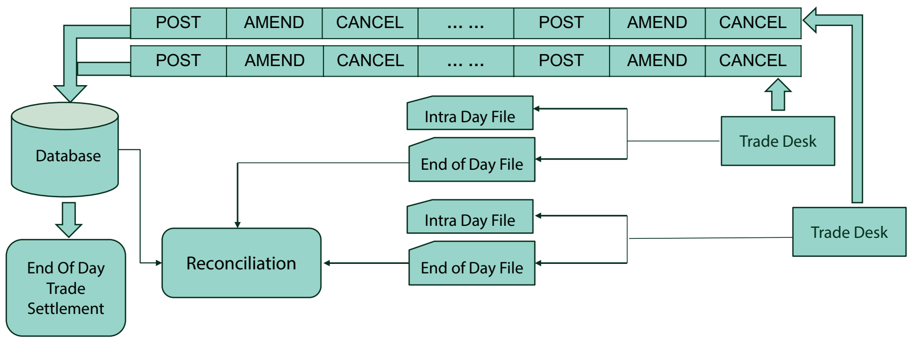


Figure 2

- Queueing-based real time trades which are reconciled at End-Of-Day.
- Real Time Queues for Intra Day processing with region based Close Of Business.
- Standard and additional attributes for different types of trades. Details are as in Figure 2

Model calculations

Model calculations use time series data as input and output. These calculations occur in different time horizons and have a specific behaviour that is not captured in any of the open-source benchmarks.

Different factors that affect the behaviour of queuing behavior in model calculations would be:

- Plain vanilla instruments which may tend to be simple and may need to be batched together.
- Exotic or bespoke instruments which are complicated and may require splitting logic for parallelization.
- Volume of trades and different instrument types which affect the end-end computation time.

Time horizons define the length of the time series and this in turn decides the time taken for modelling computations. E.g. There are different time durations calculations for Basel III. Details are as in Figure 3 and Figure 4.

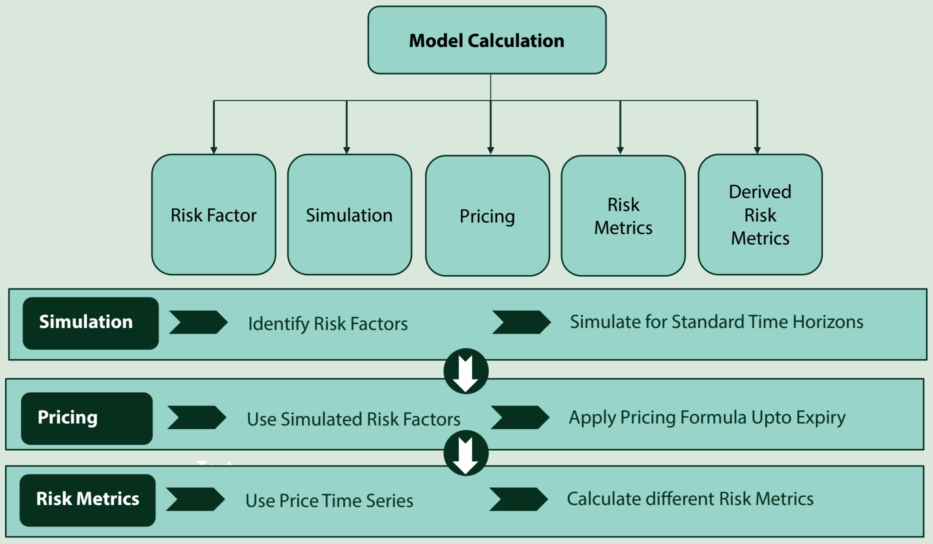


Figure 3

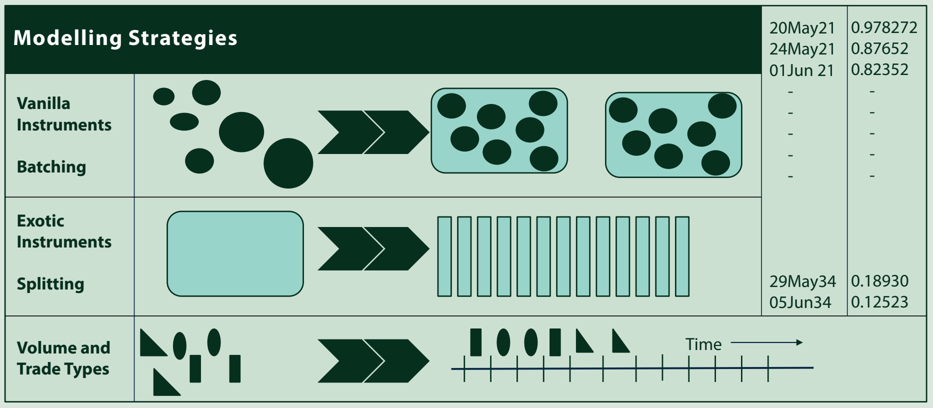


Figure 4

Conclusion

Financial Service industry involves strict regulatory guidelines and requires development on challenging timelines with regular milestones. Adhering to development benchmarks guides us to achieve complete product development on time and within budget. Regulatory changes, economic events, financial market bubbles and financial crisis apply a drastic change in workload patterns for which the applications might not have been designed. This results in exposed weak links and may either lead to catastrophic failures or major fluctuations in performance. Performance benchmarking allows us to predict and remediate these potential issues before hand.

Glossary

API	Application Programming Interface
CI/CD	Continuous Integration, Continuous Delivery
DevOps	Development Operations
HDSearch	High Dimensional search
IoT	Internet Of Things
NFR	Non-Functional Requirement
OLDI	Online Data Intensive
OS	Operating System
REST	Representational State Transfer
RPC	Remote Procedure Call
SDLC	Software Development Life Cycle
SLA	Service Level Agreement
SLO	Service Level Objectives
VM	Virtual Machine

About the Authors

Prashanth Kumar M S
Senior Technical Architect, Banking and Financial Services, Infosys Ltd

Prashanth Kumar M S has worked in the domain areas of Financial Services and Investment Banks, particularly Credit Risk. He is an Open Source enthusiast, involved in application design with global clients in areas of Finance and Investment Banking in Cloud and Microservices. He is exploring Azure Cloud and has interests in No SQL databases such as MongoDB, Node4J.

Balakarthikeyan Ananthapadmanaban
Technical Architect, Banking and Financial Services, Infosys Ltd

Balakarthikeyan Ananthapadmanaban has worked on Open-Source technologies based on Java EE platform and cloud native technologies. His areas of interests include exploring latest technologies like Microservices, Google Cloud Platform, Kubernetes, ELK Stack, Apache Kafka, GraphQL.

Biresh Choudhury
Product Technical Lead, Banking and Financial Services, Infosys Ltd

Biresh Choudhury has worked in the domain area of Retail Banking and Insurance. He is involved in technical architecture and development of software for large global clients in Investment Banking. His technology focus areas are Amazon Web Services, GraphQL, NoSQL Databases.

Acknowledgements

We are grateful to Murali Sethuraman, Senior Principal Technology Architect, for his support and guidance.

References

a. Research Paper

- Benchmark Requirements for Microservices Architecture Research by Carlos Mendes Aderaldo, Nabor C. Mendonca, Claus Pahl, Pooyan Jamshidi
- µSuite: A Benchmark Suite for Microservices by Akshitha Sriraman, Thomas F. Wenisch
- An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems by Ankitha Shetty, Brian Ritchken, Brett Clancy, Leon Zaruvinsky, Yu Gan, Priyal Rathi, Brendon Jackson, Chris Colen, Mateo Espinosa, Yanqi Zhang, Nayan Katarki, Kelvin Hu, Fukang Wen, Rick Lin, Christina Delimitrou, Dailun Cheng, Ariana Bruno, Meghna Pancholi, Catherine Leung, Zhongling Liu, Justin Hu, Yuan He, Siyuan Wang, Jake Padilla
- Performance Modeling and Benchmarking of Event-Based System By Kai Sachs

b. Online Research

- Thoughtworks: Podcast-techniques-implementing-microservices-and-cloud
- Google-Cloud: Microservice-performance