UNVEILING ISSUES AND THEIR CAUSES IN OPEN-SOURCE SERVERLESS FRAMEWORK, AN EMPIRICAL STUDY

Lappeenranta–Lahti University of Technology LUT

Master in Software Product Management and Business, Master's thesis

2024

Khac Tam Nguyen

Examiner(s): Professor Kari Smolander

Muhammad Hamza

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Software Engineering

Khac Tam Nguyen

Unveiling issues and their causes in open-source serverless framework, an empirical study

Master's thesis

Year of completion of the thesis 2024

65 pages, 16 figures, and 4 appendices

Examiner(s): Professor Kari Smolander and Junior Researcher Muhammad Hamza M.Sc.

Keywords: serverless architecture, serverless framework, AWS, GCP, Azure, cloud computing, microservice, software architecture.

This empirical research - master thesis identifies and understands the problem underlying serverless frameworks' deployment and use, with an increased area of interest focused on the most popular open-source project, Serverless Framework. The master thesis motivation is brought about due to the adoption of serverless computing, which promises to cut the cost and provide scaling but also brings about other new challenges such as cold starts, security vulnerabilities, and application complexities amidst distributed applications.

This draws a grand total of over 1,809 issues analyzed from the context of the major problematics represented by these categories of topics. Some of the examples include networking, update and installation problems, performance problems, GUI concerns, and much more. This largely adds up to the contribution of the serverless architectures into different constituencies of the network communication types of problems. On top of that, there also lies an issue related to updating and installation in a large margin. This points at the factor of complexity that can bind compatibility and configuration of thousands of serverless services. One of the problems is performance: resource spending and speed increase.

Here is the cause that included architectural limitation, technical debt, design complexities, challenges in serverless edge computing, security concern, resource limitation, and challenges of monitoring and debugging the serverless application. Some of the issues are serverless-agnostic, while there are others specifically tied to The Serverless Framework.

This thesis contributes by detailing the usual issues and challenges surrounding the serverless frameworks, with even more detail directed at the serverless framework itself. The project provides a guideline and avenues that may be of help to developers, institutions, and academicians in treading within the serverless landscape, which in turn will lead to improved performance, security, and scale of applications. Future work will include a detailed cause analysis of the issues and the development of solutions that will help in mitigating the problems.

ACKNOWLEDGEMENTS

Thank you, Hamza, for the idea and the inspiration. Serverless was not my first attention but discussing with you sparks lots of ideas to me. Even though you did not have much time during the journey of this paper when it was being made, you already helped me with ideas and directions from the beginning.

Thank you, professor Smolander, for your support and trust. Without your trust, I would have not made this paper.

ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
BaaS	Backend-as-a-Service
CD	Continuous Development
CI	Continuous Improvement
CLI	Command Line Interface
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
FaaS	Function-as-a-Service
GCP	Google Cloud Platform
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
RQ	Research Question
SLAAC	Stateless Address Auto Configuration
TCP	Transmission Control Protocol
TSIG	Transaction Signature
UDP	User Datagram Protocol

Table of contents

Appendices

Appendix 1. Extraction code

Appendix 2. Issue detail extraction code

Appendix 3. Summary number of main themes and subthemes

Appendix 4. Main themes and subthemes categorization script

Figures

1. Introduction

1.1. An overview of serverless computing

Serverless computing is a big step in cloud services. It mixes spending less with doing more. In this model, the folks who run the cloud service handle the heavy lifting. They only charge for the time you use it. This setup helps businesses focus on their work, avoiding the tough work about managing resources. This makes it faster and easier to develop and grow apps. However, serverless computing also brings up new problems, mostly about keeping everything secure (Li et al., 2023).

Serverless computing is known for using managed, short-term, and stateless computing solutions. These are things like Function-as-a-Service (FaaS) and Backend-as-a-Service (BaaS). Both big companies and schools like these services more and more. This is because they help with operations. They allow those who make apps to focus on the business part, while the cloud services handle the tech side, like deploying, resource allocation, and autoscaling. This teamwork makes the development process cleaner and the apps more efficient (Eismann et al., 2021).

The new era opens with challenges, as always. Serverless architecture comes with many serverless frameworks, there are many open-source projects that are helping and contributing to the growth of serverless architecture in general. However, there are several issues with serverless architecture and its frameworks in practice. Even though the concept of serverless architecture is not new, the frameworks are still dealing with back-and-forth challenges. Many known issues with serverless architecture and its frameworks such as networking, performance, and security, so on and so forth. There are also new challenges that frameworks of serverless architecture may not be able to keep up with, such as the dynamic of the service, wide range of supported cloud providers, different configurations or architecture, complex infrastructure and integration, compatibility and many more that is not even yet recognized.

This master thesis will go through the fundamental types of issues and their causes of one specific and popular serverless framework, called Serverless Framework. The main research questions of this master thesis are to drill down on identifying the types of issues and mentioning their causes. To answer the questions, this master thesis will analyse the issue

types and the causes of them, and based on that in future research, we will propose solutions that can prevent users from facing the issues or help guide users and readers to overcome the known issues.

1.2. Master thesis structure

This master thesis will consist of four main parts: positioning the master thesis within the studies of serverless architecture and serverless computing and its purpose; the process of extracting issues and categorize them into themes and subthemes; a short introduction of serverless, its architecture overall and the context of Serverless Framework; and finally the analysis of extracted issues and their types and answers to the research questions that are stated in the methodology section.

Starting with stating the objective of the thesis that will explain the purpose and motivation behind the master thesis. This part is important to understand why this master thesis research is valuable and it will provide an overall scenery about the expected outcomes that the thesis will deliver in the findings.

Serverless Computing in earlier research will remind the foundation of serverless architecture and its developments in academia. It will provide an overall picture of serverless architecture and Serverless Framework. Stating the advantages and disadvantages of serverless architecture, its providers, and several other aspects of serverless computing.

Methodology section will provide an overview of research methodology, research questions, explain how the data is extracted and categorized, and how the analysis is conducted.

The Findings section is to give out the results of the analysis. What are the types of issues Serverless Framework encountered and detail categories and the explanation for each theme and subtheme of issues. A brief description of about the cause of issues will be mentioned in last part of the section and it will be open for further research in the future.

A conclusion is to sum up the research findings and validate the findings against the research questions.

Further study will be shortly mentioned and discussed. References and appendices are available in the end of the master thesis.

2. Objective of the thesis

2.1. Research objectives

The primary objective of this research is to methodically investigate and understand the various issues and challenges associated with serverless computing frameworks. This study aims to identify the root causes of these issues, assess their impact, and validate the potential complications that may arise when adopting a serverless framework. The goal is to provide a comprehensive analysis that aids in determining the most suitable serverless framework for specific use cases, ensuring informed decision-making for organizations and developers venturing into serverless computing.

Development of a comprehensive list of the common problems that normally come up as part of using serverless frameworks will be the first stage in the research. This involves conducting a full investigation into the current common bottlenecks, which include cold starts, security exploits currently occurring, scaling and lack of customization capabilities. To understand what lies underneath the working of serverless computing and the places such a technology can be practically applied in various categories of use cases, deep understanding of these problems becomes a must. These include cold start times pushing up response time to standard levels for initial requests as some functions are spun up, security loopholes leaving room for breaches, and restricted customization to complex business needs. Additionally, default autoscaling limits may come into effect and have poor performance when there is a sudden bloom in traffic. Thus, there is a need to future-proof serverless solutions via identification of such performance, security, and flexibility roadblocks through comprehensive analysis and understanding their respective technical reasons.

Identifying these issues, the master thesis will carry out an in-depth investigation regarding the factors that brought on such problems and the implications that have generated. This is, therefore, an essential step towards the realization of technical and structural facets that harbinger onto challenges of serverless computing. It is in this perspective that the study attempts at developing insights into intrinsic complexities and complications within serverless frameworks thereby deconstructing these foundational components. This will

enable deeper understanding of the strengths and limitations made by these structures. While serverless architecture has diversified advantages such as reduction in cost and increased scalability, its disruptive nature can prove to be challenging during the development process. This research is aimed to help the designers to implement serverless apps better through useful insights into the elements that cause these challenges and explaining them in detail. Besides, understanding the consequences of current problems will help to beautify serverless frameworks with some time so that their usage becomes more give habit.

This investigation must comprise a comprehensive look into potential intricacies this may occasion. As a part of this strategy, an empirical analysis is conducted regarding the impact which these challenges may have on performance, management, maintenance of serverless applications. The validation strives to provide a pragmatic perspective on the outcomes of implementing serverless architectures by assessing how such considerations impact performance of the framework, economic efficiency of the framework, as well as the overall dependability of the application. It is, therefore, necessary to clearly understand how different issues that have been linked to the new serverless designing model could impact on your project before getting into it. For instance, one should analyze carefully how cold starts, resource management, and security among other problems can undermine performance, cost-effectiveness, or reliability. It also examines how the implications of those challenges could change when complexities increase across applications and workloads. A valuable insight into the benefits and possible implications of serverless frameworks is offered by this critical assessment that makes more informed choices about their adoption.

Thus, the motivation of our research endeavour is to seek and identify the serverless system best fitted for a diversity of application contexts. The objective of the study is to provide associations and designers with a manual that will help them in choosing the appropriate system or combination of systems subject to the needs of applications. Several viewpoints will be greatly influenced during this decision-making process including the frequency with which events require to be dealt with, efforts for data processing, and that of the integration of internet things. A serverless system facilitating programmed scaling and simple onboarding of capacities could be optimal in such applications needing to respond rapidly and flexibly process enormous informational collections or live information streams from IoT gadgets. On the other hand, applications that may need more or better management for design as well as execution could demand a more customizable system. The purpose of our

research is to provide clear guidance so that an organization can choose the serverless solution which is best placed to meet its needs and to achieve its goals by evaluating fully each of the available systems against a wide range of application scenarios.

2.2. The expected outcome

The master thesis shall generate knowledge and guidelines that can eventually help the users in choosing, setting up or improving serverless frameworks. Deriving outcomes from comparative assessment and problem confirmation would thereby provide the understanding through the study that can be implemented as well as ideal practices for exploration of the realm of serverless computing. The proposals are aimed at having increased the help that would be accorded to the practitioners in making enlightened decisions that would increase their effectiveness and efficiency in serverless application. However, there exist some barriers that serve while preventing one from perfecting with the use of serverless architectures. Further research still needs to be done in developing best practices only for, among others, monitoring, debugging and performance tuning. In general promises which from the low operational costs, better flexibility and ease of management using serverless computing are quite compelling. But fully realizing the benefits will require ongoing work to address limitations and standardize new approaches.

By achieving these objectives, the research expects to significantly contribute to the field of serverless computing by offering a detailed understanding of the challenges and considerations in selecting and utilizing serverless frameworks. This study aims to equip practitioners with knowledge and tools to effectively navigate the serverless landscape, ultimately enhancing the efficiency, security, and scalability of serverless applications.

3. Serverless Computing in earlier research and introduction of Serverless Framework

Serverless Computing: A Paradigm Shift in Cloud Computing

Serverless computing has emerged as a major paradigm shift in cloud computing providing scalable, cost-effective solutions abstracting the management and maintaining of servers. This literature review then compiles the new research and discussions on serverless computing particularly focusing on IoT, task scheduling, performance, security applications, challenges, and frameworks of the same.

3.1. Core concepts of serverless computing

Function-as-a-Service, also known as FaaS, is the fundamental component of serverless computing. The execution of your code if certain events take place is a simple and uncomplicated process, and you do not need to be concerned about the servers. The small pieces of code that you write are referred to as functions, and they only function when they are required. At that point, they cease their activity in a stealthy manner. Examples of this include well-known services such as Amazon Web Services Lambda, Microsoft Azure Functions, and Google Cloud Functions. Because they are responsible for the servers, you are free to concentrate solely on writing your code. Because of this, the entire process of developing applications is simplified significantly (Baldini et al., 2017).

Another type of cloud service that is utilised by serverless computing is known as Backend-as-a-Service (BaaS). For your application, BaaS is responsible for all the tasks that occur in the background, such as managing databases, determining who the users are, and storing files. Based on the findings of Leitner et al. (2019), it eliminates the challenging aspects of managing servers, allowing developers to devote more time to the aspects of their applications that are important.

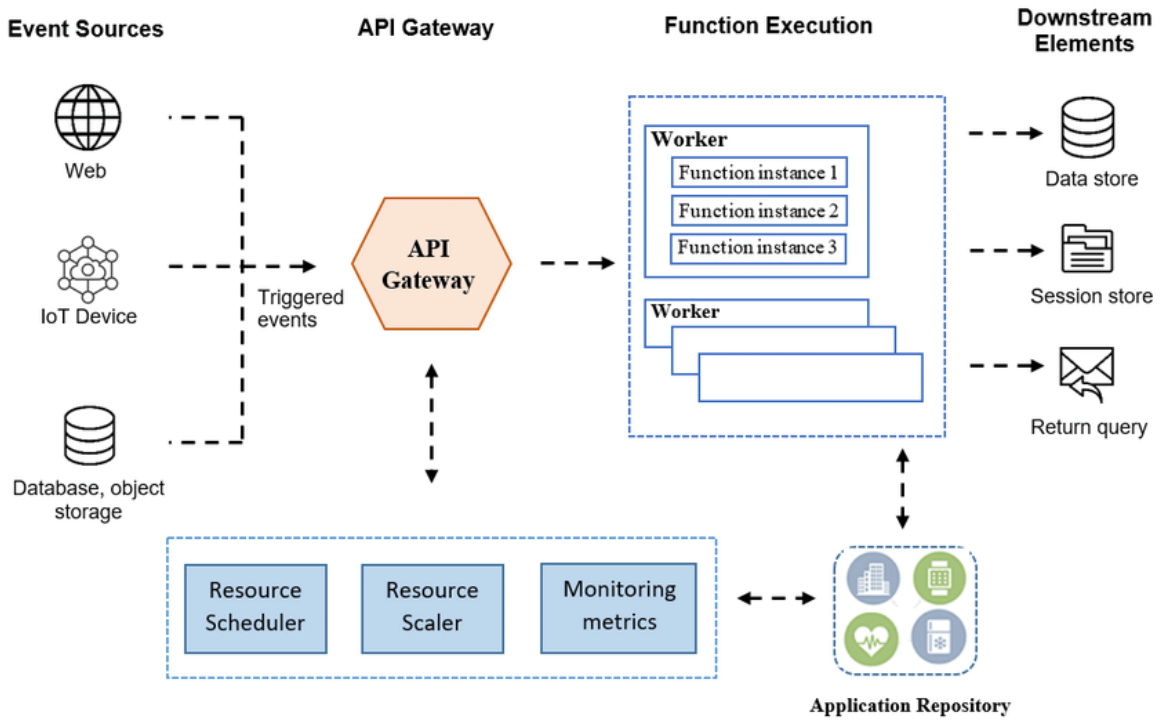


Figure 1 Serverless Architecture overview (Mampage et al., 2022)

Key aspects of serverless computing include:

Ease of Deployment and Operations: Serverless computing makes the deployment of code into production easier that allows leaving developers more time to write code but not managing and operating servers. It is because serverless platforms reduce the demand for server management and capacity planning (Ali, 2021).

Cost-Effectiveness: In the serverless service model, payment is done based on a "pay-as-you-go" model where developers pay for their code by counting the number of times their code is executed and how much of the resource it consumes, rather than paying for underused provisioned computing resources (Jiang et al., 2020).

Event-driven Architecture: Serverless computing inherently depends on events for its execution. Functions are invoked following the occurrence of an event, and co-resident resources automatically scale out and deallocate (Kelly et al., 2020).

Scalability and Flexibility: Serverless architectures will automatically scale according to the requirements presented by an application. Meaning that serverless architectures are elastic, which is good for applications with workloads that cannot be entirely defined in advance or those that are varying (Djemame, Parker, & Datsev, 2020).

Performance Considerations: While serverless computing is rewarding, it still carries as much its own reverse side like in matters of latency, especially when "cold starts" come into play and a certain function takes longer to start if it was not used so long ago (Maissen et al., 2020).

Emerging Applications: Serverless computing is being incorporated within emerging arenas more and more such as machine learning and AI applications since its ability to cut down the complexity of the machine learning system contributed to providing management easily (Bac, Tran, & Kim, 2022).

Portability and migration issues: In special, the serverless applications have identified various problems to be associated with their portability into various cloud providers given that there exists differing of scenes, models, and APIs (Yussupov et al., 2020).

Resource Management: Efficient control of computing resources in serverless environments involve provisions for, allocations, schedules, monitors, and scales critical considerations (Mampage et al., 2021).

Security Considerations: Despite some advantages that serverless computing provides such as increased cost savings, and the potential to scale for organizations, security is still a major challenge to be tackled especially regarding multi-cloud scenarios (Poorvadevi & Ramamoorthy, 2018).

3.1.1. The advantages

One of the most apparent perks of serverless computing is that it's noticeably more cost-effective than its conventional counterpart. The difference between serverless computing and the traditional server-based model is substantial. Essentially, serverless computing goes by a 'pay-as-you-go' pricing strategy. This means charges are directly tied to how much it's used and how long the functions execute. Baldini and his colleagues in 2017 pointed out that this system gets rid of any financial strain caused by servers idling, which consequently enhances a business's cost efficiency.

Serverless architectures make life easier for developers by getting rid of the need to manage servers. It takes away some serious complications. Instead, the companies that provide the cloud service deal with server maintenance, scaling, and managing the infrastructure. This

means developers can really get their teeth into creating and innovating their products. A 2021 study by Eisenmann and others found that going serverless really lowers the amount of operational work and makes the deployment process smoother.

One of the natural advantages of serverless computing is its impressive scalability. What's really great is how it automatically adjusts, scaling up or down based on demand. This means there's no need for someone to manually intercede. This is especially helpful for apps that have fluctuating workloads, or for budding startups that need the capability to grow on demand. This flexibility is particularly beneficial for applications. (Yussupov et al., 2019)

By simplifying operations and doing away with the tasks tied to server management, we've sped up the development cycle. This means that apps can hit the market quicker, giving businesses a leg up in ever-changing markets (Leitner et al., 2019).

3.1.2. The disadvantages

Securing serverless computing is challenging. Given that serverless apps are spread out across many locations, keeping all data safe poses a real challenge. Thorough knowledge of the entire system and possible issues that could crop up is a must-have to keep everything secure (Li et al., 2023.)

Moreover, there is a known issue in serverless computing popularly known as 'cold starts'. It is essentially the delay you come across when you get down to using a function that has been inactive for a while. As highlighted by Lynn and his colleagues in 2017, this delay can pose significant troubles for applications that require super-fast, real-time functionality.

Using serverless computing can create a few hurdles. A significant one is the risk of becoming too reliant on a single cloud provider's tool set and services. This situation is often called 'vendor lock-in.' Baldini and his colleagues mentioned in their 2017 study that this dependence could make it hard to switch to another provider or to integrate different services into your app.

To sum it up, while serverless computing certainly makes some things easier, it doesn't offer complete control to developers over how their applications operate. As highlighted by Yussupov et al. in 2019, this may pose challenges in making precise adjustments or in achieving the best possible performance under given circumstances.

3.2. Cloud platforms that enable serverless architectures

Serverless cloud computing is a paradigm in which the cloud provider manages the server infrastructure. This paradigm has seen rapid adoption since it is scalable, cost-effective, and easy to use. It is because of this that several significant players have emerged in the market, each of which provides distinctive characteristics and services. The introduction of serverless computing platforms by major cloud service providers like AWS, Azure, and Google Cloud Platform reflects a shift towards more cost-efficient, manageable, and rapidly scalable cloud services. AWS Lambda has been identified as a leading platform for serverless computing research (Lynn, Rosati, Lejeune, & Emeakaroha, 2017).

3.2.1. Amazon Web Services (AWS) Lambda

Amazon Web Services (AWS) and its AWS Lambda service are often seen as the first companies to use serverless computing. With its automatic scaling, high availability, and pay-per-use pricing model, Amazon Web Services Lambda quickly established itself as a standard for serverless computing within a brief period after its launch in 2014. It is a popular choice for a wide variety of applications because Lambda enables developers to execute code in response to events without the need to provision or manage servers.

3.2.2. Microsoft Azure Functions

Azure Functions is an example of a serverless computing platform that exemplifies the trend towards cloud-oriented software development, allowing developers to focus on product development rather than server maintenance and administration (Sawhney, 2019). Microsoft Azure Functions is a significant competitor in the serverless market. It offers an execution environment that is driven by events and integrates without any complications with other Azure services. The Azure Functions platform was introduced in 2016, and it can support a wide range of programming languages. Additionally, it brings Microsoft's enterprise-grade security and compliance capabilities to the realm of serverless computing.

3.2.3. Google Cloud Functions

The Google Cloud Functions is Google's response to the serverless computing trend. It gives developers the ability to execute backend code in response to HTTP requests or event triggers that are generated by Google Cloud services. It offers a fully managed environment that is scalable and has a pay-per-use pricing model. One of the most notable features of Google Cloud Functions is its extensive integration with other Google Cloud services, as well as its powerful data analytics capabilities.

3.2.4. Others

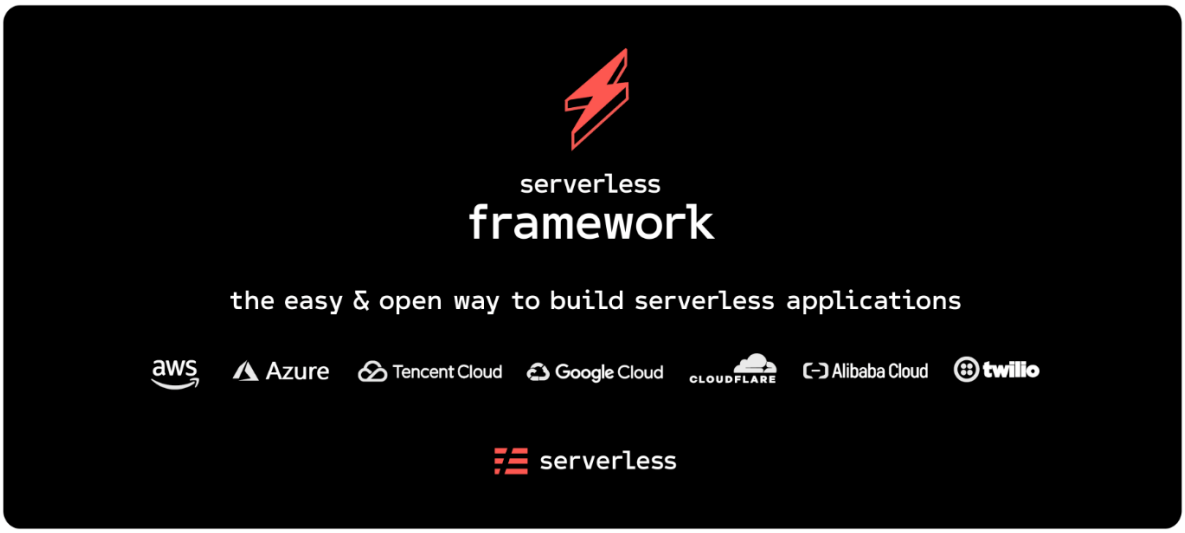
Using Apache OpenWhisk as its foundation, IBM Cloud Functions is a flexible and multi-language FaaS (Function-as-a-Service) platform. In addition to providing developers with the ability to execute code in response to a wide variety of events, it provides an open-source, event-driven model that also provides the advantage of IBM's comprehensive collection of artificial intelligence and data tools.

The Alibaba Cloud Function Compute service offers a serverless computing solution that is fully managed and driven by different events. It is gaining popularity, particularly in Asia, because it provides a platform that is both flexible and scalable, while also supporting several different programming languages and integrating with the extensive suite of services that Alibaba Cloud provides.

For the scope of this thesis, only AWS, Azure and Google Cloud will be analyzed and mentioned in serverless contexts.

3.3. Introduction of Serverless Framework

Serverless Framework (or serverless in short) is an open-source framework focuses on establishing, managing, maintaining, and operating serverless architecture. The very first commit of the whole project dated back to April 21st, 2015, by Austen Collins, founder, and CEO of Serverless, Inc. and he is also the creator of the Serverless Framework. Serverless Framework stands out from the current frameworks for serverless in the market because of its maturity and high maintenance.



Picture 1 Serverless Framework

Serverless Framework in short, is a framework that helps software engineers and cloud professionals set up new services (such as AWS Lambda, GCP Cloud Run, Azure Function App, AWS DynamoDB, etc.) and configure them in the same tool and in a comprehensive procedure. Serverless Framework supports several languages, i.e. Node.js, Python, Java, Go, C#, Ruby, Swift, Kotlin, PHP, Scala, and F# which is why Serverless Framework gains its popularity quickly within computing cloud and software engineering communities. Currently when writing this paper, Serverless Framework has over 45k stars, 5.8k forks and near 1k watching in its GitHub official repository. Şahin et al. (2019) discussed how the number of stars a repository receives is often considered a measure of its popularity, and how various factors like the number of forks and contributors influence this popularity (Şahin et al., 2019). With over 45k stars and 5.8k forks, Serverless Framework is among the popular open-source repositories in GitHub.

Over 40k repositories that depend on Serverless Framework according to GitHub repository Insight. All the numbers have painted a popular and useful open-source project that can bring serverless architecture closer to software engineers and cloud practitioners, without lengthy establishment process, complex configuration and different techniques required for each cloud provider, Serverless Framework removes those obstacles and creates a simple and yet effective process and technique to achieve complex architectures.

Throughout the years, serverless has gained huge attentions from software engineers and cloud professionals. There are over 15 thousand commits from more than 1 thousand contributors from everywhere in the world.

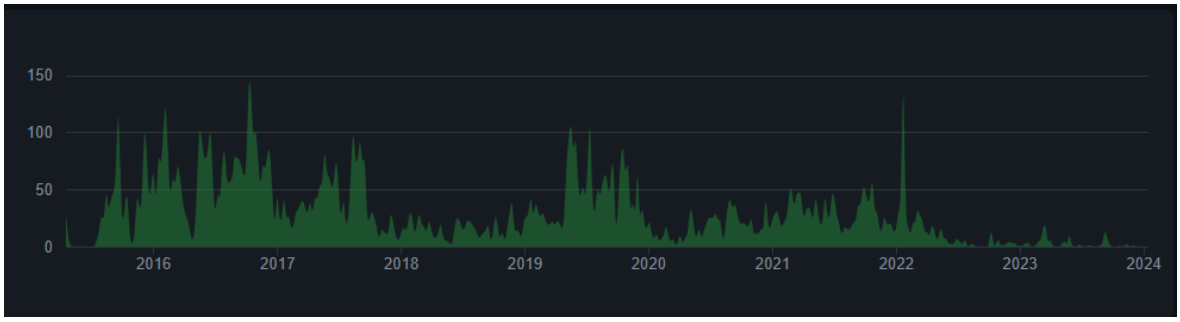


Figure 2 Commits to main of Serverless Framework GitHub repository.

Over 300 releases in the span of over 9 years (counting until the time writing this paper), that is equivalent to over 40 releases per year and on average close to 1 release every week. That is the evidence that Serverless Framework has been being updated and renewed on weekly basis with a strong community and wide range of supporting features.

All the numbers have shown the impact of Serverless Framework has been making in software and cloud development. Since Serverless Framework has supported many features and for many cloud providers, it also has its own issues. Over the years, a cumulated number of issues has been over 6 thousand in total. Over 5 thousand issues have been closed and the other 1 thousand issues are still open. Kavaler et al. (2017) studied the effect of issue discussion complexity on issue resolution times, highlighting the importance of effective communication in issue management. Based on the study, it suggests that repositories that are like Serverless Framework is a complex topic which is suggested in the discussions of the issues. Over 1 thousand issues are still open due to miscommunication, complex explanations, complex architecture, and many other reasons. These factors indicate Serverless Framework, and serverless architecture frameworks related topics are difficult to comprehend and resolve.

With all the mentioned factors, Serverless Framework can be the great case study for this paper. It cannot stand for serverless architecture and other serverless open-source frameworks in general, however it can provide an overview of what types of issues and common causes of issues that serverless architecture and open-source serverless frameworks may encounter. It is not trivial knowledge and analysis for all other serverless frameworks, but it can hint some relevant trends that all other serverless frameworks have, so that the in-depth analysis will hold true in majority of aspects.

3.4. Utilization of serverless computing

Serverless Computing and IoT: Cassel et al. (2022) have done detailed research to find out the role of serverless computing on Internet of Things (IoT). These provided a systematic literature review that shed light on the challenges and benefits gained by the usage of serverless architectures in IoT landscapes, primarily elaborating how they reduce operations due to reducing complexity and further enhancing scalability (Cassel et al., 2022).

Serverless Computing and Cloud Task Scheduling: Scheduling tasks on cloud platforms as a typical approach has been elaborated by Alqaryouti and Siyam (2018) with serverless computing. The authors, in their research, touch upon the architectural benefits of serverless computing ranging from efficiency and flexibility applied to scheduling tasks based on clouds (Alqaryouti & Siyam, 2018).

Use Cases for Serverless Computing: The analysis of Eismann et al. (2020) added significantly to the aspect associated with the serverless use case review and characteristics, giving a realistic look at how serverless characteristic can be applied and what effects in future over numerous industries and other surroundings (Eismann et al., 2020).

3.5. Different angles of serverless architecture

Design and Performance of Serverless Computing: McGrath et al. (2017) had carried out a study on the design, implementation, and performance of serverless computing. This is an important study to understand the design intricacies architectural aspects of serverless computing and implications of the same in terms of different performance parameters (McGrath & Brenner, 2017).

Security in Serverless Computing: Golec et al. (2021) introduced iFaaSBus, a security framework that specializes serverless computing environments. This typology has been able to help in resolving the challenges associated with the escalations of security as well as privacy architectural aspects of serverless by combining IoT and machine learning (Golec et al., 2021).

Performance Evaluation in Function-as-a-Service: Scheuner and Leitner (2020) conducted a comprehensive literature review on FaaS performance evaluation. These performance

benchmarks cover the judgment for serverless computing platforms (Scheuner & Leitner, 2020).

Tradeoffs and Challenges in Serverless Deployments: The outline of the trade-offs and challenges of serverless deployments could adequately depict a proper understanding of the modern-day limitations and problems that need to be resolved for improving serverless computing, comprising cold start delays, data communication overhead as well as hardware heterogeneity problems (Garcia-Lopez, 2021).

Temporal Performance Modelling: In relation to serverless computing platforms, Mahmoudi and Khazaei (2020) conducted an analysis of the temporal performance to give insights in respect to the performance dynamics, and architecture's scalability in time of serverless (Mahmoudi & Khazaei, 2020).

Quality-of-Service in Serverless Computing: Tariq et al. (2020) described the issues as well as solutions with respect to quality-of-service provisioning within serverless computing environments. Their work highlights maintaining and achieving enough service quality despite the dynamic nature of serverless computing (Tariq et al., 2020).

4. Methodology

The investigation of serverless frameworks and the problems that are associated with them is a comprehensive process that has been created with great care to unravel the complexities and nuances that are inherent in these systems. The objectives of this research methodology are accomplished through the utilisation of a combination of technological tools and analytical strategies. This methodology is comprised of two distinct phases that are interrelated to one another. In the scope of this thesis, Serverless Framework will be selected for further analysis, other frameworks will be in a further study in the future.

4.1. Research questions

RQ1: What issues do occur in the development of system utilized open source Serverless Framework?

Rationale: RQ1 is pertinent due to the growing practice of serverless architectures experienced across the software industry. This question will delve into technical, operational, and collaborative aspects associated with a popular open source serverless framework called Serverless Framework. Rationale: Understanding of these challenges essential in enhancing scalability, efficiency, and cost effectiveness in cloud computing through a smaller scale of one of the most popular serverless open sources called Serverless Framework. Additionally, it addresses complexities in debugging, integration and performance fine tuning are unique to the serverless technology. The findings of this master thesis are critical to improvement in development practices, better community driven collaboration and the future ready state of serverless technologies. This contribution not only helps in the technical advancement of serverless frameworks but also assists developers, organizations, and the whole tech community in risk mitigation, security adherence, all while staying aligned with industry standards.

RQ2: What are the causes of issues that occur in open source Serverless Framework?

Rationale: The purpose of RQ2 is an early investigation of the causes of issues in Serverless Framework and it is vital given the rapidly expanding role of serverless technology in modern software development. This research question targets the heart of challenges faced in the deployment and maintenance of this selected framework – Serverless Framework.

Limitation: For the scope of this master thesis, an overview of what are the causes of issues will be presented and not going to detail analysis due to the lack of resource and time constraints. In practice, to be able to identify the causes and base on the causes suggested proposing solutions requires a sophisticated process of selecting seasoned professionals with different backgrounds, experiences in various technologies and in several regions which will paint a more reliable overview of opinions and propose universal solutions.

4.2. Research phases

The analysis consists of two phases: Identification and Initial Extraction and Filtering and Categorization. The reason behind the two phases is they are a sequent of work that the later one depends on the previous one. Both phases are required to involve several steps and combined automated and manual work. Each phase is also required to carefully review and adjust the automated steps to reach the required accuracy and correctness.

An empirical analysis is conducted to analyze the collected and categorized data from the two phases. The methodology of empirical analysis is a cornerstone of any scientific research and underpins the importance of direct or indirect observation and experience in data collection. At the heart of this approach lies a precisely articulated and proper research question which, so to say, stages the whole investigation. This very research question must be formulated in such a manner that it could further become, in a certain way, testable by empirical means.

Interpretation of the results in here is quite delicate since it involves comparing findings to the first hypothesis and what is already in the literature. From that comparison, it is possible to identify implications from the study, considering its boundaries of reach, and putting at stake findings with bigger dynamics.

Finally, the results that are established should be reported in a clear structured way with clear information how methodology was applied, and data analysis and analysis employed draw results. This will make the other scientists feasible to evaluate, replicate or build upon it to add scientific knowledge.

4.2.1. Phase One: Identification and Initial Extraction

Phase One is the first step in the journey, and it is equally important because it establishes the groundwork for the entire study. One of the primary objectives of this endeavour is to search through the vast digital landscape to locate serverless frameworks that are of significant relevance. This selection process is not arbitrary; rather, it involves a careful consideration of a variety of factors that indicate the significance and impact of these frameworks in the field.

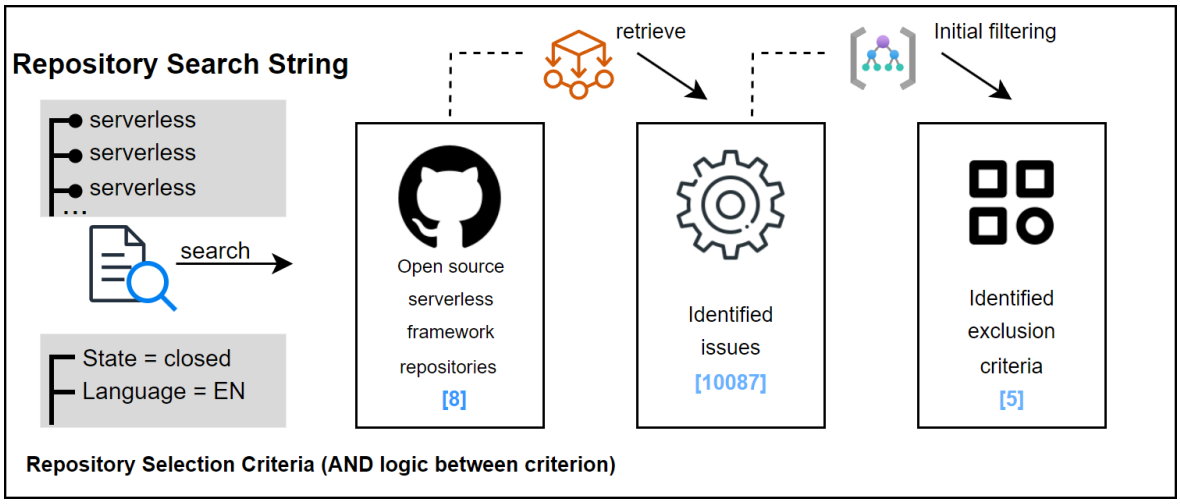


Figure 3 An overview of Phase One

After these frameworks have been identified, the attention will shift to the process of obtaining relevant data concerning the problems that are associated with both. Performing this extraction is not a simple task; it is carried out with the assistance of a complex Python script that was developed specifically for the purpose of performing this extraction. Repositories are combed through with great care by the script, which then extracts important information regarding each respective issue.

Before the script can function as expected, a personal GitHub token needs registering with the *READ* permission and storing in the .env file in the root folder of the project. The script gets the GitHub token from an .env file that is in the root folder of the project and uses the token to fetch all the issues from a defined repository. The script loops through the data it gets and extract the repository's issues to an excel file, the information that was gathered includes the following: the title of the issue, which gives an overview of the problem; the link to the issue, which acts as a portal to more in-depth discussions; the dates that mark the beginning and ending of the issue, which provide insights into the issue's lifespan; and the

number of participants involved, which reflects the level of engagement and interest in the issue.

Following this, the data is compiled into a comprehensive file, which functions as a rich repository of information and a foundational component for the subsequent phase of the study.

4.2.2. Phase Two: Filtering and Categorization

Exploring serverless frameworks and assessing their associated difficulties requires a thorough and comprehensive effort meant to untangle the nuanced complexities innate to these systems. The standards for this screening process are selected with great care to make certain only the most pertinent and meaningful problems remain for additional investigation. This is done with the goal of confirming solely the most critical issues are kept while removing less significant matters. The examination seeks to bring clarity and understanding to truly comprehend the intricacies involved.

Any issues without comprehensive descriptions are excluded from the criteria, mainly because they do not contribute much to our analysis. Similarly, the objective of this thesis is setting aside broad questions, thoughts, feedback, and ideas to focus on more specific issues. While feature requests like enhancements or proposals are considered at this stage, they're not the top priority. We also screen out announcements, especially relating to new updates, and any repeated issues. Lastly, if a problem has only been raised by one person, we take it as less influential to our study and hence ignore it.

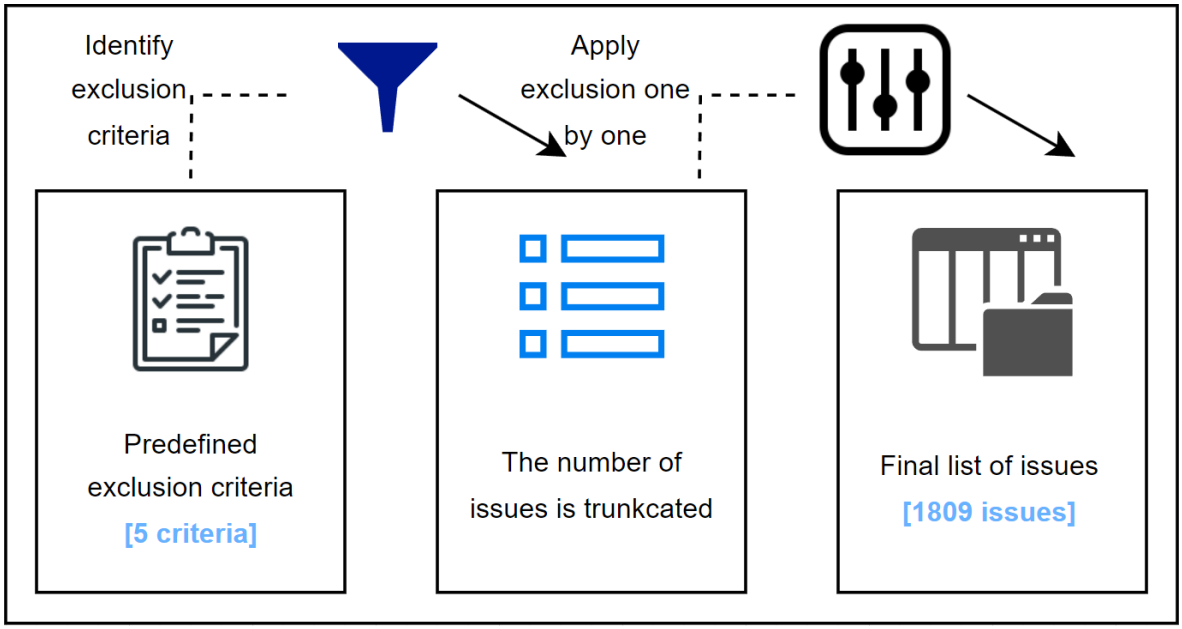


Figure 4 An overview of excluding (filtering out) out of scope issues.

We make use of an additional Python script to streamline the complicated and time-consuming filtering process. Besides identifying problem descriptions, this script delves deeper into a thorough search process. It primarily employs two strategies - a keyword search looking for pertinent terms and phrases in the text, and a context search. The context search is a sophisticated technique made possible by integrating a substantial language model. Due to its profound understanding of language and context, this model is excellent at identifying subtle nuances in problems. This capability lets it provide a categorization that's richer in detail and remarkably precise.

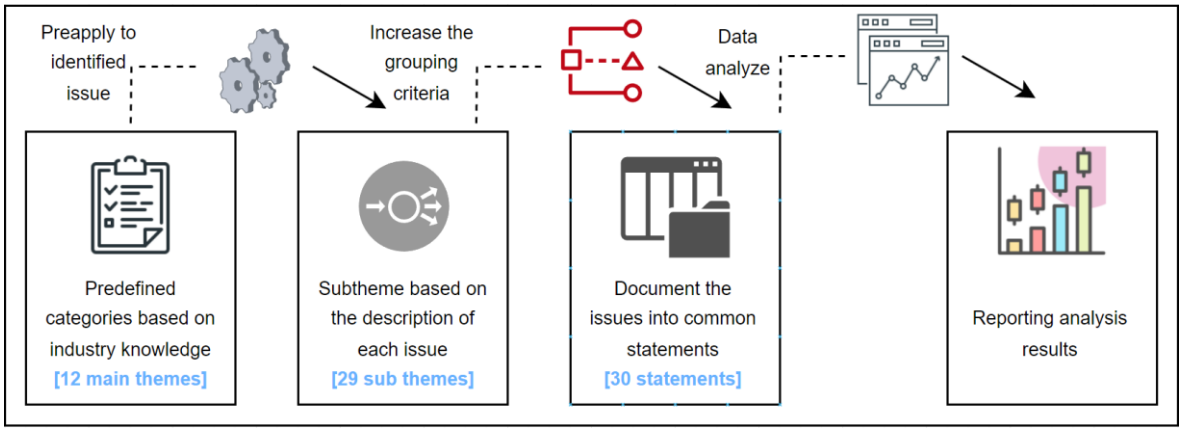


Figure 5 An overview of the categorization process

The culmination of this phase is the categorization of issues into a variety of different categories that have been predefined. It is important to note that these categories are not merely containers for data; rather, they are thematic clusters that reflect the underlying patterns, challenges, and trends in the landscape of serverless frameworks. The study offers a structured and insightful lens through which to examine and comprehend the complexities of serverless technologies. This is accomplished by classifying the issues into the categories.

After the automation search and categorization, a manual correctness check for all the issues to make sure the automated categorization works as expected. Manual adjustments and modifications are conducted if there are errors or unjustified categories. The process is considered that there are overlapped categories for an issue, which means one issue can be categorized into two themes and based on its nature cause or weighted theme, the issue is categorized to one main theme and subtheme only.

Similar to the categorization of issue types filtering, the categorization of issue causes filtering follows the same principles and process. Ten causes are identified from earlier studies and literature that will be mentioned later in the RQ2 section. A similar python script is used to conduct keyword matching in the discussion to identify the cause topic of each issue to find out what are the root causes of every issue. After that, issues are mapped with those causes and causes topics. The result is collected and show case in the RQ2 question with the minimum effort of sanitization due to the time constraints which is mentioned in the RQ2 question limitation section.

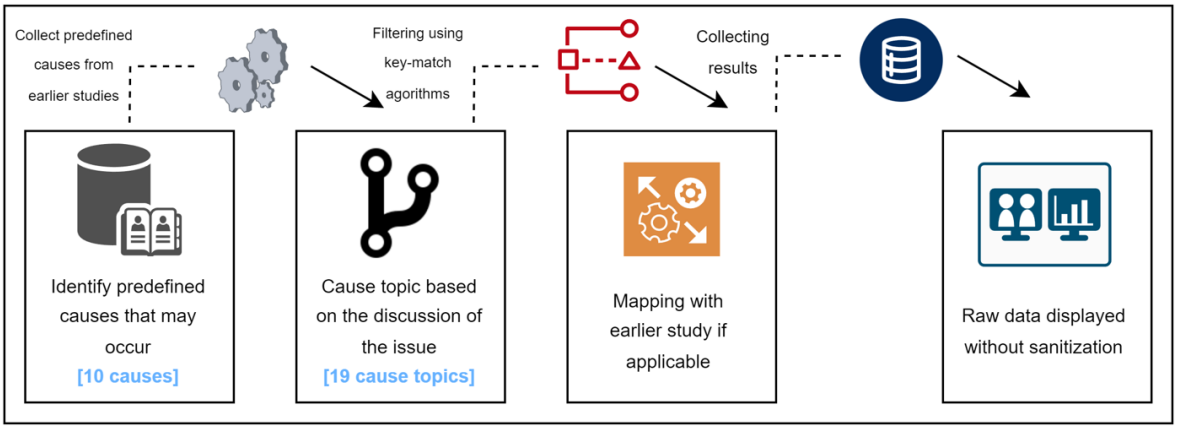


Figure 6 An overview of the cause's categorization process

A further thorough sanitization will be needed to make sure the studies of the causes and the identified causes and the issues are associated and mapped scientifically correct. The process

is time consuming and required a standardized process which is not included in the scope of the master thesis.

5. Findings

5.1. Types of Issues (RQ1): In-Depth Analysis

With the first research question, we will break down all the issues into several main themes where we can identify key areas of the issues Serverless Framework is encountered. A thorough analysis of prevailing issues is essential for further analysis of finding the cause and effect of each issue. Based on this analysis, we can provide a set of proposals which make the utilization of Serverless Framework, and further serverless architecture sustainability and cost effectiveness.

The filtered issues are categorized into 12 main themes which are defined by relevancy of the description and the discussion of each issue. Based on each main themes, each issue is further categorized into a more detailed sub-theme which states more accurately which type the issues are belonging to. The figure 6 is an overview of total issues and their themes.

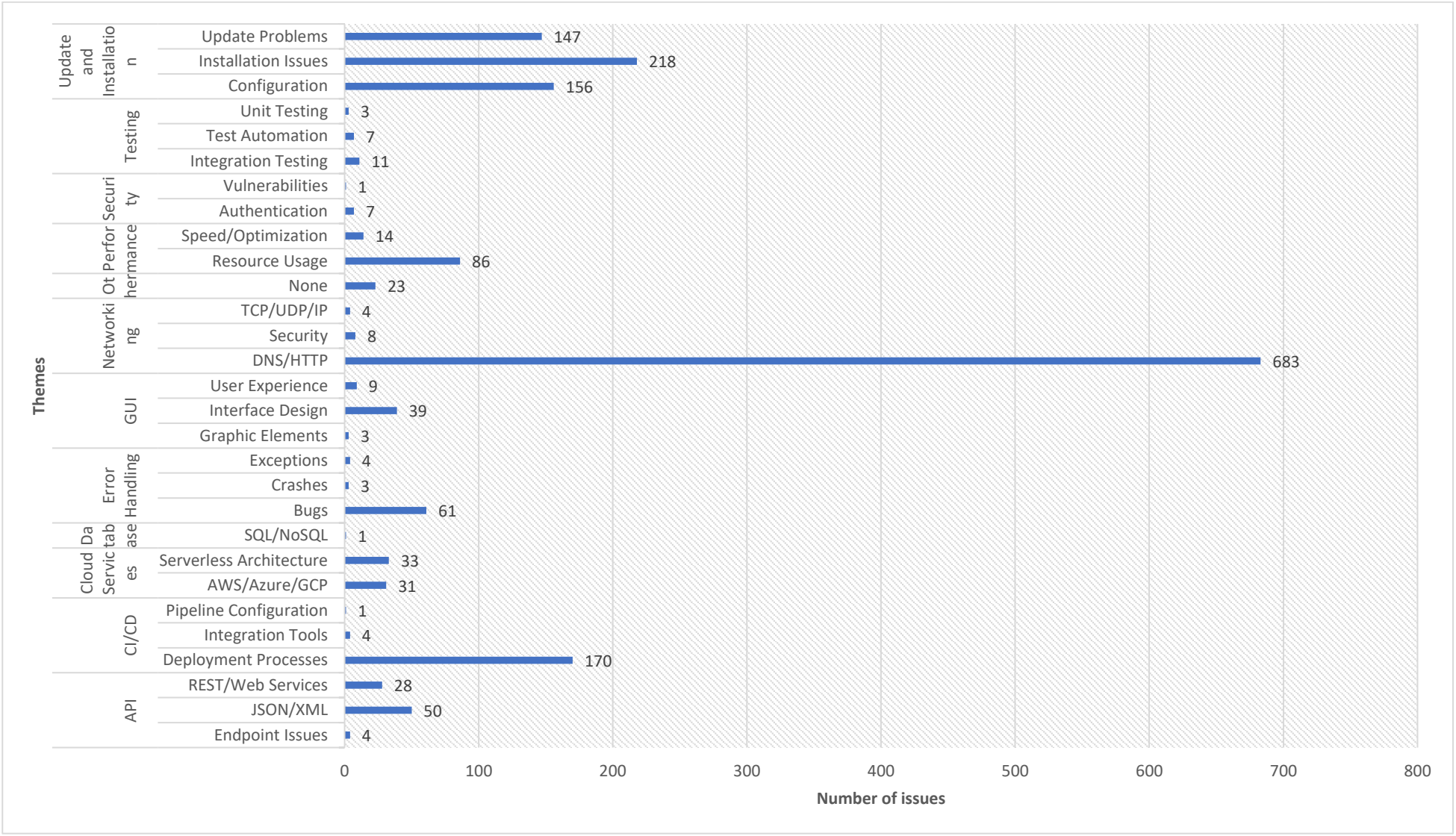


Figure 7 An overview of issues and their main themes and subthemes

5.1.1. Networking issues

With 695 of 1809 total of issues, networking issues are the most common issue type within the Serverless Framework repository issues. This finding is important to understand the core nature of the serverless architecture overall with a smaller scale repository like Serverless Framework. It does not mean that networking issues are the most common issue in serverless architecture and serverless frameworks in general, however it suggests that networking related issues are among the most common and one of the most known issues in the serverless architecture overall.

The nature of Serverless architecture is the applications are broken down into smaller individual functions or services which will only serve a specific purpose. This model requires an event-based trigger system that requires the involvement of communicating through network. The traditional architecture has minimal communication between backend services as they are usually a giant monolithic stateful application unlike serverless architecture has several stateless functions working in harmony through a sophisticated message event queue. There are several networking types of issues in serverless architecture context such as: network latency, cold start, network control limitation, security, or scaling issues and in Serverless framework is similar. The three main subthemes from the issue list are DNS/HTTP, Security and TCP/UDP/IP which is a significant factor to the total number of issues.

The high amount number of issues does not indicate that networking is the biggest issue in serverless architecture, it only shows that networking issues are common due to the nature of serverless architecture which requires communication and event management between functions or services within the architecture.

Networking issue type is taking the most significant share of issue types in Serverless Framework in its GitHub closed issues. This also indicates the common and predictable trend in serverless architecture due to its nature architecture.

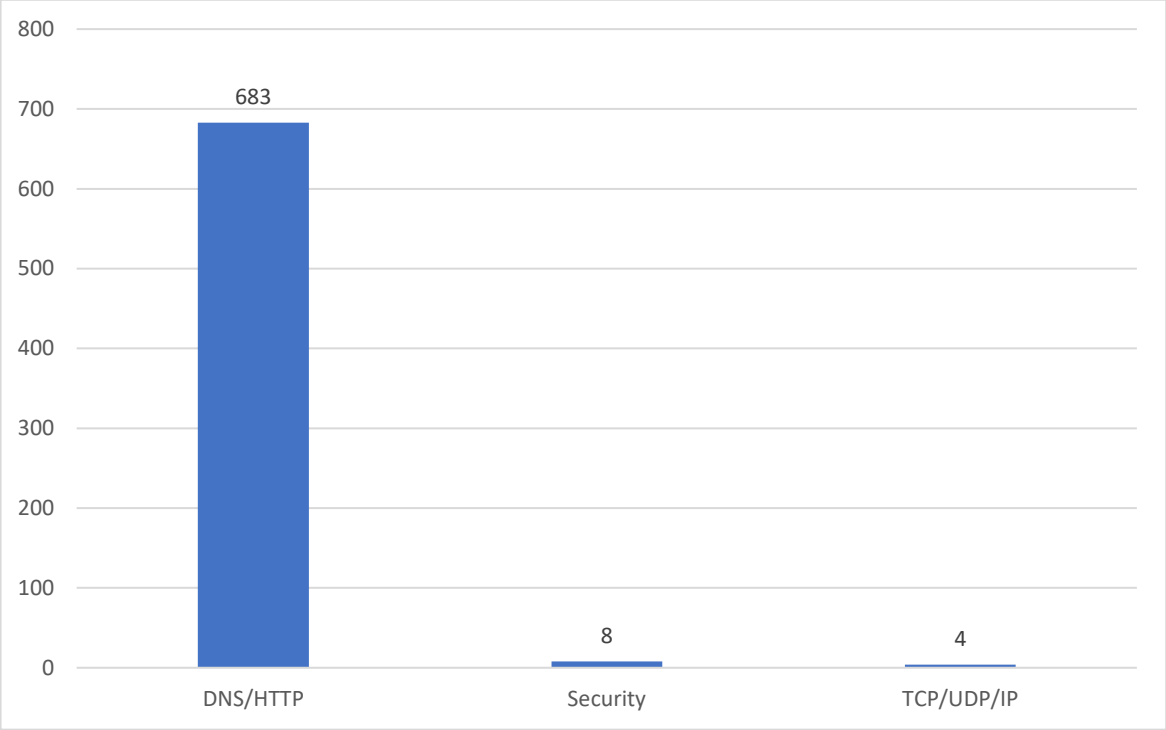


Figure 8 Overview of Networking issues and its subthemes.

- **DNS/HTTP** (683/1809, 37.76% of total issues): When working with serverless architectures, it is essential to handle HTTP requests and responses in an efficient manner. Often, serverless functions depend on HTTP to interact with external services or APIs. Every HTTP request adds some delay, which becomes increasingly obvious during instances like cold starts of functions or when they are short-lived and need constant initialization. This delay can notably affect the serverless application's overall performance. For this reason, optimizing HTTP requests for maximum efficiency is of utmost importance. One more hurdle to consider in serverless environments is managing DNS resolutions. Your serverless functions will usually need to conduct DNS queries to figure out domain names for outgoing HTTP requests. If we're not careful with how these DNS lookups are cached, it can seriously slow down the process and contribute to overall latency. Serverless functions, by their fleeting nature, don't typically have the luxury of long-term DNS caching. This results in repetitive resolutions that can cause even more delays. Of course, we can alleviate this problem by efficiently managing DNS queries and smartly applying caching strategies. Just keep in mind, it requires some well-thought-out planning and proper execution. On top of everything else, serverless architectures do struggle with managing connections. Keeping up constant connections—like those supported by

HTTP keep-alive—becomes less productive within a serverless framework. Serverless functions have a short lifespan, making it difficult to uphold long-lasting connections. This leads to extra work in creating new connections every time a function is used. This constant setting up of connections makes the system slower and lessens the overall effectiveness of serverless applications. The security of DNS operations is also a crucial aspect, especially with the use of dynamic updates in DNS records. Implementing secure mechanisms such as Transaction Signature (TSIG) and Domain Name System Security Extensions (DNSSEC) is necessary to address security vulnerabilities, especially in IPv6 networks using Stateless Address Auto Configuration (SLAAC) (Rafiee & Meinel, 2013).

- **Security** (8/1809, 0.44%): Even though they make up a smaller portion of overall problems, network security issues are extremely important. They can involve serious threats such as unauthorized entrance, breaches of data, and weaknesses in network protocols.
- **TCP/UDP/IP** (4/1809, 0.22%): Problems in this area could be linked to managing various network protocols and guaranteeing that data is transmitted effectively and dependably over these protocols.

5.1.2. Update and Installation Issues

Compatibility and harmonization are usually key considerations when it comes to update and installation in serverless architecture context. Once a part of the system gets updated or has a new version, it also requires the other to handle the changes. That comes with a cost that the architecture needs to be built in a way that it can handle partially update and sometimes, backward compatibility is also required. Configuration, in the other hand, is considered the joint between the services, therefore, issues related to configuration are often common.

Having 521 of 1809 of total issues, Update and Installation contributes a significant number of issues to the total issues. This type of issue often comes in three kinds: Configuration, Installation issues and Update issues. As mentioned before, serverless architecture has several individual functions work together in harmony, meaning that if one of the functions gets in some challenges, the whole orchestration will be compromised and fail the

application. Correct installation is the key to make sure that the chain of messages to trigger functions are handled, configuration in the other hand, will dictate how each function is setup to work with others and update is to keep all the functions or part of them up-to-date.

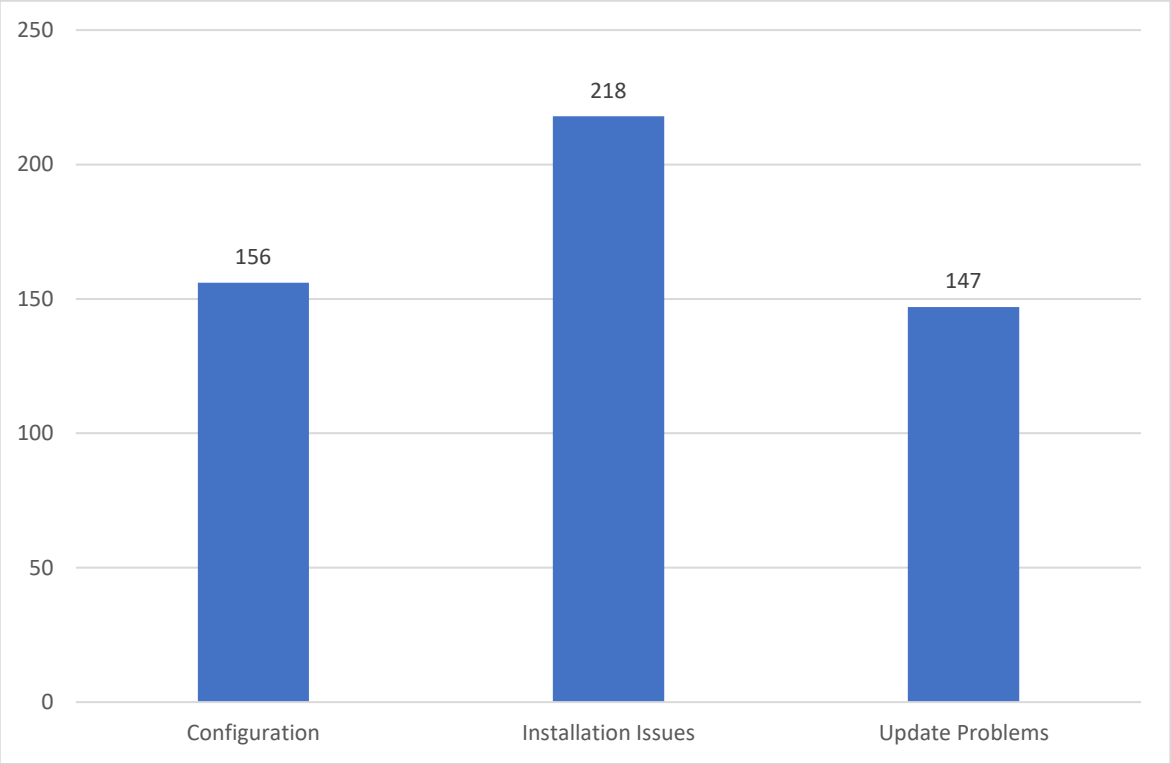


Figure 9 Overview of Update and Installation issues

- **Installation issues** (218/1809, 12.05%): It includes setting the individual functions and services in cloud providers. Also includes setting up of messaging or event management as a broker of functions and services to listen and get triggered when it is needed. This subtheme is dominating in case of number of issues under Installation and Update issue's theme because setting up and installer individual functions and its messaging broker are a procedure aimed at invoice processing and getting this very difficult right from the beginning.

- **Configuration** (156/1809, 8.62%): Having important role to connect functions together with individual and partially independent functions in configurations because it also contains configurations for functions in cloud providers. There are tiers and different pricing in cloud providers which allow the flexibility of options for various use cases, correctly configuring the functions as well as its resources in cloud providers will help the workflow running smoothly and boost up the benefits

of the functions as well as optimizing the cost. Authentication and integration also are the pitch for Serverless Framework issues in Configuration part.

- **Update Issues** (8.13%, 147/1809): Changing serverless apps isn't only about fresh code. It often gets tricky due to a system part change. This may involve the change of function configs, keeping compatibility of cloud service, and managements of dependency. The sizeable percentage of update issues underlines difficulties in looking after serverless apps, especially in making sure that updates don't mess with the current features or cause shutdown.

5.1.3. Performance Issues

Performance in serverless applications or serverless architecture means 2 things which are equivalent to two sub themes: Resource Usage and Speed/Optimization. In serverless architecture, there are not many cases that applications are paid up-front, which means there is not on-going resource but instead, an on-demand resource is preferred because of its cost effectiveness and scalability. With that in mind, selecting the right resource and optimize it to meet the demand and at the same time save cost is the most crucial decision to make.

100 of 1809 total issues indicate a demanding aspect of performance that serverless framework and serverless architecture in general are dealing with. Having many individual functions and the functions themselves work separately and independently make performance, especially optimization within each function becoming more and more crucial in terms of reasonable response time and utilization of resources. Engineers that can balance the benefits of serverless frameworks and successfully optimize each and all functions in the system are admirable.

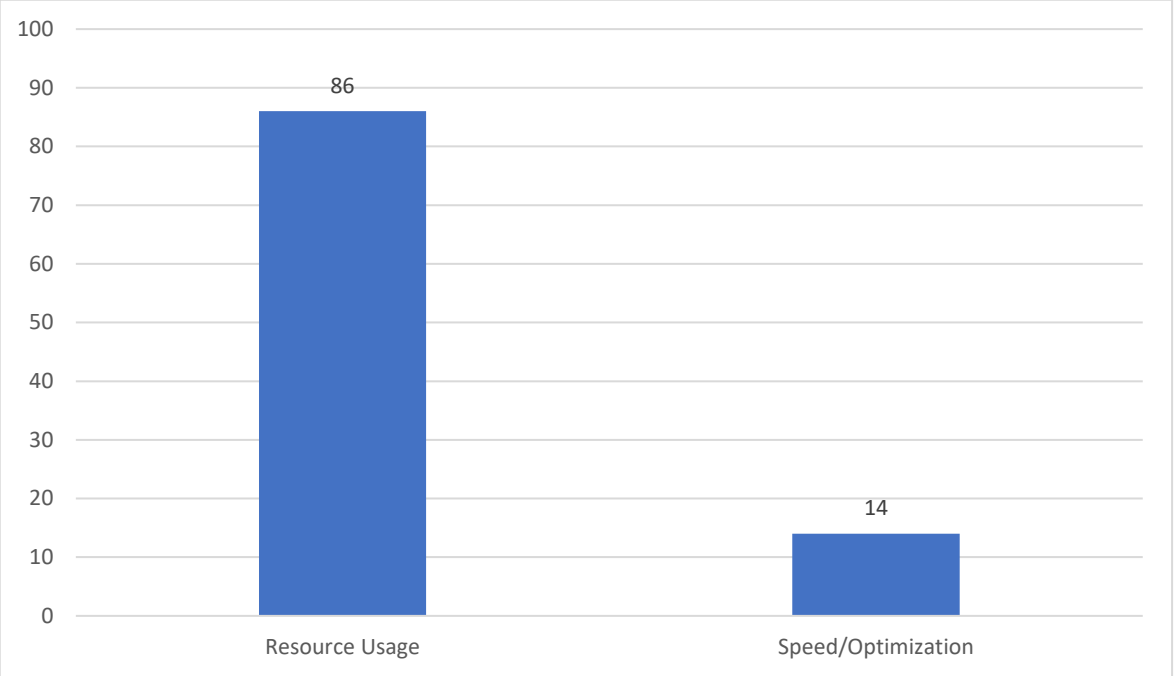


Figure 10 An overview of Performance Issues

- **Resource Usage** (86/1809, 4.75%): From overall them of subject that is subject of this theme, mainly which are in demand are using CPU, memory, and storage mostly. Each resource of the function of the computer system is considered crucial that must be effectively managed and utilized for having stable yet meet the performance requirement. There is highly essential efficient management of resource for ensuring performance and ability to scale in software applications.
- **Speed and Optimization** (14/1809, 0.77%): Not having most shared issues but this subtheme is important for cost effectiveness and utilization of the selected service. Having optimized resource in term of resource management is one step ahead to increase performance without must pay extra dollars.

5.1.4. GUI (Graphical User Interface) Issues

Unlike majority of open-source repositories, Serverless Framework provides a user interface which plays as a portal to help bridging users with various tasks and actions. A dashboard provides insight about your applications, metrics, and stack traces for debugging purposes. Software engineers can find themselves comfortable with graphs about metric aggregation, information about uncaught errors and essential traces to identify the issues without logging into each cloud provider portal or installing CLI for all the cloud providers and logging all

monitoring metrics. Other than that, Serverless Framework also mitigates the necessity of checking native portals from cloud providers when it comes to CI/CD, adjusting parameters, executing tests, or upgrading packages versions.

However, GUI provided by Serverless Framework also causes a small number of problems. These problems are categorized into three smaller subthemes with the dominance of Interface Design issues and followed by User Experience and Graphic Elements.

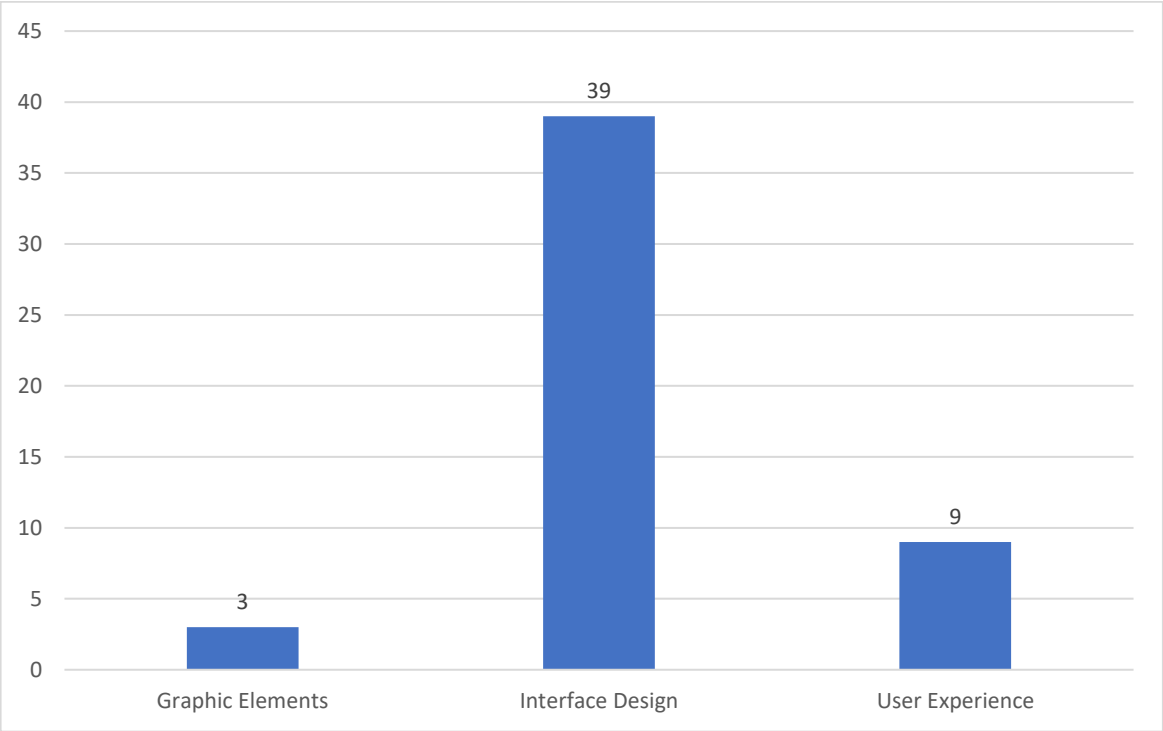


Figure 11 An overview of GUI issues.

- **Interface Design** (39/1809, 2.16%): This sub-theme looks at challenges that are inherent on designing interfaces that are both effective and user-friendly as well as aesthetically appealing. Problems here may start with the complications of aligning the design to the need of the user and how it intersects the technical constraints. Serverless Framework is heavy technical repository which difficult justify contains-it-all design. Minor issues share about Interface Design, but the way to resolve these will help increase user experience and usability.

- **User Experience** (9/1809, 0.50%): This portrays some considerations given towards the general intended user and application software interaction. Among its considerations might comprise the need to design intuitive, responsive, as well as visually appealing user interfaces. User Experience is not the most critical factor for

Serverless Framework requirements, in fact it does not bring a lot of troubles while being in competition to this framework context.

- **Graphic Elements** (3/1809, 0.17%): Issues under this subtheme might be regarding the design elements like icons, buttons, and layouts which are significant to the aesthetics and using the application designed and woven well. Constitute a statistically insignificant fraction from overall entries counted as a minor issue.

5.1.5. API-Related Issues

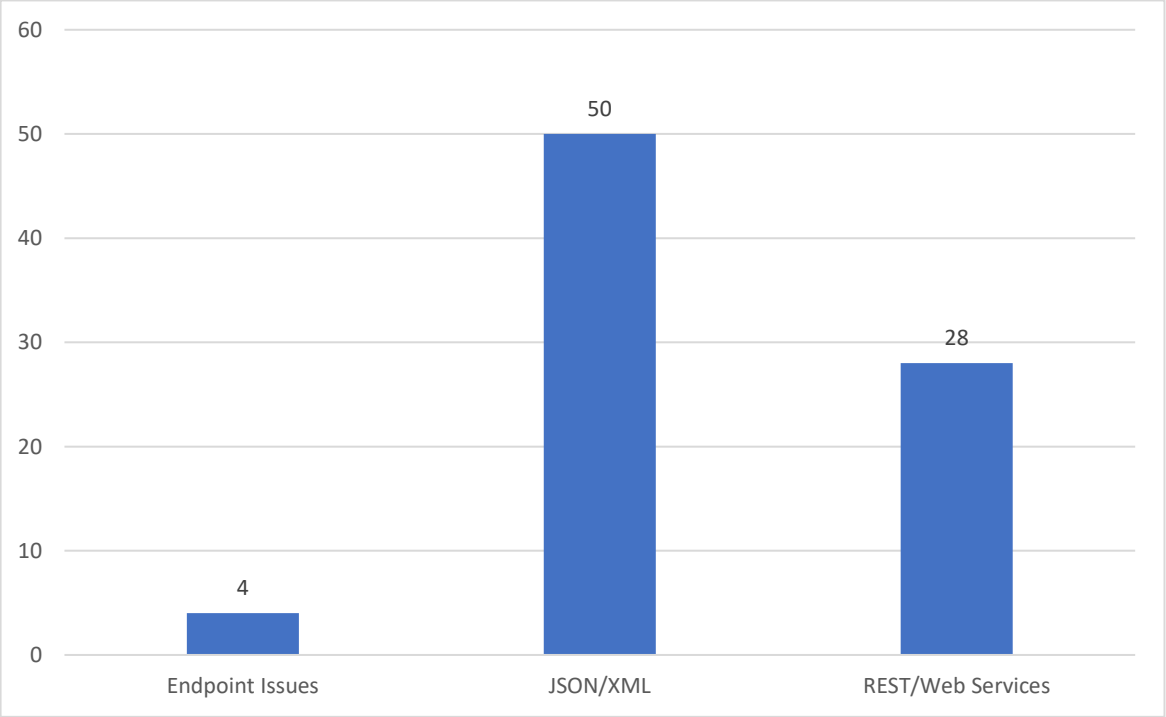


Figure 12 An overview of API-Related issues

API-Related issues are very common in serverless architecture because there are several communications between services. As a rule, the communications are often sending out from each other with the different formats and sometimes with different models and techniques which require data parsing and data handling one to another being partially altered information. Most of those steps are easily to be the complexity.

- **JSON/XML Handling** (50/1809, 2.76%): This is applicable to many issues in this sub-theme, and this is probably because JSON and XML were prevalent as the common data formats used by web services and applications. With most

communications between the multi-platforms, so that it can be complexities in the parsing, validating, and transforming these data formats when they are still popularly used.

- **REST/Web Services** (28/1809, 1.55%): This subtheme reflects issues concerning the problems of realizing RESTful services. These problems may have been originating from complexities in fulfilling statelessness and cache-ability prescription by principles of REST coupled with effective mechanisms of exchanging data.
- **Endpoint Issues** (4/1809, 0.22%): Not all the endpoint issues might just be negligible but perhaps some of the less frequent ones. It may be due to possible security vulnerability, scalability issue, or difficulty in handling an endpoint configuration in a distributed architecture.

5.1.6. CI/CD Issues

Particularly, in relation to the serverless architecture, Continuous Integration (CI) and Continuous Deployment (CD) may present specific challenges and difficulties. Serverless architectures depict a new type of computing architecture whereby they comprise functions that get triggered by events and are administered at zero. The way applies radically from the standard technique based on servers.

Serverless functions often have the dependencies to need to pack and deploy with the function code. The ones needed should get included only, and the making sure they work in the serverless environment can be hard. It often becomes more dependent on certain not so generative tooling or scripts. That is due to the fact that the 170 issues, out of the total of 1809 issues, belong to the Deployment Processes, which is roughly a 9.4%.

Decision making process in CI/CD, and especially in serverless environments, is hard given that many decisions must be made, and that fuzziness exists concerning functional requirements and architecture decisions. This complexity makes it relevant to the study on the decisions in open-source projects in the field of CI/CD, based on which it can be told that most of the decisions are associated with functional requirements (Luo, Liang, Shahin, Li, & Yang, 2022). Although Integration Tools and Pipeline Configuration do not raise a lot

of issues, this process is yet very important to give thought.

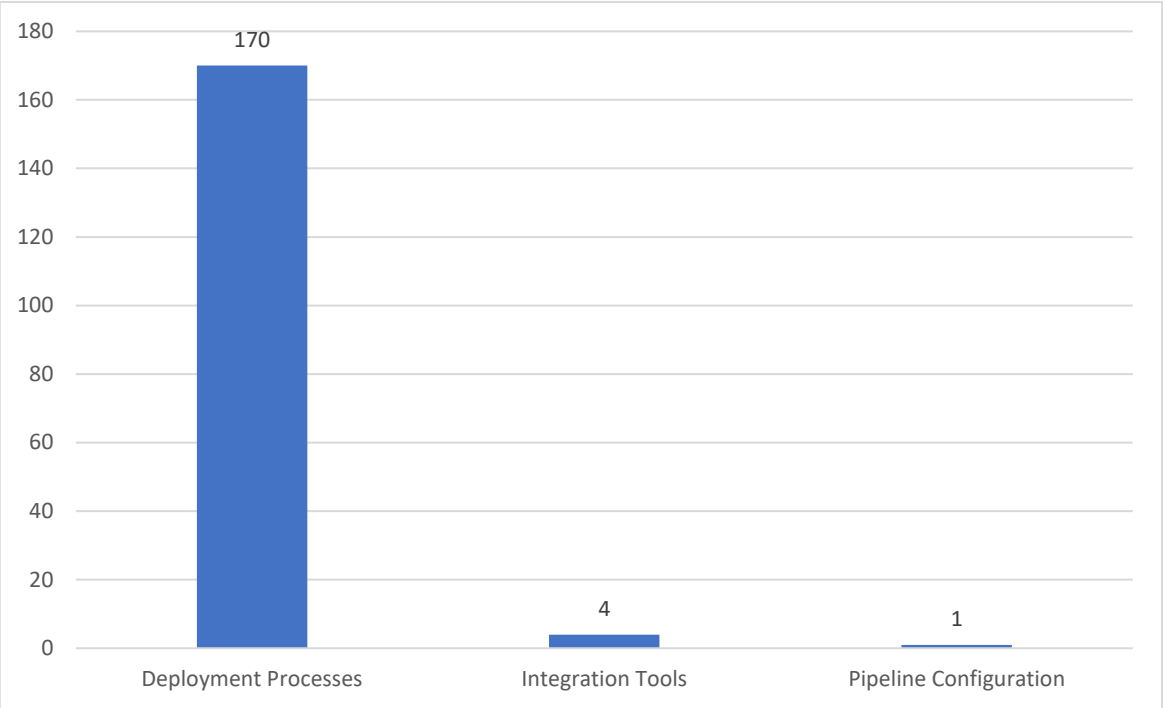


Figure 13 An overview of CI/CD issues.

- **Deployment Processes** (170/1809, 9.40%): Serverless infrastructures introduce new fault modes that do not exist in traditional deployments, which in turn introduces challenges to implement correct services (Kallas, Zhang, Alur, Angel, & Liu, 2023). This is the reflection of the dominance of a sub-theme on the CI/CD category, as this goes onto exhibit how complex and critical deployment process is in software development. Challenges here include issues linked with automation, environment configuration, version control, integration issues with various tools and platforms. It involves complexities in building and deploying serverless applications like design of a workflow and migration of other apps, especially where there are transactions that need to involve different parties (Meladakis et al., 2022).

- **Integration Tools** (4/1809, 0.22%) and **Pipeline Configuration** (1/1809, 0.06%): All less occurrence areas but equally important to have a smooth functioning CI/CD pipeline. Here, occasional issues may arise in the form of a requisite to integrate several tools properly with each other and reap optimal performance and reliability from pipelines via proper configuration.

5.1.7. Cloud Services

Cloud services are increasingly central to IT infrastructure, and the data reflects related challenges:

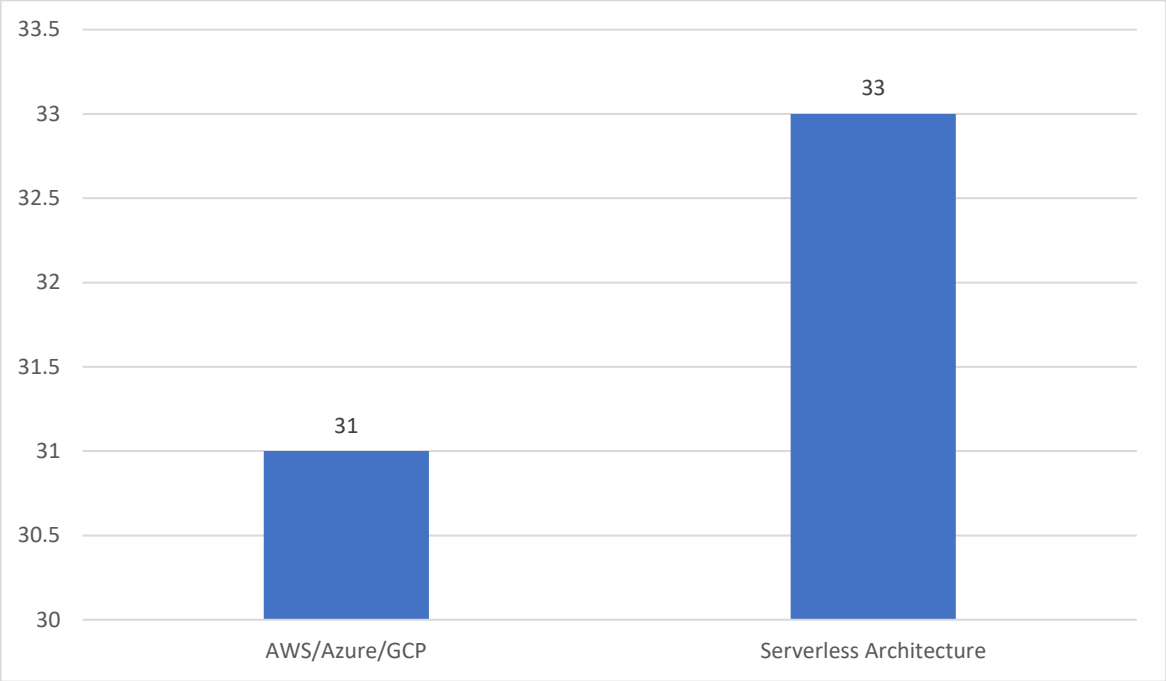


Figure 14 An overview of Cloud Services issues.

- **Serverless Architecture** (33/1809, 1.82%): Issues in serverless computing could be due to the complexity of managing stateless functions, scalability, and dealing with cold starts, where functions may have latency issues when they are invoked after a period of inactivity.
- **AWS/Azure/GCP** (31/1809, 1.71%): As the major cloud platforms, challenges in these services might include platform-specific configurations, optimization for cost and performance, and navigating the vast array of services offered by these providers.

5.1.8. Other Themes

Additional themes include Database issues (specifically SQL/NoSQL) and Error Handling (notably, bugs), each contributing a smaller yet significant part of the overall issue spectrum. Challenges in databases might relate to the efficient management of data and ensuring consistency and performance, while error handling primarily concerns identifying and

resolving software bugs effectively. Testing is also worth mentioning even though the share of issues is insignificant.

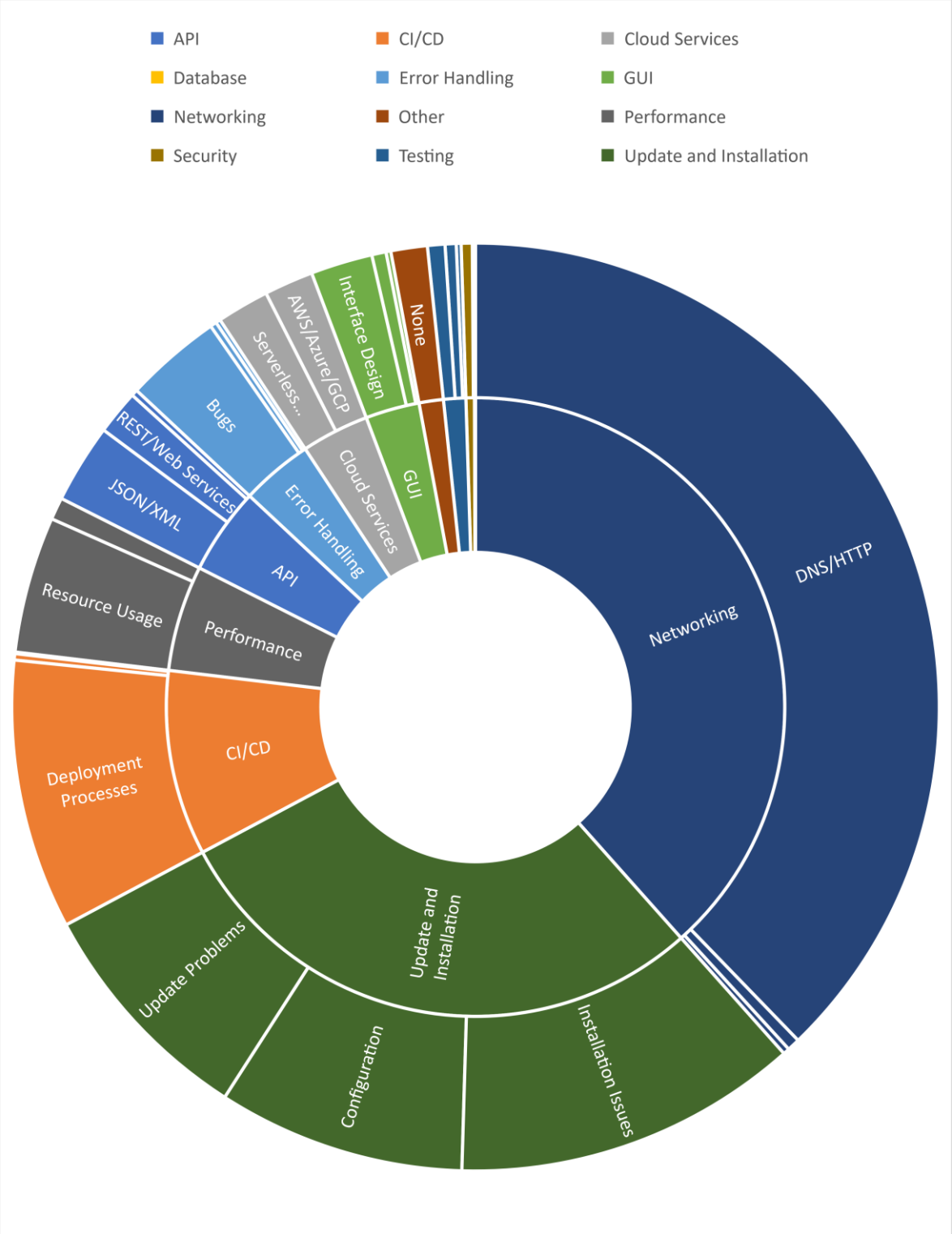


Figure 15 Portions of issue types.

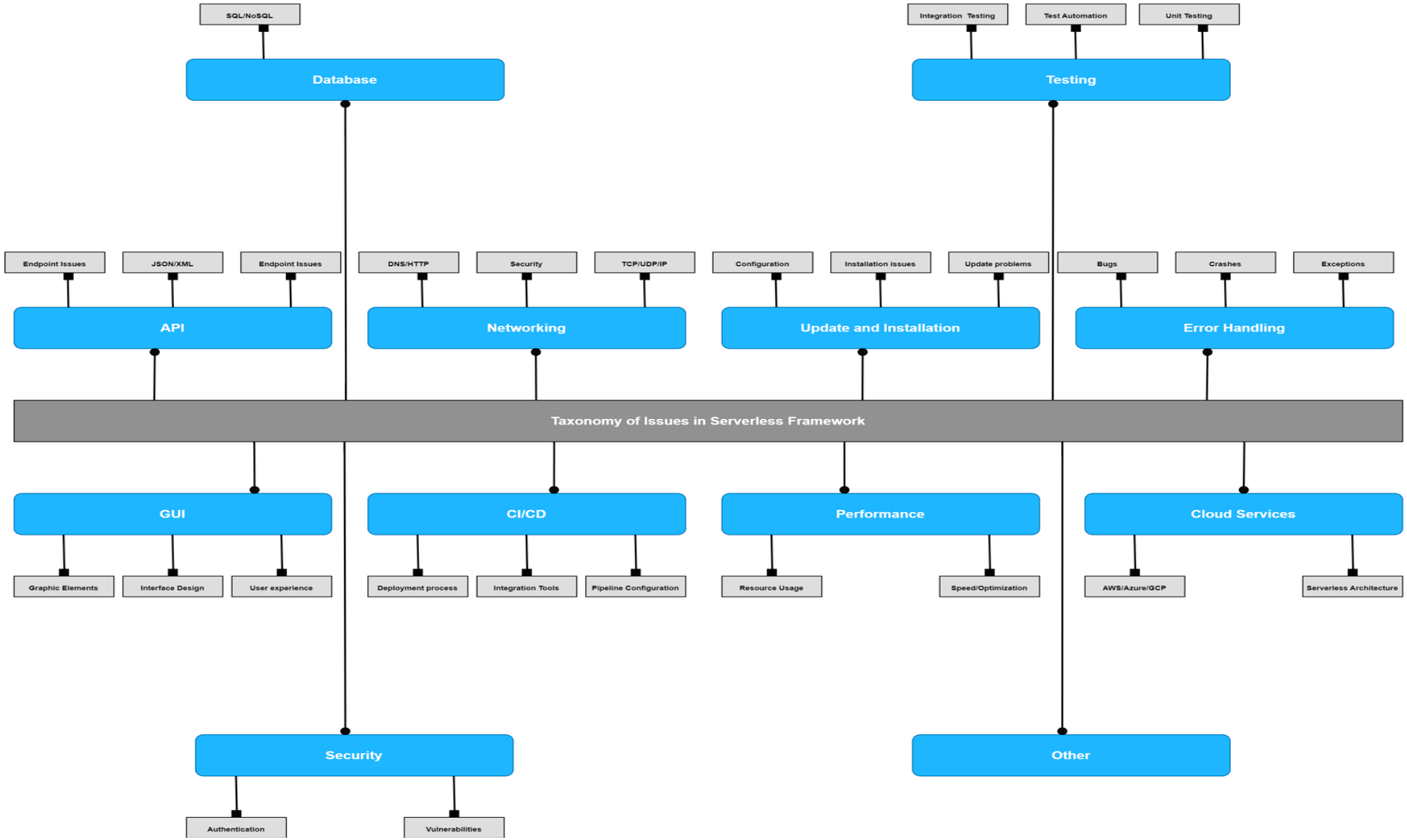


Figure 16 Taxonomy of issue types.

5.2. Causes of issues (RQ2): An overview of the cause

In this section, this master thesis will provide a brief description of the causes of the issues in the context of Serverless Framework. In the context of Serverless Framework, the causes of some issues are not always following the common trends of serverless frameworks in general. Some of the challenges are common in other serverless open-source frameworks or serverless architecture in general but in Serverless Framework, a few of the popular known causes are not making up majority of the total issues. With this understanding, Challenges of Serverless Edge Computing with 1475 of total 1809 issues which equivalent to 81.54% of all the issues that are being categorized. The challenges come with several smaller detail causes topics, including Resource Management, Function Execution and Runtime Issues, Integration and Compatibility and so on.

Limitation: In this master thesis, the causes of the total issues are listed with a brief description of what each cause means in serverless architecture context of Serverless Framework. The result is from a python using a keyword match algorithm which has a minimum effort of sanitization due to time constrains the huge amount of efforts required.

Architectural Issues and Technical Debt (41/1809, 2.27%)

- Modular Architecture: Serverless computing breaks down applications into smaller, event-driven functions. While this modularization offers scalability and agility, it also introduces complexity in orchestration and inter-service communication.
- Technical Debt: Quick deployments and easy setup can lead to rushed decisions in architectural design, accumulating technical debt. This debt manifests as increased maintenance costs, difficulties in scaling, and challenges in introducing new features or making significant changes (Lenarduzzi et al., 2021).

Complexity in Design Architecture (0/1809, 0%)

- Scalability vs. Complexity: The scalability of serverless architectures comes at the cost of increased complexity in managing numerous small, discrete components.
- Decoupling and Integration Issues: The decoupled nature of serverless functions can lead to challenges in integration and comprehensive testing, requiring more sophisticated design and testing strategies (Li et al., 2021).

Challenges in Serverless Edge Computing (1475/1809, 81.54%)

- Resource Management (335/1475, 22.7% of total issues in Challenges in Serverless Edge Computing): Balancing and managing resources effectively in serverless edge computing is complex due to the distributed nature of resources.
- Function Execution and Runtime Issues (286/1475, 19.39% of total issues in Challenges in Serverless Edge Computing): Problems related to the runtime environment of serverless functions, including execution errors, function triggers, and runtime limitations.
- Integration and Compatibility (250/1475, 16.95 % of total issues in Challenges in Serverless Edge Computing): Challenges related to integrating serverless components with existing systems, services, and third-party APIs. Compatibility issues with different platforms and technologies can also be included here.
- Service Deployment and Lifecycle Management (13/1475, 0.88%): Efficient deployment and management of services are challenging, especially in maintaining performance and reliability at the edge of the network (Xie et al., 2021).
- Others (Event Handling and Messaging – 19/1475, User Experience and Interface – 19/1475, Data Management and Storage – 15/1475): These issues are taking place in this cause however, majority of them are small and easy-to-handle issues.

Security Concerns (116/1809, 6.42%)

- New Security Paradigms: The fragmented application boundaries in serverless architectures necessitate a different approach to security compared to traditional cloud or virtualized environments.
- Security Shortcomings: Identifying and addressing the unique security shortcomings of serverless computing is crucial, particularly in areas like data protection and access control (Marin et al., 2021).

Cold Start Issue (2/1809, 0.11%)

- Startup Latency: A notable issue in serverless computing is the 'cold start' problem, where there is a delay in function execution due to the time taken to allocate resources

for a new instance. This issue is more pronounced in languages that have longer startup times.

Resource Limitations and Performance Issues (167/1809, 9.23%)

- Memory and Compute Constraints: Serverless functions are subject to limitations in terms of memory and compute capacity, which can impact the performance of compute-intensive applications.

Monitoring and Debugging Difficulties (167/1809, 9.23%)

- Observability Challenges: Due to the ephemeral and stateless nature of serverless functions, traditional monitoring and debugging tools are often inadequate, requiring new tools and approaches for effective observability.

Vendor Lock-in and Portability Issues (0/1809, 0%)

- Dependence on Specific Cloud Providers: Applications designed for one serverless platform might not be easily portable to another, leading to vendor lock-in. This limits flexibility and can increase long-term costs.

Cost Prediction and Optimization (4/1809, 0,22%)

- Unpredictable Costs: While serverless computing is often cost-effective, predicting costs can be challenging due to the dynamic nature of resource utilization. Managing and optimizing costs requires careful analysis and understanding of the pricing models of serverless services.

Skillset and Training Requirements (2/1809, 0.11%)

- Need for Specialized Knowledge: The unique aspects of serverless architecture require specialized skills and understanding, which can be a barrier for teams accustomed to traditional server-based environments.

6. Discussion

6.1. The findings and their usage

With the findings from this master thesis, software engineers and cloud practitioners can have an overall picture of Serverless Framework and its unique types of issues to make sensible decision before choosing the framework into their project. The findings also give a strong foundation for further research about the other serverless frameworks and based on that, we can have an understanding about serverless computing and its frameworks in practice.

The findings of this master thesis will also provide useful analysis for parties that are interested in serverless architecture and Serverless Framework to make decision to shift their infrastructure and architecture toward serverless architecture using Serverless Framework. It helps identifying the complexity areas and brings the transparency into the decision-making process. This will also help mitigate unforeseen complications that may occur during the adoption process of starting to use or migrating to Serverless Framework.

This master thesis will also give the audience an overall picture of serverless architecture and Serverless Framework. This is helpful for educational use and help strengthening the understanding of serverless computing in a practical way.

6.2. Limitations

Time constrains and limited resources are the main limitations of this master thesis. This master thesis is limited to one serverless framework which does not give an overall big picture of serverless frameworks in general. There are other popular serverless frameworks such as OpenFaas, Zappa, Vendia Serverless Express or kubeless, etc. which have different types of issues and causes. This master thesis is focusing on Serverless Framework which will only provide the serverless architecture picture from the lens of Serverless Framework.

The limitation on resources impacts the outcome of the research which narrows the research outcome on in-depth analysis of issue types. Briefly mentioning about their causes and the disappearance of potential solutions make the possible outcome of the research less

attractive. However, it will create possibilities for future research and development on the theories that are provided in this thesis findings.

The approach of the analysis in this master thesis also has its limitation. Analyzing the closed issues does not give the most recent status of the development of Serverless Framework. Because of that, the thesis may not be up-to-date with the trending issue types and their causes which may give a slightly different perspective from the most recent developments of Serverless Framework and serverless computing. However, the impact and the gap are considerably minimal and acceptable.

6.3. Further research topics

Further research to study the cause is essential and can be a potential research topic to understand why the causes are appearing during the utilization of Serverless Framework. It will require a more sophisticated study and interviews of cloud practitioners, software developers, software architects and solution architects with seasoned backgrounds and various experience in different cloud technologies to have wide perspectives for the analysis.

An in-depth analysis of the causes will be required to study the root causes of majority of serverless architecture issues and based on that, we can propose solutions in the future research that can help professionals identify the advantages and disadvantages when choosing the cloud architecture and framework.

Another further research topic is to unify the issue types, causes and potential solutions. Having the theory of the issue types, their causes, and possible solutions, we can make the value tree or a decision model which helps professionals, engineers, enterprises, and organizations to select the right technologies and frameworks to start with which will help increasing the performance of architecture, optimizing the workflow with the minimum unforeseen issues, cost efficiency, high level stability, and top-notch security.

Conclusions

The findings show the significant share of DNS/HTTP type of issues in Serverless Framework which indicates the common trend in serverless computing and serverless architecture in general. The mentioned type of issues boils down to the nature of serverless computing and Serverless Framework itself rely on network communication between different instances, entities and services. The result is not a surprise and as mentioned in many earlier studies, network communication is and will always be one of the most common types of issues in serverless architecture.

Taking this into account, as the conclusion draws near on this exploration into serverless architecture and the Serverless Framework, it is of key importance at this juncture to re-evaluate our initial objectives and what has been consequentially discovered. The present thesis embarked on the quest of disentangling the complexities that surround serverless computing, in an attempt through to provide a fresh understanding of its architecture and the role leading the way by the Serverless Framework that enables one harness such a modern compute paradigm.

Serverless architecture is an important thing to understand. The advent of a non-server-based serverless computing model has entirely revolutionized the paradigm in which modern applications are deployed and managed. It has been a radical departure from all the traditional common models that had servers at the core. It lets developers or organizations work more efficiently by freeing up the valuable time consumed in managing and scaling the server infrastructure and spending better time in core product development. As such, serverless computing is hence positioned as a transformational force in cloud computing, further strengthening the value proposition with respect to operational complexities that are reduced by serverless, and the cost efficiencies introduced through pay-per-use model.

The Serverless Framework: A Catalyst for Efficiency

In the perspective of serverless architecture, the Serverless Framework is an important tool. Facilitates painless scaling and operational agility in the process of deploying serverless applications. It offers a high-level capacity for developers in their application development since most of them would not really have to deal with complexities in managing servers hence turning out to be more productive and faster in their deployment cycles. This results

in a more effective development workflow, one important fact of today's rapidly changing tech landscape.

Negotiating the Trade-offs

However, with its numerous advantages, serverless computing does also pose its challenges. Challenges such as cold starts, vendor lock-ins, networking, updates and installations and intricacies of debugging serverless applications embody its significant trade-offs. This thesis has explored these challenges with a view to highlighting critical types of issues and briefly discuss about their causes for the adoption of this architecture in reasonable measure. This highlights the need for strategic review of the benefits and pitfalls ensuring that the decision to go for serverless computing resonates with the needs and restrictions in any project.

Future outlooks and changing landscape.

Continuing further, the field of serverless computing is all poised to further shape up. With more advancements in technology, there can be better cold start optimization, improved debugging tools, and more options for vendor. Thus, this trend is likely to evolve further and eliminate most of these limitations in present times as the role of serverless computing with technological ecosystems becomes increasingly well-entrenched.

I may conclude this thesis by underscoring the need for understanding serverless architecture and using tools like the Serverless Framework. To investigate what benefits and trade-offs the serverless computing offers will provide insights not just to the practitioners but also to the overall landscape of cloud computing as well. In this connection, the knowledge and perspectives accrued from this study will positively be important in influencing the future applicability as well as development of serverless architectures.

References

Ali, M. A. (2021, July 15). Serverless: Next Generation of Cloud Computing. *International Journal for Research in Applied Science and Engineering Technology*, *9*(VII), 1168–1172. https://doi.org/10.22214/ijraset.2021.36551

Alqaryouti, O., & Siyam, N. (2018). Serverless computing and scheduling tasks on cloud: a review. American Academy of Sciences Research Journal of Engineering and Technology Science, 40(1), 235–247.

Austen Collins. (2022, November 15). https://www.serverless.com/author/austencollins

Bac, T., Tran, M., & Kim, Y. (2022). Serverless Computing Approach for Deploying Machine Learning Applications in Edge Layer. *2022 International Conference on Information Networking (ICOIN)*, 396-401. https://doi.org/10.1109/ICOIN53446.2022.9687209.

Baldini, I., Cheng, P., Fink, S. J., Mitchell, N., Muthusamy, V., Rabbah, R., ... & Suter, P. (2017). Serverless computing: Current trends and open problems. In Research Advances in Cloud Computing.

Cassel, G. A. S., Rodrigues, V. F., da Rosa Righi, R., Bez, M. R., Nepomuceno, A. C., & da Costa, C. A. (2022). Serverless computing for Internet of Things: a systematic literature review. Future Generation Computer Systems, 128, 299–316.

Chen, F., Li, L., Jiang, J., & Zhang, L. (2014). Predicting the number of forks for open source software project., 40-47. https://doi.org/10.1145/2627508.2627515.

Djemame, K., Parker, M., & Datsev, D. (2020). Open-source Serverless Architectures: An Evaluation of Apache OpenWhisk. *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 329-335. https://doi.org/10.1109/UCC48980.2020.00052.

Eismann, S., et al. (2020). A review of serverless use cases and their characteristics. SPEC RG Gainesville Tech. Rep.

Eismann, S., Scheuner, J., Van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., ... & Iosup, A. (2021). Serverless Applications: Why, When, and How? IEEE Software, 38(1), 32-39.

Golec, M., Ozturac, R., Pooranian, Z., Gill, S. S., & Buyya, R. (2021). iFaaSBus: a security and privacy-based lightweight framework for serverless computing using IoT and machine learning. IEEE Transactions on Industrial Informatics, 18(5), 3522–3529.

Jiang, L., Pei, Y., & Zhao, J. (2020, January 1). Overview Of Serverless Architecture Research. *Journal of Physics: Conference Series*, *1453*(1), 012119. https://doi.org/10.1088/1742-6596/1453/1/012119

Kallas, K., Zhang, H., Alur, R., Angel, S., & Liu, V. (2023). Executing Microservice Applications on Serverless, Correctly. Proceedings of the ACM on Programming Languages, 7, 367 - 395. https://doi.org/10.1145/3571206.

Kavaler, D., Sirovica, S., Hellendoorn, V., Aranovich, R., & Filkov, V. (2017). Perceived language complexity in GitHub issue discussions and their effect on issue resolution. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 72-83. https://doi.org/10.1109/ASE.2017.8115620.

Kelly, D., Glavin, F., & Barrett, E. (2020). Serverless Computing: Behind the Scenes of Major Platforms. *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, 304-312. https://doi.org/10.1109/CLOUD49709.2020.00050.

Leitner, P., Wittern, E., Spillner, J., & Hummer, W. (2019). A mixed-method empirical study of Function-as-a-Service software development in industrial practice. Journal of Systems and Software, 149, 340-359.

Lenarduzzi, V., Daly, J., Martini, A., Panichella, S., & Tamburri, D. (2021). Toward a Technical Debt Conceptualization for Serverless Computing. IEEE Software, 38, 40-47. https://doi.org/10.1109/MS.2020.3030786.

López, P., Artigas, M., Shillaker, S., Pietzuch, P., Breitgand, D., Vernik, G., Sutra, P., Tarrant, T., & Ferrer, A. (2019). ServerMix: Tradeoffs and Challenges of Serverless Data Analytics. *ArXiv*, abs/1907.11465.

Luo, Y., Liang, P., Shahin, M., Li, Z., & Yang, C. (2022). Decisions in Continuous Integration and Delivery: An Exploratory Study., 457-462. https://doi.org/10.18293/SEKE2022-171.

Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., & Guo, M. (2021). The Serverless Computing Survey: A Technical Primer for Design Architecture. *ACM Computing Surveys (CSUR)*, 54, 1 - 34. https://doi.org/10.1145/3508360.

Lynn, T., Rosati, P., & Lejeune, A. (2017). An empirical study on the adoption of serverless computing and the impact of its use on future software development. In ACM Computing Research Repository.

Lynn, T., Rosati, P., Lejeune, A., & Emeakaroha, V. (2017). A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms. 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 162-169. https://doi.org/10.1109/CloudCom.2017.15.

Mahmoudi, N., & Khazaei, H. (2020, December 7). Temporal Performance Modelling of Serverless Computing Platforms. Proceedings of the 2020 Sixth International Workshop on Serverless Computing. https://doi.org/10.1145/3429880.3430092

Maissen, P., Felber, P., Kropf, P., & Schiavoni, V. (2020). FaaSdom: a benchmark suite for serverless computing. *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. https://doi.org/10.1145/3401025.3401738.

Mampage, A., Karunasekera, S., & Buyya, R. (2022, January 31). A Holistic View on Resource Management in Serverless Computing Environments: Taxonomy and Future Directions. ACM Computing Surveys, 54(11s), 1–36. https://doi.org/10.1145/3510412

Marin, E., Perino, D., & Pietro, R. (2021). Serverless computing: a security perspective. *Journal of Cloud Computing*, 11, 1-12. https://doi.org/10.1186/s13677-022-00347-w.

McGrath, G., & Brenner, P. R. (2017). Serverless computing: design, implementation, and performance. In IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 405–410.

Meladakis, K., Zeginis, C., Magoutis, K., & Plexousakis, D. (2022). Transferring transactional business processes to FaaS. Proceedings of the Eighth International Workshop on Serverless Computing. https://doi.org/10.1145/3565382.3565882.

Poorvadevi, R., & Ramamoorthy, S. (2018). Security Enhancement in Multi Clouds Using Serverless Computing Approach., 6, 12. https://doi.org/10.11648/j.iotcc.20180601.12.

Rafiee, H., & Meinel, C. (2013). A Secure, Flexible Framework for DNS Authentication in IPv6 Autoconfiguration. 2013 IEEE 12th International Symposium on Network Computing and Applications, 165-172. https://doi.org/10.1109/NCA.2013.37.

Sahin, S.E., Karpat, K., Tosun, A. (2019). Predicting Popularity of Open Source Projects Using Recurrent Neural Networks. In: Bordeleau, F., Sillitti, A., Meirelles, P., Lenarduzzi, V. (eds) Open Source Systems. OSS 2019. IFIP Advances in Information and Communication Technology, vol 556. Springer, Cham. https://doi.org/10.1007/978-3-030-20883-7_8.

Sawhney, R. (2019). Introduction to Azure Functions. *Beginning Azure Functions*. https://doi.org/10.1007/978-1-4842-4444-9_1.

Scheuner, J., & Leitner, P. (2020). Function-as-a-service performance evaluation: a multifocal literature review. Journal of Systems and Software, 170, 110708.

Tariq, A., et al. (2020). Sequoia: enabling quality-of-service in serverless computing. In Proceedings of the 11th ACM Symposium on Cloud Computing.

Xie, R., Tang, Q., Qiao, S., Zhu, H., Yu, F., & Huang, T. (2021). When Serverless Computing Meets Edge Computing: Architecture, Challenges, and Open Issues. *IEEE Wireless Communications*, 28, 126-133. https://doi.org/10.1109/mwc.001.2000466.

Xing Li, Xue Leng, and Yan Chen, "Securing Serverless Computing: Challenges Solutions and Opportunities," 2023.

Yussupov, V., Breitenbücher, U., Kaplan, A., & Leymann, F. (2020). SEAPORT: Assessing the Portability of Serverless Applications., 456-467. https://doi.org/10.5220/0009574104560467.

Yussupov, V., Wurster, M., Breitenbücher, U., Leymann, F., & Reinfurt, L. (2019). Research challenges in serverless computing. In 2019 IEEE International Conference on Software Architecture Companion (ICSA-C) (pp. 97-100). IEEE.

Appendices

Appendix 1. Extraction code

```
import requests
import time
import pandas as pd

# Replace with your GitHub token and repository details
token = [read access token]
owner = 'serverless'
repo = 'serverless'

url = f"https://api.github.com/repos/{owner}/{repo}/issues"
headers = {
'Authorization': f'token {token}',
'Accept': 'application/vnd.github.v3+json'
}

params = {
'state': 'closed',
'per_page': 100 # Fetch 100 issues per request
}

issues_list = []

while url:
response = requests.get(url, headers=headers, params=params)
if response.status_code == 403: # Rate limit reached
reset_time = int(response.headers.get('X-RateLimit-Reset', 0))
sleep_time = max(reset_time - int(time.time()), 0) + 1
print(f"Rate limit reached. Sleeping for {sleep_time} seconds.")
time.sleep(sleep_time)
continue
if response.status_code != 200:
print(f"Error: Unable to fetch issues. HTTP Response Code: {response.status_code}")
print(f"Response Content: {response.text}")
break
issues = response.json()
for issue in issues:
if 'pull_request' in issue: # Skip pull requests
continue
# Initialize set of participants with the creator of the issue
participants = set([issue.get('user', {}).get('login', '')])
comments_url = issue.get('comments_url', '')
if comments_url: # Fetch comments to get additional participants
```

```
comments_response = requests.get(comments_url, headers=headers)
if comments_response.status_code == 200:
comments = comments_response.json()
for comment in comments:
participants.add(comment.get('user', {}).get('login', ''))
# Append the issue details to the list
issues_list.append([
issue.get('number', ''),
issue.get('title', ''),
issue.get('html_url', ''),
issue.get('created_at', ''),
issue.get('closed_at', ''),
len(participants)
])
if 'next' in response.links:
url = response.links['next']['url']
else:
url = None

# Print the length of issues_list
print(f"Number of issues fetched: {len(issues_list)}")

# Write the fetched issues to a CSV file
df = pd.DataFrame(issues_list, columns=['Issue ID', 'Issue Title', 'Issue URL Link', 'Issue Open Date', 'Issue Closed Date', 'Number of Participants'])
df.to_csv('serverless_issues.csv', index=False, sep=',')

# Print a message after writing the file
print("CSV file has been written.")
```

Appendix 2. Issue detail extraction code

(GitHub repository: https://github.com/khactam/extract-github-issue-detail/tree/main)

```
import requests
import pandas as pd
import os
from dotenv import load_dotenv

load_dotenv()

GITHUB_TOKEN = os.getenv('GITHUB_TOKEN')

TOKEN = GITHUB_TOKEN
HEADERS = {
    "Authorization": f"token {TOKEN}",
    "Accept": "application/vnd.github.v3+json",
}

def get_issue_details(repo_url):
    # Convert the issue URL to the API URL for details
    api_url = repo_url.replace("github.com", "api.github.com/repos")

    # Fetch the issue details
    response = requests.get(api_url, headers=HEADERS)
    if response.status_code != 200:
        print(f"Error {response.status_code}: {response.text}")
        return None
    issue_data = response.json()

    # Fetch comments for the issue
    comments_url = issue_data["comments_url"]
    comments_response = requests.get(comments_url, headers=HEADERS)
    if comments_response.status_code != 200:
        print(f"Error {comments_response.status_code}: {comments_response.text}")
        return None
    comments_data = comments_response.json()

    # Extracting the issue description and comments
    details = {
        "description": issue_data["body"],
        "comments": [comment["body"] for comment in comments_data]
    }
    return details

# Read the Excel file
df = pd.read_excel("C:/Users/tamn/Downloads/extractGithubIssueProj/trialrunv1.xlsx")
```

```
# Fetch details for each GitHub issue
results = []
for _, row in df.iterrows():
    issue_id = row["Issue ID"]
    issue_title = row["Issue Title"]
    issue_link = row["Issue URL Link"]

    issue_data = get_issue_details(issue_link)
    results.append({
        "Issue ID": issue_id,
        "Issue Title": issue_title,
        "GitHub Link": issue_link,
        "Issue Details": str(issue_data)
    })

# Convert results to a DataFrame and save to CSV
output_df = pd.DataFrame(results)
output_df.to_csv("C:/Users/tamn/Downloads/extractGithubIssueProj/file.csv", index=False)
```

Appendix 3. Summary number of main themes and subthemes

Main Theme	Sub Theme	Number of Issues
API	Endpoint Issues	4
API	JSON/XML	50
API	REST/Web Services	28
CI/CD	Deployment Processes	170
CI/CD	Integration Tools	4
CI/CD	Pipeline Configuration	1
Cloud Services	AWS/Azure/GCP	31
Cloud Services	Serverless Architecture	33
Database	SQL/NoSQL	1
Error Handling	Bugs	61
Error Handling	Crashes	3
Error Handling	Exceptions	4
GUI	Graphic Elements	3
GUI	Interface Design	39
GUI	User Experience	9
Networking	DNS/HTTP	683
Networking	Security	8
Networking	TCP/UDP/IP	4
Other	None	23
Performance	Resource Usage	86
Performance	Speed/Optimization	14
Security	Authentication	7
Security	Vulnerabilities	1
Testing	Integration Testing	11
Testing	Test Automation	7
Testing	Unit Testing	3
Update and Installation	Configuration	156
Update and Installation	Installation Issues	218
Update and Installation	Update Problems	147

Appendix 4. Main themes and subthemes categorization script

```
# Redefining themes with subthemes
subthemes = {
    "Networking": {
        "DNS/HTTP": ["dns", "http"],
        "TCP/UDP/IP": ["tcp", "udp", "ip "],
        "Security": ["ssl", "tls"]
    },
    "Testing": {
        "Unit Testing": ["unit test", "unittest"],
        "Integration Testing": ["integration test"],
        "Test Automation": ["automated test", "automation"]
    },
    "GUI": {
        "User Experience": ["ux", "user experience"],
        "Interface Design": ["ui", "interface", "design"],
        "Graphic Elements": ["graphic", "visual"]
    },
    "Update and Installation": {
        "Installation Issues": ["install", "setup"],
        "Update Problems": ["update", "upgrade"],
        "Configuration": ["configure", "configuration"]
    },
    "Performance": {
        "Speed/Optimization": ["performance", "speed",
"optimize"],
        "Latency Issues": ["latency"],
        "Resource Usage": ["resource", "memory", "cpu"]
    },
    "CI/CD": {
        "Integration Tools": ["jenkins", "travis", "ci/cd"],
        "Deployment Processes": ["deployment", "deploy"],
        "Pipeline Configuration": ["pipeline"]
    },
    "Security": {
        "Authentication": ["auth", "authentication"],
        "Encryption": ["encrypt"],
        "Vulnerabilities": ["vulnerability", "security"]
    },
    "Database": {
        "SQL/NoSQL": ["sql", "nosql"],
        "Performance": ["database performance", "db performance"],
        "Configuration": ["database config", "db config"]
    },
    "API": {
        "REST/Web Services": ["api", "rest", "web service"],
        "JSON/XML": ["json", "xml"],
        "Endpoint Issues": ["endpoint"]
    },
```

```
    "Error Handling": {
        "Exceptions": ["exception"],
        "Crashes": ["crash", "fail"],
        "Bugs": ["bug", "error", "fault"]
    },
    "Cloud Services": {
        "AWS/Azure/GCP": ["aws", "azure", "gcp", "cloud"],
        "Serverless Architecture": ["serverless", "lambda"],
        "Storage Services": ["s3", "storage"]
    },
    "Other": {}
}

# Function to categorize issues based on title and description
with subthemes
def categorize_issue_with_subthemes(title, description):
    text = f"{title} {description}".lower()

    # Find the themes and subthemes that match
    for main_theme, subthemes_dict in subthemes.items():
        for sub_theme, keywords in subthemes_dict.items():
            for keyword in keywords:
                if re.search(r"\b" + re.escape(keyword) + r"\b",
text):
                    return main_theme, sub_theme

    # Default to "Other" if no match is found
    return "Other", "None"

# Apply the new categorization to each issue
issues_df["Main Theme"], issues_df["Sub Theme"] =
zip(*issues_df.apply(lambda row:
categorize_issue_with_subthemes(row["Issue Title"], row["Issue
Details"]), axis=1))

# Count of issues per main theme and sub theme
theme_subtheme_counts = issues_df.groupby(["Main Theme", "Sub
Theme"]).size().reset_index(name="Count")

theme_subtheme_counts.head()  # Display the first few rows for
review
```