TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Tatjana Kirotar 178109IABM

Performance testing of microservices in cloud-based environment

Master's thesis

Supervisor: Aleksandr Kormiltsõn
MSc

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Tatjana Kirotar 178109IABM

Mikroteenuste koormustestimine pilvetehnoloogial põhinevas keskkonnas

Magistritöö

Juhendaja: Aleksandr Kormiltsõn
MsC

Tallinn 2023

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Tatjana Kirotar

02.01.2023

Abstract

Software development has increasingly moved towards microservice and cloud based architectures, where applications are built using small independent services which are deployed and scaled according to the need by cloud service providers.

On the one hand, microservices should be easy to test and maintain since they are independent functions focused only on one specific functionality. On the other hand, dividing application into distributed functions means that a number of services grows dramatically. Therefore, testing of microservices applications actually becomes more and more complex. Testing has to be done across all components and tracing errors and performing root cause analysis is difficult across all of the services.

Performance testing of such applications involves additional hardships of analysing the performance of each component and cloud service provider metrics in comparison with overall performance of the system.

This paper conducts a case study on how to design and run performance tests for microservice based application in cloud environment and how to select and adopt the performance metrics collected from test runs to identify application performance. Through a series of experiments on Broking Manager (BM) application, the paper illustrates that not all collected metrics can identify performance issues and conducting root cause analysis for performance issues is not a straightforward process.

This thesis is written in English and is 57 pages long, including 7 chapters, 18 figures and 3 tables.

Annotatsioon

Mikroteenuste koormustestimine pilvetehnoloogial põhinevas keskkonnas

Tänapäeva tarkvara arendus on ühe enam liikunud mikroteenuste ja pilvepõhiste tehnoloogiate poole, kus rakendused on üles ehitatud väikeste ja sõltumatute teenuste abil. Mikroteenuseid juurutatakse pilveteenuste pakkujate juurde, kes omakorda pakuvad võimalust skaleerida teenuseid vastavalt vajadustele.

Ideoloogiliselt peks mikroteenuste tulek aitama kaasa testimise lihtsustamisele, kuna iga teenus keskendub ainult ühele konkreetsele funktsionaalsusele, saab igat teenust eraldiseisvalt testida. Teisest küljest on aga rakendused niivõrd hajutatud, et teenuste hulk kasvab kümnetesse ja sadadesse mikroteenustesse, mida on raske hallata. Seetõttu muutub mikroteenuste testimine tegelikult aina keerulisemaks. Testimine peab endas hõlmama kõiki komponente ning vigade otsimine ja algpõhjuste analüüsi tegemine on selliste rakenduste puhul oluliselt raskendatud.

Selliste hajutatud rakenduste koormustestimine hõlmab endas veel täiendavaid keerukusi. Lisaks sellele, et tuleb testida iga komponendi jõudlust, siis sinna juurde tuleb arvestada ka pilveteenuse pakkuja mõõdikute analüüsi ning võrrelda seda kogu süsteemi jõudlusega.

Käesolev töö kirjeldab juhtumiuuringut selle kohta, kuidas koostada ja jooksutada koormusteste mikroteenustele pilvetehnoloogia keskkonnas ning kuidas valida ja kasutada kogutud jõudlusmõõdikuid rakenduse jõudluse tuvastamiseks.

Töö käigus tehtud katsetest rakendusega Broking Manager (BM) saab järeldada, et kõik kogutud mõõdikud ei suuda tuvastada jõudlusprobleeme ning jõudlusprobleemide algpõhjuse tuvastamine on keerukas protsess.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 57 leheküljel, 7 peatükki, 18 joonist, 3 tabelit.

List of abbreviations and terms

Performance testing	Software testing practice to determine how a system performs under a particular workload.
Load testing	Load testing is one of the performance testing types, where tests are created to measuring application performance under expected production like workload.
Performance metrics	Performance metrics are measurable indicators [1], which express the performance related characteristics of a system, which usually are response time, throughput and resource utilization.
Deadlock	A situation when processes are blocked due to each process holding a resource and waiting for more resources which are reserved by some other process [2].
Cloud computing	Cloud computing is an architecture for need and interest based computing resources [3]. Outsourcing the need for personal servers to cloud service providers and sharing the resources between many companies and applications.
Cloud service provider (CSP)	A cloud service provider is a company that offers components of cloud computing, such as cloud-based platform, infrastructure, application and storage [4].
Regression testing	Regression testing is a software testing type, which is conducted to ensure that application still functions as expected after introducing any code changes or updates.
Software Development Life Cycle (SDLC)	Software Development Life Cycle is a process of software designing, developing and testing [5].
Monolithic architecture	Monolithic architecture is a traditional software model, when an application is built as a single independent unit [6].
Microservices architecture (MSA)	Microservices architecture is modern architectural method, when an application is built by a collection of small and independent services [6].
Serverless computing	Serverless computing is a model, where server is being allocated to the application only when the application is being executed [7].
Function-as-a-Service (FaaS)	Function-as-a-Service (FaaS) platforms leverage serverless infrastructure to deploy, host, and scale resources on demand for individual functions known as “microservices” [8]

AWS Lambda	Serverless computing service provided by Amazon cloud service provider [9].
End-to-end (E2E) testing	End to end testing is a software testing method which imitates the real user workflow to validate the whole application from beginning to an end.
Broking Manager (BM)	Broking Manager (BM) is an application that allows to analyse companies cyber risk [10].
Loss Model	Financial Loss model is a set of serverless functions which are designed to assist companies in understanding their financial exposure to cyber risk.
Threat Model	Threat model is a set of serverless functions which are designed to support companies in understanding their exposure to certain cyber threat scenarios.
Application programming interface (API)	An application programming interface is a way for one or more services to communicate with each other.
API Gateway	An API gateway is a proxy to all requests coming from users or transferred between the services [11].
User Interface (UI)	The user interface is a point where human user interacts and communicates with a computer.
Amazon Simple Queue Service (SQS)	Amazon Simple Queue Service (SQS) is a way to send, store and receive messages between components [12].
Functional requirements (FR)	Business requirements, which define how a product should function under specific conditions.
Non-functional requirements (NFR)	Technical requirements are often defined as quality characteristics of the system.
Use case	A user case is a written description of how users are interacting with an application.
Apache JMeter	Apache JMeter is Java-based software designed for load testing functional behaviour and measuring performance [13].
Resource utilization	Resource utilization is an utilization cost of server and network resources [14, 15].
Central processing unit (CPU)	CPU is a processor that executed the instructions provided by an application.
Asynchronous	Asynchronous is a way to describe events happening in unconsecutive manner.
Response time	Response time is the amount of time taken to respond to a request.
Throughput	Throughput is the number of requests handled by the application per second [16].

Continuous Integration and Continuous Deployment (CI/CD)	A culture of operating principles and a set of practices used in application development and delivery [17].
User Collaboration Service (UCS)	User Collaboration Service is a serverless function called to share a newly created account with every other user in a user group.
Locking Service (LS)	Locking service lambda is used to lock a shared account to specific user before allowing to make any changes.

Table of contents

List of figures

List of tables

1 Introduction

Application performance is one of the most important characteristics of any software. Performance is an important non-functional requirement, which describes applications properties in relation to timeliness and resource utilization [18].

Performance is determined by application complexity. Incremental changes added to the software are changing the overall complexity of the system which often has negative effect on the performance.

With a growth of complexity needed within applications, more and more applications are built using microservice architecture, meaning that whole application could consist of tens and hundreds of different services.

Moreover, with growing popularity of microservices and possibilities to offer software as a service, applications are more commonly built within a cloud environment. The ability of cloud computing to scale any computing resources according to the need is the fundamental reason for companies to migrate their applications to the cloud service provider (CSP).

Performance testing and regression detection is already considered challenging even in traditional systems [19, 20]. With the new approach, it means that the performance of an application is not only determined by the application itself, but also by the availability of the services, cross-service communication possibilities and scalability of the chosen cloud service provider.

This paper conducts a case study on how to run performance tests in such versatile environment and analyses how to select and adopt the performance metrics collected from test runs to identify performance regressions.

1.1 Research problem

Performance testing is a complex and time-consuming process, which is often ignored and left out of the software development cycle. This generally leads to finding performance issues only in production, which can lead to customer dissatisfaction, monetary loss and in worst cases complete outages of the entire systems [21]. Repairing performance problems in the late stage of the software development life cycle (SDLC) may require considerable adjustments in the design or architecture and the cost of such changes is the highest [22].

Performance testing difficulty highly relies on the system complexity, whereas there are additional challenges with performing these tests in microservice applications. Microservice applications could consist of hundreds of different services and have multiple internal and external dependencies. A performance degradation from one of the services can lead to several issues and will be difficult to track to the root cause [23].

Key benefits of microservices architecture (MSA) produce also the biggest issues for testing:

1) Each service can be written in their own language and using its own technology stack – performance issues are different for each technology stack, making them hard to identify and locate root causes [23].

2) Microservice applications have an ability to change only one service at a time, do it quickly and frequently – aforementioned dynamic environment provides difficulty in root cause analysis for any issues [23].

3) Cloud computing’s main selling point are the serverless (lambda) functions which use computing resources only according to the need – this makes testing of the whole application difficult and could cause performance issues when a function is initiated [40].

4) All possible metrics are collected and they are easy to access – the more services application has, the more metrics are provided and using them for investigation is time-consuming [23].

All of these aspects are making performance testing even more challenging than before. Finding a correct way to run performance tests and making sure that metrics collected and measured provide accurate enough information on the application performance is essential to the business [24].

1.2 Purpose of the study

The main goal of this thesis is to provide an investigative approach on whether performance testing results in a microservice-cloud based applications are stable and reproducible and whether there are ways to improve traditional performance metrics using cloud service provider metrics.

In particular, this thesis conducts a case study using the Broking Manager (BM) [10] application, to determine how traditional and cloud performance metrics detect performance regression.

Case study consists of experiments which will address the following research questions concerning the main pain points of microservice-cloud based application performance testing:

RQ1. How stable are the performance tests results of a microservice and serverless application in a cloud-based environment?

RQ2. How do the lambda cold starts affect the application performance?

RQ3. Do cloud service provider metrics help assess the performance of an application?

1.3 Thesis structure

Current thesis consists of seven paragraphs.

First chapter describes the problem background and purpose of the study with following research questions.

Second chapter provides a theoretical background of the microservices and explains in detail the issues that arise when testing such applications. Additionally, this chapter gives an overview of cloud computing and serverless (lambda) functions describing how these lambdas are designed and how they operate.

The following third chapter gives an overview of the Broking Manager application, it’s architectural setup from the viewpoint of the microservices and lambdas used in the application and illustrating the internal and external dependencies which need to be taken into account when conducting the performance testing.

Fourth chapter describes functional and non-functional requirements of the software, explains the different performance testing types and clarifies the decision behind using the load testing for current work. This chapter also further examines the performance metrics to be gathered and analysed during the testing cycle.

Fifth chapter has an overview of process of performance testing approach followed during this project. It discloses the performance requirements and how performance scenario was designed using collected metrics from actual everyday users. The chapter also demonstrates the implemented performance scenario and it’s execution strategy.

Chapter six analyses gathered results and draws conclusions based on done experiments. This chapter has answers to research questions stated in section 1.2. This chapter also discloses possible threats to validity, compares current work with previous studies and offers suggestions for future studies.

Final chapter summarises the results of the master’s thesis.

2 Technological background

Current chapter introduces the differences between monolithic and microservices architecture and analyses pros and cons of both solutions in the context of software testing. This chapter gives an overview of cloud computing with the main focus on serverless functions which are widely used by analysed application.

2.1 Microservices

Traditionally, application have followed the monolithic architecture, meaning that all the functionality and all needed components are handled as a single application unit.

Most of the big and successful applications have stated off as a monolith, yet there are several drawback of monolithic architecture. The main pain points being the complexity and size of such applications. Detangling the monolithic application is difficult and making small changes to single aspect of the application might cause regression issues in the whole application [25]. Regression testing has to cover all critical functionality. These issues lead to slow development cycles [26].

In order to overcome the issues of monolithic structure, the ideas is to divide the application into smaller but interconnected services, where each service is responsible for its own functionality but communicate with other services via application programming interfaces (APIs) [25].

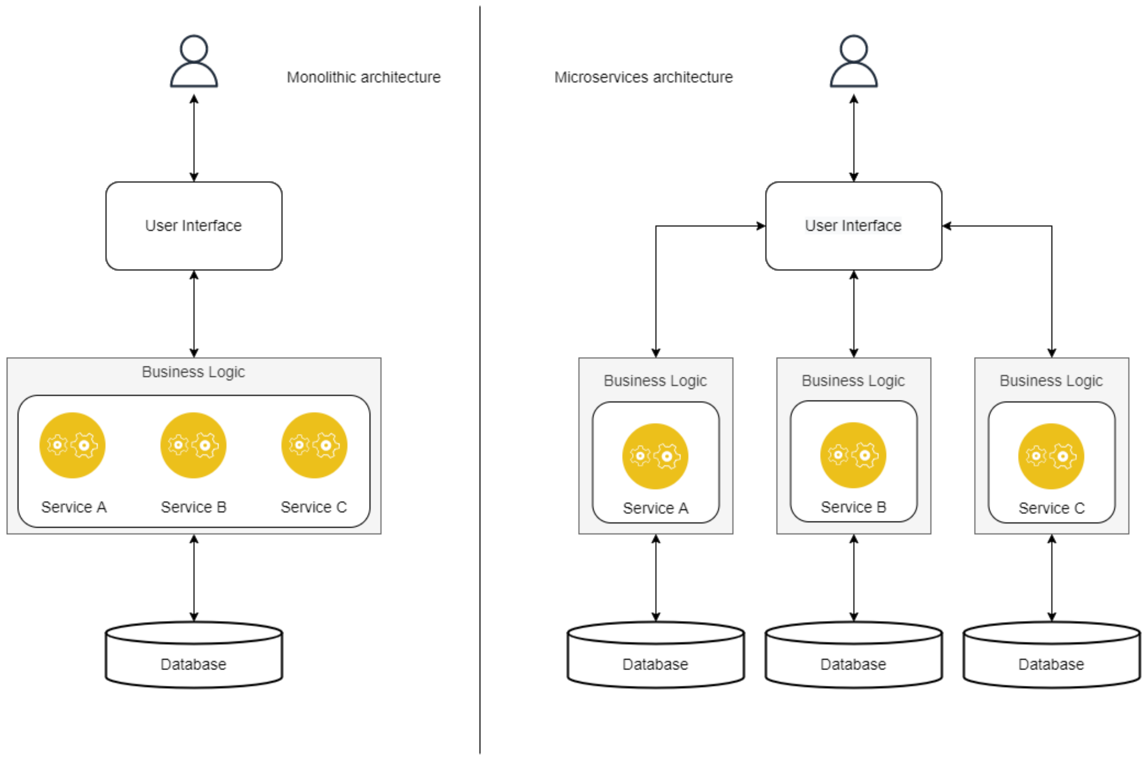


Figure 1 Monolithic vs Microservices architectures

This approach addresses the main problem of complexity of a big application by decomposing the application into a set of manageable services which are much faster to develop and much easier to maintain [27].

Additionally, it enabled dividing development into smaller teams, where each team is responsible for development and maintenance of their own service. Having a small diverse team and a small application adds even more benefits where each service can be developed using its own technology and components are loose coupled. Therefore, the service can be deployed, scaled and tested independently [28].

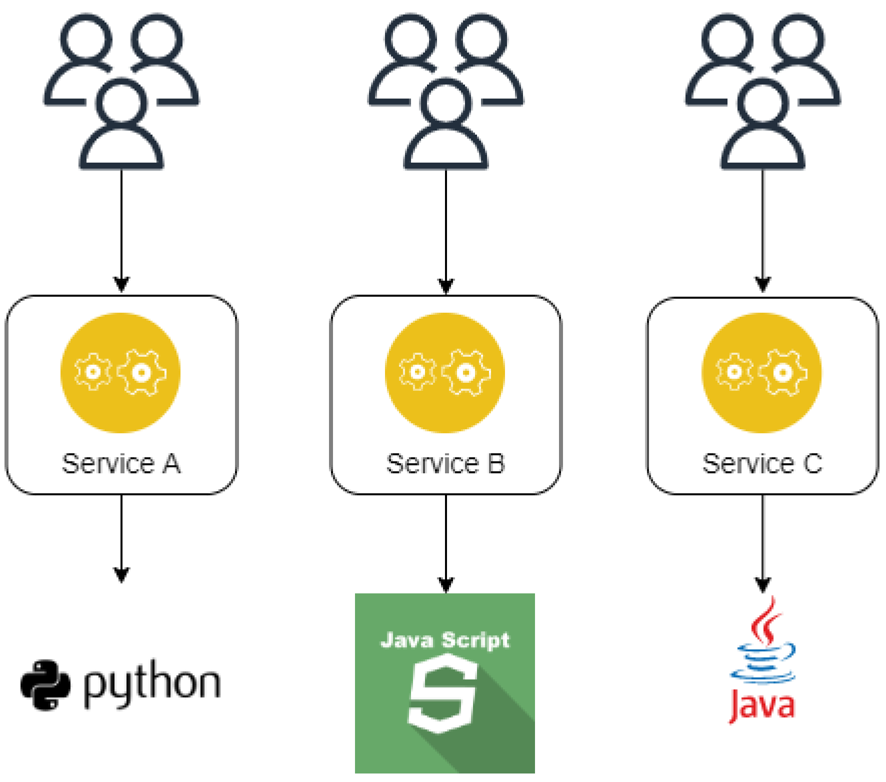


Figure 2 Team and technology independence

Seeing that microservices do solve some of the problems of a monolithic architecture, it produces a set of its own issues as well.

First of all, microservices move some of the complexity from code level to team and individual level, since using multiple services needs more collaboration between team the same way the communication is done between the microservices themselves.

Furthermore, dividing application into distributed state means that when a number of services grows, it is really hard to keep track of [28].

Deploying a microservices based application is more difficult as well, since each instance needs to be configured, deployed, scaled and monitored. Seeing that each service can have its own technological stack, they could also include their own database and model. [29]

Testing microservices applications becomes more and more complex, since end-to-end (E2E) testing has to be done across all components and tracing errors is difficult across all of the services [28]. Same issue is present when doing performance testing, even though performance issues with one of the components does not linearly translate to the whole system performance, each component can affect overall performance of the system [30].

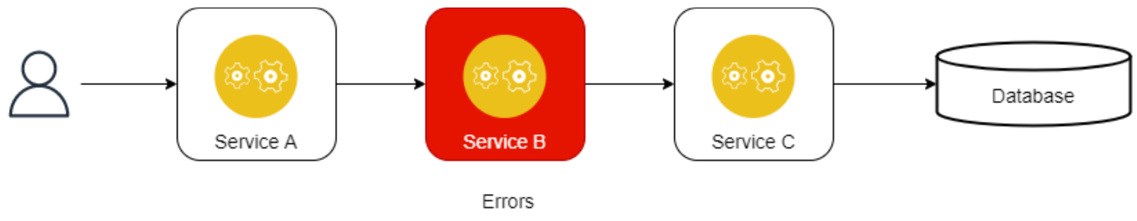


Figure 3 Errors in microservices

2.2 Cloud computing

Cloud computing and microservices are not necessarily dependent on each other, but there are several benefits of using these two approaches together.

One of the main benefits of microservices, is that they can be deployed and scaled individually, it allows to use the resource allocation and cost benefits of using the cloud hosting, including the on-demand scalability and pay-per-use infrastructure [29, 31].

Additionally, cloud service providers offer easily configurable technologies set for each component, which would be truly exhausting to manage on your own. Consuming the supporting stack as a cloud service can greatly minimise the management challenges [29].

2.2.1 Serverless computing

Further development in cloud computing and microservice architecture is running the application using serverless model [7].

Cloud functions [32], more often called Function-as-a-Service (FaaS) platforms leverage serverless infrastructure to deploy, host, and scale resources on demand for individual functions known as “microservices” [8]. This has revolutionized application development by fully eliminating the need to manage underlying infrastructure and allow developers to focus only on the code [33].

Most of the prominent cloud computing providers including Amazon [9], IBM [34], Microsoft [35], and Google [36] have released their own serverless computing capabilities. With that cloud service providers promise fine-tuned scaling of resources, high availability, errorless execution and affordable due to pay per use billing structure using their serverless computing [37, 38].

Serverless model goes another step further into better utilisation of resources and ultimately saving more energy. Rearchitecting applications for small microservices deployed in serverless model allows cloud service providers can combine user workloads to fill available capacity and deallocate any unused resources [37].

Even though, there are good reasons to use serverless, there are still several issues that come with adopting this model.

The main consideration aspect for FaaS is cost, which can be considered as a positive and a negative. Customers are only charged when function starts up and is in the use [7], but on the other hand when company grows and computation is used ineffectively, the costs could grow out of proportion. Aforementioned model and limited reporting by cloud service providers make it challenging to do any kinds of cost estimations [39].

Another issue key benefit of serverless is also that functions start up when they are needed. However, scaling up and down to zero is causing cold start issues [40].

From performance testing perspective, there are two main issues. Firstly, it is difficult to see how the functions are deployed and there is no way to replicate serverless environments [7, 41]. Additionally, function cold starts could affect the overall performance results and running performance tests in pay-per-use environment is costly.

3 Broking Manager

Broking Manager (BM) is an application that allows to analyse companies cyber risk [10]. BM allows users to search for actual companies and run an analysis that combines two main features: estimating financial losses and illustrating possible threat factors.

Application has more additional features and many external services which are not in the focus of current work.

3.1 Business workflow

Application is built using microservice architecture and serverless model with chosen cloud service provider as Amazon AWS[1], therefore current thesis will be focused only on one cloud service provider and following technologies.

BM service is a centralised service which connects user to all different services. Usual workflow of the application is described in Figure 4 below.

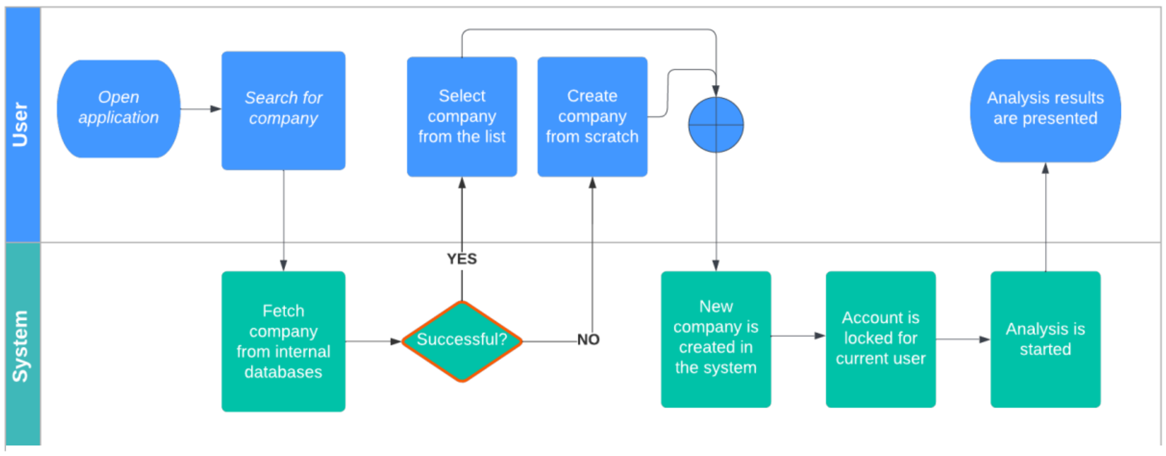

User
System
Open application
Search for company
Select company from the list
Create company from scratch
Analysis results are presented
Fetch company from internal databases
Successful?
YES
NO
New company is created in the system
Account is locked for current user
Analysis is started

Figure 4 Broking Manager process flow

[1] https://aws.amazon.com/

3.2 Architecture

In this paper, the main focus is on the company analysis, which is most important part of the application. Analysis consists of two components, Loss Model and Threat Model.

Financial Loss model is designed to assist companies in understanding their financial exposure to cyber risk. Model uses a substantial repository of unique cyber incident events, identifying the costs associated with each of the event, analysing the industry impacted, company size, region and any other unique identifiers of the event. Using the base statistical information, model runs a 50,000 year Monte Carlo Simulation of events to formulate the distribution of loss for each kind of peril.

Threat model is designed to support companies in understanding their exposure to certain cyber threat scenarios. Model is gathering data from multiple security signal vendors, firmographic information and company assessment, using this data model contemplates company's inherent and controllable exposures to nine different threat scenarios: ransomware, cloud outage, data theft, cash theft, power outage, DNS provider outage, physical infrastructure weaponization, data loss from OS provider and data theft from email services provider.

System is designed in distributed manner with one core service (BM service) and supporting serverless functions. Application architecture is shown below in Figure 5. Both models are implemented by a group of serverless functions, AWS lambda functions[1]. Serverless approach has been chosen for these calculations due to high resource need for a short period of time. Therefore, lambdas are only activated when analysis is triggered by the user. User can access the application via user interface (UI) or a series of API calls. All requests coming from users or transferred between the services are controlled by Amazon API Gateway [42].

The default Amazon API Gateway[2] timeout limit is 29 seconds for all integration types, including Lambda, HTTP, and AWS integrations, which means that some connections or combination of connections are open for longer period of time and therefore initial

[1] https://aws.amazon.com/lambda/

[2] https://aws.amazon.com/api-gateway/

connection is terminated [42]. As models require additional data and time for actual computations, they are not always able to finish within the defined time limit. Hence, the calls are designed to work in asynchronous manner and Amazon Simple Queue Service[1] (SQS) is used in order to send, store and receive messages between components [12].

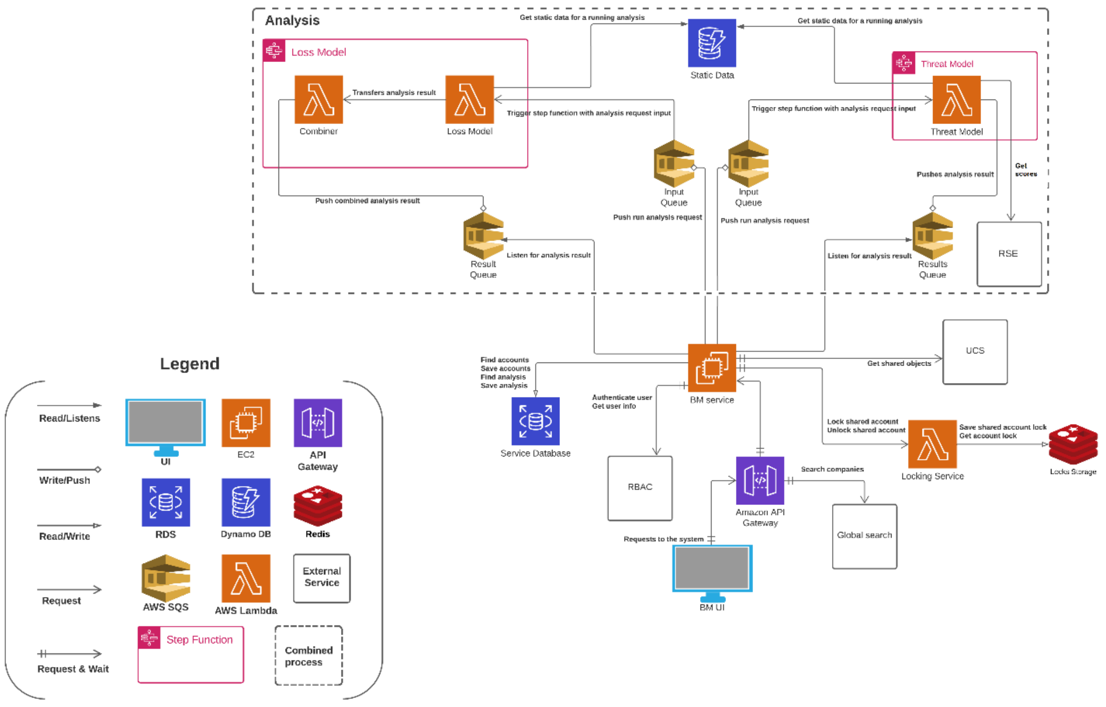


Figure 5 Broking Manager architecture

[1] https://aws.amazon.com/sqs/

4 Software testing and performance measurement

Following chapter discusses software testing according to functional and non-functional requirements of the software. Further section examines the performance aspects of the software as one of the key requirements and which performance metrics can be gathered and analysed by different performance testing types.

4.1 Software testing

Software testing is a process of verifying and validating that application or product under test works as expected and therefore meets both business and technical requirements. Software testing is a vital part of software development, if not done correctly the applications can have errors which may cause many issues to the company and the users [43].

Business requirements are more commonly called as functional requirements (FR), which define how a product should function under specific conditions. Functional requirements are provided by users or any other stakeholders to ensure that product is behaving in proposed manner and producing expected results.

Technical or non-functional requirements (NFR) [44] are often defined as quality characteristics of the system, they can either expand or add limitations to the functional requirements. Non-functional requirements describe how a system should operate, rather than what the system should do.

Non-functional requirements may vary depending on the product, technology, legislation, et cetera. However, the key requirements [1] are described in Table 1.

Table 1 Non-functional requirements

Availability	System is accessible when required.
Compatibility	System is capable of operating with other components.

Functionality	System is working according to user needs.
Maintainability	System is easily modified to new requirements or needs.
Performance	System functions in timely manner with minimum consumption of resources.
Portability	System or its component can be easily transferred from one environment to another.
Reliability	System performs its functions according to the requirements for a specified period of time.
Scalability	System is able to grow together with increased workload.
Security	System is protected against malicious access or use.
Usability	System is easy learn for a user.
Certification	System is meeting all necessary standards or conventions.
Compliance	System is meeting all necessary regulatory or legal constraints.
Localization	System can be used with several localization, including different languages, laws, currencies, cultures, etc.
Service Level Agreements	System follows the formally agreed upon rules.

Extensibility	System can be easily updated to include new functionality.

As mentioned in section 2, many of these requirements are already either fully or partly controlled by the cloud service providers, for example all cloud service providers promise customers high availability, scalability, performance efficiency, portability, and infrastructure security. Additionally, moving away from monolithic applications to microservices, greatly improves characteristics like compatibility, maintainability, and extensibility.

Issues with applications after the release are more commonly caused by applications not being able to scale to appropriate workload rather than feature related errors [45, 46, 47]. Performance issues could cause system to freeze, crash and become fully unresponsive, additionally high workload could produce issues with memory management and deadlocks [47]. Therefore, current thesis work is focused on performance, as one of the main non-functional requirements.

4.2 Performance metrics

Traditionally, the main performance indicators are response time [48], throughput and resource utilization.

Response time is the amount of time taken to respond to a request. There are several ways to measure response time [14]:

- Latency measured at the server. For serverless applications one type of latency is the duration of the lambda function to start and finish the computation process.
- Latency measured at the client. This latency is measured from the client perspective, which includes time taken by the API Gateway, request queue, lambda computation duration, result queue.

Throughput is the number of requests handled by the application per second [16]. In the context of this work, we will define throughput as the number of transactions completed in a second and also as the number of concurrent lambda executions.

Resource utilization can be identified as a utilization cost of server and network resources [14, 15]. The primary resources for microservice-cloud based environment are the following:

- CPU
- Memory
- Disk input and output (I/O)
- Network input and output (I/O)

4.3 Performance testing types

In order to gather application performance metrics and compare them against non-functional requirements, performance testing has to be conducted.

There are three main types of performance testing: load, stress and endurance tests.

Load tests are intended for measuring application performance under expected production like workload. Load tests are supposed to show the closest approximation to the real-life application usage. Main goal of load tests is to ensure that any changes made to the application continue to meet the predefined non-functional requirements.

Stress tests are designed to measure the workload under which the application starts to fail. Stress tests will provide an overview of what are the actual capacity limits of the system, which components will malfunction and how will the system recover from such failures.

Endurance tests are designed as load tests, with production-like workload, with a difference that endurance tests are long-running tests for detecting any issues that might appear only after an extended period of time.

The most effective type of performance tests depends on the objective [49] and in the context of current work, the focus will be on the load testing. The tests will be designed to run on a regular basis and eventually added to the continuous integration and continuous deployment (CI/CD) pipeline [50].

5 Performance testing process

Following chapter introduces performance testing approach used for current research, followed by description of how each step is executed and what are the results of every stage.

5.1 Performance testing approach

Current thesis uses performance testing approach proposed by Ian Molyneaux as basis. [51] The proposed approach versatile and is applicable to most performance testing projects. Followed steps are briefly described in Table 2 below.

Table 2 Performance testing steps

Performance Testing Step	**Description**
Non-functional Requirements (NFR) Capture	Gather all performance NFRs from all stakeholders: identified performance targets, key use cases, data requirements.
Performance Test Environment Build	Make a close replica of production environment: at a minimum reflect the production deployment and database size.
Use-Case Scripting	Identify key use cases and any key components that need to be monitored separately.
Performance Test Scenario Build	Identify test type (described in section 4.3) and following test volume, and duration.
Performance Test Execution	Run and monitor performance tests.
Post-Test Analysis and Reporting	Collect and analyse data from all test runs, compare data with requirements and create a following report.

5.2 Performance requirements

In order to conduct any meaningful performance testing, performance requirements have to be described in specific and verifiable manner [52]. Current system has following non-functional requirements described by a stakeholder:

- expected execution time for small requests: under 1 second;
- expected execution time for heavy (analysis) requests: 5-10 seconds;
- expected amount of simultaneous actions: 5 actions per second during at least 1 minute.

5.3 Performance environment

Performance testing environment has to be chosen as a closest replica to production environment as possible to produce any meaningful results.

Software development process used for current application addresses this issue by using cloud service provider solutions to replicate environments. All development and testing environments are built using the same setup and can be scaled up or down based on the need. For conducting performance testing, an integration environment was used to resemble the production-like experience with all services (internal and external) built and deployed to the same environment.

5.4 Performance scenario scripting

Following the fundamental performance testing strategy [53], the performance scenario is designed to mimic the main workflow that regular users follow when using the application [54]. In order to understand how application is used by everyday users, analysis has been conducted using the collected metrics from Google Analytics [55]. Analysing the user behaviour from the last year, there are two main types of events that users do:

- Create new account (15%)
- Actions with the analysis (85%)

With analysis the main actions are mainly just page clicks looking at the model results in different variations (graphs, charts, documentation, etc), this covers over 85% of analysis actions from users. But about 15% of actions are analysis reruns.

Based on these finding, the designed performance scenarios should follow described workflow:

1. User logs into the application (external service, not under test)
2. Searches for company (external service, not under test)
3. User creates a new account in the system
4. Runs an analysis (Threat and Loss calculation models)
5. Unlocks the account for editing purposes
6. Edits account information
7. Reruns an analysis (Threat and Loss calculation models)
8. Deletes the created account

There are several ways of how users can edit an account, which is why steps 6 and 7 are repeated several times during a test. The exact test case is implemented as shown in Figure 6. The actual performance test code is not included due to confidentiality reasons.

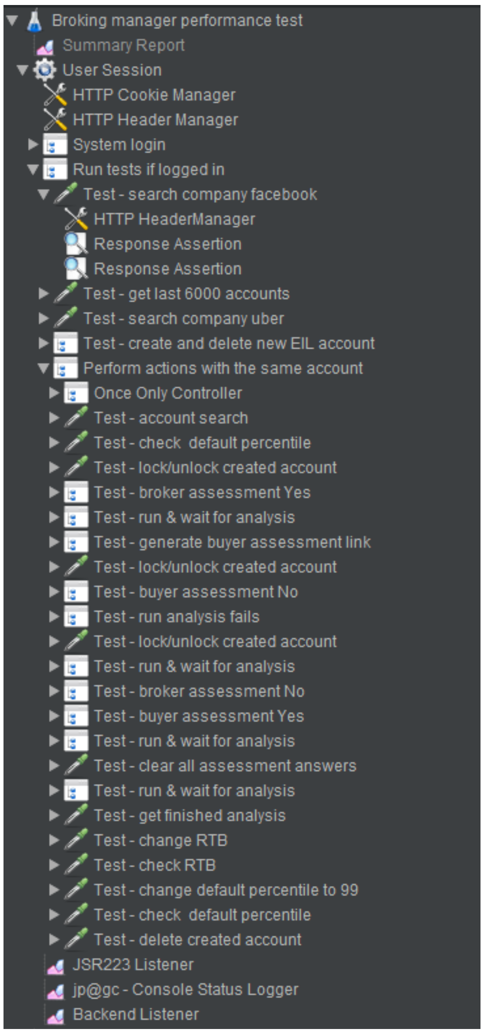

```
▼ Broking manager performance test
    Summary Report
  ▼ User Session
      HTTP Cookie Manager
      HTTP Header Manager
    ▶ System login
    ▼ Run tests if logged in
      ▼ Test - search company facebook
          HTTP HeaderManager
          Response Assertion
          Response Assertion
      ▶ Test - get last 6000 accounts
      ▶ Test - search company uber
      ▶ Test - create and delete new EIL account
      ▼ Perform actions with the same account
        ▶ Once Only Controller
        ▶ Test - account search
        ▶ Test - check  default percentile
        ▶ Test - lock/unlock created account
        ▶ Test - broker assessment Yes
        ▶ Test - run & wait for analysis
        ▶ Test - generate buyer assessment link
        ▶ Test - lock/unlock created account
        ▶ Test - buyer assessment No
        ▶ Test - run analysis fails
        ▶ Test - lock/unlock created account
        ▶ Test - run & wait for analysis
        ▶ Test - broker assessment No
        ▶ Test - buyer assessment Yes
        ▶ Test - run & wait for analysis
        ▶ Test - clear all assessment answers
        ▶ Test - run & wait for analysis
        ▶ Test - get finished analysis
        ▶ Test - change RTB
        ▶ Test - check RTB
        ▶ Test - change default percentile to 99
        ▶ Test - check  default percentile
        ▶ Test - delete created account
    JSR223 Listener
    jp@gc - Console Status Logger
    Backend Listener
```

Figure 6 Performance scenario

5.5 Performance scenario build

As described in section 4.3 testing type should be chosen by the objective and for creating a test scenario to be run after each deployment, a load testing approach is chosen.

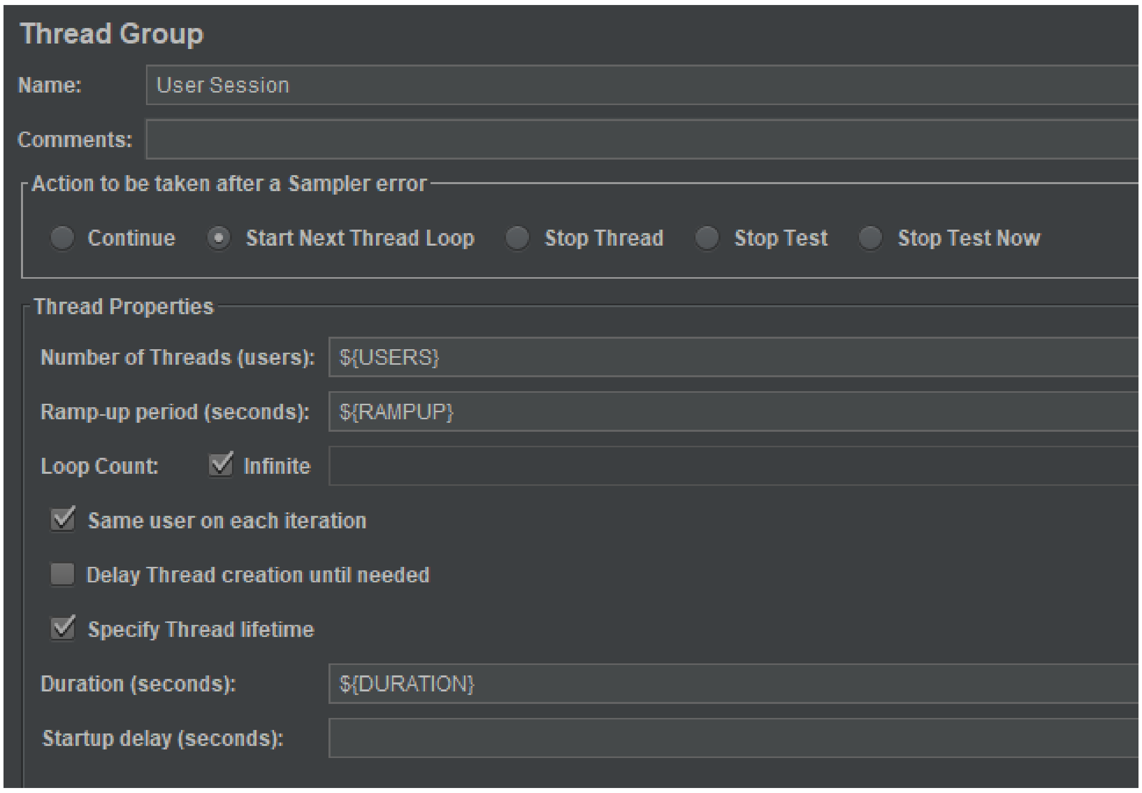
According to the case study findings by Simon Eismann, Diego Elias Costa, et [56] due to per-request pricing of the serverless models, there is a linear relationship between a cost and number of requests in a performance test. Meaning that a performance test with 500 requests per second costs 100 times more than performance test with 5 requests per second. This study also found that increasing the load from 5 requests per second to 500 requests per second did not result in visible stability increase.

Taking this finding into consideration, this study focuses on smaller amounts of requests per second. Another study on conducting repeatable experiments in cloud environment

[41] found that for reliable results are not achieved with single trial, therefore in this study each performance setup has 10 repetitions to insure stability.

Additionally, considering that application can save information to cache memory [57] and allowing lambdas to be scaled down to zero between tests, 20 minute delay between each repetition is introduced.

Following the performance requirements of at least 5 requests per second for at least 1 minute, the designed solution has a series of test runs from 5 to 15 threads (users) with a duration of 10 minutes and ramp-up of one minute to further mitigate test result fluctuation. All these parameters were configurable for each test run as shown in Figure 7 below.


Thread Group

Name: User Session

Comments:

Action to be taken after a Sampler error: ○ Continue [● Start Next Thread Loop] ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): ${USERS}

Ramp-up period (seconds): ${RAMPUP}

Loop Count: ☑ Infinite

☑ Same user on each iteration

☐ Delay Thread creation until needed

☑ Specify Thread lifetime

Duration (seconds): ${DURATION}

Startup delay (seconds):

Figure 7 Performance testing configuration

5.6 Performance test execution

Performance test have to be executed repeatedly, which requires help from an automated performance testing tool.

Performance testing tool must allow user to configure testing scenario (section 5.4) and its configuration (from section 5.5) while measuring the appropriate response time.

Based on initial needs, open source software Apache JMeter [1] [13] was chosen for writing a performance scenario. Apache JMeter is Java-based software designed for load testing functional behaviour and measuring performance [13].

JMeter is a popular tool for conducting performance tests, because of its flexibility. The chosen tool allows to run requests in specific order, organize requests into groups, add logic controllers to manage the requests and add assertions to validate the responses [58, 59].

As shown in Figure 6 above, specific scenario is written using JMeter tool to utilize scenario automation process and possibility to configure build according to requirements (section 5.5). Performance scenario automation is shown in Figure 7 below.

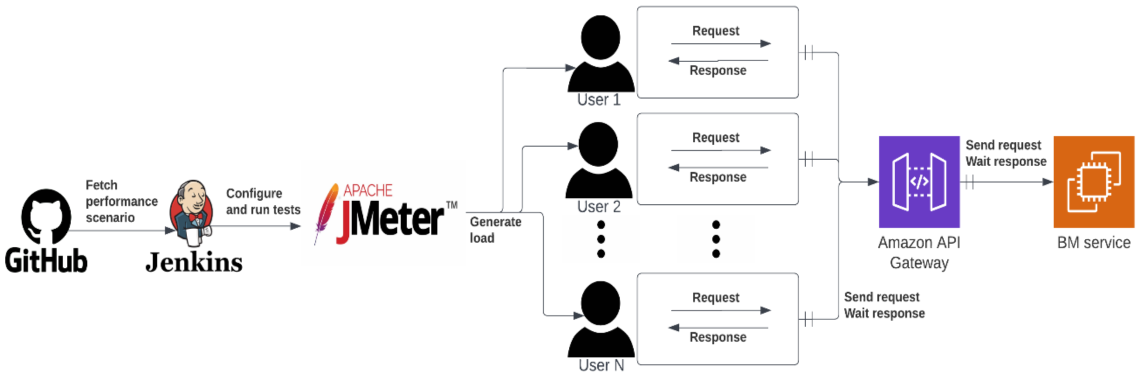


Figure 8 Performance scenario automation flow

5.7 Performance test analysis and reporting

Final step of conducting a performance test is analysing the results and reporting findings [51, 60]. Report must provide an overview whether the application meets the performance requirements described in section 5.2 and show which components have to be investigated further for root cause analysis of detected issues.

[1] https://jmeter.apache.org/

Performance test execution results and analysis are profoundly described in the next chapter.

6 Results gathering and validation

The research process was to run several iterations of load tests of BM application and compare traditional performance metrics (such as client side response time and throughput) with so-called cloud-provider metrics which are lambda duration and execution time and server-side metrics.

This work analyses how stable are traditional performance metrics and server side metrics. We compare both metrics to see if they provide the same results and analyse if we can use the cloud-provider metrics to assess the performance of the application.

In this thesis, we create a load testing scenario to evaluate a key feature of this application.

6.1 Traditional performance metrics

This chapter focuses on performance metrics gathered by JMeter:

- Client side response time
- Throughput
- Network I/O

The first performance configuration has 5 threads per second and produces on average 7200 requests for the duration of the test, which calculates up to 12 requests per second. Second performance setup has 10 threads per second with an average of 24 requests per second and the third setup has on average 26 requests per second.

The throughput doubles between first and the second setup, yet in the third setup we can see similar throughput as in the second. The reason for that is the increased number of requests increases the response time which leads to decline of the overall throughput. Similar observations can be done for Network input and output. These two metrics separately from response time do not give any additional information about application performance.

Table 3 Overall performance results

Tests average	Number of Requests	Error %	Average response time (ms)	Throughput Transactions/sec	Network (KB/sec) Received	Network (KB/sec) Sent
Configuration 1	7257.7	0.0067	141	12.09	7.39	21.05
Configuration 2	14347.6	0.0016	166	23.87	14.72	41.88
Configuration 3	15652.8	0.1390	352	25.94	16.22	45.43

Main workflow under test is creating a new account and running an initial analysis which is done in 20% of all requests, rerunning an analysis is 60% of all executed requests and the rest 20% are other actions not considered in this performance test.

Running an analysis (initial or rerun) can also be divided into two parts, since all of the calls are asynchronous, the requests have to be designed the same way. Each analysis run has 1 analysis start request to 5 requests of long-pulling an analysis response every 0.3 second.

In all setups, it can be seen that the first minute of each test takes the most time. In the first minute test creates a new account and starts an initial analysis. The cause for this is lambda cold-start time, this call wakes up all the related lambdas which are initially scaled down to zero and starting up again takes more time than it would for normal service.

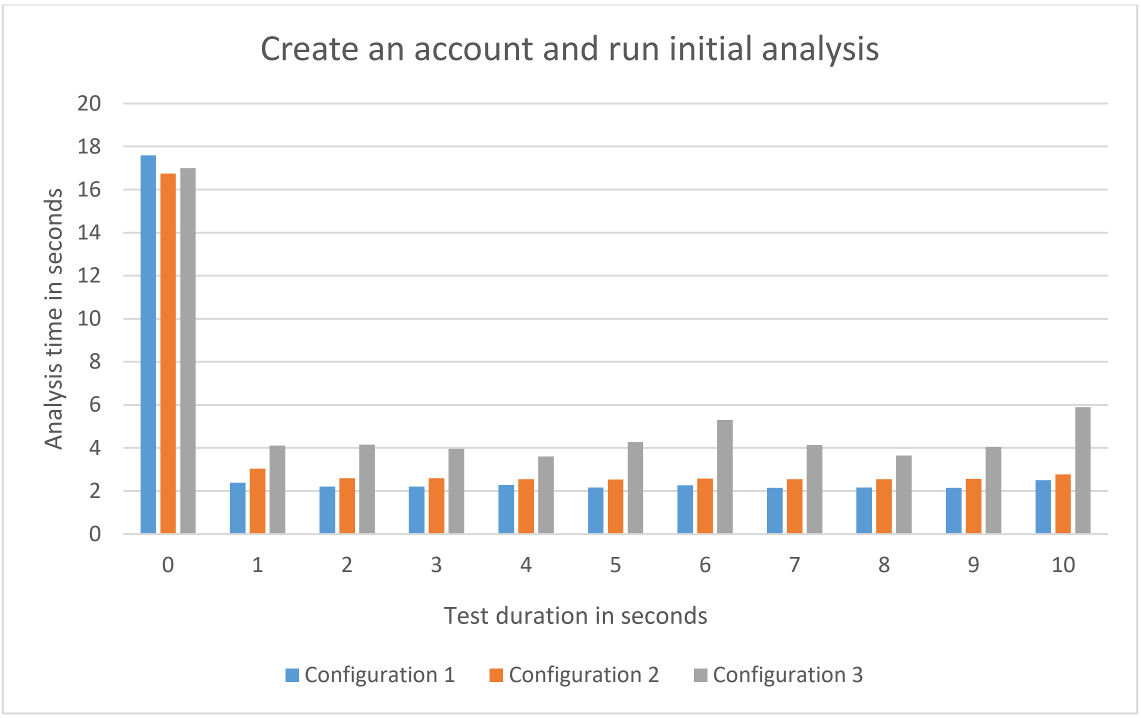


Figure 9 Initial analysis run time

If we take out the first minute of each test, overall trend changes and response times between tests are quite stable.

The second part of the test setup is to rerun the analysis after conducting changes. All these requests are finished under 3 seconds, which follows the performance requirements.

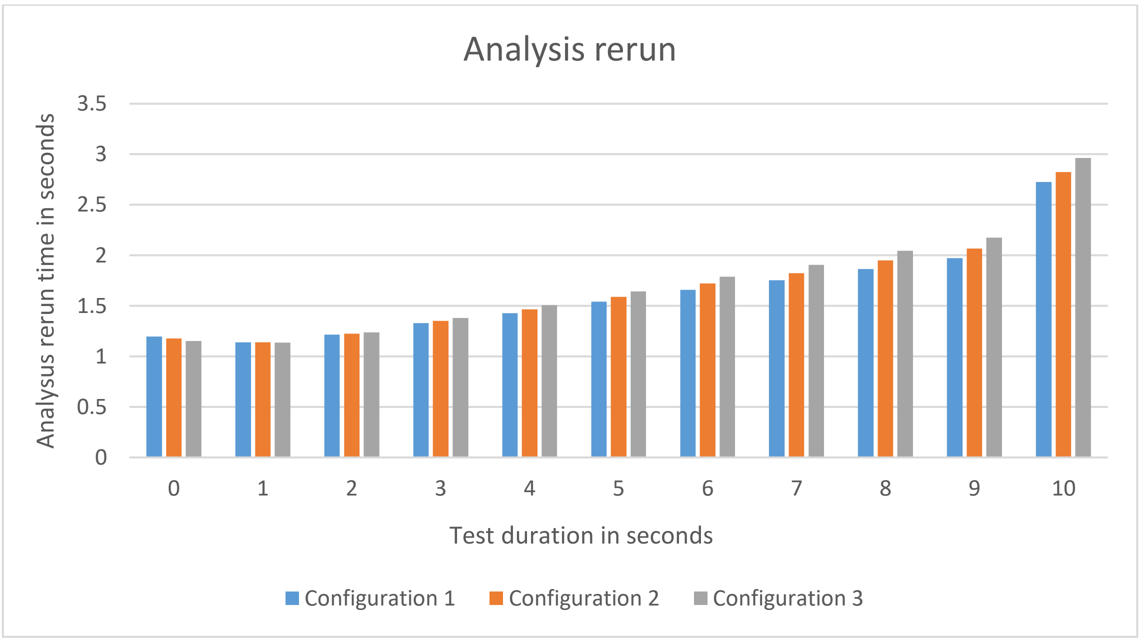


Figure 10 Analysis rerun time

Taking out the initial lambda cold star timing, both analysis runs show that the longer the test runs, the longer it takes for an analysis to finish.

For configuration of 15 threads, there are test that take more than recommended 5 seconds and several analyses take even longer than required 10 seconds. There are additional errors present, most of the errors are caused by API Gateway timeout and couple of errors are connected to external services.

Table 4 Configuration 3: Average analysis run time

Elapsed Time (granularity : 1 min)	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**	**Test 7**	**Test 8**	**Test 9**	**Test 10**
0	17.14	15.15	17.28	18.06	17.17	17.39	18.12	16.50	17.01	16.12
1	4.00	3.51	4.55	4.20	4.96	3.70	4.61	4.35	3.17	4.13
2	6.26	3.56	5.06	3.64	4.01	4.03	4.08	3.84	3.25	3.78
3	3.65	3.56	4.27	5.12	3.99	3.69	4.28	3.66	3.63	3.76
4	3.78	3.78	4.26	3.30	3.98	3.11	4.23	3.27	3.18	3.23
5	3.13	3.45	10.11	3.60	4.21	3.52	3.77	3.46	3.80	3.77
6	3.59	3.23	19.70	3.98	3.49	3.65	4.24	3.48	3.55	4.11
7	3.86	4.33	4.64	4.47	3.78	3.95	6.03	3.46	3.33	3.51
8	3.71	3.47	4.12	4.01	4.06	3.37	3.63	3.09	3.32	3.71
9	3.45	3.85	4.40	5.47	6.32	3.09	3.66	3.09	3.58	3.68
10	3.59	13.60	5.20	36.23	4.24	8.89	3.41	16.14	2.88	3.94
Average response time	5.11	5.59	7.60	8.37	5.47	5.31	5.46	5.85	4.61	4.89

RQ1. How stable are the performance tests results of a microservice and serverless application in a cloud-based environment?

Findings. With sufficient repetitions, the performance tests are stable enough to see reoccurring issues and find performance degradation. The mean response time from all of the repetitions has to be compared to the actual performance requirement.

6.2 Cloud service provider metrics

In this chapter we observe all performance runs using AWS CloudWatch[1] tool that collects and visualizes all lambda logs and metrics and in addition to serverless metrics, we will also analyse server-side metrics collected by Datadog[2] application:

- AWS Lambda duration
- AWS Lambda concurrent executions
- CPU usage
- Memory usage
- Network I/O

6.2.1 Lambda metrics

The first performance test setup has 370 concurrent lambda invocations for running an analysis, second setup has 730 and the last one has 967 concurrent lambda invocations. Seeing that each analysis run consist of running 5 different lambdas, the invocation number is matching the figures above.

Traditional metrics showed that initial analysis run is taking more time and we can observe the same behaviour from lambda runtime. The initial lambda invocation is definitely slower, although the difference is not as big as shown in the analysis runs.

[1] https://aws.amazon.com/cloudwatch/

[2] https://www.datadoghq.com/

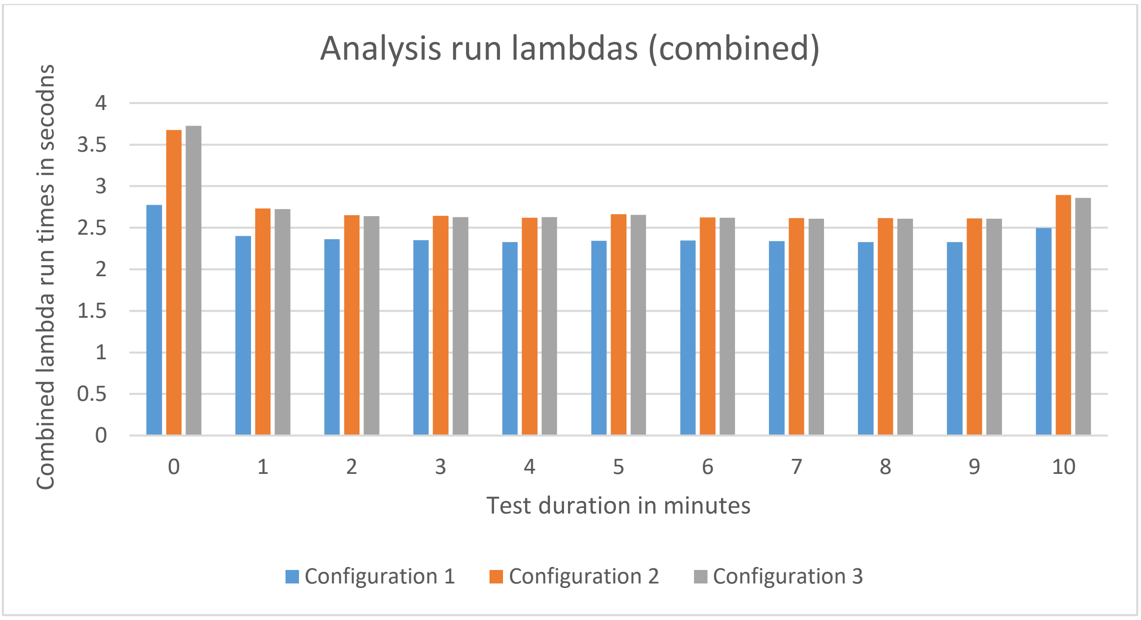


Figure 11 Combined analysis lambda runs

RQ2. How do the lambda cold starts affect the application performance?

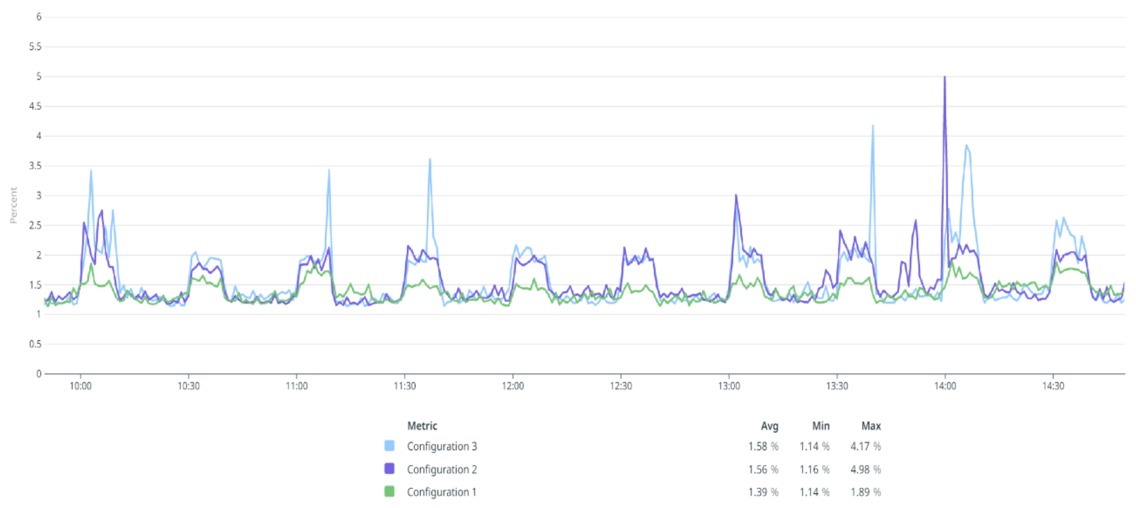
Findings. Lambda cold starts happen only within the first minute of the tests, which is also the warm up period. Since lambdas are scaling themselves down to zero, starting up again takes more time than it would for normal service.

If we take out the first minute of the test, we see that lambda runtimes are stable and so is the overall the analysis response time. If a cold start happens later than the warm up period, it is not detected by the test results.

6.2.2 Server-side metrics

Resource utilization key metric is CPU percent utilization, which represents how much of the CPU's processing power is being utilized in any state. CPU usage is expected to increase with running performance tests with a higher number of concurrent users, but the usual threshold should 1-5% for small requests and 80-100% for computation heavy requests.

During performance testing of the application, we can see in the Figure 12 that the CPU usage does not increase above 5%.



	Metric	Avg	Min	Max
🟦	Configuration 3	1.58 %	1.14 %	4.17 %
🟪	Configuration 2	1.56 %	1.16 %	4.98 %
🟩	Configuration 1	1.39 %	1.14 %	1.89 %

Figure 12 CPU usage

Memory usage is not scaled up or down between the test runs, but it is consistent over the period of time.

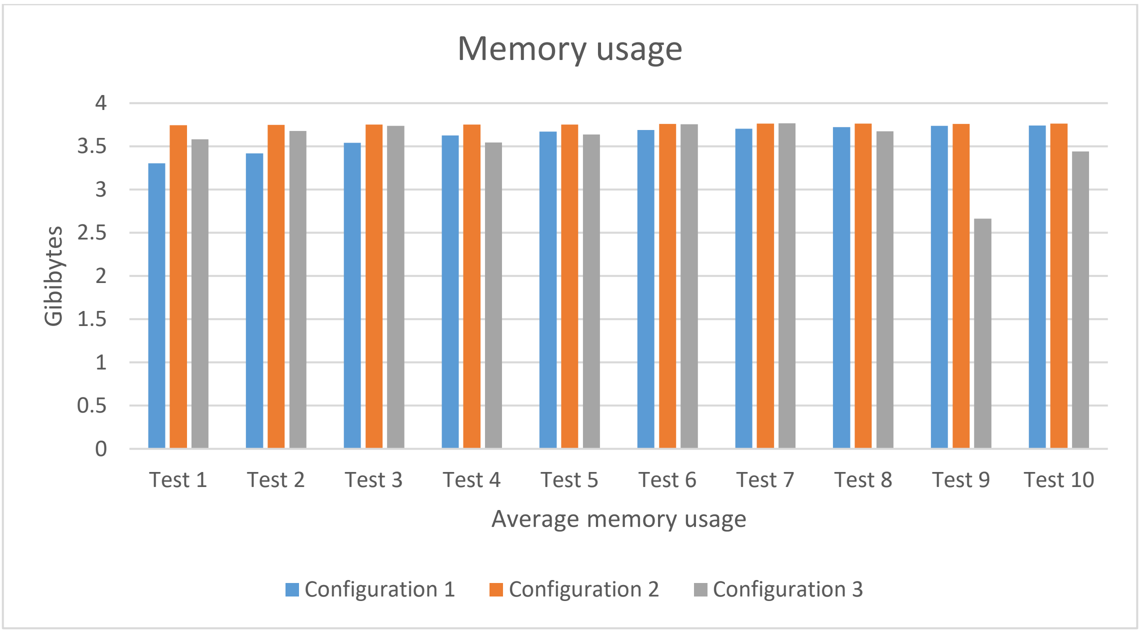


Figure 13 Application memory usage during test

Network input and output metric is important to monitor because microservices are distributed and communicate with each other via series of calls over network.

Measuring network utilization with traditional metrics, as seen in Table 3, the data being sent and its overall throughput is seen as limited, yet when comparing that to network communication on-going on the background and captured by system metrics (Figure 14 and Figure 15), the network input and output capacity is significantly higher.

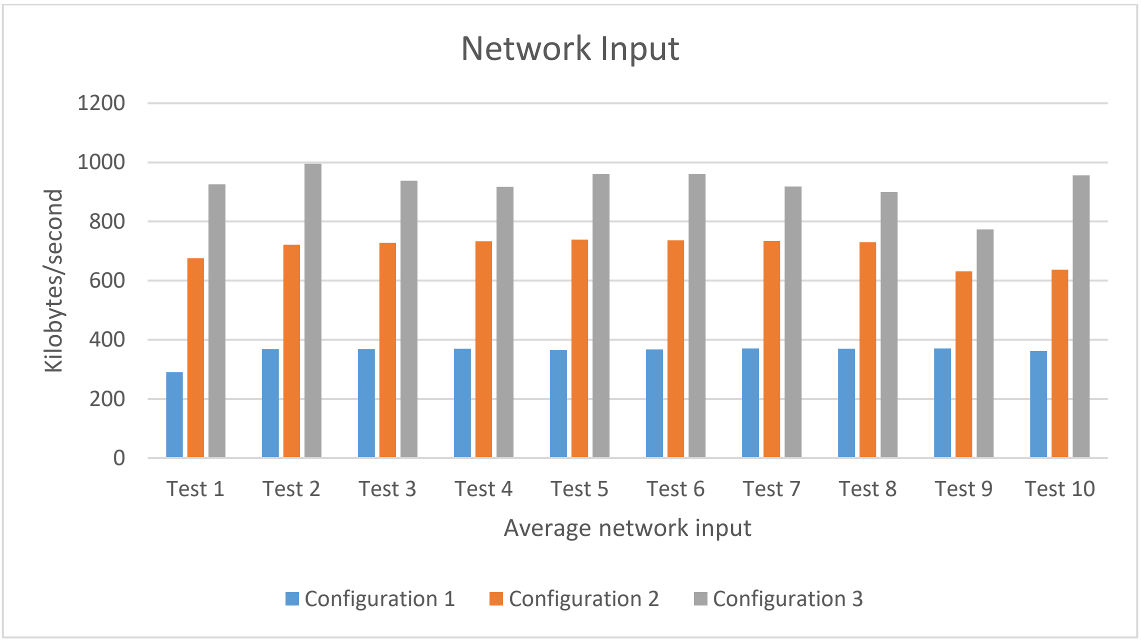


Figure 14 Average network input during test

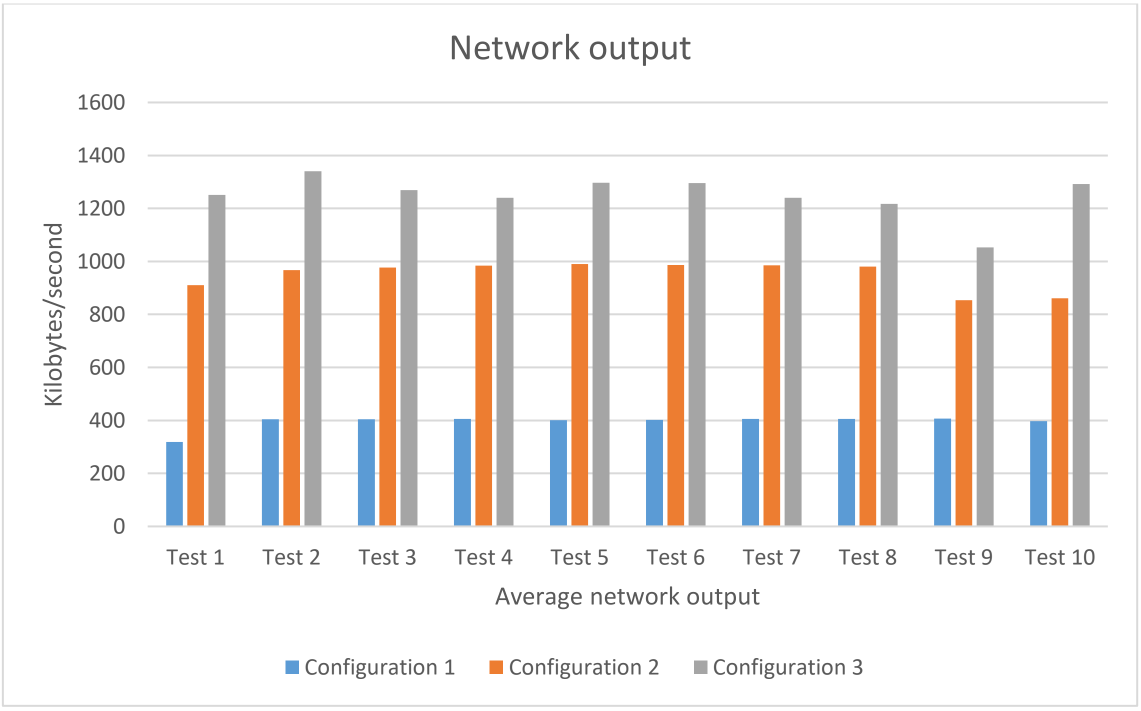


Figure 15 Average network output during test

6.3 Comparison of performance results

RQ3. Do cloud service provider metrics help assess the performance of an application?

Findings. Using lambda duration as a metric for analysing performance is not effective. All lambda duration results from all performance tests have similar run time. However, traditional response time shows that for setups 1 and 2, the results are rather similar (including the API Gateway, connecting service, and lambda start-up time). But for the biggest load, lambda duration time is not showing any issues, which is correct, but for the user the response time has grown quite a lot.

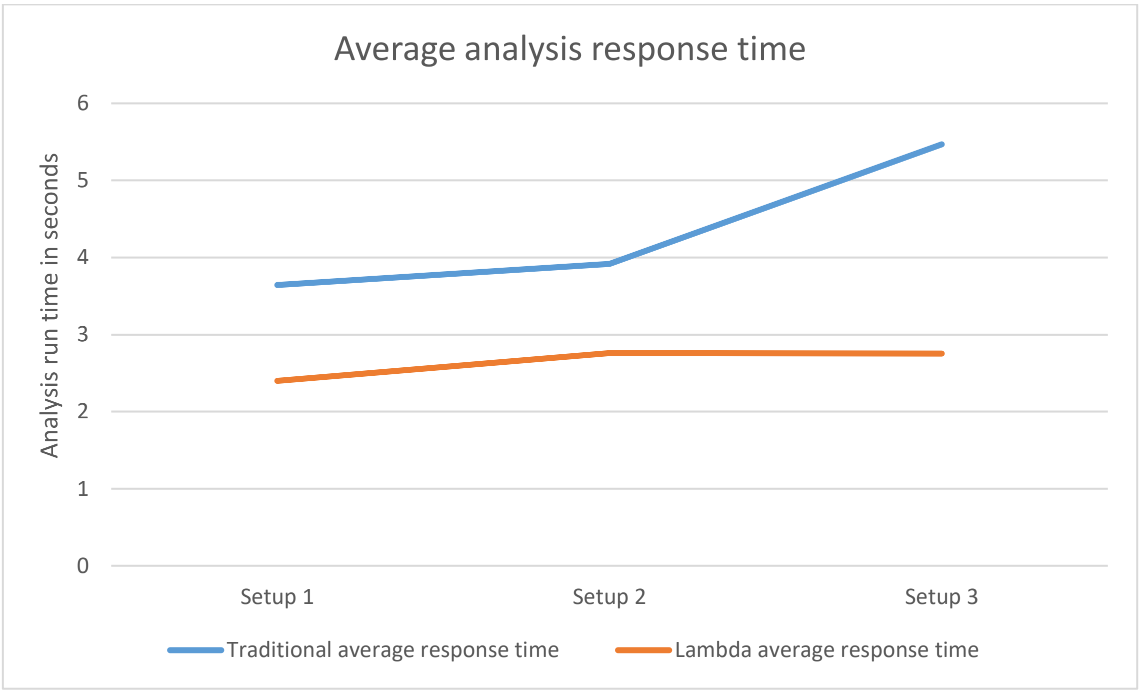


Figure 16 Response time comparison

Taking this into consideration, lambda performance metrics can be used for performance analysis, but only for specifically lambda performance. Test results show that that lambda performance does not fluctuate, except for the lambda cold-start time during the first minute of the test. All other experiments with different configurations and executions, do not affect lambda performance at all.

Test results comparison shows that the actual response times for users were decreasing for a bigger user load. Such observation would mean that the performance issues are not in the specific lambdas under test, but in the services or infrastructure surrounding the application.

These issues could not be found with any of the cloud service provider metrics, but only with traditional client-side response time gathering.

Additionally, traditional performance metrics actually show the performance requirements errors, which can lead to possible bottlenecks and areas to be investigated. During testing, there were several errors on gateway timeout, as described in section 3.2 API Gateway timeout limit is 29, which means that some calls are taking longer time and connection to API Gateway is terminated. Also, results show workflows where analysis is taking longer than required 10 seconds. As seen from the results, model calculation lambdas cannot be the source of these issues, the average lambda execution time was under 3 seconds.

Further investigation shows that there are different workflows used when running an initial analysis or when re-running an analysis. Initial analysis run included additional steps to create an account replica in User collaboration service (UCS) and lock the account for analysis run via Locking Service (LS):

- Locking service (LS) a helping lambda in BM application that is locking the account before making any changes. When running an initial analysis, locking service is called inside the application but when user is rerunning an analysis, the lock has to be set before the analysis run. Locking service can be considered as one source of the additional response times, but when analysing the actual lambda results shown in Figure 17, we see that LS is only problematic for the first run (cold-start) and after that the results are stable across all configurations.

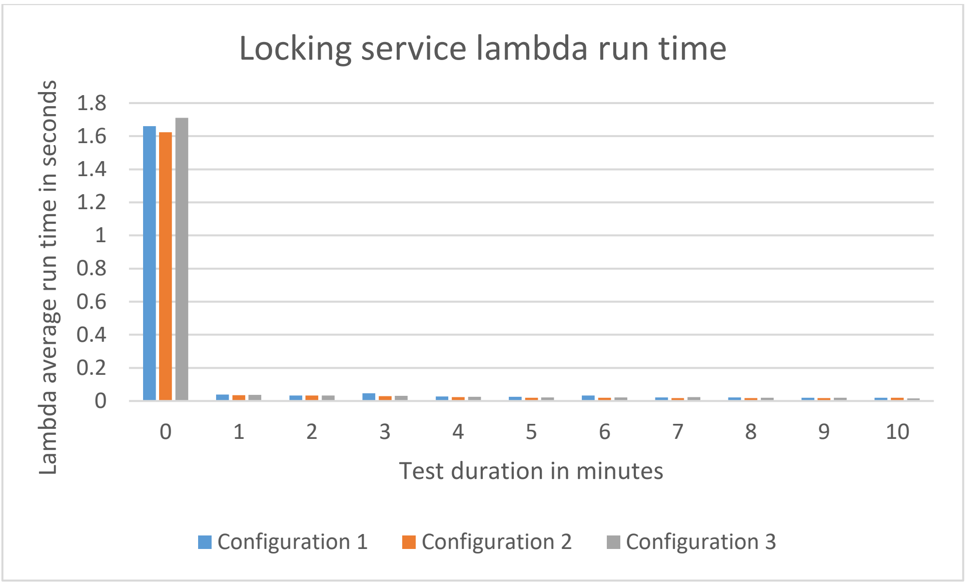


Figure 17 Locking service lambda

- User Collaboration Service (UCS) is called each time a new company is created. The UCS shares a newly created account with every other user in a user group. From UCS logs shown in Figure 18, it can be seen that with each configuration and higher user load, the response time has grown dramatically. From average duration of 800 milliseconds up to 1300 milliseconds with a maximum duration of 5 seconds. UCS is an external service, which will need a further analysis of its expected performance and application usage.

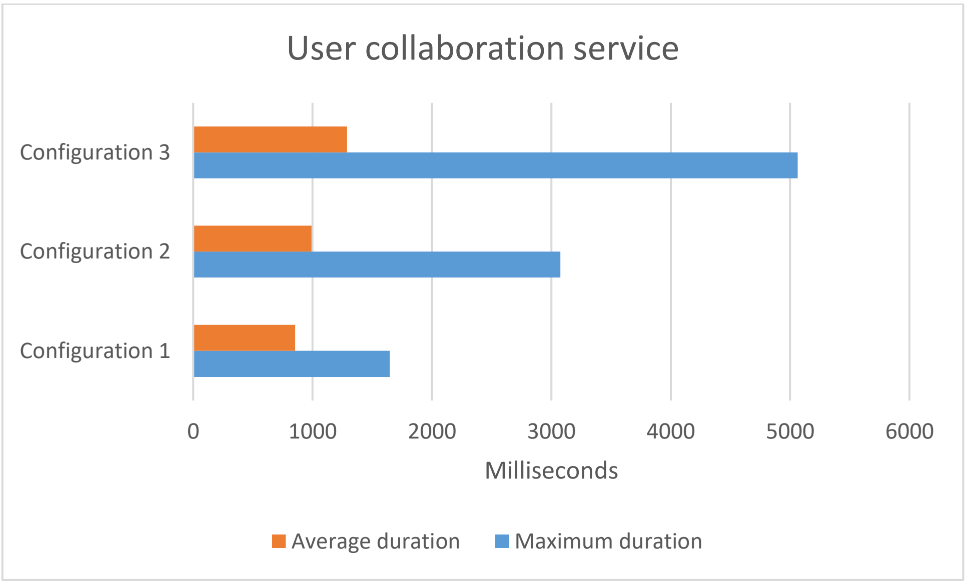


Figure 18 User Collaboration service

6.4 Limitations to current work

This work did not further analyse any of the external services shown in Figure 5. Performance analysis has to be expanded for the whole platform with full microservice infrastructure and it is hard to analyse the external services without knowing their architecture.

Broking manager core service itself is analysed during this work as means to forward API calls from one service to another, it could add a lag time for transferring API calls and will need to be analysed further.

This paper also did not analyse the infrastructure aspects, for example API Gateway. The API Gateway itself can introduce increased response times due to the additional network calls and if not scaled properly, the API Gateway can become a bottleneck on its own [61].

Additionally, the current solution did not analyse the database load. There are several different data storages used by the application (DynamoDB[1] [62], Redis[2] [63], MySQL[3] [64], Neptune[4] [65]) which all have their own performance metrics and each of these could be a possible bottleneck in application performance [66, 67].

6.5 Threats to validity

The reason for conducting this paper and following experiments, was to analyse if we can compare traditional and cloud provided metrics for performance analysis in a cloud-based environment. A single system (BM), a single cloud service provider (AWS) and a limited number of configurations was used to conduct the experiments.

---

[1] https://aws.amazon.com/dynamodb/

[2] https://redis.io/

[3] https://www.mysql.com/

[4] https://aws.amazon.com/neptune/

Limited number of configurations may have an impact on the conclusions. However, since the purpose was not to analyse the performance itself but rather compare performance metrics, the number of configurations should not be a major factor.

AWS is one of the most popular engines for deploying microservices and Broking manager application is used because it has a microservice based architecture. BM is actually a really good example following the serverless research conducted by Datadog in 2022 [33] of today's applications, where each service can use their own language, database and most of the functions are built serverless.

Author is convinced that the overall conclusions can be transferred to other cloud service providers and applications.

Further studies should investigate whether these findings apply to other microservice-based applications in different cloud environments.

6.6 Comparison with related work

Current paper used two earlier conducted case studies [18, 56] as basis for this research.

The main work by Simon Eismann, Cor-Paul Shang Bezemar, et [18] does conclude that performance testing in a cloud based environment is a nightmare for the performance testers, since there are problems with stability of the environment and reproducibility of the experiments. In their research, they offered a new research direction of analysing the stability of traditional performance metrics, which current work also found that metrics are reliable only with sufficient amounts of repetitions. Another research direction that was offered, is to assess if the cloud service provider metrics can be used to analyse application performance. In current work, it was found that neither server nor serverless metrics were sufficient enough to show decrease in application performance. (RQ3)

The second research conducted by the same authors also investigates the performance tests for specifically serverless applications [56]. In that research they had key findings regarding warm-up period being less than 2 minutes and cold starts occurring later in the test, do not impact the measurements. Both findings can be confirmed by current work. In all cases, the lambda cold-start time was present only during the first minute of the test. (RQ2).

Another key finding was that there are short-term performance fluctuations during the study, which was also expected when conducting the current study and therefore mean response times from repetitive runs were taken when analysing the results. (RQ1)

Overall, performance benchmarking in the cloud environments [68, 69] has been broadly studied in the past couple of years, the focus of these studies has not included the performance testing and analysis of microservice-based applications in cloud environments. Even the main studies mentioned above, they either analysed the performance testing on strictly serverless application or on microservice container based application and there is no paper that analyses the challenges that arise when using both approaches.

6.7 Suggestions for future work

6.7.1 Cloud service provider metrics

One of the key findings in this paper was that data gathering from cloud service providers is really simple and convenient. CloudWatch and Datadog provide a wide range of metrics and both applications provide good visualisation and analytical possibilities out of the box, yet also provide several different options of exporting the data. The performance metrics are kept over a long period of time, allowing stakeholders to see how the performance of the system has changed over time.

The future work should analyse how to better utilize the cloud service provider metrics. The tools provide a much wider range of metrics. For example, server health, resource, deployments overviews and lambda cost accumulation, cold starts, concurrency utilisation, and many more. Further studies need to analyse which metrics from cloud service providers are better at showing performance regression and which of those metrics could be included into performance monitoring [70], excluding the need to run extensive performance tests.

Additionally, moving to one source of data would make it easier to analyse the metrics. Datadog allows importing AWS lambda logs, which means that all logs could be moved to one place and since Datadog provides options to visualise and analyse AWS and system metrics, all these things can be combined.

6.7.2 JMeter

Current JMeter solution with traditional metrics gathers results from each performance run separately, therefore running several repetitions of the same configuration is causing overhead of combining the results across different runs. All ten runs have their own statistics report, which need to be summed up for comparison.

As a next step, JMeter report gathering has to be improved to combine several repetitions of the same performance run into a single result, which can be analysed in a more convenient way.

6.7.3 Other services

Also, including the database and API gateway load analysis into the gathered metrics to better analyse the issues. After which the further analysis can be done on the most time consuming component or most used component, this way the optimization effect on the system is more visible [71].

7 Summary

Microservice and cloud based architecture provides a vast world of opportunities and ways to implement software. All of this comes at the cost of testing.

Finding a correct way to run performance tests and making sure that metrics we collect and measure provide us accurate information on the application performance is crucial to any business.

The goal of this work was to develop a performance testing suite to analyse and compare traditional performance testing metrics with cloud service provider specific metrics. The research was based on a classical approach for conducting a performance test with additions from different researches using microservices and cloud based architecture.

Most important findings of the current work are:

- There are fluctuations between performance runs. In order to produce stable performance test results, a sufficient amount of repetitions has to be conducted and mean response time from all repetitions has to be compared to the actual performance requirement. (RQ1)

- Lambda cold starts happen only within the first minute of the tests and do not affect the further performance test results. (RQ2)

- Cloud service provider metrics alone were insufficient to show decrease in application performance. (RQ3)

In summary, traditional performance metrics do provide a sufficient number of metrics for performance analysis of the whole system, but with microservices further investigation is needed to analyse the issues. Combination of both performance metrics is needed to identify which service performance has declined, which means that the performance testing will take more time and the tester has more responsibility to learn the whole system architecture.

References

[1] International Institute of Business Analysis, BABOK v3 - A Guide to the Business Analysis Body of Knowledge, Toronto: International Institute of Business Analysis, 2015.

[2] GeeksforGeeks, "Introduction of Deadlock in Operating System," GeeksforGeeks, 23 02 2022. [Online]. Available: https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/. [Accessed 27 12 2022].

[3] M. Kaur, "Testing in the Cloud: New Challenges," *International Conference on Computing, Communication and Automation (ICCCA),* pp. 742-746, 2016.

[4] Microsoft, "What is a cloud service provider?," Microsoft, [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-cloud-provider/. [Accessed 27 12 2022].

[5] Tutorials Point, "SDLC - Overview," Tutorials Point, [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm. [Accessed 27 12 2022].

[6] C. HARRIS, "Microservices vs. monolithic architecture," Atlassian logo, [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith. [Accessed 27 12 2022].

[7] R. Krajewski, "Serverless vs Microservices: What Should You Choose For Your Product?," Ideamotive, 16 02 2022. [Online]. Available: https://www.ideamotive.co/blog/serverless-vs-microservices-architecture. [Accessed 13 11 2022].

[8] IBM Cloud Education, "FaaS (Function-as-a-Service)," 30 7 2019. [Online]. Available: https://www.ibm.com/cloud/learn/faas. [Accessed 13 11 2022].

[9] Amazon AWS, "AWS Lambda," [Online]. Available: https://aws.amazon.com/lambda. [Accessed 13 11 2022].

[10] CyberCube, "Insurance Advisory for Brokers," CyberCube, [Online]. Available: https://www.cybcube.com/solutions/insurance-brokers/. [Accessed 13 11 2022].

[11] Red Hat, Inc., "What does an API gateway do?," Red Hat, Inc., 08 01 2019. [Online]. Available: https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do. [Accessed 27 12 2022].

[12] Amazon Web Services, Inc., "Amazon SQS," [Online]. Available: https://aws.amazon.com/sqs/. [Accessed 16 12 2022].

[13] The Apache Software Foundation, "Apache JMeter™," [Online]. Available: https://jmeter.apache.org/. [Accessed 13 11 2022].

[14] Microsoft, "Chapter 15 — Measuring .NET Application Performance," 14 7 2010. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff647791(v=pandp.10)?redirectedfrom=MSDN. [Accessed 13 11 2022].

[15] K. Z. J. F. Y. Li, “Research the performance testing and performance improvement strategy in web,” in *2010 2nd international Conference on Education Technology and Computer,* Shanghai, 2010.

[16] E. S. &. W. S. &. M. M. Arif, “Empirical study on the discrepancy between performance,” *Empirical Software Engineering,* vol. 23, no. 3, pp. 1490-1518, 2018.

[17] I. Sacolick, “What is CI/CD? Continuous integration and continuous delivery explained,” InfoWorld , 15 04 2022. [Online]. Available: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html. [Accessed 27 12 2022].

[18] S. &. B. C.-P. &. S. W. &. O. D. &. v. H. A. Eismann, “Microservices: A Performance Tester's Dream or Nightmare?,” *ICPE '20: Proceedings of the ACM/SPEC International Conference on Performance Engineering,* p. 138–149, 2020.

[19] P. L. &. C.-P. Bezemer, “An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Project,” *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering,* pp. 373-384, 2017.

[20] W. S. &. A. E. H. &. M. N. &. P. Flora, “Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters,” *ICPE '15: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering,* pp. 15-26, 2015.

[21] Lars Rabbe, “Skype,” 12 2010. [Online]. Available: http://blogs.skype.com/en/2010/12/cio_update.html?cm_mmc=PXBL|0700_B6-_-downtime-20101229. [Accessed 13 11 2022].

[22] Sanket, “The exponential cost of fixing bugs,” DeepSource Corp., 29 1 2019. [Online]. Available: https://deepsource.io/blog/exponential-cost-of-fixing-bugs/. [Accessed 13 11 2022].

[23] L. W. &. J. T. &. E. E. O. Kao, “MicroRCA: Root Cause Localization of Performance Issues in Microservices,” *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium,* pp. 1-9, 2020.

[24] APMdigest, “16 Ways Application Performance Impacts the Business - Part 1,” APMdigest, 16 11 2015. [Online]. Available: https://www.apmdigest.com/16-ways-application-performance-impacts-business-1. [Accessed 13 11 2022].

[25] UNext Jigsaw, “What is Microservices in cloud?,” 28 11 2020. [Online]. Available: https://www.jigsawacademy.com/blogs/cloud-computing/what-is-microservices/. [Accessed 14 04 2022].

[26] M. Anastasov, “The Cracking Monolith: The Forces That Call for Microservices,” 29 04 2022. [Online]. Available: https://semaphoreci.com/blog/2017/03/21/cracking-monolith-forces-that-call-for-microservices.html. [Accessed 13 11 2022].

[27] A. Kharenko, “Monolithic vs. Microservices Architecture,” 9 10 2015. [Online]. Available: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59. [Accessed 13 11 2022].

[28] A. Amanse, “Why should you use microservices and containers?,” 13 9 2020. [Online]. Available: https://developer.ibm.com/articles/why-should-we-use-microservices-and-containers. [Accessed 14 4 2022].

[29] IBM Cloud Education, “Microservices,” IBM, 30 03 2021. [Online]. Available: https://www.ibm.com/cz-en/cloud/learn/microservices. [Accessed 13 11 2022].

[30] D. C. &. C.-P. B. &. P. L. &. A. Andrzejak, “What’s Wrong with My Benchmark Results?,” *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* vol. 47, no. 7, pp. 1452-1467, 2021.

[31] E. &. I. A. &. G. J. &. E. S. &. B. A. &. V. L. &. T. L. &. S. N. &. H. N. &. A. C. Eyk, “The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms,” *IEEE Internet Computing,* vol. 23, no. 6, pp. 7-18, 2019.

[32] K. F. A. G. a. A. Z. Maciej Malawski, “Benchmarking Heterogeneous Cloud Functions,” in *Fifteenth International Workshop HeteroPar'2017 Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms*, Santiago de Compostela, 2017.

[33] Datadog, “The state of serverless,” 06 2022. [Online]. Available: https://www.datadoghq.com/state-of-serverless/. [Accessed 13 11 2022].

[34] A. H. Ayush Maan, “Introduction to serverless,” IBM, 19 5 2019. [Online]. Available: https://developer.ibm.com/videos/introduction-to-serverless-video-page/?mhsrc=ibmsearch_a&mhq=serverless. [Accessed 13 11 2022].

[35] Microsoft Azure, “Azure functions,” [Online]. Available: https://azure.microsoft.com/en-gb/products/functions/. [Accessed 13 11 2022].

[36] Google Cloud, “Cloud Functions,” [Online]. Available: https://cloud.google.com/functions/. [Accessed 13 11 2022].

[37] R. C. &. W. S. &. W. J. Lloyd, “Predicting Performance and Cost of Serverless Computing Functions with SAAF,” *2020 IEEE Intl Conf on Dependable,* pp. 640-649, 2020.

[38] S. &. S. J. &. E. E. &. S. M. &. G. J. &. H. N. &. A. C. &. I. A. Eismann, “A Review of Serverless Use Cases and their,” 2020.

[39] S. E. &. J. G. &. E. v. E. &. N. H. &. S. Kounev, “Predicting the Costs of Serverless Workflows,” *ICPE '20: Proceedings of the ACM/SPEC International Conference on Performance Engineering,* pp. 265-276, 2020.

[40] P. C. K. C. P. C. S. F. V. I. N. M. V. M. R. R. A. S. P. S. Ioana Baldini, “Serverless Computing: Current Trends and,” in *Research Advances in Cloud Computing*, 2017, pp. 1-20.

[41] A. A. &. T. Brecht, “Conducting Repeatable Experiments in Highly Variable,” in *ICPE '17: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, 2017.

[42] Amazon Web Services, Inc., “Amazon API Gateway quotas and important notes,” [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html. [Accessed 14 11 2022].

[43] “What is Software Testing? Basics, Tutorial, Importance, Interview Questions,” tryqa.com, [Online]. Available: https://tryqa.com/what-is-software-testing/. [Accessed 14 12 2022].

[44] S. W. Ambler, “Technical (Non-Functional) Requirements: An Agile Introduction,” Ambysoft Inc., [Online]. Available: http://agilemodeling.com/artifacts/technicalRequirement.htm. [Accessed 14 12 2022].

[45] S. V. W. M. Shamila Makki, “Application of Mobile Agents in Managing the Traffic in the Network and Improving the Reliability and Quality of Service,” *IAENG International Journal of Computer Science,* vol. 33, no. 1, 2007.

[46] D. P. G. M. G. Damianos Gavalas, “Advanced network monitoring applications based on mobile/intelligentagent technology,” *Computer Communications,* vol. 23, no. 8, 1999.

[47] H. Malik, “AUTOMATED ANALYSIS OF LOAD TESTS USING PERFORMANCE COUNTER LOGS,” Queen’s University, Kingston, 2013.

[48] R. M. &. A. K. &. B. M. &. R. Subramanyan, “Performance Testing: Far From Steady State,” *IEEE Computer Software and Applications Conference Workshops,* pp. 341-346, 2010.

[49] SmartBear Software, “Most Effective Types of Performance Testing,” [Online]. Available: https://loadninja.com/articles/performance-test-types/. [Accessed 13 11 2022].

[50] C. L. &. P. Leitner, “An Evaluation of Open-Source Software Microbenchmark Suites for Continuous Performance Assessment,” *ACM/IEEE,* pp. 119-130, 2018.

[51] I. Molyneaux, The Art of Application Performance Testing, 2nd Edition, O'Reilly Media, Inc., 2014.

[52] E. J. W. &. F. I. Vokolos, “Experience with Performance,” *IEEE Transactions on Software Engineering,* vol. 26, no. 12, pp. 1147-1156, 2000.

[53] F. R. F. &. S. I. &. D. Suffian, “The Design and Execution of Performance Testing Strategy for Cloud-based,” 2014.

[54] Pegasystems Inc., “Ten best practices for successful performance load testing,” 10 09 2021. [Online]. Available: https://docs-previous.pega.com/performance/ten-best-practices-successful-performance-load-testing. [Accessed 14 12 2022].

[55] Google, “Welcome to Google Analytics,” [Online]. Available: https://analytics.google.com/analytics/web/provision/#/provision. [Accessed 13 11 2022].

[56] S. E. &. D. E. C. &. L. L. &. C.-P. B. &. W. S. &. A. v. H. &. S. Kounev, “A case study on the stability of performance tests for serverless,” *The Journal of Systems & Software,* vol. 28, p. 02, 2022.

[57] E. Taklai, “Performance Testing of X-Road Services by the Example of Estonian Business Register,” Tallinn Techincal University, Tallinn, 2020.

[58] T. Hamilton, “JMeter Tutorial for Beginners: Learn in 7 Days,” Guru99, 22 10 2022. [Online]. Available: https://www.guru99.com/jmeter-tutorials.html. [Accessed 13 12 2022].

[59] Meliora Ltd., “Integrating with Apache JMeter,” [Online]. Available: https://www.melioratestlab.com/resources/integrating-with-apache-jmeter/. [Accessed 14 12 2022].

[60] Microsoft Inc., “Chapter 1 – Fundamentals of Web Application Performance Testing,” 27 04 2010. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/bb924356(v=pandp.10). [Accessed 14 12 2022].

[61] Microsoft, “The API gateway pattern versus the Direct client-to-microservice communication,” 21 9 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern. [Accessed 14 11 2022].

[62] Amazon Web Services, “Amazon DynamoDB,” [Online]. Available: https://aws.amazon.com/dynamodb. [Accessed 14 11 2022].

[63] Redis Ltd., “Redis,” [Online]. Available: https://redis.io/. [Accessed 14 11 2022].

[64] Oracle, “MySQL,” [Online]. Available: https://www.mysql.com/. [Accessed 14 11 2022].

[65] Amazon Web Services, “Amazon Neptune,” [Online]. Available: https://aws.amazon.com/neptune/. [Accessed 14 11 2022].

[66] Techstrong Group, Inc., “Database Bottlenecks: The Hidden Cause of App Slow Downs?,” 5 6 2017. [Online]. Available: https://devops.com/database-bottlenecks-hidden-cause-app-slow-downs/. [Accessed 14 11 2022].

[67] G. D. &. A. P. &. W. Emmerich, “Early Performance Testing of Distributed Software Applications,” in *Proceedings of the Fourth International Workshop on Software and Performance,* Redwood Shores, 2004.

[68] J. S. P. L. Christoph Laaber, “Software Microbenchmarking in the Cloud. How Bad is it Really?,” *Empirical Software Engineering,* vol. 17, no. 24, p. 04, 2019.

[69] J. C. Philipp Leitner, “Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds,” *ACM Transactions on Internet Technology,* vol. 19, no. 3, p. 04, 2016.

[70] R. &. v. H. A. &. K. H. &. L. F. &. L. L. E. &. P. C. &. S. S. &. W. J. Heinrich, “Performance Engineering for Microservices: Research Challenges and Directions,” *ICPE '17 Companion: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion,* pp. 223-226, 2017.

[71] H. H. Liu, Software Performance and Scalability: A Quantitative Approach, Wiley, 2009.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Tatjana Kirotar

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Performance testing of microservices in cloud-based environment" , supervised by Aleksandr Kormiltsõn
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

02.01.2023

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.